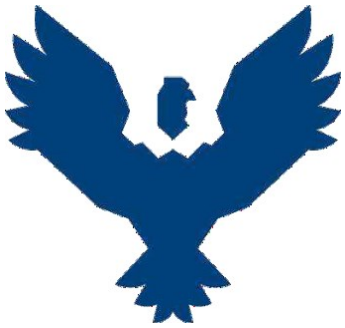
UNIVERSIDAD ANDINA DEL CUSCO

FACULTAD DE INGENIERÍA Y ARQUITECTURA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



APLICACIÓN DE LAS TÉCNICAS DE MACHINE LEARNING PARA LA DETECCIÓN EN IMÁGENES DE MONEDAS FALSAS Y VERDADERAS DE CINCO SOLES

Línea de Investigación:

Desarrollo de aplicaciones usando inteligencia artificial

Tesis presentado por: Bach. Aukgapuru Arcondo, Miguel Angel

Para optar al Título Profesional de:

Ingeniero de Sistemas

Asesor: Ing. Ivan Molero Delgado

CUSCO – PERÚ

2022

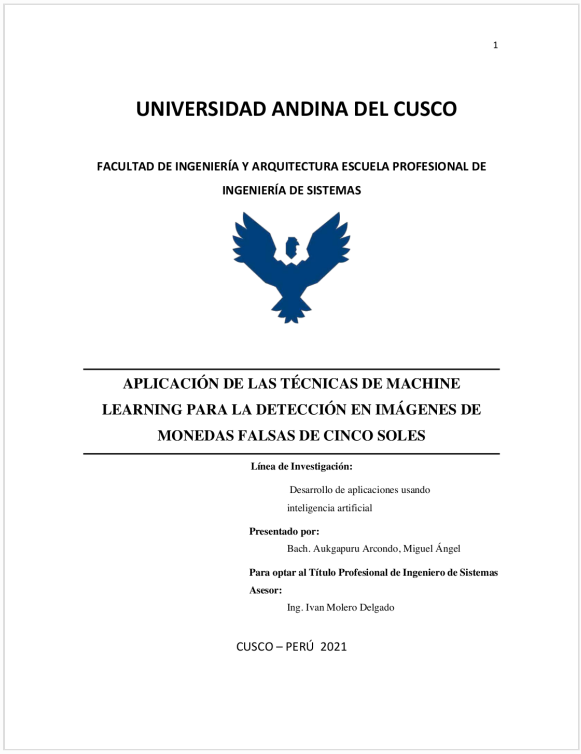
turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Miguel Angel Aukgapuru Arcondo
Assignment title:	Tesis
Submission title:	APLICACIÓN DE LAS TÉCNICAS DE MACHINE LEARNING PARA ...
File name:	ngel_Aukgapuru_efectividad_Monedas_version_v.4.1_CIENTIF...
File size:	13.94M
Page count:	182
Word count:	24,454
Character count:	133,145
Submission date:	18-Apr-2022 08:18AM (UTC-0500)
Submission ID:	1813485193



1

UNIVERSIDAD ANDINA DEL CUSCO

FACULTAD DE INGENIERÍA Y ARQUITECTURA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

APLICACIÓN DE LAS TÉCNICAS DE MACHINE LEARNING PARA LA DETECCIÓN EN IMÁGENES DE MONEDAS FALSAS DE CINCO SOLES

Línea de Investigación:

Desarrollo de aplicaciones usando inteligencia artificial

Presentado por:

Bach. Aukgapuru Arcondo, Miguel Ángel

Para optar al Título Profesional de Ingeniero de Sistemas

Asesor:

Ing. Ivan Molero Delgado

CUSCO – PERÚ 2021

APLICACIÓN DE LAS TÉCNICAS DE MACHINE LEARNING PARA LA DETECCIÓN EN IMÁGENES DE MONEDAS FALSAS DE CINCO SOLES

by Miguel Angel Aukgapuru Arcondo

LA DETECCIÓN EN IMÁGENES DE MONEDAS FALSAS DE CINCO SOLES

ORIGINALITY REPORT

17%	16%	3%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	hdl.handle.net Internet Source	2%
2	www.cs.us.es Internet Source	1%
3	sedici.unlp.edu.ar Internet Source	1%
4	sitiobigdata.com Internet Source	1%
5	Submitted to Universidad Andina del Cusco Student Paper	1%
6	files.criminalistica-peru1.webnode.es Internet Source	1%
7	repositorio.unsaac.edu.pe Internet Source	<1%
8	cvperu1.wordpress.com Internet Source	<1%

DEDICATORIA

El presente trabajo de investigación está dedicado a mis amados padres Miguel Aukgapuru Gomez y Julia Arcondo Atausupa quienes con su paciencia, amor, cariño, comprensión y esfuerzo me han permitido poder llegar a cumplir hoy, un sueño de muchos más. Gracias por inculcar en mí el ejemplo de esfuerzo, a nunca darse por vencido y por empujarme a seguir aprendiendo.

A mis hermanos Verushka, Leidi y Deivi por su amor de hermanos incondicional durante toda esta etapa de trabajo, por estar conmigo siempre, su buen humor y sus bromas, gracias. A toda mi familia porque con su buen humor y empuje hicieron de mí una mejor persona y que con su corazón me acompañan en todos mis sueños y metas.

Miguel Ángel Aukgapuru Arcondo

AGRADECIMIENTOS

En primer lugar, deseo expresar mi agradecimiento a mi asesor de esta tesis, el Ing. Iván Molero Delgado, por la dedicación y apoyo brindado a este trabajo, por la dirección y el rigor que ha facilitado a las mismas. Gracias por la confianza ofrecida desde que comenzamos esta investigación.

Un trabajo de investigación crece gracias a las ideas, consejos y esfuerzos previos que fueron realizados por otras mentes. En este caso mi más sincero agradecimiento a la Ing. Lida León Núñez y Mgt. Ing. Vivian Luz De La Vega Bellido, de la UNIVERSIDAD ANDINA DEL CUSCO, Gracias por su amabilidad, su tiempo y sus ideas.

Miguel Ángel Aukgapuru Arcondo

RESUMEN

Conseguir desarrollar un modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos

El tipo de investigación en este proyecto definió como básico tecnológico con un nivel experimental y un diseño pre-experimental al usar como métrica porcentual derivada de la matriz de confusión. La población se constituyó de un total de 496 imágenes de monedas de cinco soles entre verdaderas y falsas teniendo las cuales pasaran por la técnica de Data Augmentation para tener 800 imágenes por cada categoría expedidas en el modelo 2010-2015 y como instrumento de recolección de datos utilizaremos una cámara fotografía.

Como conclusión principal tenemos que el modelo “Final_Model” se desarrolló usando las técnicas de Machine Learning de manera teórica y práctica que conllevo a cumplir nuestro objetivo principal en esta investigación que es aplicar las técnicas de Machine Learning en la detección de monedas falsas y verdaderas de cinco soles Peruanos satisfactoriamente respondiendo al objetivo principal de la investigación.

ABSTRACT

This research focuses on the application of Machine Learning techniques for the development of a model that allows the detection by images of false and true coins of five Peruvian soles.

Since the invention of currency, counterfeiting was also born. It is necessary to make proposals for change in the aspects concerning the means of security in physical means of payment to protect the economic, social and political level.

For its part, the field of Machine Learning has grown more intensely since 2009, being able to apply to more branches of study, our country is late in the use of Machine Learning techniques since we find a lack of studies and development of tools that apply Machine Learning being a problem to reach a solution in detection of false and true coins in the REPUBLIC OF PERU.

Therefore, we propose the application of Machine Learning techniques to contribute to a future solution to this latent problem.

This research proposed the construction of a model using the Transfer Learning technique to join a pre-trained model and a personalized head model that was trained with images of true and false coins from the year 2010-2015. Analyzing the learning curve of the model and using the confusion matrix, the average error of the predictions was obtained with an approximate error of 20% in a population of 1600 photographic samples between false and true coins.

INTRODUCCIÓN

Esta investigación se enfoca en la aplicación de las técnicas de Machine Learning para el desarrollo de un modelo que permita la detección por imágenes de monedas falsas y verdaderas de cinco soles Peruanos.

Desde la invención de la moneda también nació a la falsificación de esta. Es necesario plantear propuestas de cambio en los aspectos concernientes a los medios de seguridad en medios de pago físicos para proteger el nivel económico, social y político.

Por su parte el campo del Machine Learning ha crecido con más intensidad desde el año 2009, pudiéndose aplicar a más ramas de estudio, nuestro país lleva tarde el uso de las técnicas Machine Learning ya que encontramos una falta estudios y desarrollo de herramientas que apliquen Machine Learning siendo un problema para llegar a una solución en detección de monedas falsa y verdaderas en la REPÚBLICA DEL PERÚ.

Por lo cual planteamos la aplicación de las técnicas de Machine Learning para contribuir con una futura solución de esta problemática latente.

Esta investigación plantea la construcción de un modelo utilizando la técnica de Transfer Learning para unir un modelo pre-entrenado y un modelo cabeza personalizado que fue entrenado con imágenes de monedas verdaderas y falsas del año 2010-2015. Analizando la curva de aprendizaje del modelo y usando la matriz de confusión se obtuvo el error promedio de las predicciones con un aproximado de 20% de error en una población de 1600 muestras fotográficas entre monedas falsas y verdaderas.

CONTENIDO

ÍNDICE FIGURAS

ÍNDICE TABLAS

ÍNDICE ANEXOS

1. CAPÍTULO I: ASPECTOS GENERALES

1.1 Descripción de la Situación Actual

La *Oficina Central de Lucha Contra la Falsificación de Numerario* (OCN) "Tiene como misión planificar e implementar las medidas conducentes a combatir la falsificación y alteración del numerario, a fin de contribuir a la generación de confianza en la legitimidad de la moneda que se utiliza como medio de pago" (Ley-27583, 2001).

Observando la coyuntura actual de distanciamiento provocado por la pandemia (COVID-19); "donde la detección de monedas falsas se hace imposible"(El Comercio, 2022); Ocasionando daños en la economía, como nos muestra la tabla 1.

Tabla 1

Falsificación de Monedad - Lima distritos

FALSIFICADORES DE MONEDAS - LIMA DISTRITOS			
AÑO	**CANTIDAD**		
2016	**S/**	**36,345.00**	
2017	**S/**	**123,292.00**	
2018	**S/**	**48,785.00**	
2019	**S/**	**575.00**	
	S/	**208,997.00**	**TOTAL**

Nota: El gráfico presenta cuantitativamente el dinero falsificado de cada año en el departamento de Lima. Extraído de *Oficina Nacional Contra la Falsificación* (OCN, 2019).

Según el decreto legislativo N.º 1412 Que, mediante Ley N.º 30823, "el *Congreso de la República* ha delegado en el *Poder Ejecutivo* la facultad de legislar para establecer el marco normativo para promover el despliegue transversal de las tecnologías digitales en las entidades del Estado; a fin de mejorar el alcance, condiciones, la prestación y el acceso de los ciudadanos a los servicios que presta el Estado"(El Peruano, 2018).

1.2 Formulación del Problema

Comprendiendo la situación actual que se detalla en el capítulo 1.1 podemos entender que problemas como los presentados en artículos como (El Comercio, 2020) desencadenan en efectos directos en la economía y la sociedad. Pérdida de confianza en la moneda por parte del mercado. Es así que nos centramos al campo del Machine Learning para plantear soluciones a la falsificación de monedas, pero la falta de estudios en técnicas de Machine Learning en el PERÚ dificultan que se pueda llegar a una propuesta concreta, funcional y aplicable a la falsificación de monedas.

Es así que la problemática se enfoca en que actualmente no se aplican las tecnologías de Machine Learning para detectar la falsificación en monedas, a pesar de que hay una demanda de herramientas tecnológicas como menciona (Gtd Perú, 2022) y (El Peruano, 2022). Así mismo el artículo de (Liu et al., 2017), nos da a entender que podemos teóricamente definir la detección de monedas falsas como un problema de disimilitud contraria a la de clasificación que es comúnmente abordada en las soluciones de visión por computadora dentro de Machine Learning. La falta de un estudio donde se precise la aplicación de técnicas de Machine Learning y Deep Learning para la detección por imágenes de monedas falsas y verdaderas en la república del PERÚ permitiría la mayor circulación de numerario falsificado, el cual genera inflación y desconfianza en la moneda entre otros problemas económicos y sociales (Economipedia, 2020) .Se precisa un estudio que pueda iniciar un camino para ayudar a solucionar esta problemática vigente.

1.2.1 Formulación Interrogativa del Problema General

¿Cómo aplicar las técnicas de Machine Learning para la detección por imágenes en monedas falsas de cinco soles Peruanos?

1.2.2 Formulación Interrogativa de los Problemas Específicos

1. ¿Cómo elaborar una base teórica que sustente la aplicación de las técnicas de Machine Learning a la detección de monedas falsas y verdaderas?
2. ¿Cómo desarrollar un prototipo de redes neuronales para la detección de monedas falsas y verdaderas de cinco soles Peruanos?
3. ¿Cómo interpretar los resultados obtenidos?

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos

1.3.2 Objetivos Específicos

1. Recopilar una base teórica que sustente la aplicación del Machine Learning a la detección de monedas falsas y verdaderas.
2. Desarrollar un prototipo de redes neuronales para la detección de monedas falsas y verdaderas de cinco soles Peruanos
3. Interpretar los resultados obtenidos

1.4 Hipótesis

Como métrica en nuestra investigación se usaran el resultado obtenido por (Patil et al., 2022), el cual define la misma problemática que esta investigación cambiando a billetes o dinero en papel, modelo que tiene a una precisión del 90% aproximadamente. Este resultado para medir la precisión del modelo a desarrollar con Machine Learning.

1.4.1 Hipótesis General

El modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos tendrá una precisión mayor o igual al 90%.

1.4.2 Sub Hipótesis

Hipótesis Nula

El modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos no tendrá una precisión mayor o igual al 90%. Es decir, la precisión será menor al 90%

Hipótesis Alternativa

El modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos tendrá una precisión mayor o igual al 70%.

1.5 Variables

1.5.1 Variables Independientes

X = Modelo con las técnicas de Machine Learning para la detección por imágenes.

Las técnicas usadas constan de parámetros de funcionamiento, así mismo la elección métodos específicos dentro de la arquitectura del modelo según la metodología de construcción de redes neuronales (Vasilev et al., 2019) son:

- Tipo red neuronal
- Conjunto de datos de entrenamiento (Dataset)
- Tipo de algoritmo de aprendizaje
- Algoritmo de entrenamiento

Así mismo según (Colchado, 2018)y(Jacinto, 2019), designan como indicadores de sus modelos en redes neuronales a los puntos relevantes dentro de estructura en el desarrollo.

1.5.2 Indicadores de Variables Independientes

Tabla 2

Variable independiente

Variables	*Indicadores*
X = Modelo con las técnicas de Machine Learning para la detección por imágenes.	V2.1 Metodología aplicada para desarrollar el modelo

*Nota***:** Tabla que presenta las variables delimitadas según los objetivos de la investigación y el entorno en donde se aplica esta, elaboración propia.

1.5.3 Variables Dependientes

Y = Detección de monedas falsas y verdaderas de 5 soles:

El número de aciertos en la identificación de monedas falsas y verdaderas, Se refiere al porcentaje alcanzando por una red neuronal artificial en la identificación de monedas, el criterio de evaluación de esta variable dependiente es el error cuadrático medio (ECM) la cual vemos más a fondo en el marco teórico capítulo 2 Donde:

Numeró de monedas detectadas correctamente = (1 - ECM) *100

Limitante:

El error cuadrático medio debe ser menor a 1:

ECM < 1

1.5.4 Indicadores de Variables Dependientes

Tabla 3

Variables Dependientes

VARIABLES	INDICADORES
Y = Detección de monedas falsas y verdaderas de 5 soles Peruanos	V 1.1 Error cuadrático medio (%)

*Nota***:** Tabla que presenta la variable delimitada según los objetivos de la investigación y el entorno en donde se aplica esta, elaboración propia.

1.6 Justificación

Como menciona el Artículo 83 de *la Constitución Política del Perú* del capítulo de la *Moneda y la Banca* del título III del Régimen Económico dice: "La ley determina el sistema monetario de la República. La emisión de billetes y monedas es facultad exclusiva del Estado. La ejerce por intermedio del Banco Central de Reserva del Perú"(República del Perú, 2010).

Con esto podemos determinar que cualquier esfuerzo, ya sea colaboración con la OCN (Oficina central de Lucha contra la falsificación de Numerario), destrucción de numerario falsificado, estudios que fomenten y contribuyan con la disminución de numerario falsificado, etc. Benefician al sistema económico y social defendiendo lo mencionado en el artículo 83 de la *Ley Orgánica del Banco Central de Reserva del Perú,* que contempla:

> *Artículo 2*.- "La finalidad del Banco es preservar la estabilidad monetaria."
>
> Sus funciones son regular la cantidad de dinero, administrar las reservas internacionales, emitir billetes y monedas e informar sobre las finanzas nacionales.
>
> *Artículo 24*.- Son atribuciones y deberes del directorio:
>
> Determinar y regular la emisión, características, canje y retiro de los billetes y monedas que el banco pone en circulación.
>
> *Artículo 42*.- "La emisión de billetes y monedas es facultad exclusiva del estado, quien la ejerce por intermedio del *Banco Central De Reservas*".
>
> *Artículo 48*.- El Banco sustituye, a la vista y a la par, los billetes y monedas inutilizados.
>
> *Artículo 49*.- El banco deber retener los billetes falsificados que le sean presentadas a los fines del canje, con el objeto de ponerlos, si fuere el caso, a la disposición de la autoridad policial o judicial, a los fines de la represión del delito. (Ley Orgánica Perú,

1992).

La OCN (Oficina central de Lucha contra la falsificación de Numerario) tiene como misión principal planificar e implementar las medidas conducentes a combatir la falsificación y alteración del numerario, es decir aplicar todos los métodos y herramientas posibles para combatir y detener la falsificación entre las funciones de esta tenemos establecidas (Ley Orgánica Perú, 1992).

Artículo 2º De La Ley Nº 27583

- Apoyar al *Ministerio Público* en el cumplimiento de sus funciones.
- Apoyar a la *Policía Nacional del Perú* (PNP) en el cumplimiento de sus funciones.
- Investigar preliminarmente el delito monetario.
- Analizar las falsificaciones y los casos de falsificación.
- Realizar peritajes destinados a determinar la autenticidad del numerario, nacional y extranjero.
- Centralizar la custodia de las falsificaciones.
- Brindar asistencia técnica a las autoridades competentes mediante la capacitación.

1.6.1 Relevancia Social

La falsificación de moneda afecta a quienes menos recursos económicos tienen. "Precisamente los hogares con un menor nivel de renta, son más dados al uso del dinero en efectivo como medio de pago, así mismo los negocios más vulnerables suelen ser los supermercados, restaurantes y tiendas de alimentación"(Economipedia, 2020).

Debido a la coyuntura actual, con nuevas restricciones de distanciamiento provocadas por la pandemia (COVID 19) no solo han afectado a la sociedad en su forma de convivir y socializar, sino que ha creado un refugio para la desinformación y el aprovechamiento de estas medidas de forma tal que nos perjudica directamente.

"El diario *Gestión* cubrió una noticia donde, según investigaciones policiales, se incautaron alrededor de 15 millones de soles en dinero falsificado" (Gestión, 2020).

Así mismo tiene un costo dentro del presupuesto anual del que hace uso el estado. Ya que según estadísticas de (Viles, 2015). Se calcula como 11 millones de dólares el costo social en países de Latinoamérica en torno a la falsificación de la moneda y su lucha.

"Finalmente, la falsificación de numerario provoca directamente aumento de la inflación en la moneda, así como desconfianza en la divisa por parte de la sociedad" (Viles, 2015). Para detener esto se debe hacer uso de nuevas tecnologías que permita servir de base en la elaboración de un aplicativo funcional que pueda colaborar y detener el avance de la circulación de monedas falsificadas es una necesidad dentro de nuestra coyuntura social actual, ya que beneficiara directamente a la sociedad bajando los índices de inflación, así como respaldando la confianza en el sistema monetario nacional.

1.6.2 Relevancia Aplicativa

Cómo se menciona “el correcto uso de las técnicas de Machine Learning y Deep Learning consisten en mitigar los posibles riesgos por medio de tener un modelo robusto en el diseño, lo cual baja la posibilidad de tener un modelo predictivo mal formulado”(Ipade, 2018).

Es por eso que un estudio que se enfoque en cuantificar el nivel de efectividad dentro de las herramientas de Machine Learning para la detección de monedas de cinco soles en la república del Perú, servirá como base de estudio y consulta a diferentes propuestas para el desarrollo de soluciones dentro del campo de la detección en imágenes así mismo para las soluciones habituales para las que es usado el Deep Learning.

“El campo del Machine Learning es un área que recién está dando sus primeros pasos, por ser constante en su estudio es necesario”(Dot Csv, 2020), más aún si está situado en un área específica con problemas específicos y reales.

1.6.3 Relevancia Teórica

No existen estudios referentes al numerario en América Latina y si bien existen investigaciones internacionales que se han dedicado a la problemática de la falsificación de numerario, como (Liu et al., 2017), estas son escasas y no son expuestas en su completa magnitud para su completa consulta y manipulación por lo cual esta investigación pretende servir de consulta para futuros estudios referentes al capo del Deep Learning y la problemática de falsificación de numerario.

El prototipo y resultados, será la culminación de todo lo aprendido durante este proceso para nutrir las bases teóricas científicas sobre el estudio y la aplicación de las técnicas de Machine Learning. Tiene carácter significativo mencionar que, en el desarrollo de esta

investigación, donde se determine la efectividad de las herramientas de Machine Learning para la detección de monedas de cinco soles en la república del Perú, permitirá dar un paso dentro del estudio del Machine Learning y los problemas de disimilitud como lo es la detección de monedas además demostrará tanto las habilidades como el límite de los conocimientos puestos en práctica del investigador.

1.7 Metodología

1.7.1 Tipo de Investigación

Para este proyecto el tipo de investigación tecnológica es el que más se ha adecuado, según Carrasco Díaz, porque está dirigida a conocer y aportar conocimientos al campo del Machine Learning dentro de la detección de monedas. Así mismo para dar un fundamento sólido para sus posibles aplicaciones prácticas. Todo esto usando un enfoque cuantitativo, porque es el resultado por default en cualquier desarrollo, Machine Learning será una métrica porcentual derivada de la matriz de confusión.

1.7.2 Nivel de Investigación

Se seleccionó el nivel experimental según (Carrasco Díaz, 2008), ya que se plantea medir el desempeño del modelo para el desarrollo de una futura solución aplicativa a la detección monedas falsas y verdaderas con Machine Learning. Es así que responderemos preguntas como: ¿qué mejoras se han logrado?, ¿cuál es la eficiencia del modelo?

1.7.3 Método de Investigación

El método de investigación a utilizarse, según Carrasco Diaz es el de un diseño múltiple formado por un diseño pre-experimental debido a que aplicado a nuestro problema conllevará a la aplicación de las técnicas de Machine Learning influyendo así en la variable de detección de monedas falsas o verdaderas. Así mismo el Accuracy esta en un solo grupo de control.

1.8 Matriz De Consistencia

Tabla 4

Matriz de Consistencia

Problemas	Objetivo	Hipótesis	Metodología
Principal ¿Cómo aplicar las técnicas de Machine Learning para la detección por imágenes en monedas falsas de cinco soles peruanos? **Específicos** • ¿Cómo elaborar una base teórica que sustente la aplicación del Machine Learning a la detección de monedas falsas y verdaderas? • ¿Cómo desarrollar un prototipo de redes neuronales para la detección de monedas falsas y verdaderas de cinco soles peruanos? • ¿Cómo interpretar los resultados obtenidos?	**General** Desarrollar un modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas de cinco soles Peruanos. **Específicos** • Recopilar una base teórica que sustente la aplicación del Machine Learning a la detección de monedas falsas y verdaderas • Desarrollar un prototipo de redes neuronales para la detección de monedas falsas y verdaderas de cinco soles Peruanos. • Interpretar los resultados obtenidos	**General** El modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos tendrá un error menor al 20%. **Nula** El modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos tendrá un error mayor al 20%. **Alternativa** El modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas y verdaderas de cinco soles Peruanos, tendrá un error menor al 50%.	**Tipo investigación** • Investigación de tipo básica tecnológica **NIVEL** • Nivel experimental **MÉTODO** • Diseño Pre-experimental

Nota: Instrumento metodológico que posibilita sistematizar, analizar y comprender los procedimientos y avances de una investigación.

CAPÍTULO II: MARCO TEÓRICO

2.1 Aspectos Teóricos Pertinentes

2.1.1 La Moneda

"Podemos definir moneda a aquellos instrumentos metálicos con forma de disco que sirven como forma de pago en una transacción cualquiera y a la cual se refieren como valor nominal en los precios"(Economipedia, 2020)

2.1.1.1 Origen y Evolución de la Moneda

Antes del uso de la moneda existía otra forma de intercambio valorizado por los pueblos y tribus, como las siguientes:

- Trueque
- Especies o moneda orgánica

Dentro del proceso histórico podemos señalar a la "ESTATERA", como la primera moneda de metal, acuñada en el año 700 a.c. en Lidia hecha de "Electrum" que es una aleación de plata y oro. "Posteriormente los griegos, chinos y otras culturas, que fueron artífices en la grabación de metales, fabricaron monedas en variados tipos de metal, conocidos por tener las siguientes características"(Sólo es Ciencia, 2019):

- Valores Considerables
- Escasamente inalterables
- Fraccionables (En décimos, centésimos o milésimos).

2.1.1.2 Moneda Metálica

Con estas características bien definidas se diversifica el uso de la moneda en diversas sociedades según sus necesidades y nace 2 formas de presentación:

Moneda Metálica sin acuñar

"Se crean piezas sin forma determinada y sin algún grabado que llegan a tener un solo valor intrínseco. Se hacen con materiales con valor simbólico importante (oro, plata y cobre)"(Sólo es Ciencia, 2019).

Moneda Metálica Acuñada

"El gobierno emite esta clase de moneda, pieza de metal con una forma estandarizada que tiene un símbolo único así mismo su valor empieza a ser nominal"(Sólo es Ciencia, 2019).

Actualmente las monedas que circulan en el mundo, están confeccionadas de metales más comunes como cobre, zinc, níquel, etc. "Las monedas de metal creadas con materiales innobles, llegan a conformar los valores más minúsculos del sistema económico monetario de un país, cave recalcar el valor intrínseco que tenían las monedas hechas de materiales nobles como el oro y plata"(Sólo es Ciencia, 2019).

2.1.1.2.1 Elaboración de la Moneda Metálica

La manufacturación de esta clase de moneda, se realiza en 10 etapas definidas:

Creación del molde

Es el proceso en el cual se construyen lo moldes que llegaran a ser tallados con los diseños o figuras los cuales son grabados en bajo relieve y de revés, "los cuales se van a trasmitir al metal recortado (tejuelo), mediante el uso de presión con máquinas especiales." (Sólo es Ciencia, 2019).

La Grabación

"Consiste en la utilización de objetos especiales para lograr tallar diversas figuras de cara revés a cada cara del molde de la moneda y así poder plasmar los detalles más sutiles" (Sólo es Ciencia, 2019).

El Retoque

"Consiste en corregir y eliminar los posibles errores encontrados en la creación de las monedas"(Sólo es Ciencia, 2019).

El Templado

"Proceso donde los moldes son puestos en hornos especiales los cuales alcanzan temperaturas altísimas este se enfría posteriormente, este proceso le dará la dureza necesaria al metal para ser usado seguidamente"(Sólo es Ciencia, 2019).

El Acabado

Se emplea esta técnica desarrollada por los expertos en la creación de nuevas monedas para quitar y reducir el número de impurezas o errores que tengan las matrices o moldes primarios para la producción a gran escala de las monedas, así mismo también se determinan los contornos de la representación, grabados, letras u otras figuras que se deseen agregar(Sólo es Ciencia, 2019).

Confección

"Es la construcción de discos propios de material metálico de acuerdo a ley, talla y peso de la moneda declarado por el ente nacional que emite y maneja el valor monetario de cada país (Banco Central de Reserva), dentro del proceso se conoce como tejuelo"(Sólo es Ciencia, 2019).

La Aleación

"Consiste en hacer una aleación de 2 o más metales, utilizando materiales de origen de reciclaje o previstos para este proceso; un ejemplo de lo materiales usados en la aleación sería 60 kg. de cobre, 25 kg de zinc y 15 kg de níquel. Los cuales a muy altas temperaturas crearan la materia prima de acuñación"(Sólo es Ciencia, 2019)

Laminado

Se aplastan largos rollos de metal hasta obtener el grosor estándar de las monedas.

Corte de los discos

Usando máquinas de corte especializadas se comenzará a extraer disco con las medidas deseadas las cuales son de características idénticas (casi moneda) (Sólo es Ciencia, 2019).

Acuñación de los discos

"En este proceso todos los detalles creados en las matrices se transfieren al disco para crear así la moneda como tal. La técnica utiliza máquinas de presión la cuales son llamadas acuñadoras que trabajan con pesos aproximados de 250 toneladas"(Sólo es Ciencia, 2019).

2.1.1.3 Falsificación de la Moneda

Se trata de imitar las técnicas de acuñación convencionales usadas por los bancos para generar dinero ilícito.

Diversos métodos son utilizados, pero los principales y eficientes son 3:

Moldeado

Se utiliza el yeso o productos de modelado, para generar moldes de las caras principales de la moneda (cara y sello) esto utilizando químicos que no permitan que la moneda se llegue a pegar al yeso y así mejorando la calidad de la copia.

Previo a juntar ambos moldes se abrirá un pequeño canal en la parte externa por donde se logre introducir la aleación preparada que generalmente no es del mismo material que la moneda real. Cuando esto se enfría se puede pasar a sacar los moldes, se pasará a afinar la moneda falsa, pero no se podrá llegar a un producto idéntico por el uso de métodos diferentes a los usados en las instituciones encargadas de la moneda(Sólo es Ciencia, 2019).

Cave recalcar el tipo de falsificación que tenían las monedas de materiales nobles como oro, plata, bronce, los cuales no pueden ser falsificados más si adulterados introduciendo cantidades de metales sin valor para hacer un engaño.

Galvanoplastia

Esta técnica electrolítica es usada para crear un revestimiento de metal en la moneda, que previamente fue generada por la técnica ya mencionada para así darle una apariencia parecida a la de las monedas reales al menos en la superficie.

Usando la galvanoplastia se obtendrá moldes o películas de las caras de la moneda. Para ello mete la moneda en un envase preparado para generar le proceso de electrólisis sin antes cubrir un lado de la moneda con grafito. Como podemos ver esta técnica comienza a volverse más elaborada, así se obtiene la reproducción deseada. Una vez teniendo dichas películas proceden a poner en medio de ellas un disco de una calidad básica de metal. Luego las sueldan de manera artesanal la cual es el punto donde podemos detectar si esta moneda fue hecha utilizando la galvanoplastia como técnica(Sólo es Ciencia, 2019).

Troquelado De Moneda

Este método obedece a las técnicas reales para obtener monedas auténticas, por consiguiente, se hacen las matrices o cuños, para esto existe otras maneras:

Creando las matrices con las cuales al someter al fuego una moneda y presionándola hacia la plancha obtenemos el grabado de las características requeridas(Sólo es Ciencia, 2019).

Mediante la electroerosión; procesa donde el uso de técnicas como el pantógrafo se copian y reproducen los relieves y detalles de la moneda para luego poder crear discos del tamaño de la moneda con los cuales al ejercer presión sobre otro disco metálico se grabará la cara de la moneda elegida(Sólo es Ciencia, 2019).

2.1.1.4 Falsificación de Dinero hoy en Día

Incluso además de los grandes intentos del gobierno de mantenerse al margen para detener la falsificación, desde la fundación temprana de las colonias americanas, siempre han perdurado uno cuantos falsificadores que han podido seguir cometiendo este acto. Si bien se desconoce exactamente cuánto dinero falso existe en circulación, hay estimaciones de hasta $ 3 mil millones de dólares en dinero falso en circulación en un momento dado. Hoy se informa que el Servicio Secreto se apropia de millones en dólares falsificados al año. Demostrando que el dinero falso aún tiene un impacto en Estados Unidos y otros países por esta razón los detectores de dinero falso son tan populares(Atao, 2015).

Falsificación de Moneda En El Perú

Nuestro numerario ha pasado por las distintas formas de facilitación que mostramos en los párrafos anteriores y que en la actualidad siguen vigente al momento

de producir monedas falsas.

Se conoce mediante la información de la OCN (oficina nacional contra la falsificación) que todos los métodos conocidos en el capítulo 2.3.1.1 son aplicados.

Debido a la falta de presupuesto y cultura social este tipo de acto delictivo continua en auge así mismo concuerda con mecanismos utilizados por organizaciones de crimen organizado(OCN, 2019)

2.1.1.5 Método General Para la Detección de Moneda Falsa

Si se quiere conocer de manera precisa si una moneda es auténtica o falsa, es primordial efectuar un riguroso examen el cual tendrá diferentes caminos no solo el de agudeza visual al detectar los detalles sino uno físico por los materiales empleados y químico por la reacción que tienen los elementos ciertos estímulos con el fin de esclare-ces si la moneda es verdadera(Sólo es Ciencia, 2019).

- El metal empleado.
- El peso de la moneda.
- El color, así como el índice de brillantez, considerando el pulido y la edad de la moneda.
- El diámetro de la moneda, que suele ser más grande si se usó la galvanoplastia.
- La sonoridad.
- El grosor, ya que se usó un estándar en su fabricación en masa.
- La ductilidad, resistencia, dureza y maleabilidad estándar de las monedas.
- La correcta grabación de los detalles en los bordes, el cual es un medio para garantizar la seguridad.

2.1.2 Inteligencia Artificial

La Inteligencia artificial es el campo científico de la informática que estudia y trabaja para desarrollar procesos y tecnologías que puedan ser consideradas por sus resultados como "inteligentes". Podemos decir que la inteligencia artificial contiene el concepto de "las computadoras pueden pensar como seres humanos"(Dot Csv, 2020). Habitualmente las soluciones de IA pueden analizar una gran cantidad de datos (Big Data). Identificar patrones y coincidencias y así poder llegar a tener una predicción de forma automática y rápida sin perder la precisión, Para nosotros, lo importante es que la IA permite que nuestras experiencias cotidianas sean más inteligentes. Para esto utilizamos los análisis de predicción o análisis predictivos y otras técnicas de IA en los sistemas que usamos a diario(Soledad Espezúa, 2021).

- Aprendizaje automático (Machine Learning)
- Big Data
- Análisis predictivo

2.1.2.1 Machine Learning

Machine Learning es comúnmente asociado a Big Data e inteligencia artificial sin embargo estos conceptos no soy muy similares, así mismo para comprender el Machine Learning se debe conocer lo que es Big Data y como se aplica al Machine Learning. Big data es un término utilizado para describir grandes conjuntos de datos los cuales son creados como resultado de grandes aumentos en los datos que se recopilan y almacenan. En esta época de la tecnología moderna, hay un recurso que tenemos en abundancia: un gran volumen de datos estructurados y no estructurados. Pasando los años 50 en el siglo 20, el aprendizaje automático ha evolucionado como un subcampo de estudio en la inteligencia artificial (IA) lo cual involucra algoritmos de

autoaprendizaje que analizan conocimientos de los datos para realizar predicciones(Vasilev et al., 2019).

Normalmente el humano tenía que analizar y crear reglas para construir modelos para grandes cantidades de datos en cambio el Machine Learning permite obtener este conocimiento en los datos y así poder mejorar gradualmente el rendimiento de estos modelos predictivos. El Machine Learning tiene un papel cada vez más importante en nuestra vida cotidiana. Gracias al aprendizaje automático, disfrutamos de sólidos filtros de correo no deseado, texto y voz. Conveniente software de reconocimiento, motores de búsqueda confiables en la web. Entre los algoritmos más utilizados en Inteligencia Artificial(IA) encontramos(Unpingco, 2019):

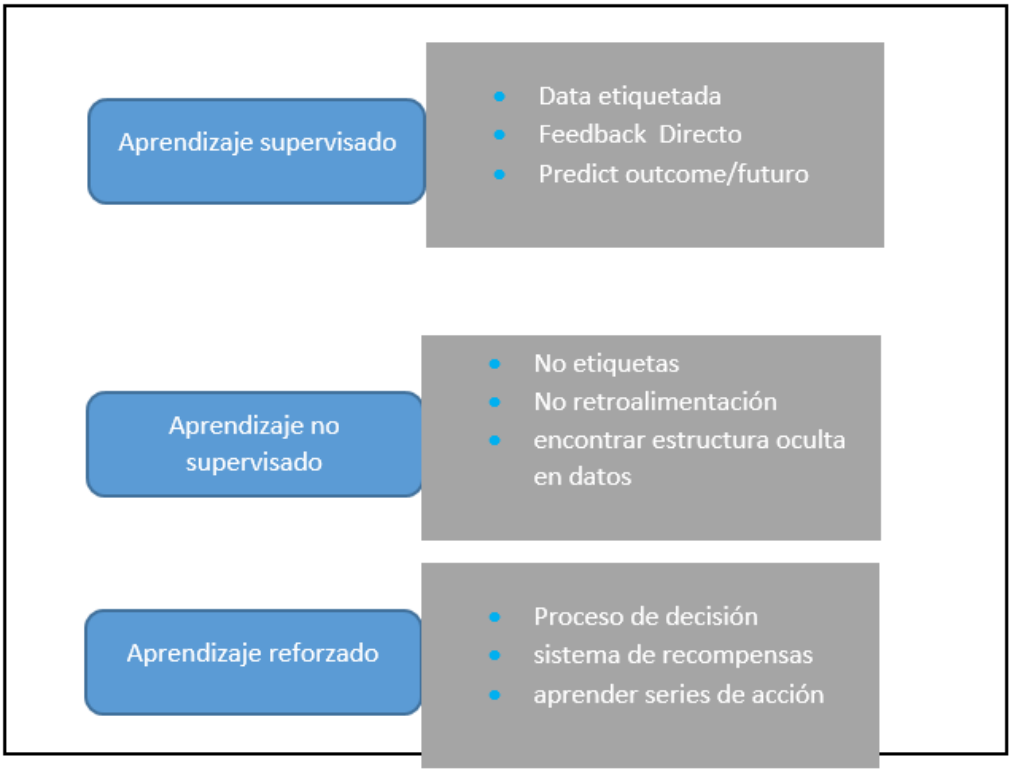
- Árboles de decisión
- Regresión lineal
- Regresión logística
- k Nearest Neighbor
- PCA / Principal Component Analysis
- SVM –Support Vector Machine
- K-MEANS
- Redes Neuronales Artificiales
- Aprendizaje profundo ó Deep Learning
- Clasificación de imágenes

2.1.2.1.1 Tipos Diferentes de Machine Learning

En esta sección, veremos los tres principales tipos de aprendizaje en el Machine Learning que son el resultado de la continua investigación.

Figura 1

Tipos de Aprendizaje Machine Learning



Nota: Mostramos según el autor Vasilev las distintas ramas del Machine Learning. Extraído de (Vasilev et al., 2019).

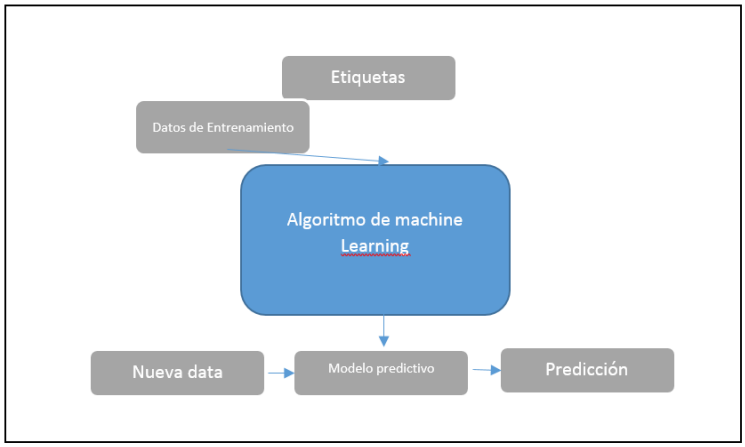
Aprendizaje Supervisado

La principal característica del aprendizaje supervisado, es generar un modelo a partir de datos ya etiquetados y con este poder hacer predicciones futuras

La palabra supervisado explica un conjunto de datos de entrenamiento de los cuales ya se conoce su etiqueta de salida, en esta imagen se explica el flujo del modelo supervisado el cual es un modelo que se ajusta con cada entrada y que finalmente predecirá datos sin etiquetas (Vasilev et al., 2019)

Figura 2

Aprendizaje Supervisado



Nota: Mostramos el proceso por el que pasa la data en uno proceso con Machine Learning.

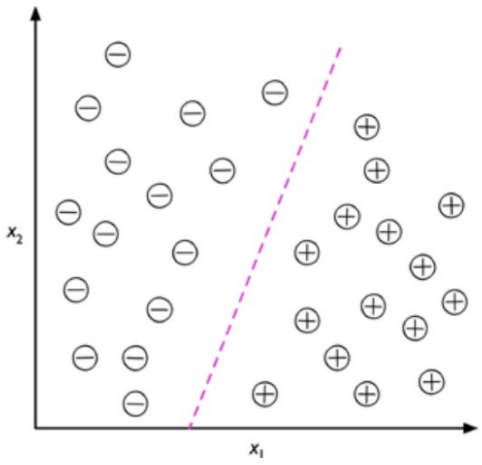
Para el ejemplo de poder seleccionar y desechar correos electrónicos por su tipo de información. Hacemos que el modelo entrene usando un algoritmo de aprendizaje supervisado con data de correos electrónicos con una identificación, estos están correctamente marcados como spam o no spam, Para pronosticar si un nuevo correo electrónico pertenece a una de las categorías, como en el ejemplo de correo no deseado, este método se denomina clasificación(Vasilev et al., 2019).

Clasificación

La clasificación es un área del aprendizaje supervisado donde se llega a pronosticar las etiquetas de categorías en las nuevas instancias, basándonos en observaciones ya hechas. Los labels de clase son variables discretas y desordenadas que se entienden como el conjunto de etiquetas para un conjunto. En el ejemplo de correo spam la detección representa un ejemplo común de un algoritmo de clasificación binaria, donde el algoritmo de aprendizaje automático aprende una serie de reglas para poder diferenciar entre dos resultados probables, correos electrónicos no deseados y comunes. En la figura 3 se muestra podemos encontrar un conjunto de 30 valores de los cuales 15 son positivos y 15 negativos estos están puestos dentro de un plano de 2 dimensiónes por lo que sus puntos de ubicación serán x1 y x2 por lo tanto se puede definir una regla para delimitar estas 2 clases y que los datos puedan ser clasificados en sus próximas entradas(Vasilev et al., 2019).

Figura 3

Ejemplo Correos Electrónicos x/y



Nota: Extraído de (Raschka & Mirjalili, 2019)

Pero el conjunto de etiquetas no tiene por qué ser de naturaleza binaria. El modelo predictivo aprendido por un algoritmo de aprendizaje supervisado tiene la capacidad de agregar una nueva etiqueta a una nueva instancia presentada que cuente con esta.

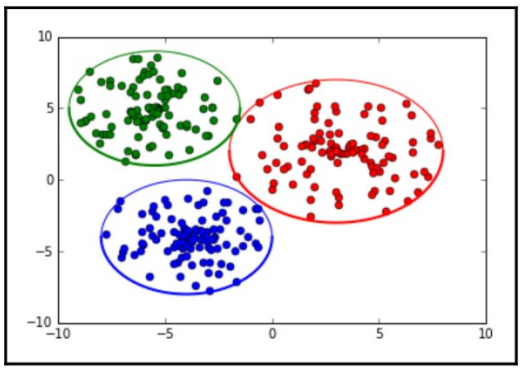
Como ejemplo, la tarea de reconocer el carácter escrito a mano, podemos recopilar un Dataset de entrenamiento que contenga dentro el alfabeto. Las letras ("A", "B", "C", etc.) representar las diferentes categorías desordenadas o etiquetas de clase que queremos predecir. Ahora, si se llega a proporcionar un nuevo carácter escrito a mano por medio de una entrada, El modelo podrá predecir la letra correcta del alfabeto con una precisión aceptable. Sin embargo, el proceso que usamos para el Machine Learning no podrá reconocer correctamente ninguno de los dígitos entre 0 y 9(Vasilev et al., 2019).

Aprendizaje no Supervisado

La segunda clase de aprendizaje es la no supervisada en esta técnica no etiquetamos los datos, sino que dejamos que la red encuentre un camino, la forma más común de cómo podemos entender esta técnica es la agrupación en subconjuntos. Para ilustrar este algoritmo usaremos el ejemplo de los correos electrónicos spam en este caso no etiquetaremos el correo, sino que pasaremos a ingresarlo al algoritmo el cual agrupara cada elemento en sub conjuntos según sus cualidades y similitudes, existen distintos tipos de algoritmos para distintos tipos de datos. En el siguiente gráfico, mostramos cómo se puede clasificar un conjunto de puntos para formar tres subconjuntos(Vasilev et al., 2019).

Figura 4

Aprendizaje no Supervisado



Nota: Extraído de(Raschka & Mirjalili, 2019)

El aprendizaje profundo también utiliza técnicas no supervisadas, aunque diferentes a la agrupación en clústeres. En el procesamiento del lenguaje natural (NLP), utilizamos algoritmos no supervisados para las representaciones vectoriales de palabras. La forma más popular de hacer esto se llama word2vec. Para cada palabra, usamos sus palabras circundantes (o su contexto) en el texto y las alimentamos a una red neuronal simple. La red produce un vector numérico, que contiene mucha información para la palabra (derivada del contexto). Luego usamos estos vectores en lugar de las palabras para varias tareas de PNL, como el análisis de sentimientos o la traducción automática.(Vasilev et al., 2019).

K-Means

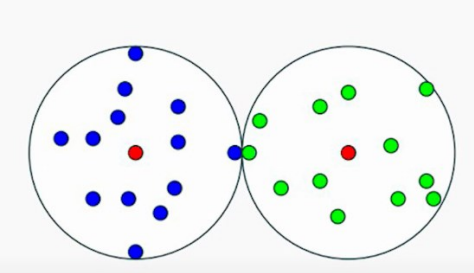
Es un algoritmo de clustering que agrupa los elementos de un conjunto de datos en k distintos subconjuntos.

Elija k puntos aleatorios, llamados centroides del espacio de características, que representan el centro de cada uno de los k grupos(Vasilev et al., 2019).

- Asigne cada muestra del conjunto de datos (es decir, cada punto en el espacio de características) al clúster con el centroide más cercano.
- Para cada grupo, volvimos a calcular nuevos centroides tomando los valores medios de todos los puntos en el grupo.
- Con los nuevos centroides, repetimos los pasos 2 y 3 hasta que se cumplan los criterios de detención.

Figura 5

Cuadro Algoritmo K-Means

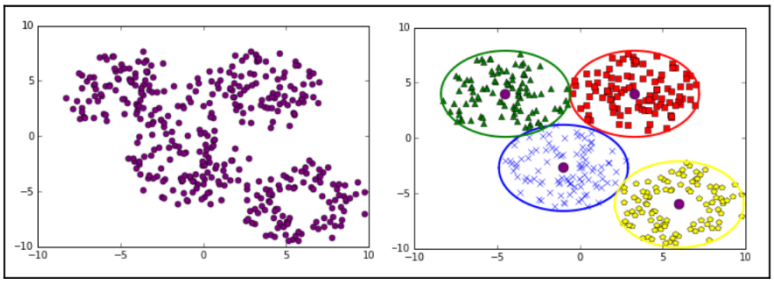


Nota: Extraído de (Fundación General de la Universidad de La Laguna, 2018)

El método anterior es sensible a la elección inicial de centroides aleatorios y puede ser una buena idea repetirlo con diferentes elecciones iniciales. También es posible que algunos centroides no estan cerca de ninguno de los puntos del conjunto de datos, lo que reduce el número de clústeres desde k(Vasilev et al., 2019).

Figura 6

K-MEANS Correos Electrónicos



Nota: Extraído de (Vasilev et al., 2019)

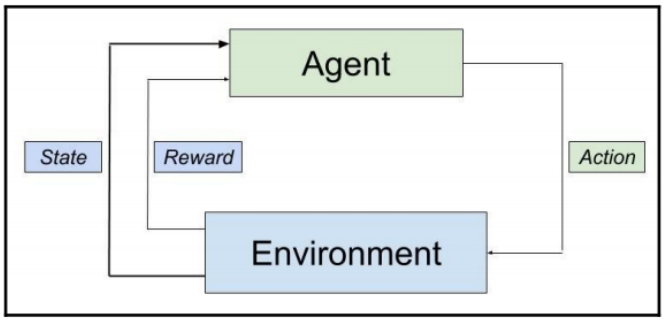
En la imagen, podemos ver la distribución de todos los correos, lo que el algoritmo crea sub grupos (clúster) dando de medida inicial del grupo lo que llamamos el centroide.

Aprendizaje reforzado

La tercera clase de técnicas de aprendizaje automático se llama aprendizaje por refuerzo (RL). Podemos entender este algoritmo con una de las aplicaciones más populares del aprendizaje por refuerzo: Enseñar a las máquinas a jugar, la máquina (o agente) interactúa con el medio ambiente. El objetivo principal del agente es ganar el juego, para esto el agente realiza acciones dentro del medio ambiente así cambiándolo. El medio ambiente manda señales al agente con el fin de acercarlo a terminar el juego que sería la máxima recompensa dentro de este ambiente. La eficiencia del algoritmo seria obtener la mayor cantidad de recompensas(Vasilev et al., 2019).

Figura 7

Línea Aprendizaje Reforzado



Nota: Extraído de (Vasilev et al., 2019)

"En el aprendizaje por refuerzo, el agente realiza una acción que cambia el estado del ambiente. El agente usa el nuevo estado y la recompensa para determinar su próxima acción"(Vasilev et al., 2019).

El ejemplo que proponemos es el de un juego de ajedrez donde el ambiente es el tablero, se recompensara al agente por matar a una pieza enemiga y se proporcionara la máxima recompensa al hacer un jaque mate y en otra instancia si el oponente mata una pieza del agente se le dará una recompensa negativa de igual modo al tener un jaque mate y ninguna acción cuando a la pieza movida no sea comida o jaque mate. Si este fuera un ejemplo de algoritmo supervisado tendríamos que poner etiquetas a cada movimiento, pero siendo aprendizaje no supervisado no es necesario acá, dejaremos que el algoritmo use sus experiencias previas para lograr su objetivo final y conseguir recompensas(Vasilev et al., 2019).

2.1.2.2 Componentes de una Solución de Machine Learning

Un algoritmo de Machine Learning es una parte de la solución y del proceso completo, por lo cual veremos las principales características que hacen aún sistema de Machine Learning.

Aprendizaje: Este algoritmo se utiliza con su filosofía de aprendizaje. La elección de este El algoritmo está determinado por el problema que estamos tratando de resolver, ya que Los problemas diferentes pueden adaptarse mejor a determinados algoritmos de aprendizaje automático(Vasilev et al., 2019).

Datos de entrenamiento: este es el conjunto de datos sin procesar que nos interesa. Puede ser etiquetado o sin etiquetar, es importante tener suficientes datos de muestra para que el algoritmo de entrenamiento comprenda la estructura del problema.

Representación: así es como expresamos los datos en términos de las características elegidas, para que podamos dárselo al algoritmo, por ejemplo, para clasificar imágenes escritas a mano de dígitos, representaremos la imagen como una matriz de valores, donde cada celda contienen el valor de color de un píxel. Una buena elección de representación de los datos es importante para lograr mejores resultados(Vasilev et al., 2019).

Objetivo: Esto representa lo que se está aprendiendo, así como el resultado final. Los destinos puede ser una clasificación de datos sin etiquetar, una representación de datos de entrada según patrones o características ocultas, un simulador de predicciones futuras, o una respuesta a un estímulo o estrategia exterior (en el caso de refuerzo aprendizaje).

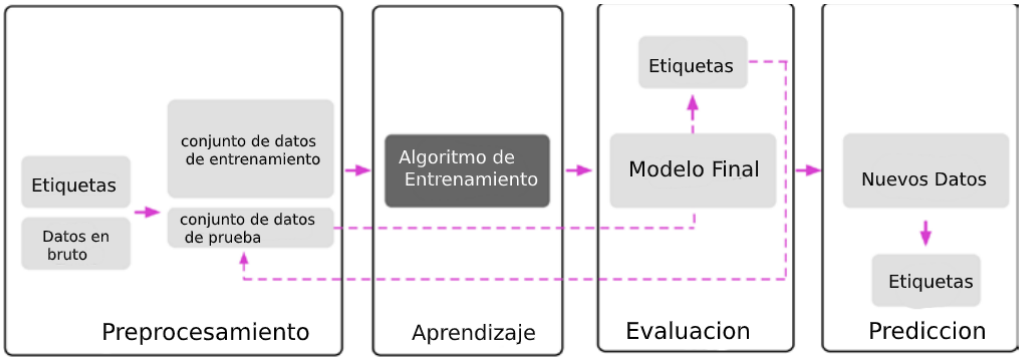
Nunca se puede enfatizar lo suficiente: cualquier algoritmo de aprendizaje automático solo puede lograr una aproximación del objetivo y no una descripción numérica perfecta. Los algoritmos de Machine Learning no son soluciones matemáticas son aproximaciones. Ciertos algoritmos de aprendizaje automático, como redes neuronales, pueden aproximarse a cualquier función en cualquier grado en teoría. Este teorema es llamado Teorema de aproximación universal, pero no implica que podamos obtener una solución precisa a nuestro problema. Además, las soluciones al problema pueden ser mejores logrado mediante una mejor comprensión de los datos de entrenamiento(Vasilev et al., 2019).

2.1.2.3 Una Hoja de Ruta para Desarrollar un Sistema de Machine Learning

En esta sección, discutiremos las posibles rutas de desarrollo metodológico para un sistema de aprendizaje automático que acompañan al algoritmo de aprendizaje.

Figura 8

Componentes de una Solución de Machine Learning



Nota: Extraído de (Raschka & Mirjalili, 2019)

Pre-procesamiento

"Los datos brutos rara vez se encuentran en la forma necesaria para el óptimo rendimiento de un algoritmo de Machine Learning. Por tanto, el preprocesamiento de los datos es uno de los pasos más importantes en una aplicación de Machine Learning"(Vasilev et al., 2019).

"Muchos algoritmos de aprendizaje automático también requieren que se habiliten funciones seleccionadas para un rendimiento óptimo, que a menudo se logra transformando características en el rango [0, 1] o una distribución normal estándar con media cero y varianza unitaria"(Vasilev et al., 2019).

Una buena herramienta resulta ser la técnica de reducción de dimensionalidad para poder comprimir características en un sub-espacio esto nos dará una ventaja al momento de que el algoritmo las procese ya que podrá ser más veloz, así mismo hay conjuntos de datos que contiene información irrelevante (ruido) es decir cambiando a la dimensión podrá tener unos datos con un ruido bajo.

Para asegurarnos de que nuestro algoritmo de Machine Learning funciona bien no solo con los datos de entrenamiento, sino también con los de prueba tendremos que hacer una división de los datos que tengamos, para cada una de estas funciones necesarias, usamos la formación conjunta de datos para entrenar y optimizar nuestro modelo de aprendizaje automático, mientras mantenemos en prueba el conjunto de datos hasta el final para evaluar el modelo final(Vasilev et al., 2019).

Entrenando y seleccionando un modelo predictivo

Un punto importante que puede ser resumido de los famosos teoremas del almuerzo gratis de David Wolpert es que no podemos aprender "gratis", por lo tanto, es primordial

comparar varias opciones de algoritmos similares para poder determinar cuál es el más efectivo a aplicar en la problemática. Primeramente, antes de empezar a comparar los algoritmos es vital elegir una métrica para medir el desempeño de este. Una problemática frecuente a tomar en cuenta y de la que ya se ha hablado es si los datos de training serán igualmente válidos que los datos de test para eso se debe dividir aún más los datos en subconjuntos de test, prueba y al mismo tiempo saber que el algoritmo no es necesariamente adecuado para estos datos. Finalmente, no se puede esperar que los parámetros predeterminados de los diferentes algoritmos de aprendizaje proporcionados por las bibliotecas de software son óptimos para nuestra problemática específica. Por lo tanto, haremos uso frecuente de técnicas de optimización de hiperparámetros que nos ayudan para ajustar el rendimiento de nuestro modelo(Vasilev et al., 2019).

Podemos pensar en esos hiperparámetros como parámetros que no se aprenden de los datos, pero representan las perillas de un modelo que podemos girar para mejorar su rendimiento. Esto será mucho más claro en capítulos posteriores cuando veamos ejemplos reales.(Vasilev et al, 2019).

Evaluar modelos y predecir instancias de datos

Después de haber seleccionado un modelo que se adecua al Dataset de test, podemos usar el conjunto de datos de prueba para realizar una aproximación de que tan eficiente podrá ser nuestro modelo. Y si estamos satisfechos con su desempeño, podemos utilizar dicho modelo para predecir datos recientemente agregados y datos futuros. Es importante señalar que los parámetros para los procedimientos mencionados anteriormente, como la escala de características y la reducción de la dimensionalidad, se obtienen únicamente del conjunto de datos de entrenamiento, y los mismos

parámetros son luego son vueltos a aplicar para transformar el conjunto de datos de prueba, así como cualquier nueva instancia de datos(Vasilev et al., 2019):

2.1.2.4 Redes Neuronales

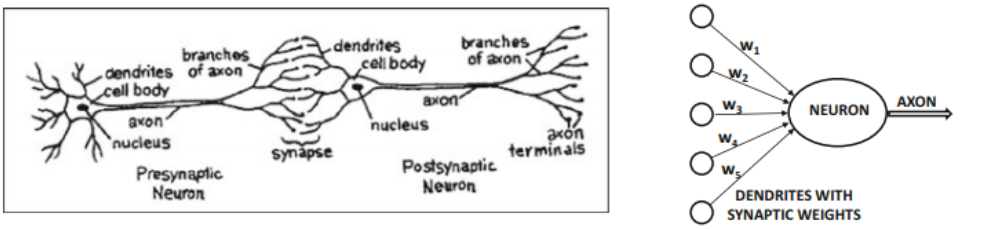
Las redes neuronales simulan el llamado comportamiento de las neuronas biológicas. Las neuronas están conectadas entre sí con el uso de mecanismos químicos y también por los axones y dendritas. Estas conexiones se ilustran en la figura número 9. La particularidad de estos canales de conexión es que cambian debido a un estímulo externo y es lo que nosotros conocemos como aprendizaje y adaptación como dice .(Vasilev et al, 2019).

Las neuronas computacionales están conectadas entre sí a través de lo que podemos conocer cómo pesos, que trabajan como la conexión sináptica en los organismos celulares biológicos. Las entradas a una neurona tienen un peso o valor que afecta al valor de la función final, esta arquitectura se ilustra en la figura 9. Una red neuronal artificial llega a tener entradas por medio de la propagación de los muchos valores calculados utilizando los llamados pesos equivalentes a parámetros en el medio. La función de aprendizaje se da en el intercambio de los pesos que conectan nuestras neuronas. Así como estímulos del exterior lo hacen para los seres biológicos, los estímulos del exterior para los algoritmos de redes neuronales proporcionan la data de entrenamiento que contiene ejemplos que se pasaran a aprender. Un ejemplo, los datos pueden contener representaciones de píxeles en imágenes con sus etiquetas como salida. Estos datos de training que ayudan a que el modelo aprenda mediante la utilización de representaciones para hacer predicciones en los labels de salida. Los datos proporcionan información de la exactitud en los pesos en esta red neuronal dependiendo de qué tan

bien corresponde la salida prevista para una entrada particular la etiqueta de salida anotada en los datos de training. Uno puede ver las fallas cometidas por la red neuronal en el proceso de una función como una especie de feedback, lo que lleva a un ajuste en las fuerzas. Del mismo modo, los pesos entre las neuronas se ajustan en una red neuronal como respuesta en fallas a la hora de predecir los datos el objetivo del cambio de pesos es afinar la red neuronal para que haga mejores predicciones. Las ponderaciones serán cambiadas de una forma correcta y analizada matemáticamente, así bajar el error en el cálculo. Si la red neuronal se entrena con muchas imágenes de diferentes plátanos, eventualmente será capaz de identificar un plátano que no ha visto, esta habilidad de calcular con precisión de entradas sobre un conjunto finito de entrada-salida. Esta es la capacidad para generalizar su aprendizaje a partir de datos de entrenamiento vistos(Vasilev et al, 2019).

Figura 9

Neurona Biológica



(a) Biological neural network (b) Artificial neural network

Nota: Extraído de(Vasilev et al, 2019)

Podemos describir una red neuronal como un modelo matemático para el procesamiento de información. Así también podemos ver algunas de sus características:

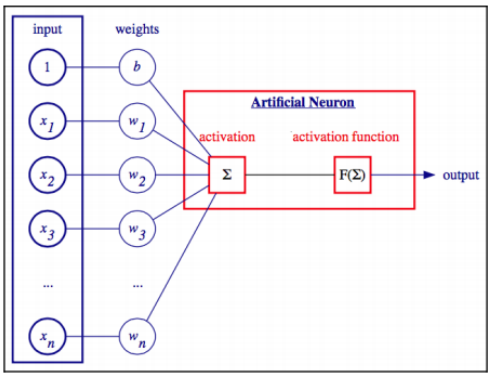
- El procesamiento de la información ocurre en su forma más simple, sobre elementos simples llamadas neuronas.
- Las neuronas están conectadas e intercambian señales entre ellas a través de enlaces de conexión.
- Los enlaces de conexión entre neuronas pueden ser más fuertes o más débiles, y esto determina cómo se procesa la información.
- Cada neurona tiene un estado interno que está determinado por todas las conexiones de otras neuronas.
- Cada neurona tiene una función de activación diferente que se calcula sobre su estado, y determina su señal de salida.

2.1.2.4.1 Una Introducción a las Neuronas Logica

"Una neurona es una función matemática que toma uno o más valores de entrada y genera un único valor numérico" (Vasilev et a, 2019):

Figura 10

Input , Output de Red Neuronal



Nota: Extraído de (Vasilev et al, 2019)

La neurona se define de la siguiente manera:

$$y = f(\sum_i x_i w_i + b)$$

Primero, calculamos la suma $\sum x_i w_i$ ponderada de las entradas xi y los pesos wi (también conocido como valor de activación). Aquí, xi es numérico valores que representan los datos de entrada o las salidas de otras neuronas (es decir, si la neurona es parte de una red neuronal):(Vasilev et al., 2019).

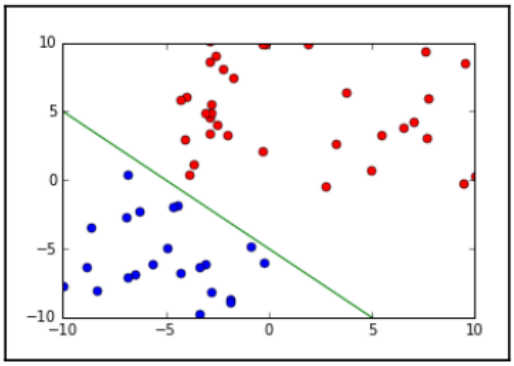
- Los pesos (Wi) son valores numéricos que representan la fuerza de las entradas o, alternativamente, la fuerza de las conexiones entre las neuronas.
- El peso b es un valor especial llamado sesgo cuya entrada es siempre 1

Luego, usamos el resultado de la suma ponderada como el ingreso para la activación función f, que también se conoce como función de activación. Hay muchos tipos de funciones de activación, pero todas deben satisfacer el requisito de ser no lineales, que explicaremos más adelante en el capítulo.(Vasilev et al., 2019).

Una introducción, el valor de activación definido previamente se puede interpretar como el producto escalar entre el vector wy el vector x: El vector x $\cdot x: y = f(\vec{x} \cdot \vec{w} + b)$. será perpendicular al vector de peso w, sí. $\vec{x} \cdot \vec{w} = 0$. Por lo tanto, todos los vectores x tales que $\vec{x} \cdot \vec{w} = 0$ definen un hiperplano en la característica espaciador norte , donde n es la dimensión de x. Para entenderlo mejor, consideremos un caso especial donde la función de activación es f (x) = x y solo tenemos un único valor de entrada, x. La salida de la neurona entonces se convierte en y = wx + b, que es la ecuación lineal. Esto muestra que, en el espacio de entrada unidimensional, la neurona define una línea. Si visualizamos lo mismo por dos o más entradas, veremos que la neurona define un plano, o un hiperplano, para un número arbitrario de dimensiones de entrada.(Vasilev et al., 2019).

Figura 11

Plano Red Básico



Nota: Extraído de (Vasilev et al, 2019)

2.1.2.4.2 Una Introducción a las Capas

Una red neuronal puede albergar un número infinito de neuronas, se organizan en capas conectadas entre sí. La capa de ingreso es la representación del conjunto de datos y las condiciones iniciales. Por ejemplo, si la entrada es una imagen, lo normal es que la salida sea un pixel de la entrada por cada capa. Por esta misma razón, generalmente no contamos la capa de entrada como parte de las otras capas. Cuando queremos decir red de 1 capa, lo que en verdad representa es red simple con solo una capa, la salida, además de la capa de entrada.

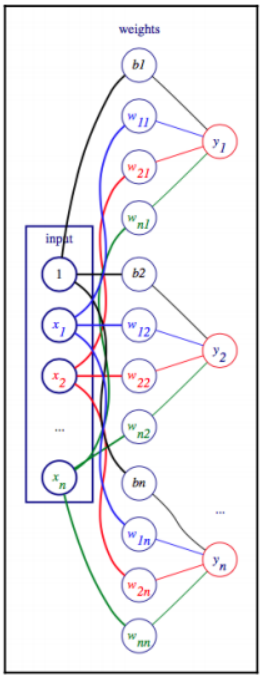
La capa de salida puede tener más de una neurona. Esto es especialmente útil en la clasificación, donde cada neurona de salida representa una clase(Vasilev et al., 2019).

Por ejemplo, tenemos 10 neuronas de salida, donde cada neurona corresponde a un dígito del 0 al 9. De esta manera, podemos usar la red de 1 capa para clasificar el dígito en cada imagen. Determinaremos el dígito tomando la neurona de salida con mayor valor de la función de activación(Vasilev et al., 2019).

Presentamos la 1 capa. Para este caso, nosotros mostraremos explícitamente los pesos w para cada conexión entre las neuronas, pero por lo general, el del borde que conecta las neuronas representan implícitamente los pesos. Peso wij conecta el i-ésimo neurona de entrada con la j-ésima neurona de salida. La primera entrada, 1, es la unidad de polarización y la peso, b1 , es el peso del sesgo(Vasilev et al., 2019):

Figura 12

Capas de Una Red Neuronal



Nota: Extraído de (Vasilev et al., 2019)

En el diagrama anterior, vemos la red neuronal de 1 capa en la que la neurona de la izquierda representa la entrada con sesgo b, la columna del medio representa los pesos para cada conexión, y las neuronas de la derecha representan la salida dados los pesos w.

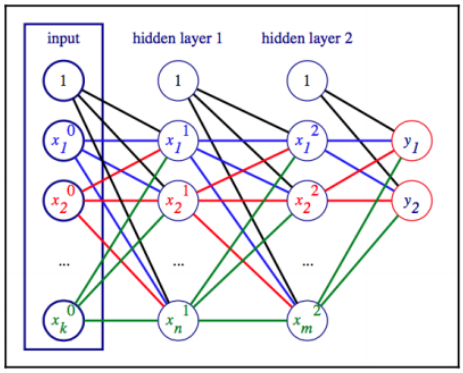
Las neuronas de una capa se pueden conectar a las neuronas de otras capas, pero no a otras neuronas de la misma capa. Las neuronas que entran están conectadas solo a la salida de neuronas. Pero Un argumento es que la neurona puede transmitir información limitada ósea solo un valor. Pero cuando combinamos las neuronas en capas, sus salidas componen un vector. En ese entender, en lugar de activación única, podemos ahora considerar el vector en su totalidad. De esta forma, se puede transmitir mucha más información, no solo porque el vector tiene múltiples valores, también pro que llevan valores adicionales(Vasilev et al., 2019).

2.1.2.4.3 Redes Neuronales Multicapa

Como hemos mencionado muchas veces, las redes neuronales de 1 capa solo pueden clasificar linealmente separables clases. Lo que propone la idea de multicapa es el uso de capas ocultas que es totalmente posible dentro de cualquier algoritmo de redes neuronales, como podemos ver en la ilustración 13(Vasilev et al., 2019).

Figura 13

Redes Neuronales Multicapa



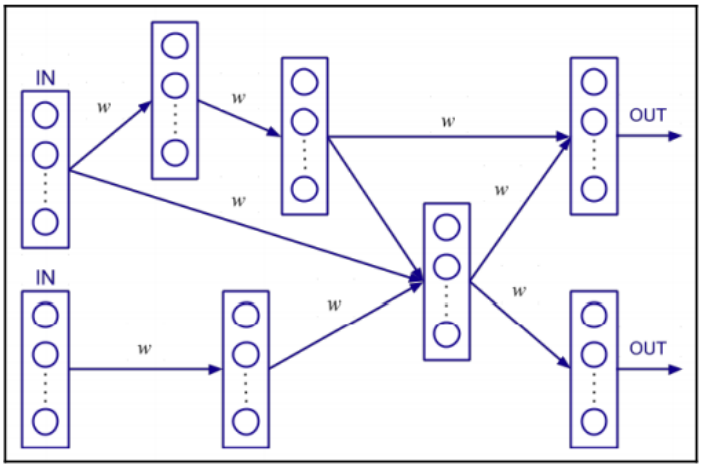
Nota: Extraído de (Vasilev et al., 2019)

Pero no estamos solamente limitados a redes de capas secuenciales, como se muestra en la ilustración. Las neuronas y sus conexiones forman gráficos que son cíclicos. En tal gráfico, la información no puede pasar dos veces por neurona y fluye en una sola dirección de la entrada a la salida. También disponemos por organizarlos en capas; por lo consiguiente, las capas también se llegan a organizar en un diagrama cíclico dirigido. La red en el anterior diagrama es solo un caso de un gráfico con capas de conexión secuencial(Vasilev et al., 2019).

"Los siguientes diagramas también muestra una red neuronal válida con dos capas de entrada, dos de salida capas y capas ocultas interconectadas aleatoriamente. Hemos representado los pesos múltiples, w, conectando las capas como una sola línea"(Vasilev et al., 2019):

Figura 14

Interconexión Entre Capas



Nota: Extraído de (Vasilev et al., 2019)

En esta sección, presentamos el tipo más básico de red neuronal, es decir la neurona, y lo expandimos gradualmente a un gráfico de neuronas, organizado en capas. Pero

podemos pensar de ello de otra manera. Así, llegamos a saber que la neurona tiene una precisión matemática de definición. Por tanto, la red neuronal, como composición de neuronas, también es una función matemática donde los datos de entrada representan los argumentos de la función y los pesos de la red, w, son sus parámetros(Vasilev et al., 2019).

2.1.2.4.4 Diferentes tipos de función de activación

Ahora sabemos que las redes multicapa pueden clasificar clases linealmente inseparables. Pero para hacer esto, necesitamos satisfacer una condición más. Si las neuronas no tienen funciones de activación, su salida sería la suma ponderada de las entradas $\sum_i w_i x_i$, que es una función lineal. Entonces toda la red neuronal, es decir una composición de neuronas, se convierte en una composición de funciones lineales, que también es una función lineal. Esto significa que incluso si agregamos oculto capas, la red seguirá siendo equivalente a un modelo de regresión lineal simple, con todas sus limitaciones. Para convertir la red en una función no lineal, usaremos activación no lineal funciones para las neuronas. Por lo general, todas las neuronas de la misma capa tienen la misma función de activación, pero diferentes capas pueden tener diferentes funciones de activación.(Vasilev et al., 2019).

La función de activación de identidad, o función de umbral, fue ampliamente utilizada, al inicio de las redes neuronales con implementaciones como el perceptron o el Adaline (adaptativa neurona lineal), pero posteriormente perdió tracción a favor del sigmoide logístico, el hiperbólico tangente, o el ReLU y sus variaciones. Las últimas tres funciones de activación difieren en las siguientes formas:

- Su rango es diferente.

- Sus derivados se comportan de manera diferente durante el entrenamiento.

2.1.2.4.5 Entrenamiento de Redes Neuronales

Hemos visto cómo las redes neuronales pueden asignar entradas a determinadas salidas, según sobre pesos fijos. Una vez definida la arquitectura de la red neuronal y incluye la red de alimentación, el número de capas ocultas, el número de neuronas por capa, y la función de activación, necesitaremos establecer los pesos y a su vez definir los estados internos de cada neurona en la red(Vasilev et al., 2019).

Primero usaremos el algoritmo descenso de gradiente para aplicarlo a una red de una capa, y luego extiéndalo a una red profunda de retroalimentación con la ayuda de retro propagación(Vasilev et al., 2019).

El concepto para poder entender es el siguiente: Cada red neuronal es una aproximación de una función, por lo que cada red neuronal no será igual a la función que se requiere, y diferirá del valor real con un error. Durante la formación el objetivo es minimizar este error, Dado que el error es una función que trabaja con los pesos de la red. La función de error es una función de muchos pesos y, por tanto, una función que hace uso de muchas variables(Vasilev et al., 2019).

"Matemáticamente, el conjunto de puntos donde esta función es cero representa una hiper superficie, y para encontrar un mínimo en esta superficie, queremos elegir un punto y luego seguir una curva en la dirección del mínimo"(Vasilev et al., 2019).

Regresión lineal

En resumen, con respecto a la utilización de la notación vectorial, la salida de un algoritmo de regresión es un valor único y es igual al producto escalar de la entrada. Valores xy los pesos w. Como sabemos ahora, la regresión lineal es un caso especial de

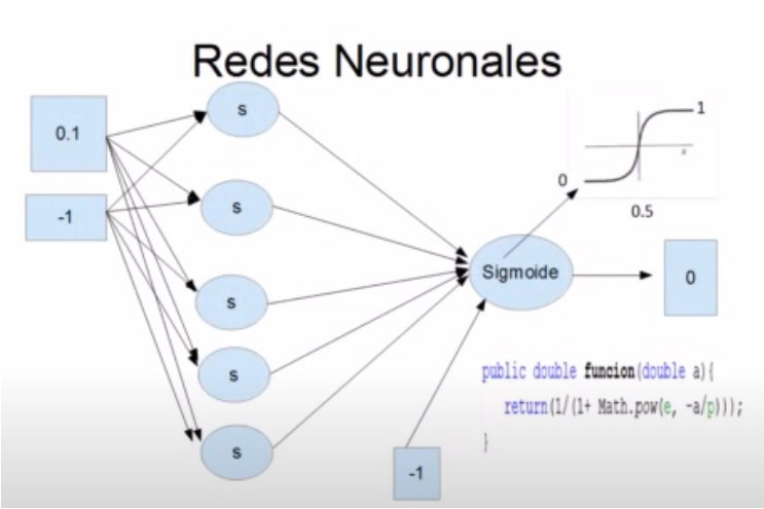
una red neuronal; es decir, es una sola neurona con la función de activación de identidad. En esto sección(Vasilev et al., 2019).

Regresión logística

La regresión logística utiliza la activación sigmoidea logística, en contraste con la regresión lineal, que utiliza la función de identidad. Como hemos visto antes, la salida del sigmoide logístico está en el (0,1) rango y se puede interpretar como una función de probabilidad. Podemos usar regresión logística para un problema de clasificación de 2 clases (binario), donde nuestro objetivo, t, puede tener dos valores, generalmente 0 y 1 para las dos clases correspondientes. Estos valores discretos no deberían ser confundidos con los valores de la función sigmoidea logística, que es un valor real continuo función entre 0 y 1(Vasilev et al., 2019).

Figura 15

Función Sigmoide



Nota: Extraído de (Viera Balanta, 2015)

Retro propagación

Hasta ahora, hemos aprendido cómo actualizar los pesos de las redes de 1 capa

con gradiente descendencia. Comenzamos comparando la salida de la red (es decir, la salida de la capa de salida) con el valor objetivo, y después actualizamos los pesos. Pero en una red multicapa solo podemos utilizar esta técnica para los pesos que conectan el final de la capa oculta a la capa de salida. Lo que haremos en su lugar es calcular el error en el final oculto. Capa y estime cuál podría ser en la capa anterior, esparciremos ese error de regreso de la capa final a la primera capa, entonces obtenemos el nombre retro propagación(Vasilev et al., 2019).

2.1.2.4.6 Metodología de Diseño

"Para realizar un proyecto de redes neuronales artificiales es necesario planificar algunas faces importantes"(Vasilev et al., 2019).

Entender el Problema

"Se basa en gran medida en una comprensión adecuada del problema, en particular las relaciones "causa-efecto". Los beneficios de las ANN sobre otras técnicas deben evaluarse antes de la selección final de la técnica de modelado"(Basheer & Hajmeer, 2001).

El Diseño del Sistema

Es el primer paso en el diseño de ANN real en el que el modelador determina el tipo de ANN y reglas de aprendizaje que se ajuste al problema. Esta fase también incluye la recopilación de datos, el preprocesamiento de datos para ajustar el tipo de ANN utilizado, el análisis estadístico de los datos y la división de los datos en tres subconjuntos distintos subconjuntos de entrenamiento, prueba y validación(Basheer & Hajmeer, 2001).

La Realización del Sistema

Incluye el entrenamiento del modelo usando la data pre procesada y organizada, y al mismo tiempo evaluar el rendimiento de la red mediante el análisis del error en la predicción. La selección óptima de los parámetros de la red como su tamaño la tasa de aprendizaje el número de ciclos y la tasa de error que será aceptable, esto puede afectar el diseño y el rendimiento de la red final. Dividir el problema en sus problemas más pequeños, si es posible, y diseñar un conjunto de redes podría mejorar la precisión general del sistema. Esto lleva al modelador a la fase 2(Skansi, 2018).

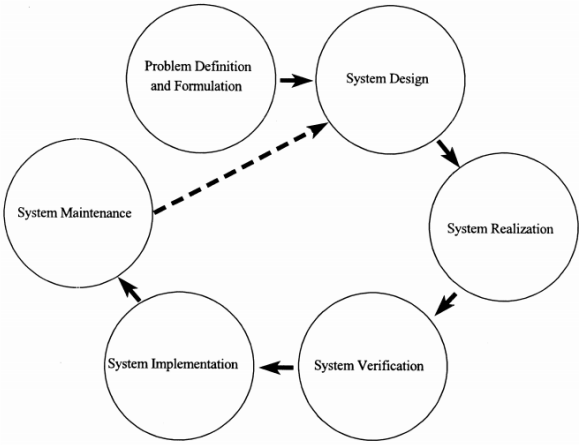
En la Verificación del Sistema

Es importante examinar con datos de evaluación para elegir a la mejor red. La verificación tiene por objetivo llegar a confirmar la capacidad del modelo basado en ANN para responder con precisión. Esta fase también incluye comparar el performance del modelo basado en ANN con los de otros enfoques (si están disponibles) como la regresión estadística y los sistemas expertos. La implementación del sistema incluye incrustar la red obtenida en un sistema de trabajo apropiado. La prueba final del sistema integrado también debe llevarse a cabo antes de su lanzamiento al usuario final(Basheer & Hajmeer, 2001).

El mantenimiento del sistema implica actualizar el sistema desarrollado a medida que ocurren cambios en el entorno o las variables del sistema (por ejemplo, nuevos datos), lo que implica un nuevo ciclo de desarrollo.(Basheer & Hajmeer, 2001).

Figura 16

Metodología de Diseño de Una Red Neuronal



Nota: Extraído de (Basheer & Hajmeer, 2001)

Definición de la Red Neuronal

Se determina la arquitectura que llevará el modelo todo esto orientado a lo que se quiere lograr y el tipo de problema que se abordará. Definiremos aspectos como el tipo de modelo, algoritmos de optimización de errores y aprendizaje etc, (Basheer & Hajmeer, 2001).

Entrenamiento de la Red Neuronal

"Definiremos la forma y paquetes de entrenamiento que tendrá el modelo en función a nuestra Dataset tomando en cuenta el peso de este y la lo para metro previamente definidos en nuestro modelo"(Basheer & Hajmeer, 2001).

Utilización de la Red Neuronal

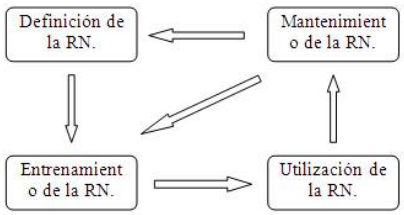
"En la utilización de nuestra red en los primeros problemas de predicción usando un valor de entrada y etiquetándolo para mostrar la salida en función a los pesos del modelo"(Basheer & Hajmeer, 2001).

Mantenimiento de la Red Neuronal

"Como toda tecnología, estar en un constante proceso de ajuste y afinación es necesario ya sea por la obtención de nuevas entradas para el aprendizaje como ajuste en la curva de aprendizaje del modelo"(Basheer & Hajmeer, 2001).

Figura 17

Etapas de Construcción en una Red



Nota: Extraído de (Basheer & Hajmeer, 2001)

2.1.2.5 Deep Learning

La reciente disponibilidad de datos a gran escala y de alta calidad y nuevas infraestructuras de computación paralela han revitalizó las redes neuronales en términos de tamaño y complejidad. Este nuevo auge de muchas topologías nuevas y complejas se denomina aprendizaje profundo. Allí han sido desarrollos en el procesamiento de imágenes y video, reconocimiento de voz y subtítulos de video automatizados basados en sistemas de aprendizaje profundo. Sin embargo, esto sigue siendo un Área de investigación muy activa. Afortunadamente, la gran empresa con importantes inversiones en esta área ha hecho gran parte de su software de investigación de código abierto con los correspondientes enlaces de Python. Para desarrollar nuestra comprensión de redes neuronales(Basheer & Hajmeer, 2001).

En cierto modo telar Jacquard sembró las semillas de lo que es el aprendizaje profundo hoy, la red neuronal profunda no solo tiene la capacidad de clasificar, sino identificar

detalles del objeto clasificado. Una red profunda que aprende representaciones básicas de su salida puede hacer clasificaciones usando las suposiciones que ha hecho. Por ejemplo, si no hay una cola peluda, probablemente no será una ardilla, sino más bien un gato. De esta manera, la cantidad de información que aprende la red es mucho más completa y robusta, y la parte más emocionante es que las redes neuronales profundas aprenden a hacer esto automáticamente(Basheer & Hajmeer, 2001).

2.1.2.5.1 Aprendizaje de funciones

Para ilustrar cómo funciona el aprendizaje profundo, consideremos la tarea de reconocer una simple figura geométrica, por ejemplo, un cubo, como se ve en la siguiente ilustración. El cubo es compuesto por aristas (o líneas), que se cruzan en vértices. Digamos que cada punto posible en el espacio tridimensional está asociado con una neurona.(Basheer & Hajmeer, 2001)

Todos los puntos / neuronas están en la primera (entrada) capa de una red de alimentación directa multicapa. Un punto de entrada / neurona está activa si el punto correspondiente se encuentra en una línea. Los puntos / neuronas que se encuentran en una línea común (borde) tienen fuertes conexiones positivas con un solo borde / neurona común en la siguiente capa. Por el contrario, tienen conexiones negativas con todas las demás neuronas de la siguiente capa. Lo único la excepción son las neuronas que se encuentran en los vértices. Cada una de estas neuronas se encuentra simultáneamente en tres bordes, y está conectado a sus tres neuronas correspondientes en la capa siguiente. (Vasilev et al., 2019)

Ahora poseemos varias capas ocultas que hacen funciones de abstracción, la primera para los puntos y el segundo para los bordes. Pero esto no es idóneo para codificar un

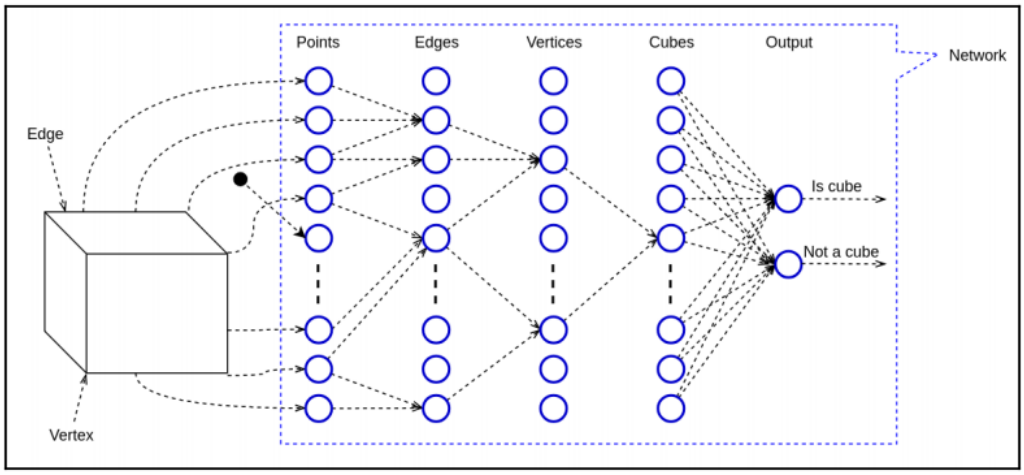
cubo completo en la red. Usaremos una capa para los vértices(Basheer & Hajmeer, 2001).

Aquí, cada tres neuronas activas del segundo capa, que forma un vértice, poseen una conexión positiva significativa con una solo común neurona de la tercera capa. Dado que un borde del cubo forma dos vértices, cada uno neurona tendrá conexiones positivas a dos neuronas y negativo conexiones con todos los demás(Basheer & Hajmeer, 2001).

Finalmente, presentaremos la última capa oculta (cubo). En la cuarta Las neuronas que forman un cubo tendrán conexiones positivas con una solo neurona de la capa(Basheer & Hajmeer, 2001):

Figura 18

Deep Learning (Multitud de Neuronas)



Nota: Extraído de(Basheer & Hajmeer, 2001)

El ejemplo de representación del cubo está muy simplificado, pero podemos sacar varias conclusiones. de eso. Uno de ellos es que las redes neuronales profundas se prestan bien a datos organizados. Por ejemplo, una imagen consta de píxeles, que forman líneas, bordes, regiones, y así, esto también es cierto para el habla, donde los

componentes básicos se denominan fonemas; así como texto, donde tenemos caracteres, palabras y oraciones(Vasilev et al., 2019).

2.1.2.5.2 Algoritmos de Deep Learning

Podríamos definir el aprendizaje profundo dentro de las técnicas de aprendizaje automático, Donde se diferencian es en la arquitectura de red (o la forma en que se organizan las capas o neuronas de la red) y también en la forma en la que se entrenan. Veamos los principales algoritmos de Deep Learning en uso hoy(Vasilev et al., 2019):

Redes Neuronales Convolucionales (CNN:

Una CNN es una red neuronal de alimentación red con varios tipos de capas especiales. Por ejemplo, capas convolucionales aplicar un filtro a la imagen de entrada (o sonido) deslizando ese filtro por toda la señal entrante, para producir un mapa de activación n-dimensional. Hay alguna evidencia de que las neuronas en las CNN están organizadas de manera similar a cómo las células biológicas están organizadas en la corteza visual del cerebro. Hemos mencionado varias CNN veces hasta ahora, y eso no es una coincidencia; Hoy, superan a todos los demás Algoritmos de aprendizaje automático en una gran cantidad de tareas de visión artificial y PNL(Vasilev et al., 2019).

Redes Recurrentes:

La arquitectura de esta red se basa en una memoria que se trabaja en la totalidad o parte de los datos de entrada a la red. La salida de una red recurrente es una combinación de memoria interna y la última muestra de entrada. Así mismo, la memoria interna cambia, por el ingreso de nuevos datos. Por estas propiedades, las redes recurrentes son buenas para tareas que funcionan con datos en secuencias, como

procesamiento de texto o datos de series temporales(Vasilev et al., 2019).

2.1.2.5.3 Aplicaciones del Aprendizaje Profundo

El aprendizaje automático en general, y el aprendizaje profundo en particular, están produciendo cada vez más resultados asombrosos en términos de la calidad de las predicciones, detección de características y clasificación. Muchos de estos resultados recientes han sido noticia. Tal es el ritmo de progreso, que a algunos expertos les preocupa que las máquinas pronto sean más inteligentes que los humanos. Pero espero que esos temores que pueda tener se alivien después de haber Lee este libro. Para bien o para mal, todavía estamos lejos de la inteligencia a nivel humano(Vasilev et al., 2019).

2.1.2.5.4 Presentamos Bibliotecas Populares de Código Abierto

Hay muchas bibliotecas de código abierto que permiten la creación de redes neuronales profundas en Python, sin tener que escribir explícitamente el código desde cero. En este libro, usaremos tres de los más populares: - *Tensorflow, Keras y Pytorch.* Todos comparten algunas características comunes, como las siguientes(Vasilev et al., 2019):

TensorFlow

Tensorflow (TF), es el aprendizaje profundo más popular biblioteca. Está desarrollado y mantenido por Google. No es necesario que exija explícitamente el uso de una *Gpu*; en su lugar, *Tensorflow* intentará usarlo automáticamente si tiene uno. Si usted tiene más de una *Gpu*, debe asignar operaciones a cada *Gpu* de forma explícita, o solo la primera uno será usado.(Vasilev et al., 2019).

Keras

Keras, es una biblioteca de Python de red neuronal de alto nivel que se ejecuta sobre Tensorflow, *Cntk* o Theano. Para los fines de este libro, asumiremos que usa *Tensorflow*

en el backend. Con *Keras*, puedes realizar experimentación rápida y es relativamente fácil de usar en comparación con TF. Va a detectar automáticamente una *Gpu* disponible e intenta utilizarla. De lo contrario, volverá al *Upc*. Si desea especificar el dispositivo manualmente, puede importar TensorFlow y usar el mismo código que en la sección anterior.(Vasilev et al., 2019).

PyTorch

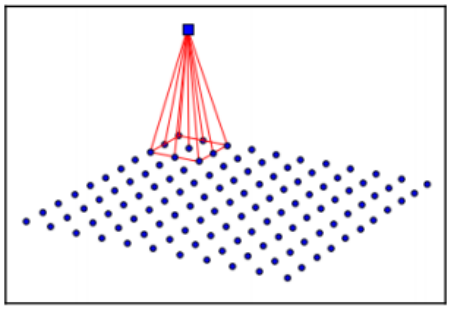
"Biblioteca de aprendizaje profundo basada en Torch y desarrollada por Facebook. Es relativamente fácil de usar y recientemente ha ganado mucha popularidad. Va a selec-cione automáticamente una *Gpu*, si hay una disponible, volviendo a la CPU en caso contrario"(Vasilev et al., 2019).

2.1.2.6 Redes Neuronales Convolucionales

La capa convolucional es el componente más importante de una CNN. Consiste en un conjunto de filtros (también conocidos como kernels o detectores de características), donde cada filtro se aplica a todas áreas de los datos de entrada. Un filtro se define mediante un conjunto de ponderaciones que se pueden aprender. Como un guiño a la tema en cuestión, la siguiente imagen ilustra esto muy bien:(Vasilev et al., 2019).

Figura 19

Redes Neuronales Convolucionales



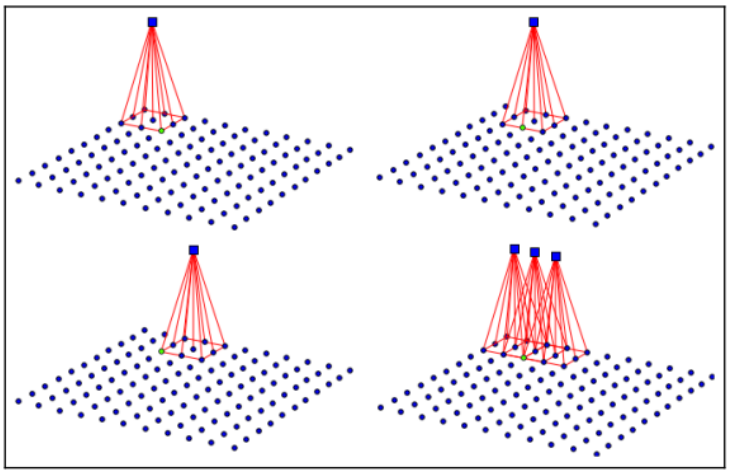
Nota: Extraído de (Basheer & Hajmeer, 2001)

Se muestra una capa de entrada bidimensional de una red neuronal. Por el bien de la simplicidad, asumiremos que esta es la capa de entrada, pero puede ser cualquier capa de la red. Como tenemos visto en los capítulos anteriores, cada neurona de entrada representa la intensidad de color de un píxel (asumiremos que es una imagen en escala de grises por simplicidad). Primero, aplicaremos un filtro de 3x3 en la esquina superior derecha de la imagen. Cada neurona de entrada está asociada con un solo peso del filtro. Tiene nueve pesos, debido a las nueve neuronas de entrada, pero, en general, el tamaño es arbitrario. (2x2, 4x4, 5x5, etc.). La salida del filtro es una suma ponderada de sus entradas (la activación de las neuronas de entrada). Su propósito es resaltar una característica específica en la entrada, por ejemplo, un borde o una línea. El grupo de neuronas cercanas, que participan en la entrada se denomina campo receptivo. En el contexto de la red, la salida del filtro representa el valor de activación de una neurona en la siguiente capa. La neurona estará activa, si la función está presente en esta ubicación espacial.(Vasilev et al., 2019).

Hasta ahora, hemos calculado la activación de una sola neurona. ¿Y los demás? ¡Es simple! Para cada nueva neurona, deslizaremos el filtro a través de la imagen de entrada y calcularemos su salida (la suma ponderada) con cada nuevo conjunto de neuronas de entrada. En el siguiente diagrama, puede ver cómo calcular las activaciones de las dos posiciones siguientes (un píxel por Correcto)(Vasilev et al., 2019):

Figura 20

Redes Neuronales Convolucionales Segunda Parte



Nota: Extraído de(Basheer & Hajmeer, 2001)

A medida que el filtro se mueve por la imagen, calculamos los nuevos valores de activación para las neuronas en el segmento de salida. Al decir "deslizar", queremos decir que los pesos del filtro no cambian en la imagen. En efecto, usaremos los mismos nueve pesos de filtro para calcular las activaciones de todas las salidas neuronas, cada vez con un conjunto diferente de neuronas de entrada. A esto lo llamamos compartir parámetros, y lo hacemos por dos motivos(Vasilev et al., 2019):

- Al reducir el número de pesos, reducimos la huella de memoria y evitar el sobre ajuste.
- El filtro resalta características específicas. Podemos suponer que esta función es útil, independientemente de su posición en la imagen. Al compartir los pesos, garantizamos que el filtro podrá localizar la característica en toda la imagen.

2.1.2.6.1 Estructura de una Red Neuronal Convolucional

Antes de continuar, recopilemos todo lo que hemos aprendido hasta ahora. En la figura

A continuación podemos ver la estructura de una CNN básica:

Normalmente alternaríamos una o más capas convolucionales con una agrupación capa. De esta manera, las capas convolucionales pueden detectar características en todos los niveles del tamaño del campo receptivo. El tamaño del campo receptivo agregado de las capas más profundas es más grande que los que están al principio de la red. Esto les permite capturar características más complejas en regiones de entrada más grandes. Ilustremos esto con un ejemplo. Imagina que la red usa convoluciones 3x3 con stride 1 y 2x2 agrupación con zancada 2:(Vasilev et al., 2019)

La mayoría de las CNN comparten propiedades básicas. Estos son algunos de ellos:

- Las neuronas de la primera capa convolucional recibirán información de 3x3 píxeles de la imagen.
- Un grupo de neuronas de salida 2x2 de la primera capa tendrá un tamaño combinado del campo receptivo de 4x4 (debido a la zancada).
- Después de la primera operación de agrupación, este grupo se combinará en una neurona única de la capa de agrupación.
- La segunda operación de convolución toma la entrada de la agrupación 3x3 neuronas. Por lo tanto, recibirá información de un cuadrado con un lado 3x4 = 12 (o un total de 12x12 = 144) píxeles de la imagen de entrada.

Usamos las capas convolucionales para extraer características de la entrada. Las características detectadas por las capas más profundas son muy abstractas, pero tampoco

son legibles por los humanos. Para resolver este problema, generalmente agregamos uno o más capas después de la última capa convolucional / agrupada. En este ejemplo, la última capa completamente conectada (salida) utilizará softmax para estimar las probabilidades de clase de la entrada. Puede pensar en las capas completamente conectadas como traductores entre el idioma de la red (que no entendemos) y el nuestro. (Vasilev et al., 2019).

Las capas convolucionales más profundas suelen tener más filtros (por lo tanto, mayor volumen profundidad), en comparación con los iniciales. Un detector de características al comienzo de la red trabaja en un pequeño campo receptivo. Solo puede detectar un número limitado de características, como bordes o líneas, compartidas entre todas las clases. Por otro lado, una La capa más profunda detectaría características más complejas y numerosas. Por ejemplo, si tenemos varias clases, como coches, árboles o personas, cada una tendría su propio conjunto de características como neumáticos, puertas, hojas y caras, etc. Esto sería requieren más detectores de funciones.(Vasilev et al., 2019).

2.1.2.6.2 Mejorando el Desempeño de CNN

Pre procesamiento de datos

Hasta ahora, hemos alimentado la red con entradas no modificadas. En el caso de las imágenes, estas son intensidades de píxeles en el rango [0: 255]. Pero eso no es óptimo. Imagina que tenemos un RGB imagen, donde la intensidad en uno de los canales de color es muy alta en comparación con el otro dos. Cuando alimentemos la imagen a la red, los valores de este canal se convertirán dominante, disminuyendo a los demás. Esto podría sesgar los resultados, porque en realidad cada canal tiene la misma impor-

tancia. Para resolver esto, necesitamos preparar (o normalizar) los datos, antes de alimentarlo a la red(Vasilev et al., 2019).

Regularización

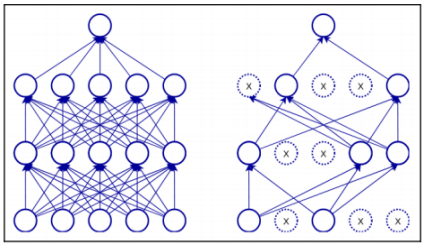
La regularización es cualquier modificación que hacemos a un algoritmo de aprendizaje con la intención de reducir su error de generalización, pero no su error de entrenamiento.

Abandonar

La deserción es una técnica de regularización, que se puede aplicar a la salida de algunas de las capas de red. La deserción elimina de forma aleatoria y periódica algunas de las neuronas (junto con sus conexiones de entrada y salida) de la red. Durante un entrenamiento de mini lotes, cada neurona tiene una probabilidad p de caer estocásticamente. Esto es para asegurar que una neurona termina confiando demasiado en otras neuronas y "aprende" algo útil para la red en su lugar. La deserción se puede aplicar después de convolución, agrupación o conexión completa en capas. En la siguiente ilustración, podemos ver un abandonar de las capas completamente conectadas(Vasilev et al., 2019).

Figura 21

Abandono de las Capas



Nota: Extraído de (Basheer & Hajmeer, 2001)

Aumento de datos

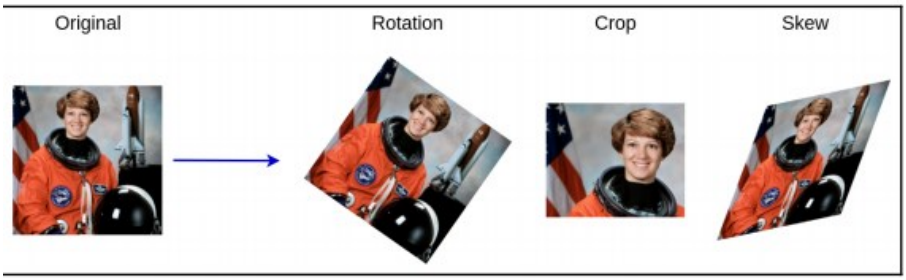
Una de las técnicas de regularización más eficaces es el aumento de datos. Si el entrenamiento los datos son demasiado pequeños, es posible que la red comience a adaptarse. El aumento de datos ayuda a contrarrestar esto aumentando artificialmente el tamaño del conjunto de entrenamiento. Usemos un ejemplo. En el MNIST y Ejemplos de CIFAR-10, hemos entrenado la red en varias épocas. La red "ver" cada muestra del conjunto de datos una vez por época. Para evitar esto, podemos aplicar aleatoriamente aumentos a las imágenes, antes de usarlas para entrenamiento. Las etiquetas permanecerán iguales. Algunos de los aumentos de imagen más populares son:(Vasilev et al., 2019).

- Rotation
- Horizontal and vertical
- flip Zoom in/out
- Crop
- Skew
- Contrast and brightness adjustment

Los aumentos envalentonados se muestran en el siguiente ejemplo:

Figura 22

Rotación de Imagen Uso de Redes Convolucionales



Nota: Extraído de (Basheer & Hajmeer, 2001)

Normalización por lotes

En Pre procesamiento de datos, explicamos por qué es importante la normalización de datos. Lote La normalización proporciona una forma de aplicar el procesamiento de datos, similares a la puntuación estándar, para las capas ocultas de la red. Normaliza las salidas de la capa oculta para cada mini-lote (de ahí el nombre) de una manera, que mantiene su valor medio de activación cerca de 0, y su desviación estándar cercana a 1. Podemos usarlo con capas convolucionales y completamente conectadas. Las redes de normalización de lotes se entrenan más rápido y pueden usar más tasas de aprendizaje. Para obtener más información sobre la normalización de lotes, consulte el original Normalización de lotes de papel: aceleración del entrenamiento de redes profundas mediante la reducción de co-variables internas Shift, de Sergey Ioffe y Christian Szegedy, que se puede ver en el siguiente enlace: Lote Normalización: aceleración del entrenamiento de redes profundas reduciendo Cambio de covarianza.(Vasilev et al., 2019).

2.1.2.7 Visión por Computadora Avanzada

Arquitecturas de red avanzadas

Ahora estamos familiarizados con la poderosa técnica del aprendizaje por transferencia. En esta sección, discutir algunas arquitecturas de red recientes y populares que van más allá de las que hemos vistohasta aquí. Podrás utilizar estas redes como modelos previamente entrenados en un aprendizaje de transferencia escenario, o si eres lo suficientemente valiente, entrénalos desde cero para resolver tus tareas.(Vasilev et al., 2019).

- Transfer Learning
- Advanced Network
- Architectures
- Capsule Networks
- Object Detection
- Semantic Segmentation
- Artistic Style Transfer

Detección de objetos

La detección de objetos es el proceso de encontrar instancias de objetos de una determinada clase, como caras, automóviles y árboles, en imágenes o videos. A diferencia de la clasificación, la detección de objetos puede detectar múltiples objetos, así como su ubicación en la imagen. Un detector de objetos devolvería una lista de objetos detectados con la siguiente información para cada objeto: La clase del objeto (persona, automóvil, árbol, etc.). Probabilidad (o puntuación de confianza) en el rango [0, 1], que transmite la confianza el detector es que el objeto existe en ese lugar. Esto es similar a la salida de un clasificador regular. Las coordenadas de la región rectangular de la imagen donde se encuentra el objeto. Situado este rectángulo se llama cuadro delimitador(Vasilev et al., 2019).

Podemos ver el resultado típico de un algoritmo de detección de objetos en la siguiente fotografía. El tipo de objeto y la puntuación de confianza están encima de cada cuadro delimitador(Vasilev et al., 2019).

Figura 23

Detección de Objetos

Nota: Extraído de (Basheer & Hajmeer, 2001

2.1.2.8 Transfer learning

Transfer Learning es una técnica de Machine Learning mediante la cual un modelo es entrenado y desarrollado para una tarea. Luego se reutiliza en una segunda tarea relacionada. Se refiere a la situación en la que lo que se ha aprendido en un entorno se explota para mejorar la optimización en otro escenario. Transfer Learning se suele aplicar cuando hay un nuevo conjunto de datos más pequeño que el conjunto de datos original utilizado para entrenar el modelo previamente entrenado.(Kar & Safari, 2020). Si las capas densas trabajan directamente con capas que tienen un mayor nivel de abstracción, el desempeño del modelo es mejor y conseguiremos:

- El número de neuronas de entradas puede ser reducida.
- El nuevo modelo requiere de menos cantidad de datos de entrenamiento.

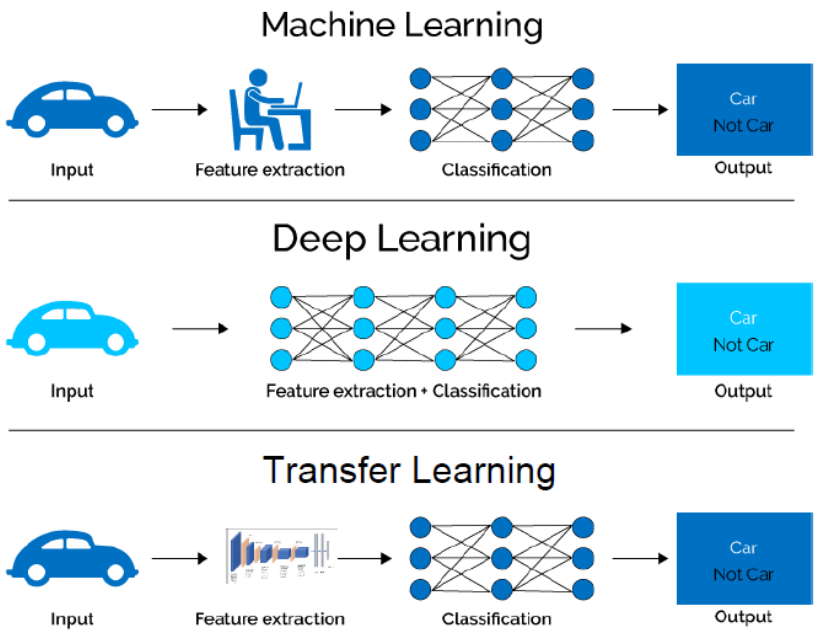
Básicamente tenemos que extraer el vector de características desde el modelo base sin la capa superior entrenada en un conjunto de datos de ImageNet (Dataset con 14

millones de imágenes y 1000 clases) y luego agregue nuestra capa totalmente conectada personalizada, incluida la activación, el abandono y softmax, para constituir nuestro modelo final. Congelamos el modelo base, pero en la capa superior del recién agregado los componentes permanecen descongelados. Entrenamos el modelo recién creado en nuestros propios conjuntos de datos para generar predicciones. Todo este proceso de transferir mapas de características aprendidos de un gran modelo y luego personalizarlos en nuestros propios conjuntos de datos ajustando el orden superior parámetros del modelo se llama transferencia de aprendizaje. (Kar & Safari, 2020).

Podemos ver en la ilustración 2-25 que el Transfer Learning conserva todas las fases tradicionales de un algoritmo e Deep learning mas es su estructura ya entrenada y pasos para ser generada lo que la diferencian así mismo la efectividad del modelo aumentaría.

Figura 24

Diferencias con Transfer Learning

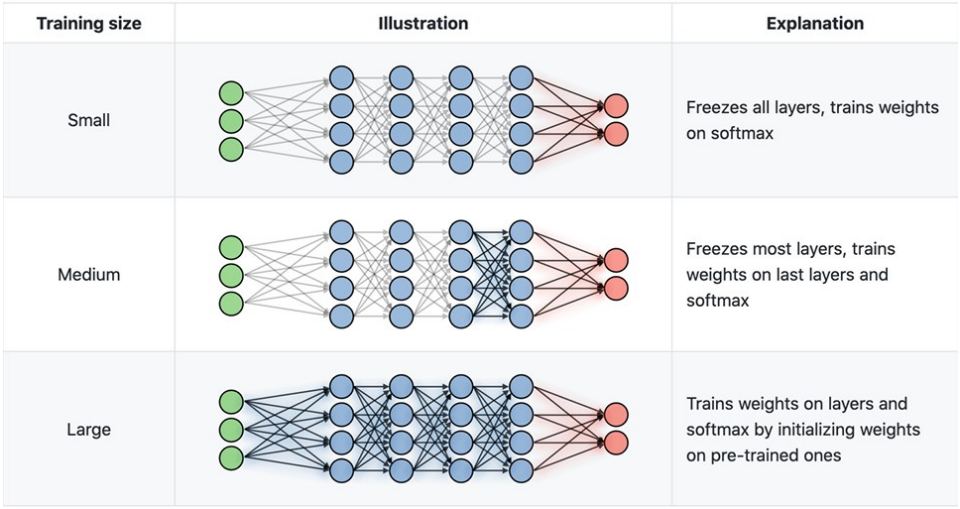


Nota: Extraído del curso Machine Learning para la industria clase, numero 8. (Soledad Espezúa, 2021)

La técnica de Transfer Learning nos permite hacer uso del modelo pre entrenado no solo para quitar la cabecera del modelo y proceder a pasar nuestra Dataset por el modelo, sino que podemos segregar distintas capas y filtros más específicos para así poder entrenarlos y que puedo dar resultados más favorables conforme a como se analice el historial de aprendizaje de la red(Soledad Espezúa, 2021).

Figura 25

Formas de Aplicación de Transfer Learning



Nota: Extraído de (Srihari, 2020)

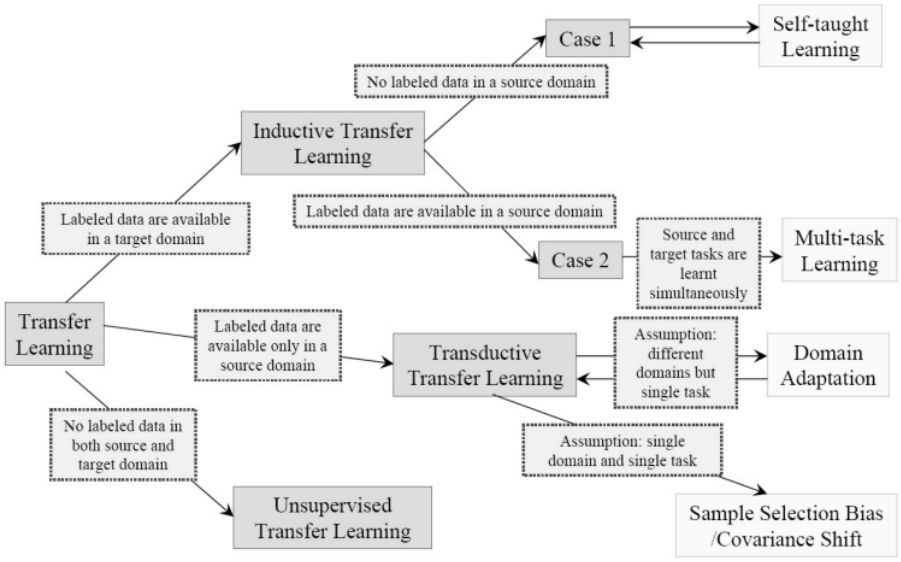
Definiremos los siguientes pasos para realizar la técnica de Transfer Learning.

- Analizar y almacenar datos.
- Configuración de parámetros del modelo
- Creación de una canalización de datos de entrada
- Generador de datos de entrenamiento
- Generador de datos de validación

- Construcción del modelo final utilizando el aprendizaje por transferencia
- Guardar un modelo con puntos de control
- Trazado del historial de entrenamiento

Figura 26

Ramas del Transfer Learning



Nota: Extraído de (Pan & Yang, 2010)

2.1.2.8.1 Modelos pre entrenadas de Deep Learning

Los avances en el campo del Deep Learning han permitido la existencia de modelos distintos con distintas características, el caso de las redes convolucionales es así que hay distintas propuestas de arquitecturas de redes de convolucion para aplicarse en distintos casos

Cambios como la cantidad de capas convolucionales, número de filtros para toda la red ETC son aspecto que se conocerán de las siguientes redes(TeamKeras, 2020):

AlexNet

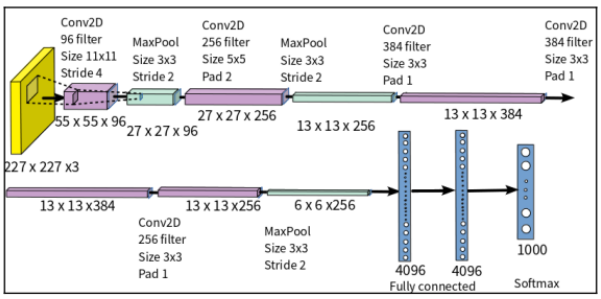
AlexNet fue presentado en 2012 por Alex Krizhevsky, Ilya Sutskever y Geoffrey E.

Hinton en un artículo titulado Clasificación de ImageNet con redes neuronales convolucionales profundas.

Fue la primera introducción exitosa de un modelo CNN optimizado para resolver problemas informáticos. Problemas de visión artificial relacionados con la clasificación de un gran número de imágenes (más de 15 millones) de muchas categorías diferentes (más de 22.000). Antes de AlexNet, problemas de visión por computadora fueron resueltas principalmente por métodos tradicionales de Machine Learning, lo que hizo incrementales mejoras al recopilar conjuntos de datos más grandes y mejorar el modelo y las técnicas para minimizar el sobreajuste. Los modelos CNN clasifican las tasas de error en términos de una tasa de error de los cinco primeros, que es el porcentaje de instancias en las que la verdadera clase de una imagen determinada no se encuentra entre las cinco principales predicciones clases AlexNet ganó el ILSVRC 2012 (reconocimiento visual a gran escala de ImageNet). Desafío) al tener una tasa de error entre los cinco primeros del 15,3%, tomando una ventaja significativa sobre el segundo mejor resultado, que obtuvo una tasa de error entre los cinco primeros del 26,2%. La arquitectura de AlexNet es se muestra en el siguiente diagrama:(Srihari, 2020)

Figura 27

Arquitectura AlexNet Fuente



Nota: Extraído de (Kar & Safari, 2020)

VGG16

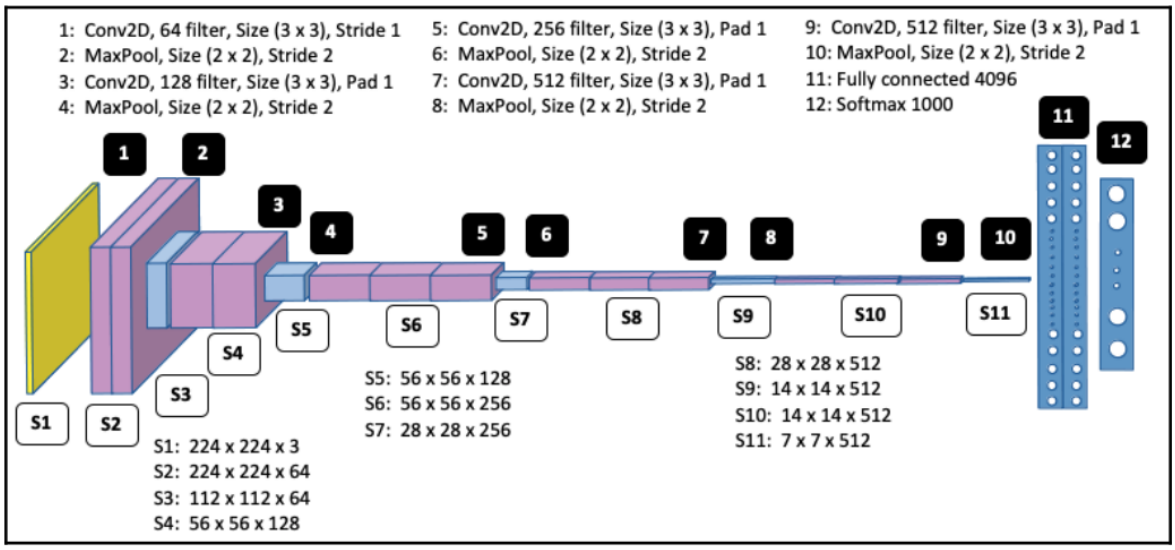
Tras el éxito de AlexNet en 2012, cada vez más investigadores trabajaron para mejorar la Arquitectura CNN de AlexNet para mejorar la precisión. El foco se desplazó a una ventana más pequeña, tamaño, filtros más pequeños y pasos más pequeños. VGG16 fue presentado en 2014 por Karen Simonyan y Andrew Zisserman en el artículo titulado Very Deep Convolutional Networks for Large-Scale Reconocimiento de imagen. El documento se puede leer en https://arxiv.org/abs/1409.1556. El modelo logró una precisión de prueba entre los cinco primeros del 92,7 % en ImageNet en ILSVRC-2014. La arquitectura VGG16 se muestra en la siguiente imagen(Team Keras, 2021):

Algunas características**:**

- El tamaño máximo del filtro es 3 x 3 y el tamaño mínimo es 1 x 1. Esto significa que un se utiliza un tamaño de filtro más pequeño con una cantidad mayor, en comparación con un tamaño de filtro más grande y menor cantidad para AlexNet.
- El paso de convolución es 1 y el relleno es 1 para una capa convolucional de 3 x 3. máx. la puesta en común se realiza sobre una ventana de 2 x 2 con un paso de 2.
- Se utilizan tres funciones ReLU no lineales en lugar de una sola en cada capa, lo que hace que la función de decisión sea más discriminatoria al reducir el problema de gradiente de fuga y permitir que la red aprenda profundamente. Aprendiendo profundamente aquí significa aprender formas complejas, como bordes, características, límites, y así.
- El número total de parámetros es de 138 millones.

Figura 28

Arquitectura VGG16



Nota: Extraído de (Kar & Safari, 2020)

ResNet

ResNet, presentado por Kaiminh He, Xiangyu Zhang, Shaoquing Ren y Jian Sun en el artículo titulado Deep Residual Learning for Image Recognition, fue desarrollado para abordar la Problema de degradación de la precisión de las redes neuronales profundas con un aumento en la profundidad. Estala degradación no es causada por el sobreajuste, sino que resulta del hecho de que después de alguna profundidad, la salida pierde la información de la entrada, por lo que la correlación entre la entrada y la salida comienza a divergir, lo que resulta en un aumento de la imprecisión. El papel se puede encontrar en https://arxiv.org/abs/1512.03385.

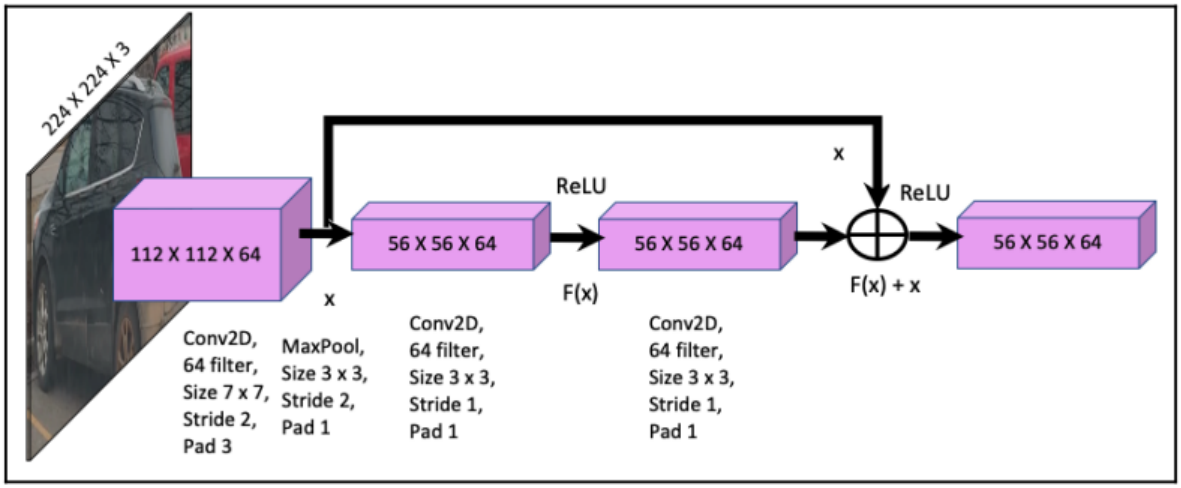
ResNet-34 logró un error de validación entre los cinco primeros del 5,71 %, mejor que BN-inception y VGG. ResNet-152 logra un error de validación entre los cinco primeros del 4,49 %. Unos conjuntos de seis modelos con diferentes profundidades lograron un

error de validación entre los cinco primeros del 3,57% y ganaron el primer lugar en ILSVRC-2015. ILSVRC significa reconocimiento visual a gran escala de ImageNet Competencia; evaluó algoritmos de detección de objetos y clasificación de imágenes desde 2010 hasta 2017. Las características clave de ResNet se describen a continuación(Team Keras, 2021)

- El problema de la degradación se aborda mediante la introducción de un profundo aprendizaje residual marco de referencia.
- El marco introduce el concepto de un atajo u omite la conexión, que salta una o más capas.
- El mapeo subyacente entre la entrada y la siguiente capa es H (x).
- La capa no lineal es F (x) = H (x) - x, que se puede reformular como H (x) = F (x) + x, donde x es el mapeo de identidad.

Figura 29

Arquitectura ResNet



Nota: Extraído de (Kar & Safari, 2020)

Inception

Antes de la introducción de la capa de inicio, la mayoría de las arquitecturas de CNN tenían un standard.

Convolucion apilada (en serie), normalización, agrupación máxima y activación capas seguidas de una capa totalmente conectada y softmax. Esta arquitectura dio lugar a una profundidad creciente de la red neuronal, que sufría de dos inconvenientes principales:

- Sobreajuste
- Mayor tiempo de cálculo

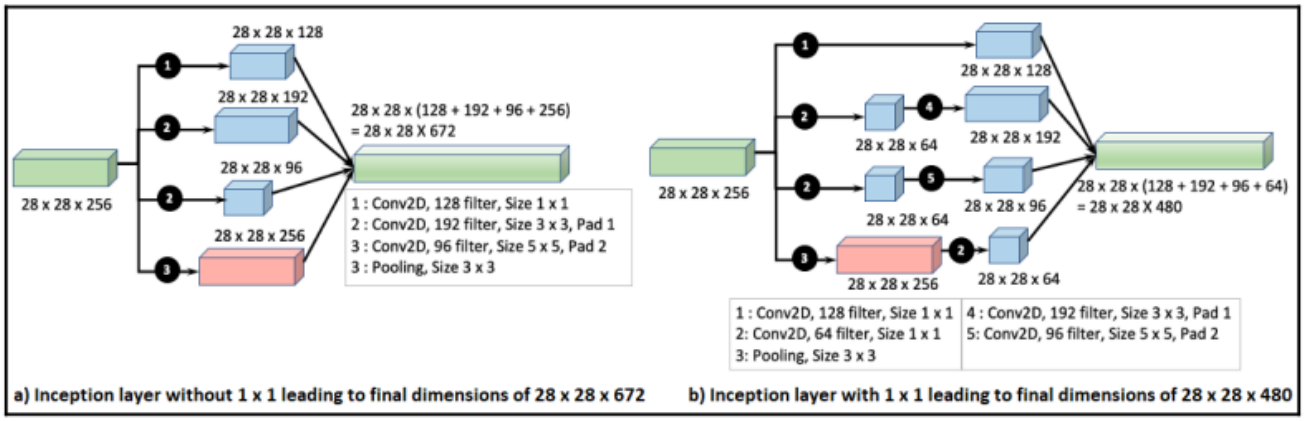
El modelo inicial resolvió ambos problemas al pasar de una red densa a matrices dispersas y agruparlos para formar submatrices densas. El modelo inicial también se conoce como GoogLeNet Fue presentado por Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragmir Anguelov, Dumitru Erhan, Vincent Vanhoucke y Andrew Rabinovich en un artículo titulado Going Deeper with Circunvoluciones. El nombre de Inception proviene del artículo Network in Network de Min Lin, Qiang Chen y Shuicheng Yan y el famoso meme de Internet Necesitamos profundizar más(TeamKeras, 2020).

La idea principal de la arquitectura de inicio se basa en averiguar cómo una estructura local óptima dispersa (múltiples 1 x 1 en paralelo) en una CNN puede ser complementado con componentes densos fácilmente disponibles (3 x 3 y 5 x 5). Los autores del artículo inicial encontraron la respuesta utilizando una convolución de 1 x 1 en paralelo con convolución de 3 x 3, 5 x 5 y una capa de agrupación. Un 1 x 1 adicional la convolución seguida de ReLU puede considerarse equivalente a la microred NIN. La convolucion 1 x 1 sirve como un mecanismo de reducción de dimensiones, y también

ayuda a aumentar el ancho de la red (al apilarlos uno al lado del otro), junto con su profundidad. El despliegue de múltiples circunvoluciones con múltiples filtros y capas de agrupación simultáneamente en paralelo dentro de la misma capa (capa de inicio) da como resultado que la capa sea una capa dispersa con un aumento en ancho(TeamKeras, 2020).

Figura 30

Arquitectura Incepción Fuente



Nota: Extraído de(Kar & Safari, 2020)

2.1.2.9 Diagnosticar el Rendimiento del Aprendizaje Automático

Una curva de aprendizaje el gráfico generado por la red para poder interpretar y hacer un seguimiento al aprendizaje del modelo en función del tiempo. “Las curvas de aprendizaje son una herramienta de diagnóstico ampliamente utilizada en el aprendizaje automático para algoritmos que aprenden de un conjunto de datos de entrenamiento de forma incremental” (Brownlee, 2019).

El modelo puede crear gráficos para analizar la curva en el proceso de entrenamiento y en el de validación para tener una comparativa de ambos en función el tamaño de Dataset

La interpretación de la curva de aprendizaje puede dar a conocer problemas dentro del

modelo o el proceso de aprendizaje, como un modelo de ajuste insuficiente o excesivo, así como también si los conjuntos de datos de entrenamiento y test son adecuadamente representativos de la totalidad de datos(Brownlee, 2019).

- Curvas de aprendizaje: Una curva de aprendizaje es una gráfica lineal en el tiempo el aprendizaje del modelo. Todo esto en el eje X e Y.
- Curva de aprendizaje del etrenamiento: la curva que se obtendrá del proceso de aprendizaje del modelo.
- Curva de aprendizaje del test: la curva que se obtendrá del proceso de validación del modelo.
- Curvas de aprendizaje de optimización: Curvas de aprendizaje en la métrica mediante se llegan a conocer aspectos para lograr optimizar los parámetros del modelo, como pueden ser el algoritmo de optimización, la función de error , o parámetros específicos del modelo como el uso de las capas o filtros en esta(Brownlee, 2019).
- Curvas de aprendizaje de rendimiento: curva la cual medirá le rendimiento de nuestro modelo en función a los aciertos obtenidos de nuestro modelo.

Diagnosticar un modelo

La curva de aprendizaje según sus bajas radicales o resultados en cierto tiempo pueden darnos a conocer ciertos problemas con nuestra red, es por eso que se conocen problemas generales lo cuales experimenta la curva de aprendizaje de un modelo de Deep Learning(Brownlee, 2019).

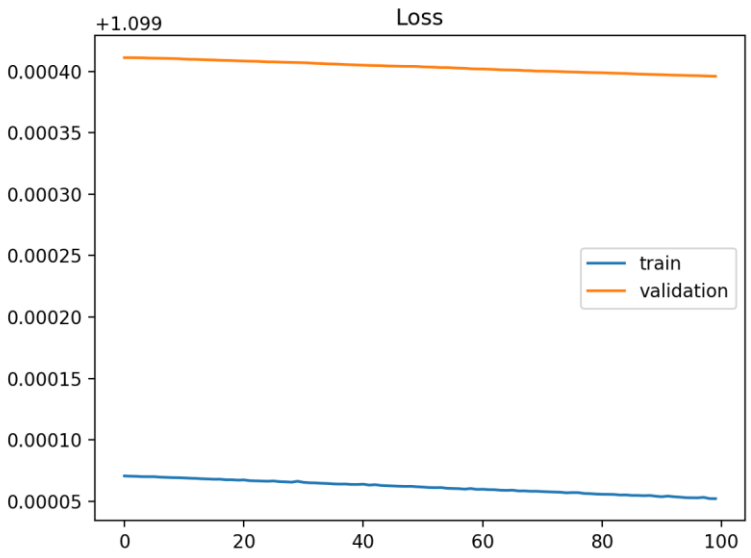
Curvas Underfit

Underfiting se trata de un modelo que no logro aprender en su totalidad o de

buena manera del conjunto de datos de entrenamiento . Para poder identificarlo con este. Puede mostrar una línea plana y /o valores con picos más conocidos como ruido y la línea de error en un alto grado de perdida , significa que el modelo no aprendió del Dataset .(Brownlee, 2019).

Figura 31

Curva Underfit

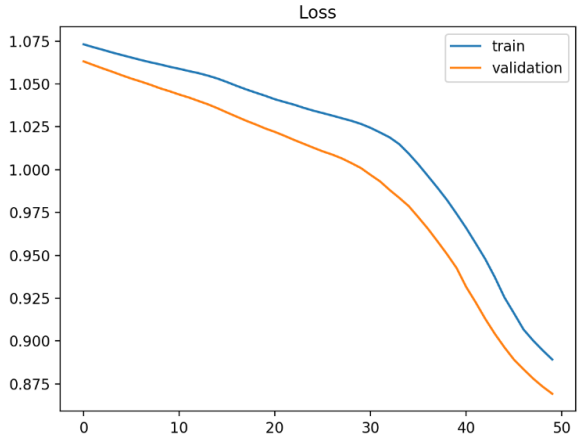


Nota: Extraído de (Brownlee, 2019)

Un modelo de Underfitting también puede asociarse a una función de error que baja y continúa bajando hasta llegar la final del gráfico. Esto quiere decir que el proceso de entrenamiento se truncó de forma prematura y con el debido ajusto el modelo si es capaz de aprender más(Brownlee, 2019).

Figura 32

Curva Underfitting 2



Nota: Extraído de (Brownlee, 2019)

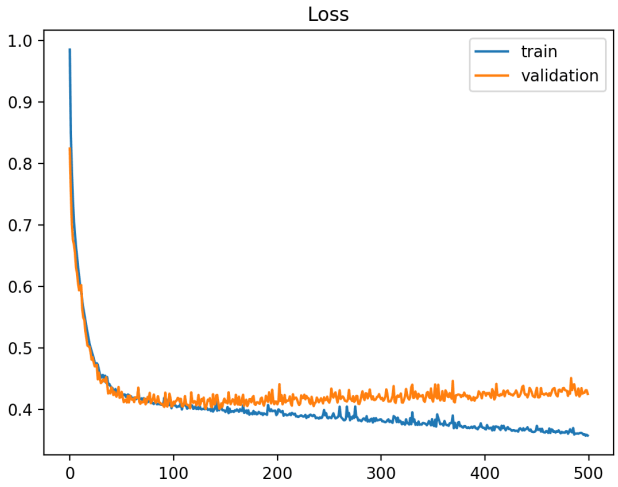
Curvas Overfit

El overfiting es un modelo que ha quedado en un sobre entrenamiento de su Dataset quedando de alguna manera muy cuadrada a los resultados que dará sin la capacidad de ser flexible a nuevos datos, incluyendo así en sus márgenes el ruido de los datos. El problema conlleva en que se perderá la capacidad de generalizar nuevos datos debido a esta especificación lo cual como dijimos deja al modelo sin flexibilidad a cambios, ya que aprendió demasiado bien su Dataset, lo que resulta en un aumento del error de generalización. Este diagnóstico se obtiene de la evaluación de la curva de validación donde veremos una baja significativa de la predicción más una subida enorme en el entrenamiento (Brownlee, 2019).

La gráfica de Overfiting tiene la pérdida del gráfico de validación disminuyendo hasta un punto de estabilidad (0) y posee una corta brecha con el Loss de Train.

Figura 33

Curva Overfit



Nota: Extraído de (Brownlee, 2019)

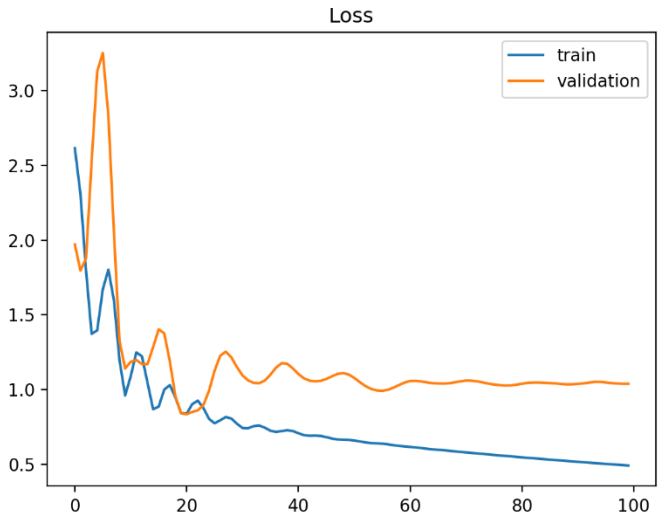
Conjuntos De Datos No Representativos

Un conjunto de datos que no representa su Dataset significa que en un análisis estadístico comparativo con datos sacados del mismo conjunto de datos este no se asemeja y por lo tanto no brinda una clara representación de este, como en el aprendizaje y la validación. Esto ocurre cuando el Dataset de muestra en ambos procesos de aprendizaje y validación son escasos o pequeños , en relación con otro conjunto de datos.(Brownlee, 2019). Podemos observar las siguientes conjeturas son:

- Dataset de Train poco representativo.
- Dataset de Test poco representativo.

Figura 34

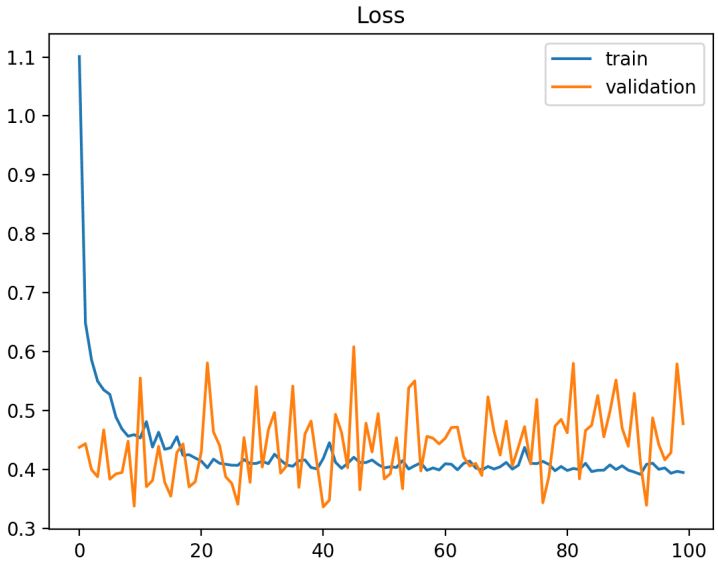
Conjunto de Datos de Entrenamiento no Representativo



Nota: Extraído de (Brownlee, 2019)

Figura 35

Conjunto de Datos de Validación no Representativo 2

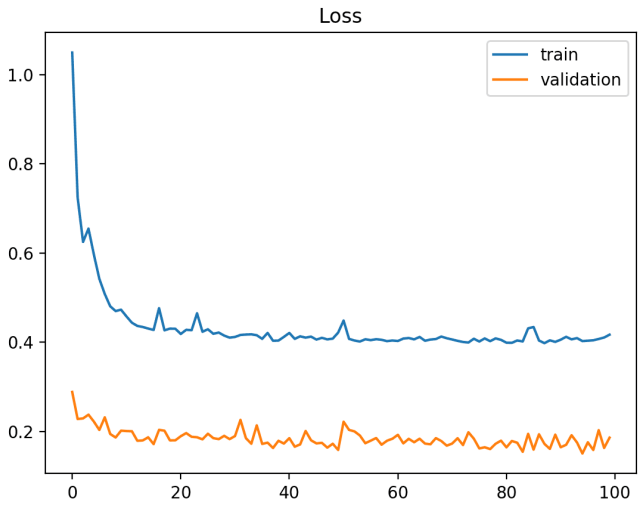


Nota: Extraído de (Brownlee, 2019)

También puede identificarse por una pérdida de validación inferior a la pérdida de entrenamiento. En este caso, indica que el conjunto de datos de validación puede ser más fácil de predecir para el modelo que el conjunto de datos de entrenamiento (Brownlee, 2019).

Figura 36

Conjunto de Datos de Validación no Representativo 3



Nota: Extraído de(Brownlee, 2019)

Modelo bien entrenado

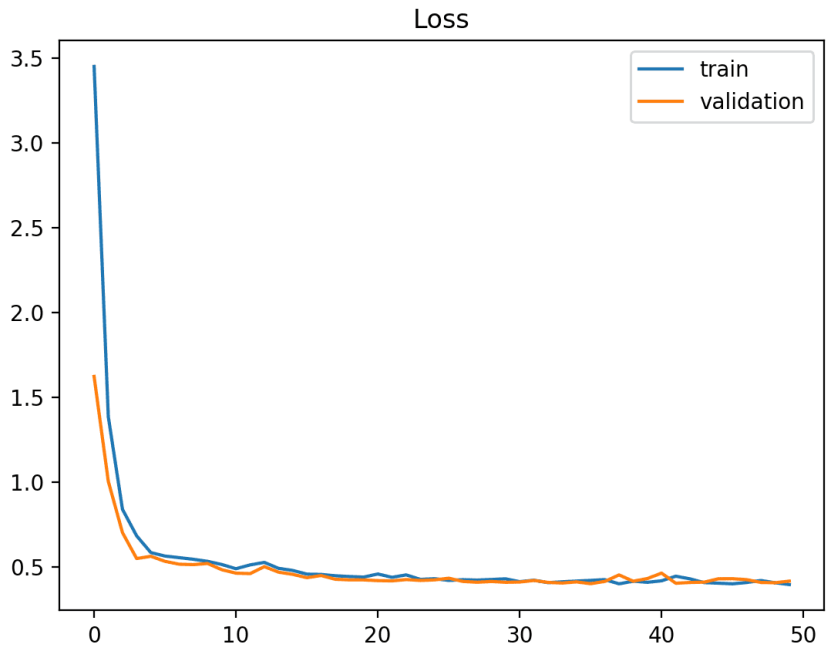
Significa un equilibro dentro de los valores esperados en el modelo teniendo un margen mínimo de diferencia entre el entrenamiento y la validación dentro de la perdida y Accuracy. Sabiendo que casi siempre la perdida será siempre menor en la curva de aprendizaje del entrenamiento. Deberíamos esperar cierta brecha entre el entrenamiento y la curva de test. Este margen es la "brecha de generalización". Y así decimos que un buen entrenamiento contiene las sig. Características(Brownlee, 2019) :

- La gráfica de pérdida de entrenamiento disminuye hasta un punto de estabilidad.
- La gráfica de pérdida de validación disminuye hasta un punto de estabilidad y tiene una pequeña brecha con la pérdida de entrenamiento.
- El entrenamiento continuo de un buen ajuste probablemente conducirá a un sobre ajuste.

La siguiente gráfica de ejemplo demuestra un caso de un buen ajuste.

Figura 37

Curva de Modelo Bien Entrenado



Nota: Extraído de(Brownlee, 2019)

2.1.2.9.1 Evaluación de Resultados Modelo de Deep Learning

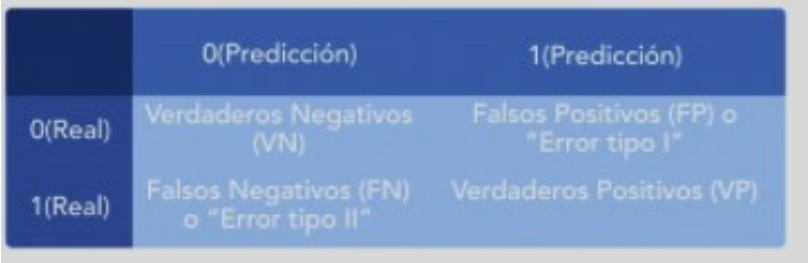
Evaluar de manera cuantitativa los resultados de nuestro modelo nos mostrará concretamente cuanto se ha avanzado y cuanta falta por avanzar en el desarrollo es por eso que conoceremos:

Métricas en Clasificación

En un modelo de clasificación habitual hablar de la matriz de confusión, que simplemente indica, para cada una de las posibles clases, cuántos ejemplos de D_{val} se clasifican en cada una de las posibles opciones: cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa las instancias en la clase real(Sancho Caparrini, 2020).

Figura 38

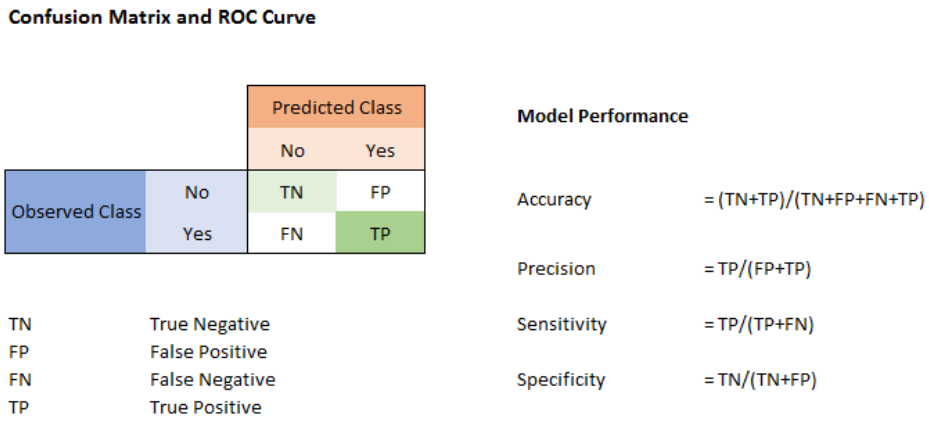
Matriz de Confusión Teórica



Nota: Extraído de (Sancho Caparrini, 2020)

Figura 39

Medidas de Performance del Modelo Deep Learning



Nota: Extraído de (Sancho Caparrini, 2020)

Accuracy

Puede definirse como el porcentaje de predicciones correctas hechas por el modelo de clasificación. Es una buena métrica para usar cuando las clases están balanceadas, es decir la proporción de instancias de todas las clases son similares. Sin embargo, no es una métrica fiable para los conjuntos de datos que tienen un desbalance de clases, es decir el número total de instancias de una clase de datos es muy inferior al número total de instancias de otra clase de datos(Sancho Caparrini, 2020):

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP}$$

Precisión: Indica, de todas las predicciones positivas, cuántas son realmente positivas. Se define como la relación entre las predicciones positivas correctas y las predicciones positivas generales:

$$Precision = \frac{TP}{TP + FP}$$

TPR/Sensitivity/Recall: Indica, de todos los valores realmente positivos, cuántos se predicen como positivos. Es la proporción de predicciones positivas correctas con respecto al número total de casos positivos en el conjunto de datos:

$$TPR = Sensitivity = \frac{TP}{TP + FN}$$

Specificity: Indica, de todos los valores realmente negativos, cuántos se predicen como negativos. Es la proporción de predicciones negativas correctas con respecto al número total de casos negativos en el conjunto de datos:

$$Specificity = \frac{TN}{TN + FP}$$

FPR: Suele ser útil trabajar con la opuesta de la Specificity y se define como:

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

2.2 Antecedentes de la Investigación

2.2.1 Antecedentes a Nivel Nacionales

2.2.1.1 Antecedente 1:

La tesis "*Prototipo de Traductor de Lenguaje de Señas Peruanas Básicas Usando Machine Learning*" (Jimenez & Qquecho, 2021). Publicado en Perú en el año 2021, se describe el desarrollo de un prototipo que permita reconocer una señal básica (cualquier letra estática del abecedario de lenguaje de señas Peruano). Utilizando las técnicas de Machine Learning usando una Convolutional Neural Network (CNN).

Comentario:

El uso de las CNN así mismo el de distintas capas para llegar al resultado esperado hace de esta investigación un paso adicional a solamente el uso de un algoritmo de Deep Learning y su entrenamiento sino no que utiliza una capa desarrollada para darle una aplicación adicional y objetiva para resolver el problema planteado.

2.2.1.2 Antecedente 2:

La tesis "*Evaluación de Técnicas de Aprendizaje de Máquina para la Identificación de Imágenes de Edificios Históricos de la Ciudad del Cusco Basado en Bag-Of-Words y Redes Neuronales Convolucionales* "(Farfan-Escobedo, 2018). Publicado en Perú en el año 2018, describe la comparación de distintas técnicas de visión computacional para detectar edificios en la ciudad del cusco, se hace inca pie en la diversidad de la arquitectura de estos para poder realizarse una detección de iglesias edificios coloniales y monumentos. Se usaron los métodos: Support Vector Machine, Random Forest, Neuronal Network y KNearest Neighbod de los cuales Support Vector Machine junto a una red neuronal convolucional obtuvieron los resultados más favorables en las distintas condiciones de ruido y oclusión de las imágenes.

Comentario:

Este estudio comparativo nacional presenta una comparación concreta de las herramientas de visión computacional con imágenes de un nivel de reconocimiento más elevado, lo cual aporta más a esta investigación, ya que se necesita un tipo de identificación de características más detallado.

2.2.1.3 Antecedente 3:

La tesis "*Reconocimiento de Patrones en Imágenes no Dermatoscopicas para la Detección de Enfermedades Malignas en la Piel, Utilizando Redes Neuronales Convolutivas y Autocodificadores*" (Coronado Pérez, 2018). Publicado en PERÚ en el año, describe el Desarrollar un método de procesamiento de imágenes no dermatoscópicas, basado en redes convolucionales y autocodificadores, para la detección del cáncer de piel. Para lo cual primero se construyó un repositorio de imágenes de casos de enfermedades de la piel categorizadas en benignas pre-malignas y malignas, Desarrollar un método para la extracción de características y así evaluar el método.

Comentario:

Se debe resaltar el uso de método como SVM para la detección de las características así mismo el uso de una RNN pre entrenada (red convolutiva propuesta con arquitectura VGG19) que será medida con el banco de datos a procesarse llegando a un 52.72% de precisión con las métricas de la matriz de confusión.

2.2.2 Antecedentes a Nivel Internacional

2.2.2.1 Antecedente 1:

El artículo "*An Image-Based Approach to Detection of Fake Coins*" (Liu et al., 2017) .Publicado en IEEE en el año 2017 ,se describe el desarrollo de una red neuronal para identificar monedas falsas (Euro , yuan chino) la cual fue entrenada solamente con monedas reales, ya que encontrar las propias falsificaciones es muy complejo además que es más eficiente el entrenamiento basado en la moneda real .Cave recalcar el uso de la normalización de la elipse ya que una moneda puede venir con una fotografía tomada desde un Ángulo inclinado, se hizo uso del detector DOG y el descriptor SIFT para la descripción de los puntos clave y mediante la relación de distancia poder encontrar similitudes hacia la moneda y su comparación.

Comentario:

Siendo este el estudio más detallado sobre el tema que se ha logrado encontrar se tratara de seguir muchos de los estándares aplicados en esta investigación.

2.2.2.2 Antecedente 2:

El artículo "Texture Based Coin Recognition Using Multiple Descriptors" (Chaki & Parekh, 2017). Publicado en Kolkata, India en el año 2017 ,este artículo presenta un nuevo enfoque para el reconocimiento de imagen de monedas. El propósito de este artículo es encontrar una serie de Técnicas que se utilizan para encontrar el conjunto óptimo de características. Adecuado para el reconocimiento de monedas. Tres enfoques diferentes son utilizados, el primero con filtro Gabor, el segundo basado en Hu y el uso de diferentes sistemas de clasificación los cuales son segregados de diferencia mínima (MDC), red neuronal (NN) y Clasificador Neuro Fuzzy (NFC).

Comentario:

La mejor precisión general obtenida es del 99% para el experimento usando filtros Gabor. La precisión reportada en este no da una gran del uso e implementación distintos filtros convolucionales para la detección de características dentro de imágenes de monedas.

2.2.2.3 Antecedente 3:

El artículo "Counterfeit Currency Detection based on AI"(Patil et al., 2022). Publicado en la India en el año 2022 ,este artículo presenta un modelo construido con TensorFlow y Keras el cual fue entrenado con billetes de distinta denominación procedentes de la india, teniendo una precisión de aproximadamente el 90%

Comentario:

La mejor precisión obtenida en esta investigación podría servir como medida para calcular el desempeño de nuestro modelo, ya que el entorno de programación es muy parecido, así como los algoritmos usados.

CAPÍTULO III: METODOLOGÍA

3.1 Tipo de Investigación

Para este proyecto se decidió que el tipo de investigación tecnológica según Carrasco Diaz "Está dirigido a descubrir y conocer que técnicas son más eficaces o apropiadas para operar y perfeccionar las actividades producidas ". El cual va adecuadamente con la proyección de la carrera en relación a las nuevas tecnologías.

3.2 Diseño de la Investigación

3.2.1 Metodología Aplicada

El desarrollo del modelo conllevo a usar y adaptar la metodología que señala Soledad Espezúa.

Figura 40

Metodología Aplicada para Desarrollo de Machine Learning

Nota: Extraído del curso Machine Learning para la industria, clase número 2 , (soledad espezúa, 2021)

Adquisición de Datos

Etapa para abstraer los datos y convertirlos en un formato digital de entrada adecuada para ser usado en nuestro entorno de programación (PYTHON) desde una fuente de almacenamiento en la nube.

Como resultado tendremos un banco de fotografías de monedas falsas y verdaderas en bruto almacenadas(Soledad Espezúa, 2021).

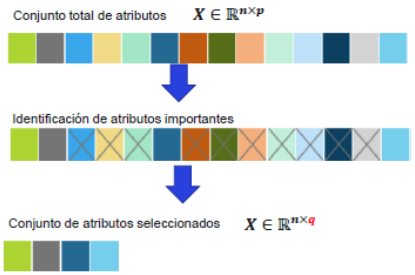
Pre Procesamiento

Se hará un análisis exploratorio de los datos para hacer una preparación de su correcta entrada en el modelo de Machine Learning, para esto se debe elegir que métodos aplicables a nuestra data set mejoraran el rendimiento en nuestro modelo ya sea en facilidad para realizar su trabajo o mejorar la clasificación de características que nuestro modelo tenga que identificar(Soledad Espezúa, 2021).

Filtraremos los datos con mayor ruido o que no salgan de un criterio personal en respuesta a dificultar el trabajo del modelo.

Figura 41

Filtro de Datos



Nota: Extraído del curso Machine Learning para la industria, clase número 3,extraido de *(Soledad Espezúa, 2021)*

Preparación de Datos

Preparamos los datos para que dentro de la red no causen esfuerzo el procesarse y puedan generarse los resultados más eficientes (Vasilev et al., 2019)

Estandarización

Es un proceso que tiene muy buenos resultados dentro de algoritmos de Deep Learning, como resultado obtendremos parámetros estandarizados que puede sr procesados con mayor facilidad haciendo a nuestro modelo más eficiente y veloz en la etapa de aprendizaje.

Ingeniería de Atributos

Se hará un análisis de los componentes y atributos principales para enfocarnos en ellos a la hora de construir el modelo.

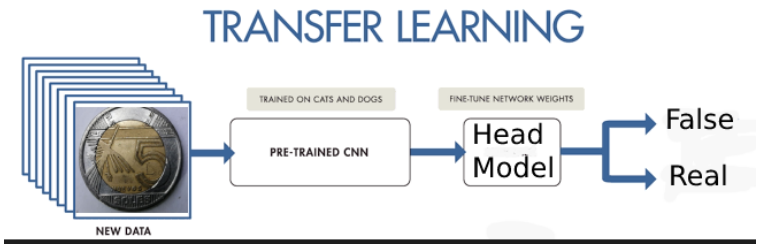
- Datos desbalanceados
- Replicar instancias de la clase minoritaria

Selección algoritmo de Machine Learning

Empezamos definiendo el enfoque que tendrá el modelo según el marco teórico en el capítulo 2.2.2 a su vez como vemos en la figura (figura 42) su adecuación a la arquitectura del Transfer Learning(Vasilev et al., 2019).

Figura 42

Arquitectura del Modelo



Nota: extraído de (Mathworks, 2022)

Procederemos a adentrarnos en los algoritmos de visión artificial y dentro de este la rama más usada dentro de la clasificación de imágenes son las CNN o redes neuronales convolucionales (Shi et al., 2019).Por esto detallamos las características más relevantes que nos serán útiles con nuestra problemática:(Soledad Espezúa, 2021)

- Aprenden las características directamente sin necesidad de extraerlas manualmente.
- Generan resultados de reconocimiento altamente precisos.
- Se pueden volver a entrenar para nuevas tareas de reconocimiento

Capas

El uso de capas convolucionales es ahora junto a las capas max_polling el método más eficiente para la detección de características en imágenes.

Se tiene que priorizar en:

- Menor exigencia en el tamaño del Dataset
- Mayor precisión debido al uso de un modelo pre entrando con datos enormes

Se muestra los parámetros elegidos para el modelo y se pasa a su desarrollo desde el punto de entrada y salida del resultado requerido. Como resultado tendremos un modelo propuesto con los parámetros requeridos para funcionar con los datos de entrada y dar una respuesta basada en los criterios de clasificación el cual entrara en desarrollo.

Entrenamiento de modelo

Se procede con el desarrollo del modelo para implementar los criterios de aprendizaje como objetivo de este paso se seleccionarán los parámetros de aprendizaje estandariza-dos para el tipo de modelo y se pasa a desarrollar progresivamente en el entorno de desarrollo (Colab) siguiendo:

- Pseudocódigo.
- Pre procesamiento
- Creación modelo base
- Creación de modelo cabeza
- Unión de modelos
- Compilar de modelo final
- Entrenamiento

Construido el modelo, setearemos los parámetros de aprendizaje tomando como criterio la performance estándar del modelo pre entrenado y la cantidad de data que se tiene para no saturar el Bach del entorno de programación(Brownlee, 2019).

Como resultados obtendremos la curva de aprendizaje en error y Accuracy.

Evaluación de modelo

Evaluamos la curva para poder hacer o no un ajuste fino en nuestro modelo y finalmente obtener la matriz de confusión como resultado final(Brownlee, 2019).

3.3 Población y Muestra

Como población se toma en cuenta las monedas reales y las monedas falsas de cinco soles del modelo 2010-2015 de las cuales por la diversa cantidad de formas y métodos de falsificación se tomará como muestra las monedas reales para hacer la disimilitud de las falsas.

Figura 43

Moneda Física

Aleación	Diámetro (mm)	Espesor (mm)	Peso (g)	Canto
Bimetálica	24,38	2,10	6,67	Estriado Continuo

Nota: Extraído de (Cono-Monetario-5-00.pdf, s. f.)

Tabla 5

Población

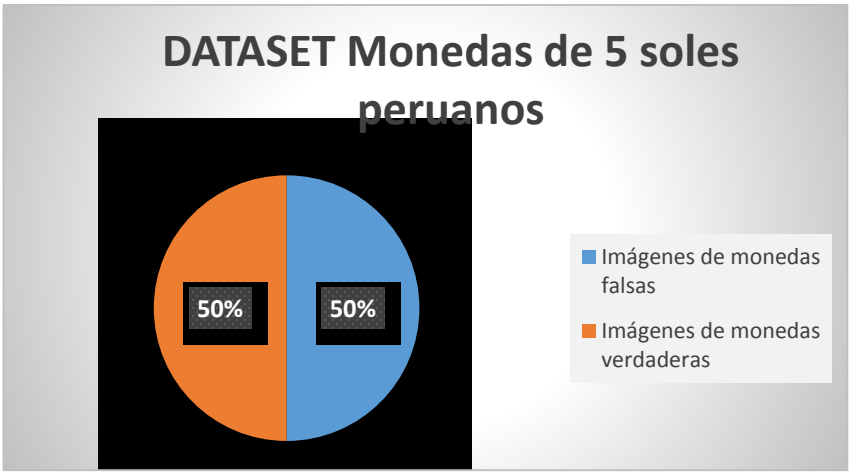
Moneda	Cantidad
5 soles verdaderos	400
5 soles falsos	96

Nota: Presentamos la cantidad de imágenes obtenidas por moneda.

La muestra constituye un total de 496 imágenes de monedas de cinco soles entre verdaderas y falsas teniendo las cuales pasaran por la técnica de Data Augmentation para tener finalmente 1600 imágenes con un 50%-50% por cada categoría expedidas en el modelo 2010-2015 donde.

Figura 44

Dataset Monedas de 5 Soles Peruanos



Nota: figura creada para graficar el balance de datos existentes en data set usada para la etapa de entrenamiento.

Cada imagen tendrá un peso aproximado de 300 Kb y una moneda por imagen así mismo las imágenes contendrán la totalidad de la moneda de externo a extremo según las métricas de (Liu et al., 2017).

Figura 45

Anverso, Reverso Moneda

Nota: Adaptado de (Cono-Monetario-5-00.pdf, s. f.)

Así mismo podemos ver que cada imagen consta de las siguientes características.

Tabla 6

Exploración Inicial de Datos

Exploración inicial de datos		
Dimensión de los datos	Matriz	4224x4224x3 (Ilustración 3-10)
Tipo de dato	Imagen	JPEG
Tamaño de datos		3.60MB
Colores predominantes en la imagen		Plomo ,Ámbar, Blanco
Patrones predominantes	Relieve	Figuras geométricas bien definidas :círculos, esquinas
Ruido	Desgaste	Tiempo de vida
Cantidad de datos	desigual	Poca densidad de imágenes de monedas falsas

Nota: Presentamos la información derivada de la digitalización de las imágenes de las monedas de 5 soles Peruanos.

3.4 Instrumentos

Como instrumento de recolección de datos utilizaremos una cámara fotografía para digitalizar la información de las monedas:

Tabla 7

Especificaciones Instrumento

Modelo	Huawei p30 lite
Resolución	48 Mpx
Apertura	*ƒ*/ 1.8
Pixel size	0.80 µm
Resolución de salida	4000x4000
formato de salida	JPG
iluminación	Lámpara 20 W
posición de iluminación	luz cenital

Nota: Presentamos los detalles de la instrumentaría usada para la digitalización.

Tabla 8

Recursos y Presupuesto

Tarea	Descripción	Costo	
Recursos Humanos			
Horas hombre de trabajo	Sumatoria tiempos actividades (183 días) Costo jornada diaria 20 soles 182* 20 = 3660 soles	S/	3,660.00
		S/	3,660.00
Recursos materiales			
Materiales Oficina	Lapiceros , Cinta para pegar ,	S/	11.00
Silla de Oficina	Silla Oficina	S/	270.00
		S/	311.00
Hardware			
Laptop	Costo compra laptop Lenovo Y580 = s./ 3614.00 Depreciación 25% anual En 183 días = 12.53% de depreciación s./ 458,59 TOTAL	S/	459.00
Teclado	Teclado De Mesa	S/	20.00
		S/	479.00
Software			
Neural Network : Coursera, pucp,	Curso de Introducción a Machine Learning y las Redes Neuronales	S/	1650.00
GOOGLE COLAB	Entorno de desarrollo dentro del lenguaje de programación python. Incluyendo librerías necesarias así como plugins para su correcto uso	S/	70.00
Machine Learning para la industria	Curso de introducción y desarrollo de soluciones de Machine Learning , Universidad Católica del Perú	S/	800.00
		S/	925.00
Servicios			
Internet	Servicio Línea fija de internet	S/	256.00
asesoría	Asesoría De Personas Con Conocimientos Superiores Que Asesoren Y Validen El Correcto Desarrollo	S/	120.00
Servicio fluido eléctrico	costo por kW =0.6660 gasto de laptop kW por hora= 0.3 kW/H horas totales 7 horas * 183 días = 1281 1281 H * 0.3 kw/H = 384.3 kw/H 384.3 kw/H * 0.6660 soles = s./ 256.00	S/	406.00
		S/	782.00
	TOTAL	S/	7,757.00

Nota: Recursos utilizados durante la investigación incluyendo la etapa de desarrollo.

3.5 Recolección y Análisis de Datos

Para esta etapa se empezó visitando pequeños comerciantes y micro medios de transporte obteniendo monedas y segregando las del año en cuestión (2010-2015)

Para luego acudir al banco central de reserva y adquirir paquetes de monedas de común uso y segregando las del año 2010-2015 recolectando un promedio de 415 monedas valorizadas en 2075 nuevos soles. Se obtuvieron a su vez monedas falsificadas de pequeños abarrotes, medios de transporte y avisos en redes sociales de las cuales se obtuvieron 96 monedas que por su estado no tiene ningún valor alguno. Dado que no es posible por las limitaciones de la investigación recolectar monedas de todo el territorio Peruano la población se considerará un subconjunto de la población global de monedas falsas y verdaderas. Adicionalmente para la captura fotográfica para tener una mejor captura de los detalles de acuñación se tomaron en cuenta las especificaciones de (Liu et al., 2017). Teniendo como resultado imágenes individuales de una resolución de 4000x4000 aproximadamente que constituirán nuestro Dataset.

Análisis de Datos

Como instrumentos de medición e interpretación usaremos los criterios detallados en el capítulo 2.2.2.4.5 del marco teórico:

- Accuracy
- Precisión
- Sensibilidad

Estos provenientes de una matriz de confusión sobre los resultados del modelo prediciendo un Dataset de validación del aprendizaje Así mismo se utilizaron los siguientes recursos tecnológicos para la elaboración del proyecto.

Tabla 9

Entorno de Ejecución

Entorno de desarrollo	Google colab
RAM	25 GB
DISCO	100 GB
GPU	12 GB
Lenguaje de programación	Python 3.7.12
Bibliotecas	TensorFlow, Keras , Skit learn, Matplotlib , Numpy , kaggle

Nota: Métricas del entorno donde se desarrollará y ejecutará el modelo con el Dataset generado.

CAPÍTULO IV: ELABORACIÓN DEL MODELO DE DETECCIÓN

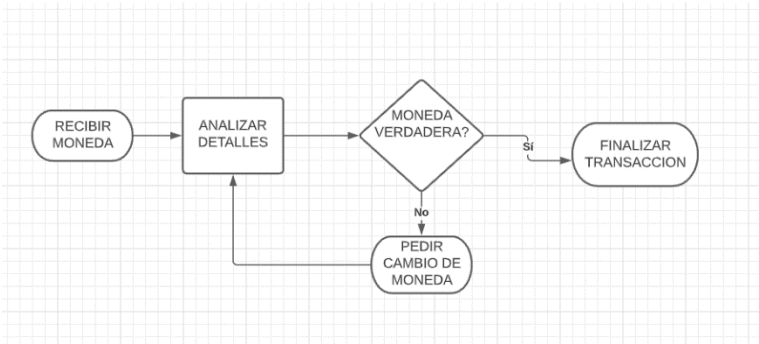
El desarrollo y evaluación del modelo de red neuronal, se hizo conforme a la metodología detallada en el capítulo 3.2.1 de la presente investigación. Esta utilizara la línea de pasos para un proyecto de Machine Learning conforme a las buenas prácticas de planificación, implementación y evaluación.

4.1 Observación

Identificamos un flujo de trabajo de como el usuario recibe la moneda y genera un resultando conforme a su clasificación como se observa en la figura 47.

Figura 46

Diagrama de Flujo Transacción Moneda



Nota: Presentamos el flujo de trabajo para que una moneda se observada y tenga un resultado (verdadero o falso).

Así mismo podemos obtener como resultado de la observación, al entorno de intercambio monetario puntos importantes que nos ayudaran a detallar el contexto como muestra la tabla 11.

Tabla 10

Observación

REFERENCIAS	
Medio de ingreso de información	Observación (ojos)
Criterio de clasificación	Detalles de acuñación (deformaciones)
Respuesta	Verdadero , falso
Tiempo	2 segundos (promedio)
Impedimentos	Facial, apuro , desinterés
Consecuentes	Aceptar transacción / cambio de moneda
Peculiaridades	No solo existe un tipo de acuñación determinada para una moneda (@Perú_coins)

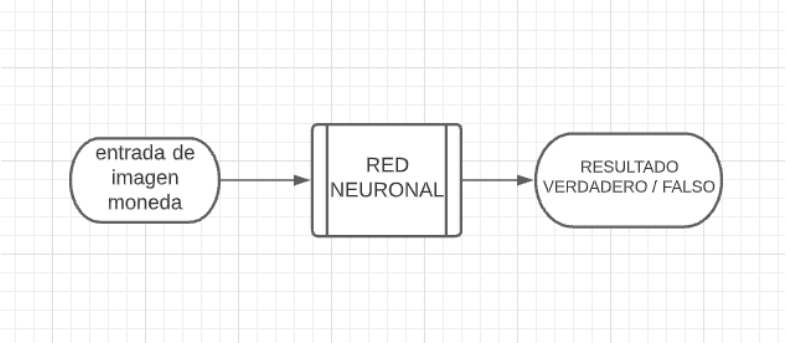
Nota: Presentamos de manera descriptiva el proceso de observación señalando y puntualizando lo encontrado.

4.2 Definir el Proceso

Tener como entrada al Dataset de imágenes de monedas de 5 soles del año 2010-2015 de la *República del Perú* y obtener un resultado de tipo clasificatorio en razón de ser verdadero o falso como muestra la figura 47.

Figura 47

Definición Lógica



Nota: Definición de la lógica de trabajo del modelo de Machine Learning desde el punto de vista del usuario (persona que ejecuta el modelo).

4.3 Adquisición de Datos (Dataset)

Se identificó qué tipo de data tenemos y que características, posee por lo tanto como mostramos en la figura 48 y 49 vemos detalles físicos como circunferencia, peso, etc.

Figura 48

Moneda 5 Soles Descripción

Nota: Vista de monda sin desgaste del banco central de reserva , Extraído de (BCRP, 2016)

Figura 49

Características Moneda de 5 Soles 2010-2015

#KM 344	
Nombre:	5 nuevos soles
Forma:	Redonda
Material:	Bimetálica
Peso:	6,67 g
Diámetro:	24,38 mm
Canto:	
Precios:	1,00€ - 5,00€

Nota: Presentamos características de la moneda física, Extraído de (BCRP, 2016)

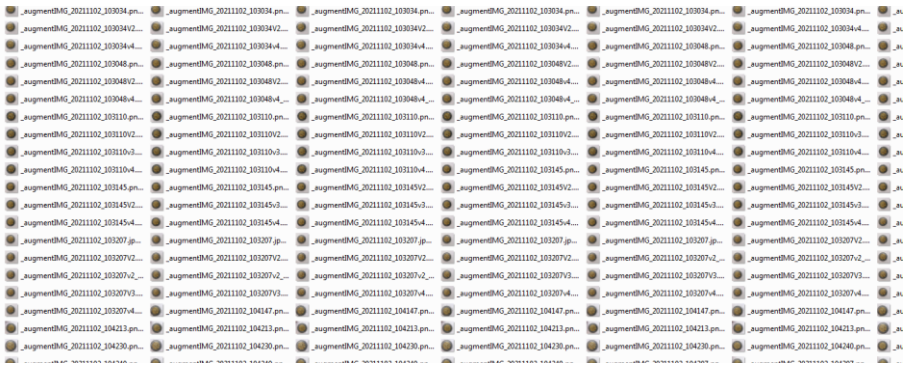
Podemos reconocer los siguientes rasgos en el reverso que fue reconocida según el criterio de identificación de coleccionistas de monedas (@peru_coins).

- Figuras geométricas bien definidas
- Patrones continuos
- Falta de puntos de lectura braille
- La acuñación continua en las 2 clases de aleaciones metálicas

Como parte de la investigación en la figura 50, mostramos una cantidad de la moneda recolectas.

Figura 50

Monedas en Bruto

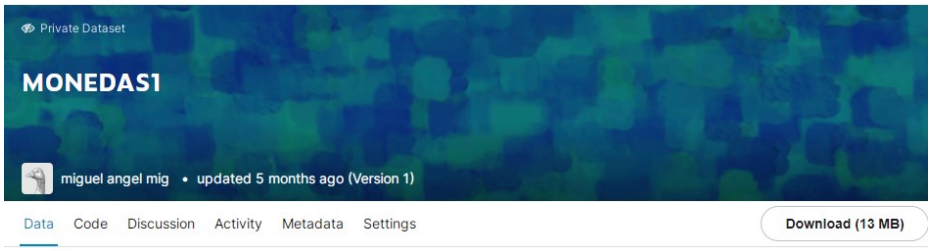


Nota: Cuantificamos las imágenes en 496 por el medio digital de entrada en el que será generado el Dataset.

Elegimos un repositorio para almacenar y poder visualizar, descargar y examinar la data para lo cual elegimos el servicio: *https://www.kaggle.com/* el cual es muy usado dentro de los proyectos de MC por su facilidad de conexión con los entornos más usados como Google Colab y Anaconda (https://www.kaggle.com/datasets/miguelangelmig/coins-500x500).

Figura 51

Ventana Dataset Kaggle



Nota: Presentamos el alojamiento en la nube que tendrá el Dataset “MONEDAS1”.

4.4 Pre Procesamiento

Definiremos nuestra data según (Soledad Espezúa, 2021) como :

Tabla 11

Análisis de Entrada Datos

Tipo de Datos	No Estructurados
Dimensión de datos (shape)	4300x4300
Cantidad de datos	496
Medidas de entrada	496x4300x4300x3

Nota: Detallamos las características que tendrá nuestra data en bruto.

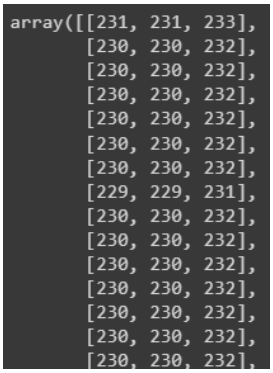
Figura 52

Monedas en Bruto 2

Nota: Corroboramos la limpieza y buena proporción de las imágenes obtenidas.

Figura 53

Exploración Inicial de Datos



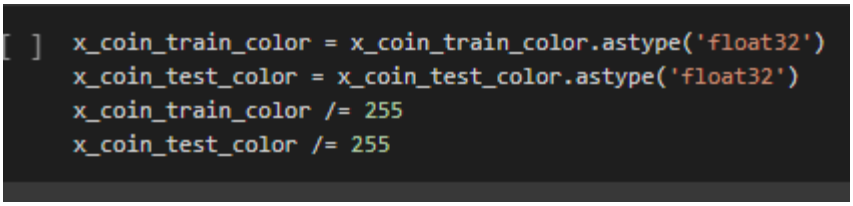
Nota: Observamos en la revisión de que cada pixel posee una matriz de colores RGB.

Aplicamos criterios técnicos de Machine Learning para la data (preprocesamiento):

- Limpieza de datos: Seleccionaremos manualmente los datos, quitando los que puedan tener un ruido que dificulte el aprendizaje, en nuestro caso no se eliminó ninguna imagen.
- Estandarización: proceso donde estandarizaremos los datos (0 -255) entre el número 0 y 1 mediante la división simple de estos valores como se observa en la figura 54.

Figura 54

Estandarización de Datos



Nota: Mostramos proceso de estandarización común para las matrices RGB.

4.5 Ingeniería de Atributos

Determinamos que al encontrarse datos desbalanceados se hará uso de algoritmos de Data Augmentation. Usaremos la biblioteca de keras .

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

La cual nos permite utilizar la función

```
ImageDataGenerator
```

Dentro de la cual usamos parámetros de zoom y ángulo para hacer pequeños cambios en la imagen y generar nueva data. Como mostramos en la figura 55.

Figura 55

Data Augmentation

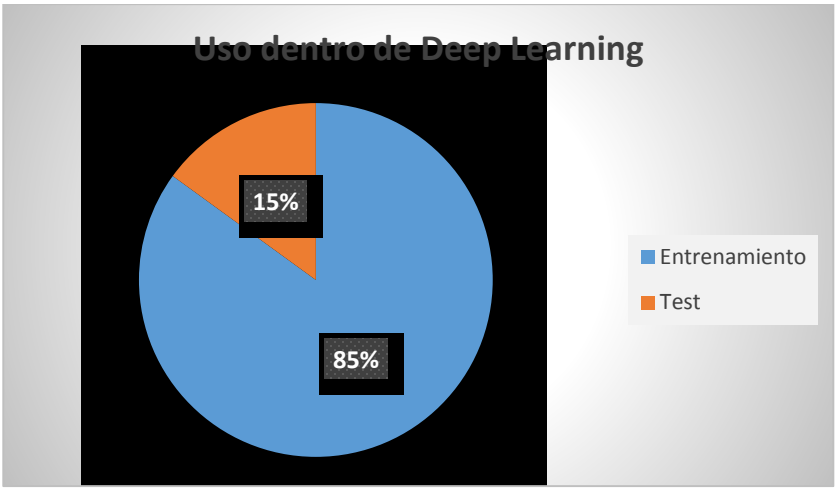
```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
for ga in ficheros :
  img = image.load_img('/content/dataset/train/146 500x500/'+ga)
  img_array = image.img_to_array(img)
  img_array = img_array.reshape((1,) + img_array.shape)

  datagen = ImageDataGenerator(rotation_range=15) #no arguments, no augmentations
  save_to_dir = r'/content/drive/MyDrive/Data_augmentation/ejemplo_2/'

  i = 0
  for batch in datagen.flow(img_array, batch_size=2, save_format='jpg'):
      img_save = image.array_to_img(batch[0], scale=False) #scale=False did the trick
      img_save.save(save_to_dir + fr'\augment'+ga+str(i)+'_{i}.jpg') #save image manually
      if i == 4:
          break
      i += 1
```

Nota: Presentamos el proceso de data Augmentation listando, convirtiendo y almacenando cada imagen nueva.

Obteniendo 1600 imágenes que repartimos para las etapas de entrenamiento y test como muestra la figura 56.

Figura 56

Repartición de Datos en Entrenamiento



Nota: Mostramos la repartición según las delimitaciones recopiladas del Marco Teórico.

Definiremos nuestra data según (Soledad Espezúa, 2021) como:

Tabla 12

Análisis de Entrada datos con Ingeniería de Atributos

Tipo de datos	No Estructurados
Dimensión de datos (shape)	500x500
Cantidad de datos	1600
Medidas de entrada	1600x500x500x3

Nota: Detallamos las características de la data pre procesada y lista para entrar en el modelo.

4.6 Selección del Algoritmo Machine Learning

Definimos el modelo como aprendizaje supervisado en la rama de clasificación, así como el problema de disimilitud.

Figura 57

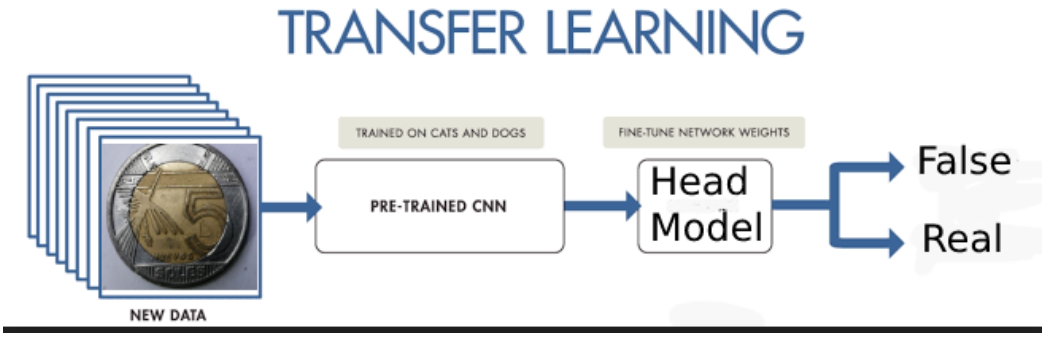
Data Plot

Nota: Mostramos la data descargada del servicio Kaggle, elaboración propia.

Usaremos a sí mismo la técnica de Transfer Learning (Lotfi et al., 2019).

Figura 58

Transfer Learning Conceptual



Nota: Adaptado de (Mathworks, 2022)

Elegiremos el modelo VGG19 (Keras, 2021), tomando las siguientes características relevantes:

- Modelo obtuvo 92.7% de efectividad en ImageNet (Team Keras, 2021).
- Entrenado con GPU's NVIDIA de alta gama por semanas.
- ImageNet es una Dataset de 15 millones de imágenes de alta resolución con 2200 categorías (www.image-net.org)
- Tiene un mejor rendimiento que otros modelos populares como AlexNet y GoogLeNet (Shi et al., 2019).

Figura 59

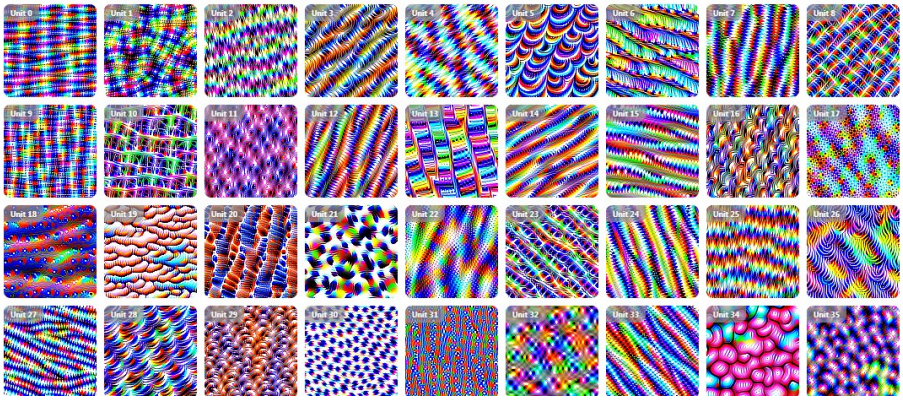
VGG-16 Arquitectura

Nota: Extraído de (Great Learning, 2021)

Analizando una fotografía de los filtros pre entrenados de la web, donde podemos observar diversas formas en filtros, lo cual nos garantiza una variedad en la detección de características que es lo que necesitamos para nuestra problemática por la diversidad de texturas dentro de una moneda de 5 soles Peruanos como muestra la figura 60.

Figura 60

VGG-19 Illustracion de Filtros



Nota: Mostramos los filtros de aprendizaje de la red vgg-16 y de cómo logran captar patrones que ayudaran a identificar objetos extraídos de (OpenAI Microscope, 2021)

Definimos la arquitectura:

Tabla 13

Definición Modelo de Red

Característica	Detalle
Tipo de red	CNN (Red Neuronal Convolucional).
Lenguaje de programación	Python 3.7.12
Entorno de programación	GOOGLE COLAB.
Librería principal de trabajo	TENSORFLOW 2.0 , KERAS
Librerías adicionales	NUMPY, PANDAS, MATPLOTLIB, SKLEARN
Modelo para red de Transferencia	VGG-19
Arquitectura del modelo pre entrenado para nuestra dimensión de imagen (500x500x3)	(ANEXO 1)
Arquitectura red de cabeza con una salida de 2 clases de predicción	(ANEXO 2)
Función de Error	Categorical Crossentropy y softmax

	(Gombru, 2020).
Optimizador	Adam como resultado de una comparación entre distintos optimizadores aplicados en una CNN

Nota: Mostramos los parámetros que usaran el modelo. paso previo a su construcción.

4.7 Entrenamiento de La Red

Pseudocódigo

Se muestra en el Anexo 6 con las siguientes etapas:

- Preprocesamiento
- Creación modelo base
- Creación de modelo cabeza
- Unión de modelos
- Compilar de modelo final
- Entrenamiento
- Ajuste de la red

Despliegue de datos

Cargaremos el archivo de "login.json" para ingresar a nuestro Dataset almacenado en kaggle.

Figura 61

Despliegue de Datos Part-1

Nota: Mostramos la preparación de los datos.

Verificamos el uso de la tarjeta gráfica para acelerar el tiempo de aprendizaje.

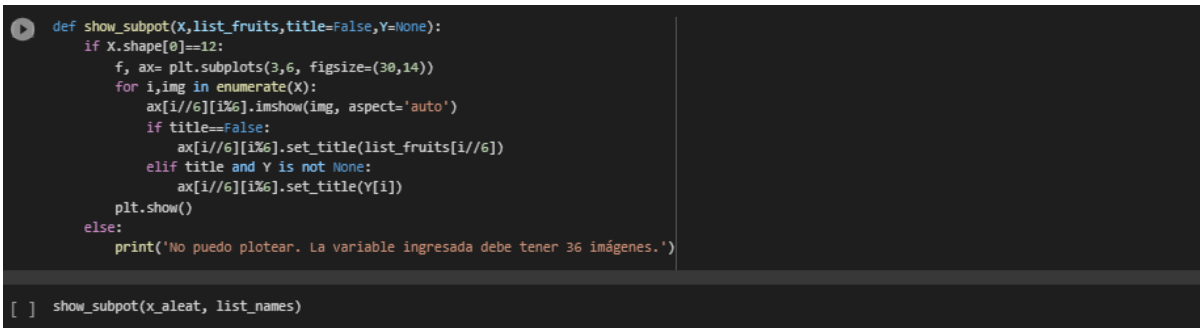
Como resultado Descargamos y desplegamos el Dataset " coins-500x500 (figura 61).

Pre Procesamiento Antes de Aprendizaje

Pasamos a visualizar nuestros datos en el entorno de desarrollo (figura 62).

Figura 62

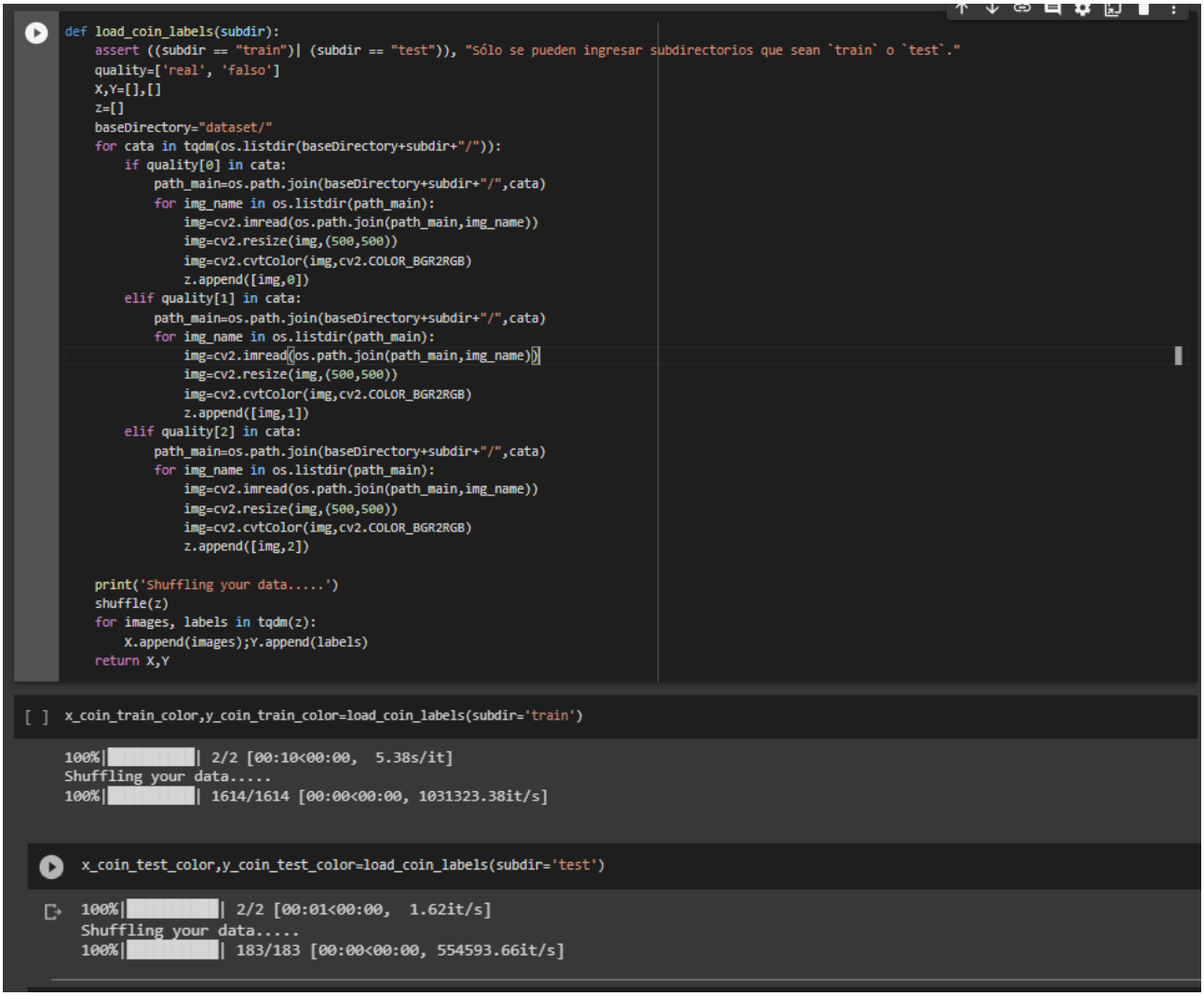
Visualización Cuadricula de Dataset



Nota: Mostramos la visualización de la data, verificando la correcta carga del Dataset.

Figura 63

Shuffle de Dataset y Almacenamiento

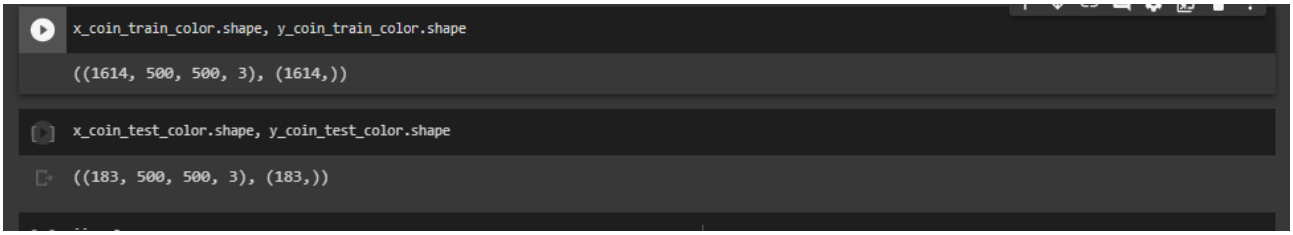


Nota: Mostramos la generación de un suflé en la data para mejorar el proceso de aprendizaje.

Visualizamos el shape de nuestros datos (figura 63) con el cual prepararemos la entrada hacia nuestro modelo base.

Figura 64

Verificación de Dimensiones



Nota: Mostramos las dimensiones del Dataset.

La categorización de la matriz simple donde la posición "0" marcada con 1 significa una moneda verdadera, mientras que la posición "1" marcada con el número 1 una moneda falsa como muestra la figura 71.

Figura 65

Visualización en Categorización de Clases

Nota: Definimos en la matriz la categorización de nuestras etiquetas para ser ingresadas en el modelo.

Visualizamos las monedas ya convertidas en matrices y con su asignación correcta de clase. El Dataset está listo para entrenar a la red (figura 72).

Figura 66

Imágenes Cargadas

Nota: Mostramos la verificación de la categorización en nuestra data.

Creación modelo base

Comenzamos llamando a las bibliotecas de Keras, TensorFlow y SKlearn necesarias para este construir el modelo (figura 73).

Figura 67

Bibliotecas(Vgg19 ,Keras)

```
#bibliotecas necesarias para transfer learning
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint

from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications. imagenet_utils import preprocess_input, decode_predictions

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D , Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator # para generar data augmentation
#from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.applications.vgg16 import preprocess_input
#from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
```

Nota: Se instancian las bibliotecas que se utilizaran para el desarrollo del modelo.

Creamos un modelo vgg19 (figura 69), con una entrada de 500x500x3, sin incluir la cabecera del modelo, ya que incluiremos nuestra propuesta, como vimos en el marco teórico de las soluciones de Transfer Learning como muestra la figura (figura 68).

Figura 68

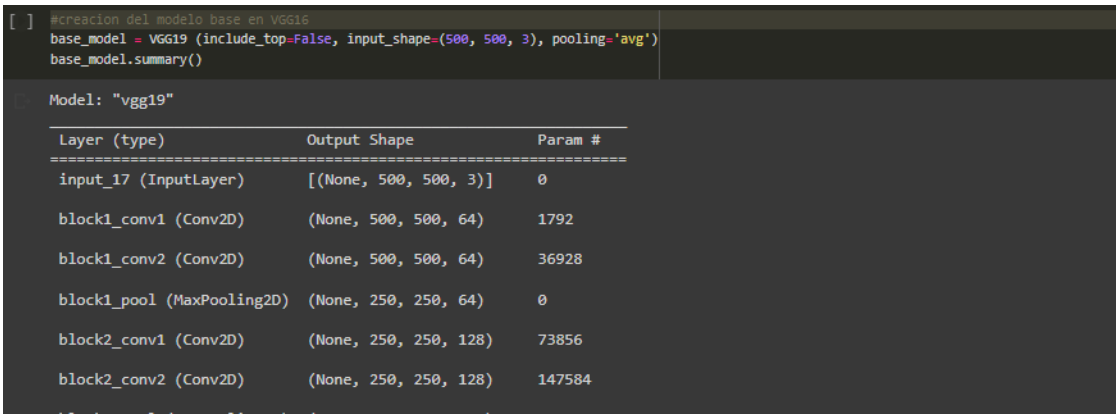
Keras Applications (Vgg19

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
VGG19	549	0.713	0.900	143,667,240	26	84.75	4.38

Nota: Tomamos en cuenta las características del modelo pre entrenado descargarlo en un área con espacio suficiente, extraído de (Keras, 2021).

Figura 69

Creación de Modelo Vgg19



```
[ ] #creacion del modelo base en VGG16
    base_model = VGG19 (include_top=False, input_shape=(500, 500, 3), pooling='avg')
    base_model.summary()

    Model: "vgg19"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     input_17 (InputLayer)       [(None, 500, 500, 3)]     0

     block1_conv1 (Conv2D)       (None, 500, 500, 64)      1792

     block1_conv2 (Conv2D)       (None, 500, 500, 64)      36928

     block1_pool (MaxPooling2D)  (None, 250, 250, 64)      0

     block2_conv1 (Conv2D)       (None, 250, 250, 128)     73856

     block2_conv2 (Conv2D)       (None, 250, 250, 128)     147584
```

Nota: Mostramos la creación del modelo base.

Pasamos a definir la red en modo “no entrenable”, esto con el fin de conservar los pesos o aprendizaje que ya tiene la red. en el modelo *IMAGENET* que es lo que utilizaremos para nuestro modelo de Transfer Learning, elaboración propia.

Así mismo podemos analizar las entradas del modelo (inputs) y sus salidas (outputs) como muestra la figura 70.

Figura 70

Configuración de Modelo Vgg19 , Visualizaciones, Entrada y Salida

```
[ ] #setear el modelo para no entrenarlo desde 0
    base_model.trainable= False

[ ] base_model.input, base_model.output

    (<KerasTensor: shape=(None, 500, 500, 3) dtype=float32 (created by layer 'input_17')>,
     <KerasTensor: shape=(None, 512) dtype=float32 (created by layer 'global_average_pooling2d_14')>)
```

Nota: Configuramos los parámetros de entrada donde irá el Dataset.

Dentro del resumen de nuestro modelo podemos apreciar el número de parámetros con los que cuenta la red VGG19 que son 20,024,384 parámetros de los cuales ninguno se entrenara, ya que conservaremos los pesos de este modelo de la biblioteca de Keras.

Figura 71

Parámetros Modelo Base

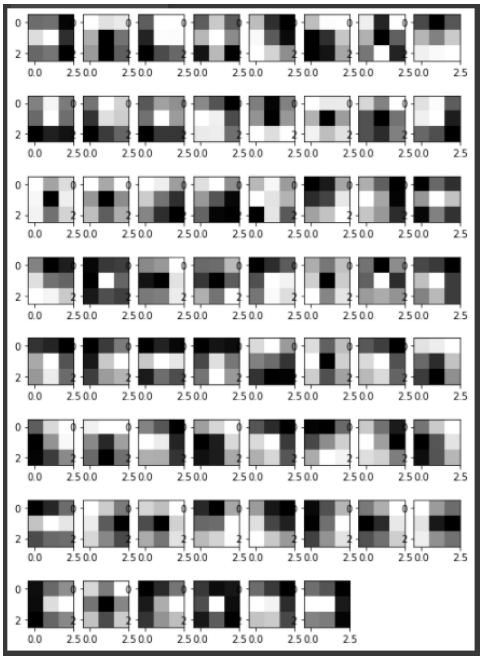
```
=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
```

Nota: Mostramos el número total de parámetros o neuronas de la red.

Las capas convolucionales generarán distintas tomas de nuestra imagen para así identificar y resaltar características de las monedas y así poder crear un mapa de características. Como vemos en la figura 72 se tiene distintas matrices 3x3 a la cuales llamamos kernel que pasaran a operar cada pixel de nuestras imágenes, cave recalcar que en los píxeles cercanos al borde de la imagen se agregara por defecto un pixel de negro para no alterar ningún resultado existente.

Figura 72

Kernel 3x3 de Red vgg19



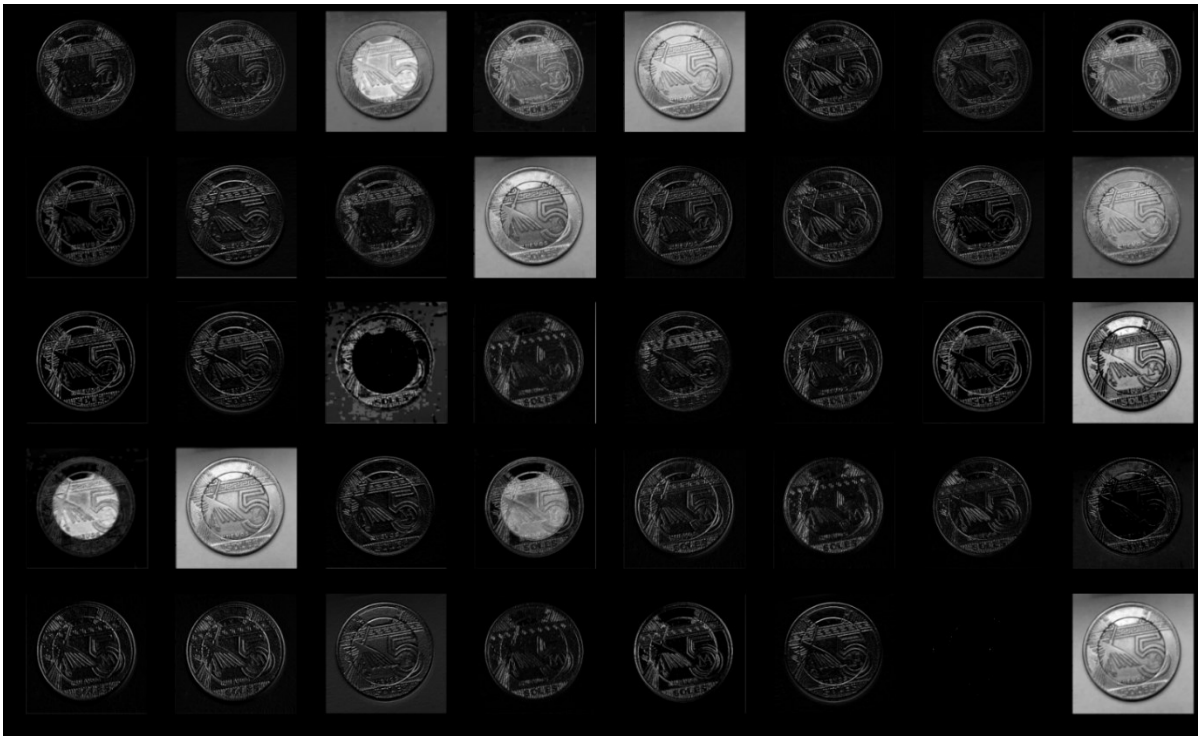
Nota: Presentamos algunos de los filtros CNN de nuestra red.

Como resultado conocemos que nuestros filtros funcionan de manera correcta. Esto a su vez combinado con las capas Maxpooling cuya labor es reducir la dimensione de la imagen podremos tener una ventaja al momento reconocer mejor características básicas (líneas, direcciones).

La figura 73 nos muestra una imagen tratada en las primeras capas y retirada de la red para demostrar el cambio que se obtiene en la imagen, así como los diferentes rasgos que resalta.

Figura 73

Imagen Tratada con Primeros Kernels Vgg19



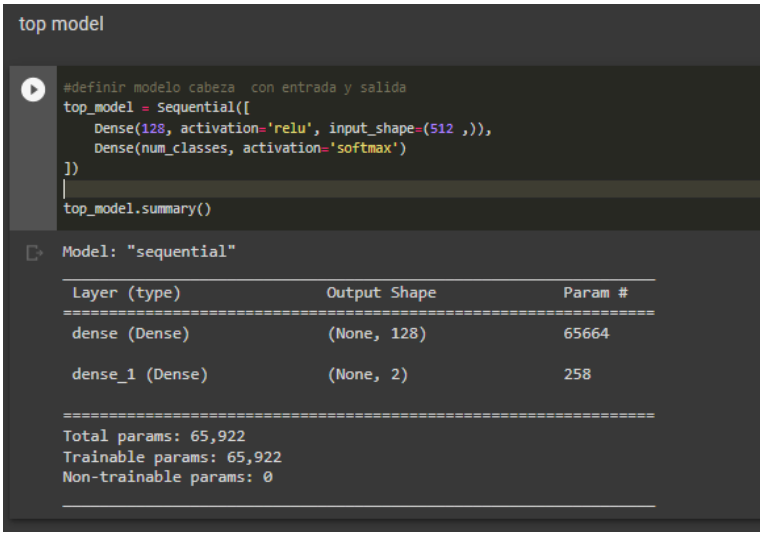
Nota: Mostramos los filtros después de pasar por ciertas imágenes, notamos la diferencia de rasgos resaltantes donde con las constantes iteraciones lograran parametrizar figuras, formas, etc.

Creación de Modelo Cabeza

Creamos el modelo de cabeza, con las especificaciones de salida del modelo vgg19 y el número de clases a clasificar como muestra la (figura 74).

Figura 74

Creación de Modelo Cabeza

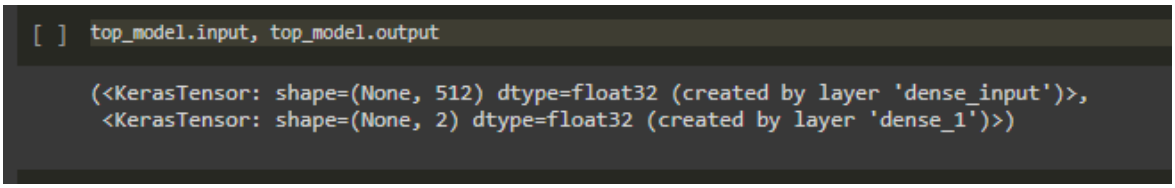


Nota: Mostramos al modelo cabeza el cual tendrá por entrada la salida del modelo base.

Para crear el modelo debemos encajamos el output del modelo VGG con la entrada a nuestro modelo cabeza, el cual es de 512 entradas, como vemos en la (figura 75).

Figura 75

Inputs y Outputs Modelo Cabeza



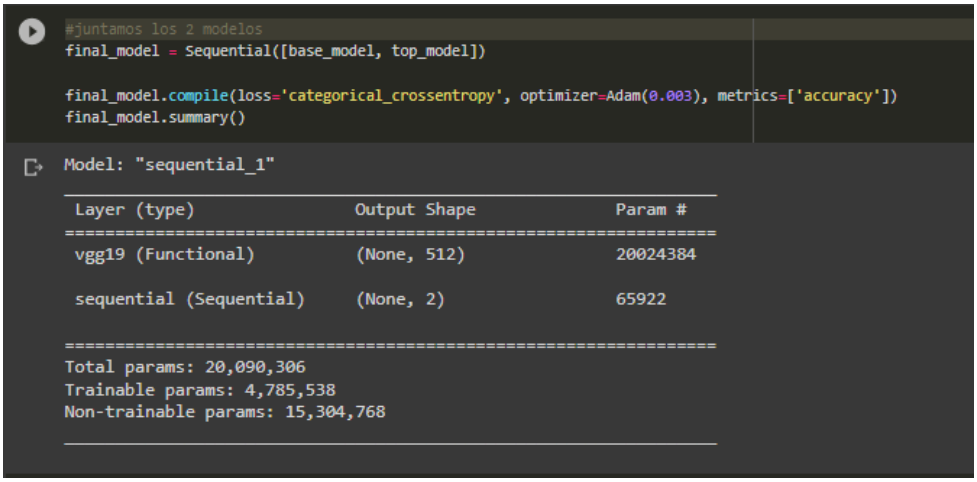
Nota: Mostramos el ingreso salido del modelo cabeza concluyendo en las 2 categorías existentes en el modelo (verdadero y falso).

Unión de Modelos

Compilamos y unimos ambos modelos mediante la función sequential de Keras, al cual llamaremos “final_model”. Junto a la función “categorical_crossentropy”, el optimizador Adam y como medida de evaluación de la red “accuracy” o efectividad.

Figura 76

Final_Model y Copilacion



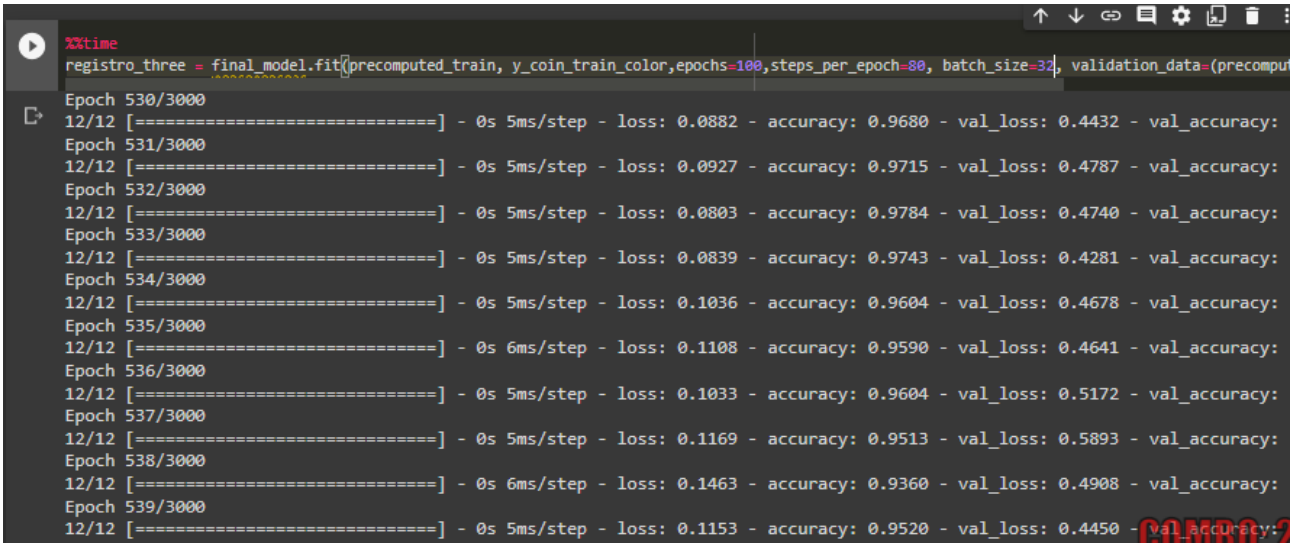
Nota: Mostramos la unión y copulación del modelo bajo los parámetros de optimización, función de error, y optimizador del aprendizaje.

Entrenamiento

Sesteamos los parámetros de entrenamiento con un número de épocas de 100 pasos de 80 y un batch size de 32 para que el modelo no se sature en memoria. (TensorFlow, 2022) este proceso toma un periodo de tiempo según la envergadura del modelo y su Dataset ,podemos ver todo el proceso en la (figura 77).

Figura 77

Entrenamiento Final_Model



Nota: Mostramos la definición de los parámetros de aprendizaje y el inicio de este.

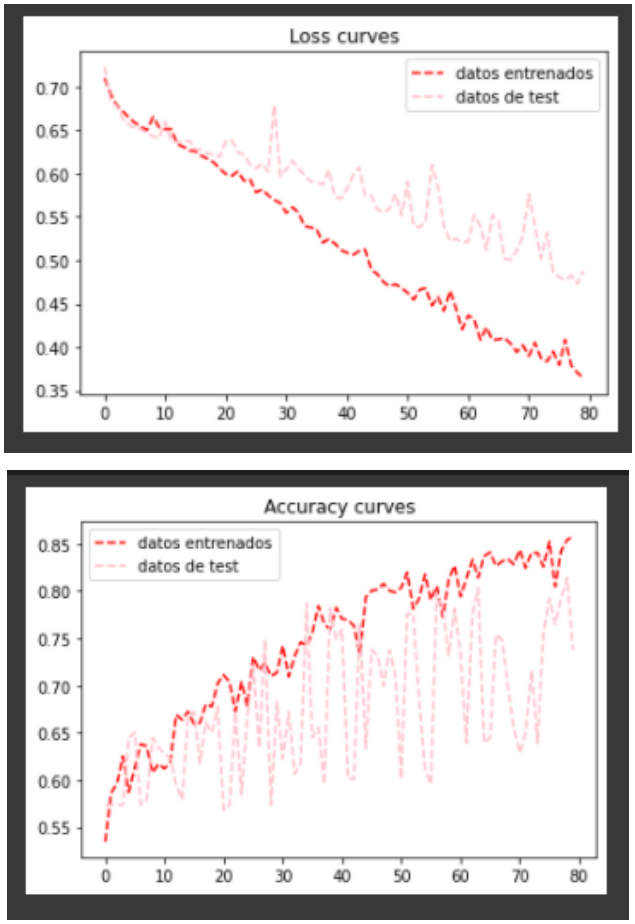
El aprendizaje de la red se hizo haciendo uso del servicio GOOGLE COLAB con una suscripción que provee al usuario de una GPU potente, así como una memoria RAM suficiente para hacer las pruebas necesarias en algoritmos de Deep Learning y CNN.

4.8 Evaluación del Modelo

El Accuracy o efectividad aproximada de 85% el cual es una aceptable para el modelo, paso siguiente es revisar la evolución de nuestra red (figura 78).

Figura 78

Grafica Resultado Entrenamiento Final_Model



Nota: Mostramos las gráficas de la curva de aprendizaje en función de error y la precisión del modelo.

Como podemos ver, el modelo tiene problemas con alinear en los Dataset de train y test. Este problema se conoce como overfiting. Basándonos en (Brownlee, 2019)

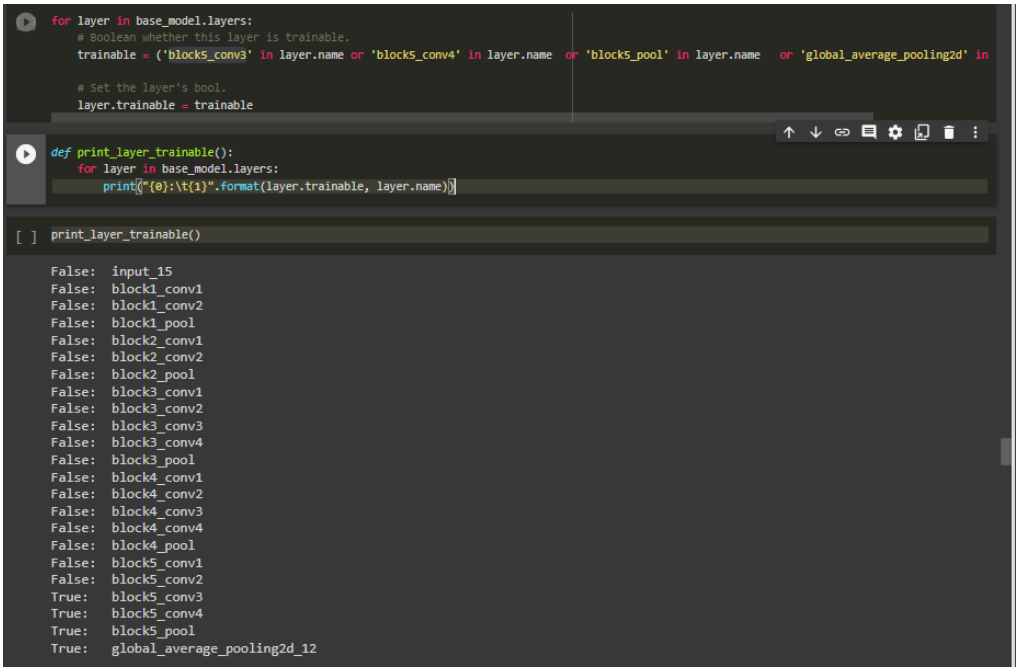
El cual puede ser causado por muchas épocas de entrenamiento, así como capas internas en el modelo pre entrenado.

Ajuste del Modelo

Se determinó que el mejor ajuste para la red que pueda equilibrar el Accuracy y la pérdida del modelo corresponde a entrenar el modelo desde la capa “block5_conv3” esto podrá aumentar la adaptación de nuestra red a los rasgos más distintivos de nuestra red (figura 85).

Figura 79

Gráfica Ajuste de Capas Entrenables Vgg19



Nota: Mostramos la definición del ajuste en torno a los bloques y capas de la red pre entrenada.

Podemos ver que nuestro modelo base ahora tiene 4,719,616 parámetros entrenables los cuales mejorara nuestro resultado y acercaran las perdidas y hará nuestro modelo más consistente (figura 80).

Figura 80

Capas Entrenables Vgg19 con Ajuste

```
=================================================================
Total params: 20,024,384
Trainable params: 4,719,616
Non-trainable params: 15,304,768
```

Nota: Presentamos el número de neuronas entrenadas es más bajo ya que se incluirán en el proceso de aprendizaje.

Bajaremos las épocas de aprendizaje de nuestra red para equilibrar el overfitting , así mismo mostramos el proceso completo .

Figura 81

Entrenamiento Final_Model con Ajuste

```
registro_three=final_model.fit(x_coin_train_color, y_coin_train_color,epochs=30,steps_per_epoch=25, batch_size=32, validation_data=(x_coin_test_
25/25 [==============================] - 22s 920ms/step - loss: 0.6758 - accuracy: 0.5891 - val_loss: 0.6670 - val_accurac
Epoch 3/30
25/25 [==============================] - 14s 559ms/step - loss: 0.6712 - accuracy: 0.5913 - val_loss: 0.6613 - val_accurac
Epoch 4/30
25/25 [==============================] - 14s 556ms/step - loss: 0.6612 - accuracy: 0.6182 - val_loss: 0.6690 - val_accurac
Epoch 5/30
25/25 [==============================] - 14s 561ms/step - loss: 0.6689 - accuracy: 0.5850 - val_loss: 0.6676 - val_accurac
Epoch 6/30
25/25 [==============================] - 14s 556ms/step - loss: 0.6585 - accuracy: 0.6005 - val_loss: 0.6555 - val_accurac
Epoch 7/30
25/25 [==============================] - 14s 562ms/step - loss: 0.6592 - accuracy: 0.5913 - val_loss: 0.6590 - val_accurac
Epoch 8/30
25/25 [==============================] - 14s 558ms/step - loss: 0.6428 - accuracy: 0.6308 - val_loss: 0.6415 - val_accurac
```

Nota: Mostramos el proceso de entrenamiento con los ajustes hechos al modelo.

Como resultado del entrenamiento logramos obtener un Accuracy o efectividad del 84.8 % (figura 82).

Figura 82

Resultado Entrenamiento Final_Model con Ajuste

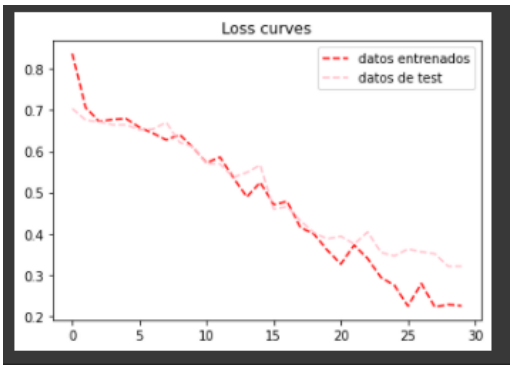
```
Test loss: 0.3209337890148163
Test accuracy: 0.8481481671333313
CPU times: user 1.16 s, sys: 237 ms, total: 1.4 s
Wall time: 3.55 s
```

Nota: Obtenemos la Perdida y Precisión del Modelo.

El modelo tiene una evolución estable comparada con la primera medida que se obtuvo. El problema de overfitting se resolvió y los resultados de predicciones son adecuados. (figura 83).

Figura 83

Evolución Perdidas Final_Model con Ajuste

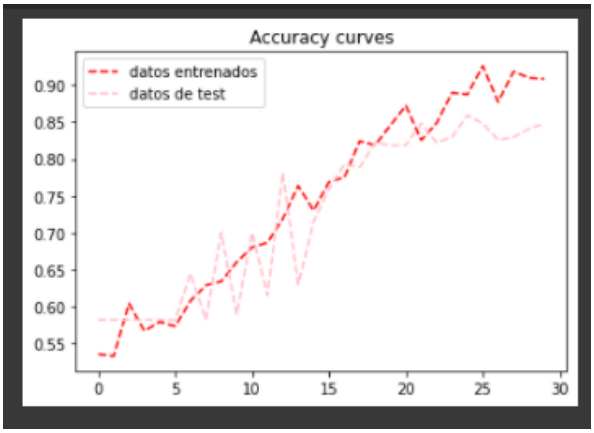


Nota: Mostramos el gráfico de la curva de aprendizaje con respecto a las pérdidas o error en el modelo.

Vemos la gráfica de efectividad o Accuracy así mismo más estable que el primer que se obtuvo (figura 84).

Figura 84

Evolución Perdidas Final_Model con Ajuste



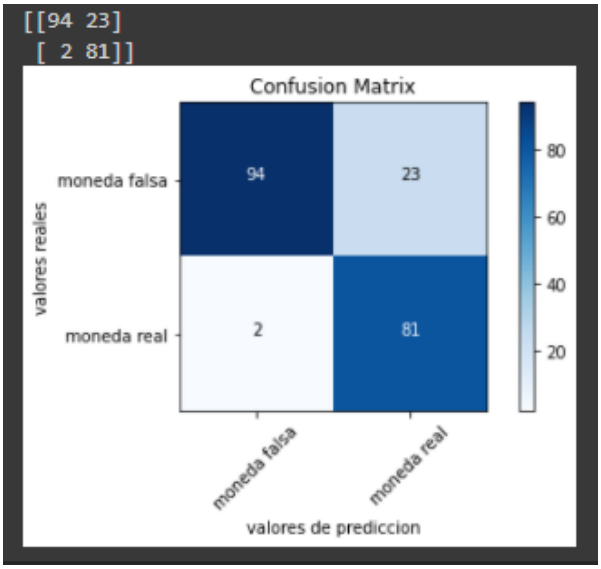
Nota: Mostramos la curva de aprendizaje con respecto a la precisión del modelo.

Matriz de Confusión

Como resultado de esta investigación y teniendo la garantía de que el entrenamiento se ha realizado de manera segura, la matriz de confusión da como resultado una efectividad del 83%. Obtenida de la predicción de 200 imágenes seleccionadas de nuestro Dataset de validación o test, como observamos en la figura 85

Figura 85

Matriz de Confusión



Nota: Presentamos el resultado del modelo en forma de matriz de confusión

Podemos describir:

- Se detectaron 94 que son falsas como falsas.
- Se detectaron 2 monedas reales como falsas.
- Se detectaron 23 monedas falsas como reales.
- Se detectaron 81 monedas reales como reales.

Observamos una concordancia sutil entren el resultado inicial que nos dio la curva de aprendizaje ajustada con la capa 4 del bloque 5 de nuestro modelo base puesta en TRAIN.

Comparamos el resultado obtenido con nuestra hipótesis comprendiendo una diferencia del 7% de nuestra primera hipótesis, lo cual nos traslada a la hipótesis alterna, la cual fue alcanzada por el resultado de precisión de nuestro modelo Final_Model.

CAPÍTULO V: RESULTADOS

5.1 Cumplimiento de Objetivos

Recopilar una base teórica que sustente la aplicación del Machine Learning ***a la detección de monedas falsas y verdaderas.***

Nos encontramos que con que se logró recopilar una base teórica con fundamento analítico y descriptivo para principalmente entender las técnicas de Machine Learning contrastando con lo que pensaba en un inicio sobre la aplicación de fórmulas prediseñadas.

Comprender y tener una base teórica sólida ha sido fundamental a la hora de tomar decisiones con respecto al desarrollo del modelo. Autores como vasilev (Vasilev et al., 2019), Brownlee (Brownlee, 2019) y (Kar & Safari, 2020), fueron relevantes así como también la teoría aplicada(código) como las encontramos en los repositorios de Keras y TensorFlow (Keras, 2021) .

Desarrollar un prototipo de redes neuronales para la detección de monedas falsas y verdaderas de cinco soles Peruanos

Se logró desarrollar el modelo "final_model" que se construyó bajo una base teórica que respalda su desarrollo y funcionamiento, siguiendo una metodología aplicada de desarrollo para proyectos de Machine Learning logrando completar el objetivo.

Interpretar los resultados obtenidos

Como fruto obtuvimos la matriz de confusión la cual paso por las técnicas de interpretación de resultados, dando como resultado el Accuracy (83% de precisión) dándonos así una perspectiva precisa que cumple con el objetivo de interpretar los resultados obtenidos de nuestro modelo y las variables que tienen el mismo impacto del modelo a la capacidad de detección dentro de nuestra investigación.

Tabla 14

Correlación de variables

	Modelo con las técnicas de Machine Learning para la detección por imágenes	Detección de monedas falsas y verdaderas de 5 soles Peruanos
Modelo con las técnicas de Machine Learning para la detección por imágenes	1	0.8333
Detección de monedas falsas y verdaderas de 5 soles Peruanos	0	1

Nota: Mostramos la correlación que existió entre la detección existente y nuestro modelo desarrollado.

5.2 Contribuciones

El uso de las técnicas de Machine Learning son un paso enorme para resolver problemas de disimilitud, clasificación y detección. Esta investigación contribuye a futuros proyectos de distintas índoles dentro del campo del Machine Learning y más específicamente la comunidad habla hispana, este proyecto se vuelve uno de muchos proyectos que suman esfuerzos para implementar esta técnica en más problemas dentro de distintos contextos sociales.

Ya que el prototipo usa bibliotecas open source y puede correr en un sistema operativo libre, reduce costos de implementación y de pago de licencias a futuro; Además de que no requiere ninguna cámara especial de reconocimiento de profundidad como es la Kinect, esto contribuye al uso de software libre y el ahorro económico a los desarrolladores.

Damos una respuesta a nuestro problema general, el cual genera más preguntas que podrán ser materia de nuevas investigaciones con más documentación y recursos técnicos.

CAPÍTULO VI: DISCUSIÓN

Se ha encontrado que durante el desarrollo del fundamento teórico para la toma de decisiones al momento de desarrollar el modelo, el uso de CNN para el procesamiento de imágenes ,resulta ser una técnica suficiente para obtener resultados validados, en ese sentido se concuerda con el trabajo desarrollado por(Jimenez & Qquecho, 2021).

Para el proceso de elección de un modelo se debe conocer bien la problemática y las distintas alternativas que nos plantean las técnicas de Machine Learning. en este sentido Farfan Escobedo (Farfan-Escobedo, 2018),nos muestra alternativas las que consideramos, efectivas aplicadas conociendo bien la problemática a tratarse.

La matriz de confusión es una herramienta indispensable para evaluar los resultados obtenidos del entrenamiento y test, en soluciones de Machine Learning ya que nos brinda objetividad al resultado. así como hizo (Coronado Pérez, 2018) nosotros re afirmamos el uso de la matriz de confusión para obtener resultados de soluciones suficientes.

Durante la formulación del problema encontramos la clara distinción de problemas de clasificación comunes y problemas de disimilitud en las monedas de cinco soles Peruanos. siendo este también la base por donde (Liu et al., 2017), aborda su investigación para definir su propuesta final, nosotros reafirmamos esta postura como base para resolver este tipo de problemáticas.

Las CNN hacen uso de inmensas cantidades de filtros convolucionales diferentes, con el fin de poder encontrar peculiaridades distintas que pueden ser usadas para definir puntos críticos e identificar imágenes. Concluimos que el uso de un filtro principal , como menciona (Chaki & Parekh, 2017)para priorizar la definición de puntos críticos, es así como nuestra propuesta presenta una opción alternativa, la detección de características.

GLOSARIO

TensorFlow: librería de herramientas y procesos usados para Machine Learning desarrollada por la compañía Google.

Keras: biblioteca que simplifica funciones y procesos que mejora el rendimiento y uso de la biblioteca TensorFlow.

Vgg19: Modelo de Deep Learning desarrollado y entrenado por K. Simonyan y A. Zisserman en la Universidad de Oxford en 2014.

Overfiting: nombre atribuido a un estado en una red neuronal donde la red a aprendido demasiado y por lo tanto reduce su confiabilidad para nuevos datos ingresados.

Accuracy: magnitud de medida para analizar el rendimiento de una red en español conocido como efectividad.

Estandarización: procesos para equilibrar los valores dentro del rango 0 a 1 (estandarización más usada).

Dataset: conjunto de datos de tipo texto, tablas, imágenes, o conjuntos de datos que sirven de entrenamiento y validación para una solución de Machine Learning.

Matriz De Confusión: Es una herramienta que se usa normalmente para el campo de la I.A, permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado.

Open Source: En este caso hablamos de un software que puede ser modificado por las personas y en algunos casos compartido y de dominio público. Muchas veces sin costo alguno.

Relu: Una de las muchas funciones de activación usadas en Machine Learning.

Tarjeta Gráfica: Es una tarjeta que usualmente usa un computador para llevar a cabo los procesos de procesamiento de imagen y video.

CONCLUSIONES

El modelo “final_model” se desarrolló usando las técnicas de Machine Learning de manera teórica y práctica que conllevo a cumplir el objetivo principal de esta investigación.

Mediante la revisión bibliográfica de libros, artículos, repositorios de código, manuales y videos prácticos, se recopiló un marco teórico consistente que pudo fundamentar de manera concreta la toma de decisiones paso a paso del modelo elaborado en la investigación respondiendo al objetivo específico número 1.

Se desarrolló el modelo “final_model” con una metodología de desarrollado bajo la cual se logró desarrollar modelo funcional con una entrada de Dataset y salida de clases, cumpliendo con el objetivo específico número 2.

Como resultado del modelo se obtuvo la matriz de confusión que, pasando a ser interpretada en resultados concretos, nos dan una perspectiva completa de nuestra investigación cumpliendo con el objetivo específico número 3.

Se obtuvo una precisión del 84% incumpliendo con la hipótesis planteada inicialmente, pero cumpliendo con la hipótesis alternativa planteada.

RECOMENDACIONES

Aplicar más a profundidad las técnicas de Machine Learning, respondiendo al objetivo principal.

Explorar y recopilar más material teórico, para concebir una mejor toma de decisiones al momento de aplicar las tecnologías de Machine Learning. Respondiendo al objetivo específico 1.

Agrandar el Dataset a un tamaño de al menos, 20000 imágenes por categoría. Existen peculiaridades dentro de las monedas que con una mayor población podrán ser mejor descritas por el modelo. Respondiendo al objetivo específico 2.

Hacer un convenio a acuerdo con los organismos afines y con interés de hacer una mejor interpretación de resultados hasta llegar al aplicativo que pueda clasificar monedas reales y falsas. Respondiendo al objetivo específico 3.

Usar la técnica de Transfer Learning Hibrido incluye algoritmo de clasificación en Machine Learning como SVM y árbol de características.

Se recomienda tomar en cuenta la viabilidad técnica y económica de la investigación.

REFERENCIAS

Atao, L. (2015). *Detección de moneda*.

Basheer, & Hajmeer. (2001). Artificial Neural Networks: Fundamentals, Computing, Design, and Application. *Journal of microbiological methods*, *43*, 3-31. https://doi.org/10.1016/S0167-7012(00)00201-3

BCRP. (2016). *Cono-Monetario-5-00.pdf*. https://www.bcrp.gob.pe/

Brownlee, J. (2019a). How to use Learning Curves to Diagnose Machine Learning Model Performance. *Machine Learning Mastery*. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Brownlee, J. (2019b, febrero 26). How to use Learning Curves to Diagnose Machine Learning Model Performance. *Machine Learning Mastery*. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Carrasco Díaz, S. (2008). *Metodología de la investigación científica: Pautas metodologicas para diseñar y elaborar el proyecto de investigación*. San Marcos.

Chaki, J., & Parekh, R. (2017). Texture Based Coin Recognition Using Multiple Descriptors. *2017 International Conference on Computer, Electrical & Communication Engineering (ICCECE)*, 1-8. https://doi.org/10.1109/ICCECE.2017.8526206

Cono-Monetario-5-00.pdf. (s. f.). Recuperado 11 de mayo de 2021, de https://www.bcrp.gob.pe/docs/Billetes-Monedas/Cono-Monetario/Cono-Monetario-5-00.pdf

Coronado Pérez, R. (2018). *RECONOCIMIENTO DE PATRONES EN IMAGENES NO DERMATOSCOPICAS PARA LA DETECCIÓN DE ENFERMEDADES MALIGNAS EN LA*

PIEL, UTILIZANDO REDES NEURONALES CONVOLUTIVAS Y AUTOCODIFICADORES. UNIVERSIDAD NACIONAL DE SAN AGUSTÍN.

Dot Csv (Director). (2020, noviembre 12). *Redes Neuronales Convolucionales—La VISIÓN de la Inteligencia Artificial □□*. https://www.youtube.com/watch?v=V8j1oENVz00

Economipedia. (2020a). *Moneda*. Economipedia. https://economipedia.com/definiciones/moneda.html

Economipedia. (2020b, marzo 3). *¿Por qué la moneda falsa es dañina para la economía?* Economipedia. https://economipedia.com/actual/por-que-la-moneda-falsa-es-danina-para-la-economia.html

El Comercio. (2020, enero 16). *Ancón: Policía incautó 30 millones de soles falsos a la banda 'La Casa de Papel' | Billetes falsificados | nndc | LIMA*. El Comercio Perú; NOTICIAS EL COMERCIO PERÚ. https://elcomercio.pe/lima/policiales/ancon-policia-incauto-30-millones-de-soles-falsos-a-la-banda-la-casa-de-papel-billetes-falsificados-nndc-noticia/

El Comercio, peru. (2022). *Noticias sobre falsificación de billetes hoy jueves 16 de junio*. El Comercio Perú. https://elcomercio.pe/noticias/falsificacion-de-billetes/

El Peruano. (2018). *Decreto Legislativo-N° 1412*. http://busquedas.elperuano.pe/normaslegales/decreto-legislativo-que-aprueba-la-ley-de-gobierno-digital-decreto-legislativo-n-1412-1691026-1/

El Peruano. (2022). *Estrategia Nacional de Inteligencia Artificial (IA)*. https://www.gob.pe/15762-estrategia-nacional-de-inteligencia-artificial-ia

Farfan-Escobedo, J. D. (2018). *Evaluacio´n de te´cnicas de aprendizaje de maquina para la identificacio´n de ima´genes de edificios histo´ricos de la ciudad del Cusco basado en*

Bag-Of-Words y Redes Neuronales Convolucionales [Tesis]. UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO.

full scan. (2020, mayo 4). *DEEPLEARNING: Como elegir un optimizador?* youtube. https://www.youtube.com/watch?v=BJnBBEDBdKI

Fundación General de la Universidad de La Laguna (Director). (2018). *AGRUPAMIENTO. Algoritmo k-means*. https://www.youtube.com/watch?v=YcJF94-ht74

Gestión, N. (2020, mayo 6). *Policía peruana decomisó billetes falsos equivalentes a US$ 15 millones | PERU*. Gestión; NOTICIAS GESTIÓN. https://gestion.pe/peru/policia-peruana-decomiso-billetes-falsos-equivalentes-a-us-15-millones-noticia/

Gombru (Director). (2020, marzo 23). *Categorical/Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss y todos esos nombres tan confusos*. https://www.youtube.com/watch?v=59mmkjmugBU

Great Learning, G. (2021, septiembre 23). Everything you need to know about VGG16. *Medium*. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

Gtd Perú. (2022). *Gtd Perú: Machine learning enfocado en sectores de minería y finanzas*. Peru-Corporaciones. https://www.gtdperu.com/es/w/novedades/machine-learning-enfocado-en-sectores-de-miner%C3%ADa-y-finanzas

Ipade. (2018, agosto 30). Machine learning y su importancia en la actualidad. *IPADE Business School*. https://www.ipade.mx/2018/08/30/machine-learning-y-su-importancia-en-la-actualidad/

Jimenez, & Qquecho. (2021). *PROTOTIPO DE TRADUCTOR DE LENGUAJE DE SEÑAS PERUANAS BÁSICAS USANDO MACHINE LEARNING*. 106.

Kar, K., & Safari, an O. M. C. (2020). *Mastering Computer Vision with TensorFlow 2.x*. https://learning.oreilly.com/library/view/-/9781838827069/?ar

Keras, K. (2021). *Keras documentation: Keras Applications* [Develement]. Keras Applications. https://keras.io/api/applications/#usage-examples-for-image-classification-models

Ley Orgánica del Banco Central de Reserva del Perú. (1992). Ley Orgánica Del Banco Central De Reserva Del Perú. http://www2.congreso.gob.pe/sicr/cendocbib/con4_uibd.nsf/F0D75D03215B195405257BAB000E840D/$FILE/Ley-Organica-BCRP-26123.pdf

Liu, L., Lu, Y., & Suen, C. (2017a). An Image-Based Approach to Detection of Fake Coins. *IEEE Transactions on Information Forensics and Security*, *PP*, 1-1. https://doi.org/10.1109/TIFS.2017.2656478

Liu, L., Lu, Y., & Suen, C. Y. (2017b). An Image-Based Approach to Detection of Fake Coins. *IEEE Transactions on Information Forensics and Security*, *12*(5), 1227-1239. https://doi.org/10.1109/TIFS.2017.2656478

Lotfi, A., Bouchachia, H., Gegov, A., Langensiepen, C., & McGinnity, M. (Eds.). (2019). *Advances in Computational Intelligence Systems: Contributions Presented at the 18th UK Workshop on Computational Intelligence, September 5-7, 2018, Nottingham, UK* (Vol. 840). Springer International Publishing. https://doi.org/10.1007/978-3-319-97982-3

Mathworks, mathworks. (2022). *Redes neuronales convolucionales*. https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html

OCN. (2019). *Informe estadistico flasificacion en el PERU* (p. 4) [Estadistico]. OCN peru.

OpenAI Microscope. (s. f.). *OpenAI Microscope*. OpenAI Microscope. Recuperado 26 de enero de 2022, de https://microscope.openai.com/models/vgg19_caffe/conv3_4_conv3_4_0?models.op.feature_vis.type=channel&models.op.technique=feature_vis

Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345-1359. https://doi.org/10.1109/TKDE.2009.191

Patil, Prof. D. P., Varma, G., Poojary, S., Sawant, S., & Sharma, A. (2022). Counterfeit Currency Detection based on AI. *International Journal for Research in Applied Science and Engineering Technology*, *10*(4), 3022-3027. https://doi.org/10.22214/ijraset.2022.41980

Raschka, S., & Mirjalili, V. (2019). *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. http://proquest.safaribooksonline.com/?fpi=9781789955750

Republica del Peru. (2010). *Establecidas en el Artículo 2º de la Ley Nº 27583*. Republica del Peru. http://www.ocn.gob.pe/acerca-de-la-ocn/funciones.pdf

Sancho Caparrini, F. (2020). *Medir la eficacia de un aprendizaje—Fernando Sancho Caparrini*. http://www.cs.us.es/~fsancho/?e=231

Shi, Z., Hao, H., Zhao, M., Feng, Y., He, L., Wang, Y., & Suzuki, K. (2019). A deep CNN based transfer learning method for false positive reduction. *Multimedia Tools and Applications*, *78*(1), 1017-1033. https://doi.org/10.1007/s11042-018-6082-6

Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer International Publishing. https://doi.org/10.1007/978-3-319-73004-2

Soledad Espezúa, D. S. E. (2021). Machine Learning para la Industria. *pre-procesamiento*, *2*, 24.

Sólo es Ciencia. (2019). La acuñación de moneda. *Sólo es Ciencia*. https://soloescien-cia.com/2019/10/09/la-acunacion-de-moneda/

Srihari, S. (2020). 15.2 TransferLearning. *Deep Learning*, 34.

Team Keras, K. (2021). *Keras documentation: VGG16 and VGG19* [Keras documentation: VGG16 and VGG19]. https://keras.io/api/applications/vgg/#vgg16-function

TeamKeras, K. (2020). *Keras documentation: Transfer learning & fine-tuning*. https://keras.io/guides/transfer_learning/

TensorFlow. (2022). *Transferencia de aprendizaje y ajuste | TensorFlow Core*. TensorFlow. https://www.tensorflow.org/tutorials/images/transfer_learning?hl=es-419

Unpingco, J. (2019). *Python for Probability, Statistics, and Machine Learning*. Springer International Publishing. https://doi.org/10.1007/978-3-030-18545-9

Vasilev, I., Slater, D., Spacagna, G., Roelants, P., & Zocca, V. (2019). *Python deep learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*.

Viera Balanta (Director). (2015). *Regresión Con Redes Neuronales*. https://www.youtube.com/watch?v=u9Cg0jguzlQ&t=103s

Viles, N. (2015). *Los costos sociales de la falsificación de moneda*. 26.

ANEXOS

ANEXO 1

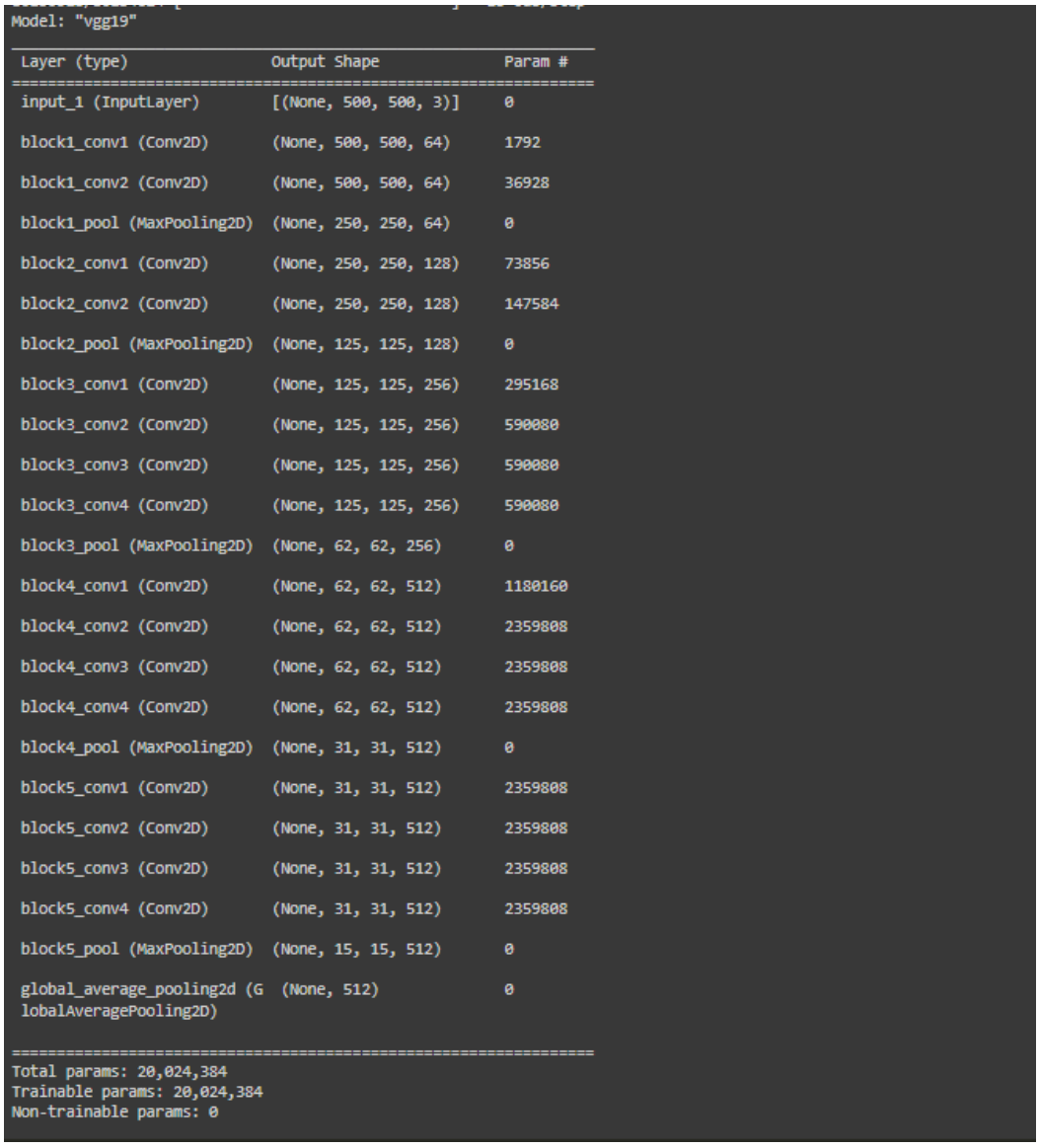
Matriz de Consistencia

Problemas	**Objetivo**	**Hipótesis**	**Metodología**
Principal ¿Cómo aplicar las técnicas de Machine Learning para la detección por imágenes en monedas falsas de cinco soles peruanos? **Específicos** • ¿Cómo elaborar una base teórica que sustente la aplicabilidad del Machine Learning a la detección de monedas falsas? • ¿Cómo desarrollar un prototipo de redes neuronales para la detección de monedas falsas de cinco soles peruanos? • ¿Cómo interpretar los resultados obtenidos?	**General** Desarrollar un modelo con técnicas de Machine Learning para la detección por imágenes en monedas falsas de cinco soles peruanos. **Específicos** • Elaborar una base teórica que sustente la aplicabilidad del Machine Learning a la detección de monedas falsas • Desarrollar un prototipo de redes neuronales para la detección de monedas falsas de cinco soles peruanos • Interpretar los resultados obtenidos	**General** Aplicar las técnicas Machine Learning para la detección por imágenes de monedas falsas de cinco soles peruanos tendrá un error menor al 20%. **Nula** Aplicar las técnicas Machine Learning para la detección por imágenes de monedas falsas de cinco soles peruanos tendrá un error mayor al 20%. **Alternativa** Aplicar las técnicas Machine Learning para la detección por imágenes de monedas falsas de cinco soles peruanos tendrá un error menor al 50%.	**Tipo investigación** • Investigación de tipo básica tecnológica **NIVEL** • Nivel experimental **METODO** • Diseño Pre-experimental

NOTA: mostramos la matriz de consistencia de la investigación.

ANEXO 2

Arquitectura de Modelo Vgg19



```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500, 500, 3)]     0

 block1_conv1 (Conv2D)       (None, 500, 500, 64)      1792

 block1_conv2 (Conv2D)       (None, 500, 500, 64)      36928

 block1_pool (MaxPooling2D)  (None, 250, 250, 64)      0

 block2_conv1 (Conv2D)       (None, 250, 250, 128)     73856

 block2_conv2 (Conv2D)       (None, 250, 250, 128)     147584

 block2_pool (MaxPooling2D)  (None, 125, 125, 128)     0

 block3_conv1 (Conv2D)       (None, 125, 125, 256)     295168

 block3_conv2 (Conv2D)       (None, 125, 125, 256)     590080

 block3_conv3 (Conv2D)       (None, 125, 125, 256)     590080

 block3_conv4 (Conv2D)       (None, 125, 125, 256)     590080

 block3_pool (MaxPooling2D)  (None, 62, 62, 256)       0

 block4_conv1 (Conv2D)       (None, 62, 62, 512)       1180160

 block4_conv2 (Conv2D)       (None, 62, 62, 512)       2359808

 block4_conv3 (Conv2D)       (None, 62, 62, 512)       2359808

 block4_conv4 (Conv2D)       (None, 62, 62, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 31, 31, 512)       0

 block5_conv1 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_conv2 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_conv3 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_conv4 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 15, 15, 512)       0

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
```

Nota: Mostramos la envergadura del modelo base con todas sus capas.

ANEXO 3

Arquitectura Top_Model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               65664

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 65,922
Trainable params: 65,922
Non-trainable params: 0
_________________________________________________________________
```

Nota: mostramos el modelo cabeza

ANEXO 4

Ingreso Kaggle

```
[ ] ! pip install -q kaggle

[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] from google.colab import files

    files.upload()

    Elegir archivos  Ningún archivo seleccionado  Upload widget is only available when the cell has been executed in the current browser session. Pleas
    Saving kaggle.json to kaggle.json
    {'kaggle.json': b'{"username":"miguelangelmig","key":"da4e7b15cf3b9e4eb2ba7db42e8f0e25"}'}

    ! mkdir ~/.kaggle

    ! cp kaggle.json ~/.kaggle/

[ ] ! chmod 600 ~/.kaggle/kaggle.json

[ ] ! kaggle datasets download -d miguelangelmig/coins-500x500

    Downloading coins-500x500.zip to /content
     89% 177M/198M [00:08<00:01, 16.2MB/s]
    100% 198M/198M [00:08<00:00, 25.0MB/s]

[ ] ! ls

    coins-500x500.zip  drive  kaggle.json  sample_data

[ ] !unzip \*.zip  && rm *.zip

    Archive:  coins-500x500.zip
      inflating: dataset/test/falso/_augmentIMG_20211102_103034.png2_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103034V2.png3_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103034v3.jpg1_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103034v3.jpg2_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103034v3.jpg4_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103048.png0_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103048V2.png1_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103048v3.jpg1_i.jpg
      inflating: dataset/test/falso/_augmentIMG_20211102_103048v3.jpg2_i.jpg
```

Nota: mostramos el proceso de descarga del servidor kaggle.

ANEXO 5

Guía Transfer Learning

Cree el modelo base a partir de las convnets preentrenadas

Creará el modelo base a partir del modelo **MobileNet V2** desarrollado en Google. Esto se entrena previamente en el conjunto de datos de ImageNet, un gran conjunto de datos que consta de 1,4 millones de imágenes y 1000 clases. ImageNet es un conjunto de datos de entrenamiento de investigación con una amplia variedad de categorías como `jackfruit` y `syringe` . Esta base de conocimientos nos ayudará a clasificar perros y gatos a partir de nuestro conjunto de datos específico.

Primero, debe elegir qué capa de MobileNet V2 utilizará para la extracción de características. La última capa de clasificación (en "arriba", ya que la mayoría de los diagramas de modelos de aprendizaje automático van de abajo hacia arriba) no es muy útil. En su lugar, seguirá la práctica común de depender de la última capa antes de la operación de aplanado. Esta capa se denomina "capa de cuello de botella". Las características de la capa de cuello de botella conservan más generalidad en comparación con la capa final/superior.

Primero, cree una instancia de un modelo MobileNet V2 precargado con pesos entrenados en ImageNet. Al especificar el argumento **include_top=False** , carga una red que no incluye las capas de clasificación en la parte superior, lo que es ideal para la extracción de características.

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

Este extractor de características convierte cada imagen de `160x160x3` en un bloque de características de `5x5x1280` . Veamos qué le hace a un lote de imágenes de ejemplo:

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 5, 5, 1280)
```

Extracción de características

En este paso, congelará la base convolucional creada en el paso anterior y la utilizará como extractor de características. Además, agrega un clasificador encima y entrena el clasificador de nivel superior.

Congelar la base convolucional

Es importante congelar la base convolucional antes de compilar y entrenar el modelo. Congelar (estableciendo layer.trainable = False) evita que los pesos en una capa determinada se actualicen durante el entrenamiento. MobileNet V2 tiene muchas capas, por lo que establecer la bandera `trainable` de todo el modelo en False las congelará todas.

```
base_model.trainable = False
```

Agregar un encabezado de clasificación

Para generar predicciones a partir del bloque de entidades, promedie las ubicaciones espaciales de `5x5` utilizando una capa `tf.keras.layers.GlobalAveragePooling2D` para convertir las entidades en un único vector de 1280 elementos por imagen.

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

Aplique una capa `tf.keras.layers.Dense` para convertir estas características en una sola predicción por imagen. No necesita una función de activación aquí porque esta predicción se tratará como un `logit` o un valor de predicción sin procesar. Los números positivos predicen la clase 1, los números negativos predicen la clase 0.

```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

Cree un modelo encadenando las capas de aumento de datos, reescalado, `base_model` y extractor de características mediante la API funcional de Keras . Como se mencionó anteriormente, use `training=False` ya que nuestro modelo contiene una capa `BatchNormalization` .

```
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Compilar el modelo

Compile el modelo antes de entrenarlo. Dado que hay dos clases, use la pérdida `tf.keras.losses.BinaryCrossentropy` con `from_logits=True` ya que el modelo proporciona una salida lineal.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
ional)

 global_average_pooling2d (G  (None, 1280)            0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)            0

 dense (Dense)               (None, 1)               1281

=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

Los 2,5 millones de parámetros en MobileNet están congelados, pero hay 1,2 mil parámetros *entrenables* en la capa densa. Estos se dividen entre dos objetos `tf.Variable` , los pesos y los sesgos.

```
len(model.trainable_variables)
```

```
2
```

entrenar al modelo

Después de entrenar durante 10 épocas, debería ver una precisión de ~94 % en el conjunto de validación.

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
26/26 [==============================] - 2s 16ms/step - loss: 0.7428 - accuracy: 0.5186
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.74
initial accuracy: 0.52
```

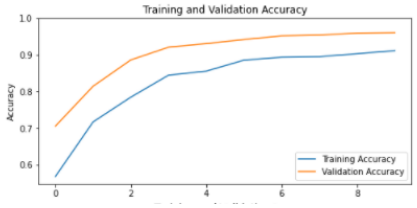
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
63/63 [==============================] - 4s 23ms/step - loss: 0.6804 - accuracy: 0.5680 - val_loss: 0
Epoch 2/10
63/63 [==============================] - 1s 22ms/step - loss: 0.5044 - accuracy: 0.7170 - val_loss: 0
Epoch 3/10
63/63 [==============================] - 1s 21ms/step - loss: 0.4109 - accuracy: 0.7845 - val_loss: 0
Epoch 4/10
63/63 [==============================] - 1s 21ms/step - loss: 0.3285 - accuracy: 0.8445 - val_loss: 0
Epoch 5/10
63/63 [==============================] - 1s 21ms/step - loss: 0.3108 - accuracy: 0.8555 - val_loss: 0
Epoch 6/10
63/63 [==============================] - 1s 21ms/step - loss: 0.2659 - accuracy: 0.8855 - val_loss: 0
Epoch 7/10
```

Curvas de aprendizaje

Echemos un vistazo a las curvas de aprendizaje de la precisión/pérdida de capacitación y validación cuando se usa el modelo base de MobileNetV2 como un extractor de características fijas.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Sintonia FINA

En el experimento de extracción de características, solo estaba entrenando algunas capas sobre un modelo base de MobileNetV2. Los pesos de la red preentrenada **no** se actualizaron durante el entrenamiento.

Una forma de aumentar aún más el rendimiento es entrenar (o "afinar") los pesos de las capas superiores del modelo preentrenado junto con el entrenamiento del clasificador que agregó. El proceso de entrenamiento obligará a ajustar los pesos de los mapas de características genéricas a las características asociadas específicamente con el conjunto de datos.

> ★ **Nota:** Esto solo debe intentarse después de haber entrenado el clasificador de nivel superior con el modelo preentrenado establecido en no entrenable. Si agrega un clasificador inicializado aleatoriamente encima de un modelo previamente entrenado e intenta entrenar todas las capas juntas, la magnitud de las actualizaciones de gradiente será demasiado grande (debido a los pesos aleatorios del clasificador) y su modelo previamente entrenado olvidar lo que ha aprendido.

Además, debe intentar ajustar una pequeña cantidad de capas superiores en lugar de todo el modelo de MobileNet. En la mayoría de las redes convolucionales, cuanto más arriba está una capa, más especializada es. Las primeras capas aprenden características muy simples y genéricas que se generalizan a casi todos los tipos de imágenes. A medida que avanza, las funciones son cada vez más específicas para el conjunto de datos en el que se entrenó el modelo. El objetivo del ajuste fino es adaptar estas características especializadas para que funcionen con el nuevo conjunto de datos, en lugar de sobrescribir el aprendizaje genérico.

Descongele las capas superiores del modelo.

Todo lo que necesita hacer es descongelar el `base_model` y configurar las capas inferiores para que no se puedan entrenar. Luego, debe volver a compilar el modelo (necesario para que estos cambios surtan efecto) y reanudar el entrenamiento.

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  154
```

Compilar el modelo

Como está entrenando un modelo mucho más grande y quiere readaptar los pesos previamente entrenados, es importante usar una tasa de aprendizaje más baja en esta etapa. De lo contrario, su modelo podría sobreajustarse muy rápidamente.

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

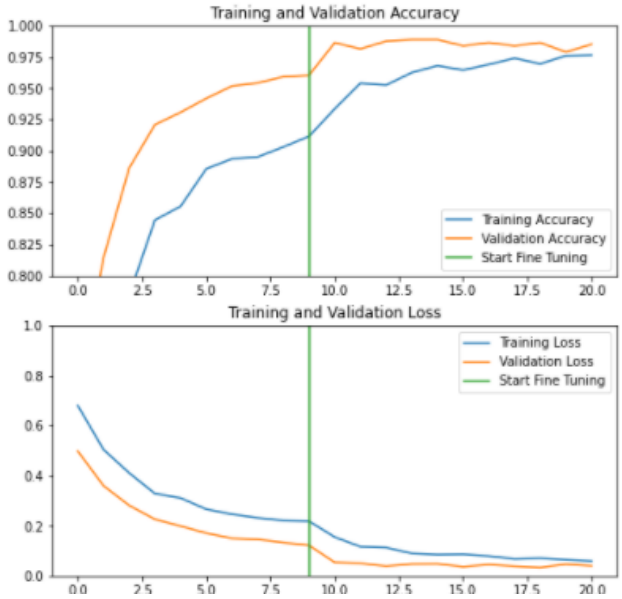
```
model.summary()
```

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
63/63 [==============================] - 7s 40ms/step - loss: 0.1545 - accuracy: 0.9335 - val_loss: 0
Epoch 11/20
63/63 [==============================] - 2s 28ms/step - loss: 0.1161 - accuracy: 0.9540 - val_loss: 0
Epoch 12/20
63/63 [==============================] - 2s 28ms/step - loss: 0.1125 - accuracy: 0.9525 - val_loss: 0
Epoch 13/20
63/63 [==============================] - 2s 28ms/step - loss: 0.0891 - accuracy: 0.9625 - val_loss: 0
Epoch 14/20
63/63 [==============================] - 2s 28ms/step - loss: 0.0844 - accuracy: 0.9680 - val_loss: 0
Epoch 15/20
63/63 [==============================] - 2s 28ms/step - loss: 0.0857 - accuracy: 0.9645 - val_loss: 0
Epoch 16/20
```

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Nota: Extraído de(Team Keras, 2021)

ANEXO 6

Resultado de Entrenamiento

```
registro_three=final_model.fit(x_coin_train_color, y_coin_train_color,epochs=30,steps_per_epoch=25, batch_size=32, validation_data=(x_coin_test_color, y_coin_test_color))

Epoch 2/30
25/25 [==============================] - 21s 859ms/step - loss: 0.6901 - accuracy: 0.5626 - val_loss: 0.6909 - val_accuracy: 0.5815
Epoch 3/30
25/25 [==============================] - 12s 488ms/step - loss: 0.6782 - accuracy: 0.5913 - val_loss: 0.6724 - val_accuracy: 0.6259
Epoch 4/30
25/25 [==============================] - 12s 488ms/step - loss: 0.6715 - accuracy: 0.5626 - val_loss: 0.6486 - val_accuracy: 0.6148
Epoch 5/30
25/25 [==============================] - 12s 490ms/step - loss: 0.6738 - accuracy: 0.5875 - val_loss: 0.6622 - val_accuracy: 0.5741
Epoch 6/30
25/25 [==============================] - 12s 488ms/step - loss: 0.6600 - accuracy: 0.6005 - val_loss: 0.6812 - val_accuracy: 0.5704
Epoch 7/30
25/25 [==============================] - 12s 490ms/step - loss: 0.6578 - accuracy: 0.5763 - val_loss: 0.6449 - val_accuracy: 0.5926
Epoch 8/30
25/25 [==============================] - 12s 488ms/step - loss: 0.6522 - accuracy: 0.6182 - val_loss: 0.6315 - val_accuracy: 0.6296
Epoch 9/30
25/25 [==============================] - 12s 490ms/step - loss: 0.6123 - accuracy: 0.6375 - val_loss: 0.5998 - val_accuracy: 0.6296
Epoch 10/30
25/25 [==============================] - 12s 486ms/step - loss: 0.5768 - accuracy: 0.6928 - val_loss: 0.5672 - val_accuracy: 0.6407
Epoch 11/30
25/25 [==============================] - 12s 490ms/step - loss: 0.5420 - accuracy: 0.7000 - val_loss: 0.5799 - val_accuracy: 0.6926
Epoch 12/30
25/25 [==============================] - 12s 487ms/step - loss: 0.4939 - accuracy: 0.7573 - val_loss: 0.4893 - val_accuracy: 0.7407
Epoch 13/30
25/25 [==============================] - 12s 493ms/step - loss: 0.5410 - accuracy: 0.7150 - val_loss: 0.5200 - val_accuracy: 0.7556
Epoch 14/30
25/25 [==============================] - 12s 487ms/step - loss: 0.4285 - accuracy: 0.7927 - val_loss: 0.4373 - val_accuracy: 0.7889
Epoch 15/30
25/25 [==============================] - 12s 491ms/step - loss: 0.3550 - accuracy: 0.8338 - val_loss: 0.4499 - val_accuracy: 0.7333
Epoch 16/30
25/25 [==============================] - 12s 489ms/step - loss: 0.3090 - accuracy: 0.8748 - val_loss: 0.3929 - val_accuracy: 0.8185
Epoch 17/30
25/25 [==============================] - 12s 492ms/step - loss: 0.3797 - accuracy: 0.8300 - val_loss: 0.4303 - val_accuracy: 0.7222
Epoch 18/30
25/25 [==============================] - 12s 487ms/step - loss: 0.2856 - accuracy: 0.8824 - val_loss: 0.4805 - val_accuracy: 0.8000
Epoch 19/30
25/25 [==============================] - 12s 491ms/step - loss: 0.3356 - accuracy: 0.8525 - val_loss: 0.5558 - val_accuracy: 0.7630
Epoch 20/30
25/25 [==============================] - 12s 489ms/step - loss: 0.2328 - accuracy: 0.9140 - val_loss: 0.3338 - val_accuracy: 0.8593
Epoch 21/30
25/25 [==============================] - 12s 490ms/step - loss: 0.2224 - accuracy: 0.9187 - val_loss: 0.3286 - val_accuracy: 0.8222
Epoch 22/30
25/25 [==============================] - 12s 488ms/step - loss: 0.2494 - accuracy: 0.8913 - val_loss: 0.4508 - val_accuracy: 0.7481
Epoch 23/30
25/25 [==============================] - 12s 489ms/step - loss: 0.2655 - accuracy: 0.8875 - val_loss: 0.2990 - val_accuracy: 0.8593
Epoch 24/30
25/25 [==============================] - 12s 491ms/step - loss: 0.1909 - accuracy: 0.9305 - val_loss: 0.3384 - val_accuracy: 0.8704
Epoch 25/30
25/25 [==============================] - 12s 489ms/step - loss: 0.1504 - accuracy: 0.9494 - val_loss: 0.3343 - val_accuracy: 0.8222
Epoch 26/30
25/25 [==============================] - 12s 492ms/step - loss: 0.1380 - accuracy: 0.9425 - val_loss: 0.2963 - val_accuracy: 0.8741
Epoch 27/30
25/25 [==============================] - 12s 486ms/step - loss: 0.1738 - accuracy: 0.9241 - val_loss: 0.3057 - val_accuracy: 0.8778
Epoch 28/30
25/25 [==============================] - 12s 490ms/step - loss: 0.1237 - accuracy: 0.9538 - val_loss: 0.2674 - val_accuracy: 0.8852
Epoch 29/30
25/25 [==============================] - 12s 489ms/step - loss: 0.2074 - accuracy: 0.9216 - val_loss: 0.3343 - val_accuracy: 0.8667
Epoch 30/30
25/25 [==============================] - 12s 488ms/step - loss: 0.1405 - accuracy: 0.9463 - val_loss: 0.3207 - val_accuracy: 0.8778
```

Nota: Mostramos el entrenamiento completo del modelo.

ANEXO 7

Pseudocodigo de Modelo

```
#preprarar los datos para almacenamiento desde kaggle
Instalar instacias Kaggle
subir y leeer archivo de login Kaggle
descargar DataSet "miguelangelmig/coins-500x500"

desplegar zip "coins-500x500.zip"
ingresar carpeta "dataset"

#importar bibliotecas necesarias
importar biblioteca  numpy como np
importar biblioteca  pandas como np
importar biblioteca matplotlib.pyplot como plt
importar bilbioteca tensorflow.keras.utils
importar biblioteca matplotlib.pyplot como plt

#etiquetar y mezclar datos --preprocesamiento
funcion definir etiquetas
(
  permitir subdirectorios[["train", "test "]
  cata_rreglo["real","falso"]
  directorio="dataset/"
  para cata_rreglo en directorio+lista_subdirectorios:
  {
    si "real" en cata_rreglo:
   {
      para iamgen en lista_subdirectorios
      imagen=leer imagen de lista_subdirectorios
      imagen=redimencionar(imagen(500,500))
      agregar imagen en arreglo Z
   }
    entonces si "falso" en cata_rreglo:
   {
      para iamgen en lista_subdirectorios
      imagen=leer imagen de lista_subdirectorios
      imagen=redimencionar(imagen(500,500))
      agregar imagen en arreglo Z
   }
  }
  imprimir"mezclando etiquetas"
  mezclar(Z)
  para imagenes, etiquetas en Z
    x.agregar (imagen)
     y.agregar(etiqueta)
  retornar X,Y
)
```

```
x_entrenamiento_monedas,y_entrenamiento_monedas =definir(subdirectorio='train')
x_validacion_monedas,y_validacion_monedas =definir(subdirectorio='test')

x_entrenamiento_monedas=np.array(x_entrenamiento_monedas)
y_entrenamiento_monedas=np.array(y_entrenamiento_monedas)

x_validacion_monedas=np.array(x_validacion_monedas)
y_validacion_monedas=np.array(y_validacion_monedas)


#verificamos la forma de nuestros datos
x_entrenamiento_monedas.shape, y_entrenamiento_monedas.shape
((1614, 500, 500, 3), (1614,))
x_validacion_monedas.shape, y_validacion_monedas.shape
((183, 500, 500, 3), (183,))



#agregamos las siguientes bibliotecas que usaremos

de tensorflow.keras.optimizers importar Adam
de tensorflow.keras.callbacks importar TensorBoard, ModelCheckpoint
de tensorflow.keras.preprocessing importar image
de tensorflow.keras.applications. importar preprocess_input, decode_predictions
de sklearn.model_selection importar train_test_split
de tensorflow.keras.models importar Sequential
de tensorflow.keras.layers importar Dense, Dropout, Activation, Flatten ,Conv2D, MaxPooling2D , Input
de tensorflow.keras.preprocessing.image importar ImageDataGenerator # para generar data augmentation
de tensorflow.keras.applications.vgg16 importar VGG16 ,preprocess_input
de tensorflow.keras.utils importar to_categorical

#estandarizar imagenes  enter valores 0 y 1
x_entrenamiento_monedas = x_entrenamiento_monedas.a_tipo('float32')
x_validacion_monedas = x_validacion_monedas.a_tipo('float32')
x_entrenamiento_monedas /= 255
x_validacion_monedas /= 255


#cambiar valores de etiquetas en categorias
y_entrenamiento_monedas = to_categorical(y_entrenamiento_monedas)
y_validacion_monedas = to_categorical(y_validacion_monedas)

#creamos el modelo base VGG!&
base_model = VGG16 (incluir_cabezera=False, forma_de_entrada=(500,500,3), pooling='avg')
base_model.summary()

#seteamos su training en false
base_model.trainable= False
```

```
#precomputamos al data en la red vgg16
precoputado_train = base_model.predecir(x_entrenamiento_monedas, batch=128,)
precoputado_train.forma
precomputado_test = base_model.predecir(x_validacion_monedas, batch=128, )
precomputado_test.forma

#definimos el nuemrod e clases
num_classes = 2

#creamos el mdoelo de cabeza

top_model = modelo_secuencial([
    capa_densa(128, funcion_activacion='relu', forma_entrada=(1280,)),
    capa_densa(num_classes, funcion_activacion='softmax')
])

#copilamos el mod elos
top_model.compilar(funcion_perdida='categorical_crossentropy',
optimizador=Adam(0.001), metricade evaluacion=['accuracy'])
top_model.resumen()

#entrenamos el modelo con los datos del modelo base
%%time
registro = top_model.entrenar(precoputado_train, y_entrenamiento_monedas, epocas=50, batch=64
, data_validacion=(precoputado_test, y_validacion_monedas))
```

Nota: Mostramos el seudocódigo del desarrollo del modelo.

ANEXO 8

Código Usado en el Modelo

```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download -d miguelangelmig/coins-500x500
```

Ingresamos a kaggle para descargar nuestro Dataset “coins-500x500”

```
!unzip \*.zip  && rm *.zip
```

Descomprimimos nuestra data descarga en formato ZIP

Importamos las bibliotecas que usaremos dentro de nuestro desarrollo

```
import numpy as np
import pandas as pd
import os
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm
from random import shuffle
from tensorflow.keras.utils  import to_categorical
import pickle
import matplotlib.pyplot as plt
import matplotlib as mpl
```

```
tf.test.gpu_device_name()
```

constatamos que nuestro entorno de programación colab tenga encendida la tarjeta de video

```
!/opt/bin/nvidia-smi
```

Verificamos la cantidad de memoria de video que tenga nuestra tarjeta de video

```
def show_subpot(X,list_fruits,title=False,Y=None):
    if X.shape[0]==12:
        f, ax= plt.subplots(3,6, figsize=(30,14))
        for i,img in enumerate(X):
            ax[i//6][i%6].imshow(img, aspect='auto')
            if title==False:
                ax[i//6][i%6].set_title(list_fruits[i//6])
            elif title and Y is not None:
                ax[i//6][i%6].set_title(Y[i])
        plt.show()
    else:
        print('No puedo plotear. La variable ingresada debe tener 36 imágenes.')
```

Usamos la función show_subplot para poder mostrar la imagen de nuestros datos seleccionados en forma aleatoria.

```
show_subpot(x_aleat, list_names)
```

Llamamos a la función show_subplot para mostrar las imágenes.

```
def load_coin_labels(subdir):
    assert ((subdir == "train")| (subdir == "test")), "Sólo se pueden ingresar sub-
directorios que sean `train` o `test`."
    quality=['real', 'falso']
    X,Y=[],[]
    z=[]
    baseDirectory="dataset/"
    for cata in tqdm(os.listdir(baseDirectory+subdir+"/")):
        if quality[0] in cata:
            path_main=os.path.join(baseDirectory+subdir+"/",cata)
            for img_name in os.listdir(path_main):
                img=cv2.imread(os.path.join(path_main,img_name))
                img=cv2.resize(img,(500,500))
                img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
                z.append([img,0])
        elif quality[1] in cata:
            path_main=os.path.join(baseDirectory+subdir+"/",cata)
            for img_name in os.listdir(path_main):
                img=cv2.imread(os.path.join(path_main,img_name))
                img=cv2.resize(img,(500,500))
                img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
                z.append([img,1])
        elif quality[2] in cata:
            path_main=os.path.join(baseDirectory+subdir+"/",cata)
            for img_name in os.listdir(path_main):
                img=cv2.imread(os.path.join(path_main,img_name))
                img=cv2.resize(img,(500,500))
                img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
                z.append([img,2])

    print('Shuffling your data.....')
    shuffle(z)
    for images, labels in tqdm(z):
        X.append(images);Y.append(labels)
    return X,Y
```

function load_coin_labels que nos ayudara a cambiar la dimensión de nuestras imágenes y repartirlas aleatoriamente entre las distintas divisiones train y test.

```
x_coin_train_color,y_coin_train_color=load_coin_labels(subdir='train')
x_coin_test_color,y_coin_test_color=load_coin_labels(subdir='test')
```

almacenamos las salidas de la función load_coin_labels y así repartir el data set en este caso

```
100%|                    | 2/2 [00:02<00:00,  1.07s/it]
Shuffling your data.....
100%|                    | 270/270 [00:00<00:00, 590439.04it/s]
```

las imágenes .

```
x_coin_train_color=np.array(x_coin_train_color)
y_coin_train_color=np.array(y_coin_train_color)

x_coin_train_color.shape, y_coin_train_color.shape
```

corroboramos las dimensiones que tiene nuestra data.

```
((1527, 500, 500, 3), (1527,))
```

```
def ClassNumberToCoinName(namber):
  listaNames = ['moneda 2015', 'moneda falsa']
  return listaNames[namber]
```

la función ClassNumberToCoinName es utilizada para categorizar las etiquetas de moneda falsa y verdadera a cifras entendibles por el modelo (0, 1).

```
x_coin_train_color = x_coin_train_color.astype('float32')
x_coin_test_color = x_coin_test_color.astype('float32')
x_coin_train_color /= 255
x_coin_test_color /= 255
```

pasamos a standarizar la data de las imágenes con una división simple que podrá los valores entre 0 y 1.

```
#bibliotecas necesarias para transfer learning
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint

from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications. imagenet_utils import preprocess_input, de-
code_predictions

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D , Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator # para gen-
erar data augmentation
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.densenet import DenseNet201
from tensorflow.keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.applications.resnet import ResNet50
```

Pasamos a llamar específicamente a las utilidades de transfer learning.

```
base_model = VGG19 (include_top=False, input_shape=(500,500,3), pooling='avg')
base_model.summary()
```

Pasamos a crear nuestro modelo base, gracias a Keras la forma de crear un modelo pre entrenado es solamente instanciar el modelo que se necesita con para metros de entrada y salida

```
Downloading data from https://storage.googleapis.com/tensorflow/kera
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500, 500, 3)]     0

 block1_conv1 (Conv2D)       (None, 500, 500, 64)      1792

 block1_conv2 (Conv2D)       (None, 500, 500, 64)      36928

 block1_pool (MaxPooling2D)  (None, 250, 250, 64)      0

 block2_conv1 (Conv2D)       (None, 250, 250, 128)     73856

 block2_conv2 (Conv2D)       (None, 250, 250, 128)     147584

 block2_pool (MaxPooling2D)  (None, 125, 125, 128)     0

 block3_conv1 (Conv2D)       (None, 125, 125, 256)     295168

 block3_conv2 (Conv2D)       (None, 125, 125, 256)     590080

 block3_conv3 (Conv2D)       (None, 125, 125, 256)     590080

 block3_conv4 (Conv2D)       (None, 125, 125, 256)     590080

 block3_pool (MaxPooling2D)  (None, 62, 62, 256)       0

 block4_conv1 (Conv2D)       (None, 62, 62, 512)       1180160

 block4_conv2 (Conv2D)       (None, 62, 62, 512)       2359808

 block4_conv3 (Conv2D)       (None, 62, 62, 512)       2359808

 block4_conv4 (Conv2D)       (None, 62, 62, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 31, 31, 512)       0

 block5_conv1 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_conv2 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_conv3 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_conv4 (Conv2D)       (None, 31, 31, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 15, 15, 512)       0

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
```

```
precomputed_train = base_model.predict(x_coin_train_color, batch_size=32, verbose=1)
precomputed_train.shape
precomputed_test = base_model.predict(x_coin_test_color, batch_size=32, verbose=1)
precomputed_test.shape
```

Hacemos una “pre entrada” del Dataset para tener resultados y no esperar un tiempo aprendizaje largo ya que el modelo ya se encuentra pre entrenado.

```
top_model = Sequential([
    Dense(128, activation='relu', input_shape=(512 ,)),
    Dense(num_classes, activation='softmax')
])

top_model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001), met-
rics=['accuracy'])
top_model.summary()
```

usamos la función Sequential dar paso a la creación del modelo cabeza, así como con nuestro modelo pre entrenado instanciamos los datos de entrada que será la salida de nuestro modelo y su salida que será el número de clases que tenemos, en este caso 2.

```
%%time
registro_three = top_model.fit(precom-
puted_train, y_coin_train_color,epochs=100,steps_per_epoch=80, batch_size=128, vali-
dation_data=(precomputed_test, y_coin_test_color))
```

pasamos a entrenar el modelo cabeza con el Dataset pre computada, así como ingresar los parámetros de entrenamiento, todo esto nos permite la función. fit característica de cada modelo de Machine Learning existente, almacenando nuestro entrenamiento en la variable registro_three.

```
plt.plot(registro_three.history['loss'], '--', color='red', label='datos entrena-
dos')
plt.plot(registro_three.history['val_loss'], '--', color='pink', label='da-
tos de test')
plt.title('Loss curves')
plt.legend()
plt.show()
```

Dibujamos la curva de aprendizaje de la variable registro_three.

```
for layer in base_model.layers:
    # Boolean whether this layer is trainable.
    traina-
ble = ('block5_conv3' in layer.name or 'block5_conv4' in layer.name  or 'block5_pool
' in layer.name   or 'global_average_pooling2d' in layer.name )

    # Set the layer's bool.
    layer.trainable = trainable
```

usamos la siguiente línea de código para cambiar a trainable la capa convolucional 4 del bloque 5 de nuestra red pre-entrenada.

```
def print_layer_trainable():
    for layer in base_model.layers:
        print("{0}:\t{1}".format(layer.trainable, layer.name))
```

definimos la función print_layer_traainable para conocer las capas entrenadas y no entrenadas de nuestro modelo base

llamando a la función obtenemos:

```
False:  input_1
False:  block1_conv1
False:  block1_conv2
False:  block1_pool
False:  block2_conv1
False:  block2_conv2
False:  block2_pool
False:  block3_conv1
False:  block3_conv2
False:  block3_conv3
False:  block3_conv4
False:  block3_pool
False:  block4_conv1
False:  block4_conv2
False:  block4_conv3
False:  block4_conv4
False:  block4_pool
False:  block5_conv1
False:  block5_conv2
True:   block5_conv3
True:   block5_conv4
True:   block5_pool
True:   global_average_pooling2d
```

```
predictions = final_model.predict(x_coin_test_color[:200])

# Print our model's predictions.
print("Clases predichas por el modelo: ", np.argmax(predic-
tions, axis=1)) # [0, 0, 0, 0, 0]
print(y_coin_test_colorF[0:200])
# Check our predictions against the ground truths.
y_coin_test_color.shape
```

en la siguiente línea de código usamos la función predict de del modelo entrenado para pasar 200 datos de validación y conseguir un resultado reflejado en la matriz de confusión.

```
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

Usamos la función plot_confusion_matrix de la librería matplolib para dibujar la matriz dando como entradas nuestros resultados.

Nota: Mostramos el código completo del modelo.