Escuela
Politécnica
Superior

Aumentado de datos para reconocimiento automático de notación musical con técnicas de Deep Learning

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Juan Carlos López Gutiérrez
Tutor/es:
Jorge Calvo Zaragoza

Universitat d'Alacant
Universidad de Alicante

Julio 2020

Aumentado de datos para reconocimiento automático de notación musical con técnicas de Deep Learning

Autor
Juan Carlos López Gutiérrez

Tutor/es
Jorge Calvo Zaragoza
Departamento de Lenguajes y Sistemas Informáticos

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2020

Preámbulo

Esta memoria se describe el desarrollo realizado para mejorar un sistema de reconocimiento de símbolos notación notación musical donde se utiliza una arquitectura de red convolucional recurrente.

Para conseguir este objetivo, se ha implementado un algoritmo de aumentado de datos el cual se encarga de modificar las imágenes utilizadas en el proceso de entrenamiento del modelo. Consiguiendo reforzar el proceso de entrenamiento mejorando la precisión del modelo.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

Hoy en día contamos con la potencia computacional suficiente para poder resolver muchos problemas que en el pasado eran imposibles de llevar a cabo por un ordenador. Pero actualmente la cosa ha cambiado, ya que podemos entrenar redes neuronales que nos ayuden a solventar diversos problemas, tareas que antes no se podían realizar con unos buenos resultados o bien que tenían un coste demasiado elevado.

Estas redes neuronales son modelos estadísticos que implementan algoritmos los cuales son capaces de aprender —gracias a grandes conjuntos de datos— a distinguir características, dotándoles de la capacidad de hacer predicciones sobre datos de entrada que no ha visto nunca el modelo.

Uno de los problemas más conocidos donde las redes neuronales juegan un gran papel es en la clasificación de imágenes de manera automática. Dichas redes son capaces de distinguir si la imagen que está tomando como dato de entrada contiene un perro, un gato o cualquier clase de animal si se ha entrenado para desarrollar dicha capacidad. También pueden ser capaces de distinguir números, objetos, personas y un largo etcétera.

En concreto, el problema con el que se ha trabajado en este proyecto es la extracción de cadenas de símbolos a partir de imágenes de partituras manuscritas, un problema que está siendo investigado en el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante, en el grupo GRFIA.

Esta investigación consiste en conseguir reconocer a través de Inteligencia Artificial (IA) diversas notaciones musicales con la finalidad de conseguir un modelo capad de reconocer secuencias de símbolos contenidas en los pentagramas dispuestos en una partitura tomando como entrada una imagen de la esta. Con lo cual conseguir almacenar información de partituras antiguas de una manera rápida y sencilla, evitando el problema del proceso tan lento y tedioso que conlleva obtener dicha información a mano.

Partiendo de este modelo, la investigación que se ha realizado en este trabajo de investigación ha consistido en mejorar el entrenamiento del modelo de red neuronal utilizada en este departamento. Se ha tratado de mejorar la capacidad de clasificación de los símbolos comprendidos en un pentagrama con la minimización de errores cometidos obteniendo dicha información.

Para mejorar estos resultados, se ha tratado de mejorar el dataset utilizado para el entrenamiento de los modelos ya que, a pesar de que esta arquitectura contaba con una tasa de acierto elevada, se necesitaba afinar más la clasificación.

Uno de los puntos clave a la hora de entrenar redes neuronales es contar con un buen conjunto de datos de entrenamiento con el que entrenar nuestra red. Este conjunto de datos está formado con un conjunto de imágenes etiquetadas, siendo en este caso imágenes de partituras de las cuales se cuenta con la localización de los distintos pentagramas dispuestos en la partitura y una lista de símbolos con los cuales cuenta cada uno de ellos. Esta lista de símbolos es la lista que se espera obtener de la red.

Para conseguir una mayor precisión trabajando con el conjunto de datos con el que se cuenta, lo que se ha hecho es aplicar una técnica conocida como aumentado de datos (data augmentation), la cual se utiliza para conseguir modelos más robustos ante variaciones en los datos introducidos como entrada las cuales no están contempladas en el dataset del que se dispone.

Estas variaciones podrían ser diversos niveles de luminosidad o imágenes tomadas desde distintos ángulos. Esta carencia de variabilidad puede provenir por un conjunto de datos reducido o bien porque las imágenes con las que se cuenta sean demasiado parecidas entre sí.

Dicha técnica consiste en aplicar diversas transformaciones a las imágenes de las que se dispone, modificando la salida esperada de la red si es necesario, consiguiendo de esta manera aumentar la variabilidad de datos de entrenamiento. Gracias a este aumento se consigue que la red neuronal pueda aprender de manera más genérica a extraer información de las imágenes en el momento en el que hacemos una predicción consiguiendo una mayor tasa de acierto en imágenes que nunca ha visto.

En este caso se piensa el conjunto de datos es demasiado uniforme, siendo este el motivo por el cual modelo no llega a dar mejores resultados. Esto se debe a que el modelo no aprende a extraer las características clave para poder distinguir entre símbolos y posiciones en el pentagrama de una manera más genérica clasificando correctamente símbolos en pentagramas que no ha visto en el proceso de entrenamiento como pueden ser imágenes de pentagramas tomadas a distintos ángulos de los pentagramas con los que se ha entrenado.

1.1 Motivación

En la sociedad actual la música es un componente crucial de nuestras vidas ya que convivimos con ella todos los días tanto en la televisión, en la radio o incluso en nuestros móviles donde podemos escuchar a nuestros artistas preferidos sea cual sea el lugar en el que nos encontremos. ¿Pero qué pasa con la música tradicional? ¿Con la música no está codificada en un ordenador?

La música ha sido interpretada por artistas desde hace siglos, no solo se trata de mero entretenimiento, sino que forma parte de nuestra sociedad y de nuestra cultura. Y todavía a día de hoy gran parte de la música compuesta a lo largo de toda nuestra existencia todavía se conserva en manuscrita pudiendo llegar a perderse por culpa del deterioro de los materiales o por el extravío de partituras.

Hoy en día, gracias a los avances con las técnicas de Deep Learning (LeCun y cols., 2015), se puede lograr evitar el lento y tedioso proceso que implica la trascripción de las partituras de todos estos manuscritos de forma manual, evitando de esta manera que puedan perderse en el tiempo para siempre.

Para realizar esta tarea hay que encontrar el mejor modelo de red neuronal y refinarla al máximo consiguiendo que consiga identificar todos y cada uno de los símbolos escritos en estas partituras en función de la notación que en esta se esté aplicando pudiendo almacenar de manera correcta la información musical que estas contienen.

1.2 Objetivos

Los objetivos que tiene este trabajo son los siguientes:

- Comprensión de un modelo de red neuronal diseñada para el reconocimiento de notación musical mejorando los conocimientos sobre estructuras de redes neuronales, tratamiento de datos e interpretación de resultados.

- Mejora de los conocimientos previos sobre procesamiento de imágenes. Profundizando en en el funcionamiento de la aplicación de diversas transformaciones a las imágenes de manera práctica.

- Mejora del funcionamiento de una red neuronal utilizada en un proyecto real. Pudiendo afianzar los conocimientos sobre el mecanismo de desarrollo de un proyecto de investigación.

- Análisis de los datos de los que se dispone para poder desarrollar transformaciones coherentes las cuales puedan ayudar al proceso de aprendizaje para prevenir la introducción de error.

1.3 Estructura

Para facilitar la lectura, los contenidos del trabajo se han desglosado en diversos apartados para tener una visión más general del tema que se va a tratar en cada uno de ellos. Los apartados con los que cuenta el trabajo son los siguientes:

- **Capítulo 1 - Introducción:** Introducción en el trabajo donde se explica la importancia del mismo y los objetivos que se han llevado a cabo a lo largo del proyecto.
- **Capítulo 2 - Marco teórico:** Capítulo explicativo de las bases teóricas del problema de la clasificación de imágenes y como se ha tenido en cuenta para la realización de este trabajo.
- **Capítulo 3 - Tecnologías:** Apartado explicativo de las distintas tecnologías utilizadas a lo largo de todo el desarrollo del trabajo.
- **Capítulo 4 - Metodología:** Desglose detallado de las distintas modificaciones implementadas para conseguir el aumentado de datos buscado.
- **Capítulo 5 - Experimentación:** Descripción del diseño de los experimentos realizados junto con los resultados obtenidos con los mismos.
- **Capítulo 6 - Conclusiones:** Comparación de los resultados esperados con el impacto real que ha tenido la investigación.

2 Marco teórico

Para llevar a cabo esta predicción, se utiliza una estructura de red convolucional recurrente la cual es capaz de reconocer la secuencia de símbolos contenida en un pentagrama.

Esta red devuelve una lista ordenada de principio a fin la cual está formada por parejas que constan del tipo de nota o símbolo que representa un área determinada del pentagrama y una altura en función de la posición que esta nota toma en el mismo (el número de línea o espacio donde se sitúa la nota en el pentagrama).

El problema a abordar es intentar mejorar las predicciones de la red intentando generalizar el aprendizaje de la misma. Intentando conseguir que la red generalice el conocimiento, para lo cual se va a aplicar la técnica de aumentado de datos.

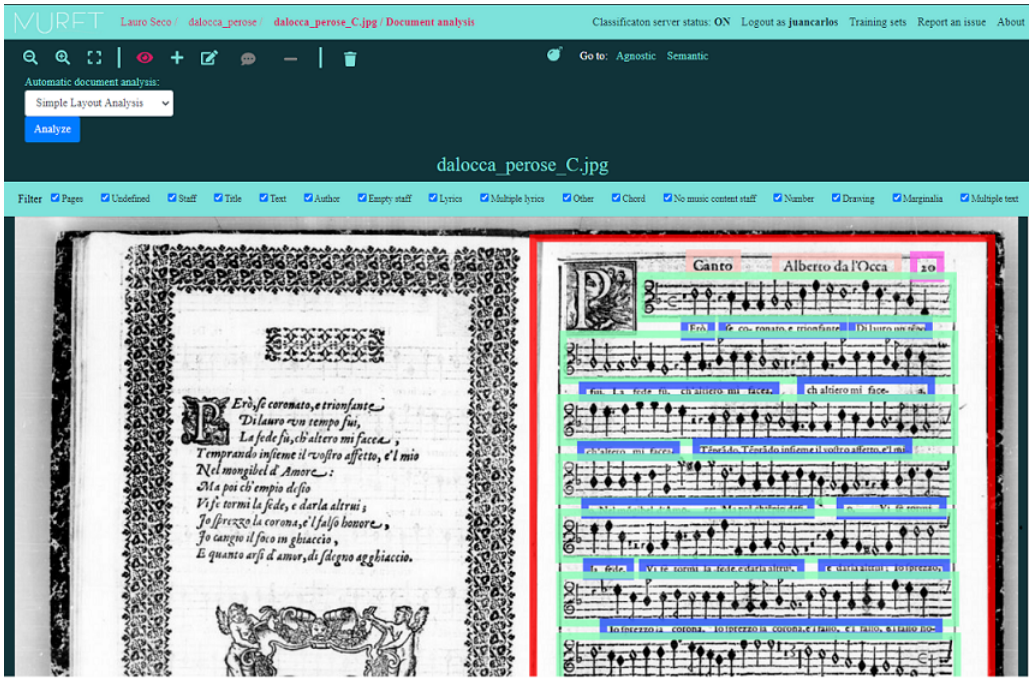
Para conseguir esto, la primera tarea es familiarizarse con el problema tratando de encontrar las modificaciones adecuadas para aplicar al conjunto de imágenes basándose en las imágenes con las que se cuenta e investigando modificaciones que se podrían dar las cuales no se cubren con el conjunto de datos con el que se dispone.

La información sobre los pentagramas dispuestos en las distintas páginas está almacenada en un archivo json por cada una de las imágenes. Dentro de este fichero están descritas las coordenadas donde están distintos los pentagramas de dicha página junto cadena de símbolos a identificar dentro de cada uno de ellos.

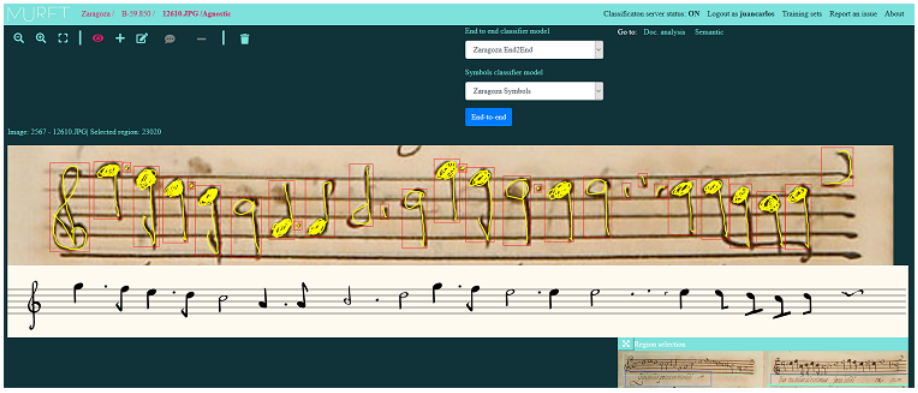
Esta estructura se adquiere gracias a una herramienta incluida en muret (Figura 2.1), la cual permite, con un entorno gráfico, seleccionar las áreas donde se encuentran dispuestos los distintos pentagramas dentro de cada una de las partituras. Tras esto se puede etiquetar cada uno de los símbolos comprendido en cada uno de los pentagramas ya etiquetados. Una vez seleccionados, se genera un fichero json con el formato que se muestra a continuación.

Código 2.1: Ejemplo de la estructura en formato json de la información capturada en la etiquetación de imágenes

```
{
"filename": "fichero.jpg",
"pages": [
    {
        "regions": [
            {
                "bounding_box": {
                    "fromX": 643,
                    "toX": 1123,
                    "fromY": 42,
                    "toY": 146
                },
                "id": 26425,
                "type": "staff",
                "symbols": [
                    {
                        "agnostic_symbol_type": "clef.C",
                        "approximateX": 643,
                        "id": 136715,
                        "position_in_staff": "L4"
                    },
                    {
                        "agnostic_symbol_type": "metersign.Ct",
                        "approximateX": 662,
                        "id": 136716,
                        "position_in_staff": "L3"
                    }
                ]
            }
        ],
        "bounding_box": {
            "fromX": 620,
            "toX": 1128,
            "fromY": 32,
            "toY": 733
        },
        "id": 3063
    }
],
"id": 3965,
"collection": ""
}
```



Etiquetación de bounding boxes de los pentagramas



Etiquetación de los símbolos de cada pentagrama

Figura 2.1: Herramienta de etiquetado integrada en MURET

2.1 Clasificación de imágenes

Para poder entender cómo es posible que una inteligencia artificial sea capaz de reconocer y listar todas las notas comprendidas en los pentagramas, primero hay que entender las bases de la inteligencia artificial, qué es la visión por computador. Explicando lo que son las características, los filtros, y como a partir de esta información, se puede clasificar una imagen con estructuras como las Support Vector Machine (SVM), con las que podemos clasificar imágenes con una alta probabilidad de acierto siempre y cuando hayamos obtenido unas buenas características.

2.1.1 Características

El ser humano es capaz de reconocer en un escenario, ya sea una foto como en la vida real, cualquier forma de animal, objeto o persona conocida sin hacer demasiado esfuerzo. Para poder hacer esto, nosotros podemos identificar formas que conocemos ya que somos capaces de reconocer características propias aquello que estamos observando, como podría ser el cuello largo de una jirafa o rasgos de un familiar en una fotografía. Gracias a estas características que podemos identificar de una manera tan sencilla, somos capaces de distinguir millones de objetos, animales y personas diferentes.

Ahora bien, ¿Cómo podemos transmitirle ese conocimiento a una inteligencia artificial para que pueda identificar esos objetos, personas o animales de la misma manera que nosotros? Desafortunadamente, esa pregunta por el momento no tiene respuesta, pero sí que existen mecanismos para que, al igual que nosotros somos capaces de distinguir objetos o cosas por ciertos rasgos, un ordenador sea capaz de encontrar esos ”rasgos” (características) dentro de las imágenes, características con las que en u proceso posterior, sea capaz de identificar de qué objeto o cosa se trata.

2.1.2 Filtros

Una de las partes más importantes para poder llevar a cabo esta identificación es la extracción de características. Hasta hace no demasiados años, por culpa de la escasa potencia computacional, la mejor manera que se conocía era la extracción de estas características con la aplicación de filtros a las imágenes. Tratando de encontrar los mejores filtros para cada tipo de problema a base de ensayo y error tratando de identificar los objetos en las imágenes con modelos como las SVM de las cuales hablaremos más adelante.

La eficacia de estos filtros se basaba en la capacidad de extraer las características más importantes para poder clasificar las imágenes de entrada despreciando toda la información irrelevante como puede ser una región de la imagen en la que el color es el mismo en toda ese área por lo que no aportar información alguna para poder clasificar dicha imagen y dejando información como pueden ser esquinas o bordes, información verdaderamente relevante para su clasificación.

Dichos filtros funcionan gracias a lo que se conoce como kernel. Un kernel se trata de una matriz NxN con el cual se van haciendo cálculos por pasos sobre toda la imagen. Estos cálculos se van aplicando por toda la imagen recorriendo el kernel a modo de ventana deslizante. Esto da como resultado, para cada posición en la que se mueve la ventana, una matriz con los valores resultantes de la suma de la multiplicación de cada uno de los valores de la matriz con los correspondientes valores de la imagen en la posición en la que se encuentra dicha ventana como podemos ver en la Figura 2.2.

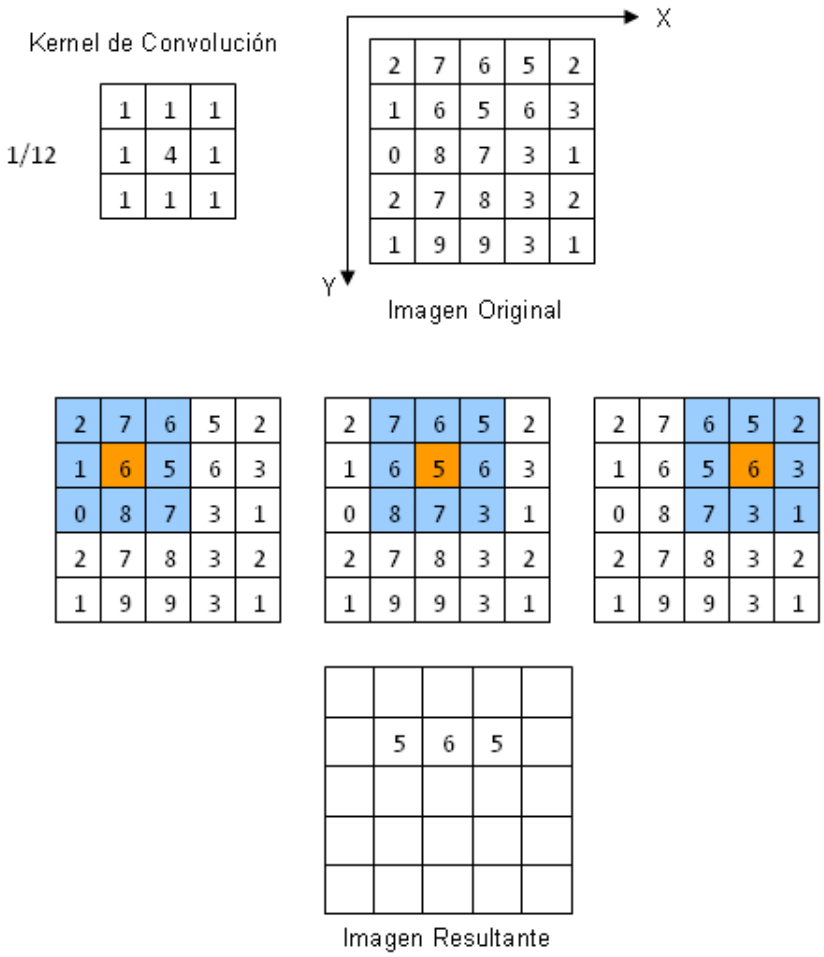


Figura 2.2: Aplicación del kernel a una matriz

Uno de los filtros más conocidos es el cálculo de gradiente con el operador Sovel. Este filtro trata de identificar los bordes en los distintos objetos en escena descartando así la información de zonas poco relevantes. La teoría es que si no hay un cambio en la intensidad de color en esa zona, no aporta información relevante para la clasificación de la imagen mientras que las zonas que haya cambios más bruscos, son las zonas que más información nos pueden aportar.

Este operador en concreto utiliza dos filtros (kernels) 3x3 con los que se extrae la información del gradiente en X y el gradiente en Y, identificando los bordes sobre el eje X y el eje Y respectivamente. Para conseguir esto, los kernel de Sobel tienen la forma que se puede apreciar en la Figura 2.3.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Figura 2.3: Kernels para el cálculo de gradiente Y y X

Lo que consiguen con estos kernels es remarcar los aumentos de intensidad en el eje de las Y [1] y en el eje de las X respectivamente. Pudiendo de esta manera obtener los bordes tanto en X como en Y de las imágenes como podemos ver en la Figura 2.4.

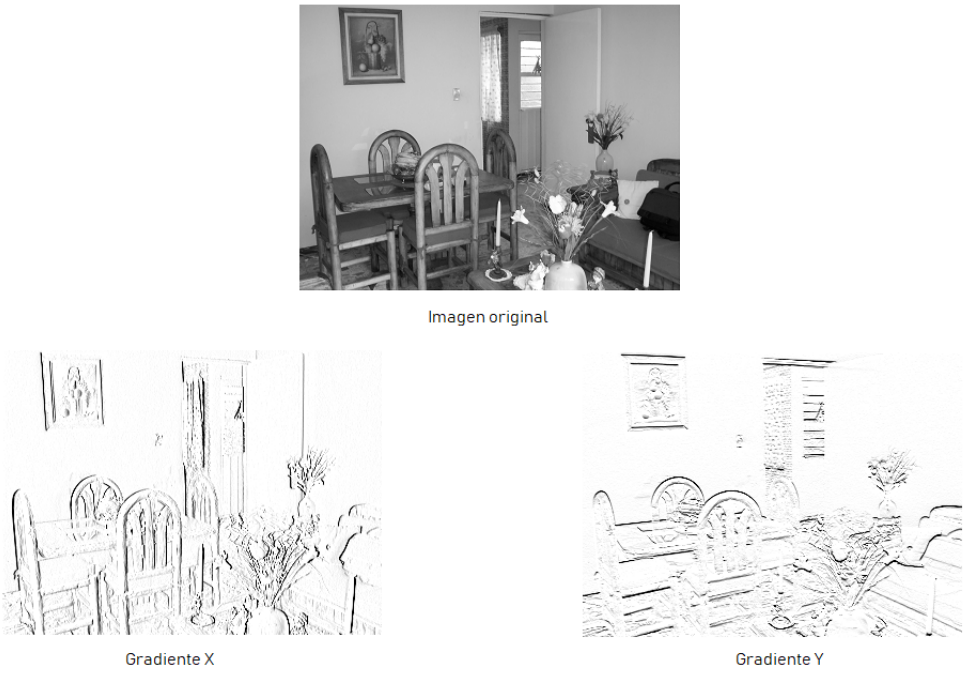


Figura 2.4: Aplicación de cálculo de gradientes

[1] Hay que tener en cuenta que las imágenes tienen el punto de origen en la esquina superior izquierda por lo que el eje Y aumenta desde la parte superior de la imagen hasta la parte inferior. Por este motivo el kernel que calcula los gradientes en el eje Y, tiene los valores más pequeños en la parte superior y los más grandes en la inferior.

Tras la aplicación de ambos kernels, las matrices resultantes se combinan para la obtención de una sola matriz de características donde se encuentran las esquinas y bordes en ambos ejes. Esta combinación se hace con la siguiente fórmula:

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

Como resultado obtenemos una imagen como la de la Figura 2.5.

Figura 2.5: Combinación de matrices de gradientes X e Y

2.1.3 Perceptrón

En el momento en el cual se han conseguido unas buenas características con la aplicación de diversos filtros, el siguiente paso es conseguir una estructura la cual nos permita clasificar haciendo uso de la información obtenida. Para conseguir esto, la estructura más básica es el perceptrón. Su estructura aparece en la Figura 2.6.

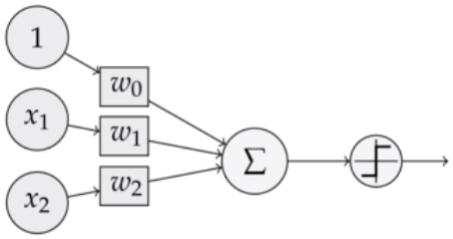


Figura 2.6: Estructura perceptrón

Un perceptrón es un modelo matemático diseñado con la finalidad de separar de manera lineal un conjunto de datos (Stephen, 1990). Para poder conseguir esto, el conjunto que se quiere separar ha de ser linealmente separable.

El perceptrón está compuesto por una serie de de entradas, por las cuales se les proporciona la información del objeto a clasificar y se obtiene como salida un único valor el cual puede tener un valor de 0 o 1. Esta salida se interpreta tomando el valor 1 como perteneciente a la clase y 0 cuando no pertenece.

El cálculo de la salida del perceptrón se realiza con la siguiente ecuación:

$$out = b + \sum_{i=1}^{n} x_i * w_i$$

Donde cada una de las variables son:

- **N:** Número de valores de entrada del perceptrón.
- **Xi:** Valores de la entrada del perceptrón.
- **Wi:** Peso asociado a cada una de las entradas del perceptrón.
- **B:** Bias, parámetro encargado de separar la función discriminatoria del origen de coordenadas. Se puede ver de manera gráfica en la Figura 2.7.

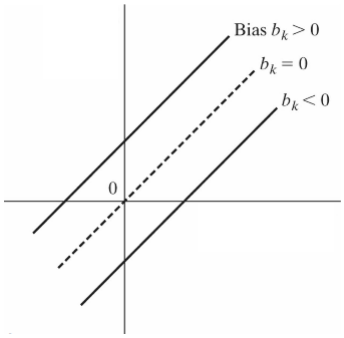


Figura 2.7: Efecto del Bias

Para poder ver un ejemplo visual de esta separación, nos podemos en el conjunto plasmado en la Figura 2.8. Este conjunto de datos está formado por puntos distribuidos en una superficie bidimensional

Dentro de este conjunto se quiere clasificar los puntos según si pertenecen a la clase roja o a la clase negra. Para esto se utiliza un perceptrón formado por dos entradas como el de la figura 2.6.

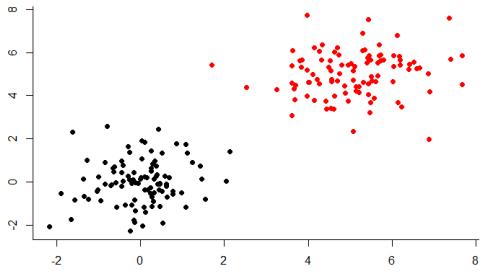


Figura 2.8: Colección de puntos en el espacio

Para que el perceptrón sea capaz de clasificar los valores de entrada, este tiene que ser entrenado modificando el valor de pesos asociados a cada una de las entradas junto al valor del bias. Para entrenar un perceptrón se aplica la siguiente fórmula:

$$w(j)' = w(j) + \alpha(\delta - y)x(j)$$

Dicha fórmula cuenta con las siguientes partes:

- **x(j):** Elemento en la posición j del vector de entrada.
- **w(j):** Peso asociado al valor de entrada j.
- **y:** La salida del perceptrón.
- **delta:** La salida esperada.
- **alfa:** Tasa de aprendizaje.

Esta fórmula se aplica para ajustar los pesos del perceptrón tomando como valores de entrada las predicciones fallidas. Este proceso se efectúa en bucle tratando de encontrar la calibración con la que el perceptrón con la cual es capaz de clasificar de manera correcta todos los puntos del conjunto [2]. Se pueden apreciar los resultados en la Figura 2.9.

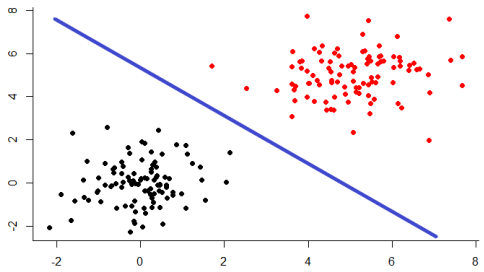


Figura 2.9: Colección de puntos separados

[2]Esto solo funciona en conjuntos que son linealmente separables, es por esto por lo que el bucle de entrenamiento se limita a un número determinado de iteraciones ya que el algoritmo puede no terminar nunca.

2.1.4 Perceptrón multicapa

El principal problema del perceptrón es la manera con la cual divide el espacio en dos partes gracias a un hiperplano impidiendo que pueda clasificar de forma correcta estos conjuntos.

El ejemplo más sencillo donde se puede ver este problema es tratando de entrenar un perceptrón para que imite el comportamiento de una puerta lógica XOR. Como podemos ver en la Figura 2.10, es imposible separar los puntos con un hiperplano de manera que se queden a un lado los valores que dan como resultado 1 (puntos en la región azul) y al otro los valores que dan como resultado 0.

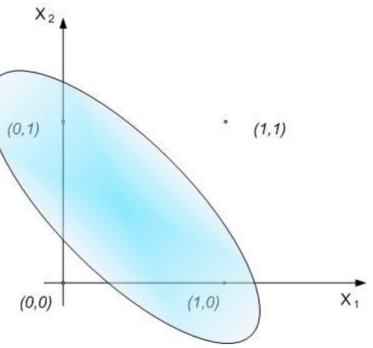


Figura 2.10: Representación gráfica de una puerta XOR

Es por esto por lo que nace lo que se denomina perceptrón multicapa. Esta estructura utiliza el concepto de perceptrón para poder dividir el espacio en formas más complejas a la de un simple hiperplano.

Para esto se añaden lo que se conoce como hidden layers (capas ocultas), 2.11. Estas capas están formadas por nuevas neuronas las cuales tienen como entrada la salida de las neuronas de la capa anterior. Dichas capas son las encargadas de generar esta deformación del hiperplano consiguiendo nuevos tipos de separación no lineal del espacio (Pal y Mitra, 1992).

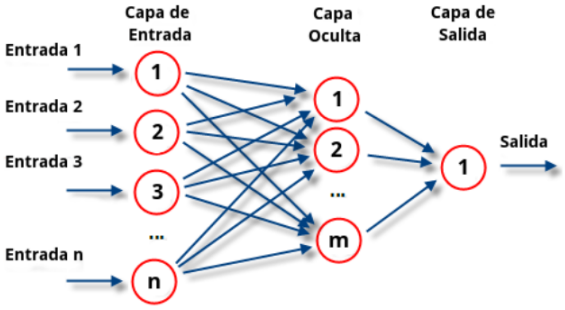


Figura 2.11: Estructura de un perceptrón multicapa

También se modificó la estructura del perceptrón, en el perceptrón multicapa sustituye la decisión forzosa de 1 o 0 en la salida de la red por una función sigmoidea la cual se encarga de este trabajo. La finalidad de este cambio es para poder propagar el error detectado en el proceso de entrenamiento de una manera efectiva, pudiendo saber qué pesos han influido en la toma de la decisión para poder ajustarlos sin alterar los pesos que no han intervenido en dicha decisión consiguiendo una mejora significativa en el aprendizaje.

2.2 Redes Neuronales

2.2.1 Capas convolucionales

Las capas convolucionales fueron un gran avance en el campo de la clasificación de imágenes ya que estas capas sustituyen la búsqueda de los mejores filtros para extraer características. Estas capas son las encargadas de aprender a identificar las mejores características del conjunto de imágenes de entrenamiento para conseguir clasificar dicho conjunto.

Estas capas cuentan con uno o varios kernels por capa, los cuales se aplican a la imagen para extraer características. Estos kernels se ajustan para encontrar las mejores características de las imágenes gracias a la propagación del error en el algoritmo de entrenamiento como se explica en (Wu, 2017) y en (Sermanet y cols., 2012).

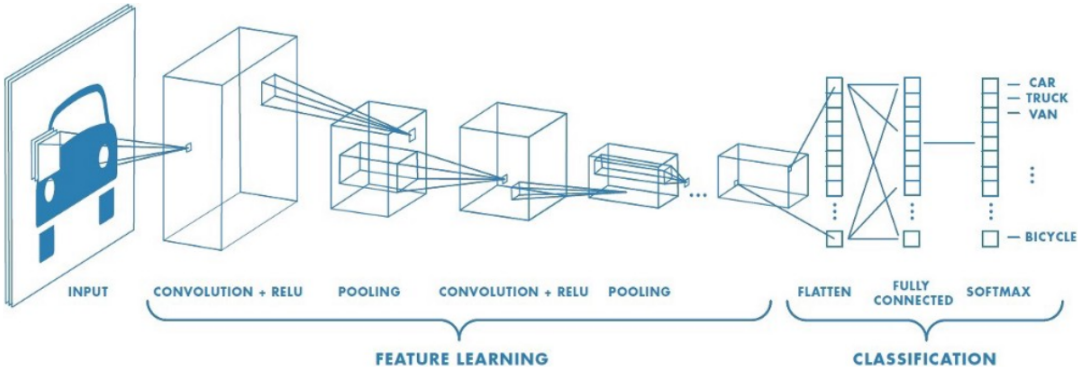


Figura 2.12: Estructura de una CNN

Estas capas suelen ser las primeras dentro de una estructura de red neuronal donde la entrada es una imagen. De esta manera se extraen las características de la imagen de entrada antes del proceso de clasificación, proceso el cual se realiza en las últimas capas las cuales suelen ser capas Fully Connected (FC).

Para utilizar las características más relevantes, tras una capa convolucional (o varias) se suele aplicar una capa la cual aplica una función de pooling, con lo cual se reduce el tamaño de la información obtenida por la capa convolucional extrayendo las características más relevantes de las distintas partes de la imagen como se explica en (Phan y cols., 2016).

2.2.2 Capas recurrentes

Las redes convolucionales recurrentes, o RNN, son redes basadas en la toma de decisiones teniendo en cuenta las predicciones que ya se han realizado.

Las entradas de estas redes son una secuencia de valores los cuales se van proporcionando a la red de manera secuencial. De esta manera, en el momento en el que hacemos la predicción del segundo valor de la secuencia, la red tiene en cuenta el valor de la predicción que se ha generado con el primer valor de la secuencia.

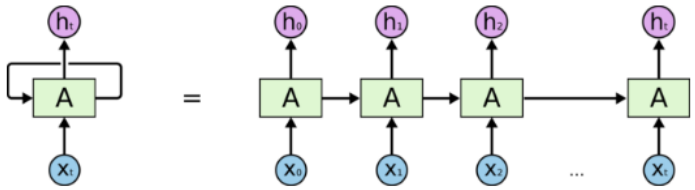


Figura 2.13: Estructura de una red recurrente

Estas capas son capaces de considerar las predicciones anteriores gracias a las celdas por las que están compuestas. Las celdas más conocidas son las celdas LSTM y las celdas GRU.

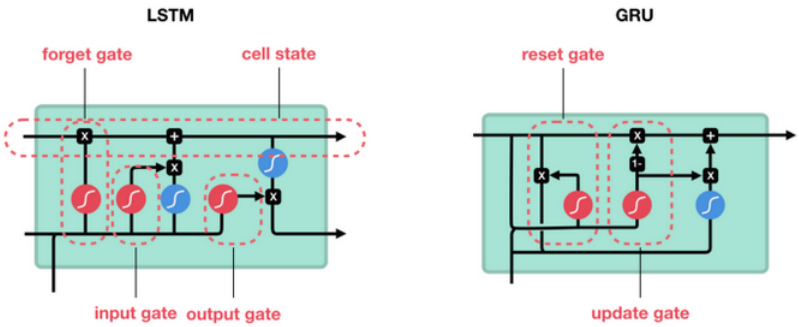


Figura 2.14: Celdas LSTM y GRU

Estas celdas son capaces de almacenar conocimiento previo gracias a la estructura que se puede observar. Ambas estructuras funcionan gracias a las puertas representadas en la imagen. Estas puertas son las encargadas de controlar el flujo de información controlando la influencia que va a tener la información obtenida de la predicción anterior en la predicción resultante en esta etapa.

Estas estructuras son ampliamente utilizadas en el reconocimiento de textos ya que tienen la peculiaridad de poder lidiar con secuencias largas las cuales no tienen un tamaño establecido, pudiendo realizar ateas tan asombrosas como el análisis de sentimientos tanto en textos (Tang y cols., 2015) como en voz (Lee y Tashev, 2015). Pero también son utilizadas en otros campos (Gregor y cols., 2015)

2.2.3 Capas CTC

La capa CTC, o clasificación temporal conexionista, es una estrategia utilizada para entrenar redes recurrentes en las cuales una misma categoría puede ocupar varios instantes temporales, sin tener dicha información de manera explícita en el conjunto de entrenamiento.

Esta técnica modela la red para generar una salida basada en la probabilidad de aparición de una clase en la secuencia, así como la probabilidad de que aparezca lo que se conoce como "blank". El blank, o espacio vacío, se añade en esta capa como una clase extra para poder identificar dos símbolos de la misma clase que estén dispuestos de manera consecutiva como se explica en (Graves y cols., 2006).

En la Figura 2.15, podemos ver una estructura diseñada para el reconocimiento de cadenas de texto manuscritas donde se cuenta con una red neuronal formada por las siguientes partes:

- **CNN** Capas encargadas de la extracción de características de las imágenes correspondientes a cada una de las particiones de la imagen.
- **RNN** Capas encargadas de la interpretación secuencial de las características obtenidas por las capas convolucionales las cuales contemplan la variable temporal de las características extraídas.
- **CTC** La capa ctc encargada de interpretar la salida de las capas recurrentes y clasificar dichas salidas entre el conjunto de símbolos a identificar o un símbolo de tipo blank, según la probabilidad de cada una de las clases a lo largo del tiempo.

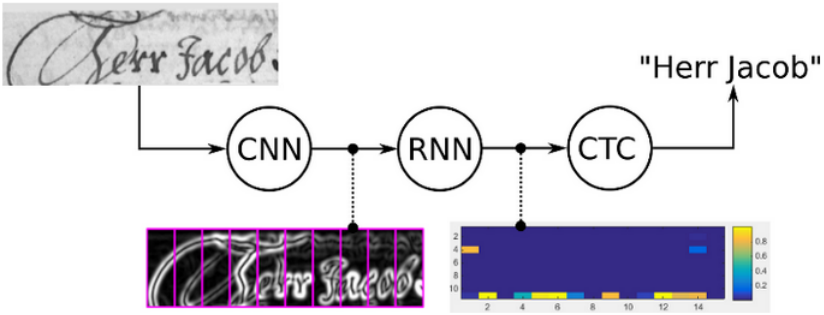


Figura 2.15: Modelo para el reconocimiento de textos escritos a mano

La salida de esta capa se interpreta de la siguiente manera:

- Se genera una lista con las clases más probables a lo largo del tiempo, incluyendo la clase blank.
- Se reducen apariciones consecutivas de una misma clase en la lista formada.
- Se eliminan las ocurrencias de la clase blank en la lista.

Se puede observar un ejemplo de una salida de una estructura similar la cual ha sido implementada en (Calvo-Zaragoza y Rizo, 2018) con esta misma finalidad en la Figura 2.16 donde se ve la salida de la capa CTC antes de hacer la decodificación de la salida.

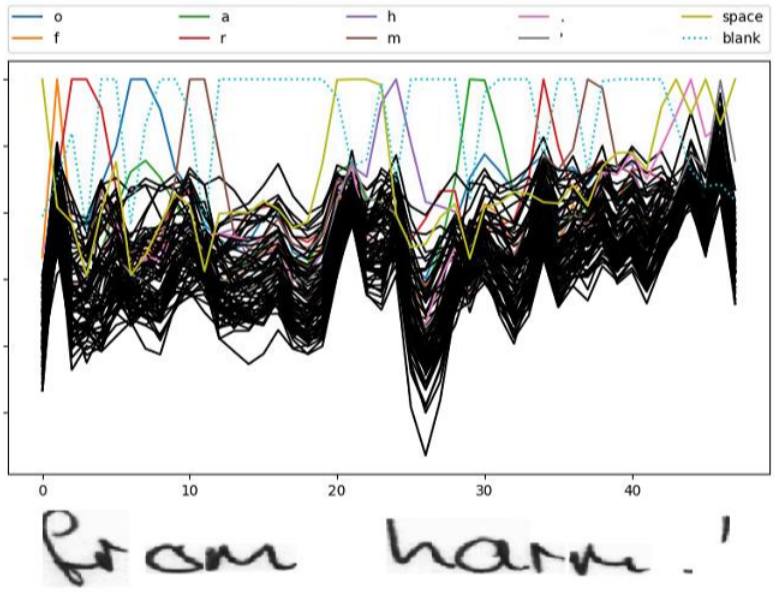


Figura 2.16: Salida de una capa ctc

Una simulación del proceso de decodificación es el mostrado en la Figura 2.17 explicada en el artículo (Hannun, 2017) basado en Chan y cols. (2015).

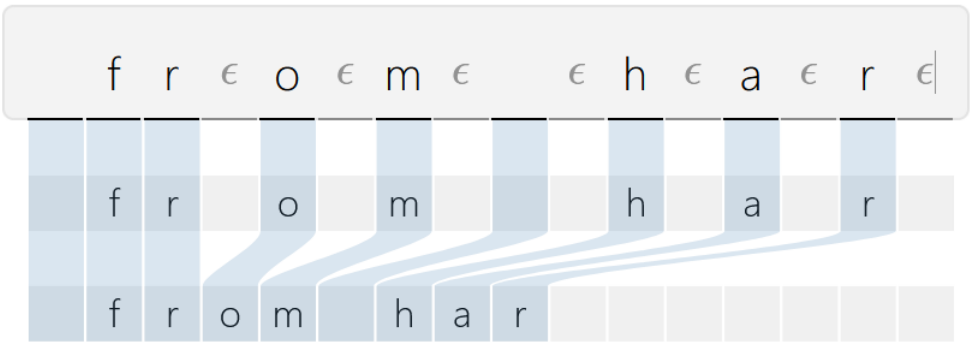


Figura 2.17: Decodificación de la salida de una capa CTC

3 Tecnologías

3.1 Python

Python es el lenguaje utilizado para este proyecto, es un lenguaje interpretado de alto nivel. Python permite generar código el cual es sencillo de comprender, rápido de generar y con unas herramientas muy potentes para poder desarrollar proyectos complejos. Pero el motivo principal por el que se ha utilizado este lenguaje de programación es por las librerías de Machine Learning, operaciones matemáticas y de Computer Vision con las que cuenta.

3.1.1 Tensorflow

Tensorflow (Abadi y cols., 2015) es la principal librería de python para el desarrollo de redes neuronales. Esta librería te permite generar estructuras de redes neuronales con infinidad de formas de una manera muy sencilla. Y cuenta con algoritmos implementados de una manera muy eficiente como es el algoritmo de aprendizaje de estas redes. También es compatible con el uso de GPU lo que agiliza procesos lentos como el algoritmo de aprendizaje de las redes neuronales.

3.1.2 OpenCV

OpenCV (Bradski, 2000) es una librería de computer vision con la cual puede hacer infinidad de operaciones matemáticas con imágenes de cualquier tipo.

Esta librería se ha utilizado principalmente para implementar las distintas transformaciones que se le han aplicado a las imágenes en el proceso de aumentado de datos gracias a todas las operaciones que esta librería tiene implementadas.

3.1.3 Matplotlib

Matplotlib (Hunter, 2007) para visualizar las distintas modificaciones que se han ido aplicando, pudiendo comprobar de manera visual que aplicaban de la manera correcta. Por otro lado, se ha utilizado para representar de manera gráfica la información obtenida en proceso de aprendizaje para comprender mejor el avance de este.

3.1.4 Numpy

Numpy Oliphant (2006) es una librería matemática la cual tiene implementadas infinidad de operaciones de manera eficiente. Esta librería es utilizada por todas las librerías que se han comentado para llevar a cabo los cálculos que cada una de ellas hacen. Se ha utilizado para realizar diversos cálculos de manera sencilla y eficiente como la extracción de los pentagramas de una página y la generación de números aleatorios.

3.2 Google Colaboraty

Google Colaboraty es una herramienta proporcionada por Google, la cual nos permite ejecutar código en servidores dedicados. Esta herramienta ha sido utilizada para comprobar el correcto funcionamiento de cada una de las partes del proceso de aprendizaje. Consiguiendo de esta manera ejecutar algoritmos pesados como son el aprendizaje de redes neuronales sin la necesidad de contar con un ordenador demasiado potente.

Ha sido utilizado principalmente para comprobar la correcta integración del aumentado de datos en el algoritmo de aprendizaje.

3.3 GitHub

GitHub es un repositorio que permite el versionado de código de manera sencilla proporcionando la capacidad que controlar cualquier cambio que se ha realizado, volver a cualquier punto en el que se ha versionado el código.

Gracias a esto es una herramienta muy utilizada en el mundo de la programación permitiendo también contar con diversos estados del código pudiendo elegir qué partes quieres fusionar y cuales quieres mantener en paralelo de manera sencilla.

4 Metodologia

4.1 Introducción

Para este problema, se cuenta con una estructura de red para la que necesitamos que todas las imágenes de pentagramas con las que queramos entrenar tengan la misma altura, mientras que la anchura de la imagen puede ser variable puesto que las predicciones se van haciendo por tramos.

Para generar los conjuntos de entrenamiento se cuenta con conjuntos de imágenes etiquetadas. Dichas etiquetas cuentan con una lista de bounding boxes donde están especificadas las coordenadas de las regiones en las que se encuentran cada uno de los distintos pentagramas dispuestos en la partitura, junto con una lista de símbolos asociados a cada uno de ellos. Estas listas de símbolos representan la salida esperada de la red, el ground of truth de la red neuronal, teniendo como entrada la imagen comprendida entre las coordenadas del boundig box, el pentagrama.

Cuando se quiere hacer aumentado de datos, lo más habitual es aplicar modificaciones a las imágenes se van a pasar por entrada conservando la salida esperada. Como ya hemos visto es una parte clave para conseguir un aprendizaje mucho más genérico para conseguir una detección de mayor calidad aumentando así la tase de acierto de la red ya entrenada.

Ejemplos de estas mencionadas modificaciones son rotar las imágenes, aplicarles un efecto espejo, utilizar un filtro que emborrone las imágenes, un efecto blur, entre otras.

Keras dispone de una herramienta para implementar varias de estas transformaciones haciendo modificaciones de diversos tipos a las imágenes de entrada. En la mayoría de los problemas, esta herramienta es muy útil para conseguir un aumentado de datos correcto, pero la falta de información introduce errores.

Uno de los casos más habituales de introducción de error es en la rotación de las imágenes ya que una rotación normal, conservando el tamaño de la imagen y sin perder parte de la misma, provoca que partes de las imágenes resultantes tengan huecos sin información los cuales hay que rellenar.

Existen diversas maneras de rellenar esos campos como puede ser rellenar esos huecos con 0s. Pero ya que se dispone no solo de la imagen que se va a utilizar como entrada de la red, la imagen del pentagrama, sino que se dispone de la imagen de la partitura en la que este se encuentra. Se puede extraer la zona en la que se encuentra el pentagrama tras rotar la página entera, aportando información contextual más valiosa para enterar el modelo, que rellenar

los espacios vacíos con 0s.

4.2 Estructura de red

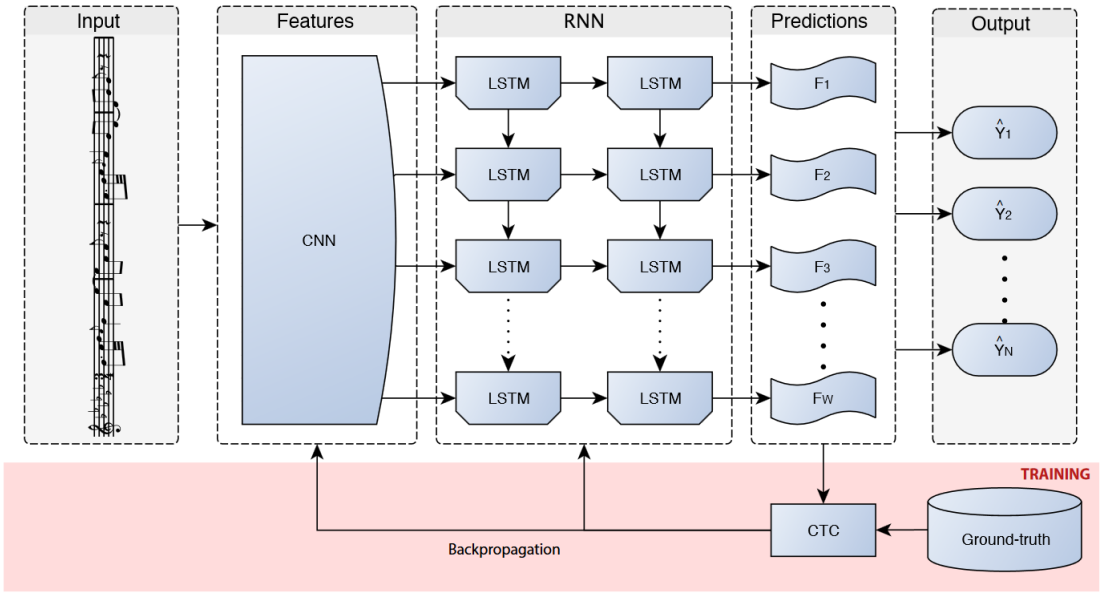


Figura 4.1: Estructura modelo end-to-end

La arquitectura de red que se ha utilizado ha sido la arquitectura diseñada para el reconocimiento de notación musical descrita en el artículo (Calvo-Zaragoza y Rizo, 2018).

Dicha red tiene una estructura similar a la utilizada en el reconocimiento de texto manuscrito. En este caso, la estructura de la red cuenta con un extractor de características, modelado mediante una red convolucional, junto a una sucesión de capas recurrentes, pero en este caso se utiliza la capa CTC para determinar el error de la red en el proceso de entrenamiento, error que se propaga para actualizar los pesos de la red, pero no se utiliza en el proceso de predicción una vez el modelo ha sido entrenado. Esta estructura se puede observar en la Figura 4.1.

4.3 Modificaciones

4.3.1 Rotación

Como se ha comentado, la rotación es una de las modificaciones más comunes a aplicar en los conjuntos de datos de entrada en el proceso de entrenamiento de las redes neuronales. En este problema, las imágenes con las que se cuenta son imágenes hechas por una persona con una cámara. Esto implica que la mayoría de las imágenes, tanto las que se van a utilizar ara entrenar el modelo, como las imágenes de las que se quieran hacer predicciones posteriores, no van a estar cuadradas a la perfección por lo que hay que hacer al modelo robusto ante rotaciones.

Es por esto que esta es la primera modificación implementada. Para realizar esta modificación se realizan los siguientes pasos:

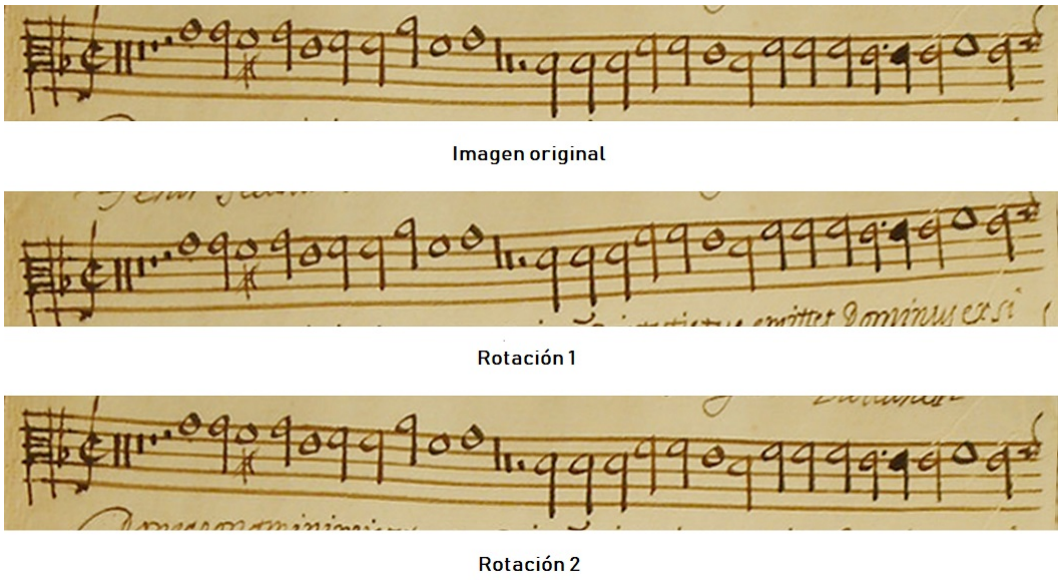


Figura 4.2: Rotación a pentagramas pre recortado

Cálculo de ángulo: Generar un número aleatorio el cual está comprendido entre dos valores definidos. Este valor representa el grado de inclinación se le va a aplicar a la imagen.

Matriz de transformación: Generar la matriz de transformación con la que se aplicarán los grados de inclinación obtenidos en el paso previo

Rotación de imagen: Obtener la imagen rotada aplicando la matriz de transformación a la imagen original

Rotación de puntos: Rotar los puntos del bounding box del pentagrama que se va a recortar

Ajuste de puntos: Ajustar los puntos rotados para que la imagen recortada sea rectangular

Obtención del pentagrama Extraer de la imagen rotada, la región en la que se encuentra el pentagrama rotado

Tras estos pasos, se puede observar el resultado en la Figura 4.2

4.3.2 Margen

La segunda de las modificaciones consiste en aumentar o disminuir el tamaño de la bounding box almacenada en la etiquetación de las imágenes. Al igual que la toma de imágenes, la etiquetación de pentagramas dentro de las imágenes es un trabajo manual por lo que lo normal es que las bounding boxes no siempre se ajusten de la misma manera a los pentagramas reales. Esta modificación se ha implementado de la siguiente manera:

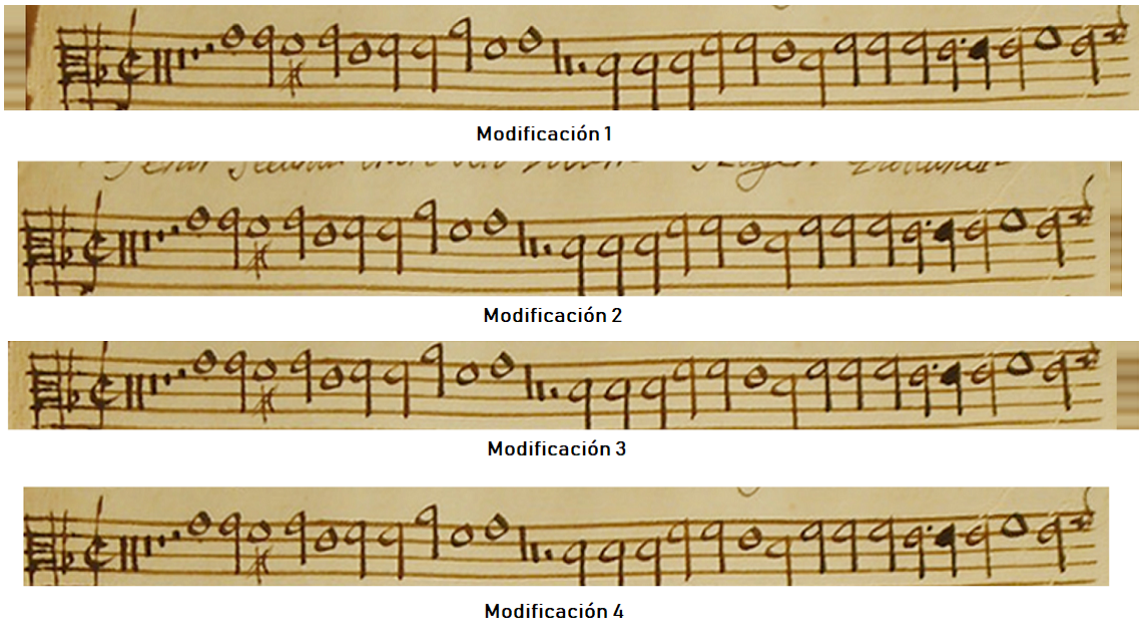


Figura 4.3: Modificación del tamaño de las bounding boxes

Calculo de porcentaje: Se generan 4 números aleatorios tomando valores según un porcentaje de la altura de la bounding box del pentagrama a modificar definido por un parámetro especificado.

Ajuste de puntos: Se ajustan las coordenadas de la bounding box ajenado o acercando los bordes de la bounding box respecto del centro del pentagrama simulando una selección imprecisa de un ser humano.

Obtención del pentagrama: Extraer la región comprendida entre las nuevas coordenadas calculadas.

Tras estos pasos, se puede observar el resultado en la Figura 4.3

4.3.3 Erosión y Dilatación

En este caso, ya la gran mayoría de las partituras de las que se dispone son manuscritas, se ha querido imitar el efecto de escribir con diversas plumas. Para ello se ha aplicado lo que se conoce como erosión y dilatación en las imágenes para conseguir un trazo más grueso o un trazo más fino respectivamente.

Esto parece contradictorio, ya que si quisiéramos un trazo más fino se debería erosionar la nota. Esto se debe a que el efecto de dilatación se consigue seleccionando para cada píxel el valor más alto en la vecindad de este, siendo los valores respectivos al color blanco los mayores valores y negros los de menor valor. De esta manera, como las notas de los pentagramas son de tonos oscuros, dilatar los colores claros implica consumir parte de la nota mientras que erosionar produce el efecto contrario. También hay que tener en cuenta que cuanto mayor es el rango de píxeles en el que se busca el valor más alto, la vecindad, mayor es este efecto de dilatación o erosión.

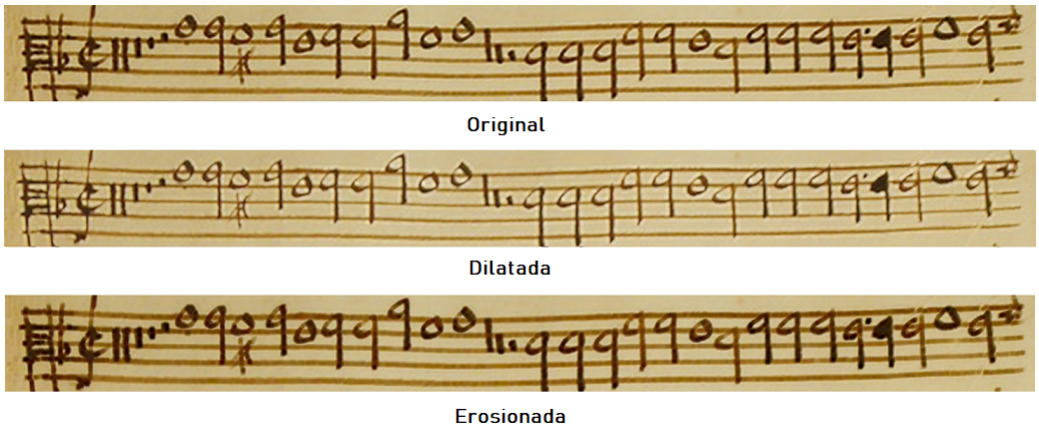


Figura 4.4: Erosión y dilatación

Los pasos para son los siguientes:

Extracción de pentagrama: El primer paso es recortar la imagen para quedarnos con la imagen del pentagrama, ya se le haya aplicado algún efecto o no. Se recorta previamente a la aplicación del efecto para ahorrar tiempo y cálculos innecesarios.

Distinción entre efectos: Para poder distinguir cuál de los dos efectos vamos a aplicar, se calcula un número aleatorio, comprendido entre un valor definido y el mismo en negativo, el cual se utiliza para tomar la decisión del filtro a aplicar dependiendo de si el valor obtenido es positivo o negativo y la magnitud de este para la intensidad del filtro.

Aplicación del filtro: Extraer la región comprendida entre las nuevas coordenadas calculadas.

4.3.4 Contraste

Ya que las imágenes que se quieren clasificar pueden provenir de cualquier tipo de cámara, las imágenes pueden tener una distribución de color diferente a pesar de que se tome la foto a la misma partitura. Es por esto por lo que se ha aplicado este efecto para replicar distintos grados de contraste. Para ello, se ha aplicado un contraste aleatorio dentro de un rango definido. Quedando imágenes como la de la Figura 4.5

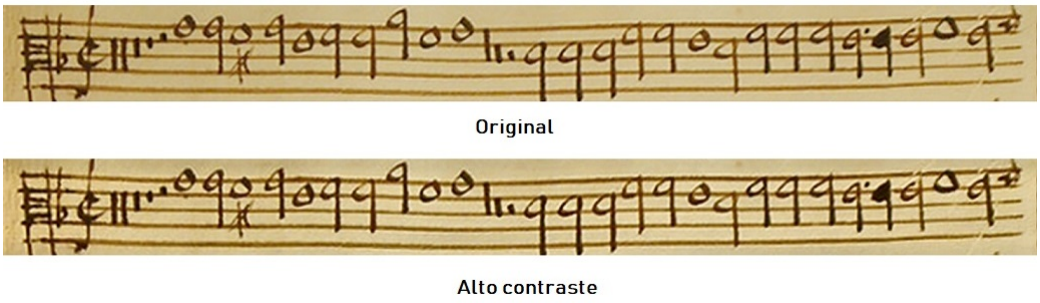


Figura 4.5: Contraste

Obteniendo resultados como el observado en la Figura 4.5 y viendo un ejemplo gráfico de la nueva distribución de los colores de la imagen en la Figura 4.6.

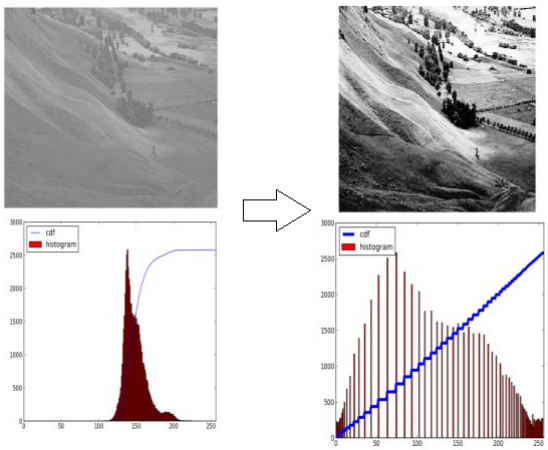


Figura 4.6: Histograma

4.3.5 Ojo de pez

Esta distorsión se ha aplicado para simular diferentes efectos, por una parte, distintos parámetros de calibración de distintas cámaras o incluso diferentes objetivos que pueden tener las mismas. Por otro lado, el efecto más interesante que se intenta imitar con este efecto es la forma en la que se quedan las páginas de un libro cuando lo abres por la mitad por lo cual las páginas no se quedan completamente rectas, planas sobre la mesa.

Para esto se ha tratado de aplicar un efecto de ojo de pez, el cual aplica una matriz de distorsión a la imagen la cual se calibra con la anchura de la imagen del pentagrama.

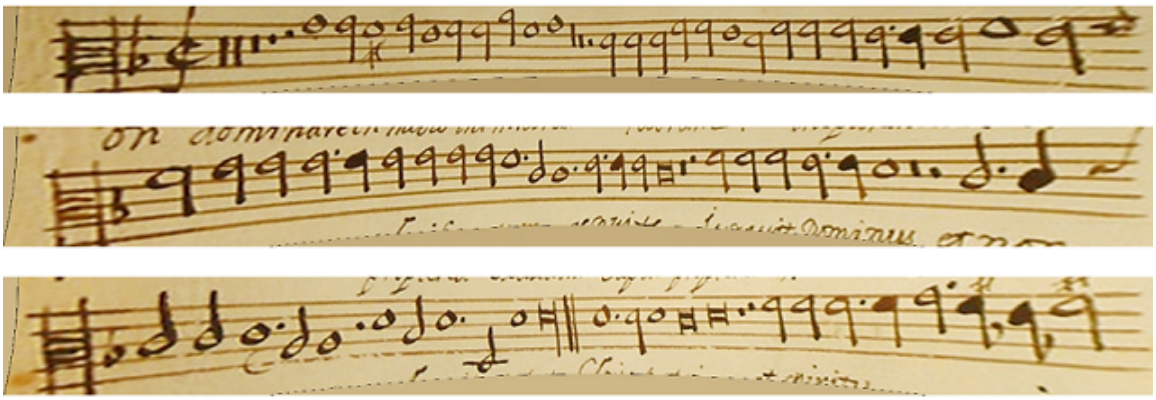


Figura 4.7: Efecto ojo de pez

Tras aplicar este efecto se quedan zonas en negro, las cuales se rellenan en un proceso posterior con el valor de color medio de toda la imagen antes de aplicar este efecto. Ejemplos de este efecto son los pentagramas que aparecen en la Figura 4.7.

4.4 Detalles de las modificaciones

4.4.1 Combinación de transformaciones

Para poder realizar diversas modificaciones a un mismo pentagrama, se ha seguido una estructura la cual permite realizar cualquier combinación de modificaciones con los parámetros deseados para cada una de ellas reduciendo el coste computacional que esto conlleva.

1. Se aplica padding a las imágenes, consiguiendo que los márgenes de la imagen se aumenten.

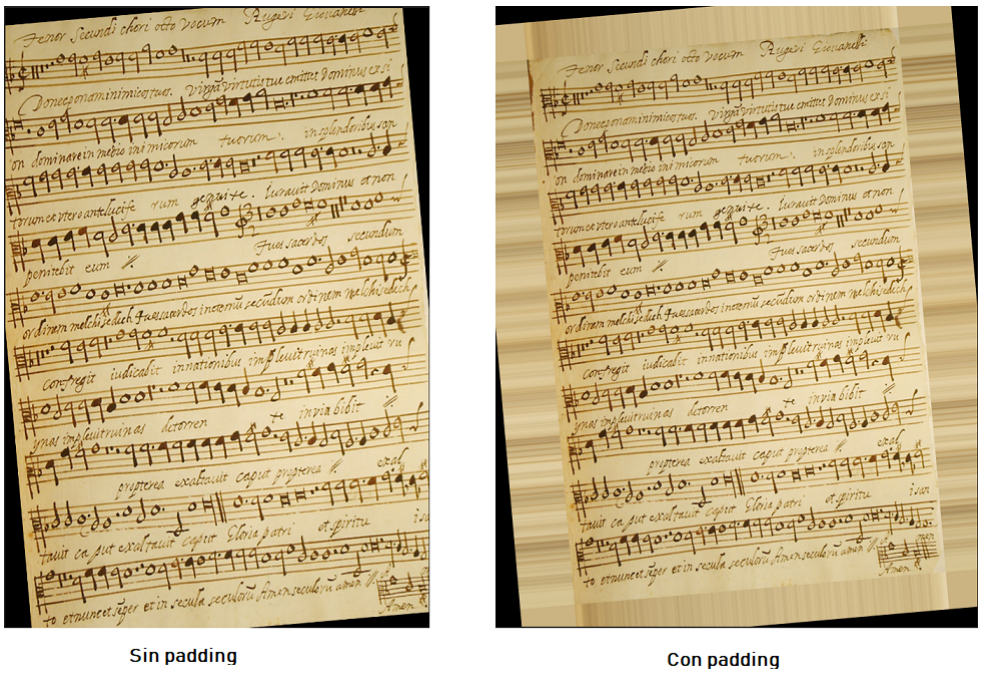


Figura 4.8: Aplicación de padding

Este es un proceso previo el cual se ha tenido que realizar ya que a pesar de contar con la imagen completa a la que se le puede aplicar ciertas modificaciones como rotación y la ampliación y disminución de los márgenes de las boinding box de los pentagramas, los pentagramas situados en la parte superior, inferior o que se ciñan demasiado a los bordes, pueden provocar que en el momento en el que se recorta el pentagrama de dicha imagen, haya zonas las cuales se hayan quedado sin información sobre la imagen original por lo que se rellenen con ceros.

Para solventar esta introducción de ceros se ha optado por incrementar el tamaño de la imagen introduciendo un margen alrededor de la misma el cual está formado con la media de los colores de cada una de las filas para ampliar en el eje de las x y la media de los colores en cada columna para el eje de las y como se puede apreciar en la Figura 4.8. Consiguiendo de esta manera prolongar la imagen con información más relevante que ampliarla con ceros.

2. Se rota tanto la imagen completa como las coordenadas de las cuatro esquinas del pentagrama a recortar y se ajustan las coordenadas para que formen un rectángulo.
3. Se modifica la posición de los bordes en las cuatro direcciones del bounding box
4. Se recorta la imagen con las coordenadas calculadas obteniendo el pentagrama rotado con los márgenes distorsionados.
5. Se le aplica un nivel de contraste variable según un valor aleatorio.
6. Se erosiona o dilata la imagen del pentagrama según el valor aleatorio calculado.
7. Se le aplica la transformación de ojo de pez.

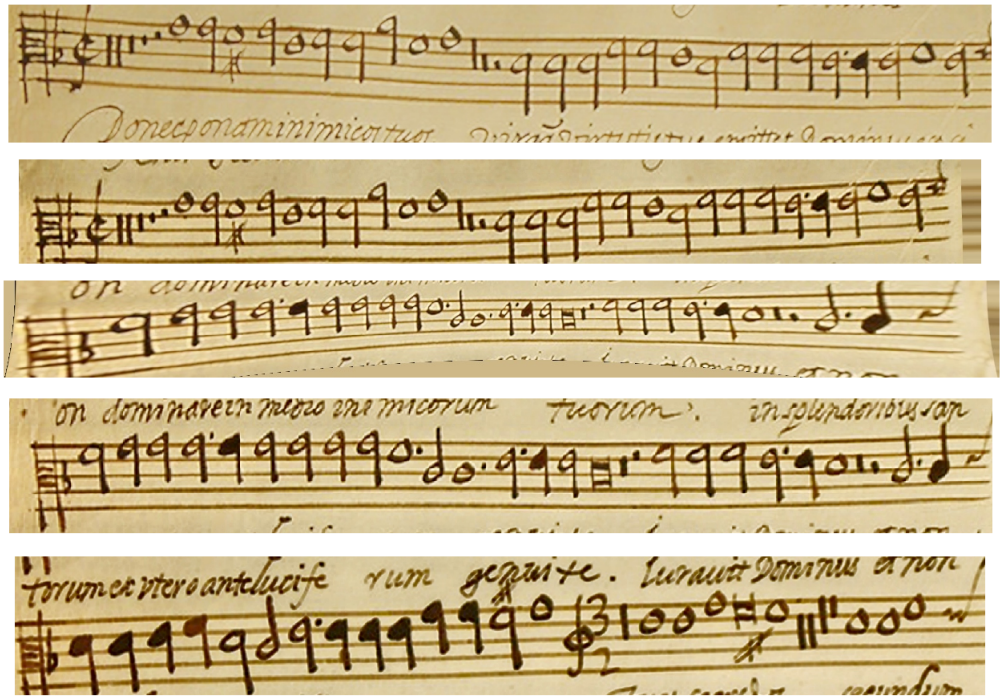


Figura 4.9: Aplicación de diversas modificaciones

Tras aplicar todas estas transformaciones de la manera que se ha descrito, algunos de las imágenes de los pentagramas resultantes se pueden visualizar en la Figura 4.9.

4.4.2 Números aleatorios

Para el cálculo de números aleatorios, en concreto para las modificaciones de rotación y margen, se ha optado por el uso de una función de numpy especial. Dicha función es random.normal. La finalidad de esta función es que no todos los números comprendidos entre los umbrales definidos tengan las mismas probabilidades de ser elegidos.

Los valores más probables son los más cercanos a 0. Una aproximación de este suceso se puede ver reflejado en la Figura 4.10

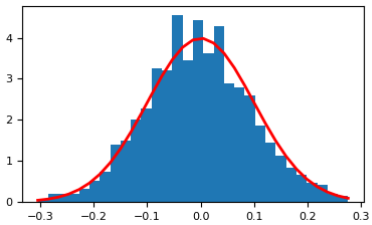


Figura 4.10: Números aleatorios

Esta técnica consigue que los casos más probables sean los que cuentan con una modificación más sutil, lo cual beneficia en el proceso de entrenamiento ya que, como se explica en el siguiente punto, se aplica una gran cantidad de transformaciones a cada una de las imágenes haciendo, con este método de cálculo de números aleatorios, que la red se entrene alguna vez con imágenes con una distorsión elevada, pero en la mayoría de las transformaciones se asemejen a lo que podemos calificar como normal.

4.5 Integración de las modificaciones en el algoritmo de entrenamiento

Este es uno de los puntos clave que ha mejorado drásticamente la tasa de acierto de las redes neuronales entrenada. Dichas modificaciones propias se suelen aplicar para aumentar el conjunto de datos de entrenamiento haciendo diversos tipos de modificación, incluso combinando varias de ellas, e incluyendo las imágenes modificadas al conjunto de datos de entrenamiento. De esta forma podemos conseguir que el volumen del conjunto de datos para el entrenamiento se multiplique por el número de veces que queramos aplicar las diferentes transformaciones que tengamos implementadas.

Esta es una técnica muy recurrida, pero a lo largo de todo el proceso de entrenamiento todas las imágenes del conjunto de entrenamiento, incluidas las generadas, permanecen estáticas haciendo que la red aprenda a clasificar estas imágenes las cuales puede que por la propia aleatoriedad el conjunto de imágenes extra generadas no le aporte una información relevante para generalizar el aprendizaje y con ello mejorar la tasa de acierto de la red.

Por este motivo se ha optado por aplicar las transformaciones de una manera especial. En este caso, estas transformaciones se generan en cada etapa del proceso de aprendizaje, consiguiendo así que la red tenga información de muchas más imágenes con distintas transformaciones sin la necesidad de aumentar el conjunto de datos. Consiguiendo de esta manera que la red aprenda con muchas más modificaciones que en el caso previamente comentado sin la necesidad de aumentar el conjunto de datos, por lo cual también se agiliza el proceso de entrenamiento.

5 Experimentación

5.1 Configuración de experimentos

Para llevar a cabo de manera efectiva toda la experimentación que ha llevado el trabajo a lo largo de todas las etapas del desarrollo, una de las primeras implementaciones fue la capacidad de controlar las modificaciones que se iba a aplicar en cada uno de los experimentos con un fichero externo de configuración. Este fichero se le proporciona al código de entrenamiento, permitiendo que se cargue al inicio de la prueba controlando los distintos tipos de transformación, junto con los parámetros de estas, que van a ser aplicadas a los conjuntos de entrenamiento en cada uno de los experimentos durante el proceso de aprendizaje.

Esto, a de permitir hacer experimentos con cualquier configuración de una manera cómoda, permite que el código sea escalable a la inclusión de nuevos tipos de modificación ya que simplemente sería necesario añadir campos extra en este fichero.

Gracias a esto se pudo hacer pruebas en cada una de las partes del desarrollo de las transformaciones siendo de gran utilidad para prevenir fallos tanto en la aplicación de transformaciones como en el propio algoritmo de entrenamiento.

Un ejemplo de archivo de configuración es el siguiente:

Código 5.1: Ejemplo fichero JSON de configuración para los experimentos

```
{
    "rotation_rank" : 3,
    "random_margin" : 10,
    "erosion_dilation" : flase,
    "contrast" : true,
    "fish_eye" : false
}
```

Aquí se puede apreciar los distintos parámetros que va a cargar el algoritmo de aprendizaje donde el experimento a realizar es una prueba de entrenamiento aplicando contraste a las imágenes, una rotación máxima de 3 grados y la posibilidad de expandir o reducir el bounding box del pentagrama en hasta un 10% de la altura del mismo.

5.2 Métricas

Para evaluar la calidad de un modelo entrenado, existen diversas métricas que ayudan a comprender si un modelo funciona de manera correcta o no. Para la evaluación de modelos, las métricas más conocidas son Accuracy, Binary Accuracy y Categorical Accuracy.

5.2.1 Accuracy

Esta métrica calcula como de precisa ha sido la salida comparando cada uno de los elementos de la salida esperada con los correspondientes valores en la salida obtenida. Se cuentan cuantos de estos valores coinciden y se normaliza con el tamaño de la salida de la red. Un ejemplo sería el siguiente:

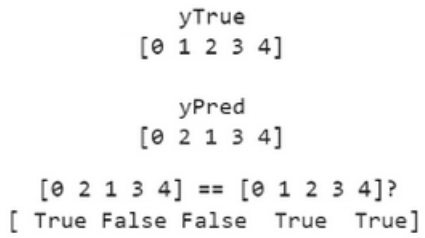


Figura 5.1: Calculo de accuracy

De esta manera, ya que la salida de la red cuenta con 5 valores, el accuracy en este caso es del 60%.

5.2.2 Binary Accuracy

La binnary accuracy se utiliza como métrica cuando se quiere evaluar un modelo con el cual se espera una salida en la que los valores tiendan a ser 0 o 1. Para ello se define un threshold el cual funciona como filtro, contando como 1 los valores obtenidos por la salida de la red que está por encima del mismo y como 0 los que están por debajo. Un ejemplo con un theshold de 0.5 es el siguiente:

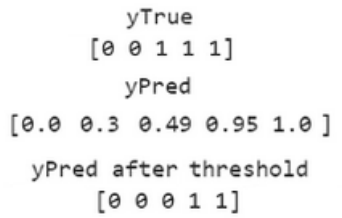


Figura 5.2: Calculo de binary accuracy

Como podemos apreciar, tras aplicar el threshold coinciden 4 de 5 valores entre la salida esperada y la obtenida, por lo que, en este caso el valor de la métrica binnary accuracy sería de 80%.

5.2.3 Categorical Accuracy

Categorical Accuracy es una de las métricas más utilizadas para comprender la calidad de un modelo entrenado para clasificar imágenes. El cálculo de dicha métrica se basa en vectores de etiquetas one-hot, esta técnica se basa en vectores donde cada una de las componentes del mismo hace referencia a una categoría, haciendo así que la salida esperada de una red esté formada por vectores que estén completamente rellenos con 0s menos la posición que hace referencia a la clase a la que pertenece, posición en la cual hay un 1.

De esta manera, se puede comparar si la predicción es correcta, comparando la posición del valor más elevado en la salida de la red y la posición del valor en la salida esperada. Se puede ver un ejemplo de este proceso en la siguiente imagen:

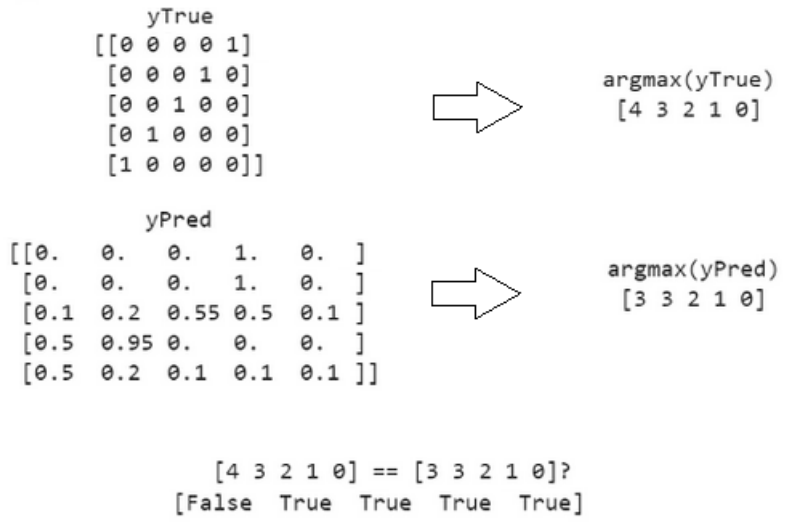


Figura 5.3: Calculo de categorical accuracy

Como podemos apreciar, para obtener los índices que posteriormente se comparan, se utiliza la función argmax. Dicha función es la encargada de proporcionar el índice del valor más elevado que encuentre en el vector.

De esta manera podemos ver que en este caso son correctos cuatro de las cinco categorías a evaluar por cada vector, por lo cual el resultado de esta métrica es del 80% de acierto.

5.2.4 SER

Las métricas comentadas se pueden aplicar en problemas donde la salida esperada tiene una determinada forma, pero no es el caso de este problema ya que la salida esperada y la salida obtenida de la red pueden no tener la misma forma.

Ya que el algoritmo va reconociendo símbolos por partes del pentagrama, la salida es una lista ordenada con los símbolos que ha reconocido de izquierda a derecha. Por este motivo, los fallos que puede cometer el modelo en la clasificación pueden ser detectar símbolos donde no los hay, no reconocerlos en partes en los que sí que existen o equivocarse en la clasificación de un símbolo detectado.

Es por este motivo por el cual se diseñó la métrica Symbol Error Rate (SER). Esta métrica se basa en el cálculo de la distancia de Levenshtein, el cual da como resultado el número mínimo de operaciones necesarias para convertir una cadena en otra. Estas operaciones son inserción, borrado y sustitución, las cuales encajan perfectamente con los errores que puede cometer el modelo en el momento en el cual se hace una predicción.

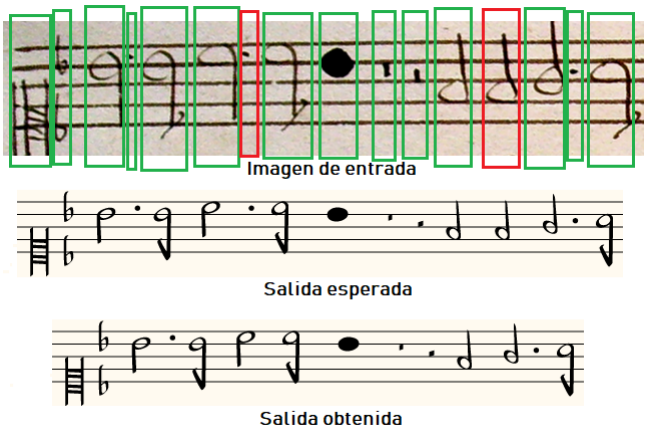


Figura 5.4: Clasificación errónea

Esta métrica nos permite evaluar de una manera más precisa la fiabilidad que tiene un modelo de clasificar los distintos símbolos en un pentagrama. En el caso de la Figura 5.4 el modelo se ha saltado un par de símbolos en la clasificación, pero el resto de símbolos están clasificados de manera correcta, por esto la distancia de edición en este caso valdría 2 ya que solo se precisa de borrar un par de símbolos para transformar la salida obtenida a la salida esperada.

Y, por último, esta distancia de edición se contrasta con el número máximo de símbolos generando un valor de error proporcional a la longitud de la cadena, al número de símbolos en el pentagrama que se quiere reconocer. Quedando en este caso un SER de 12'5%, tomando como un mejor modelo el cual cuenta con los menores valores de SER.

5.3 Conjuntos de datos

Para comprobar que el aumentado de datos que se le va a aplicar al conjunto da datos de entrenamiento en el proceso de aprendizaje, se ha probado con dos tipos de notación musical.

El primer conjunto, el conjunto SEILS está formado por partituras las cuales han sido escritas a máquina por lo cual todos los símbolos están escritos de manera uniforme lo cual permite que, sin la necesidad de aumentado de datos, la precisión de los modelos entrenados para clasificar este conjunto de símbolos tenga una precisión más elevada que los entrenados para clasificar símbolos escritos a mano. Podemos ver una página junto a un par de pentagramas de este conjunto en la Figura 5.5

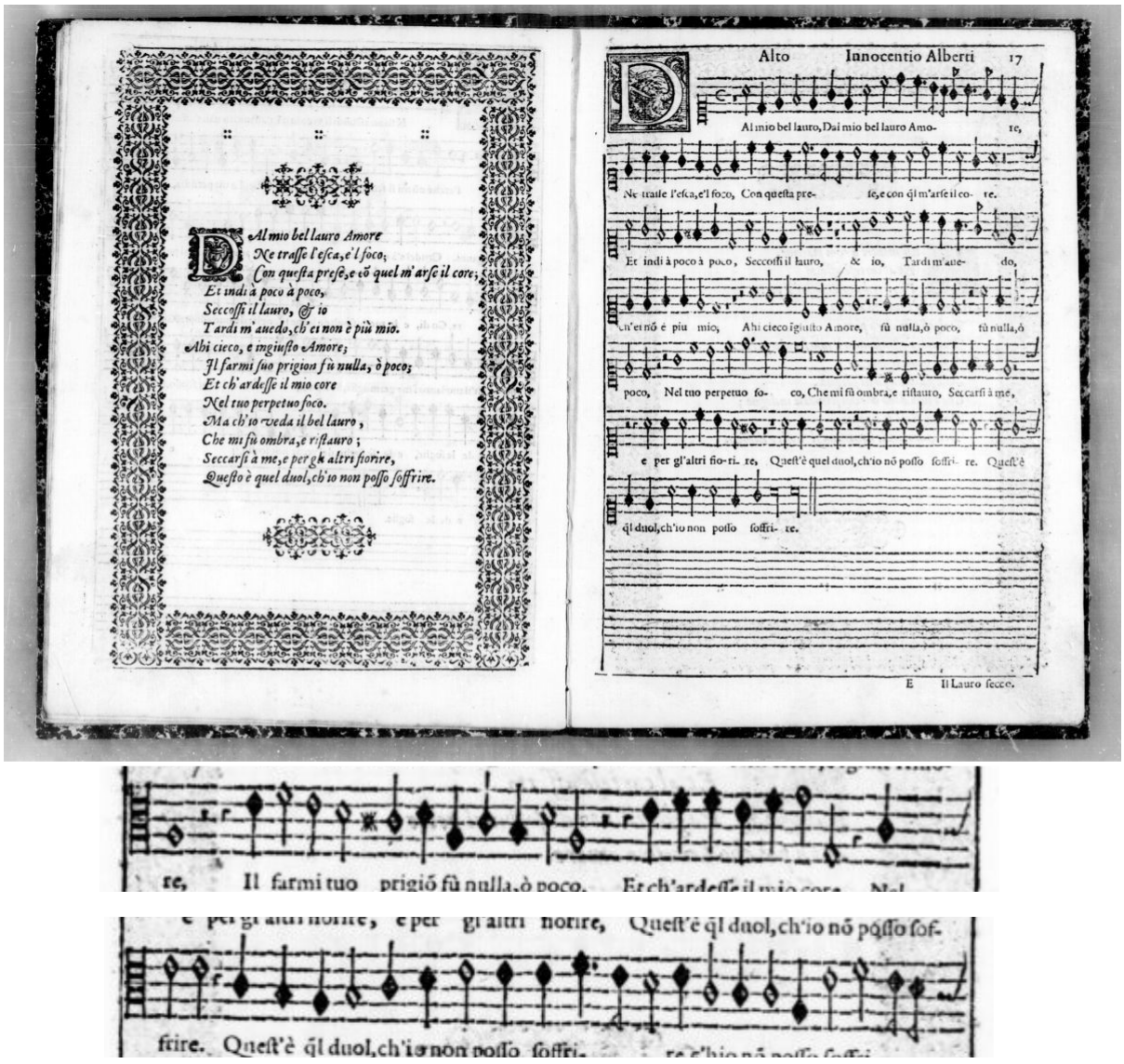


Figura 5.5: Conjunto SEILS

Por otro lado, se ha querido comprobar el funcionamiento de este aumentado con conjuntos de partituras las cuales han sido escritas a mano. Este tipo de clasificación es la más complicada ya que no ocurre como en el caso anterior, los símbolos en este caso se parecen entre los del mismo tipo, pero se han plasmado de la misma manera exacta lo cual añade dificultad al reconocimiento. Para hacer la comprobación del impacto del aumentado de datos en este tipo de pentagramas, se ha utilizado el conjunto de datos CAPITAN, el cual cuenta con partituras escritas a mano como las de la Figura 5.6.

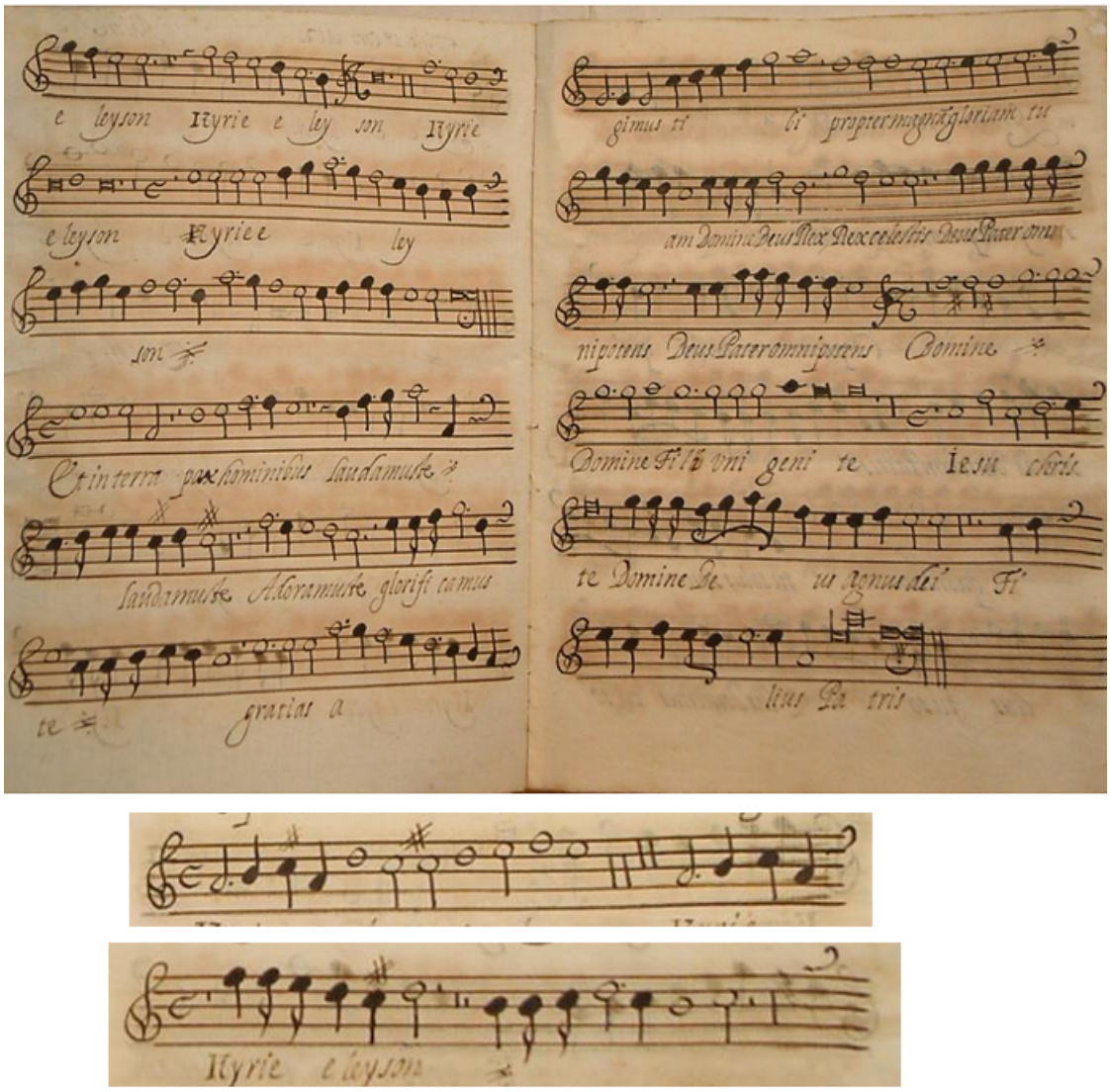


Figura 5.6: Conjunto CAPITAN

5.4 Experimentos

Para realizar las pruebas necesarias para saber si el aumentado de datos que se genera tiene un impacto positivo se ha evaluado cada una de las distintas modificaciones sobre dos conjuntos de datos diferentes.

Para ello se han generado los archivos de configuración necesarios para dichos experimentos con el formato que se ha explicado en el apartado anterior.

También se ha agregado un experimento en el cual se aplican todos los filtros implementados para observar el impacto final con la aplicación de todos ellos.

Contando con dos modelos entrenados, no podemos distinguir cuál de los dos es mejor mirando la métrica de cada uno de ellos y quedándonos con el mejor porque no es representativa ya que al partir el conjunto de datos en entrenamiento y validación, dicha partición puede ser favorable para uno de los dos experimentos el cual obtendría un resultado más favorable a pesar de que con cualquier otra partición diera peores resultados.

Para evitar estos errores al comparar dos modelos, se aplican varios folds. Esta técnica consiste en hacer varias particiones del conjunto de datos y entrenar con cada una de estas particiones. Se puede ver un ejemplo de dichas particiones en la Figura 5.7.

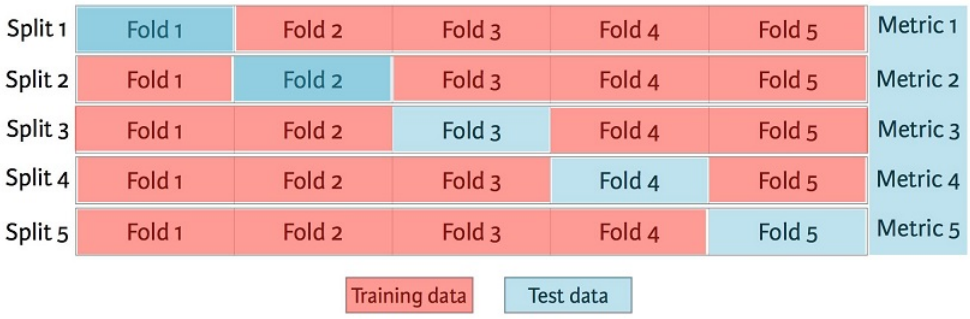


Figura 5.7: Particiones en 5 folds

Una vez que se obtienen las métricas de un mismo modelo para cada una de las particiones, se obtiene la media de dichas métricas junto con el error. Y es con este valor con el cual se pueden comparar distintos modelos.

Para estos experimentos se han aplicado, al igual que en la Figura 5.7, 5 particiones diferentes en los distintos conjuntos de datos. Quedando una distribución del 80% del conjunto de datos destinado al entrenamiento del modelo y el 20% restante para la validación del mismo.

5.5 Resultados

5.5.1 SEILS

En la tabla 5.1 podemos observar los resultados de los experimentos llevados a cabo con el conjunto SEILS en el cual las partituras están escritas a máquina.

Tipo de aumentado	avg +- std	max	min	nº folds
Sin modificación	4.31 +- 1.09	5.32	2.93	5
Contraste	4.01 +- 1.13	5.04	2.52	5
Erosión y dilatación	4.59 +- 1.22	5.94	2.98	5
Ojo de pez	4.72 +- 2.03	8.07	2.36	5
Margen	**2.92 +- 0.93**	**3.85**	**1.59**	**5**
Rotación	4.10 +- 1.06	5.07	2.78	5
Todas las modificaciones	3.47 +- 1.05	4.87	2.19	5

Tabla 5.1: Comparativa de los resultados en el corpus SEILS sin aplicar aumentado de datos frente a la aplicación de las distintas transformaciones

Podemos observar que la mayor mejora que se ha obtenido es aplicar una modificación en los márgenes de las bounding box. Con esta modificación se ha conseguido porcentaje de error más bajo de entre todos los experimentos realizados, el menor SER.

Otro dato a tener en cuenta es va variabilidad del error obtenido aplicando un efecto de ojo de pez a las imágenes. Era un resultado esperable tratando de partituras escritas a mano ya que, al aplicar esta transformación, la forma de los símbolos con los que se entrena la red neuronal se ven alterados cuando los símbolos escritos a máquina tienen una forma determinada y normalmente no hay variaciones. Esto provoca una introducción de error en el conjunto de entrenamiento que repercute en la precisión del modelo.

Por otro lado, no vemos cambios notorios en la aplicación del resto de efectos respecto al modelo entrenado sin modificaciones. Pese a que podemos apreciar una sutil mejora al aplicar efectos de contraste y de rotación, y un pequeño aumento del error aplicando erosión y dilatación, esos pequeños cambios de precisión no son lo suficientemente elevados para tenerlos en cuenta.

5.5.2 CAPITAN

En la tabla 5.2 podemos observar los resultados de los experimentos llevados a cabo con el conjunto CAPITAN en el cual las partituras están escritas a mano.

Tipo de aumentado	avg +- std	max	min	nº folds
Sin modificación	12.78 +- 1.28	14.41	11.16	5
Contraste	12.03 +- 0.47	12.69	11.38	5
Erosión y dilatación	13.48 +- 3.12	19.65	11.28	5
Ojo de pez	13.04 +- 1.40	15.36	11.19	5
Margen	**9.80 +- 0.82**	**10.69**	**8.43**	**5**
Rotación	12.56 +- 0.90	14.02	11.24	5
Todas las modificaciones	10.85 +- 1.12	12.93	9.56	5

Tabla 5.2: Comparativa de los resultados en el corpus CAPITAN sin aplicar aumentado de datos frente a la aplicación de las distintas transformaciones

En este caso tenemos un panorama parecido al experimento anterior, ya que el efecto más destacado es la modificación del tamaño de las bounding boxes.

Esto ha llevado a pensar en las partes en común de ambos conjuntos. Ambos tienen en común el proceso de etiquetación en el cual se identifica cada una de las boinding boxes de los pentagramas y la lista de símbolos que contienen cada pentagrama. Dicho proceso es manual por lo cual parece bastante intuitivo que aplicar este tipo de modificación, consiga que el modelo sea robusto ante distintas bounnding boxes definidas alrededor de cada pentagrama.

También se puede observar que el efecto de erosión y dilatación ha incrementado la variabilidad del modelo resultante con lo que se puede interpretar que este efecto empeora la precisión. Y el resto de las modificaciones no difieren demasiado con los resultados obtenidos sin ningún tipo de modificación.

Con estos resultados podemos sacar en claro que el efecto con el cual se consigue una mayor precisión en los modelos de entre los efectos que se ha investigado es modificando la forma de las bounding boxes.

6 Conclusiones

En un primer momento, ya se pensaba que esta idea de generar un aumentado de datos de esta forma tan característica podría dar buenos resultados teniendo en cuenta los conjuntos de datos con la que se contaba. De manera que se consiguiera bajar las tasas de SER obtenidas al entrenar nuevos modelos aplicando este aumentado.

Como se ha podido observar con los resultados obtenidos con los experimentos, no todas las modificaciones han tenido el impacto esperado. Sin embargo, los resultados obtenidos han sido positivos ya que hemos podido observar cómo aplicando una de las modificaciones implementadas ha tenido un gran impacto positivo en la precisión de los modelos.

Es por esto por lo que este aumentado de datos se está aplicando en los modelos investigados en el Departamento de Lenguajes y Sistemas Informáticos de la UA - GRFIA. Donde el mejor modelo con el que se cuenta, ha sido entrenado con el aumentado de datos desarrollado.

Dejando de lado los resultados, este trabajo ha resultado muy enriquecedor ya que se ha conseguido adquirir grandes conocimientos gracias a la investigación de técnicas de aumentado de datos y de nuevas estructuras de redes neuronales no estudiadas durante la carrera. Sin obviar la experiencia de realizar un trabajo de investigación experimentando el trabajo que conlleva, los fallos imprevistos y la obtención de grandes conocimientos más allá de lo que podía aparentar el trabajo en los inicios de este.

Este trabajo ha sido una gran introducción personal en la investigación en el campo del OMR donde se ha conseguido mejorar un modelo ya existente gracias al aumentado de datos particular para este problema.

Todavía quedan muchas técnicas que perfeccionar, técnicas que probar y modificaciones que testear. Se ha sido un gran avance pudiendo mejorar la precisión de los modelos con los que se contaba, pero se puede seguir investigando nuevas técnicas que sigan mejorando los modelos.

También gracias a la estructura diseñada en este trabajo, se pueden seguir investigando nuevas transformaciones y filtros de manera sencilla para poder conseguir modelos con un conocimiento más genérico sobre los conjuntos de datos proporcionados tratando de refinar la precisión de los modelos.

Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Descargado de `https://www.tensorflow.org/` (Software available from tensorflow.org)

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*

Calvo-Zaragoza, J., y Rizo, D. (2018, 04). End-to-end neural optical music recognition of monophonic scores. *Applied Sciences*, *8*, 606. doi: 10.3390/app8040606

Chan, W., Jaitly, N., Le, Q. V., y Vinyals, O. (2015). Listen, attend and spell. *arXiv preprint arXiv:1508.01211.*

Graves, A., Fernández, S., Gomez, F., y Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. En *Proceedings of the 23rd international conference on machine learning* (pp. 369–376).

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., y Wierstra, D. (2015). Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623.*

Hannun, A. (2017). Sequence modeling with ctc. *Distill.* (https://distill.pub/2017/ctc) doi: 10.23915/distill.00008

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. doi: 10.1109/MCSE.2007.55

LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lee, J., y Tashev, I. (2015). High-level feature representation using recurrent neural network for speech emotion recognition. En *Sixteenth annual conference of the international speech communication association.*

Oliphant, T. E. (2006). *A guide to numpy* (Vol. 1). Trelgol Publishing USA.

Pal, S. K., y Mitra, S. (1992). Multilayer perceptron, fuzzy sets, classifiaction.

Phan, H., Hertel, L., Maass, M., y Mertins, A. (2016). Robust audio event recognition with 1-max pooling convolutional neural networks. *arXiv preprint arXiv:1604.06338.*

Sermanet, P., Chintala, S., y LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. En *Proceedings of the 21st international conference on pattern recognition (icpr2012)* (pp. 3288–3291).

Stephen, I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, *50*(2), 179.

Tang, D., Qin, B., y Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. En *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1422–1432).

Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, *5*, 23.