INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




A deep learning approach to monitoring workers stress at office

JACQUELINE MARCHETTI
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

A deep learning approach to monitoring workers' stress at the office

Jacqueline Marchetti

A dissertation submitted for obtaining the Master's Degree in Computer Engineering, Area of Specialization in Information and Knowledge Systems Computer Systems

Supervisor: Doutora Fátima Rodrigues

Porto, October 13, 2022

Dedicatory

I dedicate this thesis to my parents, Cleide and René, who supported and motivated me even in the toughest moments. To my late grandfather, Carlos Marchetti, who inspired my pursuit in science.

To my supervisor, Fatima Rodrigues, for her commitment and support which contributed in no small part to the project's success.

Finally, I express my sincere gratitude to who has crossed my path and helped me to become what I am today.

"All our dreams can come true if we have the courage to pursue them." - Walt Disney

Abstract

Identifying stress in people is not a trivial or straightforward task, as several factors are involved in detecting the presence or absence of stress. Since there are few tools on the market that companies can use, new models have been created and developed that can be used to detect stress.

In this study, we propose developing a stress detection application using deep learning models to analyze images obtained in the workplace. It will provide information from these analyses to the company so they can use it for occupational health management.

The proposed solution uses deep learning algorithms to create prediction models and analyze images. The new non-invasive application is designed to help detect stress and educate people to control their health conditions. The model trained achieved an F1=79.9% with a binary dataset of stress/non-stress that have an imbalanced ratio of 0.49.

Keywords: Stress detection, Deep Learning, Convolutional Neural Network, Workplace, Application, TensorFlow

Resumo

Identificar o estresse nas pessoas não é uma tarefa trivial ou simples, pois vários fatores estão envolvidos na detecção da presença ou ausência de estresse. Como existem poucas ferramentas no mercado que as empresas podem utilizar, foram criados e desenvolvidos novos modelos que podem ser utilizados para detectar o estresse.

Neste estudo, propomos desenvolver um aplicativo de detecção de estresse usando modelos de aprendizado profundo para analisar imagens obtidas no local de trabalho. Ele fornecerá informações dessas análises para a empresa para que possa utilizá-las para a gestão da saúde ocupacional.

A solução proposta usa algoritmos de aprendizado profundo para criar modelos de previsão e analisar imagens. O novo aplicativo não invasivo foi projetado para ajudar a detectar o estresse e educar as pessoas para controlar suas condições de saúde. O modelo treinado alcançou um F1=79,9% com um conjunto de dados binários de estresse/não estresse que continha um ratio de desbalanceamento de 0.49.

Contents

List of Figures

List of Tables

List of Source Code

Chapter 1

Introduction

This chapter provides a context for the problem studied and discusses the reasons that motivated the present dissertation, its purpose, expected contributions, the methodology adopted, and the document's organization.

1.1 Context

For many people, work is the central feature that structures their lives and identities. Quality and meaningful work can positively impact mental health, well-being, and community inclusion. As a negative working environment may lead to physical and mental health problems, substance use disorders, absenteeism, and loss of productivity, there is a growing recognition, across the European Union and globally, of mental health problems' economic and social impact (OSHWiki 2020).

The World Health Organisation (WHO) suggests that work-related stress is the response people may have when presented with work demands and pressures that do not match their knowledge and abilities and which challenge their ability to cope. It is caused by poor work organization, poor work design, poor management, unsatisfactory working conditions, and lack of support from colleagues and supervisors (Organization et al. 2020).

The disability caused by stress is just as significant as the disability caused by workplace accidents or other common medical conditions, such as hypertension, diabetes, and arthritis (Kalia 2002). In Europe, researchers estimate that 25% of European citizens will experience a mental health problem in their lifetime, and approximately 10% of long-term health problems and disabilities can be related to mental and emotional disorders (OSHWiki 2020). Studies also suggest that stress is a factor between 50% and 60% of all lost working days. Moreover, according to the survey 'The Workforce View in Europe 2018', 18% of European workers say that they endure stress at work every day, which is 5% higher than the 2017 results (Polska 2018).

Thus, the contemporary management of stress at work must encompass the design of a more straightforward organizational structure and transparent practices, and selection and training must be more appropriate to the reality of the work. Communication should emphasize the resolution of problems so that teams' social environment can promote conditions to avoid stress in the workplace. Therefore, identifying the potential sources of stress for an organization's employees is the first step in dealing with this problem.

1.2 Problem

Demanding jobs are a significant cause of stress for people. There is evidence that stressful conditions are preventable and treatable in the workplace, and workers who receive treatment are more likely to be more productive (Almeida and Rodrigues 2021). Hence, non-intrusive stress sensing tools that continuously monitor stress levels, with a minimal impact on workers' daily lives, could be used to initiate stress-reduction interventions automatically.

1.3 Purpose

The aim of this dissertation is to provide an alternative to detect stress in the workplace through the analysis of images collected in a non-invasive way in order to create models that can detect stress using deep learning techniques.

1.4 Contributions

The contributions and results expected by the realization of this thesis can be synthesized with the following artifacts:

- Data transformation: organization of data files and folders from a public experiment into a format that can be used for machine learning and deep learning analysis;
- Stress detection model: the development of a case study, where a stress detection model is created using deep learning techniques which explore the processes behind parametrizing a deep learning architecture and the usage of a public stress detection experiment dataset;
- Model Evaluation: the study is comparing precision, accuracy, and recall for the architecture models defined previously.

This dissertation has also contributed to the student's personal and professional development by providing a chance to explore deep learning techniques and exploring alternatives to overtake the challenges of stress detection in the workplace.

1.5 Methodology

Published in 2000 (Chapman et al. 2000) to standardize data mining processes across industries, the CRoss Industry Standard Process for Data Mining, CRISP-DM (Rüdiger Wirth and Hipp 2000), is a process model that provides a framework for conducting data mining projects independently of the industry sector and technology used. In this framework, any software can either be used for analysis or be applied to data mining problems. This methodology has six phases that naturally describe the data science life cycle. Whatever the nature of the data mining project, CRISP-DM will provide sufficient structure for the project (Rogalewicz and Sika 2016).

In this project, the six phases of CRISP-DM can be described as follows:

- Business Understanding: studies performed on physiological markers used in stress detection;

- Data Understanding: selection of a public data source that uses non-invasive ways to detect stress;
- Data preparation: data quality assessment of the public database information that is consumed in the construction of the models;
- Modeling: the definition of a deep learning architecture to be used to create the model for stress detection;
- Evaluation: statistical analysis of the performance of the model in stress detection;
- Deployment: publish the stress detection model on a web page so that it can be reused by other people.

1.6 Document Organization

This document is organized into seven chapters which contain several sections, subsections, and, at times, sub-sub-sections that provide an incremental specification for the theme discussed in each. The chapters are as follows:

Introduction: This chapter provides a context overview of stress in the workplace. Post contextualizing comes to the definition of the motivation for this thesis, and objectives for the solution are determined. Finally, it presents the methodology used for the development of the thesis.

Value Analysis: The value analysis of this development is presented succinctly, as well as the potential of this dissertation as a business idea and how it would fit into the market.

State of the Art: This chapter confines a descriptive presentation of deep learning, convolutional neural networks, and an overview of deep learning technologies.

Design: In this chapter, it is presented the information about the system view from a structural point of view, analyzing the intended features, the architecture, and alternatives to the architecture of the system.

Implementation: This chapter features most of the technical details through which the implementation of the project was possible and the methodology used in the development.

Experimentation and Evaluation: In this chapter, it is presented the experiments that were conducted as the analysis of the performance of the models built accordingly to the considered performance metrics.

Conclusions: In this chapter, a summary of the dissertation is made, presenting what conclusions can be made and what should be the next steps.

Chapter 2

Value Analysis

In this chapter, an analysis of the solution proposed in this project will be carried out. In this analysis, the five key elements of the NCD model (New concept development model) are identified, the value proposition for the customer will be presented, and finally, the Canvas business is developed.

2.1 The New Concept Development (NCD)

"The New Concept Development Model (NCD) provides a common language and definition of the key components of the Front End of Innovation" (Martikainen 2017). The model divides the front end into three distinct areas. The first is the engine, or center of the model, which accounts for the vision, strategy, and culture which drives the Front End of Innovation. The second or inner part defines the five activity elements of the front end. The third element consists of the external environmental factors. The model is represented in figure 2.1.

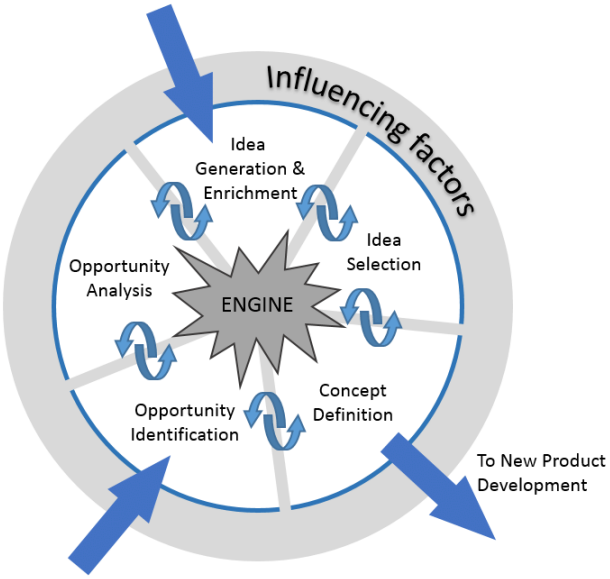


Figure 2.1: NCD Model. Retrieved from (Martikainen 2017)

2.1.1 Opportunity Identification

The opportunity analysis process starts with market analysis to identify the market needs to explore opportunities for new or different tools capable of adding value to the market.

The workplace stress management market expects to grow in the forecast period of 2020 to 2026. Recent research shows that the market accounts for USD 15.4 billion by 2026 with a CAGR (Compound Annual Growth Rate) of 8.7% in the forecast for the mentioned period (Factors 2022).

The rise in competition in the workplace and the increase in stress management awareness are the major factors driving the market growth. However, the lack of skilled counselors and limited awareness of workplace stress management in developing countries are expected to limit the growth of the workplace stress management market.

In addition, due to the spread of coronavirus disease (COVID-19) worldwide, the market in the early stage of the forecast period has undergone significant changes. The impact of COVID-19 has resulted in the reduction of employees in many organizations. For this reason, many companies have been forced to lay off employees to cut costs. These layoffs as part of cost-cutting measures during the COVID-19 pandemic have negatively impacted the workplace stress management market.

Workplace-related stress and anxiety have rapidly increased the level of disorder with negative consequences for physical health, mental well-being, workplace productivity, and career opportunities. The COVID-19 pandemic worsened the already widespread and largely unaddressed mental health workplace problems. Therefore, the factors mentioned earlier are expected to affect the market growth during the forecast period.

2.1.2 Opportunity Analysis

Opportunity analysis is a process that aims to study business opportunities to understand how the market will be able to understand and take advantage of the solution so that the solution design is profitable.

It is essential to detect and diagnose stress symptoms and conditions at early stages. Clinically, the methods which are in practice involve direct observation and interpretation by the examiner for stress assessment. These methodologies include questions related to the participant's life and the evaluator grades based on the answers. In this manner, interpretations are made regarding the stress status depending on the question. Besides these clinical practices, many different techniques have also been developed to recognize its patterns by physiologic signals and images.

The preliminary works for detecting stress have started in controlled laboratory environments. However, scientists realized that the stress level experienced in the laboratory is different compared to the stress in daily life (Picard 2016). Another important reason for this trend shift in stress recognition studies was that users do not want to be evaluated by invasive measurement techniques because they are uncomfortable. For these reasons, non-invasive stress measurements in daily life have become a trend for researchers.

This project wants to provide a non-invasive alternative to detect stress in the workplace to start addressing the problem in its early stages. With that information, it is also possible to provide more specific workplace stress management tools for employees to decrease occupational health problems in the long term.

2.1.3 Idea Generation and Enrichment

Idea generation is the creative process of generating new methods to solve problems and improve the product's conditions or the company itself. It is based on factors like idea development, group discussion, choosing the best alternative, and finally, implementing the idea in real-world scenarios. It has a direct relation to the opportunity identification phase and can usually be translated into a high-level view of the envisioned solution for the problem creating the opportunity.

Currently, there are several studies related to stress identification or stress in the workplace that covers medical, psychological, or technological points of view. From the technological perspective, there are other methodologies for data processing and analysis, using, for example, machine learning for pattern recognition. The main difference present in this dissertation is the use of a recently published stress experiment dataset.

In this phase, the main idea was how to explore the existing information in the dataset; here were the suggestions:

- use deep learning techniques to identify stress through image analysis;
- transpose physiological data to graphics and create models that predict stress from graphics;
- explore the dataset and learn which features related to stress can be derivated from the original information from the dataset;
- to present this service in an application with a user-friendly interface for occupational health technicians.

2.1.4 Idea Selection

After generating ideas, it is necessary to define and prioritize ideas; according to author Robert B. Tucker (Tucker 2008), a clear set of criteria for screening ideas and having properly trained people on the selection team is critical to systematic innovation. To transform an idea into the development of a solution, a transparent selection process is essential.

Thus, for the solution, it was defined that it would be an asset for the first phase to be able to identify stress with the highest possible rate of success and try to understand which physiological data are most relevant for that same diagnosis.

These ideas were selected based on technological risk, development costs, and the benefit they bring to the occupational health professionals. The remaining ideas will remain as future work to be carried out after this project.

2.1.5 Concept Definition

Concept development is one of the essential and critical stages in the development of a new product. After an idea is generated and screened for its quality, the concept development stage follows. The development of the concept is established by acknowledging the needs of the customers as well as the nature of the product.

The identification of stress is a sensitive area since stress is a normal human reaction that happens to everyone. It can be affected by numerous factors, known as stressors, and they are not easily identified, such as physical and emotional symptoms, like aches and pains,

chest pain or a feeling like the heart is racing, exhaustion or trouble sleeping, headaches, dizziness or shaking, high blood pressure, muscle tension or jaw clenching, anxiety or irritability, depression, panic attacks, sadness, and etc.

This development will be carried out using deep learning, applying different technologies for the processing and classification of data.

With this development, it will be possible to improve the identification of stress in the workplace and support occupational health technicians.

2.2 Value

This concept can be defined in many different ways, and according to theoretical contexts where it was used, it was defined as need attitudes and preferences (Nicola, E. P. Ferreira, and J. P. Ferreira 2012).

2.2.1 Business Model

Verna Allee's value network

Verna Allee's value network, shown in figure 2.2, is defined as any network of relationships that create tangible and intangible value through the exchange of individuals, groups, and organizations.

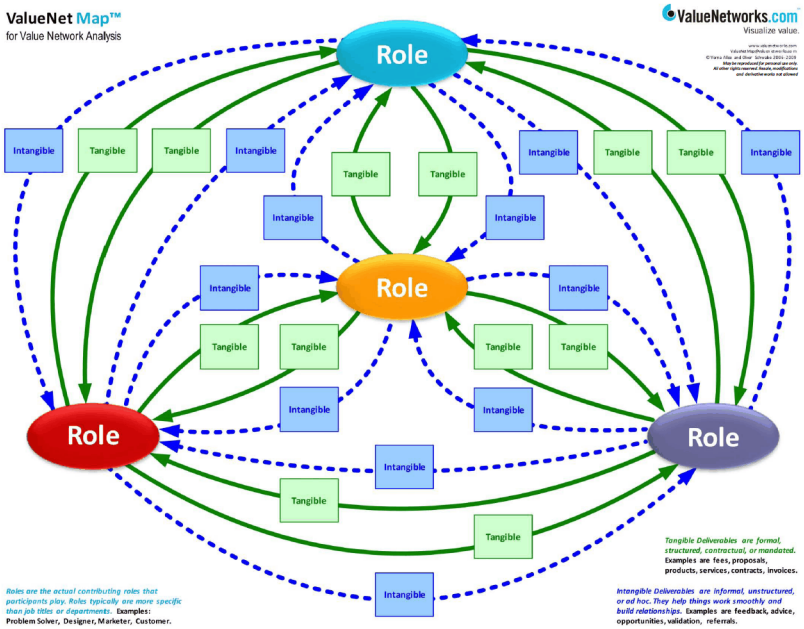


Figure 2.2: Verna Allee's value network (Allee 2006)

The tangible value can be interpreted as any interaction to which it is possible to assign a numerical value. Be it the sale of goods or services, payments, requests for proposals, invoices, etc. However, the intangible value is more directed to the human factor. It bases on interactions and exchange of information between stakeholders in the business process, who exchange knowledge and benefits. The purpose of value networks is to create the maximum benefit for the actors involved, from suppliers to producers and finally to customers. The

value network differs from the value chain in that it accepts the value produced by the interrelationships of the stakeholders, not only by the activities performed by them. While according to the value chain, the business process can be optimized only by changing the modus operandi of activities. In the value network, sharing information and knowledge also adds value to the process. If an employee does not know how to perform a task, it is more feasible to provide someone who can teach or offer training than to always fire and hire new employees until he finds one who knows how to perform all tasks (Allee 2002).

Porter's value chain

The idea of a value chain is based on the organization process overview, the systemic idea of an organization as a manufacturer or a service, made up of subsystems, each with inputs, transformation processes, and outputs. Inputs, transformation processes, and outputs involve acquiring and consuming resources like money, labor, materials, equipment, buildings, land, administration, and management.

Most organizations have hundreds, even thousands, of activities that convert inputs to output. Porter's value chain allows modeling a company's activities as a sequence by analyzing the links between activities and looking for ways to improve or optimize them. If in the analysis of the activities it is found that one of them is stopped due to another needing the same resource, then it can be concluded that an optimization possibility would be the availability of additional relevant resources.

Porter's value chain, as shown in Figure 2.3, is defined by the set of activities that a company/organization performs in order to distribute a product or service to the market.

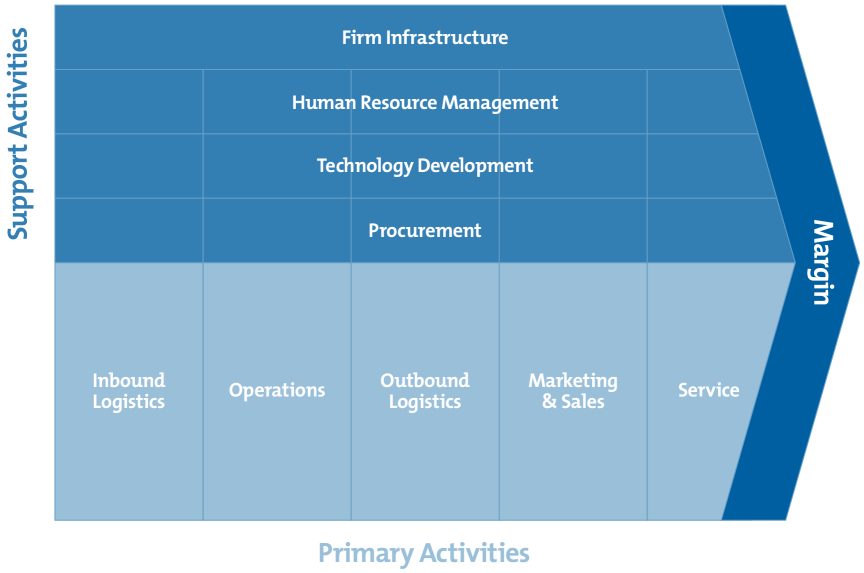


Figure 2.3: Porter's value chain (Michael et al. 1985)

Analysis

According to the two models presented, Verna Allee's model seems to be the one that best fits this project since the exchange of information and value are central aspects and one of the reasons why its implementation brings several advantages for all actors.

The relationships established with companies that hold a large amount of data from their workers would be the basis of success in a business linked to this project, as it would be through a web of influencing factors that it would be possible to discover the knowledge, more specifically, the correlation of patterns.

2.2.2 Value for the customer

This concept represents the advantage of personal association with the organizational offer. It can occur through the reduction of sacrifices, the existence of benefits, and the result obtained through the weighted combination of sacrifice and benefit (Woodall 2003). The customer perceives this balance before, during, and after purchasing the product or service (Nicola, E. P. Ferreira, and J. P. Ferreira 2012). The way a customer evaluates a product or service varies from customer to customer.

As already mentioned, the combination of the products offered is quantified in a balance between benefits and sacrifices for the customer before, during, and after the purchase of the product or service (Nicola, E. P. Ferreira, and J. P. Ferreira 2012). The way a customer evaluates a product or service varies from customer to customer. This variation in perceived value is so fragile that the perceived value of that product varies from a longitudinal perspective. The customer's perception of the value of the product changes during the acquisition and use process. A customer has an idea of the value of the item before buying it; it can vary at the time of purchase due to several factors, such as price or construction. The perception of value can also change after purchase and also after using the product or service.

Table 2.1 presents the longitudinal perspective of the value with the benefits and the sacrifices for the client in this context.

	Benefits	**Sacrifices**
Before Purchasing	-	Price
Transaction	-	Acquisition cost
After Purchasing	• Customization • Support	Learning time
Utilization	• Early detection of stress • Reliability	Effort to insert new data

Table 2.1: Longitudinal perspective of value

2.2.3 Perceived Value

Perceived value is a customer's evaluation of a product or a service's merit or desirability to them, especially in comparison to a competitor's product.

Prior research (R. Li and Liu 2020) has shown that analyzing physiological signals is a reliable predictor of stress. Such signals are collected from sensors that are attached to the human body. There have been attempts to detect stress by using traditional machine learning methods to analyze physiological signals. Results, ranging between 50 and 90% accuracy, have been mixed. A limitation of traditional machine learning algorithms is the requirement for hand-crafted features. Accuracy decreases if features are misidentified.

To address this deficiency, the development of deep neural network models does not require hand-crafted features but instead extracts features from raw data through the layers of the

neural networks. The deep neural networks can analyze not only physiological data collected from sensors but also images collected by cameras to perform the classification. This new non-invasive image analysis tool can be used to perform stress assessments in the workplace without supervision.

2.2.4 Value Proposition

Develop a deep learning model that helps classify images with stress with a high accuracy rate to ensure the reliability of the forecast.

In this way, we are providing users and occupational health technicians with a new tool that does not require the presence of a technician or specialist and can ensure the early detection of stress in the workplace.

With the increase in the amount of correctly cataloged people data, the increase in the volume of data trained in the deep learning model, and the constant update in the stress predictions over time, the result is an increasingly effective model to be used in stress detection.

In short, early detection of stress in the workplace can help occupational health technicians to prepare prevention plans so that employees can learn how to manage stress in the company, which translates into a significant improvement in the quality of life.

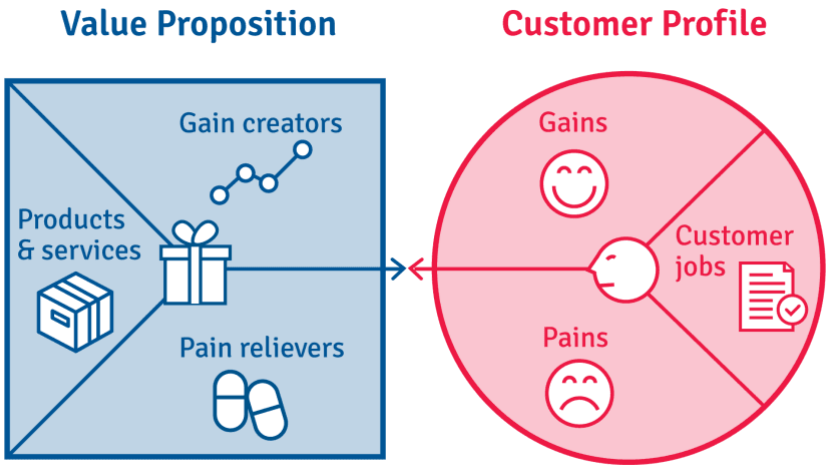


Figure 2.4: Value proposition canvas (Osterwalder et al. 2015)

2.3 Canvas Business Model

Alex Osterwalder invented the canvas as a visual chart with various elements that describe an organization's value proposition, infrastructure, markets, and finances.

The analysis of the value of a given product allows us to identify the reason why the market should accept the product that we are analyzing at the expense of a possible competitor. In this sense, the canvas business, presented in Figure 2.5 is a great help to demonstrate a value proposal, describing the strategic plan in a quick, agile, and elucidative way.

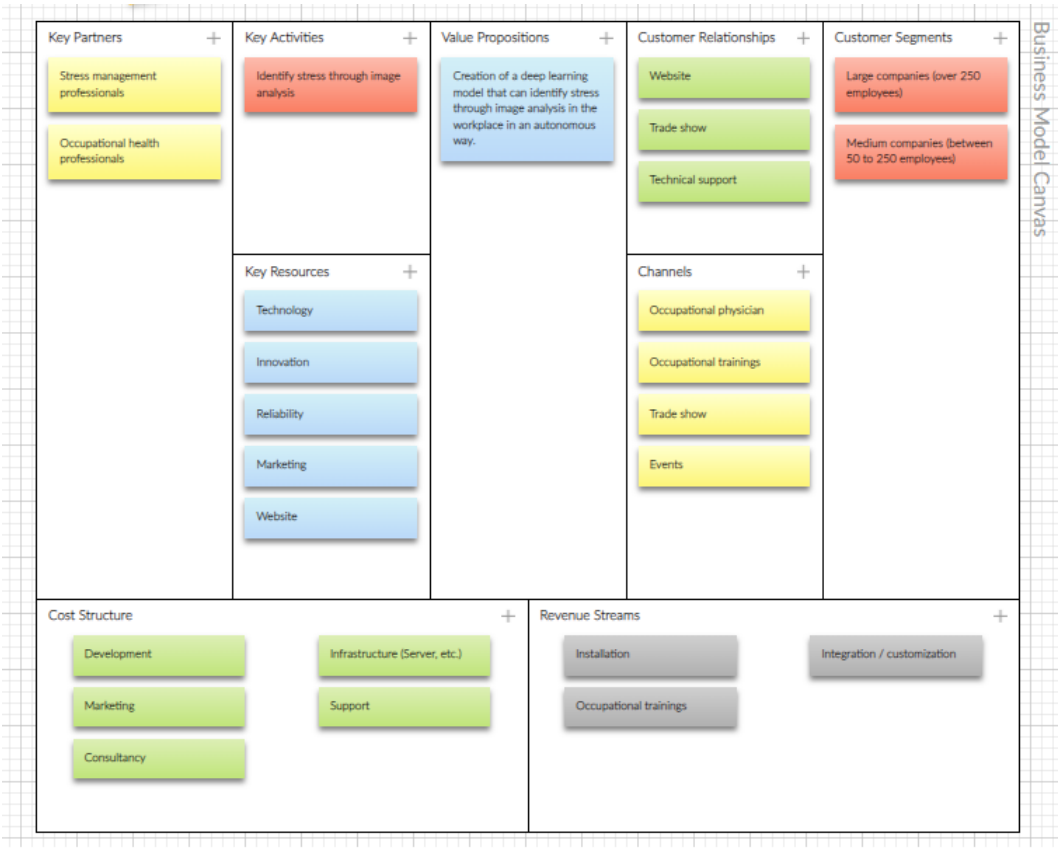


Figure 2.5: Canvas Model

2.4 AHP Method

The AHP (Analytic Hierarchy Process) method, initially proposed by Saaty is one of the multicriteria decision methods found in the literature that creates a hierarchy for the decision divided into levels.

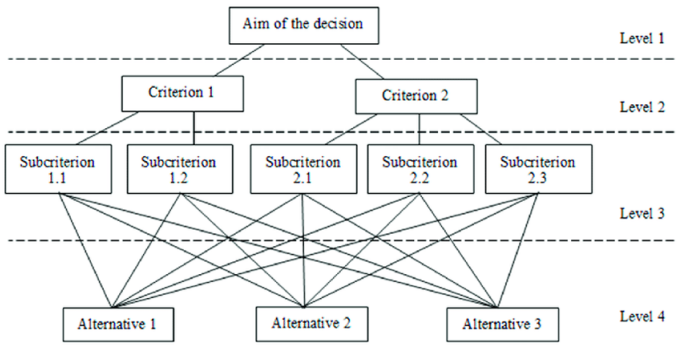


Figure 2.6: AHP decision tree (Xi and Qin 2013)

The first step of the AHP method is the construction of the decision tree, as in the example of figure 2.6, where at the top level, the goal is defined. At the middle level, the criteria used for decision-making are located, and these criteria can be grouped into clusters. At the last level, decision alternatives are reached. The advantage of a hierarchical method is the

clarity of information devoted to decision-making.

AHP uses a pair-wise comparison between criteria according to the scale of importance. This scale is called the Saaty scale, shown in Figure 2.7, and quantitatively evaluates between 1 and 9, being 1 for equal importance and 9 for very important (Mu and Pereyra-Rojas 2017). This evaluation will generate a comparison matrix (Ramos Filho, Atamanczuk, and Marçal 2010).

Intensity of Importance	Definition	Explanation
1	Equal importance	Two activities contribute equally to the objective
3	Weak importance of one over another	Experience and judgment slightly favor one activity over another
5	Essential or strong importance	Experience and judgment strongly favor one activity over another
7	Demonstrated importance	An activity is strongly favored and its dominance is demonstrated in practice
9	Absolute importance	The evidence favoring one activity over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between the two adjacent judgments	When compromise is needed

Figure 2.7: A summary of Saaty's nine-point ratio scale

After obtaining the comparison matrix, it must be normalized, and all priorities need to be leveled to the same unit. After the normalization of the matrix, the relative priority of each criterion will have to be identified. This calculation is done through the average of each row of the comparison matrix.

The fourth phase is to calculate the consistency ratio (CR) to assess whether the evaluations were consistent, for which the following equation must be executed:

$$CR = \frac{CI}{RI} \tag{2.1}$$

To obtain the Consistency Index (CI), the following equation should be used, where n is the number of criteria used, and λ_{max} is the average of the matrix division compared with the respective weights.

$$CI = \frac{\lambda_{max} - -n}{n - -1} \tag{2.2}$$

To obtain the Randomness Index (RI), Table 2.2 is used to identify this value, where n is the number of criteria used.

N	1	2	3	4	5	6	7	8	9	10
RI	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

Table 2.2: Randomness Index

After obtaining the CR value, if it is less than 0.10, we can conclude that the values of the priorities used are consistent; if the CR is greater than 0.10, this conclusion is not verified (Ramos Filho, Atamanczuk, and Marçal 2010).

The fifth phase of the AHP method consists of making a comparison matrix, where each row of the matrix will be an alternative, and each column will be a criterion; these values will be evaluated using the table in Figure 2.7.

The sixth phase is to multiply the alternative comparison matrix by the weights of the criteria obtained in phase three, thus making it possible to choose the best alternative in the seventh and last step.

For this project, we intend to identify the best deep learning libraries using AHP, taking into account that further tests will be carried out in the future to determine the best libraries to use. This identification will be made in the following sections.

2.4.1 Decision Tree

Figure 2.8 represents the decision tree of the AHP method, where the problem to solve is to find the best Deep Learning library to classify stress images; the criteria for evaluating the alternatives are layer methods available in each library, activation function available to help deal with the non-linearities of the input data, loss function which measures the performance of the data transformation, and optimizers which adjust the data to improve the performance of the loss function. The alternatives were selected based on state-of-the-art and are as follows: TensorFlow, Keras, PyTorch, and Caffe (Stančin and Jović 2019).

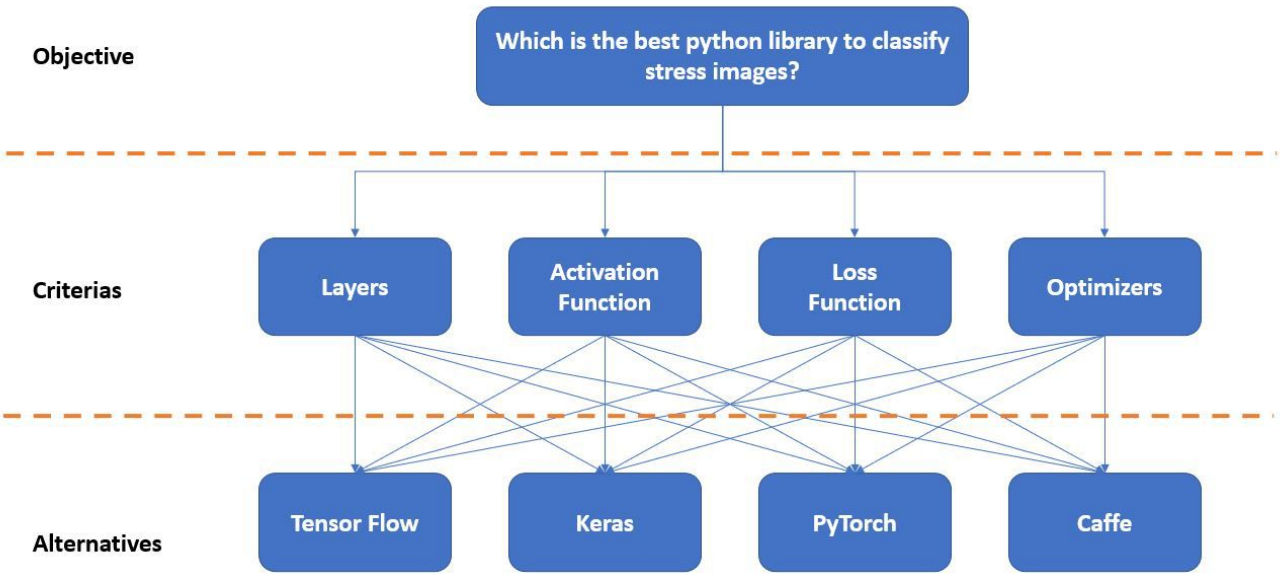


Figure 2.8: Decision tree of the project.

2.4.2 Pair-wise comparison between criterias

After building the decision tree, a Criteria Comparison Matrix was constructed based on the scale in Figure 2.8 and then the matrix was normalized. The comparison matrix and its comparison are shown in Tables 2.3, 2.4.

Criteria	Layers	Activation Function	Loss Function	Optimizers
Layers	1	1	1/5	1/9
Activation Function	1	1	1/3	1/9
Loss Function	5	3	1	1
Optimizers	9	9	1	1
Total	16	14	38/15	20/9

Table 2.3: Pair-wise comparison criteria matrix

Criteria	Layers	Activation Function	Loss Function	Optimizers	Criterias Weight
Layers	1/16	1/14	3/38	1/20	0.0657
Activation Function	1/16	1/14	5/38	1/20	0.0789
Loss Function	5/16	3/14	15/38	9/20	0.3429
Optimizers	9/16	9/14	15/38	9/20	0.5125

Table 2.4: Normalized pair-wise comparison criteria matrix

Eigenvector

The contribution of each criterion to the goal is determined by calculations made using the priority vector (or Eigenvector). The Eigenvector shows the relative weights between each criterion; it is obtained in an approximate manner by calculating the mathematical average of all criteria, as depicted in Table 2.5. The sum of all values from the vector is always equal to one.

Criteria	Eigenvector Calculation	EigenVector
Layers	[1/16 + 1/14 + 3/38 + 1/20]/4 = 0.0657	**0.0657**
Activation Function	[1/16 + 1/14 + 5/38 + 1/20]/4 = 0.0789	**0.0789**
Loss Function	[5/16 + 3/14 + 15/38 + 9/20]/4 = 0.3429	**0.3429**
Optimizers	[9/16 + 9/14 + 15/38 + 9/20]/4 = 0.5125	**0.5125**

Table 2.5: Eigen Vector

Consistency Index

The next step is to look for any data inconsistencies. The objective is to capture enough information to determine whether the decision-makers have been consistent in their choices.

The inconsistency index is based on the maximum Eigenvalue, which is calculated by summing the product of each element in the Eigenvector Table 2.5 by the respective line total of the original comparison matrix Table 2.3. Table 2.6 demonstrates the calculation of maximum Eigenvalue λ_{max}.

The calculation of the consistency index Saaty is given by the following formula:

$$CI = \frac{\lambda_{max} - -n}{n - -1} = \frac{4.1634 - 4}{4 - 1} = 0.0545 \tag{2.3}$$

Criteria	Layers	Activation Function	Loss Function	Optimizers
Eigen Vector	0.0657	0.0789	0.3429	0.5125
Total (Sum)	16.00	14.00	2.53	2.22
Maximum Eigenvalue λ_{max}	[(0.0657 * 16) + (0.0789*14) + (0.3429*38/15) + (0.5125*20/9)] = 4.1634			

Table 2.6: Eigenvalue λ_{max}

Consistency Ratio

The fourth step is to calculate the Consistency Ratio, the calculations to obtain this ratio are present in Equation 2.1 (the RI value is 0.9 since the project uses four criteria, this value is obtained in Table 2.2), and through this ratio, it was possible to conclude that the criteria priorities and weights are consistent since the ratio is less than 0.1.

$$CR = \frac{CI}{RI} = \frac{0.0545}{0.9} = 0.0605 \tag{2.4}$$

2.4.3 Pair-wise comparison between criteria and alternatives

In this step, it is necessary to evaluate each alternative in the different criteria, thus obtaining a comparison matrix for each criterion. Calculating the weight of each alternative in each criterion allows evaluating the alternatives in each criterion as it is possible to analyze in Figure2.9.

Layers	TensorFlow	Keras	PyTorch	Caffe	Criterias Weights
TensorFlow	1	1/5	2	3	0.1855
Keras	5	1	5	7	0.6266
PyTorch	1/2	1/5	1	2	0.1186
Caffe	1/3	1/7	1/2	1	0.0693
Total	6 5/6	1 1/2	8 1/2	13	

Table 2.7: Layers comparison matrix

Activation Function	TensorFlow	Keras	PyTorch	Caffe	Criterias Weights
TensorFlow	1	1/5	1	5	0.1946
Keras	5	1	3	5	0.5294
PyTorch	1	1/3	1	3	0.1777
Caffe	1/5	1/5	1/3	1	0.0688
Total	7 1/5	1 3/4	5 1/3	14	

Table 2.8: Activation Function comparison matrix

Loss Function	TensorFlow	Keras	PyTorch	Caffe	Criterias Weights
TensorFlow	1	1	1	2	0.2665
Keras	1	1	1	2	0.2665
PyTorch	1	1	1	2	0.2665
Caffe	1/2	1/2	1/2	1	0.1332
Total	3 1/2	3 1/2	3 1/2	7	

Table 2.9: Loss Function comparison matrix

Optimizer	TensorFlow	Keras	PyTorch	Caffe	Criterias Weights
TensorFlow	1	2	1	2	0.4285
Keras	1/2	1	1	3	0.2674
PyTorch	1	1	1	2	0.2665
Caffe	1/2	1/3	1/2	1	0.1062
Total	3	4 1/3	3 1/2	8	

Table 2.10: Optimizer comparison matrix

Through the weights that each alternative has in each criterion, it is possible to build the final matrix that contains the evaluation of each alternative in each criterion, like the matrix shown in Figure 2.9.

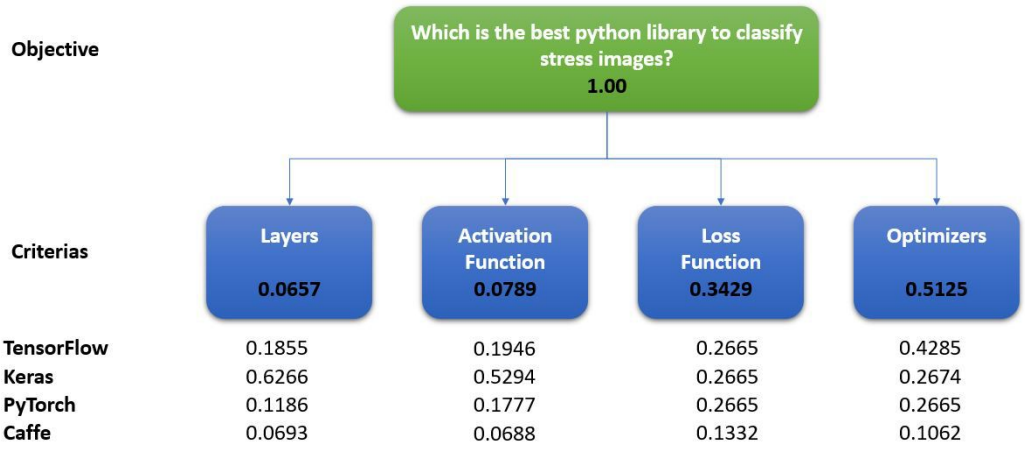


Figure 2.9: Decision tree with all weights

2.4.4 Alternatives priorities and Choice of the best alternative

Finally, the last step is to multiply the last matrices obtained, presented in Tables 2.7, 2.8, 2.9 and 2.10, by the weights of each criterion also presented in the Figure2.4. This final calculation, shown in Equation 2.5, allowed us to conclude that the best option would be the TensorFlow library.

$$\begin{bmatrix} 0.1855 & 0.1946 & 0.2665 & 0.4285 \\ 0.6266 & 0.5294 & 0.2665 & 0.2674 \\ 0.1186 & 0.1777 & 0.2665 & 0.2665 \\ 0.0693 & 0.0688 & 0.1332 & 0.1062 \end{bmatrix} x \begin{bmatrix} 0.0657 \\ 0.0789 \\ 0.3429 \\ 0.5125 \end{bmatrix} = \begin{bmatrix} 0.3385 \\ 0.3114 \\ 0.2498 \\ 0.1101 \end{bmatrix} \tag{2.5}$$

Given the lack of prerequisites in the current project, it was agreed with Professor Susana Nicola that it did not make sense to implement the QFD (Quality Function Deployment) and FAST (Function Analysis and System Technique) techniques.

Chapter 3

State of the art

This chapter presents the literature review of the existing technologies to approach the proposed problem.

3.1 Artificial Intelligence, Machine Learning and Deep learning

Artificial Intelligence, Machine Learning, and Deep Learning are three terms that are often used interchangeably, but they do not entirely refer to the same things. Here is an illustration with the fundamental differences between artificial intelligence, machine learning, and deep learning.

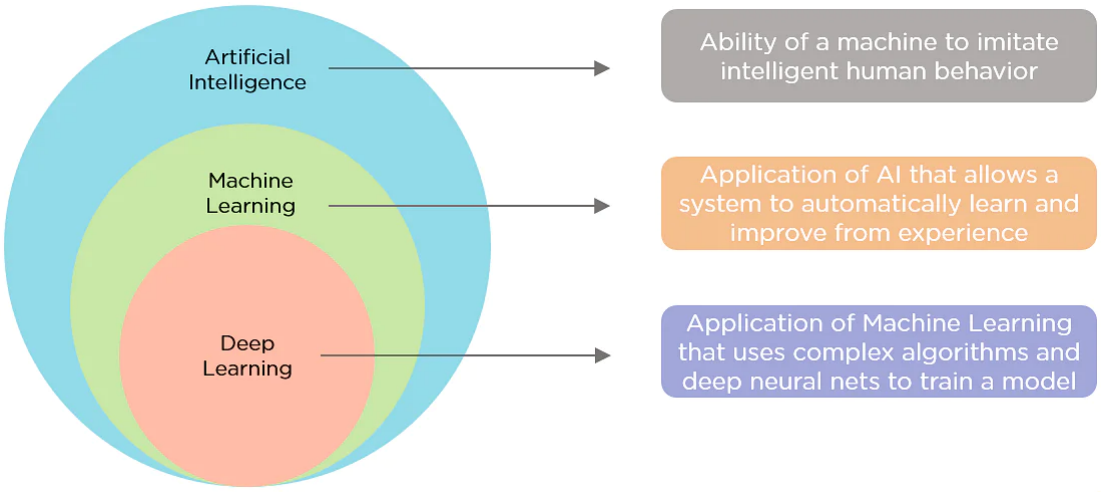


Figure 3.1: Artifial Intelligence diagram (Shruti 2022)

Figure 3.1 shows that Artificial Intelligence is the concept of creating innovative, intelligent machines, as Machine Learning is a subset of artificial intelligence that helps to build AI-driven applications. Deep Learning is a subset of machine learning that uses vast volumes of data and complex algorithms to train a model.

3.1.1 Artificial intelligence

Artificial intelligence, commonly referred to as AI, is the process of imparting data, information, and human intelligence to machines. The main goal of Artificial Intelligence is to develop autonomous machines that can think and act like humans. These machines can

mimic human behavior and perform tasks by learning and problem-solving. Most AI systems simulate natural intelligence to solve complex problems.

Types of Artificial Intelligence

- Reactive Machines - These are systems that only react. These systems do not form memories, and they do not use any past experiences to make new decisions.
- Limited Memory - These systems reference the past, and information is added over a period of time. The referenced information is short-lived.
- Theory of Mind - This covers systems that are able to understand human emotions and how they affect decision-making. They are trained to adjust their behavior accordingly.
- Self-awareness - These systems are designed and created to be aware of themselves. They understand their own internal states, predict other people's feelings, and act appropriately.

3.1.2 Machine learning

Machine learning is a discipline of computer science that uses computer algorithms and analytics to build predictive models that can solve business problems. As per McKinsey & Co., machine learning is based on algorithms that can learn from data without relying on rules-based programming (Pyle and José 2015).

Machine learning accesses vast amounts of data (both structured and unstructured) and learns from it to predict the future. It learns from the data by using multiple algorithms and techniques. Below is a diagram that shows how a machine learns from data.

Figure 3.2: Machine Learning Flow (Shruti 2022)

Types of Machine Learning

Machine learning algorithms are classified into three main categories:

Supervised Learning

In supervised learning, the data is already labeled, which means it knows the target variable. Using this learning method, systems can predict future outcomes based on past data. It requires at least an input and output variable to be given to the model for it to be trained.

Unsupervised Learning

Unsupervised learning algorithms use unlabeled data to discover patterns from the data on their own. The systems can identify hidden features from the input data provided. Once the data is comprehensive, the patterns and similarities become more evident.

Reinforcement Learning

The goal of reinforcement learning is to train an agent to complete a task within an uncertain environment. The agent receives observations and a reward from the environment and sends actions to the environment. The reward measures how successful action is concerning completing the task goal.

3.1.3 Deep learning

Deep learning is a subset of machine learning that deals with algorithms inspired by the structure and function of the human brain. Deep learning algorithms can work with an enormous amount of structured and unstructured data. Deep learning's core concept lies in artificial neural networks, which enable machines to make decisions.

The significant difference between deep learning and machine learning is how data is presented to the machine. Machine learning algorithms usually require structured data, whereas deep learning networks work on multiple layers of artificial neural networks.

This is what a simple neural network looks like:

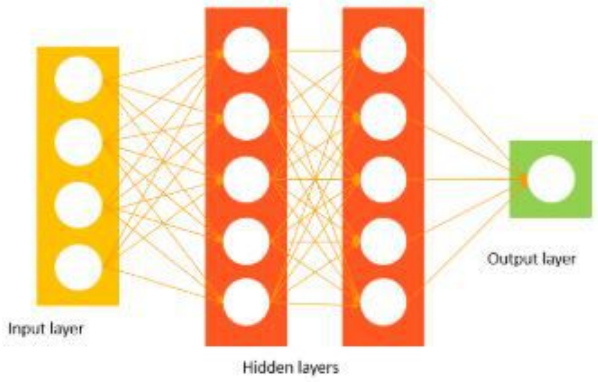


Figure 3.3: Deep learning overview (Shruti 2022)

The network has an input layer that accepts inputs from the data. The hidden layer is used to find any hidden features from the data. The output layer then provides the expected output.

Deep Learning Flow

1. Calculate the weighted sums.
2. The calculated sum of weights is passed as input to the activation function.
3. The activation function takes the "weighted sum of input" as the input to the function, adds a bias, and decides whether the neuron should be fired or not.

4. The output layer gives the predicted output.
5. The model output is compared with the actual output. After training the neural network, the model uses the backpropagation method to improve the network's performance. The cost function helps to reduce the error rate.

Types of Deep Neural Networks

There are some types of neural networks in deep learning, such as:

Convolutional neural networks (CNN) are one of the most popular models today. It uses a variation of multi-layer perceptrons and contains one or more convolutional layers that can be either entirely connected or pooled so that the convolutional layers can create feature maps that record a region of the image, which is ultimately broken into rectangles and sent out for nonlinear processing.

The main reason for the development of CNN was that the previous neural network architectures (e.g., feed-forward neural networks) could not scale up to handle image and video processing tasks. So, CNNs were explicitly tailored to image and video processing tasks.

The CNN applications to solve problems in the real world include a wide range of scientific fields which fall into the two main categories: pattern identification and classification (Bhujel and Pant 2017).

Recurrent neural network (RNN) is a type of network more complex because it saves the output of processing nodes and feeds the result back into the model (they do not pass the information in one direction only). It is how the model is said to learn to predict the outcome of a layer. Each node in the RNN model acts as a memory cell, continuing the computation and implementation of operations. If the network's prediction is incorrect, the system self-learns and continues working towards the correct prediction during backpropagation.

3.1.4 Difference between ML and DL

	Machine Learning	**Deep Learning**
Data dependencies	Excellent performances on a small/medium dataset	Excellent performance on a big dataset
Hardware dependencies	Work on a low-end machine.	Requires powerful machine, preferably with GPU: DL performs a significant amount of matrix multiplication
Feature engineering	Need to understand the features that represent the data	No need to understand the best feature that represents the data
Interpretability	Some algorithms are easy to interpret (logistic, decision tree), some are almost impossible (SVM, XGBoost)	Difficult to impossible
Training time	Short	Long

Table 3.1: Comparison between Machine Learning and Deep Learning

The first advantage of Deep Learning over machine learning is the needlessness for Feature Extraction. Traditional machine learning methods like Decision Trees, SVM, Naïve Bayes Classifier, and Logistic Regression cannot be applied directly to the raw data (such as .csv, images, text, and others.) but requires Feature Extraction. Feature Extraction results are an abstract representation of the given raw data that these classic machine learning algorithms can now use. Feature Extraction is usually complicated and requires detailed knowledge of the problem domain. This step must be adapted, tested, and refined over several iterations for optimal results.

Neural networks do not require feature extraction because the layers can learn an implicit representation of the raw data directly on their own. A more and more abstract representation of the raw data is produced over several layers to be used to produce the result with automatic optimization for the best possible abstract representation of the input data with no manual effort to perform and optimize the feature extraction process.

Deep Learning models increase their accuracy with the increasing amount of training data, but traditional machine learning models like SVM and Naive Bayes classifier stop improving after a saturation point.

3.2 Modeling Lifecycle

In contrast with the traditional approach of feature engineering followed by model learning, deep learning is an end-to-end learning approach. The features do not need to be previously labeled; instead, they should be learned automatically from the input data. Besides, the features learned are usually complex and have a hierarchy along with the network representation.

Since it requires less domain expertise and experience from the modeler, adjusting and tuning the learned model (its network structure and hyper-parameters) are crucial when developing new models (Miao et al. 2017). The main complexity of this process relies on understanding and explaining the models' results. Figure 3.4 shows a typical deep learning modeling lifecycle.

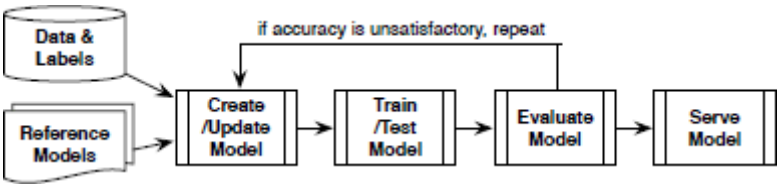


Figure 3.4: Deep learning modeling lifecycle (Miao et al. 2017)

Much time is spent at the beginning gathering data. Even if it has enough data available to create a model, data may require some processing. When working with image data for object detection, it needs bounding box data for the items in each image.

At the start of the project, it must also define the categories or labels in which data falls. In addition to gathering "correct" data examples to make predictions for the types of data expected, it is also necessary to gather "incorrect" data to create a negative label.

The deep learning cycle is iterative, and the model creation process involves frequent loops of model creation and testing. The overall process looks like these steps.

1. Use initial data to create the dataset and model.
2. Test the model.
3. Refine the dataset based on the model test results.
4. Create a new test model and compare it to the previous models.
5. When satisfied with the results, create a final production model from the most recent dataset.

Due to a lack of understanding about why models work, the adjustments and tuning inside the loop are driven by heuristics, for instance, adjusting hyperparameters that appear to have a significant impact on the learned weights, and applying novel layers of tricks seen in recent empirical studies, and so on. Thus, many similar models are trained and compared, and a series of model variants must be explored and developed.

Due to the expensive learning/training phase, each iteration of the modeling loop takes a long period of time and produces many (checkpointed) snapshots of the model.

3.3 Pre-processing

Data preprocessing is the process of transforming raw data into a suitable format for a machine learning or deep learning model. It is impossible to work with raw data in data mining, so data quality should be checked before applying machine learning or data mining algorithms (García, Luengo, and Herrera 2015).

Real-world data generally contains noises, missing values, and maybe a format that cannot be used for machine learning models. Data preprocessing requires cleaning the data and making it suitable for data mining, machine learning, and other data science tasks. It is a technique generally used to increase the accuracy and efficiency of a machine learning model (Chandrasekar and Qian 2016).

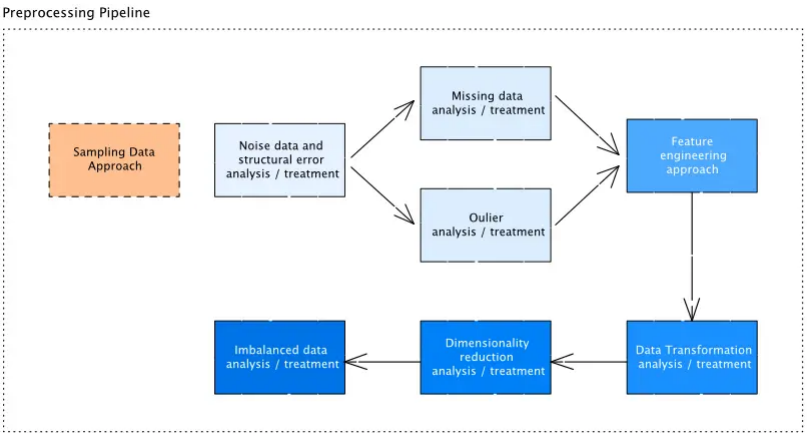


Figure 3.5: Data preprocessing pipeline (Azevedo 2021)

Among these techniques, some of the most fundamental approaches will be presented in the following sections.

3.3.1 Data Integration

Data Integration refers to a technique that involves combining data from multiple heterogeneous data sources into a coherent data store and providing a unified view of the data. These sources may include multiple data cubes, databases, or flat files. The data sources can be homogeneous or heterogeneous. The data obtained from the sources can be structured, unstructured, or semi-structured (Mhon and Kham 2020).

3.3.2 Data Cleaning

Data cleaning is the process that increases the quality of the input data, removing noisy data, completing incomplete data, and correcting inconsistencies in the data. If this step is not applied, it becomes complicated to consider that the data is reliable, leading to distrust in any algorithm learning process results (Alasadi and Bhaya 2017).

3.3.3 Data Transformation

Data Transformation is a technique used to convert the raw data into a suitable format that efficiently eases data mining and retrieves strategic information. Data transformation includes data cleaning techniques and a data reduction technique to convert the data into the appropriate form. It must be performed on the data before data-mining to understand patterns better. It can change the data's format, structure, or values and convert them into clean, usable data (Mhon and Kham 2020).

Some algorithms expect that the input data be transformed, so if it does not complete this process, it may get a poor model performance or even create bias.

Data Transformation involves the following:

- Normalization: It scales the data values in a specified range (-1.0 to 1.0 or 0.0 to 1.0)
- Attribute Selection: In this strategy, new attributes are constructed from the given set of attributes to help the mining process.
- Discretization: It replaces the raw values of a numeric attribute with interval levels or conceptual levels.
- Concept Hierarchy Generation: Converts the attributes from a lower level to a higher level in the hierarchy, i.e., the attribute "city" can be converted to "country".

3.3.4 Imbalanced Data

Imbalanced Data Distribution happens when perceptions in one class are much higher or lower than in different classes. It poses a difficulty for learning algorithms, as they will be biased towards the majority group. At the same time, the minority class is usually the one more important, as, despite its rareness, it may carry essential and valuable knowledge. Therefore, when facing such disproportions, one must design an intelligent system to overcome such a bias. This domain is known as learning from imbalanced data (García, Luengo, and Herrera 2015).

One of the most exciting directions in imbalanced binary classification is that the imbalance ratio is not the only source of learning difficulties. Even if the disproportion is high, but

the classes are well represented and come from non-overlapping distributions, it may obtain reasonable classification rates using canonical classifiers (Krawczyk 2016).

There are three main techniques that can be used to address this deficiency in the dataset:

- Oversampling: The oversampling approach increases the dataset with synthetic data from the minority class. The most popular technique is the Synthetic Minority Oversampling Technique (SMOTE) (Chawla et al. 2002). It takes a random example from the minority class. Then another random data point is selected through k-nearest neighbors of the first observation, creating a new record between these two selected data points.
- Undersampling: The undersampling technique, in contrast, is the process of reducing the dataset and removing inaccurate data from the majority class. The main algorithms used in this approach are the TomekLinks, which removes the observation based on the nearest neighbor, and the Edited Nearest Neighbors (ENN), which uses the k-nearest neighbor instead of only one as in Tomek.
- Hybrid: The hybrid approach combines the oversampling and undersampling techniques in the dataset. One of the algorithms that are used in this method is the SMOTEENN, which for the oversampling technique, it uses the Synthetic Minority Oversampling Technique (SMOTE) that randomly creates artificial samples along the line joining a minority sample and one of its nearest neighbors. The undersampling technique uses Random undersampling, a sampling technique to improve imbalance levels of the classes to the desired target by randomly removing instances from the majority class(es).

3.4 Convolutional neural network

Convolutional Neural Networks (CNNs) are the most used Deep-Learning model to solve Computer Vision tasks, particularly for image classification. The basic building blocks of CNNs are convolution, pooling (downsampling) operators, activation functions, and fully connected layers, which are essentially similar to the hidden layers of a Multi-layer Perceptron (MLP). Architectures such as VGG and ResNet, presented in Section 3.6, became very popular and used subroutines to obtain offered representations as input to other algorithms to solve different tasks.

It is possible to describe a network as a composition of a sequence of functions $\mathbf{f}_l(.)$ (related to some layer l) that takes as input a vector $\mathbf{x}_l$ and a set of parameters $\mathbf{W}_l$, and outputs a vector $\mathbf{x}_{l+1}$:

$$f(x) = f_L(\cdots f_2(f_1(x_1,\ W_1); W_2)\cdots), W_L) \tag{3.1}$$

where $\mathbf{x}_1$ is the input image, and in a CNN, the most characteristic functions $\mathbf{f}_l(.)$ are convolutions. Convolutions and other operators function as fundamental building blocks in constructing a CNN and its layers; they are the activation function, pooling, normalization, and linear combining (produced by the fully connected layer).

3.4.1 Convolutional Layer

A convolutional layer comprises a set of filters, each applied to the entire input vector. Each filter is a matrix $k \times k$ of weights (or values) $\mathbf{w}_i$ and the weights are parameters of the model to be learned (Gonzalez and Woods 2007).

The result of the sum of each filter produces the transformation of the input. In other words, each filter produces a linear combination of all pixel values in a neighborhood defined by the size of the filter, and each region that the filter processes is called a local receptive field so that the output value (pixel) combines the input pixels in this local receptive field (see Figure 3.6). In a convolutional layer, the output value $f(i, x, y)$ is based on a filter i, and local data are coming from the previous layer centered at a position (x, y).

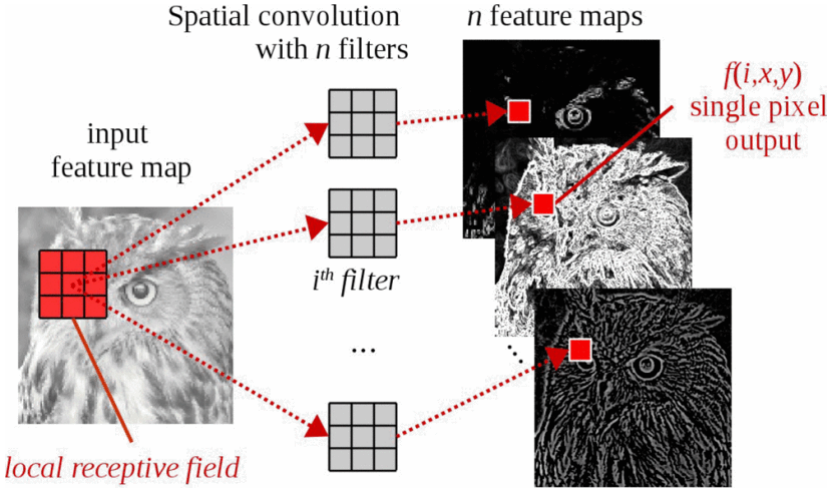


Figure 3.6: Example of a convolution processes

Figure 3.6, shows a convolution processes local information centred in each position (x, y): this region is called local receptive field, whose values are used as input by some filter i with weights $\mathbf{w}_i$ in order to produce a single point (pixel) in the output feature map $f(i, x, y)$

For example, if the input is an RGB image, and it wants the first convolutional layer to have four filters of size 5×5 to operate over this image, that means it needs a $5 \times 5 \times 3$ filter (the last 3 is for each color channel). Let the input image have size $128 \times 128 \times 3$, and a convolutional layer with four filters; then the output will have four matrices; stacked, it has a tensor of size $128 \times 128 \times 4$. Assuming that it uses a zero-padding approach to perform the convolution to keep the image's dimensions.

The most commonly used filters are $5 \times 5 \times d$, $3 \times 3 \times d$, and $1 \times 1 \times d$, where d is the depth of the tensor, considering that in the case of 1×1, the output is a linear combination of all feature maps for a single pixel and not the spatial neighbors but only the depth neighbors.

It is important to emphasize that the convolution operator can have different strides, which defines the step taken between each local filtering. For the default stride equals 1, the convolution considers all pixels, but in stride equals 2, it processes every odd pixel, skipping the others. As a result, it is common to use an arbitrary value of stride s, for example $s = 4$ in AlexNet (Krizhevsky, Sutskever, and Hinton 2012) and $s = 2$ in DenseNet (Huang et al. 2017), to reduce the running time.

3.4.2 Activation Function

The rectified linear function (ReLU) is often used in CNNs after convolutional layers or fully connected layers (Nair and Hinton 2010), in contrast to the use of a sigmoid function such as the logistic or hyperbolic tangent in MLP. Although ReLU can be employed before layers in a pre-activation setting (Larsson, Maire, and Shakhnarovich 2016), activation functions are not useful after pooling layers because such layer only downsamples the input data.

Figure 3.7 shows the plots of some activation functions: ReLU, in Figure 3.7(c), cancels out all negative values, and it is linear for all positive values. It is related to the non-negativity constraint often used to regularize image processing methods based on subspace projections (Ponti et al. 2015). The Parametric ReLU (PReLU), in Figure 3.7(d),allows small negative features, parametrized by $0 \leq a \leq 1$ (He et al. 2015). Designing the layers to be learnable during the training stage is possible when it is fixed at $a = 0.01$, it has the Leaky ReLU.

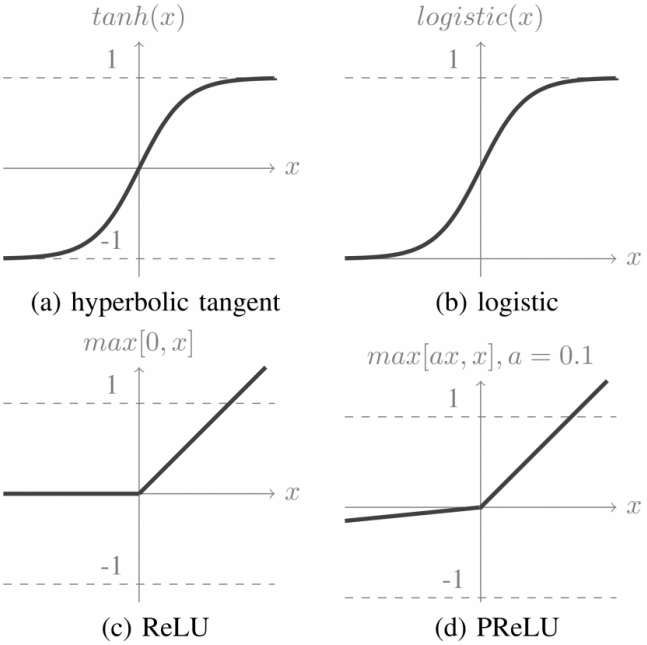


Figure 3.7: Activation functions

Figure 3.7, illustrates that the activation functions, (a) and (b) are often used in multilayer perceptron networks, while ReLUs (c) and (d) are more common in CNNs. Note (d) with $a = 0.01$ is equivalent to leaky ReLU.

3.4.3 Feature or Activation Map

Each convolutional neuron produces a new vector that passes through the activation function, and it is then called a feature map. Those maps are stacked, forming a tensor offered as input to the next layer.

For example, in Figure 3.8, the first convolutional layer outputs a $64 \times 64 \times 4$ tensor. If a second convolutional layer is set with filters of size 3×3, those will be $3 \times 3 \times 4$ filters. So, each one independently processes the entire tensor and outputs a single feature map. Figure 3.8 illustrates two convolutional layers producing a feature map.

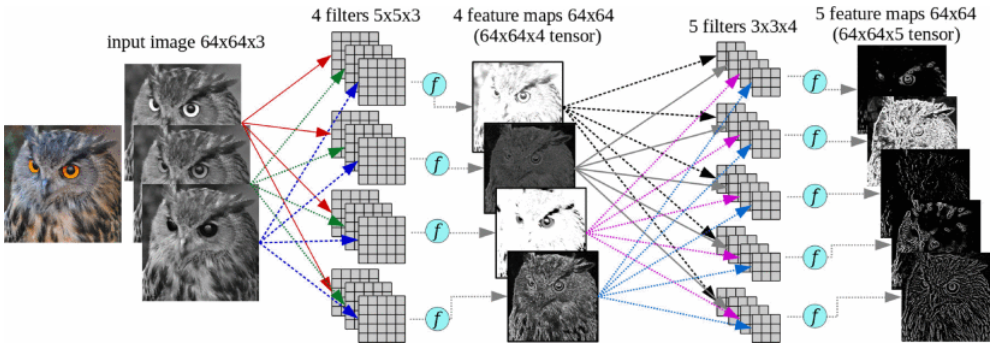


Figure 3.8: Feature maps

Figure 3.8 illustrates two convolutional layers, the first with four filters $5 \times 5 \times 3$ that get as input an RGB image of size $64 \times 64 \times 3$ and produces a tensor of feature maps. A second convolutional layer with five filters $3 \times 3 \times 4$ gets as input the tensor from the previous layer of size $64 \times 64 \times 4$ and produces a new $64 \times 64 \times 5$ tensor of feature maps. The circle after each filter denotes an activation function, for example, ReLU.

3.4.4 Pooling

After a few convolutional layers, pooling is applied, and it is used to downsample the image to reduce the spatial dimensionality of the vector. The max-pooling operator is the most frequently used. This type of operation has two primary purposes: first, it reduces the size of the data: as it will show in the specific architecture, the depth (3rd dimension) of the tensor often increases, and therefore it is convenient to reduce the first two dimensions. Second, by reducing the image size, it is possible to obtain a multiresolution filter bank by processing the input in different scale-spaces.

Although, there are studies in favor of discarding the pooling layers to reduce the size of the representations via an enormous stride in the convolutional layers (Springenberg et al. 2014). The effort to train a generative model with pooling layers is even greater because there is probably a tendency for future architectures to avoid pooling layers.

3.4.5 Normalization

It is common to apply normalization to both the input data and after convolutional layers. In preprocessing, it is common to apply to the input data a z-score normalization (centered by subtracting the mean and normalizing the standard deviation to unity) as described by (LeCun et al. 2012), which can be seen as a whitening process. For layer normalization, there are different approaches, such as the channel-wise layer normalization, that normalize the vector at each spatial location in the input map, either within the same feature map or across consecutive channels/maps, using L1-norm, L2-norm, or variations.

In AlexNet architecture (Krizhevsky, Sutskever, and Hinton 2012) the Local Response Normalization (LRN) is used: for every particular input pixel (x, y) for a given filter i, it have a single output pixel $\mathbf{f}_i(x, y)$, which is normalized using values from adjacent feature maps j, in other words $\mathbf{f}_j(x, y)$. This procedure incorporates information about the outputs of other kernels applied to the same position (x, y).

Nevertheless, more recent methods such as GoogLeNet (Szegedy et al. 2016) and ResNet (He et al. 2016a) do not mention the use of Local Response Normalization. Instead, they use Batch normalization (BN) (Ioffe and Szegedy 2015).

3.4.6 Fully Connected Layer

After many convolutional layers, it is common to include fully connected (FC) layers to learn the weights to classify the representation. In contrast to a convolutional layer, each filter in a fully connected layer produces a matrix of local activation values. The fully connected layer looks at the entire input vector, producing a single scalar value. It reshapes the data from the last layer as input. For example, if the last layer outputs a $4 \times 4 \times 40$ tensor, it reshapes it to be a vector of size $1 \times (4 \times 4 \times 40) = 1 \times 640$. Therefore, each neuron in the FC layer will be associated with 640 weights, producing a linear combination of the vectors.

The last layer of a CNN is often the one that outputs the class membership probabilities for each class c using logistic regression:

$$P(y = c|\mathbf{x}; \mathbf{w}; b) = \text{softmax}_c(\mathbf{x}^T\mathbf{w} + b) = \frac{e^{\mathrm{x}^T \mathrm{w}_c + b_c}}{\sum_j e^{\mathrm{x}^T \mathrm{w}_j + b_j}}, \tag{3.2}$$

where y is the predicted class, x is the vector coming from the previous layer, and w and b are, respectively, the weights and the bias associated with each neuron of the output layer.

3.5 CNN Architectures and Parameters

Typical CNNs are organized using blocks of convolutional layers (Conv) followed by an activation function (AF), eventually Pooling (Pool), and then a series of fully connected layers (FC) which are followed by activation functions. Normalization of data before each layer can also be applied.

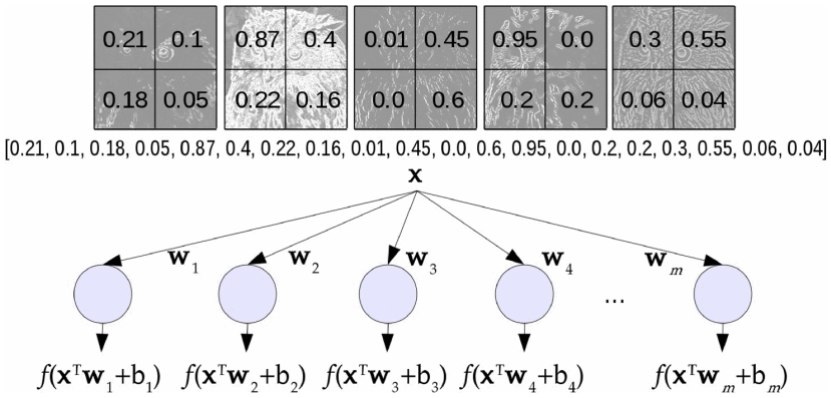


Figure 3.9: Transition between a convolutional and a fully connected layer

Figure 3.9 illustrates the transition between a convolutional and a fully connected layer: the values of all five activation/feature maps of size 2×2 are concatenated in a vector $\mathbf{x}$ and neurons in the fully connected layer will have full connections to all values in this previous layer producing a vector multiplication followed by a bias shift and an activation function in the form $f(\mathbf{x}^T\mathbf{w} + b)$.

To build a CNN, it is necessary to define the architecture, which is given first by the number of convolutional layers (for each layer also the number of filters, the size of the filters, and the stride of the convolution). Typically, a sequence of $Conv + AF$ layers is followed by Pooling (for each $Pool$ layer, it is necessary to define the window size and the stride that will define the downsampling factor). After that, it is common to have a fully connected layer (each FC layer should have a defined number of neurons). The pre-activation is also possible (He et al. 2016b), in which the data goes through AF and then to a $Conv.$ layer.

The number of parameters in a CNN is related to the number of weights used for learning. Those correspond to the values of all filters in the convolutional layer, all weights of fully connected layers, and bias terms.

3.5.1 Loss Function

A loss or cost function is a way to measure how bad the current model performs for a given input and an expected output; because it is based on a training set, it is an empirical loss function. Considering that the CNN is used as a regressor to discriminate between classes, then the loss $\ell(y, \hat{y})$ will express the penalty for predicting some $\hat{y}$, while the true output value should be y. The hinge loss or max-margin loss is an example of this function used in the SVM classifier optimization in its primal form. Let $\mathbf{f}_i \equiv f_i(x_j, W)$ be a score function for some class i given an instance x_j and a set o parameters W, then the hinge loss is:

$$\ell_j^{([\text{ht}])} = \sum_{c \neq y_j} \max(0, f_c - f_{y_j} + \triangle), \tag{3.3}$$

where class y_j is the correct class for the instance j, the hyperparameter Δ minimizes the loss, while the score of the correct class increases and gets more significant than the incorrect class scores by Δ at least. The squared version of this loss is called squared hinge loss.

The softmax classifier is often used in neural networks when the cross-entropy loss is employed. The minimization of the cross-entropy between the estimated class probabilities can be expressed as:

$$\ell_j^{(\text{ce})} = -\log\left(\frac{e^{f_{y_{j_j}}}}{\sum_k e^{f_k}}\right) \tag{3.4}$$

in which $k = 1 \cdots C$ is the index of each neuron for the output layer with C neurons, one per class.

This function transforms a vector of real-valued scores into a vector of values between zero and one with a unitary sum. There is an interesting probabilistic interpretation of minimizing Equation 3.4 in which the minimization of Kullback-Leibler divergence between two class distributions with zero entropy for true distribution has a single probability of 1 (Ioffe and Szegedy 2015). Also, we can interpret it as minimizing the negative log-likelihood of the correct class, which is related to the Maximum Likelihood Estimation.

The full loss of some training set is the average of the instances' x_j outputs, $\mathbf{f}(x_j; W)$, given the current set of all parameters W:

$$\mathcal{L}(W) = \frac{1}{N}\sum_{j=1}^{N} \ell(y_j, f(x_j; W)). \tag{3.5}$$

Now, it needs to minimize $\mathcal{L}(W)$ using some optimization method.

3.5.2 Regularization

The exclusive use of the loss function can lead to a possible problem. It happens because there might be many similar W for which the model can classify the training set correctly, which can mislead the process of finding suitable parameters via minimization of the loss, in other words, making it difficult to converge. In order to avoid ambiguity in the solution, it is possible to add a new term that penalizes undesired situations, called regularization. The most common regularization is the L2-norm: by adding a sum of squares, we want to discourage individually large weights:

$$R(W) = \sum_{k}\sum_{l} W_{k,l}^2 \tag{3.6}$$

After, we expand the loss by adding this term, weighted by a hyperparameter λ to control the regularization influence:

$$\mathcal{L}(W) = \frac{1}{N}\sum_{j=1}^{N} \ell(y_j, f(x_j; W)) + \lambda R(W). \tag{3.7}$$

The parameter λ controls how large the parameters in W are allowed to grow, and it can be found by cross-validation using the training set.

3.5.3 Optimization Algorithms

After defining the loss function, it is necessary to adjust the parameters to minimize the loss. First, the gradient descent is chosen as the optimization algorithm, followed by the backpropagation method that helps calculate the gradient of a loss function concerning all the weights in the network.

Note that $\mathcal{L}(W)$ is based on a finite dataset, so it has to calculate Montecarlo estimations of the actual distribution that generates the parameters. Since CNNs can have many parameters to be optimized, there is the need to be trained using thousands or millions of images. Nevertheless, an extensive set of examples is not viable for Gradient Descent because the algorithm has to calculate the gradient for all examples individually. The difficulty here is that if it tries to run an epoch, in other words, a pass-through of all the data, it would have to load all the examples into a limited memory, which is impossible. The alternatives to overcome this problem are described below, including the Stochastic Gradient Descent (SGD), Momentum, AdaGrad, RMSProp, and Adam.

Stochastic Gradient Descent (SGD)

One possibility to accelerate the process is to use approximate methods that go through the data and create samples composed of random examples drawn from the original dataset.

This method is called Stochastic Gradient Descent, and it is not interested in inspecting all available data at a time. However, it creates a random sample which adds uncertainty to the process. It is possible to calculate the Gradient Descent using a single example at a time (a method often used in streams or online learning). Nevertheless, in practice, it is common to use mini-batches with size B to perform enough iterations (each iteration will calculate the new parameters using the examples in the current mini-batch), and it approximates the Gradient Descent method.

$$W_{t+1} = W_t - \eta \sum_{j=1}^{B} \nabla \mathcal{L}(W; x_j^{\mathsf{B}}), \tag{3.8}$$

where η is the learning rate parameter: a large η will produce more significant steps in the direction of the gradient, while a small value produces a minor step in the direction of the gradient. It is common to set a larger initial value for η and then exponentially decrease it as a function of the iterations.

Stochastic Gradient Descent is a rough approximation, producing a nonsmooth convergence. Because of that, variants were proposed to compensate, such as the Adaptive Gradient (AdaGrad), Adaptive learning rate (AdaDelta), and Adaptive moment estimation (Adam). Those variants use the ideas of momentum and normalization, as we describe below.

Momentum

Adds a new variable α to control the change in the parameters W. It creates a momentum that prevents the new parameters W_{t+1} from deviating too much from the previous direction:

$$W_{t+1} = W_t + \alpha(W_t - W_{t-1}) + (1 - \alpha)[-\eta \nabla \mathcal{L}(W_t)] \tag{3.9}$$

where $\mathcal{L}(W_t)$ is the loss calculated using some examples using the current parameters W_t (often a mini-batch). Notice the magnitude of the step for the iteration $t + 1$, which constrains the step taken in the iteration t.

AdaGrad

It works by putting more weight on infrequent parameters. It creates a history of how a parameter can change the loss by accumulating the individual gradients $g_{t+1} = g_t + \nabla \mathcal{L}(W_t)^2$. Next, it is scaled/normalized for each parameter:

$$W_{t+1} = W_t - \frac{\eta \nabla \mathcal{L}(W_t)^2}{\sqrt{g_{t+1}} + \epsilon} \tag{3.10}$$

since this historical gradient is calculated feature-wise, the infrequent features will have more influence in the next gradient descent step.

RMSProp

It calculates averages of recent gradient magnitudes and normalizes them using these averages to normalized loosely gradient values. It is similar to AdaGrad, but here g_t calculates an exponentially decaying average and not the simple sum of gradients:

$$g_{t+1} = \gamma g_t + (1-\gamma)\nabla\mathcal{L}(W_t)^2 \tag{3.11}$$

g is called the second order moment of $\mathcal{L}$. The final parameter update is given by adding the momentum:

$$W_{t+1} = W_t + \alpha(W_t - W_{t-1}) + (1-\alpha)\left[-\frac{\eta\nabla\mathcal{L}(W_t)}{\sqrt{g_{t+1}} + \epsilon}\right] \tag{3.12}$$

Adam

It is similar to AdaGrad and RMSProp, but the momentum is used for the first and second-order moment, so it has α and γ controlling the momentum of respectively W and g. The influence of both decays is perceived over time so that the step size decreases when it approaches a minimum. It is used as an auxiliary variable m for clarity:

$$m_{t+1} = \alpha_{t+1} g_t + (1-\alpha_{t+1})\nabla\mathcal{L}(W_t)$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\alpha_{t+1}}$$

m is called the first order moment of $\mathcal{L}$ and $\hat{m}$ is m after applying the decaying factor. Then it needs to calculate the gradients g to use in the normalization:

$$g_{t+1} = \gamma_{t+1} g_t + (1-\gamma_{t+1})\nabla\mathcal{L}(W_t)^2$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1-\gamma_{t+1}}$$

g is called the second order moment of $\mathcal{L}$. The final parameter update is given by:

$$W_{t+1} = W_t - \frac{\eta\hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}} + \epsilon} \tag{3.13}$$

3.5.4 Adjustments and tuning

The process of setting the hyper-parameters requires expertise and extensive trial and error. There are no simple and easy ways to set hyper-parameters — learning rate, batch size, momentum, and weight decay. These hyper-parameters act as knobs that can be tweaked during the model's training. For our model to provide the best result, we need to find the optimal value of these hyper-parameters. Here are some extra configurations that can help in its optimization.

Initialization

Random initialization of weights is important to the convergence of the network, and the Gaussian distribution $\mathcal{N}(\mu, \sigma)$ is often used to produce the random numbers. However, for models with more than eight convolutional layers, the use of a fixed standard deviation (for example, $\sigma = 0.0]$) as in (Krizhevsky, Sutskever, and Hinton 2012) was shown to hamper convergence. Therefore, when using rectifiers as activation functions it is recommended to use $\mu = 0, \sigma = \sqrt{2/n_l}$, where n_l is the number of connections of a response of a given layer l; as well as initializing all bias parameters to zero (He et al. 2015).

Mini-batch size

Due to the need to use SGD optimization and variants, it is necessary to define the size of the mini-batch of images that will be used to train the model, taking into account memory constraints and the behavior of the optimization algorithm. For example, while a small batch size can make the loss minimization more difficult, the convergence speed of SGD degrades when increasing the mini-batch size for convex objective functions (M. Li et al. 2014). In particular, assuming SGD converges in T iterations, then mini-batch SGD with batch size B runs in $O(1/\sqrt{BT} + 1/T)$.

However, increasing B is essential to reduce the variance of the SGD updates (by using the average of the loss), which allows it to take bigger step-sizes (Bottou, Curtis, and Nocedal 2018). Also, larger mini-batches are interesting when using GPUs since it gives better throughput by performing backpropagation with data reuse using matrix multiplication (instead of several vector-matrix multiplications) and needing fewer transfers to the GPU.

Therefore, it can be advantageous to choose the batch size to fully occupy the GPU memory and choose the largest experimentally found step size. While popular architectures use from 32 to 256 examples in the batch size, a recent paper used a linear scaling rule for adjusting learning rates as a function of mini-batch size, also adding a warmup scheme with large step sizes in the first few epochs to avoid optimization problems.

Dropout

A technique proposed by Goyal et al. that, during the forward pass of the network training stage, randomly deactivates neurons of a layer with some probability p in particular from FC layers (Goyal et al. 2017). It has relationships with the Bagging ensemble method (Warde-Farley et al. 2013) because, at each iteration of the mini-batch SGD, the dropout procedure creates a randomly different network by subsampling the activations, which is trained using backpropagation. This method became known as a form of regularization that prevents the network from overfitting. In the test stage, the dropout is turned off, and the activations are re-scaled by p to compensate for those activations that were dropped during the training stage.

Batch normalization (BN)

Also used as a regularizer, it normalizes the layer activations at each batch of input data by maintaining the mean activation close to 0 (centering) and the activation standard deviation close to 1 and using parameters γ and β to produce a linear transformation of the normalized vector, in other words:

$$\mathrm{BN}_{\gamma,\beta}(x_i) = \gamma \left(\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \beta, \tag{2}$$

In which γ and β are parameters that can be learned during the backpropagation stage (Ioffe and Szegedy 2015). It allows, for example, to adjust the Normalization and even restore the data to its un-normalized form; so, when $\gamma = \sqrt{\sigma_B^2}$ and $\beta = \mu_B$.

BN became a standard in recent years, often replacing the use of both regularization and dropout (for example, ResNet (He et al. 2016a) and Inception V3 (Szegedy et al. 2016) and V4).

Data-augmentation

As mentioned previously, CNNs often have many parameters to optimize, requiring many training examples. Because it is tough to have a dataset with sufficient examples, it is common to employ some data augmentation. Images in the same dataset often have similar illumination conditions, a low variance of rotation, pose, and others; therefore, one can augment the training dataset by many operations in order to produce five to ten times more examples (Chatfield et al. 2014) such as:

- cropping images: CNNs often have a low-resolution image input (for example, 224×224), so it can create several cropped versions of an image with higher resolution;
- flipping images horizontally and/or vertically;
- adding noise (Nazaré et al. 2017);
- creates new images using PCA as in the Fancy PCA proposed by (Krizhevsky, Sutskever, and Hinton 2012).

The augmentation must be performed to preserve the labels.

Fine-tuning

When there is a small dataset, that can be challenging to train a CNN. Data augmentation can be insufficient since augmentation will create perturbed versions of the same images. In this case, it is beneficial to use a model already trained in a large dataset (for example, the ImageNet dataset (Deng et al. 2009)) with initial weights that were already learned. To obtain a trained model, adapt its architecture to the dataset, and re-enter the training phase using such a dataset is a fine-tuning process.

3.6 CNN Architectures for Image Classification

There are many proposed CNN architectures for image classification, so in this section, we compare the main image classification architectures used in this work and their performance. First, each architecture is described, and later it shows an overall comparison in Table 3.2.

3.6.1 VGG-Net

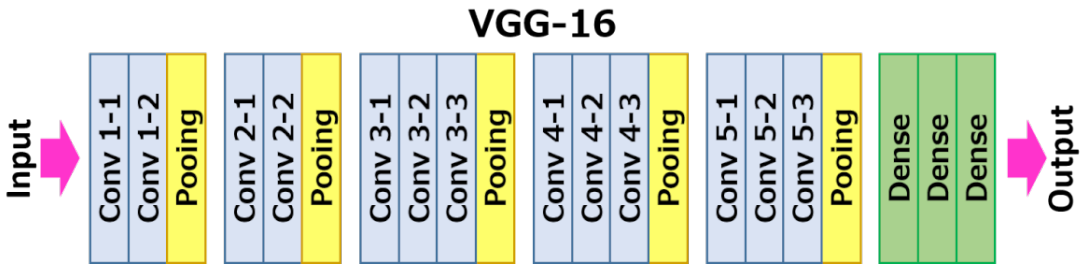


Figure 3.10: VGG-16 architecture (networks 2018)

It is an architecture developed to increase the depth while making all filters with at most 3×3 size. This idea is that two consecutive 3×3 convolutional layers will have an effective receptive field of 5×5, and 3 of such layers an effective receptive field of 7×7 when combining the feature maps.

Thus, stacking three convolutional layers with 3×3 filters instead of addressing just one with filter size 7×7 has the advantage of incorporating more rectification layers, making the decision function more discriminative. In addition, it incorporates 1×1 convolutional layers that perform a linear projection of a position (x, y) across all feature maps in a given layer. The two most commonly used versions of this CNN are VGG-16 and VGG-19, respectively, with 16 and 19 weight layers. In Figure 3.10, there is a diagram of the layers of VGG-16.

3.6.2 Residual Networks (ResNet)

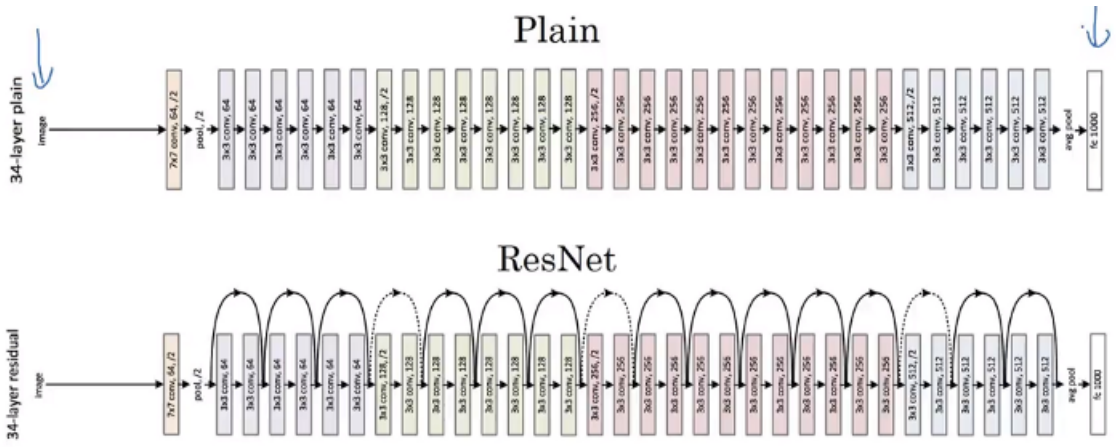


Figure 3.11: ResNet architecture (Joseph 2021)

He et al. claims this could be an optimization problem and proposes using residual blocks for CNNs with 34 to 152 layers (He et al. 2016a). Those blocks' designs preserve the characteristics of the original vector x before its transformation by some layer $f_l(x)$ by skipping weight layers and performing the sum $f_l(x) + x$.

Figure 3.11 shows an architecture with 34 layers plain and with shortcuts that implement the residual block and residual block with pooling. As a result of the gradient being an additive term, it is unlikely to vanish even with many layers. However, this architecture does

not contain any fully connected layer hidden; after the last convolution layer, the pooling calculates an average, followed by the output layer.

3.6.3 Comparison of archictectures

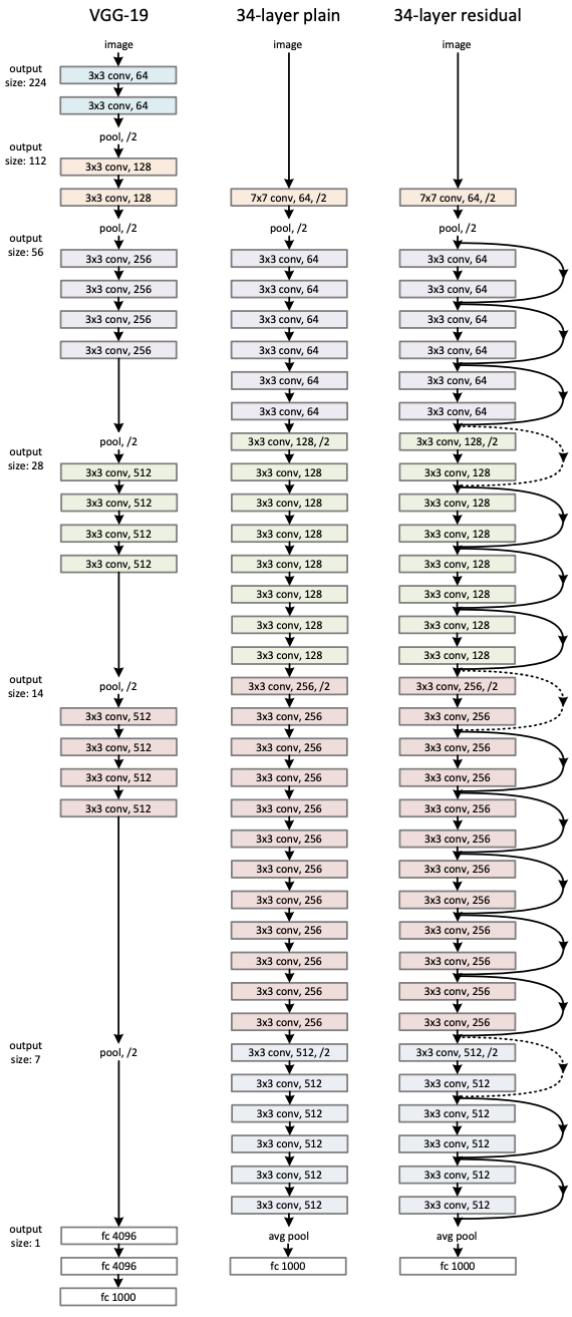


Figure 3.12: Example network architecture for ImageNet (He et al. 2016a)

Figure 3.12 compares VGG-19 to Resnet 34, on the left is the VGG-19 model as a reference. In the middle, there is a plain network with 34 layers. On the right is a residual network with 34 layers. The dotted shortcuts increase dimensions. The comparison of architectures

continues in Table 3.2 and uses the same criteria, which were the training parameters, model size, and top one errors in the VGGNet and ResNet.

	Training Parameters						Model		ImageNet
CNN	Batch size	Learn.rate	Optm.alg.	Optm.param.	Epochs	Regularization	# Layers	Size	top-1 error
VGGNet	256	0.01	SGD	α =0.9	74	L2 + Dropout	16-19	574MB (19 lay.)	24.4%
ResNet	256	0.1	SGD	α =0.9	120	BN	34-152	102MB (50 lay.)	19.3%

Table 3.2: Comparison of CNN models

Since Resnet blocks' designs can preserve the original vector's characteristics, increasing dimensions by adding layers to the model can improve the gradient performance as additive terms are unlikely to vanish within the layers. Opting for Resnet has advantages, such as having more layers than VGG but a smaller model size, which is crucial when working with limited hardware performance.

3.7 Evaluation Metrics

Many performance metrics are available to compare and evaluate deep models. However, one must ponder which metric is the best for the problem. There are different metrics for the tasks of classification and regression. Using different metrics for performance evaluation, we should be able to improve our model's overall predictive power.

3.7.1 Classification

Classification is about predicting the class labels given input data. In binary classification, there are only two possible output classes.

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figure 3.13: Confusion Matrix

- True Positive (TP): number of instances correctly accepted or predicted/classified as positive.
- True Negative (TN): number of instances correctly rejected or predicted/classified as negative.
- False Positive (FP): number of instances incorrectly accepted or predicted/classified as positive.
- False Negative (FN): number of instances incorrectly rejected or predicted/classified as negative.

Considering that there are many ways for measuring classification performance, these are some of the most popular metrics:

- Accuracy: proportion of true results (both positive and negative)

$$Accuracy = \left(\frac{TP + TN}{TP + TN + FP + FN}\right) \tag{3.14}$$

- Recall: true positive rate

$$Sensitivity = Recall = \left(\frac{TP}{TP + FN}\right) \tag{3.15}$$

- Precision: true negative rate

$$Specificity = \left(\frac{TN}{TN + FP}\right) \tag{3.16}$$

- F1 Score: harmonic mean of precision and recall. This evaluation measure calculates the effectiveness of a classifier algorithm, combining the accuracy and sensitivity measures.

$$F1Score = \left(\frac{2 \times TP}{2 \times TP + FN + FP}\right) \tag{3.17}$$

3.8 Stress Detection Review

There are many studies on stress detection that still use machine learning as a feature extractor to classify a person's stress state. This requires the developer to have previous knowledge about the area of study to interpret the results obtained. However, in recent years, with the improvement in hardware performance and the popularization of deep learning, it is possible to use deep learning to achieve results equal to or better than machine learning. The developer can create models without previous knowledge about the area of study.

Here we present the latest efforts to automate the prediction and detection of stress by machine learning and deep learning models, which are trained using psychological experiments to address the responses to stress and emotional stimuli.

Addressing the employees' reports on stress at work, Koldijk et al. developed automatic classifiers to examine the relationship between working conditions and mental stress-related conditions from sensor data: body postures, facial expression, computer logging, and physiology (ECG and skin conductance). They found that the performance of the specialized model was just as well or better than a generic model in almost all cases when similar users were subgrouped and models were trained on specific subgroups. In order to differentiate between a stressor and non-stressor working conditions, posture provides the most crucial information among the most valuable modalities. Performance could be further improved by adding data about one's facial expressions. They got an accuracy of 90% employing the SVM classifier (Koldijk et al. 2014).

Facial cues are another significant factor that can define a person's stress. Adding to this, Giannakakis et al. developed a framework for detecting and analyzing emotional states of stress/anxiety via video-recorded facial cues. The investigated features were mouth activity, events related to the eye, camera-based photoplethysmographic heart rate estimation, and head action parameters. Participants were placed 50 cm apart in front of a camera-integrated computer monitor. Methods like Generalized Likelihood Ratio, Naive Bayes classifier, Support Vector Machines, K-nearest neighbors, and AdaBoost classifier were used and tested. In the social exposure process, the highest classification performance was achieved using the Adaboost classifier, reaching an accuracy of 91.68% (Giannakakis et al. 2017).

Video-based stress detection leverages facial expressions without the interference of artificial traits and factors to predict stressful states. Zhang et al. developed a Two-leveled Stress

Detection Network (TSDNet), which firstly learns face-level and action-level representations separately, and then fuses the results through a stream weighted integrator for stress identification. In comparison, the Facial Cues-based approach, Action Units-based approach, and TSDNet results demonstrated the feasibility and advantage of using the TSDNet classifier, reaching an accuracy of 85,42% (Zhang et al. 2020).

Sabour et al. focuses on the link between stress and physiological responses. Although public datasets have limitations, the new dataset UBFC-Phys proposes to evaluate the possibility of using video-based physiological measures compared to more conventional contact-based modalities. The experimental results of contact and non-contact data comparison and stress recognition obtained a stress state recognition accuracy of 85.48%, achieved by remote Pulse Rate Variability features (Sabour et al. 2021).

Early detection of mental stress can prevent many health problems associated with stress. Bobade and Vani proposes different machine learning and deep learning techniques for stress detection on individuals using multimodal datasets. The accuracies for three-class (amusement vs. baseline vs. stress) and binary (stress vs. non-stress) classifications were evaluated and compared by using machine learning techniques like K-Nearest Neighbour, Linear Discriminant Analysis, Random Forest, Decision Tree, AdaBoost, and Kernel Support Vector Machine. During the study, using machine learning techniques, the accuracy of up to 81.65% and 93.20% were achieved for three-class and binary classification problems, respectively. By using deep learning, the achieved accuracy is up to 84.32% and 95.21%, respectively (Bobade and Vani 2020).

Since prolonged exposure to stress leads to physical and mental health problems. Given that most laptops have inbuilt cameras using video data for personal tracking of stress levels could be a more affordable alternative. Viegas et al. uses five one-hour-long videos from the dataset collected by Lau (Maxion and Lau 2018) to perform binary classification using several simple classifiers on AUs extracted in each video frame and were able to achieve an accuracy of up to 74% in subject independent classification and 91% in subject dependent classification (Viegas et al. 2018).

No.	Title	Author	Methodology & Performance
1	The swell knowledge work dataset for stress and user modeling research	Koldijk, Saskia and Sappelli, Maya and Verberne, Suzan and Neerincx, Mark A and Kraaij, Wessel. 2014	Computer logging, facial expression, posture of Employees. Accuracy of 90% using SVM.
2	Stress and anxiety detection using facial cues from videos	G. Giannakakis and M. Pediaditis and D. Manousos and E. Kazantzaki and F. Chiarugi and P.G. Simos and K. Marias and M. Tsiknakis. 2017.	Analysis of stress/anxiety emotional states through video-recorded facial cues. A feature selection procedure followed by classification. Accuracy of 91,68% using AdaBoost
3	Video-based stress detection through deep learning	Zhang, Huijun and Feng, Ling and Li, Ningyun and Jin, Zhanyu and Cao, Lei. 2020.	Action Units (AUs) based applied Random Forest, Gaussian Naive Bayes, and Decision Tree classifiers, Facial Cues (FCs) based applied Deep Learning with accuracy of 85,42%
4	Ubfc-phys: A multimodal database for psychophysiological studies of social stress	Sabour, Rita Meziati and Benezeth, Yannick and De Oliveira, Pierre and Chappe, Julien and Yang, Fan. 2021.	Four classifiers were considered: Support Vector Machine (SVM) with a linear kernel, SVM with Radial Basis Function (RBF), Logistic Regression (Log Reg) and K-Nearest Neighbors (KNN). Accuracy of 85.48%, achieved by remote PRV features.
5	Stress Detection with Machine Learning and Deep Learning using Multimodal Physiological Data	Bobade, Pramod and Vani, M. 2020.	Machine learning techniques like K-Nearest Neighbour, Linear Discriminant Analysis, Random Forest, Decision Tree, AdaBoost and Kernel Support Vector Machine and Deep Learning. Machine learning techniques, accuracy 93.20% for binary classification problems and by using deep learning, the achieved accuracy is 95.21%.
6	Towards Independent Stress Detection: A Dependent Model Using Facial Action Units	Viegas, Carla and Lau, Shing-Hon and Maxion, Roy and Hauptmann, Alexander. 2018.	Classifiers used were Random Forest, LDA, Gaussian Naive Bayes and Decision Tree. Accuracy of up to 74% in subject independent classification and 91% in subject dependent classification.

Table 3.3: Summary table of reviewed articles with methodology & performance evaluation.

3.9 Technologies

All the investigated frameworks support the standard routines for CNNs, which are crucial for image classification to perform Computer Vision related tasks and applications. Since a video frame is a sequence of single images, it is possible to compare the following frameworks as long as they are suited for the more generic use case of image recognition when effectively combining them with computer vision libraries such as OpenCV.

In general, GPUs (Graphics Processing Units) are strongly preferred over CPUs (Central

Processing Units) for CNNs and computer vision applications due to the performance gain (Chollet 2021), (Redmond and Sauer 2017). Concerning performance improvements via hardware acceleration, all four compared frameworks support GPU acceleration via the NVIDIA cuDNN (CUDA Deep Neural Network) library, which was released in 2014 (Redmond and Sauer 2017) and provided highly optimized access to the standard routines used in Convolutional Neural Networks.

It is relevant for the development since shorter training times lead to shorter testing times, likely resulting in better overall CNN models. CUDA dominates OpenCL (its Open-Source non-proprietary variant) (Redmond and Sauer 2017), which can be linked to NVIDIA's extensive research and investments in Deep Learning (Chollet 2021). All of the investigated frameworks support the popular Python programming language.

3.9.1 TensorFlow

TensorFlow is a machine learning framework publicly introduced by a deep learning research team called the Google Brain in 2015 (Redmond and Sauer 2017) that is intended to be used in large-scale system environments using dataflow graphs. It enables developers to use parallelization schemes and share the state across multiple machines and devices. It can run on multicore CPUs, GPUs, and specialized hardware units for Machine Learning known as TPUs (Tensor Processing Units) (Abadi et al. 2016).

TensorFlow offers extensive documentation and tutorials in industrial and research applications (Dubovikov 2017). For visualization, TensorBoard can display the TensorFlow graph, which includes images that compare different training runs of a model and can be utilized as a powerful debugging tool (Dubovikov 2017). With extended features such as distributed training and specialized deployment options for the models via TensorFlow Serving, TensorFlow can be considered very flexible for developers who want to use the full potential of their Deep Learning models and also deploy on mobile devices (Dubovikov 2017).

Because of TensorFlow's origin, image classification is an essential task for the framework, engineered with computer vision applications and performance in mind(Abadi et al. 2016).

3.9.2 Keras

Compared to TensorFlow, Keras is a higher-level library released in early 2015 (Chollet 2021) and sits on top of a backend that handles the low-level actions. Keras is a model-level library for a specified backend comparable to a "frontend" made for ML developers. The backend can be exchanged anytime without changing the Keras model implementation. At the same time, it enables the developer to test and use multiple backends, which can also be helpful to compare performance among the different backends with additional support to TensorFlow, the Microsoft Cognitive Toolkit, and Theano (Chollet 2021). When using TensorFlow as a backend, Keras can be used to run on multiple processing units seamlessly (such as GPU and CPU) and is therefore qualified to use CNNs for computer vision applications (Chollet 2021).

Keras is one of the most used libraries by entrants specializing in Deep Learning (to solve perceptual problems like image classification) on Kaggle.com (Chollet 2021). After all, Keras' purpose is to make Deep Learning more accessible and more usable for everyone by providing a user-friendly and high-level library that abstracts the often more complicated code constructs of its backends (Chollet 2021).

3.9.3 PyTorch

PyTorch was initially released in 2016 (Redmond and Sauer 2017) by the Facebook development team and is the Python descendant of the Torch Lua framework (Dubovikov 2017). The framework is highly Python-specific, while the core logic of PyTorch itself is written in C++ for better performance and less overhead (Dubovikov 2017).

The main difference to TensorFlow: PyTorch features dynamic computational graph definitions, making it easier to interact with external data. As in TensorFlow, special placeholders have to be used, which will be replaced by the actual data at runtime; the graph itself has to be defined statically before. Because the computational graph is dynamic in PyTorch, it is possible to use dynamic inputs for a model. In contrast, in TensorFlow, it is necessary to import an additional library called TensorFlow Fold.

For the same reason, debugging is possible with standard Python tools, where a specialized debugging tool like TensorFlow Debugger (tfdbg) is needed in TensorFlow to view the internal states of the computational graph (Dubovikov 2017). PyTorch is extensively used by the transport network company Uber. Its AI Labs team has built Pyro based on PyTorch to improve the transportation experience by predictions and route optimizations (Goodman 2017).

3.9.4 Caffe

Caffe stands for Convolutional Architecture for Fast Feature Embedding and was created by Yangqing Jia. At the same time, he was a student at the University of California, Berkeley starting in 2014. Caffe is a C++ library utilizing CUDA with well-supported bindings to Python and MATLAB. It attempts to follow software engineering best practices to offer a good code quality made possible by complete unit tests (code coverage) and clean coding, for example, through modularization, separation of concerns, and abstraction (Jia et al. 2014).

Caffe targets multimedia scientists, researchers, and the industry. The first release in 2014 focused on computer vision for object classification, which still is one of Caffe's leading applications. Later, it has grown through community contributions to support uses such as robotics and speech recognition (Jia et al. 2014). The members of Berkeley also cooperated with significant industry corporations like Adobe and Facebook.

A significant architectural difference from the other frameworks is the separation of the model's representation and the implementation. Hence, model definitions are stored in separate configuration files written in the Protocol Buffer Language (contrary to the other frameworks, not actual code). As a result, Caffe has a static computational graph structure (like TensorFlow) and is not dynamic like Keras (Jia et al. 2014).

A famous architecture that uses Caffe for person re-identification was introduced in 2015 (Dinghofer and Hartung 2020), which uses a CNN to detect if two images show the same person more efficiently by comparing local relationships of images using a thoughtfully adapted nearest neighbor layer and another one that outlines differences between the images (Ahmed, Jones, and Marks 2015).

For the means of evaluating the best framework for this project, the Analytical Hierarchy Process (AHP) was utilized in Section 2.4. The frameworks were evaluated using the criteria: the number of layers, activation function, loss function, and optimizers, and accordingly to

this multi-criteria decision support method, the best option would be the TensorFlow library, which will be adopted in this project.

Chapter 4

Analysis and Design

This chapter contains information about the view of the system from a structural point of view, analyzes the intended features and the architecture, and then presents the approach to solve the problem. In this chapter, it is possible to view the proposed solution architecture that contains details about each procedure performed to solve the problem and the prototype delivered.

4.1 Requirements Analysis

The project's first critical issue and requirement is the availability of different input/upload options for the user, such as images, videos, and streaming. Consequently, the capacity of the system folder needs to be managed by the application for uploading images or videos since these files will be saved in a temporary folder. However, in the case of streaming, it will not be necessary to control the size of the files since the images will be stored using the in-memory allocation to perform real-time analysis.

Second, the critical point of this solution is creating and training Deep Learning models, which will use the UBFC-Phys public dataset of video captures as a reference source for identifying facial expressions associated with stress (Sabour et al. 2021).

Finally, the system should provide an interface to check the output predictions. However, this interface should be built on the premise that the users who will use it have potentially little technical experience, especially regarding deep learning concepts. Therefore, the interface should be simple and concise to show all the information for understanding the output prediction.

4.1.1 Functional Requirements

After the analysis of the requirements that were collected above, these are the 5 use cases idealized:

- UC1: Upload a video
- UC2: Predict stress
- UC3: View prediction
- UC4: Manage algorithm
- UC5: Manage parameters

The Use Case diagram is represented in Figure 4.1.

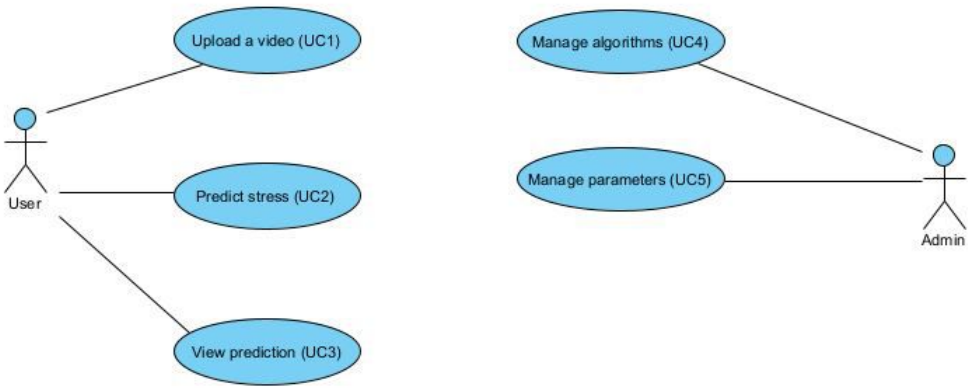


Figure 4.1: Use case diagram

In Figure 4.1, UC3 represents the model prediction process. It will be necessary for the user to push a button to start the prediction process (UC3). It happens when a prediction request is made, and the generated model stored in the folder is accessed to make a prediction. In this case, the system will predict a new image and show the user the result.

4.1.2 Non-Functional Requirements

To evaluate and assess the non-functional requirements, the FURPS+ model was used. This model categorizes the non-functional requirements into 5 + categories: Functionality, Usability, Reliability, Performance, and Supportability. The "plus" refers to Design Constraints, Implementation requirements, Interface requirements, and Physical requirements. Table 4.1 contains the identified non-functional requirements, organized by category.

Code	Requirements	Type
F01	HTTPS protocol should be used in the communications between components	Functionality
F02	All videos should be converted to images so the models can be used to make predictions.	Functionality
F03	Share with the user the main steps to make a predition using Deep learning	Functionality
U01	The User Interface should be simple, effective, and adapted to the User's needs	Usability
U02	The User Interface for showing the predictions that were made should not show technical data that could not possible be understood by the maintenance managers	Usability
P01	The prediction for the stream videos should not block the usage of the website	Performance
S01	The system should allow only supported images and videos format. It will require a specification of all available formats.	Supportability
S02	The system should have two different forms for input information: an upload link or the computer camera for stream.	Supportability

Table 4.1: Non-Functional Requirements

Implementation constraints

The analytical data used in this dissertation come from the UBFC-Phys public dataset (Sabour et al. 2021), which has considerably labeled data that helps reduce the overhead of starting the project. Ideally, a real-life dataset could be used; however, it was decided that a public dataset would be a better option to achieve high-performance results in training the model during the system's development and evaluation stages.

In addition, there is a list of allowed formats for images and videos. This list contains the following formats:

- image: bmp, jpeg, jpg, jpe, jp2, png, tiff, tif, hdr, pic
- video: aiff, asf, avi, bfi, caf, flv, gif, gxf, hls, iff, mp4, mov, qt, 3gp, mkv, mk3d, mka, mks, mpeg, mpg, mxf, nut, ogg, oma, rl2, txd, wtv

Physical constraints

One of the main requirements for building the models using Deep Learning techniques is the machine's processing units, GPUs. The GPU uses thousands of smaller and more efficient cores for a massively parallel architecture that handles multiple functions simultaneously. Modern GPUs provide superior processing power, memory bandwidth, and efficiency over their CPU counterparts, as mentioned in Section 3.9.

4.2 Model Architecture Design

Unlike the traditional software development lifecycle that meets functional and non-functional requirements, deep learning models' development aims to optimize a specific metric, for example minimizing a loss function. Therefore, a typical deep learning modeling lifecycle like in Figure 3.4 goes through many cycles before reaching a satisfactory result. It includes experimenting with different architectures, fine-tuning the hyperparameters, and trying different software libraries.

The experiments are organized in Section 6.2 to show the impacts of the parametrization in a neural network.

4.3 System Architecture Design

This dissertation emphasizes the creation of deep learning models to detect stress, so a prototype of a centralized solution was created for the role of the standard user using a single-page application (SPA) to enable the use of the classification application, whose architecture is represented in Figure 4.2.

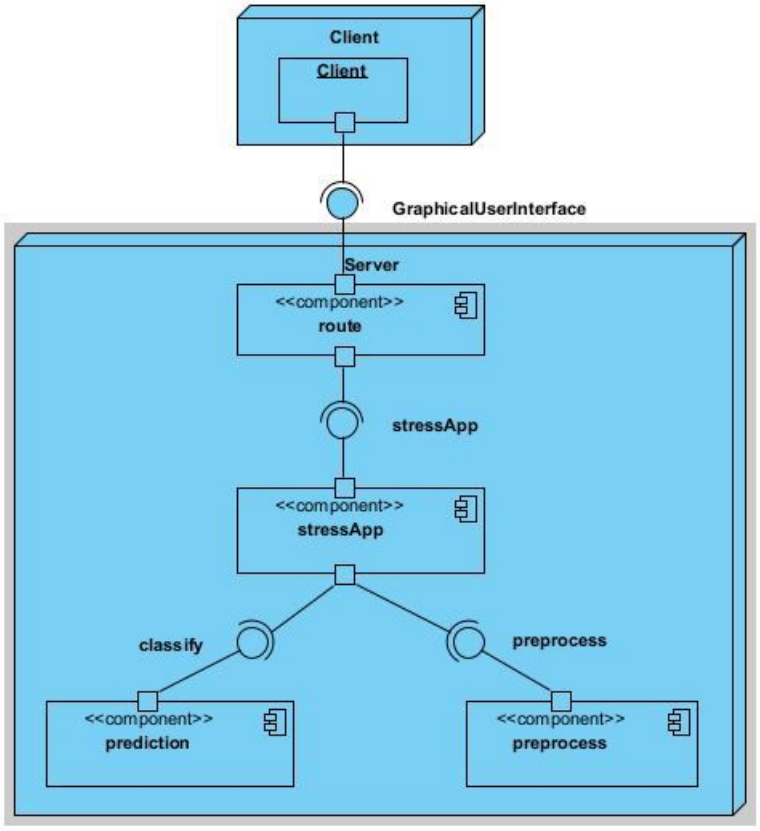


Figure 4.2: Component diagram

The Deep Learning prediction application is implemented in Flask [1], a lightweight framework for Python applications, and adopts the MVC architecture pattern for implementing User Interface (UI).

[1] https://flask.palletsprojects.com/en/2.1.x/

Considering the basic idea of Client-Server, where the Client sends a request to the Server and after manipulating the Database response, the Server then sends a response to the Client. In this approach, the Server response returns the entire page requested by the Client.

For example, in Figure 4.3 the user requests the page */stress* to do a prediction using an image and it returns all the HTML, CSS, and JS for that. This is a practical understanding of the render_template method from Flask, and the format is limited to a specific Client and RESTful APIs because it removes the backend responsibility to render templates.

Figure 4.3 presents the sequence of the application.

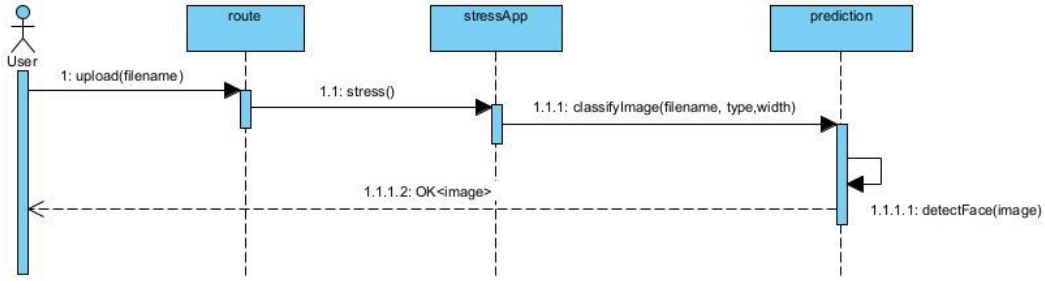


Figure 4.3: Sequence diagram

4.4 Alternative analysis

This section will focus on the alternatives of the application as a whole, that is, different combinations of components to find the best one. Alternatives:

- Centralized Solution: This solution consists of having the entire application in the same language. Such as, the data extraction is done in python language, followed by data storage in the python compatible databases such as SQLAlchemy. Creating models using python libraries such as Tensorflow or PyTorch, and the web application using the python frameworks, such as Flask. The main advantage of this solution is to allow direct interaction between the project components; as a disadvantage, there are possible conflicts between them.
- Decentralized Solution: This solution consists of having various components in different languages and locations, unlike the previous solution. Such as, the application would be developed in HTML and PHP, the models in python, the database stored in MySQL, and then data preprocessing would be done with some python algorithm or external tool. This solution has the advantage that there are no conflicts between the components, but it has the disadvantage that the interaction between the components is not so direct.

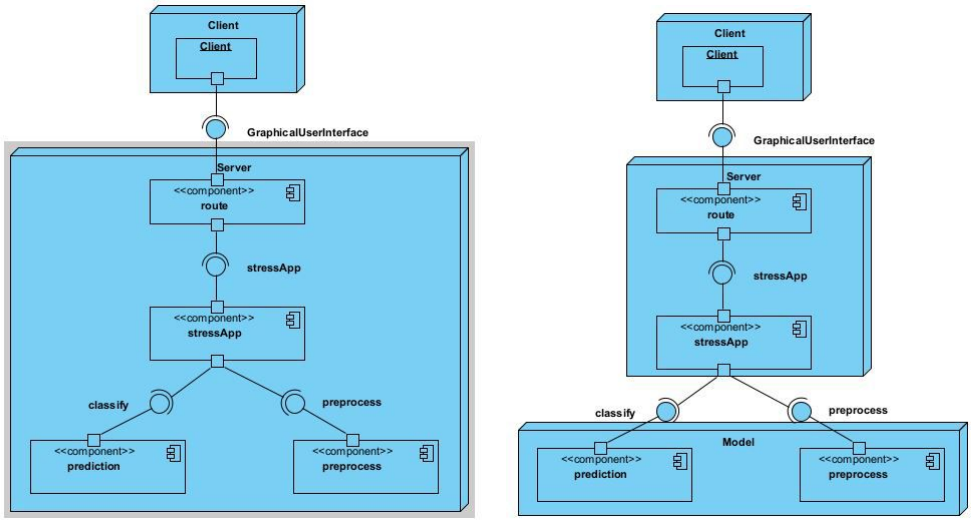


Figure 4.4: Alternative application solution

Figure 4.4 illustrates the different architectures; it is possible to verify that in the centralized solution, the elements appear in the same block and interact directly with each other, while in the decentralized solution, each component of the system is isolated in its block, being necessary to have interoperability between the blocks for the system work. As this project requires a great deal of interaction between the data and the models, the best choice is the centralized solution because it permits a direct interaction between data and models.

Chapter 5

Implementation

The content of this chapter provides visibility on how the implementation was handled considering the approach defined earlier in this document, mainly in the Design chapter. It also provides technical details on the decisions taken during the execution phase, which were previously mapped.

5.1 Dataset

This project uses the UBFC-Phys dataset (Sabour et al. 2021), which is a public multimodal dataset that has 56 participants in the experiment with a rigorous protocol inspired by the well-known Trier Social Stress Test (TSST), and was conducted in three stages: a rest, a speech, and arithmetic tasks with different levels of difficulty. During the experiment, participants were filmed with a RGB digital camera, with a Motion JPEG compression and a 35 frame per second rate. The frame resolution was of 1024x1024 pixels.

Before and after each experiment, the participants completed a form to calculate their self-reported anxiety scores. The dataset contains participants' video recordings, blood volume pulse (BVP), and electrodermal activity (EDA) signals measured during the three tasks. Also, their anxiety score was calculated before and after the experimental sessions.

Data is organized into 56 folders, corresponding to individual folders for each participant. In each folder, we find three videos, one for each task inspired by the $TrierSocialStressTest$. Only three minutes were kept for concerns of equalizing the duration of the three tasks and alleviating the dataset size.

In this study, each video frame was converted into an image, creating a dataset with 1,038,769 images. The dataset comprised 340,374 images classified as stress and 698,395 images classified as non-stress. Moreover, the experiments used subsets of 20,000 images randomly sampled due to computational capacity limitations.

5.2 Pre-processing

As mentioned earlier, in Section 4.1.1 , preprocessing is an essential part of UC2 for stress detection of the system. It allows formatting, cleaning, and transforming data for model creation and prediction. This session will present the implementations used both in the development of the model and the web application.

5.2.1 Data Integration

This step regards the applications since there are two different ways to input information, as mentioned in the non-functional requirement S02. Images, videos, and streams will need to be processed to deliver the same formatted information to the algorithm that detects faces. So, that information is fed to the prediction algorithm.

```
            # preditions (pass to pipeline models)
            if file_type in image_types:
                w = getwidth(path)
                classify_image(path, filename,'bgr')
                return render_template('stress.html',fileupload=True,
    type='image', img_name=filename,w=w)

            if file_type in video_types:
                return render_template('stress.html',fileupload=True,
    type='video',filename=filename,status=status)

    return render_template('stress.html',fileupload=False,img_name="
    imageNotFound.png")
```

Listing 5.1: Example of different datasource integration

```
    # step-1: read video in cv2
    print('Classifing Video .....')
    if status == 'stream':
        cap = cv2.VideoCapture(0) # enable video capture from stream
    else:
        cap = cv2.VideoCapture(path) # enable video capture from file

    while True:
        # read each frame from a video
        success,frame = cap.read() # read the video file
```

Listing 5.2: Example of data source input

5.2.2 Data Cleaning

This step is handled by the application and seeks to answer the non-functional requirement S01, which needs to constrain the format types of video and image available to be treated by the application. This is achieved by the two if conditions (if filetype in image types) and (if filetype in video types).

```
# Settings
image_types = ['bmp','jpeg','jpg','jpe','jp2','png','tiff','tif','hdr','
    pic']
video_types=['aiff','asf','avi','bfi','caf','flv','gif','gxf','hls','iff
    ','mp4','mov','qt','3gp','mkv','mk3d','mka','mks','mpeg','mpg','mxf',
    'nut','ogg','oma','rl2','txd','wtv']

def stress():
    if request.method == "POST":
        status = request.form['btnradio']
        if status == 'stream':
            return render_template('stress.html',fileupload=True,type='
    video',filename='null',status=status)
        else:
            f = request.files['image']
```

```
            filename=  f.filename
            file_type = filename.lower().split('.')[1]
            path = os.path.join(UPLOAD_FOLDER, filename)
            f.save(path)
            print('Uploading file .....')

            # preditions (pass to pipeline models)
            if file_type in image_types:
                w = getwidth(path)
                classify_image(path, filename,'bgr')
                return render_template('stress.html',fileupload=True,
    type='image', img_name=filename,w=w)

            if file_type in video_types:
                return render_template('stress.html',fileupload=True,
    type='video',filename=filename,status=status)

    return render_template('stress.html',fileupload=False,img_name="
    imageNotFound.png")
```

Listing 5.3: Filter image and video type in stress function

5.2.3 Data Transformation

This step applies to both models and applications; since we have already removed the unwanted file formats, it is necessary to convert videos to image sequences so the model can make predictions, as required by the non-functional requirement F02.

```
import cv2
import sys

def convert2Image(video_path):
    video_name = video_path.split('.')[0]
    vidcap = cv2.VideoCapture(video_path)
    success,image = vidcap.read()
    count = 0
    while success:
        framecount = "{number:06}".format(number=count)
        name = video_name + "_"+framecount+".jpg"
        cv2.imwrite(name, image)     # save frame as JPEG file
        success,image = vidcap.read()
        count += 1
```

Listing 5.4: Function that converts video in images

Since the application does not have constraints related to the image size, it is necessary to address this situation by standardizing the size of the input images and considering that the prediction algorithms usually have a predefined input shape.

```
# Create a function to import an image and resize it to be able to be
    used with our model
def prep_image(img, img_shape=224):
  """
  Reads an image from memory, turns it into a tensor
  and reshapes it to (img_shape, img_shape, colour_channel).
  """
  # Convert image to RGB color_mode
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
  # Resize the image (to the same size our model was trained on)
  img = tf.image.resize(img, size = [img_shape, img_shape])
  # Adjust the image for batch format
  img = tf.expand_dims(img, axis=0)
  # Normalize image
  img = img/255.
  return img
```

Listing 5.5: Example of image resize

Normalization is also crucial because it helps the algorithms converge more quickly. It is essential to normalize the images in order to use some transfer learning algorithms.

```
## Set data inputs
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_dir = "short_crop/train/"
test_dir = "short_crop/test/"
validation_dir = "short_crop/validation/"

train_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)
validation_datagen = ImageDataGenerator(rescale=1/255.)
```

Listing 5.6: Example of image normalization

5.3 Models

The development of a classification model was one of the critical steps of the project since it concerns one of the central questions of this dissertation: "It is possible to create an application capable of classifying stress in an image, with a degree of confidence, capable of being viable for a professional in the sector?"

5.3.1 Split Dataset

The recommended approach for model evaluation is to separate the input data randomly into different sets:

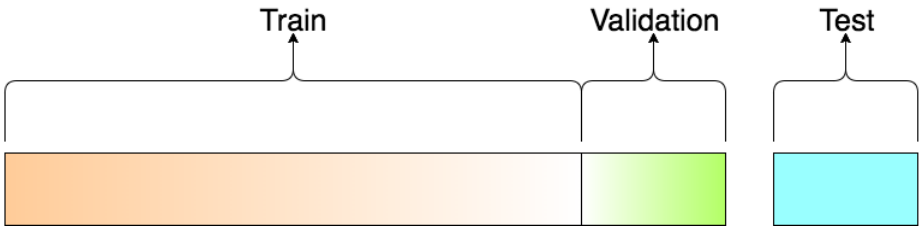


Figure 5.1: Split visualization

- Training Dataset: It is the data set used to train and make the model learn the hidden features/patterns in the data.
- Validation Dataset: It is a set of data, separate from the training set, used to validate our model performance during training. The validation process gives information that helps tune the model's hyperparameters and configurations accordingly.
- Test Dataset: It is a separate set of data used to test the model after completing the training. It provides an unbiased final model performance metric regarding accuracy, precision, and others.

Here are the functions used to split the dataset.

```
import os
import random
import shutil
import glob

source='C:/Isep/PyNotebooks/T - ImageData/all'
dest='C:/Isep/PyNotebooks/T - ImageData/all_split'
split_train = 0.7
split_test=0.2
split_validation=0.1

#### SPLIT DATASET
def split_train_test_validation(source,destination,split_train,
    split_test,split_validation):
    # Split dataset
    split_set = []
    split_set.append(split_validation)
    split_set.append(split_test)
    split_set.append(split_train)

    # Setting folders
    folders = os.listdir(source)
    directories = ['validation', 'test', 'train']

    # create new dataset
    for group in folders:
        index = 0
        origin_path = source +'/'+ group
        count = len(glob.glob1(origin_path,"*"))

        for name in directories:
            destination_path = destination + '/' + name + '/' + group

            # Split Dataset
            files = os.listdir(origin_path)
            no_of_files = round(split_set[index] * count)
            for file_name in random.sample(files, no_of_files):
                shutil.move(os.path.join(origin_path, file_name),
    destination_path)
            index +=1
```

Listing 5.7: Function to split dataset

```
import os
import random
import shutil
import glob

source='C:/Isep/PyNotebooks/T - ImageData/all'
dest='C:/Isep/PyNotebooks/T - ImageData/all_split/'
split_train = 0.7
split_test =0.2
split_validation =0.1

split_train_test_validation(source,dest,split_train,split_test,
    split_validation )
```

Listing 5.8: implementation of split function

5.3.2 Balanced Data

This step concerns the balance of the train dataset that was initially unbalanced through undersampling. This step shows a function to perform sample reduction and balancing.

```
import os
import random
import shutil
import glob

def undersample_dataset(source,destination,balance,max_size):
    folders = os.listdir(source)
    path = source + '/**/*'
    nb_files = len(glob.glob(path))

    for i,folder in enumerate(folders):
        no_of_files = 0
        origin_path=source+'/'+folder
        destination_path=destination+'/'+folder
        if not os.path.exists(destination_path):
            os.makedirs(destination_path)
        nb_folder = len(glob.glob1(origin_path,"*")) #dir is your
    directory path as string
        if balance:
            ratio = 0.5
            no_of_files = int(max_size*ratio)
        else:
            ratio = round(nb_folder/nb_files,2)
            no_of_files = int(max_size*ratio)
            print('ratio: ',ratio)

        print('numbr of files: ',no_of_files)
        files = os.listdir(origin_path)
        for file_name in random.sample(files, no_of_files):
            shutil.copy(os.path.join(origin_path, file_name),
    destination_path)
```

Listing 5.9: Example of image undersample function

```
import os
import random
import shutil
import glob

source='D:/Dataset/all_crop'
dest='D:/Dataset/short_crop_2'
balance=True
max_size=20000

undersample_dataset(source, destination, balance, max_size)
```

Listing 5.10: Reference of image undersample function

5.3.3 Face Recognition

This step applies to both models and applications, considering that the project's main interest is to analyze the expression of the faces, regardless of the context. We chose to use the Haar Classifier (*Cascade Classifier* n.d.), which allows recognizing faces, as required in the UC3. After extracting faces from the images, they are sent to the algorithm for analysis.

```
# loading models
haar = cv2.CascadeClassifier('./model/haarcascade_frontalface_default.
    xml')

def face_detect(img, color='bgr'):
    if color == 'bgr':
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    else:
        gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    faces = haar.detectMultiScale(gray,1.3,5)
    for x,y,w,h in faces:
        # Crop image
        predict_img = img[y:y+h,x:x+h] # crop image
        # Make a prediction
        prediction = pred_and_plot(classification_model, predict_img,
    class_names)
        # Draw face rectangle
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0),3)
        # Write Prediction text
        text = f"Prediction: {prediction}"
        cv2.putText(img,text,(x,y),font,1,(0,255,0),2)
    return im
```

5.3.4 Model Development

To detect stress in an image, it was necessary to create, train and test a few models so that we could have options and choose the model that performs best during the test stages. The following architectures were available to choose from:

- VGG-16
- VGG-19
- Resnet101
- Resnet50

To create a specific model to detect stress, we opt for the transfer learning technique, and these are the steps to create our model.

Create a model

In this initial stage, it is necessary to choose one of the available architectures so that a model can be customized. After selecting the architecture, it is necessary to adjust the model's top layer using a GlobalAveragePooling layer to flatten the data and a Dense layer at the top so that the model can comply with the number of classes analyzed, which in our case is a binary classification.

```
# Make the creation of our model a little easier
import tensorflow as tf

# Setup input shape, base_model and freeze the base_model layers
input_shape = (224,224,3)
base_model_vgg16 = tf.keras.applications.vgg16.VGG16(include_top=False)
base_model_vgg16.trainable=False
# Create input layer
inputs = tf.keras.layers.Input(shape=input_shape, name='input_layer')

# Give base_model the inputs and don't train it
x = base_model_vgg16(inputs, training=False)

# Pool out features on the base_model
x= tf.keras.layers.GlobalAveragePooling2D(name='
    global_average_pooling_layer')(x)

# Put a Dense layer on as the output
outputs = tf.keras.layers.Dense(1,activation='sigmoid', name='
    output_layer')(x)

# Combine the inputs with the outputs into a model
model_vgg16_1=tf.keras.Model(inputs, outputs)
```

Listing 5.11: Create the model

Compile the model

Further, it is necessary to compile the model by choosing a loss function and an optimizer. In our project, we use binary cross-entropy as a loss function and Adam as Optimizer to compile the model.

```
# Compile the model
model_vgg16_1.compile(loss='binary_crossentropy',
                      optimizer=tf.keras.optimizers.Adam(),
                      metrics=['accuracy'])
#model_vgg16_1.summary()
```

Listing 5.12: Compile the model

Fit the model

To fit the model, each custom model goes through a formal training and testing stage. In the project, we chose to train each model for 20 epochs for the most accurate test results, and in this stage, we used the train and validation dataset.

```
# fit the model
history_vgg16_1 = model_vgg16_1.fit(train_data,
                                    epochs=20,
                                    steps_per_epoch=len(train_data),
                                    validation_data=validation_data,
                                    validation_steps=len(validation_data),
                                    callbacks=[create_tensorboard_callback
    (dir_name="tensorflow_hub",

     experiment_name="vgg16_model_1"),
                                               cp_callback])
```

Listing 5.13: Fit the model

Predict with new data

In this stage, we export the model and make it available to predict unseen data. In our case, the unseen data is the test dataset.

```
# Export model
model_vgg16_1.save('/content/drive/MyDrive/Dataset/saved_models/
    vgg16_model_1.h5')

# Make a prediction
prediction = model_vgg16_1.predict(test_data)
```

Listing 5.14: Predict using the model

5.4 Application

We use the Flask framework in this project to build the server application. Flask is also extensible and does not force a particular directory structure. As part of the flask framework, the Jinja template engine dynamically builds HTML pages.

In our application, we developed for the client-side using a single page application (SPA), a web application that runs in a browser and does not require a page refresh to load new content. Django is the framework used for handling the backend of single-page applications.

5.4.1 Directory structure

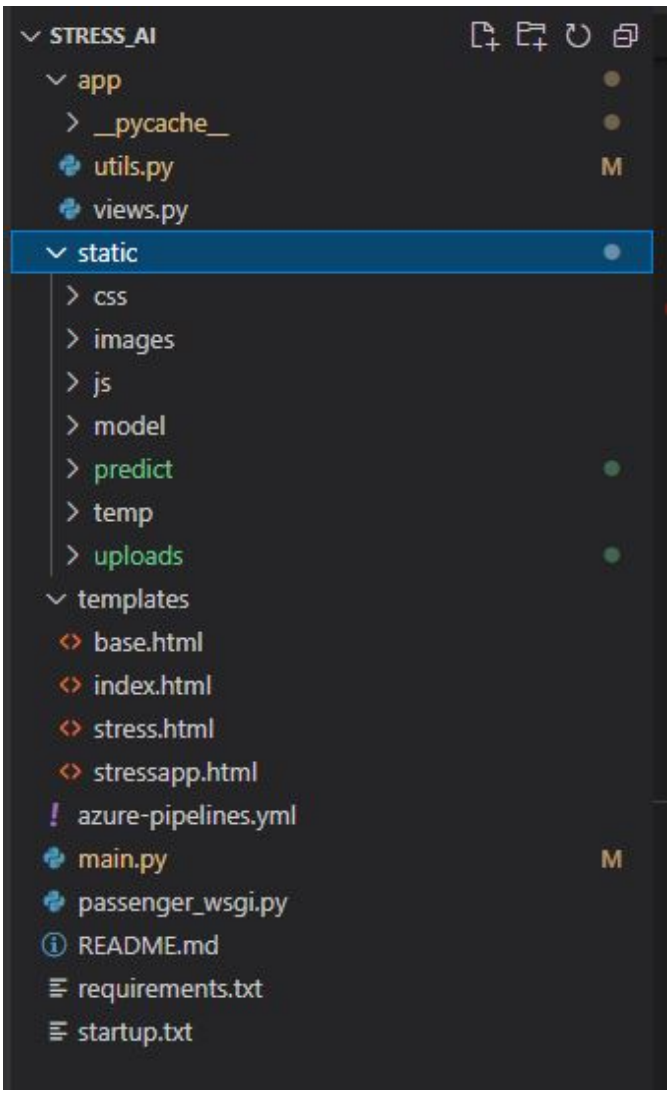


Figure 5.2: Application layout

The project directory contains:

- app/, a Python package containing the application code and files.
- venv/, a Python virtual environment where Flask and other dependencies are installed.

- Installation files tell Python how to install the project.
- Version control config, such as git. It should habitually use some version control for all the projects, no matter the size.
- Any other project files it might add in the future.

5.4.2 Creating Views

A view function is a code written to respond to application requests. Flask uses patterns to match the incoming request URL to the view that should handle it. The view returns data that Flask turns into an outgoing response.

Flask can also go the other direction and generate a URL to a view based on its name and arguments.

```
def base():
    return render_template("base.html")

def index():
    return render_template("index.html")

def stressapp():
    return render_template("stressapp.html")

def image():
    return render_template("image.html")

def video(filename,status):
    if request.method == "GET":
        print('Loading video ........ ')
        path = './static/uploads/'+filename
        print('Loading path .....', path)
        print('Loading status .....', status)
        return Response(classify_video(path,status,'bgr'), mimetype='
    multipart/x-mixed-replace; boundary=frame')
    return render_template('stress.html',fileupload=False,img_name="
    imageNotFound.png")
```

Listing 5.15: Views implemented

5.4.3 Routing URLs

After creating our views, it is necessary to map each view, so an URL must be created to map that. Modern web applications use meaningful URLs to help users. Users are more likely to like a page and come back if it uses a meaningful URL they can remember and visit a page directly. It is through these URLs that they can access the application's views.

```
# url
app.add_url_rule('/base','base',views.base)
app.add_url_rule('/','index',views.index)
app.add_url_rule('/stressapp','stressapp',views.stressapp)
app.add_url_rule('/stressapp/stress','stress',views.stress,methods=['GET
    ','POST'])
app.add_url_rule('/stressapp/stress/<filename>/<status>','video',views.
    video,methods=['GET','POST'])
```

Listing 5.16: Routing URL

5.4.4 Prediction

In machine learning, it is generally unnecessary to retrain the entire model every time new data arrives, as models can be saved and should be stable enough over time.

```
def pred_and_plot(model, image, class_names):
    """
    Imports an image located at the filename, makes a prediction on it
    with
    a trained model and plots the image with the predicted class as the
    title.
    """
    # Import the target image and preprocess it
    img = prep_image(image)
    # Make a prediction
    pred = model.predict(img)
    # Get the predicted class
    pred_class = class_names[int(tf.round(pred)[0][0])]
    return pred_class
```

Listing 5.17: Prediction function

Also, in real-time systems, the predicted data arrives precisely at the same time it is being processed, which is why the prediction should be separated into many systems.

```
def classify_video(path,status, color='bgr'):
    # step-1: read video in cv2
    print('Classifing Video .....')
    if status == 'stream':
        cap = cv2.VideoCapture(0) # enable video capture from stream
    else:
        cap = cv2.VideoCapture(path) # enable video capture from file
```

Listing 5.18: Real-time processing option

5.4.5 Templates

As the user makes requests with the app running, it will get an exception that Flask cannot find the templates. The templates in the template folder use Jinja2 syntax and have auto-escaping enabled by default.

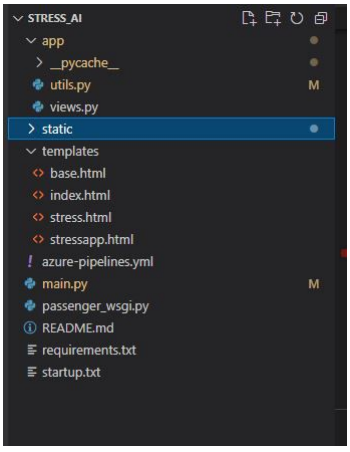


Figure 5.3: Templates folders

5.4.6 Static Files

Dynamic web applications also need static files. That is usually where we store the CSS and JavaScript files. Ideally, the web server is configured to serve them for the application, but Flask can also do that during development.

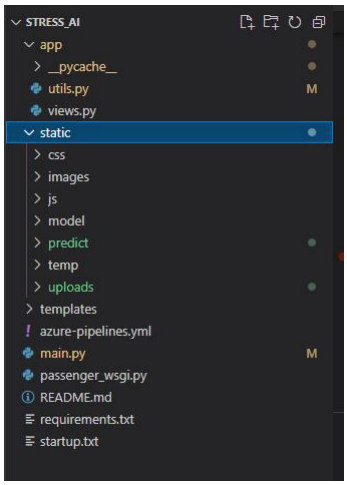


Figure 5.4: Static folders

5.4.7 Dependencies

The solution depends on a number of external packages, which are documented on the file requiremets.txt.

```
flask
opencv-python
pillow
pandas
matplotlib
ipykernel
nbconvert
notebook
tensorflow
```

Listing 5.19: Dependencies file

5.4.8 Interfaces

Creating a web application was one of the essential requirements in the project, as it is one of the first layers of interaction between the AI model and the user. The Flask web application allows the creation of a core layer, which other systems will use.

The initial page where the stress detection application is presented to the user is presented in Figure 5.5.

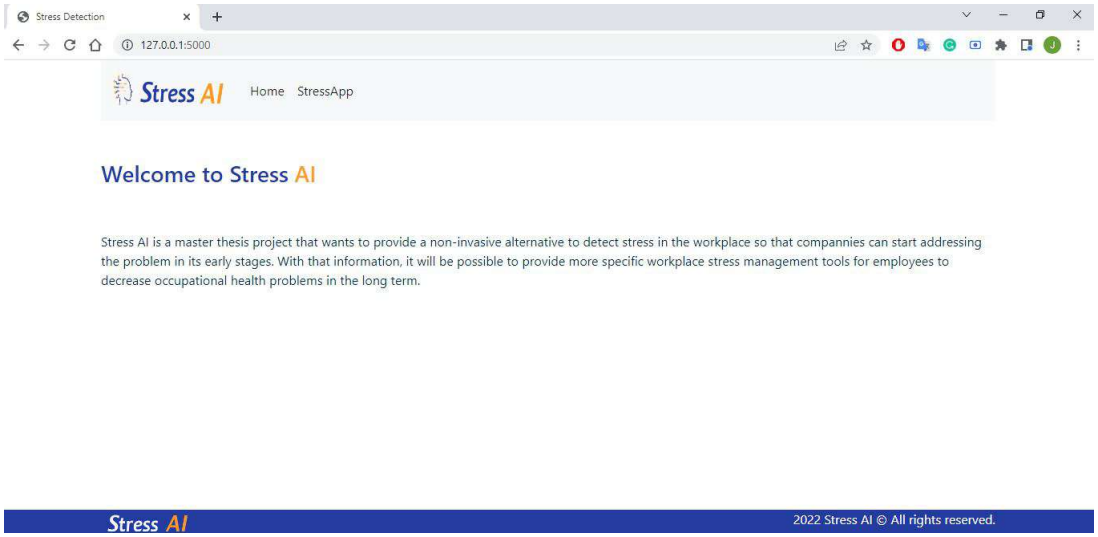


Figure 5.5: Main page

Followed by the Stress APP, in Figure 5.6, which has an initial explanation of how the application works to detect stress.

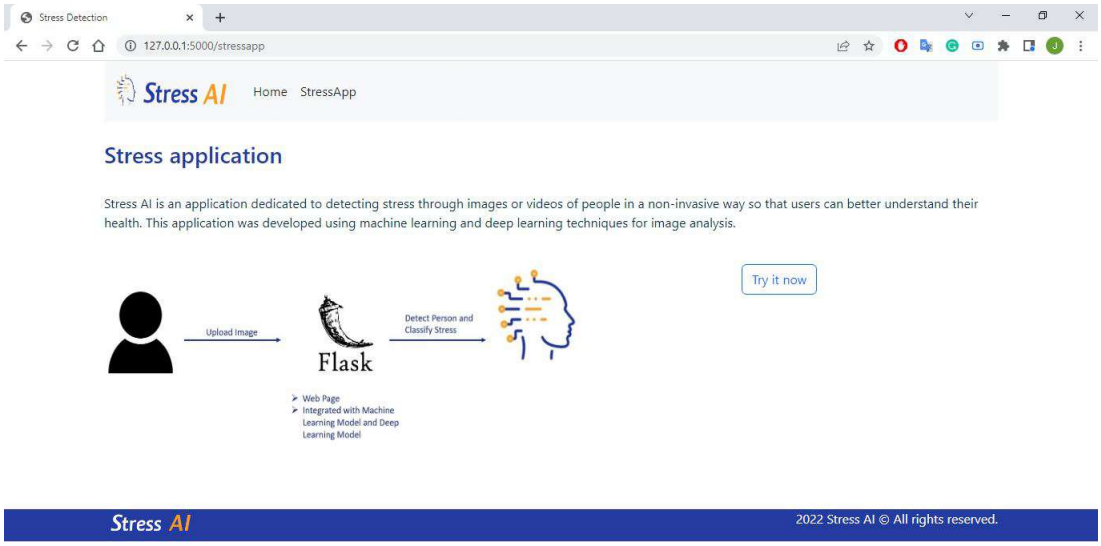


Figure 5.6: Stress App

The Stress Classification page is where the user is invited to upload an image or video and click for "Upload & Prediction". As shown in Figure 5.7, the user interface has a button to select a file and another to request a prediction of stress.

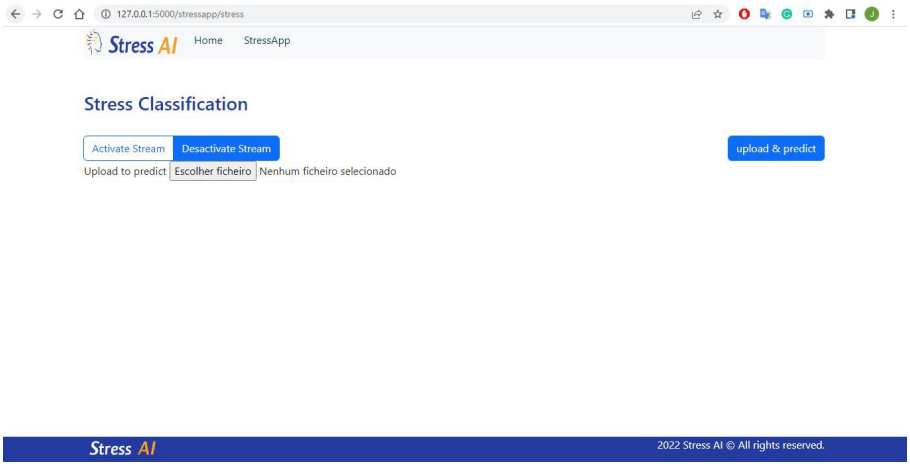


Figure 5.7: Stress Classification

Finally, this is the results interface for the stress classification, as shown in Figure 5.8.

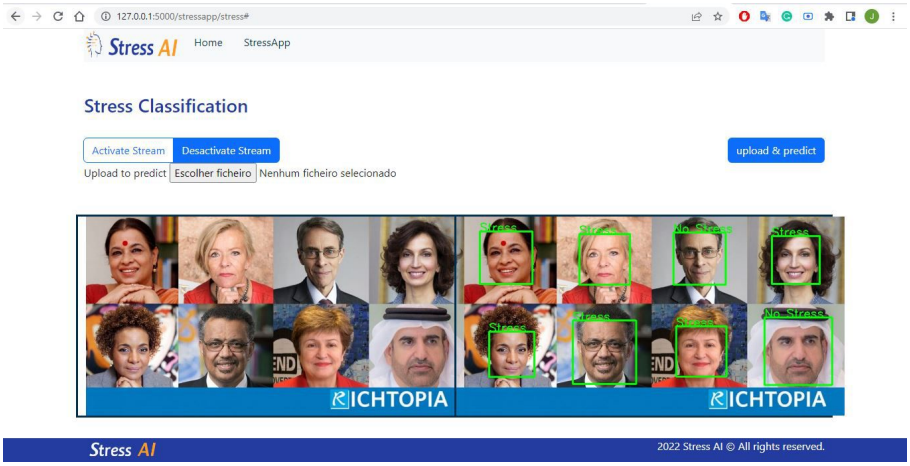


Figure 5.8: Stress App

5.4.9 Technology

In the development of this prototype, the main technologies used were the following:

- Python, version 3.8;
- Pandas, version 1.1.5;
- Flask, version 2.0.1;
- Numpy, version 1.19.3;
- Tensorflow, version 2;
- OpenCV, version 4.5.5.64

Chapter 6

Experiments and Evaluation

In this chapter, the measures that will be used in the evaluation of the experiences made in the various classification models implemented are identified, as well as the evaluation methodologies used in them.

6.1 Model Parameters

Numerous articles have been published that compare the performance of various deep learning architectures and their parameters (Srivastava et al. 2021). However, no clear solution has been found to the question which algorithm is best for a particular application. Mooney (Mooney 1996) argues that no inductive algorithm is universally better than any particular other. Wolpert and Macready (Wolpert and Macready 1997) formed the "No Free Lunch Theorems of Optimizations" in which they stipulate that there is no existing universal optimization technique, however a particular algorithm can outperform others if it is specifically designed for the structure of a particular problem.

This section assesses the experiments and results obtained by manipulating the parameters to verify, within those parameters, which are the best options and if there are options that make the objectives of the dissertation unfeasible.

6.1.1 Learning Rate

The learning rate is a parameter that allows defining the speed at which a CNN changes its weights. For high learning rates, the weights of the neural network vary constantly. However, models focusing on the last identified patterns have lower learning rates, and the weights vary slowly.

6.1.2 Batch size

The batch size is a parameter linked to the number of epochs, as it defines the number of elements the dataset uses in each epoch iteration cycle. An epoch represents a data iteration composed of several cycles, where each cycle has a size corresponding to batch size.

6.1.3 Epochs

The number of epochs means the number of times the neural network iterates over training values; that is, the number of times that the data iterates through the neural network to provoke the adjustment of the weights. The epoch number impacts the training time when a system has an adequate learning rate, making the model more accurate. Usually, the

variations between earlier epochs impact more than later because of knowledge gained from earlier epochs.

6.2 Experiments

In this section, we present the comparisons between various experiments to demonstrate the influence of parameterization in elaborating models using the Tensorflow framework. This process aims to show the impacts of the parametrization in a neural network.

6.2.1 Imbalance Data

Assessing the effects caused by imbalanced data in a prediction model is relevant to the model's performance, so two datasets were created for the data imbalance test.

Parametrization

The first dataset is balanced and has 20,000 images divided equally between the two classes, 10,000 images classified as stress and 10,000 images classified as the absence of stress. After the classification classes were balanced, the dataset was divided into three groups, 70% train, 20% test, and 10% validation, corresponding in each group: 14,000 train, 4,000 test, and 2,000 validation, as referred to in Table 6.2.

The second dataset is unbalanced and follows the original dataset's characteristics, with 67% of the images classified as the absence of stress and 33% of the images classified as stress. To experiment with the data under the same conditions as the previous dataset, we created an imbalanced dataset with 20,000 images divided into 6,600 images classified as stress and 13,400 images classified as the absence of stress, as refered in Table 6.1.

	No Stress	Stress
Balanced Dataset	10,000	10,000
Imbalanced Dataset	13,400	6,600

Table 6.1: Dataset Distribution

After the classification classes were unbalanced, the dataset was divided into three groups, 70% train, 20% test, and 10% validation, corresponding to the following amounts of images in each group. For the stress class, each group has the following number of images: 4,620 train, 1,320 test, and 660 validation. In the stress absence class, each group has the following number of images: 9,380 train, 2,680 test, and 1,340 validation, as referred to in Table 6.2.

	No Stress			Stress		
	Train	Test	Validation	Train	Test	Validation
Balanced Dataset	7,000	2,000	1,000	7,000	2,000	1,000
Imbalanced Dataset	9,380	2,680	1,340	4,620	1,320	660

Table 6.2: Dataset Split

To conduct this experiment, we set it for 20 epochs with a batch size of 32 and a learning rate of 0.001. The CNNs architecture used were VGG-16, VGG-19, Resnet101, and Resnet50. The experiment variable is the type of dataset.

Observation

The image 6.1 represents the experiment summary and shows that the accuracy reached in each architecture group has a different range, so to make consistent comparations, we divided the experiments by the architecture groups VGGNet and ResNet.

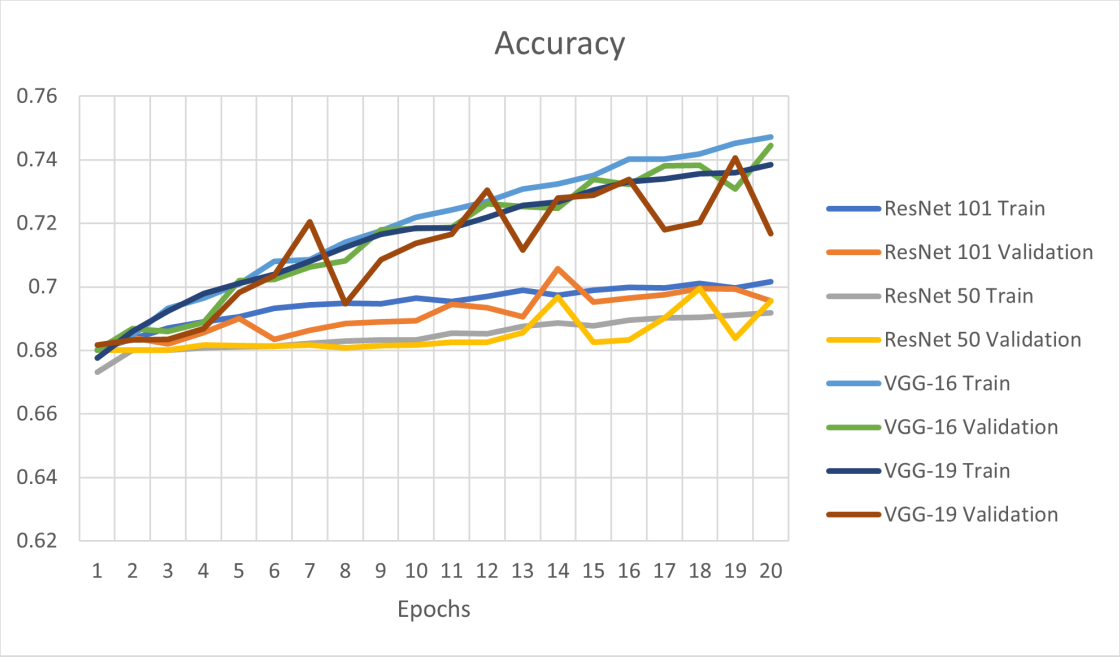
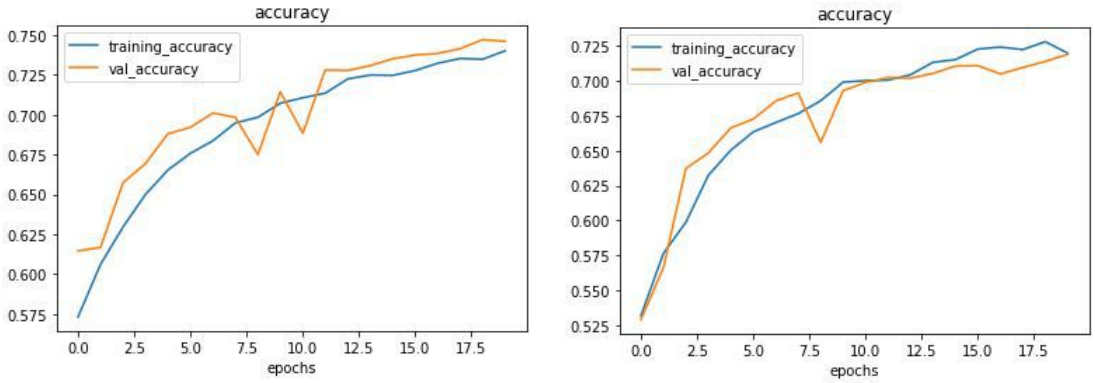


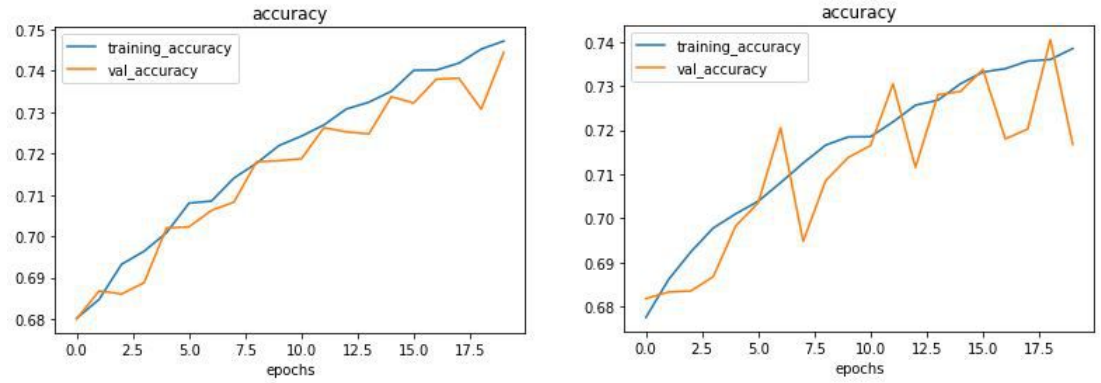
Figure 6.1: Imbalanced Experiment

VGGNet



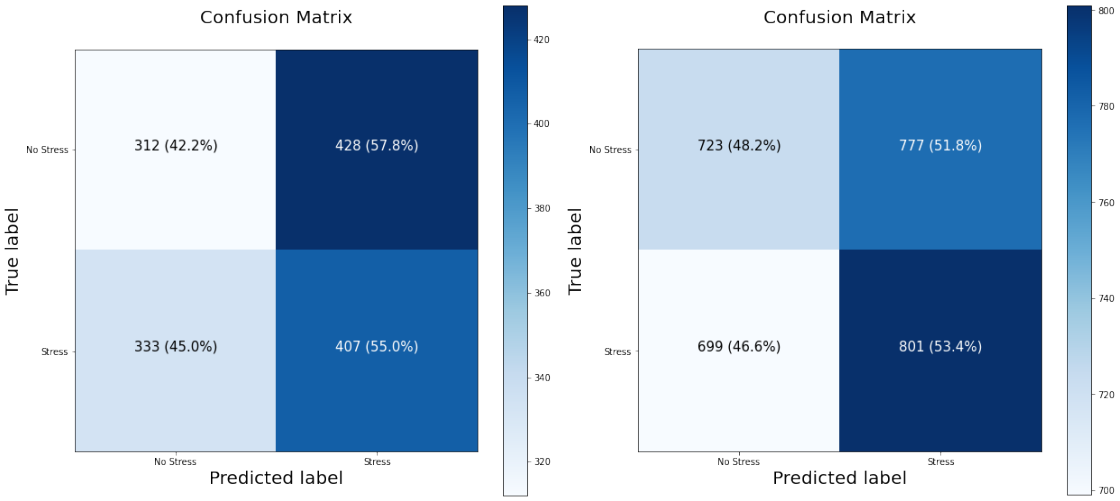
(a) VGG-16 (b) VGG-19

Figure 6.2: VGGNet for balanced experiment



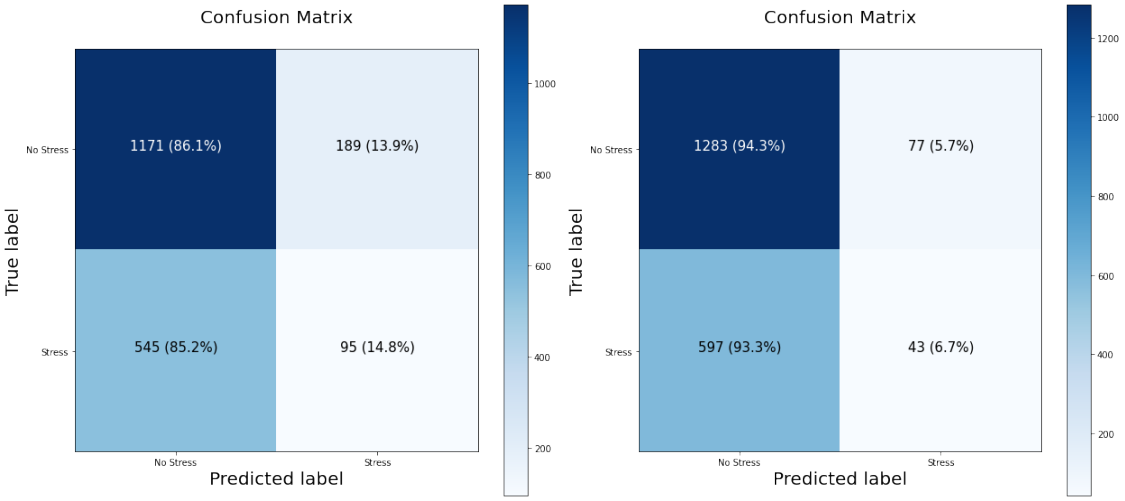
(a) VGG-16 (b) VGG-19

Figure 6.3: VGGNet imbalanced experiment



(a) VGG-16 (b) VGG-19

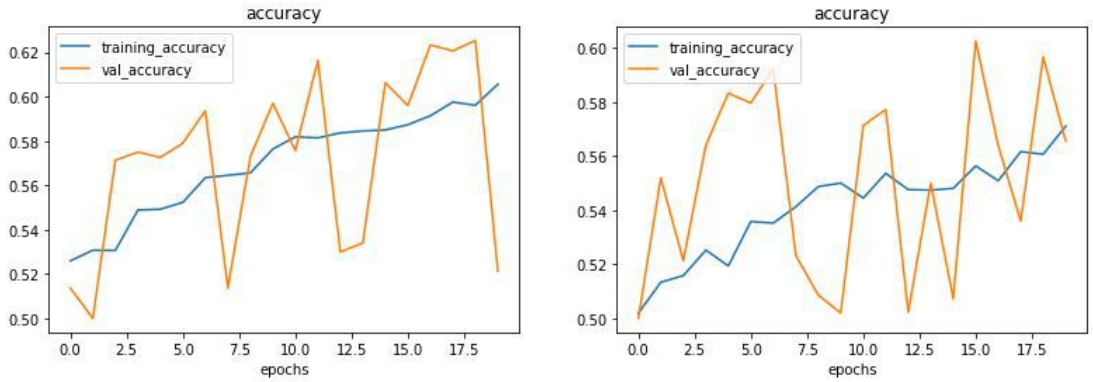
Figure 6.4: VGGNet confusion matrix for balanced experiment



(a) VGG-16 (b) VGG-19

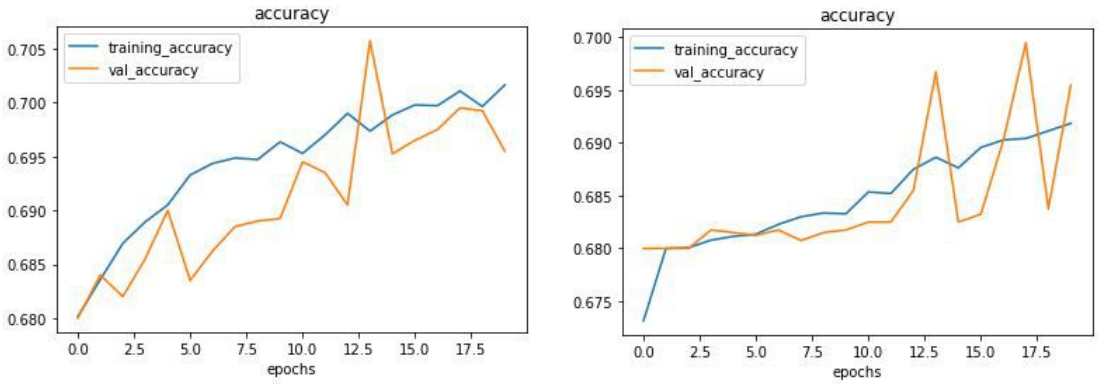
Figure 6.5: VGGNet confusion matrix of imbalanced experiment

ResNet



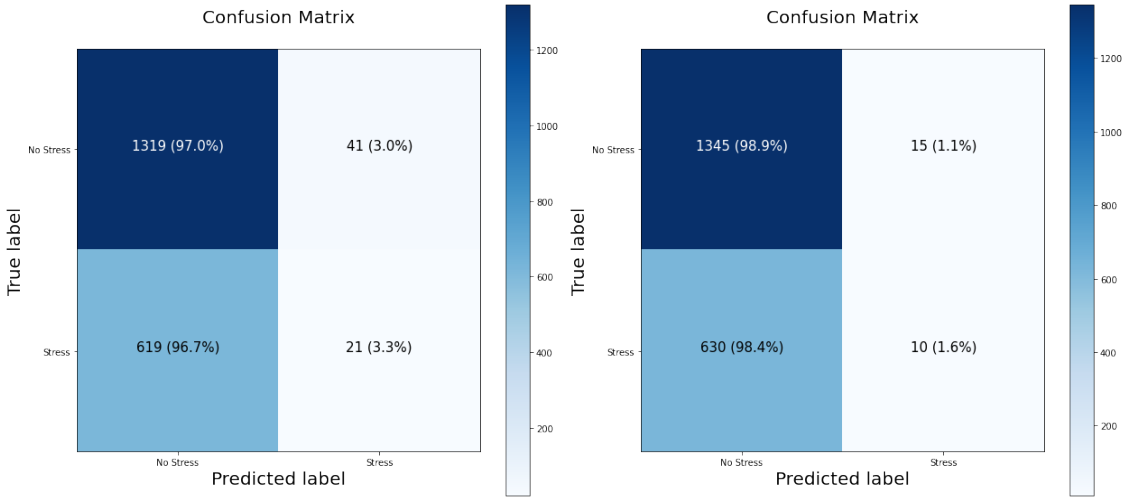
(a) Resnet101 (b) Resnet50

Figure 6.6: ResNet balanced experiment



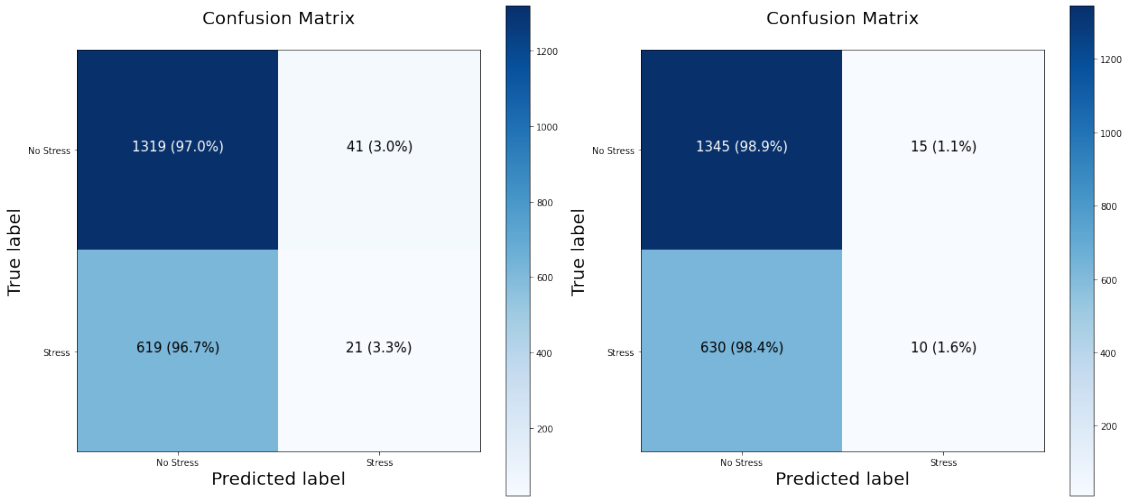
(a) Resnet101 (b) Resnet50

Figure 6.7: ResNet imbalanced experiment



(a) Resnet101 (b) Resnet50

Figure 6.8: ResNet confusion matrix of balanced experiment



(a) Resnet101 (b) Resnet50

Figure 6.9: ResNet confusion matrix of imbalanced experiment

Metrics

	Accuracy	Precision	Recall	F1
VGG-16	0.49433	0.49483	0.54267	0.51765
VGG-19	0.50800	0.50761	0.53400	0.52047
Resnet101	0.46667	0.47371	0.60067	0.52969
Resnet50	0.49867	0.49778	0.29933	0.37386

Table 6.3: Metrics of balanced experiment

	Accuracy	Precision	Recall	F1
VGG-16	0.63300	0.33451	0.14844	0.20563
VGG-19	0.66300	0.35833	0.06719	0.11316
Resnet101	0.67000	0.33871	0.03281	0.05983
Resnet50	0.67750	0.40000	0.01563	0.03008

Table 6.4: Metrics of imbalanced experiment

Results

Although accuracy is a good baseline, it does not tell the whole story, especially in class imbalanced datasets such as the ones used in this work. Precision and Recall despite giving results that explore the data more in depth, are two sides of the same coin, with Precision focusing on the amount of correct entries retrieved by the model while Recall focuses on how many correct entries were identified. Consequently, F1-score was selected as the metric of choice given its good balance between Precision and Recall while paying attention to class imbalance existent in the data.

As the imbalanced dataset's accuracy has improved in both architecture, VGGNet and ResNet, we use the F1 score to account for the number of prediction errors and the type of errors made since the F1 score combines the precision and recall metrics into a single metric. In our imbalanced dataset experiment, all architectures have an F1 score lower than 50%, which indicates that precision and recall are very low, so our model has problems classifying the data and may not find all the positives.

The confusion matrices in Figures 6.5, 6.9 show that for the imbalanced dataset model bias towards predictions labeled with "No stress" because there are more examples of absence of stress in the dataset, as refered in Table 6.2.

As such, we will use a training dataset with a slight imbalance, so the model learns the imbalance of the original dataset when training, and a test set with the original imbalance data distribution to assess models.

6.2.2 Data Augmentation

Data Augmentation is used to solve the problem of having imbalanced image classification data. Having imbalanced training data can lead to bias in the classifier in scenarios where it is not feasible to get more training data for under-represented classes. This process aims to artificially increase the amount of data by generating new data from existing data.

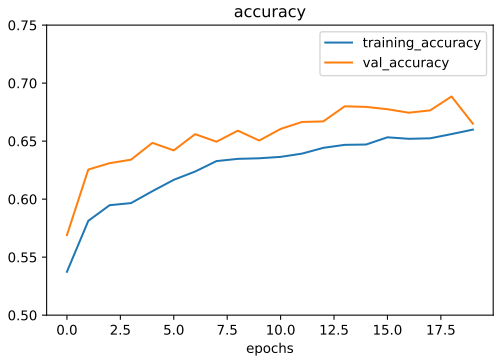
Parametrization

In this experiment, we added a data augmentation layer to the base model and trained it using the following parameters: an imbalanced dataset with 20,000 images divided 55% no stress and 45% stress; learning rate of 0,001; the batch size of 32 and 20 epochs. The CNNs architecture used were VGG-16, VGG-19, Resnet101, and Resnet50.

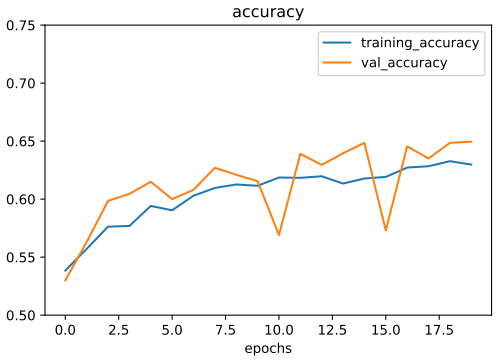
Observation

Since the accuracy reached in each architecture group has a different range, we divided the experiments by the architecture groups VGGNet and ResNet.

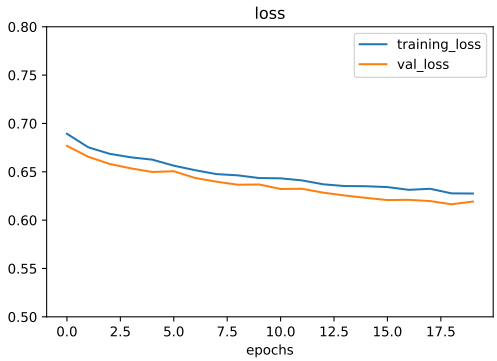
VGGNet



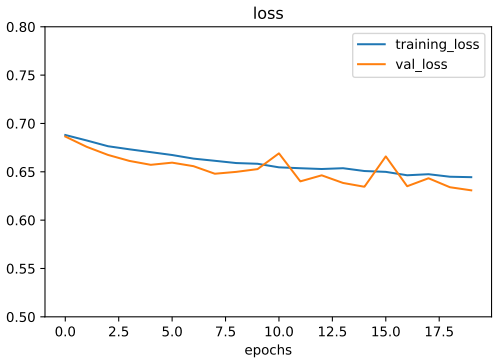
(a) VGG-16



(b) VGG-19

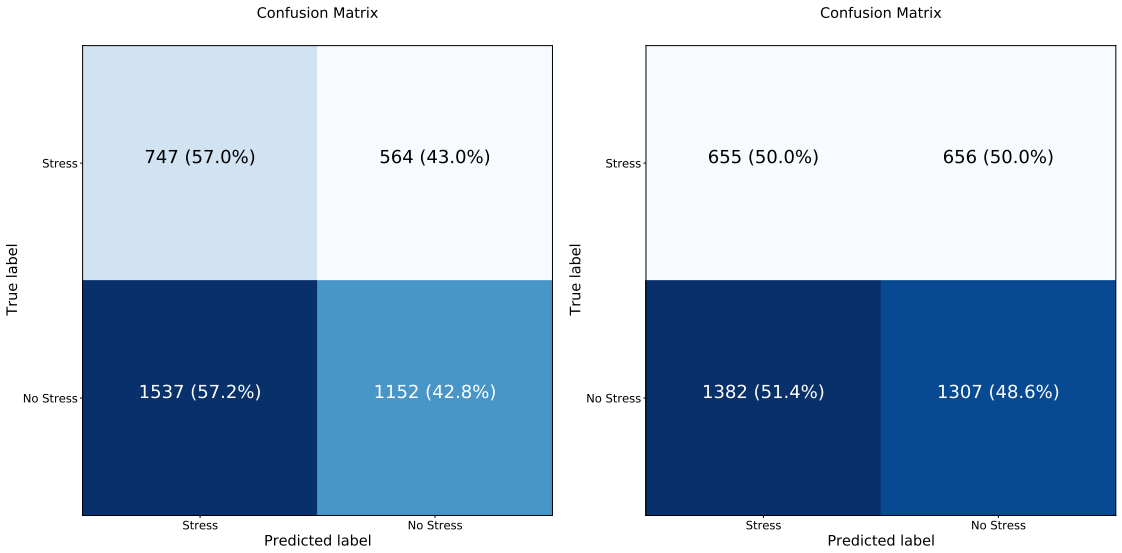


(c) VGG-16



(d) VGG-19

Figure 6.10: VGGNet for augmentation experiment

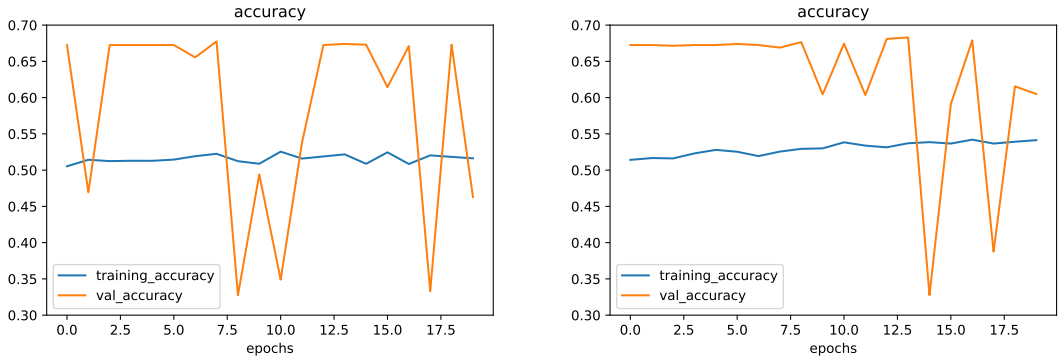


(a) VGG-16

(b) VGG-19

Figure 6.11: Confusion Matrix for VGGNet augmentation experiment

ResNet



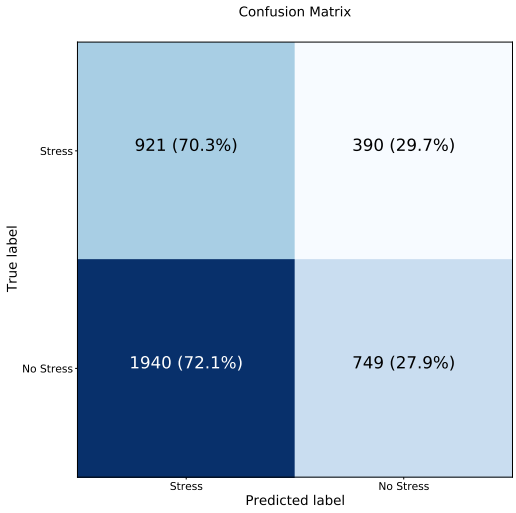
(a) Resnet101

(b) Resnet50

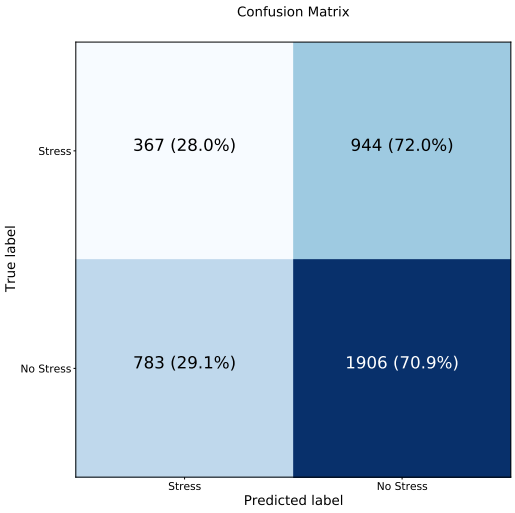
(c) Resnet101

(d) Resnet50

Figure 6.12: ResNet for augmentation experiment



(a) Resnet101



(b) Resnet50

Figure 6.13: Confusion Matrix for ResNet augmentation experiment

Metrics

	Accuracy	Precision	Recall	F1
VGG-16	0.47475	0.327058	0.569794	0.415577
VGG-19	0.4905	0.321551	0.499619	0.391278
Resnet101	0.4175	0.321915	0.702517	0.441515
Resnet50	0.56825	0.31913	0.279939	0.298253

Table 6.5: Metrics of augmented experiment

Results

Random under-sampling is an essential technique to address a class imbalance problem. However, more promising techniques try to improve the disadvantages of random-based approaches, such as data augmentation, a technique used to increase the diversity of the training set by applying random transformations, such as image rotation.

In this project, we created a data augmentation layer to address the demands of each model. Each model configured the augmentation layers individually and added them as a preprocessing layer. The results of the data augmentation layers are presented in Table 6.5. These F1 scores are the starting point of our model improvement experiments.

6.2.3 Learning Rate

The learning rate hyperparameter controls the rate or speed at which the model learns. It controls the amount of apportioned error that the model weights are updated with each time they are updated, such as at the end of each batch of training examples.

Parametrization

In this experiment, we kept the data augmentation layer with the best results from the previous experiment in the base model so that each experiment aggregates the best results yet achieved.

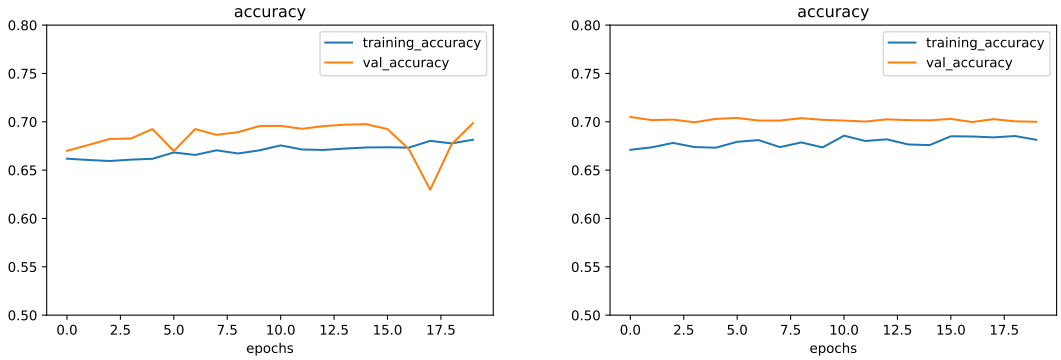
Moreover, to evaluate the influence of the learning rate on the model's performance, we fixed some parameters to the following conditions, the number of images is 20,000 divided by 55% no stress and 45% stress images, using a batch size of 32 and 20 epochs.

Regarding the evaluation of the learning rate impact in the model results, the experiments considered the respective learning rates of 0.1, 0.01, 0.001, and 0.0001.

Observation

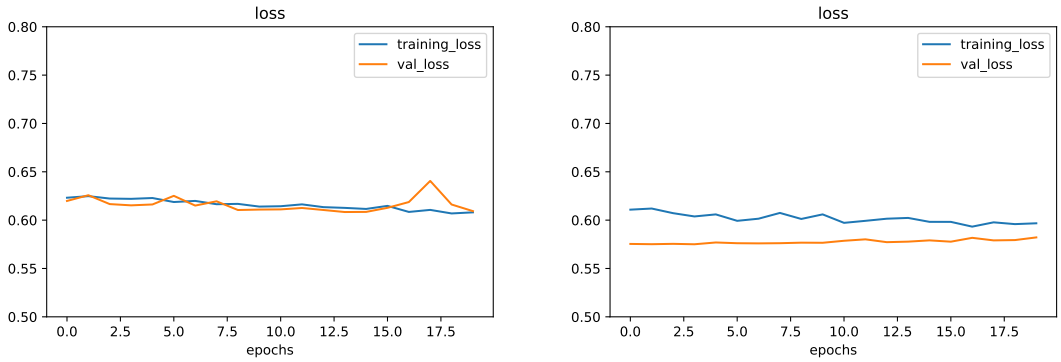
Since the accuracy reached in each architecture group has a different range, we divided the experiments by the architecture groups VGGNet and ResNet.

VGGNet



(a) VGG-16

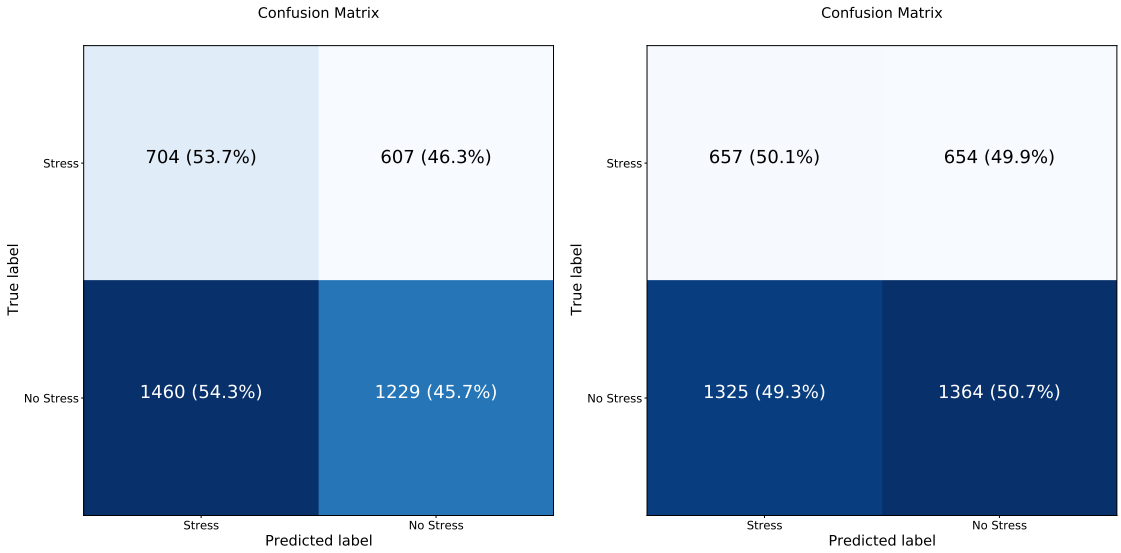
(b) VGG-19



(c) VGG-16

(d) VGG-19

Figure 6.14: VGGNet for learning rate experiment

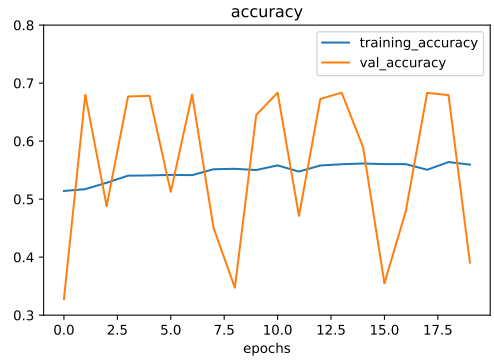


(a) VGG-16

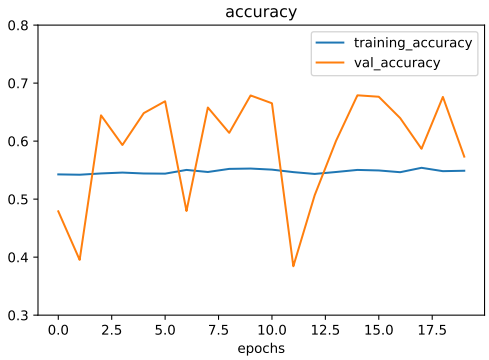
(b) VGG-19

Figure 6.15: Confusion Matrix for VGGNet learning rate experiment

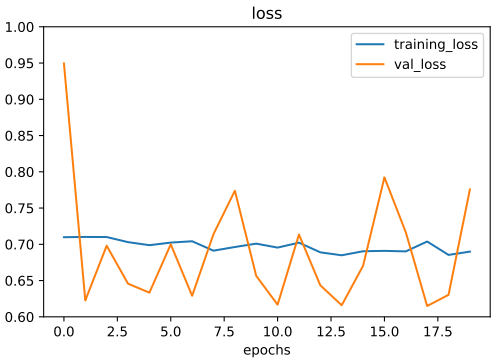
ResNet



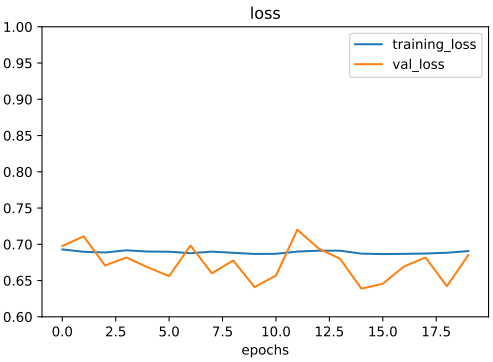
(a) Resnet101



(b) Resnet50

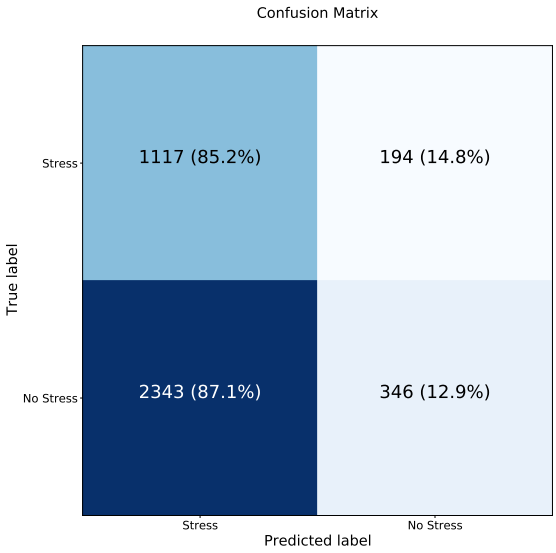


(c) Resnet101

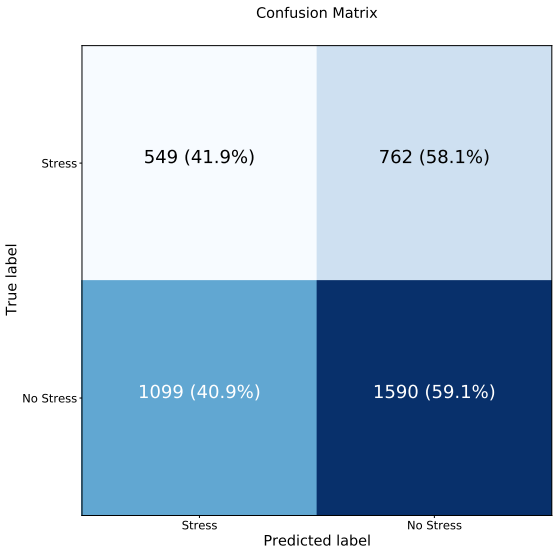


(d) Resnet50

Figure 6.16: ResNet for learning rate experiment



(a) Resnet101



(b) Resnet50

Figure 6.17: Confusion Matrix for ResNet learning rate experiment

Metrics

	Learning Rate			
	0.1	**0.01**	**0.001**	**0.0001**
Accuracy	0.481500	0.483500	0.493250	0.487500
Precision	0.321812	0.321175	0.334566	0.325296
Recall	0.525553	0.517162	0.552250	0.524790
F1	0.399189	0.396259	0.416691	0.401635

Table 6.6: Metrics VGG-16 learning rate experiment

	Learning Rate			
	0.1	**0.01**	**0.001**	**0.0001**
Accuracy	0.573000	0.511000	0.517250	0.505250
Precision	0.341327	0.329815	0.320602	0.331483
Recall	0.325706	0.476735	0.422578	0.501144
F1	0.333333	0.389894	0.364594	0.399028

Table 6.7: Metrics VGG-19 learning rate experiment

	Learning Rate			
	0.1	**0.01**	**0.001**	**0.0001**
Accuracy	0.445750	0.365750	0.480000	0.580250
Precision	0.326967	0.322832	0.316643	0.321012
Recall	0.652937	0.852021	0.506484	0.251716
F1	0.435734	0.468246	0.389671	0.282172

Table 6.8: Metrics Resnet101 learning rate experiment

	Learning Rate			
	0.1	**0.01**	**0.001**	**0.0001**
Accuracy	0.538750	0.671000	0.534750	0.600250
Precision	0.321762	0.333333	0.333131	0.297753
Recall	0.367658	0.003814	0.418764	0.161709
F1	0.343183	0.007541	0.371071	0.209590

Table 6.9: Metrics Resnet50 learning rate experiment

Results

Generally, a significant learning rate allows the model to learn faster, at the cost of arriving at a sub-optimal final set of weights. In our experiment, we considered learning rates equal to or greater than 0.01 as large, which comprises Resnet101 best result for the learning rate of 0.01, as in Table 6.8.

A lower learning rate may allow the model to learn a more optimal or globally optimal set of weights but may take significantly longer to train. The learning rates lower than 0.01 comprises VGG-16, VGG-19, and ResNet50, which have the best learning rates results of 0.001, 0.0001, and 0.001, respectively, as shown in Tables 6.6, 6.7 and 6.9.

At extremes, a learning rate that is too large will result in weight updates that are too large, and the model's performance, i.e., loss curve on the training dataset, will oscillate over training epochs, such as Figure 6.16. Divergent weights cause oscillating performance. At the same time, a learning rate that is too small may never converge or get stuck on a suboptimal solution.

6.2.4 Batch size

Batch size is one of the most critical hyperparameters to tune in modern deep learning systems. During training, larger batch sizes are often used to train models as it allows computational speedups from the parallelism of GPUs.

Parametrization

In this experiment, we chose the best results achieved by the learning rate in the previous experiment. We added them to the base model so that each experiment aggregates the best results yet achieved.

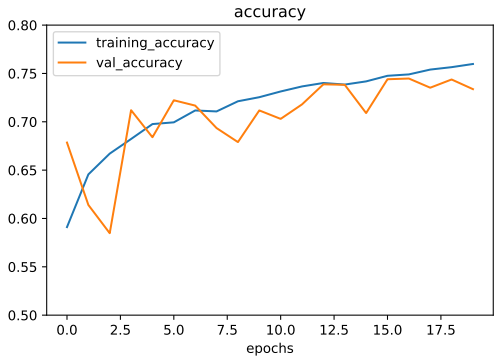
Moreover, to evaluate the influence of the batch size on the model's performance, we fixed some parameters to the following conditions, the number of images is 20,000 divided by 55% no stress and 45% stress images, augmentation layer, and 20 epochs. In the previous experiment, the learning rate achieved by the following architectures, VGG-16, VGG-19, Resnet101, and Resnet50, was 0.001, 0.0001, 0.01, and 0.001, respectively.

Regarding evaluating the batch size impact on the model results, the experiments considered the respective batches sizes 16, 32, 64, 128, and 256.

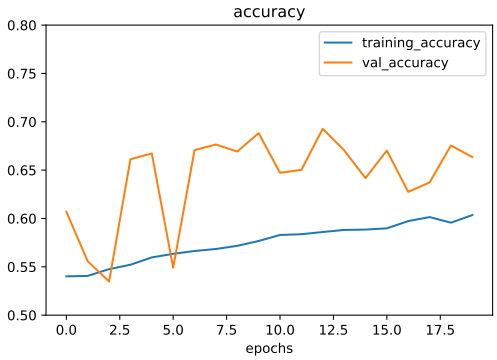
Observation

Since the accuracy reached in each architecture group has a different range, we divided the experiments by the architecture groups VGGNet and ResNet.

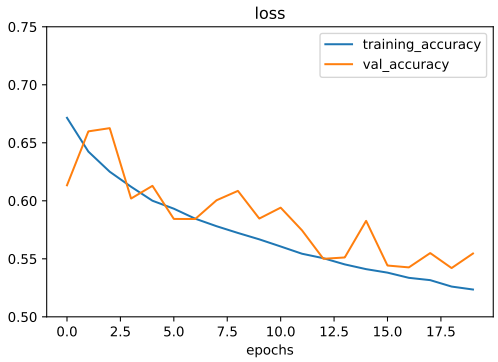
VGGNet



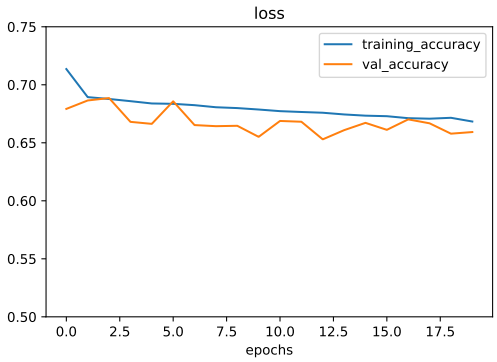
(a) VGG-16



(b) VGG-19

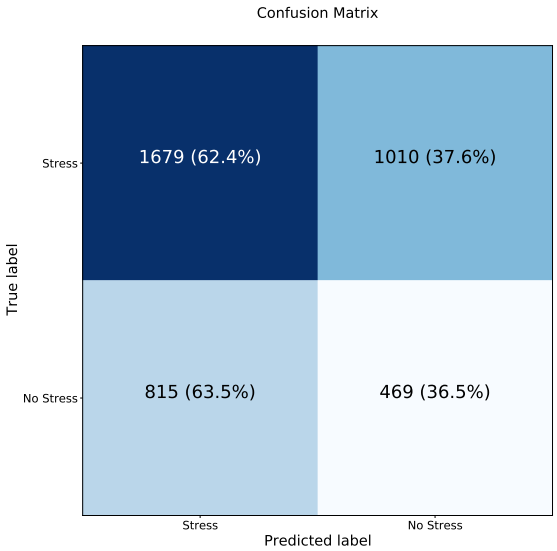


(c) VGG-16

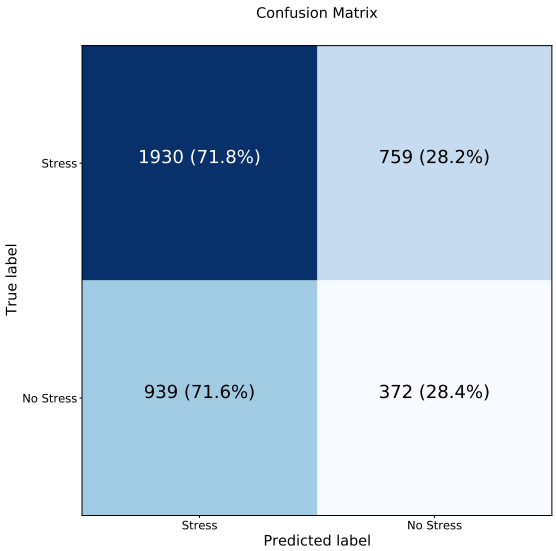


(d) VGG-19

Figure 6.18: VGGNet for batch size experiment



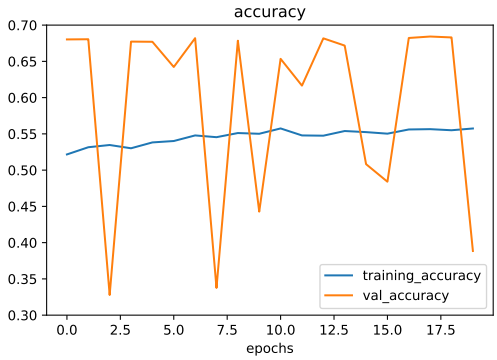
(a) VGG-16



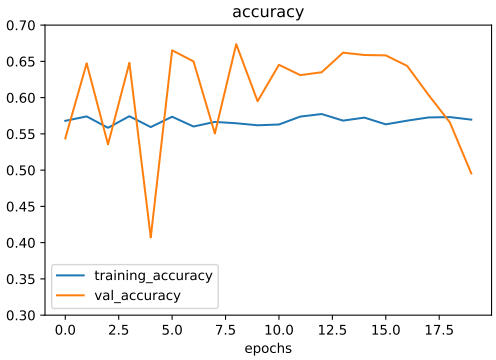
(b) VGG-19

Figure 6.19: Confusion Matrix for VGGNet batch size experiment

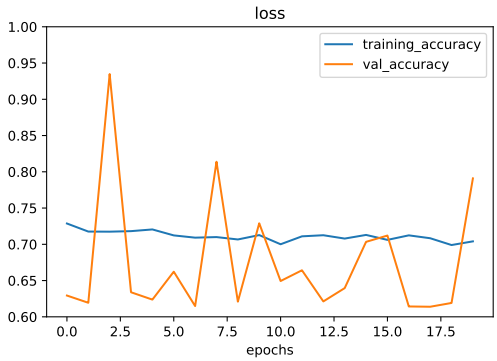
ResNet



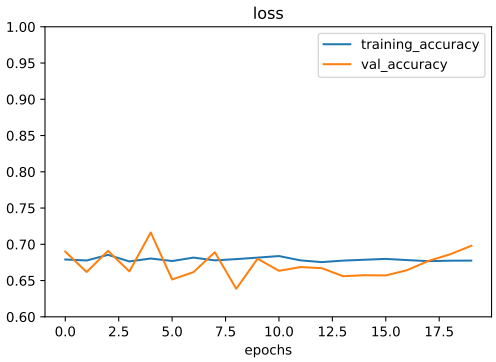
(a) Resnet101



(b) Resnet50

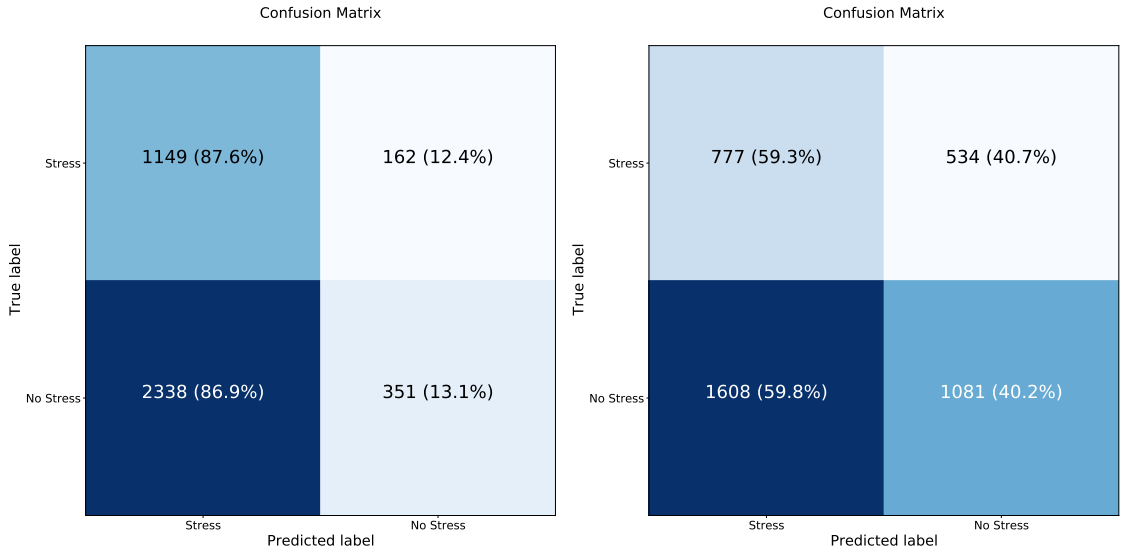


(c) Resnet101



(d) Resnet50

Figure 6.20: Resnet for batch size experiment



(a) Resnet101

(b) Resnet50

Figure 6.21: Confusion Matrix for Resnet batch size experiment

Metrics

	Batch size				
	16	**32**	**64**	**128**	**256**
Accuracy	0.543750	0.583500	0.569000	0.540500	0.555500
Precision	0.673216	0.340234	0.316444	0.320137	0.327675
Recall	0.624396	0.288330	0.271548	0.357742	0.338673
F1	0.647887	0.312139	0.292282	0.337896	0.333083

Table 6.10: Metrics VGG-16 experiment

	Batch size				
	16	**32**	**64**	**128**	**256**
Accuracy	0.575500	0.556000	0.578000	0.556000	0.569750
Precision	0.672708	0.301112	0.330333	0.317647	0.332242
Recall	0.717739	0.268497	0.279939	0.308924	0.309687
F1	0.694494	0.283871	0.303055	0.313225	0.320568

Table 6.11: Metrics VGG-19 experiment

	Batch size				
	16	**32**	**64**	**128**	**256**
Accuracy	0.375000	0.666000	0.525250	0.644750	0.627250
Precision	0.329510	0.307692	0.315327	0.309028	0.302632
Recall	0.876430	0.015256	0.382914	0.067887	0.105263
F1	0.478950	0.029070	0.345849	0.111320	0.156197

Table 6.12: Metrics Resnet101 experiment

	Batch size				
	16	**32**	**64**	**128**	**256**
Accuracy	0.656750	0.572500	0.625750	0.464500	0.574500
Precision	0.331522	0.331075	0.341837	0.325786	0.290011
Recall	0.046529	0.298246	0.153318	0.592677	0.205950
F1	0.081605	0.313804	0.211690	0.420455	0.240856

Table 6.13: Metrics Resnet50 experiment

Results

It is known that too large a batch size will lead to poor generalization, and there is an inherent dispute between the benefits of smaller and bigger batch sizes for function convergence. On one side, using a batch equal to the entire dataset guarantees convergence to the global optima of the objective function, but empirical convergence to that optima gets slower. On the other hand, smaller batch sizes have been empirically shown to have faster convergence to “good” solutions.

In our project, most of our models have better results for a smaller batch size of 16 for VGG-16, VGG-19, and ResNet101, but the downside of using a smaller batch size is that

the model is not guaranteed to converge to the global optima, as in Figure 6.20(a,c). It will bounce around the global optima; staying outside the optima will depend on the ratio of the batch size to the dataset size. On the other hand, the Resnet50 has an intermediate batch size of 128, which better regulates the loss curve bounce around the global optima, as in Figure 6.20(d).

6.2.5 Epochs

The number of epochs is a hyperparameter that defines the number of times the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

Parametrization

In this experiment, we chose the best results achieved by the batch sizes in the previous experiment. We added them to the base model so that each experiment aggregates the best results yet achieved.

Moreover, to evaluate the influence of the epochs on the model's performance, we fixed some parameters to the following conditions, the number of images is 20,000 divided by 55% no stress and 45% stress images and augmentation layer.

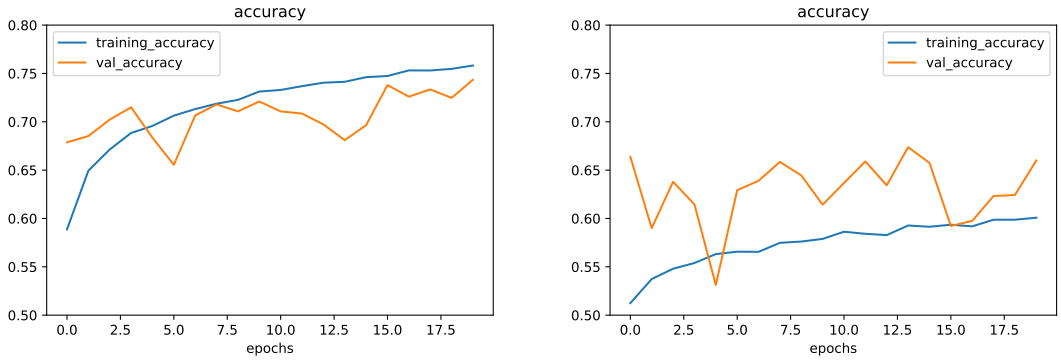
In the previous experiment, the architectures VGG-16, VGG-19, Resnet101, and Resnet50 had the best results or the learning rate of 0.001, 0.0001, 0.01, and 0.001, respectively. In addition, the batch sizes also impacted the models, and the best results obtained in each architecture were for the respective batch sizes 16, 16, 16, and 128.

Regarding evaluating the epochs' impact on the model results, the experiments considered the respective epochs 20, 40, 60, 80, and 100.

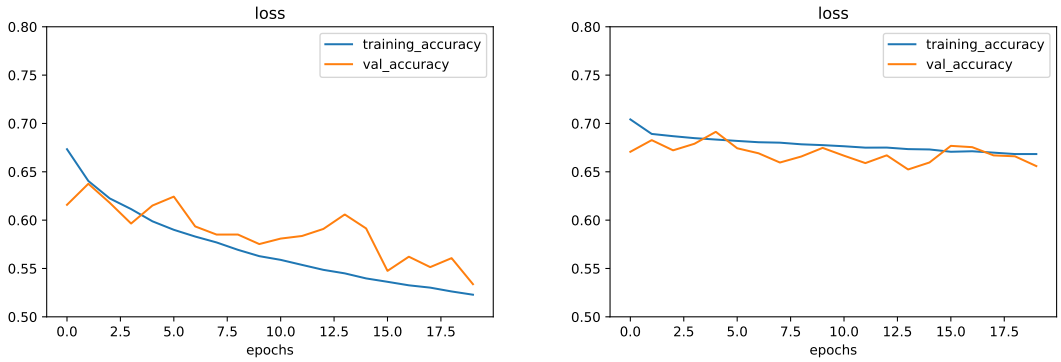
Observation

Since the accuracy reached in each architecture group has a different range, we divided the experiments by the architecture groups VGGNet and ResNet.

VGGNet

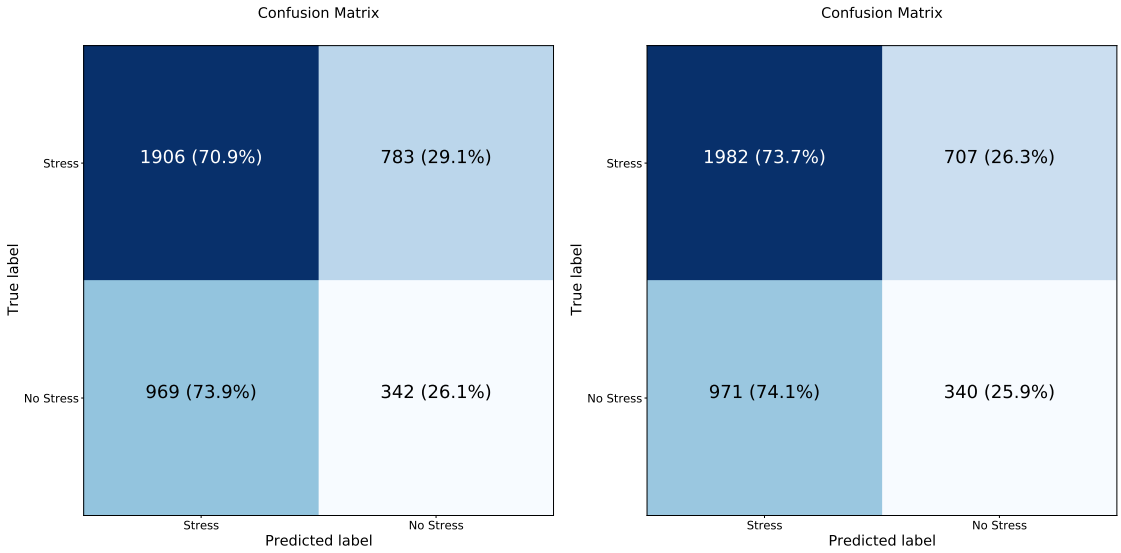


(a) VGG-16 (b) VGG-19



(c) VGG-16 (d) VGG-19

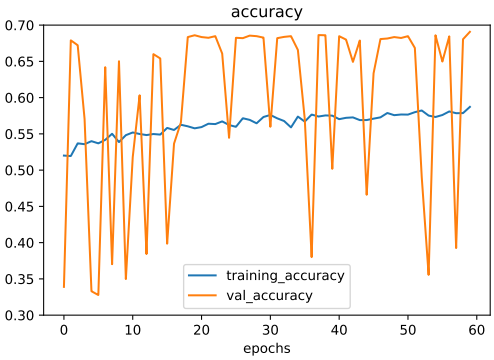
Figure 6.22: VGGNet for epochs experiment



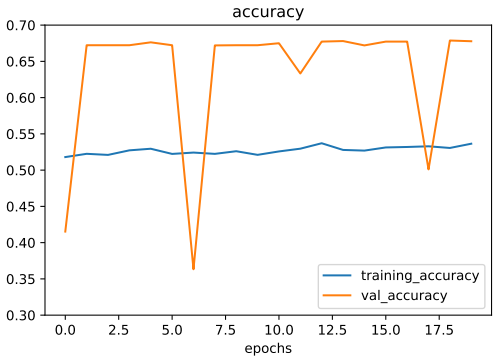
(a) VGG-16 (b) VGG-19

Figure 6.23: Confusion Matrix for VGGNet epochs experiment

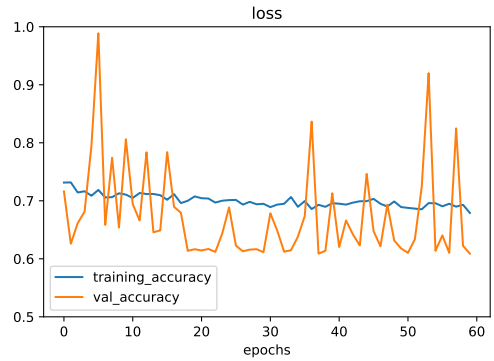
ResNet



(a) Resnet101



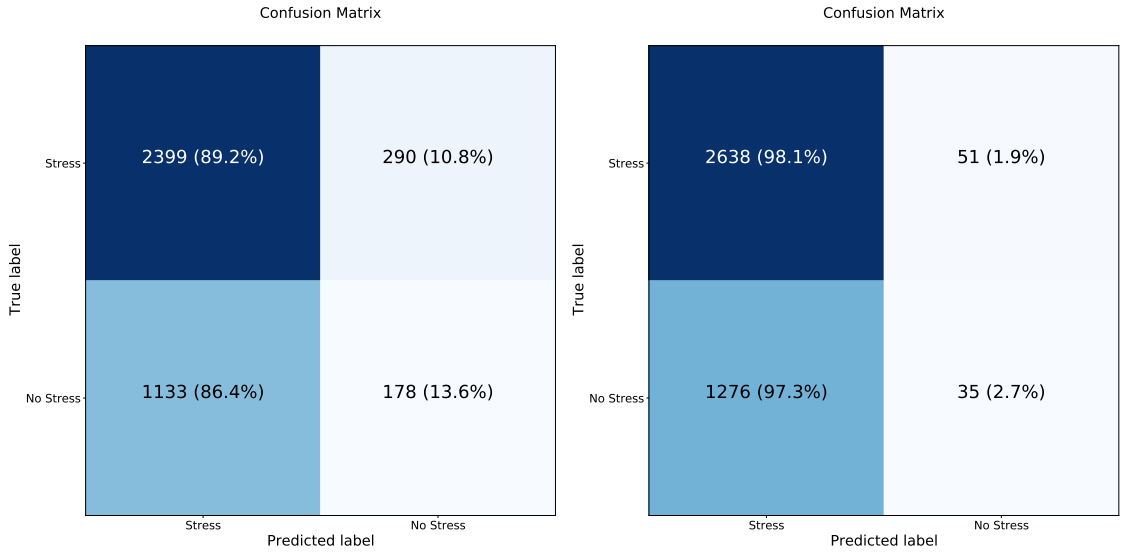
(b) Resnet50



(c) Resnet101

(d) Resnet50

Figure 6.24: Resnet for epochs experiment



(a) Resnet101

(b) Resnet50

Figure 6.25: Confusion Matrix for Resnet epochs experiment

Metrics

	Epochs				
	20	**40**	**60**	**80**	**100**
Accuracy	0.562000	0.552500	0.563500	0.543500	0.546750
Precision	0.662957	0.668541	0.665729	0.667313	0.674086
Recall	0.708814	0.663072	0.704351	0.640015	0.630718
F1	0.685119	0.665795	0.684496	0.653379	0.651681

Table 6.14: Metrics VGG-16 experiment

	Epochs				
	20	**40**	**60**	**80**	**100**
Accuracy	0.580500	0.570000	0.581500	0.536250	0.553250
Precision	0.671182	0.666552	0.679012	0.662637	0.670317
Recall	0.737077	0.721086	0.71588	0.631833	0.660097
F1	0.702588	0.692747	0.696959	0.646868	0.665168

Table 6.15: Metrics VGG-19 experiment

	Epochs				
	20	**40**	**60**	**80**	**100**
Accuracy	0.595500	0.545000	0.644250	0.532750	0.460000
Precision	0.665944	0.674568	0.679219	0.670266	0.672763
Recall	0.799182	0.624396	0.892153	0.600223	0.383042
F1	0.726504	0.648513	0.771259	0.633314	0.488152

Table 6.16: Metrics Resnet101 experiment

	Epochs				
	20	**40**	**60**	**80**	**100**
Accuracy	0.66825	0.633	0.45725	0.64075	0.63375
Precision	0.673991	0.672117	0.651817	0.676836	0.671717
Recall	0.981034	0.886575	0.413537	0.891038	0.890294
F1	0.799031	0.764593	0.50603	0.769305	0.765712

Table 6.17: Metrics Resnet50 experiment

Results

When the number of epochs used to train a neural network model is more than necessary, the training model learns "noise," or irrelevant information, within the dataset. The model becomes “overfitted,” and it cannot generalize well to new data, shown in Figure 6.22(a,c). If a model cannot generalize well to new data, then it will not be able to perform the classification or prediction tasks that it was intended for, shown in Tables 6.14, 6.15,6.16 and 6.17.

So, the configuration of epochs is seen as a kind of trade-off. On the one hand, the lower number of epochs configuration conditions the adjustment of the weights, which makes

training faster but may cause an increase in loss function and decrease in accuracy, as in Figure 6.24(a,c). On the other hand, an increase in the number of epochs increases the total training time of the model as it increases the number of iterations and the adjustment of weights which improve the loss function and the accuracy.

Nevertheless, to mitigate overfitting and increase the neural network's generalization capacity, the models were trained for an optimal number of epochs, represented in the models VGG-16, VGG-19, Resnet101, and Resnet50, with the respective values 20, 20, 60, and 20.

6.3 Evaluation

This section evaluates the results to identify the best model for stress classification using images. As previously presented, the experiments progressively show their results in each subsection using the best results of each experiment. Therefore, in this section, we justify the decision process for our best model.

As previously mentioned, due to computational capacity limitations, the original dataset that comprised 1,038,769 images was undersampled in the experiments for a 20,000 image subset representing only 1.93% of the total images available.

In addition, our original dataset had an imbalance that corresponds to an imbalance ratio of 0.49, so our original dataset classified 340,374 images as stress and 698,395 images as non-stress.

Our decision process considers the best result obtained for the F1 score in the Tables 6.14, 6.15,6.16 and 6.17, as they have a good balance between Precision and Recall, paying attention to the imbalance of data classes. The best results found for the F1 score in these tables were 0.685119, 0.702588, 0.771259, and 0.799031 for the respective architectures VGG-16, VGG-19, Resnet101, and Resnet50.

As for the accuracy and loss curves, in Figures 6.22 (a,b), 6.24 (b), they should have the smallest possible gap between the training and validation data in both the curves. The "generalization gap", as it is called, has the smallest possible distance between the training and validation in these architectures: VGG-16, VGG-19, and Resnet50.

Also, attention must be paid to the overfitting that occurs when the model has a high variance; that is, the model performs well on the training data but does not perform accurately on the evaluation set. So, the model learns "noise," or irrelevant information in the training dataset but fails to generalize to new data. This type of curve occurs when the loss of the validation dataset is less than the training dataset or when the precision in the validation dataset is greater than the training dataset. In our experiments this can be seen in Figure 6.24 (a).

Last but not least, the confusion matrix of each model should follow the same distribution of the original unbalanced dataset of 30% stress and 70% no stress, which is also used in the test set and can be seen in Figures 6.23 (a,b) and 6.25 (a).

To summarize, our best model was Resnet50 which presented the best F1 score. This is followed by reliable learning curves that present neither overfitting nor underfitting. Finally, the confusion matrix shows that our model has very high Recall, which means that the classifier predicted almost all positives as positives.

6.4 Comparison with other systems

In this section, we compare our results to another similar study of stress classification that uses the same dataset UBFC-Phys (Sabour et al. 2021). The UBFC-Phys dataset is a large, multimodal, and publicly shared dataset that was collected to respond to the analysis of the impact of social stress on physiological responses. The set consists of 168 videos, measures of contact blood volume pulse (BVP) and electrodermal activity (EDA), and a self-reported state anxiety score.

As previously mentioned, due to computational capacity limitations, the original dataset that comprised 1,038,769 images converted from the video frames was undersampled in the experiments for a subset o 20,000 images.

So, our study aims to identify the efficiency of deep learning algorithms in contrast to machine learning algorithms when recognizing stress remotely.

In the reference articles (Sabour et al. 2021), the machine learning classifiers used were: Support Vector Machine (SVM) with a linear kernel, SVM with Radial Basis Function (RBF), Logistic Regression (Log Reg), and K-Nearest Neighbors (KNN). The four classifiers were used to predict contact and remote pulse rate variability (PRV) measures, electrodermal activity (EDA) measures, a combination of contact PRV and EDA features, and facial expression.

Type	Classifiers	Accuracy (%)	F1 score (%)
Machine Learning	SVM - linear kernel	54.42	-
	SVM - RBF kernel	53.29	-
	Log Reg	54.68	-
	KNN	55.07	-
Deep Learning	VGG-16	56.20	68.51
	VGG-19	58.05	70.26
	Resnet101	64.43	77.13
	Resnet50	66.83	79.90

Table 6.18: Algorithm comparison

Table 6.18 contains a comparison between the algorithms used in the reference article and the results obtained in our experiments.

In the reference study, the evaluation metrics only consider accuracy; however, due to the existing imbalance in the original dataset, our study considers the F1 score a more appropriate metric to assess results rather than accuracy. In Table 6.18, our best model was the Resnet50 with an accuracy of 66.83% and an F1 score of 79.93%.

As a result, our Resnet50 model outperformed the best machine learning classifier (KNN) in the reference article, with an accuracy of 66.83% over an accuracy of 55.07%. Moreover, we can affirm that despite the "video-based PRV modality has proven to be reliable in recognizing the stress state", deep learning models are also reliable in recognizing the stress levels using video-based features.

Finally, it proves that images and videos can be used in stress detection even though it is not a simple activity. It demands a multimodal experiment to deal with all the complexity involved in stress detection.

Chapter 7

Conclusion

This chapter presents the conclusions and hypotheses resulting from the preparation of the present work, as well as an approach to potential future work that could be carried out within the scope of the stated theme and objectives.

7.1 Overview

Stress is often described as a complex psychological, physiological, and behavioral state triggered by perceiving a significant imbalance between the demands placed on the person and their perceived ability to meet those demands.

With the sudden onset of stress, the muscles tense up all at once and then release their tension when the stress passes. Chronic stress causes the muscles in the body to be in a more or less constant state of guardedness. Thus, stress also affects the respiratory, digestive, and muscular systems, especially the facial muscles, where it causes facial tension.

Darwin argued that facial expressions are universal; that is, most emotions are expressed in the same way on the human face, regardless of race or culture. In several recent studies, there are reports on findings of facial signs and expressions that can provide information about stress analysis and classification. Nevertheless, extracting craft resources requires considerable time.

With the development of deep learning, visual feature extraction has become a standard method for video emotion recognition. This study focuses on automated stress identification using deep learning.

The main challenge of creating a stress detector application was to formulate a model capable of learning how to detect stress reliably. Many challenges were faced when deciding on the architecture and its parametrization to achieve valuable results. Nevertheless, the web application developed for stress detection achieved its goals.

Like the valuable achievements, numerous lessons have been learned about using and applying deep learning terms, concepts, and knowledge. The knowledge acquired allowed us to evaluate the results using state-of-the-art architectures critically. Also, the new libraries enable us to create models capable of making predictions with a high F1 score. At last, many lessons were learned on how to deal with unbalanced data.

7.2 Limitations and improvements

The main limitation found was the processing capacity of the GPU used. Our GPU only had 2GB of dedicated memory for processing, and this caused many bottlenecks in processing. Therefore, our proposal to overcome this obstacle is to use a cluster solution or GPUs with greater processing capacity.

Secondly, there was a limitation in the deployment of the application, which required a particular host server for applications that use machine learning and deep learning resources. So, assessing the server's configuration before deploying the application on a regular web hosting service is essential.

Therefore, as the objectives were accomplished, one could think there was no space for improvement, but that is not the case. We suggest using a more extensive or full dataset, as the original dataset comprises 1,038,769 images. Also, fine-tuning could be important in the search for the key parameters that can be used to create a dedicated stress detection model.

7.3 Final thoughts

When dealing with algorithms that run millions of iterations of non-trivial procedures, it is vital to utilize every last bit of performance available. This project allowed me to understand better the performance impacts of the decisions applied to the design and implementation of applications, understanding the cost each operation might hold upon the execution of the entire algorithm.

I am satisfied with the project's outcome, as finding a model that detects stress using images is a reality. I consider the final product acceptable, but the application could be incremented with more information for the user or the use of more information available in the multimodal dataset.

The investigation of topics such as the one presented focuses on helping not only companies but also occupational health technicians to be able to identify stress in people in its early stages. It would result in a more significant set of information on which companies could act to prevent occupational health problems.

Bibliography

Abadi, Martín et al. (2016). "{TensorFlow}: A System for {Large-Scale} Machine Learning". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283.

Ahmed, Ejaz, Michael Jones, and Tim K Marks (2015). "An improved deep learning architecture for person re-identification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3908–3916.

Alasadi, Suad A and Wesam S Bhaya (2017). "Review of data preprocessing techniques in data mining". In: *Journal of Engineering and Applied Sciences* 12.16, pp. 4102–4107.

Allee, Verna (2002). "A value network approach for modeling and measuring intangibles". In: *Transparent Enterprise Conference*.

– (2006). "Value network mapping basics". In: *ValueNet Works™ Fieldbook*.

Almeida, José and Fátima Rodrigues (2021). "Facial Expression Recognition System for Stress Detection with Deep Learning". In.

Azevedo, Nicolas (Nov. 2021). *Data Preprocessing: 6 Techniques to Clean Data*. Online; accessed 10-June-2022. url: `https://www.scalablepath.com/data-science/data-preprocessing-phase`.

Bhujel, Anil and Dibakar Raj Pant (2017). "Dynamic convolutional neural network for image super-resolution". In: *Journal of Advanced College of Engineering and Management* 3, pp. 1–10.

Bobade, Pramod and M. Vani (2020). "Stress Detection with Machine Learning and Deep Learning using Multimodal Physiological Data". In: *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 51–57. doi: `10.1109/ICIRCA48905.2020.9183244`.

Bottou, Léon, Frank E Curtis, and Jorge Nocedal (2018). "Optimization methods for large-scale machine learning". In: *Siam Review* 60.2, pp. 223–311.

Cascade Classifier (n.d.). Online; accessed 08-June-2022. url: `https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html`.

Chandrasekar, Priyanga and Kai Qian (2016). "The Impact of Data Preprocessing on the Performance of a Naive Bayes Classifier". In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2, pp. 618–619. doi: `10.1109/COMPSAC.2016.205`.

Chapman, Pete et al. (2000). "CRISP-DM 1.0: Step-by-step data mining guide". In: *SPSS inc* 9, p. 13.

Chatfield, Ken et al. (2014). "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531*.

Chawla, Nitesh V et al. (2002). "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16, pp. 321–357.

Chollet, Francois (2021). *Deep learning with Python*. Simon and Schuster.

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Dinghofer, Kai and Frank Hartung (2020). "Analysis of Criteria for the Selection of Machine Learning Frameworks". In: *2020 International Conference on Computing, Networking and Communications (ICNC)*, pp. 373–377. doi: 10.1109/ICNC47757.2020.9049650.

Dubovikov, Kirill (2017). "PyTorch vs TensorFlow-spotting the difference". In: *Medium-Towards Data Science*.

Factors, Facts & (Feb. 2022). *Global Workplace Stress Management Market Share Will Rich USD 15.4 Billion by 2026 at 8.7% CAGR Growth: Says Facts & Factors*. Online; accessed 13-February-2022. url: https://www.globenewswire.com/news-release/2022/02/07/2380177/0/en/Global-Workplace-Stress-Management-Market-Share-Will-Rich-USD-15-4-Billion-by-2026-at-8-7-CAGR-Growth-Says-Facts-Factors.html.

García, Salvador, Julián Luengo, and Francisco Herrera (2015). *Data preprocessing in data mining*. Vol. 72. Springer.

Giannakakis, G. et al. (2017). "Stress and anxiety detection using facial cues from videos". In: *Biomedical Signal Processing and Control* 31, pp. 89–101. issn: 1746-8094. doi: https://doi.org/10.1016/j.bspc.2016.06.020. url: https://www.sciencedirect.com/science/article/pii/S1746809416300805.

Gonzalez, Rafael C and Richard E Woods (2007). "August 31". In: *Digital Image Processing Third Edition. Prepared by Pearson Education*, pp. 142–161.

Goodman, Noah (2017). *Uber ai labs open sources pyro, a deep probabilistic programming language*.

Goyal, Priya et al. (2017). "Accurate, large minibatch sgd: Training imagenet in 1 hour". In: *arXiv preprint arXiv:1706.02677*.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

– (2016a). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

– (2016b). "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer, pp. 630–645.

Huang, Gao et al. (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR, pp. 448–456.

Jia, Yangqing et al. (2014). "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678.

Joseph, Anup (Oct. 2021). *Unlocking Resnets*. Online; accessed 10-January-2022. url: https://medium.com/analytics-vidhya/opening-resnets-46bb28f43b25.

Kalia, Madhu (2002). "Assessing the economic impact of stress[mdash]The modern day hidden epidemic". In: *Metabolism* 51.6, Part B, pp. 49–53. issn: 0026-0495. doi: https://doi.org/10.1053/meta.2002.33193. url: https://www.sciencedirect.com/science/article/pii/S0026049502437924.

Koldijk, Saskia et al. (2014). "The swell knowledge work dataset for stress and user modeling research". In: *Proceedings of the 16th international conference on multimodal interaction*, pp. 291–298.

Krawczyk, Bartosz (2016). "Learning from imbalanced data: open challenges and future directions". In: *Progress in Artificial Intelligence* 5.4, pp. 221–232.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25.

Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich (2016). "Learning representations for automatic colorization". In: *European conference on computer vision*. Springer, pp. 577–593.

LeCun, Yann A et al. (2012). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.

Li, Mu et al. (2014). "Efficient mini-batch training for stochastic optimization". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670.

Li, Russell and Zhandong Liu (2020). "Stress detection using deep neural networks". In: *BMC Medical Informatics and Decision Making* 20.11, pp. 1–10.

Martikainen, Atte (2017). "Front End of Innovation in Industrial Organization". In.

Maxion, Roy A and Shing-hon Lau (2018). "Facial expressions during stress and nonstress conditions: A benchmark video collection". In: *Tech. Rep. CMU-CS-18–110, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA 15213*.

Mhon, Gant Gaw Wutt and Nang Saing Moon Kham (2020). "ETL Preprocessing with Multiple Data Sources for Academic Data Analysis". In: *2020 IEEE Conference on Computer Applications(ICCA)*, pp. 1–5. doi: 10.1109/ICCA49400.2020.9022824.

Miao, Hui et al. (2017). "Modelhub: Deep learning lifecycle management". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, pp. 1393–1394.

Michael, Porter E et al. (1985). "Competitive advantage: creating and sustaining superior performance". In: *New York: FreePress*.

Mooney, Raymond J (1996). "Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning". In: *arXiv preprint cmp-lg/9612001*, pp. 82–91.

Mu, Enrique and Milagros Pereyra-Rojas (2017). "Understanding the analytic hierarchy process". In: *Practical decision making*. Springer, pp. 7–22.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Icml*.

Nazaré, Tiago S et al. (2017). "Deep convolutional neural networks and noisy images". In: *Iberoamerican Congress on Pattern Recognition*. Springer, pp. 416–424.

networks, Popular (Nov. 2018). *VGG16 – Convolutional Network for Classification and Detection*. Online; accessed 10-January-2022. url: https://neurohive.io/en/popular-networks/vgg16/.

Nicola, Susana, Eduarda Pinto Ferreira, and JJ Pinto Ferreira (2012). "A novel framework for modeling value for the customer, an essay on negotiation". In: *International Journal of Information Technology & Decision Making* 11.03, pp. 661–703.

Organization, World Health et al. (Oct. 2020). *Occupational health: Stress at the workplace*. Online; accessed 25-December-2021. url: https://www.who.int/news-room/questions-and-answers/item/occupational-health-stress-at-the-workplace.

OSHWiki (2020). *Mental health at work — OSHWiki*. Online; accessed 25-December-2021. url: http://oshwiki.eu/index.php?title=Mental_health_at_work.

Osterwalder, Alexander et al. (2015). *Value proposition design: How to create products and services customers want*. Vol. 2. John Wiley & Sons.

Picard, Rosalind W (2016). "Automating the recognition of stress and emotion: From lab to real-world impact". In: *IEEE MultiMedia* 23.3, pp. 3–7.

Polska, ADP (2018). "The Workforce View in Europe 2018". In: *Pobrane z: www. adp. pl/assets/vfs/.../Workforce-View-2018/PL/ADP-Workforce-View-2018-PL. pdf (15.02. 2018)*.

Ponti, Moacir et al. (2015). "Image restoration using gradient iteration and constraints for band extrapolation". In: *IEEE Journal of Selected Topics in Signal Processing* 10.1, pp. 71–80.

Pyle, Dorian and Cristina San José (June 2015). *An executive's guide to machine learning*. Online; accessed 10-January-2022. url: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/an-executives-guide-to-machine-learning.

Ramos Filho, Jaime André, Mauricio João Atamanczuk, and Rui Francisco Martins Marçal (2010). "Seleção de técnicas de manutenção para processo de armazenagem através do Método de Análise Hierárquica". In: *Revista Produção Online* 10.1.

Redmond, Sam and Christopher Sauer (2017). "Exploring Hardware Parallelism's Influence on Programming Through Convolution in CUDA and PyTorch". In.

Rogalewicz, Michał and Robert Sika (2016). "Methodologies of knowledge discovery from data and data mining methods in mechanical engineering". In: *Management and Production Engineering Review*.

Saaty, Roseanna W (1987). "The analytic hierarchy process—what it is and how it is used". In: *Mathematical modelling* 9.3-5, pp. 161–176.

Sabour, Rita Meziati et al. (2021). "Ubfc-phys: A multimodal database for psychophysiological studies of social stress". In: *IEEE Transactions on Affective Computing*.

Shruti, M (May 2022). *Discover the Differences Between AI vs. Machine Learning vs. Deep Learning*. Online; accessed 10-January-2022. url: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/ai-vs-machine-learning-vs-deep-learning.

Springenberg, Jost Tobias et al. (2014). "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806*.

Srivastava, Shrey et al. (2021). "Comparative analysis of deep learning image detection algorithms". In: *Journal of Big Data* 8.1, pp. 1–27.

Stančin, Igor and Alan Jović (2019). "An overview and comparison of free Python libraries for data mining and big data analysis". In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, pp. 977–982.

Szegedy, Christian et al. (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Tucker, Robert (Sept. 2008). *Effective Idea Selection is Critical to Systematic Innovation*. Online; accessed 13-February-2022. url: https://innovationmanagement.se/2008/09/29/effective-idea-selection-is-critical-to-systematic-innovation/.

Viegas, Carla et al. (2018). "Towards Independent Stress Detection: A Dependent Model Using Facial Action Units". In: *2018 International Conference on Content-Based Multimedia Indexing (CBMI)*, pp. 1–6. doi: 10.1109/CBMI.2018.8516497.

Warde-Farley, David et al. (2013). "An empirical analysis of dropout in piecewise linear networks". In: *arXiv preprint arXiv:1312.6197*.

Wirth, Rüdiger and Jochen Hipp (2000). "CRISP-DM: Towards a standard process model for data mining". In: *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*. Vol. 1. Manchester, pp. 29–40.

Wolpert, David H and William G Macready (1997). "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1, pp. 67–82.

Woodall, Tony (2003). "Conceptualising 'value for the customer': an attributional, structural and dispositional analysis". In: *Academy of marketing science review* 12.1, pp. 1–42.

Xi, Xi and Qiuli Qin (2013). "Product quality evaluation system based on AHP fuzzy comprehensive evaluation". In: *Journal of Industrial Engineering and Management (JIEM)* 6.1, pp. 356–366.

Zhang, Huijun et al. (2020). "Video-based stress detection through deep learning". In: *Sensors* 20.19, p. 5552.