POLITECNICO DI TORINO

Master’s Degree Program

in Computer Engineering

MSc thesis

Using Artificial Intelligence for Thermographic Image Analysis: Applications to the Arc Welding Process

Supervisors

Prof. RAFFAELLA SESANA

Candidate

NILOUFAR ZAMINDAR

Co-supervisor

Prof. FRANCESCA MARIA CURA'

Dr. VALENTINO RAZZA

LUCA SANTORO

Academic Year 2023-2024

Summary

This study has focus on making a model with high reliability to classify thermography videos, for achieving high accuracy through series of steps that we did carefully. We started by thoroughly preparing and preprocessing the thermography video data. Custom scripts were developed to extract thermal frames and convert them into a format optimized for neural networks. In this preprocessing we resized the frames, also we normalized the pixel values, and used data augmentation methods to make the dataset more diverse and power. These steps were crucial to ensure the model could accurately handle different thermal patterns and conditions. In this model we combined Convolutional Neural Networks (CNNs) with Long Short-Term Memory (LSTM) networks. The CNN layers pulled out some spatial features from the thermal frames, while the LSTM layers obtained the time-based relationships between frames and then making the model especially efficient at processing video data. The model was trained using the sparse_categorical_crossentropy loss function and the Adam optimizer, both chosen because they're good at efficiently training deep learning models. Training involved multiple epochs, during which the model's parameters were fine-tuned to reduce the loss function. To prevent overfitting and ensure the model could generalize well to new data, techniques like early stopping and regularization were used. We did some important adjustments which are setting the learning rate, batch size, and the model's structure according to the number of layers and neurons that we had. During the process, we kept the model's efficiency which we used a validation dataset that can focus on accuracy to make sure which was working efficiently. After finishing the training phase, the model reached a final accuracy of about 97%, showing how well the chosen architecture and training process worked.

Acknowledgements

I would like to express my deepest appreciation to everyone who contributed to making this research project possible. I am especially grateful to all my supervisors and my company tutors, with a special thanks to Prof. Sesana for their invaluable advice, unwavering support, and insightful feedback that guided me throughout this journey.

Lastly, I want to extend my heartfelt thanks to my lovely and wonderful parents and brothers and loved ones for their unwavering encouragement, patience, and understanding, which have been pillars of strength throughout this challenging yet rewarding journey.

Contents

List of figures

Acronyms

FNN	Feed-Forward Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
RL	Reinforcement learning
DNN	Deep Neural Network
ANN	Artificial Neural Network
MSE	Mean Squared Error
CE	Cross-Entropy
SGD	Stochastic Gradient Descent
RMSE	Root Mean Squared Error
ReLU	Rectified Linear Unit
ELU	Exponential Linear Unit
FC	Fully Connected
AI	Artificial Intelligence
LLM	Large Language Model
BPTT	Backpropagation Through Time
BRNN	Bidirectional Recurrent Neural Network
MAE	Mean Absolute Error
FLIR	Forward-Looking Infrared
LSTM	Long Short-Term Memory

Chapter 1

Introduction

1.1 Purpose

Arc welding, a fusion process, has been extensively utilized across various industrial domains for numerous years. Being sure from the quality control of welding progress has a great value in some areas like nuclear engineering and aerospace. Typically, offline non-destructive testing methods, including X-rays, liquid penetrants, magnetic particle testing, ultrasonics, among others, are commonly employed to detect any faults or deficiencies in welded joints [1]. Arc welding is a welding technique that employs an electric arc to heat and fuse metals together [2]. It's one of the most usual and adaptive welding techniques which is used across different industries. It works by making an electric arc between an electrode and the pieces that was worked on. The heat generated by the arc melts the base metal and, if applicable, a consumable electrode, forming a weld pool. As the weld pool gradually cools, it undergoes solidification, resulting in a durable and cohesive bond that firmly unites the welded components [3]. Thermography in arc welding involves using thermal imaging cameras to monitor the temperature distribution and heat patterns generated during the welding process. If we review the history of thermography in arc welding, it can notice that is a fascinating tour that lasts for many decades and involves improvements in technology. Here's an overview:

Early Developments: The use of thermography in welding traces back to the mid-20th century when infrared cameras and thermal imaging technology began to emerge. Mostly, the primary requests focused on qualitative analysis, that is allowing welders to imagine temperature distributions and detect potential flaws during the welding process.

Advancements in Thermal Imaging: As thermal imaging technology evolved, so did its applications in arc welding. It was possible to take accurate temperature measurements and thus better present the quality of welds with recovered sensitivity, clearness, and infrared cameras which are portable [92].

Research and Development: Temperature fluctuation, weld parameters, and the quality of welding are interrelated. People who search on Welding, try a lot to study and understand the relation on changing of temperature and results which are obtained on welding. During the arc welding, thermal data process has been detected by researchers and prepared some testable investigations and numerical simulation studies, extending predictive models to optimize welding parameters and reduce flaws.

Industrial Applications: In automotive, aerospace, and construction and manufacturing, among other industries, thermography has wide applicability. In arc welding, industrial application is used for monitoring some application that are in process also for controlling the quality and detecting the flaws. It allows welders and engineers to assess weld integrity, identify discontinuities such as lack of fusion, porosity, or cracks, and make real-time adjustments to welding parameters to ensure optimal performance.

Integration with Automation and Robotics: With the rise of automation and robotics in welding operations, thermography has become an integral part of advanced welding systems. By working with robotic arms and control systems, infrared cameras can be equipped in requests with robotic arc welding to make sure the quality of welds and apply control of welding parameters. Then this allows to monitor in the time, which is real, so setting can be made to be sure that consistency in the welding process is always obtained and done at a high standard. The IR cameras measure temperature changes in the welding process and detect potential issues; the control systems automatically correct the welding parameters to prevent such deviations.

Future Directions: Thermography in welding is a sector which is inspiring for the improvement of technologies and their interacting with new methods and orientations. Improved data analytics should have ability to make a prediction for maintenance, prepare to monitor the time, which is exact, and make a basis for welding processes.

Here's how thermography is typically applied in arc welding:

1. Monitoring Temperature Distribution

Thermal imaging cameras capture infrared radiation emitted by objects, allowing for the visualization of temperature distribution. In arc welding, these cameras can be used to monitor the temperature of the weld pool, the heat-affected zone, and surrounding materials in real-time.

2. Detecting Defects

Thermography can help detect defects such as lack of fusion, porosity, or cracks by identifying abnormal temperature patterns during welding. Variations in temperature can indicate areas of potential defects, allowing welders to take corrective actions.

3. Optimizing Parameters

By analyzing thermal images, welders and engineers can optimize welding parameters such as current, voltage, travel speed, and shielding gas flow rate to achieve desired weld quality and minimize defects.

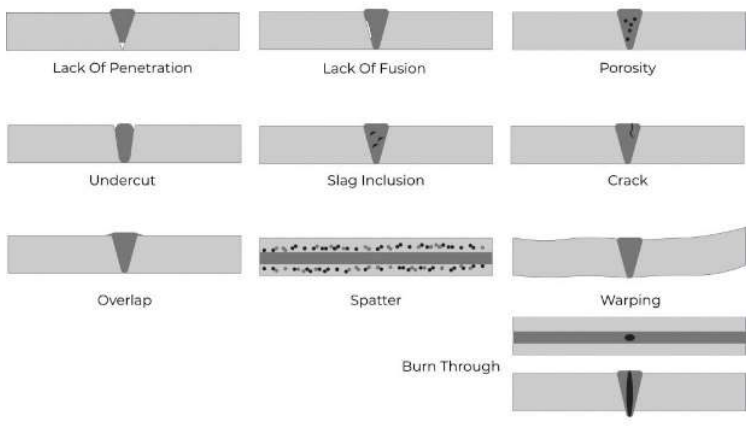


Fig 1.1 Weld defects [4].

Indeed, the main goal is recognizing three different thermography classes by classifying thermography videos of arc welding which will be done by labeling the videos and then it needs to measure the accuracy that shows how much classification is correct.

1.2 Work introduction

The task of classifying thermography videos in arc welding based on a neural network presents a significant challenge within the field of welding inspection and quality control. As thermography technology, this technology becomes more usual in welding also during arc welding there's an increased need for automated ways for analyzing and organizing the huge number of thermal data which are produced.

One of the primary difficulties in this problem is the complexity and variability of thermal patterns observed in welding processes. Arc welding generates dynamic and intricate thermal signatures influenced by factors such as welding parameters, material properties, and environmental conditions. Analyzing thermography videos to identify and classify patterns manually takes a lot of time and it can help to have more mistakes.

The development of a neural network-based solution to classify thermography videos in arc welding requires addressing several technical challenges. This causes making neural networks that can obtain spatial and temporal exact details from thermographic data, fine-tuning models to perform well under different welding conditions, and making sure they can be scaled and efficient for real-time use in industrial environments.

Chapter2

Background

2.1 overview

Through this aim, it is necessary to find different meaning and concepts related to this issue. Machine learning, specifically deep learning, has been entered to a large number of fields which are engineering, economics, healthcare, and beyond [5]. Machine learning is part of artificial intelligence that focuses on making algorithms and models which allow computers to be able to learn from data and generate predictions or decisions without direct programming. It includes different ways, such as supervised learning, unsupervised learning, and reinforcement learning, each is matched to tasks and data types [6]. Deep Learning has obtained a lot of attention mainly because it is good at tasks like detecting images and understanding speech. Unlike older methods, it doesn't need humans to design complicated features; it figures out important things from the data all by itself. The main advantage of deep learning is its capability to automatically learn and extract elaborate designs and features from large quantities of data. This allows deep learning samples to attain cutting-edge implementation across different functions, such as image recognition, and speech recognition, it doesn't have any need for explicit trait engineering. Furthermore, deep learning samples are flexible and adaptive, and it can make adaption to different types of data also make them ideal for a lot of requests. On the other hand, is its heavy reliance on extensive datasets for training Deep learning instances frequently necessitate vast datasets to generalize well and avoid overfitting, which can be challenging and costly to obtain, especially for niche or specialized domains [7]. What is more, deep learning samples can be expensive to train if it is computed, it needs significant computational resources and time. Furthermore, deep learning instances are often considered "black boxes" meaning that they are not easily interpretable, making it difficult to comprehend the process behind their decision-making or predictions. When there is not a power of interpretability, it can be a problem in requests where resolution is important for instance, in areas like finance and healthcare. The last thing that we should know about it is that deep learning models are easy to be attacked where there are hidden changes which causes the incorrect predictions. Now we want to survey various architectures in deep learning, so we explore a range of the neural network samples that have been created for different tasks and domains [8]. Here are some common architectures to consider:

Feed-Forward Neural Networks (FNNs): also recognized as Multilayer perceptron's (MLPs), serve as a foundational kind of artificial neural network. Within FNNs, data advances linearly from input to output layers, without any feedback loops. These networks are adaptive and commonly categorized for some tasks such as classification and regression [9][10].

Convolutional Neural Networks (CNN): This neural network consists of convolutional, pooling, and fully connected layers. The convolutional and pooling layers function are acting like filters, which are pulling out significant specifications from input patterns. Conceptually, this process resembles performing finite-impulse response (FIR) filtering on the input data, with the filter designed to extract the most salient parts of the data. In the context of modulation classification, it's essential to reconstruct the input data accurately to enable classification [11].

Recurrent Neural Network (RNN): is one of the neural networks created to handle consecutive data by maintaining a memory of previous inputs. Unlike traditional neural networks that process each input individually RNNs have connections that allow them to retain information over time, making them ideal for duties such as time series prediction, natural language processing, and speech recognition. They can identify samples and connections orderly and is capable to manage input series of various lengths. Variants like LSTM and GRU address issues such as the disappearance gradient issue, enabling RNNs to more effectively capture long-term attachments in data [12]. These are three architectures which can be introduced as Feed-Forward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), that are important bases in the artificial neural networks, and each of these architectures are defined for specific tasks and data types. In machine learning, the method of learning from data is built around three main principles which are supervised learning, unsupervised learning, and reinforcement learning. Each of these ways show a separate method to pulling out the samples and decide based on data that is available.

2.1.1 Supervised Learning

In supervised learning, a model is trained using labeled data, meaning each input is linked to a specific output label. According to the aim of this algorithm, it specifies how the feature of input and output label can communicate to each other. based on the provided labeled examples in the training dataset. This pattern is ideally suited for tasks such as classification, in classification tasks, the model predicts distinct class labels, whereas in regression tasks, it forecasts continuous numerical values. Supervised learning algorithms focus on reducing the difference between predicted outputs and actual labels. Firstly, the process starts with raw data, which is labeled to form a training dataset later. The algorithm is instructed during training by this dataset and its corresponding labels. Once the model is trained, it processes new data to generate predictions, thereby completing the supervised learning workflow [10].

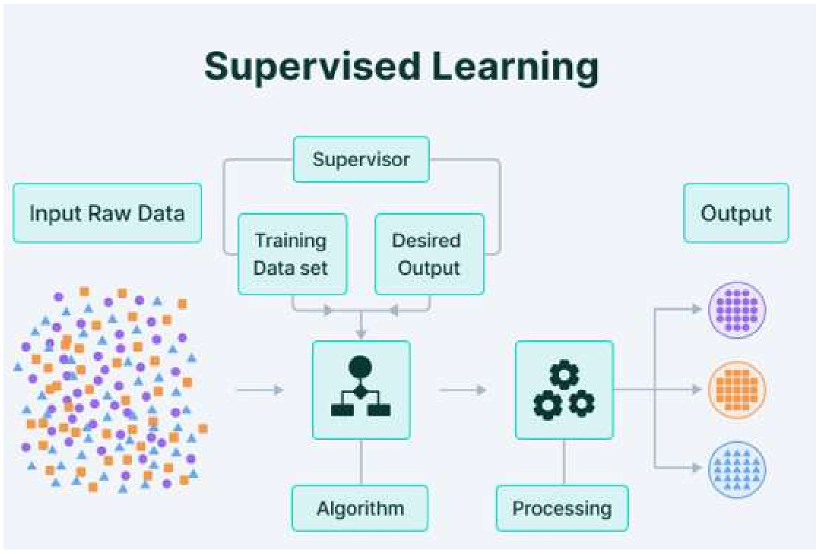


Fig 2.1 Supervised learning architecture [13].

2.1.2 Unsupervised Learning

In unsupervised learning, a model is trained on data that lacks specific output labels, meaning the input examples are unlabeled. In this approach, the model tries to learn patterns, structures, or relationships within the data without guide from out. These duties contain clustering, which classifies similar data points, reducing the dimensionality, which obtains important features while it is simplifying the data, and the last one which is generative modeling, where the model learns to make new data samples like the original data. Unsupervised learning algorithms aim is to find hidden patterns and relationships in the data, helping with exploration and discovery. These methods which are important to machine learning, providing flexible ways to obtain insights and predict in different sections and areas [14].

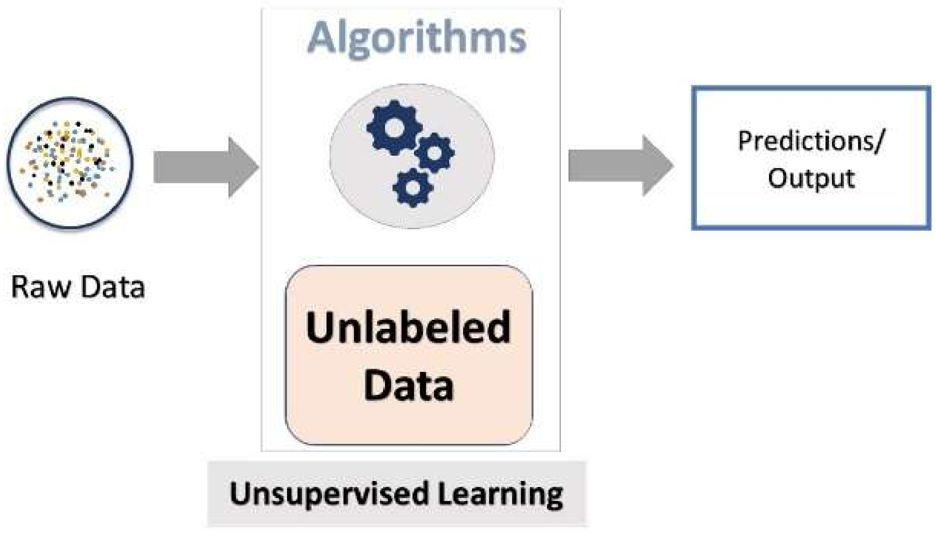


Fig 2.2 Unsupervised learning [15].

2.1.3 Reinforcement learning (RL)

A type of machine learning paradigm involves an agent interacting with its environment to accomplish a specific objective. This agent figures out to decide by adopting some actions and obtaining feedback through rewards or penalties. The primary aim of the agent is to optimize the total rewards accumulated over time. In reinforcement learning, an agent evaluates the current state of its environment and chooses actions according to a policy that links states to actions. Upon executing an action, the agent moves to a new state and receives a reward from the environment. This feedback is then utilized by the agent to adjust its policy, thereby enhancing its decision-making abilities over time [16].

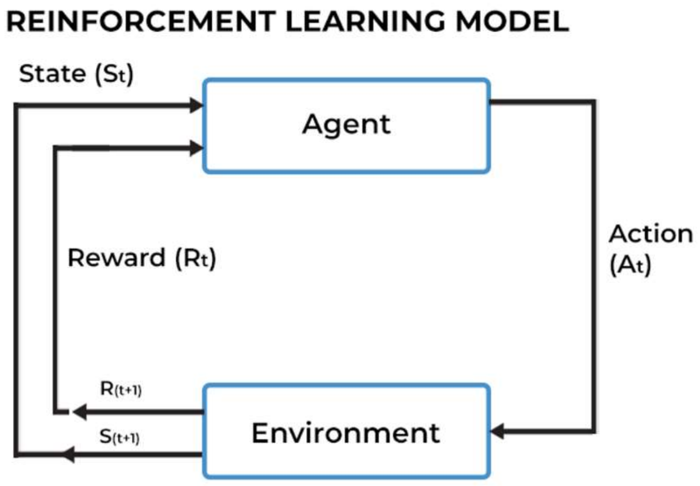


Fig 2.3 Reinforcement learning model [17].

2.2 Neural Network Layers

Deep neural networks (DNNs) are a variant of artificial neural networks (ANNs) characterized by multiple layers of interconnected neurons. Unlike shallow neural networks with just a few hidden layers, deep neural networks (DNNs) incorporate many hidden layers. This form let them to obtain and figure out complicated samples and depiction within the data [18]. However, training DNNs can be challenging due to issues such as overfitting and underfitting:

Overfitting

Overfitting occurs when a model tries to how it can memorize the training data instead of generalizing from it. This leads to poor performance on unseen data, as the model is overly complex and ends up capturing noise or irrelevant patterns present in the training data. Techniques like dropout and weight decay can mitigate overfitting by discouraging excessively complex models [19].

Underfitting

Underfitting occurs when a model is too simple to recognize the underlying patterns in the data. This leads to high bias and poor performance on both the training and test sets. To address underfitting, one can increase the model's capacity by adding more layers or neurons, or by using more complex architectures such as convolutional or recurrent neural networks. We want to summarize all the things related to deep neural networks which are powerful and can acquire the knowledge of complex samples from data, but they can run to different problems such as overfitting and underfitting. Using effective regularization ways and carefully choosing models are important for training deep neural networks (DNNs) that work well on new, unseen data. In a deep neural network (DNN), each layer serves a specific purpose in the transformation of input data into meaningful output predictions [6]. This image shows a layer within a neural network that we are presenting:

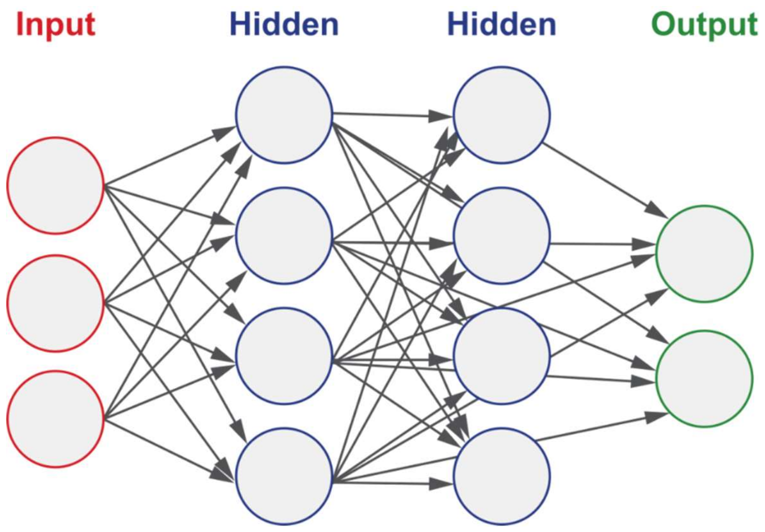


Fig 2.4 Main layers in Deep learning model.

Here's an introduction and explanation of each layer commonly found in DNN architectures:

Input Layer

The neural network's initial layer is the input layer. and serves as the entry point for the input data. The input layer's neurons represent the features or attributes of the input data. Each neuron is associated with a particular input feature, and the values of these neurons reflect the raw input data [20].

Hidden Layers

Hidden layers, situated between the input and output layers, are where the core computations and feature extraction occur. Every hidden layer is composed of numerous neurons, also referred to as units or nodes. Neurons in a hidden layer accept inputs from the previous layer, apply a nonlinear transformation via an activation function, and transmit the modified outputs to the next layer. For doing this action, hidden layers make the neural network able to detect complex samples within the input data [21].

Output Layer

The final layer of the neural network, known as the output layer, generates the network's predictions or outputs. Neurons in the output layer represent the desired output of the network, such as class probabilities in classification tasks or continuous values in regression tasks. The tasks nature shows the total number of neurons in the output layer. (e.g., binary classification, multi-class classification, regression). Depending on the task, the output layer usually employs a suitable activation function, such as the softmax function for classification tasks or linear activation for regression tasks [20].

2.3 Steps for Training Neural Networks

Training a neural network entail optimizing its parameters, specifically the weights and biases, to minimize error. a predefined loss function, typically by adjusting them using a method called backpropagation in conjunction with an optimization algorithm. Also training a neural network involves instructing it to generate accurate predictions or classifications by processing input data [6].

Here's a comprehensive explanation of the steps involved in training a neural network:

1. Initialization

The initial step in training a neural network is to set its parameters, including weights and biases. These parameters are usually initialized randomly, though certain initialization strategies can be employed to enhance convergence [22].

Weights Initialization:

$w_{ij\sim N}(0,\sigma^2)$ where $N(0,\sigma^2)$ represents the Gaussian distribution with a mean of 0 and a variance of σ^2.

Biases Initialization: $b_i = 0$

2. Forward Propagation

During forward propagation, each layer in the neural network computes its output based on the inputs received from the previous layer. This computation is multiplying the input values by the corresponding weights, adding biases, and then it should apply an activation function is for introducing nonlinearity. Sigmoid, tanh, and ReLU are among the common activation functions, each offering distinct characteristics and advantages [21].

Linear Transformation:

$$z_i = \sum_{j=1}^{n} w_{ij \cdot x_j + b_i}$$

Activation:

$a_i = g(z_i)$ where $g(z_i)$ is the activation function.

3. Loss Calculation

The loss function quantifies the accuracy of the neural network's predictions that match the true labels in the training data. It measures how the network is well, where lower values showing a closer match between predictions and actual results. The choice of a loss function is tailored to the task at hand [23]. For instance, mean squared error (MSE) is commonly used for regression tasks, whereas cross-entropy loss is ideal for classification tasks [24].

For regression: Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{N} \Sigma_{i=1}^{N} (y_i - \hat{y}_i)^2$$

For classification: Cross-Entropy Loss:

$$CE = -\frac{1}{N} \Sigma_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

4. Backpropagation

Backpropagation is like adjusting the sails of a boat to steer it in the right direction. When a boat moves forward, it uses feedback to make adjustments, ensuring it stays on course. Similarly, in backpropagation, a neural network adjusts its parameters (weights and biases) based on the difference between its predictions and the actual outcomes. It accomplishes this by calculating the gradients of the loss function in relation to each parameter, indicating how much each parameter should be adjusted to minimize the loss. These gradients are then used to improve parameters, also it helps to improve its performance. This procedure tries to be repeated couple of times until the network learns to predict correctly and exactly . In the core of this process, backpropagation is the method which a neural network learns from its mistakes and revises its performance during the time. Backpropagation is an important part of training a neural network. It entails calculating the gradient of the loss function with respect to each network parameter by applying the chain rule of calculus. These gradients are then used to change the parameters in the opposite way to the gradient, which has a goal to decrease the loss function and revise the value for the accuracy of the model. Backpropagation enables efficient computation of gradients through the network, allowing for the optimization of thousands or even millions of parameters [25].

Gradient of Loss with respect to Output [26]:

$$\frac{\partial\, loss}{\partial\, \hat{y}_i}$$

Chain Rule:

$$\frac{\partial\, loss}{\partial\, z_i} = \frac{\partial\, loss}{\partial\, \hat{y}_i} \cdot \frac{\partial\, \hat{y}_i}{\partial\, z_i}$$

Gradient of Loss with respect to Weights and Biases [26] :

$$\frac{\partial\, loss}{\partial\, w_{ij}} \quad and \quad \frac{\partial\, loss}{\partial\, b_i}$$

The basic equation for updating a neural network weight (W) during backpropagation entails adjusting the weight by subtracting a portion of the loss function's gradient concerning that weight, multiplied by the learning rate (α) [27]. Mathematically, it appears as:

$$w = w - \alpha \frac{\partial\, loss}{\partial w}$$

5. Parameter Update

With the gradients computed, an optimization algorithm like stochastic gradient descent (SGD), is used to update the network parameters in the direction that minimizes the loss function. The learning rate parameter specifies the measurement of the alignment which is created to the parameters of this model with each repetition, effectively handling how quickly the network updates its weights and biases. It dictates how significantly the parameters are altered in response to the calculated gradients, thus influencing the speed and stability of the learning process. A well-chosen learning rate strikes a balance between making sufficient progress toward minimizing the loss function and ensuring that the model converges smoothly without overshooting optimal solutions [28].

Gradient Descent:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{\partial\, loss}{\partial\, w_{ij}}$$

$$b_i \leftarrow b_i - \alpha \frac{\partial\, loss}{\partial\, b_i}$$, where α is the learning rate.

6. Iterative Optimization

Training a neural network involves iteratively repeating the forward propagation, loss calculation, backpropagation, and parameter update steps for a predefined number of epochs. The goal is to gradually reduce the value of the loss function, indicating that the network is learning to create better predictions on the training data [29].

7. Validation

Validation is essential for monitoring the network's performance on data that it hasn't seen during training. It assists to identify the overfitting and makes ready to have ability for producing the network. Hyperparameter tuning, such as adjusting the learning rate or the number of hidden units, can be performed based on validation performance.

Measure the effectiveness of the model on a validation dataset by employing relevant metrics such as accuracy, precision, recall, or F1-score [31].

8. Testing

After the training phase, the final model undergoes evaluation on an independent test dataset to determine its real-world performance. This evaluation, uses particular criteria to measure how the model can conform with new data in a good way and predict accurately.

In the last step, the process of training of a neural network should adopt some measures to provide a precise balance of initializations, forward and backward propagation, parameter updates, and validation to optimize the model's performance and be sure of its capacity to make popular for new data [18].

2.4 Fundamental tasks in Machine Learning

Regression and classification are two fundamental tasks in machine learning, each serving distinct purposes in analyzing and interpreting data:

Regression function

Regression is a method used in machine learning to predict the relationship between input features (like independent variables) and an output (dependent variable) by fitting a mathematical model to the data. It is simple to mention that regression function plays a good role to help us for understanding how the value of one variable improves the response for another variable's changes. It makes prediction for values of continuous numerical based on input data, which gives us an allowance to make an estimation for future results, spot samples, and understanding relationships between variables. For example, regression can be used to estimate the price of the house based on some elements like square footage, the number of bedrooms, and location. It can also be used in finance to make a prediction in stock prices or in healthcare areas to predict the result of patient based on different elements. Different types of regression models, such as linear regression, polynomial regression, ridge regression, and lasso regression, are tailored to handle different situations and types of datasets.Each regression method is suited to particular characteristics of the data and the underlying relationships being modeled. The effectiveness of these regression models is commonly assessed through metrics like mean squared error (MSE), root mean squared error (RMSE), and the coefficient of determination (R-squared). These criteria bring insights into how well the model adapts the data, with lower mistakes values and higher R-squared values which is indicating better efficiency and predictive accuracy. The connection between the independent variables (features) and the dependent variable (target) is expressed through a linear equation [30][33]. The basic formula for linear regression is given as:

$$\hat{y} = b_0 + b_1\varkappa_1 + b_2\varkappa_2 + \cdots + b_n\varkappa_n$$

Parameters details:

$\hat{y}$ is the anticipated value of the dependent variable,

b_0 is the y-intercept (bias term),

b_1 , b_2 , … , b_n are the coefficients (weights) corresponding to the independent variables

$\varkappa_1$, $\varkappa_2$, … , $\varkappa_n$ respectively.

This equation describes a line in a multidimensional space that best aligns with the observed data points. Throughout the training process, the model determines the optimal coefficients (b0, b1, ..., bn) that reduce the difference between the predicted and actual values in the training set.

These coefficients are usually estimated using the least squares method, which minimizes the sum of the squared deviations between the observed and predicted values:

$$\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

We have:

N is the number of data points,

y_i represents the true value of the dependent variable for the ith data point,

$\hat{y}_i$ is the value forcasted of the dependent variable for the ith data point.

2.4.1 Classification Methods

Classification functions as a predictive model designed to estimate a mapping function from input variables to distinct output categories, such as labels or categories. This mapping function, inherent to classification algorithms, is responsible for predicting the label or category associated with given input variables. Although classification algorithms can handle both discrete and continuous variables, they must classify examples into one of two or more classes [32]. Various types of classification algorithms include:

1. Decision tree classification

Decision tree classification involves building a model by creating a decision tree. Each node within the decision tree represents a specific test or decision point based on an attribute, and each branch extending from that node corresponds to one of the possible outcomes or results of that test. This structure makes an allowance for the tree to assess various attributes systematically and manage the data downward that reflect the different potential outline or classifications, and in the last stage it causes the final decision or prediction. [32][33].

2. Random forest classification

This tree-based algorithm consists of an ensemble of decision trees, each generated from a randomly selected subset of the original training data. The random forest classification algorithm aggregates the outputs from all individual decision trees to produce the final prediction, resulting in greater accuracy than any single tree alone [34].

3. K-nearest neighbor

The K-nearest neighbor algorithm assumes that similar entities are positioned close to one another. It uses similarity of features to make prediction for the values of new data points, allowing to group similar points based on their distance and closeness. The main goal of the algorithm is to determine the probability that a data point belongs to a specific group [35].

2.4.2 Regression versus Classification in Machine Learning

The key distinction between regression and classification lies in their predictive targets:
regression forecasts continuous quantities, whereas classification predicts discrete class labels. Nonetheless, there exist certain intersections between these two categories of machine learning algorithms. A regression algorithm can guess whole numbers, like counting items, while a classification algorithm can estimate the chance of different labels, like the probability of something being a certain category [36].

Activation functions with Neural Networks

Activation functions introduce non-linearity into neural networks, allowing them to model complex relationships between input and output variables. Without activation functions, neural networks would be confined to linear transformations, capable of representing only simple relationships. By incorporating non-linear activation functions, neural networks can learn and show complicated samples also the way that connect data with each other.

Here are some key reasons why activation functions are used in neural networks:

Introduction of Non-Linearity: Activation functions allow neural networks to capture and model non-linear relationships between input and output variables. This non-linearity is important for obtaining the complicated samples which are available in real data.

Enable Learning of Complex Features: Non-linear activation functions allow neural networks to learn and represent intricate features and patterns within the data. This is the vital capability which is specified for some tasks such as detecting the image, processing the natural language, and different kinds of detecting samples.

Gradient Propagation: Activation functions help in efficient gradient propagation during the backpropagation algorithm, which is used to train neural networks. Some activation functions that are certain have eligible attributes that help in more permanent and efficient training of neural networks.

Output Transformation: Activation functions transform the raw output of a neuron into a form that is suitable for the next layer of the network. They help in controlling the range and scale of the neuron outputs, ensuring that the network learns effectively [37].

Neural networks employ various types of activation functions. Some common ones include:

Sigmoid: The sigmoid function compresses a neuron's output into the [0, 1] range. It is usually requested in binary classification duties to determine probabilities.

Hyperbolic Tangent (tanh): Like the sigmoid function, the tanh function constrains a neuron's output to the range [-1, 1]. It is centered around zero and can help alleviate the vanishing gradient problem compared to the sigmoid function.

Rectified Linear Unit (ReLU): The ReLU function outputs zero for negative inputs and the input value itself for positive inputs. The ReLU function is one of the activation functions which used a lot according to its simplicity and performance in training deep neural networks.

Leaky ReLU: The Leaky ReLU function is like the ReLU function however, it allows a small non-zero gradient for negative input values. This change prevents neurons from the general disabling, which is done for negative inputs, which is allowing some information to still cross the network. By showing this small gradient, Leaky ReLU helps to reduce the issue of dead neurons which can happen in traditional ReLU when neurons are not active, and they stop learning. This setting increases the capability of network to make complicated paradigms more as model and keep robustness during training. This feature tries to prevent the issue for the ReLU where neurons doesn't work during training.

Exponential Linear Unit (ELU): The ELU function is like ReLU for positive inputs but has a non-zero output for negative inputs, allowing smoother gradients during training.

Softmax: The softmax function is commonly utilized in the output layer of a neural network for multi-class classification tasks. It converts the outputs generated by the neurons in the output layer into a probability distribution that spans across all the possible classes. This conversion makes sure that the network's predictions are introduced as probabilities, which is returning the probability that each input belongs to a particular class, thereby simplifying the classification process [38].

2.5 Different Neural Network Architectures

2.5.1 Feed-Forward Neural Networks

A feedforward neural network, also referred to as a fully connected neural network, is one of the earliest neural network architectures developed in artificial intelligence. This network learns from input data independently to do tasks and roles. The primary objective of a feedforward neural network is to predict the classification label for a given image. This means giving a score or prediction probability to each possible label, which is creating a vector for showing the network's output. The label with the highest score is selected as the network's predicted category for the image. Typically, a feedforward neural network contains one or more hidden layers of neurons between the input and output layers. These hidden layers make the network able to analyze and identify the features of the input data more by aim of improving the prediction accuracy [40].

2.5.2 Convolutional Neural Networks

Convolutional Networks (ConvNets) stand out as the most effective deep learning models for image classification tasks. Drawing inspiration from biological processes, their multilayered architectures facilitate the hierarchical and automatic learning of invariant features. Beginning with the detection of basic features, they progress to discerning and amalgamating these features to grasp more intricate patterns. The various levels of features originate from distinct layers within the network. Each layer comprises a specific number of neurons and is represented in three dimensions: height, width, and depth. Convolutional Neural Networks (CNNs) are a type of deep neural network specifically designed for processing and analyzing visual data, such as images and videos. They excel at tasks like image classification, object detection, and image segmentation. CNNs are inspired by the organization and functioning of the human visual system and are created to automatically and adaptively learn hierarchical representations of data. The primary benefit of the activation map lies in its ability to capture distinctive features from an image while simultaneously reducing the data volume for processing. The convolution operation involves applying a feature detector matrix to the input data, essentially comprising a set of values compatible with the machine. By utilizing various values of the feature detector, multiple versions of the image are generated [6]. Furthermore, the convolutional model undergoes training via backpropagation to minimize errors in each layer. Based on the achieved minimal error, the depth and padding settings are determined [42].

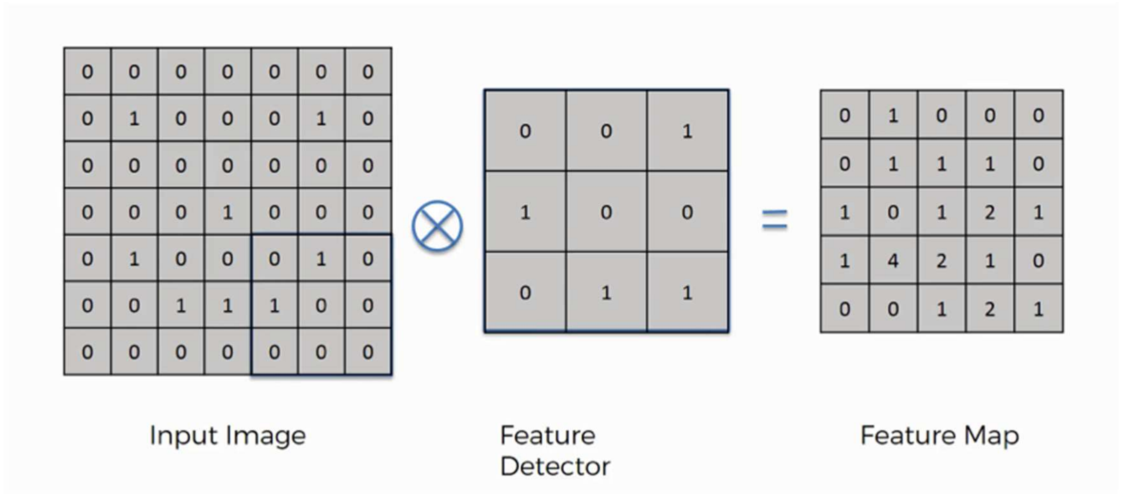


Fig 2.5 Convolution to generate an Activation Map [41].

Pooling is an important step in decreasing the size of activation map, which is keeping only the most vital features while improving the model's ability to identify the objects, even when they will be presented in different shapes or angles. This procedure shrinks the number of parameters which model needs to learn, and this action helps to avoid overfitting. Max pooling gives an allowance to CNNs to control the different sizes of an image, which causes to have precise detection for an object. There are different pooling ways such as max pooling, average pooling, stochastic pooling, and spatial pyramid pooling, with max pooling being the most common.

Max pooling selects the highest value from each sub-matrix of the activation map, creating a new, distinct matrix. This way decreases the number of learnable features effectively while it is keeping the important attributes of the image. Generally, max pooling utilizes a 2x2 filter for this operation [43].

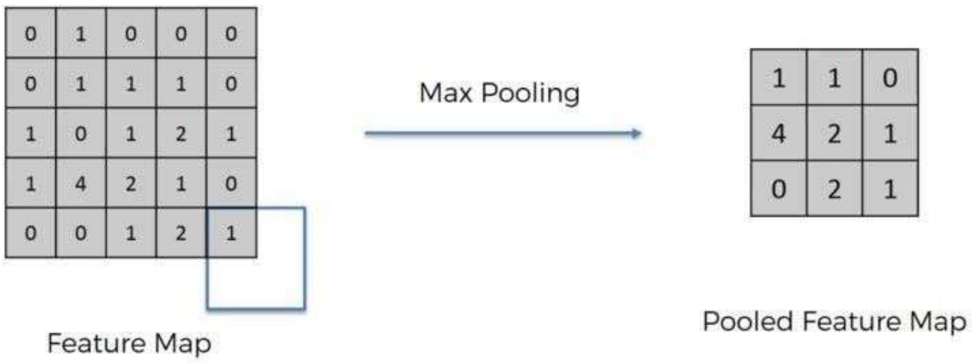


Fig 2.6 Max Pooling of a Feature Map [41].

The Fully Connected Layer, often known as the Hidden Layer, represents the concluding phase of the convolutional neural network. In this stage, a combination of an Affine transformation and a Non-Linear activation function is utilized to integrate and refine the extracted features. This process allows the network to synthesize the information gathered from the previous layers and make final predictions or decisions based on the learned representations [43].

Affine Function : $y = Wx + b$

The Fully Connected layer receives input from the Flatten Layer, which transforms the data into a one-dimensional format (1D Layer). Subsequently, the data from the Flatten Layer undergoes processing by the Affine function followed by the Non-Linear function. This combination of one Affine function and one Non-Linear function constitutes a single Fully Connected (FC) or Hidden Layer. Additional layers of this nature can be incorporated based on the desired depth of the classification model, with the determination influenced by the characteristics of the training dataset. The output from the final hidden layer is subsequently fed into the Softmax or Sigmoid function to generate a probability distribution over all the classes [44].

In the below picture shown different layers of Convolutional Neural Network (CNNs):

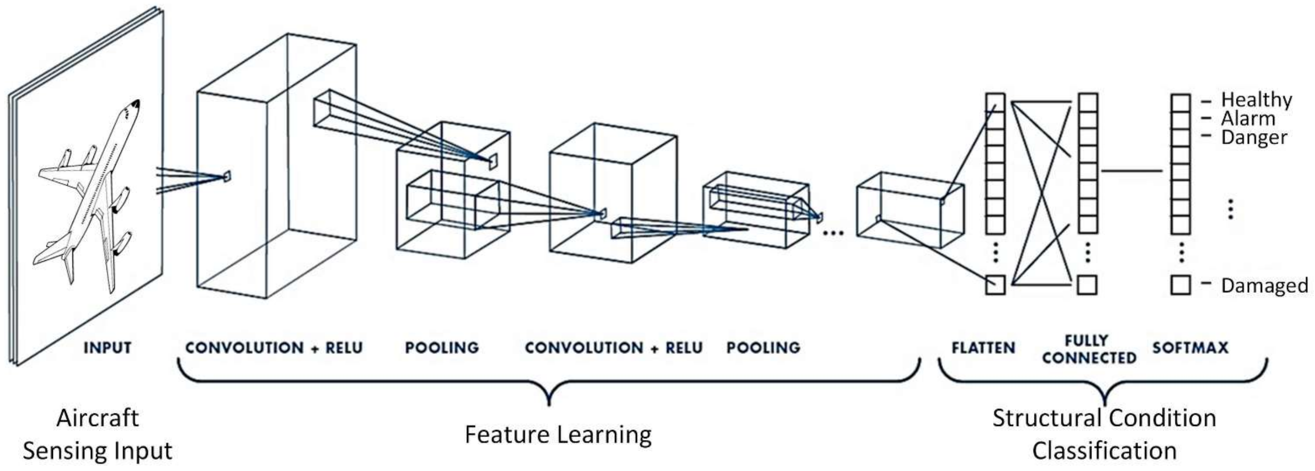


Fig 2.7 Process of analysing the input in convolutional neural network [93].

2.5.3 Recurrent Neural Networks

A recurrent neural network (RNN) is a deep learning model designed to transform sequential input data into corresponding sequential output. Sequential data, such as words, sentences, or time-series data, relies on complex semantic and syntactic relationships. Emulating human sequential data processing, an RNN comprises numerous interconnected components performing tasks like language translation. However, transformer-based artificial intelligence (AI) and large language models (LLMs) are increasingly replacing RNNs due to their superior efficiency in managing sequential data. RNNs consist of neurons, which are computational units that work together to perform complex operations. These neurons are organized into input, output, and hidden layers. The input layer gathers data for processing, the output layer delivers the results, and the hidden layer is where data processing, analysis, and prediction occur. RNNs are called recurrent because they execute the same task for each element in a sequence, with each output depending on previous computations. They have a "memory" feature that retains information from earlier calculations [45].

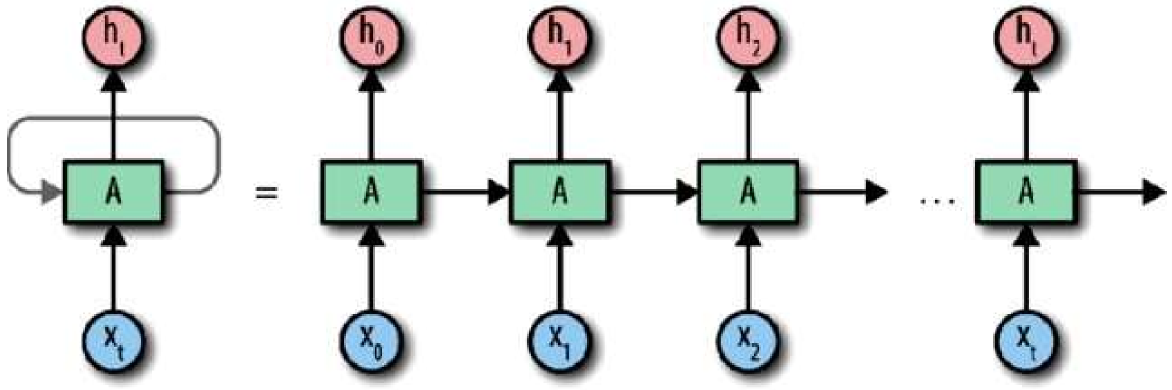


Fig 2.8 RNN Architecture [96].

Hidden state calculation [45]:

$$h_t = f\left(w_{hx}x_t + w_{hh^{h}t-1} + b_h\right)$$

h_t : Hidden state at time step t.
x_t : Input at time step t.
w_{hx} : Weight matrix connecting input to hidden state.
w_{hh} : Weight matrix connecting hidden state to itself.
b_h : Bias term for the hidden state.
f : Activation function, typically tanh or ReLU.

Output Calculation:

$$y_t = g\left(W_{hy}h_t + b_y\right)$$

y_t: Output at time step t.
h_t: Hidden state at time step t.
W_{hy}: Weight matrix connecting hidden state to output.
b_y : Bias term for the output.
g : Activation function, typically softmax for classification or linear for regression.

Loss Calculation: $L = \frac{1}{N}\sum_{t=1}^{\mathbb{N}} L(y_t, \hat{y}_t)$

L: Total loss over $\mathbb{N}$ time steps.
y_t: Predicted output at time step t.

$\hat{y}_t$: True output at time step t.

$L(y_t, \hat{y}_t)$: Loss functions like cross-entropy for classification tasks or mean squared error for regression tasks.

2.5.4 Backpropagation Through Time (BPTT)

The backpropagation algorithm, adapted for RNNs, computes gradients of the loss function with respect to all model parameters across the entire sequence. This process involves calculating the gradients at each time step, accumulating them over the sequence, and then updating the model parameters using gradient descent [46]. RNNs function by sequentially transmitting the data they receive to hidden layers, processing one step at a time. They utilize a self-looping or recurrent mechanism where the hidden layer retains past inputs to enhance future predictions through a short-term memory feature. This involves using both the current input and stored memory to predict the next sequence. RNNs are typically associated with a one-to-one architecture, where each input sequence corresponds to one output. However, they can be adapted to various configurations for specific purposes. Below are several common types of RNNs [47]:

One-to-many

In this RNN variant, a single input is directed towards multiple outputs. This architecture finds utility in linguistic tasks such as image captioning, where it generates a sentence based on a solitary keyword.

Many-to-many

The model uses multiple inputs to predict multiple outputs. For example, an RNN can be used to create a language translator, where it analyzes a sentence and skillfully arranges the words in a different language.

Many-to-one

Multiple inputs contribute to a single output, which is useful for tasks like sentiment analysis. Here, the model determines whether customer sentiments from input testimonials are positive, negative, or neutral [47]. The RNN architecture was the first to introduce language processing capabilities in machine learning models, leading to the development of several variants that maintain its memory retention principle while improving its original functionality. Here are some illustrative examples [12]:

Bidirectional recurrent neural networks

A bidirectional recurrent neural network (BRNN) processes data sequences using both forward and backward layers of hidden nodes. The forward layer operates like a typical RNN, retaining previous inputs in the hidden state to predict the next output. Conversely, the backward layer traverses in the reverse direction, leveraging both the present input and forthcoming hidden states to refine and update the current hidden state. This bidirectional approach lets the network to use information from both past and future, which is increasing its ability to obtain complicated dependencies within the sequence. This combination allows the BRNN to enhance prediction accuracy by considering both past and future contexts. For instance, a BRNN can predict the word "trees" in the sentence "Apple trees are tall" [48].

Long short-term memory

LSTM is an improved version of the regular RNN, aimed at handling long-term connections more effectively in sequential data. While a regular RNN's hidden state activation mostly relies on nearby activations, acting as "short-term memory," and network weights are influenced by computations across entire sequences, akin to "long-term memory," LSTM was redesigned with an activation state that serves as weights. This allows it to retain information over longer distances, hence the name "Long Short-Term Memory." LSTMs are specifically crafted to address long-term dependency issues. They naturally excel at retaining information over extended periods, making it an inherent feature rather than something they need to learn [49].

Temporal order

Temporal order in RNNs and LSTMs refers to the sequential nature of data, where the order of elements matters. In the context of RNNs and LSTMs, temporal order implies that the input data is presented in a sequence, such as words in a sentence, frames in a video, or time steps in a time series. These models are built to handle data in a sequential manner, processing one element at a time and maintaining an internal state that retains information from previous elements. This ability to preserve and use information about the temporal order of input data is critical for tasks like language modeling, speech recognition, and time series prediction [50].

2.6 Metrics in Machine Learning

2.6.1 Accuracy

Accuracy is a crucial metric in machine learning that measures a model's overall correctness in its predictions across all classes or categories in a classification task. It represents the ratio of instances that were accurately classified to the total number of instances evaluated, providing a measure of how well the model correctly identifies the categories or labels across the entire dataset. In essence, accuracy gauges the model's capability to accurately identify positive and negative instances within a dataset [51].

$$\text{Accuracy} = \frac{Total\ Number\ of\ predictions\ whic\ \ is\ true}{The\ whole\ number\ of\ predictions}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+F\ \ +FN}$$

Parameter: True Positives (TP), True Negatives (TN), False Positives (FP),

False Negatives (FN)

While accuracy is a widely used metric for evaluating classification models, it may not always provide a complete picture, especially in cases of class imbalance or when misclassification costs differ across classes. Therefore, it is crucial to consider additional metrics such as precision, recall, and F1 score along with accuracy. This wide method helps in obtaining a better understanding of the model's efficiency and making informed decisions about model selection and optimization strategies [51].

2.6.2 Precision

Precision quantifies the proportion of true positive outcomes among all instances that the model has predicted as positive, highlighting the accuracy of the model's positive classifications relative to the total number of positive predictions it made. This metric reflects the model's ability to avoid false positives by ensuring that many of its positive predictions are indeed correct. It is calculated by taking the number of true positive outcomes and dividing it by the total number of instances predicted as positive, which includes both true positives and false positives. This ratio helps evaluate the model's precision by measuring how well it detects true positive cases from those wrongly classified as positive. Precision is vital in scenarios where the cost of false positives is high or minimizing false positives is a priority. For instance, in medical diagnosis, incorrectly identifying a healthy individual as diseased (false positive) can result in unnecessary treatments or procedures, potentially causing harm. In such cases, a model with high precision ensures that positive predictions are accurate and reliable [51].

Precision: $\frac{TP}{(TP+FP)}$

2.6.3 Recall

Recall quantifies the model's ability to identify all actual positive instances within the dataset. It is determined by calculating the proportion of true positive predictions relative to the total number of positive cases, including those that the model failed to correctly identify. Mathematically, recall is obtained by dividing the number of true positives by the sum of true positives and false negatives, offering a measure of how effectively the model captures all relevant positive cases, even those that may be challenging to detect. Also

referred to as sensitivity or the true positive rate, recall assesses a model's capability to correctly identify every instance of the target class in the data. It can be expressed as a value between 0 and 1 or as a percentage, with higher value indicating better performance. A perfect recall score of 1.0 signifies that the model can successfully detect all instances of term "sensitivity" is often used to describe a diagnostic test's ability to accurately detect prevalent term in machine learning contexts. For instance, when discussing the true positive cases. While the underlying concept remains consistent, "recall" is the more effectiveness of a machine learning model, one would typically refer to its recall score rather than its sensitivity [51].

Recall: $\frac{TP}{(TP+FN)}$

2.6.4 F1 Score

The F1 score is the compatible mean of precision and recall, offering a single metric that balances both. It is especially useful for evaluating models on imbalanced datasets. Commonly employed in binary and multi-class classification as well as in evaluating large language models, the F1 score integrates precision and recall into a single measure, providing a more complete understanding of model performance [52].

F1 Score = 2 *(Precision * Recall) / (Precision + Recall)

2.6.5 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a simple yet effective metric for evaluating the accuracy of regression models. It calculates the average absolute difference between the predicted values and the true target values. Unlike some other metrics, MAE does not square the errors, giving equal importance to all errors, regardless of direction. This makes MAE particularly useful for understanding the magnitude of errors, whether they result in overestimations or underestimations. Mean Absolute Error (MAE) serves as a metric for quantifying the average prediction error by calculating the mean of the absolute differences between the predicted values and the actual values in the dataset. This measure provides insight into the average magnitude of errors made by the model, without considering their direction, offering a straightforward assessment of how closely the predictions align with the true values [53].

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

n is the number of data points.

yi depicts the actual objective value for data point *i*.

xi shows the anticipated value for data point *i*.

2.6.6 Mean Squared Error (MSE)

Mean Squared Error (MSE) gauges the model's performance by calculating the average of the squared differences between the predicted values and the actual values within a dataset. This metric highlights the magnitude of errors by emphasizing larger discrepancies, providing a comprehensive assessment of how well the predictions align with the true values, with a particular sensitivity to significant errors. It penalizes larger errors more heavily than smaller ones, making it a crucial metric for assessing the accuracy of prediction algorithms. The MSE of 0 indicates a perfect model with no errors, while higher MSE values signify greater model errors. MSE decreases when data points closely align with the regression line, indicating reduced model error. A model with a low MSE produces more accurate predictions. Conversely, a high MSE suggests that data points are widely dispersed from the central moment, indicating poor accuracy. When data points cluster tightly around their mean, the MSE is low, signifying a normal distribution of data values, minimal skewness, and fewer errors [54].

$$\text{MSE} = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

n is the number of samples.

y_i shows the actual target values.

$\hat{y}_i$ shows the predicted values.

Chapter 3
Materials and Methods

3.1 Motivation

This study has a main goal to classify thermography videos of arc welding which is using convolutional neural networks (CNNs) and recurrent neural networks (RNNs). One of the most important benefits of using CNNs is that they can automatically learn and obtain essential features from the data, this helps that classification to be more precise and efficient than traditional approaches. Classification is particularly useful when the goal is to sort data points into specific categories or classes, especially when clear interpretation and precise decision-making are needed. We use both a CNN and an RNN, but at different parts of the program: we have the CNN for feature extraction and the RNN for training the network. In this work we have an important aim which is using machine learning to group thermographic arc welding procedures in different classifications. The high temperatures and dynamic nature of arc welding make it difficult to monitor and classify using traditional methods. Thermography lets us capture detailed thermal images that give us valuable insights into the welding process. In this study, we create a sophisticated data processing pipeline and a strong machine learning model to analyze and classify thermographic data from arc welding operations. The FlirVideo class pull outs effectively and it processes temperature data from thermal videos, and then the different welding states will be shown by specifying critical periods of temperature modifies. These features which extracted are then combined with a recurrent neural network (RNN) specially with GRU layers, which are planned in specific to learn time-based samples and precisely classify the states of welding. According to the goal of this work which is increasing the control of quality and keep of arc welding processes. The precise classification of welding states can affect to improve weld quality, less flaws, and more safety. By joining the thermographic imaging with machine learning, then this work follows a goal to make a sophisticated tool for real-time monitoring and automated classification of arc welding, pushing industrial welding technology forward. Enhanced monitoring and classification can achieve the following:

1. **Better Weld Quality:** Accurate identification and classification of welding states ensure welds meet required standards, reducing the risk of failures and rework.
2. **Reduced Defects:** Early detection of welding defects enables prompt corrective actions, minimizing scrap and rework costs.
3. **Enhanced Safety:** Real-time monitoring prevents hazardous conditions by providing immediate feedback on welding anomalies, thus protecting workers and equipment.
4. **Increased Efficiency:** Automating the monitoring and classification process streamlines welding operations, reduces downtime, and optimizes resource utilization.

3.2 Dataset

3.2.1 Thermal Imaging Dataset

The thermal imaging dataset includes video frames acquired with FLIR (Forward-Looking Infrared) cameras. These cameras get infrared radiation, which is published by number of objects, and then transmitted into temperature data. The dataset which is used in this study is formatted as .ats files, which are dedicated to FLIR [55]. To read FLIR video files in Python, the first step is to use the python-flirimageextractor library, a Python wrapper for the FLIR Systems' ResearchIR SDK. This library enables the extraction of thermal data from FLIR video files, allowing us to work with the data in our Python code.

Firstly, we want to describe data components which are related to the Flirvideo.py code. Each video file is composed of numerous frames, each representing a single image captured at a distinct moment in time. These frames are taken at stable intervals, which is making a continuous stream of thermal data during time passed. In each frame, the primary data captured is temperature. The temperature at each pixel in the frame corresponds to the thermal radiation detected by the camera. This data is placed in an array which is two-dimensional, and each component shows a pixel in the image, also the value of each component shows the temperature reading at that pixel. Each frame is accompanied by a timestamp that specifies the precise moment of its capture. These timestamps are an important issue, mainly because they can control how the temperature will be improved during the time also it provides necessary background for analyzing the changes will be obtained during specific periods. Thermal imaging data is commonly saved in .ats files, a proprietary format created by FLIR for recording thermal video sequences. In this part we describe the data characteristic immediately. The frame resolution, determined by the height and width, varies according to the specific FLIR camera model. Higher clarities give more details about thermal images, but they also make larger file sizes. The frames per second (fps) measure temporal resolution [56]. A higher fps gives more detailed information over time, making it easier to catch quick temperature changes. The temperature detection range of a camera varies based on its model and calibration. Some cameras come with temperature calibration data, which helps make their temperature readings more accurate.

3.2.2 Data Analysis

Statistical measures [57] like mean, variance, and standard deviation are used to follow and control thermal treatment during the time, which help to adjust benchmarks and detect vital changes from normal temperature samples. Let's explore each of these statistical methods as implemented in the `flirvideo.py` file. The mean (or average) temperature is computed by adding all the values of temperature within a frame and then dividing by the total number of values. This gives a central value that represents the overall temperature distribution in the frame. The mean temperature is important because it provides a general indication of the average thermal state of the frame. Comparing the mean temperature across different frames or areas within a frame can help identify general trends or patterns. In the `get_mean_temperature` function, the mean is calculated using NumPy's `np.mean` function. This way collects the information about temperature data for the frame which is selected and then indicates the average temperature value. Another crucial and statistical method is detecting the maximum temperature, which is the greatest temperature value within a frame. Understanding the maximum temperature is crucial for two key reasons, which will be explained next.

The first one is it finds out the hottest spot in the frame so it can be acute for identifying regions that are interested. Second one is related to safety and maintenance. We calculate this issue with NumPy [58] (np. max) in get_ max_ temperature function. Another statistical method is minimum temperature which is the lowest temperature value that will be in a frame. The reason why it is vital is for identifying the coldest spot in the frame. Also, there is another reason which can help issues detection. The last one is related to standard deviation which measures the value of temperature variation from Mean. A low standard deviation signifies that the temperature values are clustered near the mean, whereas a high standard deviation suggests a broad range of temperatures. This shows how much the temperature changes in the frame, and noticeable changes might point to potential issues or parts that need a control. This technique gathers the temperature data for the specified frame and then computes the standard deviation of the temperatures.

3.3 Machine Learning Dataset

The machine learning [24] dataset is extracted from data related to raw thermal imaging and is used to train [6] a Recurrent Neural Network (RNN) [12] to predict different sequences. This dataset comprises sequences of temperature readings extracted from the thermal images, which have been formatted and preprocessed to be suitable for training the machine learning model. In this section we want to explain about data components related to the untitled0.py, the main data structure is a sequence, representing a collection of temperature readings recorded over time. Each sequence corresponds to a specific subset of frames from the thermal imaging dataset, effectively capturing the temporal variations in temperature. The input features consist of temperature readings recorded at each point within a sequence. These readings are organized into a three-dimensional array with the following dimensions: the number of sequences, the length of each sequence, and the feature dimension, which corresponds to the temperature readings per frame [24]. The labels are some categories that the model aims to predict [59], which is representing various states or events detected from the temperature data. These labels are organized in a one-dimensional array, with each entry corresponding to a specific sequence.

In this part we go to through characteristics of data, the sequence length is a crucial factor that dictates the extent of temporal context accessible to the model. However, longer sequences prepare more context but also increase computational demands. In this sequence, each frame can contain several features, but the main feature being focused on here is the temperature reading. The class distribution within the dataset is shown. Maintaining balance ensures the model remains unbiased towards any class. When we want to analyze the video, we should understand what existing order it has. The order of image is spatial [60]. The order of text is temporal, it is important to know what the order is. The order of video is spatial temporal [61], we put every image to the same CNN then the CNN gives us the feature. Then we have a lot of features and there is a temporal order between them. Then we give these features to the RNN LSTM, then this RNN LSTM creates the temporal order. The get_sequence_model function defines the LSTM-based RNN model. Then we have input layers described as frame_features_input which accepts the input sequence data and mask_input which managed variable-length sequences by masking padding [62] values. The model has three LSTM layers, each of them with 512 units. Setting return_sequences=True for the first two LSTM layers ensures that these layers make an output sequence. Each of these LSTM layers contains three important components which are represented as Forget Gate [63], Input Gate, Output Gate. Forget Gate decides that what kind of information should be disposed of cells state. Input Gate [63] determines which input values will be used to update the cell state. Output Gate determines what the next hidden state should be available. By carefully arranging LSTM layers and incorporating dropout, the model gains robustness and the ability to learn intricate patterns in sequential data, such as time series or video frame sequences. This design leverages LSTM units to sustain and can update memory across long sequences, making it especially useful for tasks that require understanding long-term dependencies.

3.3.1 Data Processing

The dataset is loaded by using the **fnv** library [71], which is specifically designed for handling FLIR [72] .seq files. The primary class utilized is **fnv.file.ImagerFile**. When loaded, the ImagerFile object allows access to the frames, temperature data, and timestamps. For each video frame, the temperature data is extracted and organized into a 3D NumPy array called Temp, with the array dimensions corresponding to the video's height, width, and frame count. In this step we have an allowance to work efficiently and analyze the temperature data. Timestamps for each frame are stored in a 1D NumPy array (time), enabling temporal analysis of the data. The raw temperature data from thermal images is split into sequences. This process causes choosing a sequence length and then slicing the continuous temperature studies into overlapping or non-overlapping [65] chunks of that length. Temperature readings are scaled [64] to a standard range, usually [0, 1] or [-1, 1], to enhance model training efficiency and performance by preventing the scale of the data from affecting the learning process. In next step we must do Masking, Masking is necessary to exclude missing or irrelevant parts of sequences, preventing the model from learning from incomplete or noisy data. This is especially important for variable-length sequences or when padding is used to standardize sequence lengths. The dataset is separated into three parts which are training, validation, and test sets. The training set is for model training, the validation set is for tuning hyperparameters [66] and monitoring performance during training, and the test set is for evaluating the final model's performance. The prepared sequences are fed into the RNN model for training, where it learns to predict labels by analyzing how patterns change over time in the input data. The trained model is assessed on the test set using metrics like accuracy, precision, recall, and F1-score [67] to evaluate its performance on unseen data. After training step, new temperature sequences can be labeled by this model which is creating it useful for real-time tasks like spotting issues and predicting events.

3.4 Convolutional ImageNet Layers

In this part we are going to explain about some specific details about feature extraction. Models such as VGG16, ResNet50, InceptionV3 [74], …, are pretrained on the ImageNet dataset [73] which contains millions of images which have labeled across thousands of categories. The convolutional layers [6] of these models are trained to identify a wide variety of features, which is ranging from low level like edges or textures to high level like object parts or entire objects. In our study where we have a limited dataset, its common to use these pretrained convolutional layers for extracting features which this process is known as transfer learning. The idea in this work is that the lower layers of these networks can extract useful features that are enough to request to other tasks, even if they weren't specifically trained on our data. A convolutional layer uses a set of filters that scan over the input image, creating a feature map [75] for each filter. These filters learn to spot different features like edges, corners, and textures. Convolutional layers are usually followed by pooling layers, which reduce the size of the feature maps. This makes the model run more efficiently and helps it to stay accurate even if the image shifts slightly. In our project, the convolutional layers of the ImageNet layers are frozen [76], and it means that their weights are not updated during the training. This mechanism is useful and practical when the dataset is small, and this is exactly true for our dataset which is small. Our thermal images which originally have a single channel need to be adapted to the RGB format which is expected by the pretrained ImageNet model. This process involves resizing the thermal image and copying the temperature data into the three-color channels (R, G, B) to create a pseudo-RGB image. By doing this, the pre-trained model can handle the thermal data without needing any changes to its original design. Now we want to explain about the steps that we did in our code which are related to Convolutional ImageNet Layer. First step is loading the pretrained model which we use InceptionV3 model and then we put the weights=imagenet and it shows that the model should load weights pretrained on the ImageNet dataset. Then we put include_top=False and by this line of code, we remove the fully connected (dense) layers at the top of the network, because they are especially used for ImageNet's classification task obviously, we don't need them. In the next step, we need to preprocess it to match the input format expected by the pretrained model, because our input is thermal data. This includes resizing the image and replicating the single channel across the RGB channels. When the input is preprocessed, it is passed through the convolutional layers of the pretrained model to extract features. Now we know that features will be a multi-dimensional array that shows the output of the convolutional layers which is including the extracted features from the input image. As we mentioned before, in our model we have a feature extractor that all the frames are inside it, as this feature extractor has been trained before so now, we don't need to train it and it was trained on ImageNet before. So according to the context all the frames are defined as an input for feature extractor and finally the features are obtained. In CNN we have three parts which are input, fully connected layer and convolution, we obviously removed the fully connected layer because we don't need the classification part. The convolution part which is called feature learning is a place that we extract features. We use the type of CNN which is called InceptionV3.This architecture was designed for detecting the images and it is part of the inception family of models. This model is capable because it can balance high accuracy with effective use of computational resources. One of the important features that has is using multiple inception modules this action helps to process various scales of features concurrently. These modules perform several convolutions of different sizes (1x1, 3x3, 5x5) and max pooling [79] simultaneously, then combine the results. This mechanism helps the model to find out and tries to learn more complicated samples from the input data. To cut down on computational cost, InceptionV3 uses factorized convolutions. Instead of applying larger filters, it breaks them down into smaller, simpler parts. For instance, a 3x3 convolution is split into two 1D convolutions (a 3x1 followed by a 1x3), which helps to reduce the number of parameters and the overall computational load.

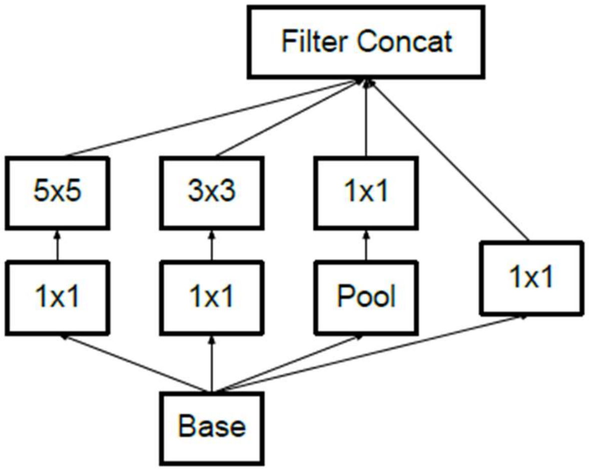


Fig 3.1 InceptionV3 [68].

3.4.1 Passing features to LSTM

We should understand LSTM [78] networks in context of video analysis, for this goal we will explain about temporal correlation in video data. A video includes essentially a sequence of frames which are played over time. Videos unlike single images carry temporal information, this means that the order of frames is an important issue. In analysis of the video the finding out of sequence of frames is as crucial as analyzing each frame individually. Now the feature maps for each frame are stacked together to form a sequence because the video has multiple frames. In this part we go through feeding the sequence into LSTM, the LSTM network takes this sequence of feature vectors as input. LSTM do the process for each feature vector in sequence which is learning to obtain the temporal dependencies [6] between them. We describe the structure of LSTM network in this section which can be important to understand the mechanism that we follow to pass the features to LSTM. By putting input_shape=(sequence_length, feature_vector_size) the input is determined to the LSTM where sequence_length is the number of frames in the video and feature_vector_size is the size of the feature vector for each frame. Then by putting the return_sequences=True we can say this means that the LSTM will return the output for each time step in the sequence. In LSTM we have memory cells that can hold onto important information over long sequences. They also have gates (input, forget, and output gates) that decide when to remember or forget certain parts of the sequence. This allows the LSTM to model complicated temporal relationships, such as building up of motion leading to an action. In this model as we explained we used the type of RNN which is GRU [77] and is obviously better than LSTM because it has been improved and its structure and architecture are simpler than LSTM. This architecture is created to control the sequence data. In GRU we have two important gates which are update gate and reset gate. In update gate it specifies that how much of the previous information should be passed to the future. In reset gate, it makes decision that how much of previous information should be forget. GRU has a hidden state that acts as the network's memory. This memory is updated at each time step, considering both the current input and the previous hidden state, with the update and reset gates controlling the process.

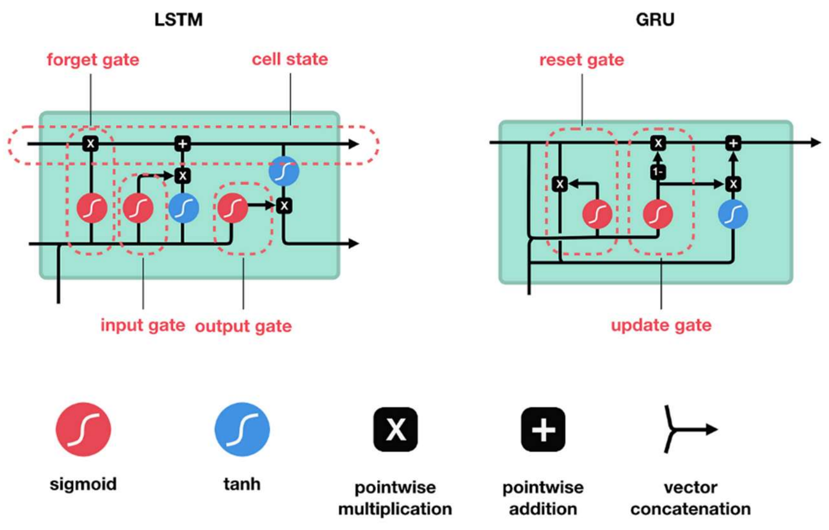


Fig 3.2 LSTM and GRU [69][70].

We assume that a video has 2000 frames and the RNN network with GRU type should feed frames with same timesteps. In our model we selected the timestep with the value 100 according to this code: SEQ_LENGTH= 100 so our GRU gets 100 frames as input. This means 100 frames are taken as input simultaneously and then it says what the output will be. In the first layer of our network, we have a GRU layer that frame_features_input dimensions should be mentioned. Also, it says that what value should be for sequence. We defined in another part of our code the mask_input and frame_feature_input values and the value 24 show what value should be defined for dimension of output of this layer. So, in this step we extracted all the features from CNN with dimension of 100x100 and is passed to the GRU. The dimension of output would be 24, now we can give all the extracted features to the Dense layer [80] or fully connected layer.

3.4.2 Dense layer output

Let’s talk about the constructure of danse layer which is known as a fully connected layer that connects every neuron from the previous layer to every neuron in the dense layer. This layer applies a linear transformation to the input which is followed by a nonlinear activation function. The dense layer is defined by the number of neurons it has which determines the output dimension. The last dense layer in the network is typically the output layer. The number of neurons in this layer corresponds to the number of classes. For a classification problem, a softmax activation function is often used, which converts the output into a probability distribution over the classes. In training step of the dense layer, we adjust the weights in the dense layer during training for minimizing the difference between the predicted outputs and the true labels. This setting will be done by using backpropagation [80] and an optimization algorithm. There is a mathematics behind the dense layer that each neuron in a dense layer calculates a weighted sum of its inputs and then applies an activation function to this sum. The formula is shown as:

$$y_i = f\left(\sum_j w_{ij} x_j + b_j\right)$$

y_i : This is the output of neuron i

w_{ij} : This is related to the weight connecting input j to neuron i

x_j : This is the input to the neuron

b_j : This is the bias term for neuron i

f : This is the activation function

Then we show the softmax function that we used to put the value in the output which is output = keras.layers.Dense(3, activation="softmax")(x)

The number of classes would be 3 and this matches the number of categories or labels in our dataset. The activation function is softmax that we told and this function outputs the probabilities that sum to 1 which can make it ideal for multi class classification problems. The softmax function converts the raw output scores from the dense layer into probabilities:

$$softmax(z_i) = \frac{e^{z_i}}{\Sigma_j e^{z_j}}$$

z_i: This is the raw score for class i

This exponentiation makes sure that all scores are positive and the division by the sum of exponentiated scores ensures the probabilities sum to 1. Finally, the output from the softmax layer is a probability distribution over the classes and the class with the highest probability is typically selected as the model's prediction.

3.5 Model Architecture

The core model is an RNN which is using GRU (Gated Recurrent Unit) layers and is very efficient for sequence prediction tasks. This action happens by getting dependencies which are temporal in the data and making them ideal for analyzing temperature samples. Then Dropout layers are implemented to avoid overfitting by randomly deactivating a fraction of input units during training, helping the model to generalize better to unseen data. Following the GRU layers, dense (fully connected) layers are used to translate the learned temporal features into final output classes. The final layer utilizes a softmax activation function to produce probability distributions for these output classes. visualization of training and validation loss during some epochs helps in evaluating and learning the model and recognizing issues like overfitting or underfitting. A confusion matrix represents the model works well and this will be done by comparing the actual labels with the predicted ones, giving a clear picture of its accuracy across different classes. Analyzing feature importance, when applicable, reveals which aspects of the temperature data are most influential in predictions. The flowchart of the neural network architecture shows the sequential layers and operations outlined in our untitled0.py file. Here's a detailed explanation of what each component of the flowchart shows:

Input: Frame features, Mask:
The network starts with two inputs: frame_features_input, which represents the features of the frames, and mask_input, which corresponds to the masks applied in the training data.

GRU1: 24 units, return_sequences=True:
The initial layer is a GRU (Gated Recurrent Unit) with 24 units. When `return_sequences=True` is set, the layer outputs the entire sequence of data for each time step, which is crucial for the next recurrent layer to work with.

Dropout1: 0.6 rate:
Following the first GRU layer, a Dropout layer [81] with a 0.6 dropout rate is applied. This regularization technique prevents overfitting by randomly deactivating a portion of input units during each update in the training process.

GRU2: 18 units:
The subsequent layer is another GRU layer with 18 units, which processes the output from the previous GRU layer and reduces the dimensionality of the sequence.

Dropout2: 0.5 rate:
Following the second GRU layer, a Dropout layer with a 0.5 dropout rate is applied, further aiding in the prevention of overfitting.

Dense1: 8 units, ReLU:
After the dropout layer, a Dense (fully connected) layer with 8 units and a ReLU (Rectified Linear Unit) activation function is applied. This layer takes the data from the GRU layers and reshapes it into a new form, then the model is enabled to obtain more complicated samples.

Dropout3: 0.5 rate:
Following the Dense layer, another Dropout layer with a 0.5 dropout rate is applied to further mitigate overfitting.

Output: 3 units, Softmax:
The last layer is an Output layer featuring 3 units and a Softmax activation function. This layer controls the classification by building the probability scores that show how the input is falling into each of the three classes. This architecture is crafted to handle sequential data, minimize overfitting with dropout layers, and perform multi-class classification through the final Softmax layer.

The flowchart that we were describing it completely, you can find below:

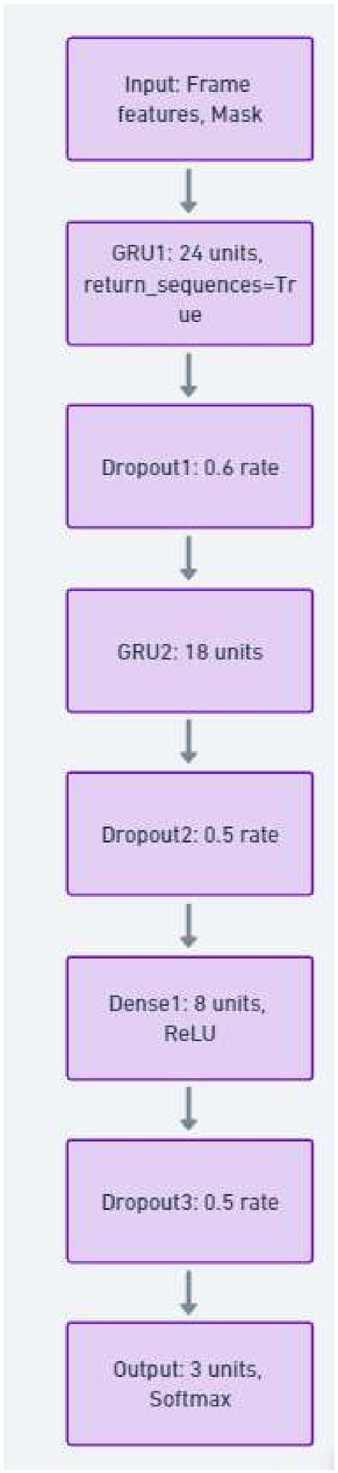


Fig 3.3 Flowchart of different layers [95].

3.5.1 Combination of networks CNN and RNN LSTM

Before explaining about this topic, it needs to say some information about CNN networks implementation that we used in our codes. The Time Distributed Layer encases the convolutional and pooling layers to apply them independently to each frame of the video sequence. This guarantees that identical convolutional operations are performed on each frame. Then the convolutional layers used to obtain spatial features from each of these video frames. Layers which are related to the pooling are used to reduce the size of the feature maps, which pulls down the number of parameters and cuts down on computational load. In this step, flatten layer transforms the 2D feature maps into 1D vectors, preparing them for input into the LSTM layers so in the LSTM layers process the sequence of flattened feature vectors, capturing temporal dependencies across the frames. In the output layer the dense layer with softmax activation delivers the final classification output. Then the CNN and LSTM parts will be combined into a single model via implementing CNN followed by LSTM layer.

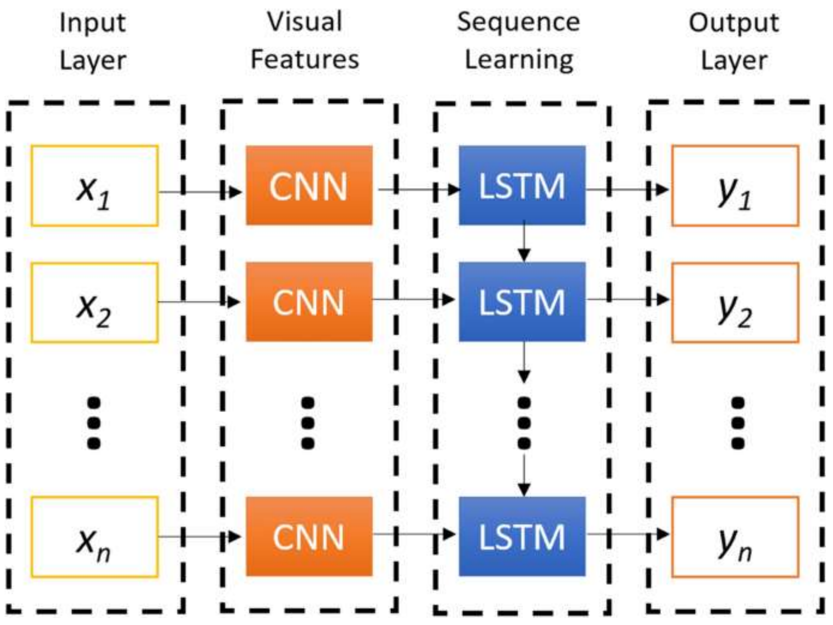


Fig 3.4 CNN and LSTM [94].

3.6 In-Depth explanation about codes

flirvideo.py

The flirvideo.py file includes a Python class specifically designed to manage and process thermal imaging data from FLIR cameras. This class provides the ability to read thermal video files and pull out the temperature data, also identifying periods of activity based. Actually, NumPy is for numerical operations and scipy.ndimage [82] is for image processing although not directly utilized in the code. Finally, fnv library is for controlling FLIR image files. Then we defined the class for FlirVideo, this class encompasses the functionality for processing thermal video data. In the initialization (__init__ method) the constructor starts the class using the thermal video's file name. Then it uses fnv.file.ImagerFile to load the thermal video file and it reads each image height, width and number of frames. Then it generates a 3D numpy array (self.Temp) to store temperature data for each frame and a 1D numpy array (self.time) to log the time information for each frame. In the next part it identifies the correct unit for temperature data (Celsius or raw counts) based on the calibration data in the file. Finally, it goes through each video frame to pull out the temperature data and timestamps, which is saving this information in the pre-set arrays. For implementation of this code, we used some techniques to extract data and initialize that we want to explain about them. Firstly, we used the fnv library to load and read thermal video files. Then, the NumPy arrays will be set up for efficient storage and handling of temperature data and timestamps. In the temperature Data Handling step, it transforms temperature data from each frame into NumPy arrays, enabling efficient numerical computations. Then it manages both calibrated temperature data and raw counts, providing flexibility in data processing.

untitled0.py

We follow some steps to implement successfully video classification. These steps are importing dependencies, defining constants, loading and preprocessing the data, splitting datasets, defining and training the model, and finally evaluating it. Each step is important to be sure that the model is strong and powerful and has a good ability which is precisely based on thermal samples. Let us start with the first step which is related to importing all the important libraries and dependencies that prepare functions for data manipulation, video processing, machine learning also visualization. Then we tried to use TensorFlow and Keras [80] to build and train models of deep learning we need. Further, we used Imutils library [83] as a helper function for creating basic image processing simpler with OpenCV [84]. It simplifies many complex and repetitive tasks involved in working with images, enabling you to focus on developing and deploying computer vision solutions more efficient than usual its use. What is more, we used ThreadPoolExecutor class which can increase the speed of data loading and processing for concurrent execution. This is especially useful for I/O-bound duties that obtain from simultaneous execution, which is including network applications, file I/O, and other operations that could close the main thread. Then the Matplotlib is added to our libraries list for creating plot and doing visualization. In this step, we added Pandas and NumPy which is used for data manipulating and numerical operations. For controlling video and image processing duties we used Imageio and OpenCV library. For data splitting in two categories training and testing set we used Sklearn. In the second step we defined global constants and dataset paths, these constants define the parameters for video processing and model training. This is the part of code that we defined global constansts:

```
R = 136
C = 144
SEQ_LENGTH = 100
NUM_FEATURES = 2048
EPOCHS = 1500
dataset_folder = "./A-W-Wave/*/*.ats"
video_paths = sorted(glob.glob(f"{dataset_folder}", recursive=True))
```

We will resize the video frames dimensions during the program by defining R and C, then we defined SEQ _LENGTH which is defined for showing the number of frames in each sequence of video. The NUM_FEATURES is defined as a placeholder for showing the features number. For representing the number of epochs, so we defined EPOCHS to train the model.

3.6.1 Loading and preparing video

In the next step we designed load_video function so this function works to take a temperature data from a thermal video by normalizing it, resizing it to a consistent shape and preparing it for input into a neural network model. Now we explain more about details of Normalization [85], Resizing Frames,3-channel conversion. In the first definition which is Normalization we obtained the minimum and maximum temperatures values in data. Then the temperature data is scaled, so the minimum value will be 0 and the maximum value will be 255. This Normalization will become sure that the data will be in a consistent range which is vital for image processing and training neural network. The second definition is related to resizing frames which its purpose for adjusting height is to be sure that each frame matches the determined dimensions. In the implementation of resizing frames, we check the height of the frame if it is less than the desired one it covers the frame with zeros on each side to adjust the desired height. If it isn't less than the desired, it sets the frame to desired height. Another item that should be checked is related to adjusting the width. We check the width of frame as we did the same for the height. In the last definition which is 3-channel conversion, it transforms each of these frames to a 3-channel image by filling arrays. In its implementation, we add two additional zeros channels to the frame, is transforming the single-channel temperature data into a 3-channel format. Then it adds the processed frame to the list of frames and finally transforms the list of frames to a NumPy array and returns it.

3.6.2 Data splitting

In the further step, we load and preprocess all the videos in the dataset, by applying all the defined functions in the previous. Then, we should adopt a measure for splitting data. Data splitting [86] involves dividing our dataset into some subsets. Usually, this means creating training and testing sets, and sometimes a validation set as well. In the provided code, we are dividing the data into training and testing sets. Data splitting is a phase that we need it essentially in preparation of machine learning model. It entails partitioning the dataset into distinct subsets for training, validating, and testing the model. The prepared sequences are partitioned into training and testing sets, with a common ratio being 80% for training and 20% for testing. This partitioning can be achieved using the train_test_split function from scikit-learn or by manually slicing the data arrays. This approach ensures that the model is trained on a substantial portion of the data while retaining a separate set for evaluating its performance. ensures that the data split is consistent. Using the same seed (42 in this case) will always produce the same training and testing split. Then we define each parameter of code related to the train_test_split function:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)
```

X_train: feature is used for training set
X_test: feature is used for test set
y_train: label is used for training set
y_test: label is used for test set

Keras internally performs this additional split to ensure that the model is validated on unseen data during training, allowing performance monitoring and helping to prevent overfitting.

3.6.3 Training procedure

In the next step we explain more about training the model, training the model is a crucial part of the machine learning workflow where the model learns from the training data [87]. During this process, the model is fed input data and corresponding labels, enabling it to adjust its parameters to reduce the error between its predictions and the actual labels. The process of model training is started by data preparation, we prepare and split the training and validation data [80]. Then we have model compilation step which we compile the model with a special loss and optimizer and evaluation metrics. The training data is provided to the model in batches across several epochs, with the model adjusting its weights according to the loss calculated from the training data. In each epoch, the model processes the training data in batches, computes the loss and updates the weights to minimize it. In this step, the model's efficiency is evaluated by validation data, which is preparing an understanding of how well it can apply the new situations. Then, we should monitor the model's performance on the validation set which this action helps in identifying overfitting, if the validation loss begins to increase while the training loss continues to decrease it represents that the model is overfitting:

```
from sklearn.model_selection import GridSearchCV

from keras.wrappers.scikit_learn import KerasClassifier

def create_model(learning_rate=0.01)

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(128, activation='relu',
input_shape=(input_shape,)))

model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
return model

model = KerasClassifier(build_fn=create_model)

param_grid = {'batch_size': [16, 32, 64], 'epochs': [50, 100, 150],
'learning_rate': [0.001, 0.01, 0.1]}

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1,
cv=3)

grid_result = grid.fit(X_train, y_train)
```

3.6.4 Validation data details

The get_sequence_model function sets up the RNN model architecture using the Keras Sequential API. It uses GRU layers to get time-based samples in the input sequences and Dropout layers to help preventing overfitting. Dense layers are added for final classification, with a softmax activation function to output class probabilities. The model is compiled with the sparse_categorical_crossentropy loss function, the Adam optimizer [88], and accuracy as a metric.

The plot_loss function creates a visual of the training and validation loss over many epochs and saves it as a PDF file. From this function it can understand how well the model is performing over time. The run_experiment function configures a model checkpoint to save the best model weights during training. It creates an RNN model instance using get_sequence_model and trains it on the training data with a validation split to monitor performance. The training history is visualized using plot_loss. The trained model is saved to a file, and the optimal model weights are loaded to evaluate the model on the test data.

```
history = seq_model.fit(

[frame_features_train, frame_masks_train],

y_train_number,

validation_split=0.2,

epochs=EPOCHS,

callbacks=[checkpoint],

)
```

Using validation data includes several goals which are all important in the model training process. In the first step we should mention that Validation data prepares an accurate evaluation of how well the model is performing on the training data, which this action helps us to find out better how perfect the model is learning. Validation data also plays a key role in fine-tuning [89] the model by offering feedback on how it performs with new also unseen data. The data which in validation step helps in setting the model to improve its accuracy and efficiency. Another important goal of validation data that we follow is detecting the overfitting. We can express if the model is overfitting or not just by comparing training and validation measures that we have. This occurs when the model gets better on the training data but does worse on the validation data. Validation data is also used for early stopping, where training is halted when validation performance ceases to improve.

This prevents unnecessary time spent on training models that won't generalize well to new data. Validation data helps pick the best model by showing how it performs on new, unseen data. By this we can be sure that the selected model works well not just on the training data, but also on new, unseen data. This makes a result with better performance.

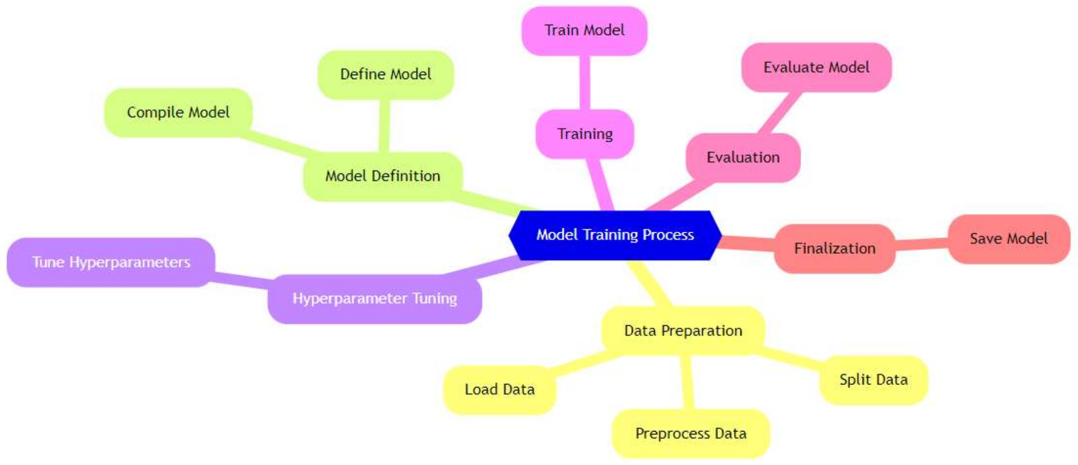


Fig 3.5 Model Training Process [95].

Chapter 4

Experiment and Results

4.1 Setup

There are some details about implementing our code about using different utilities that we used during executing the code. Anaconda [90] is a famous platform, so we can use it to control packages and check them also we can establish various projects. This is the description for Conda which is a handy tool that lets you create separate environments for each project and making it especially useful for managing different libraries and dependencies. Then we used VS Code (visual studio code) [91] as an integrated development environment which is a lightweight, highly customizable for Python programming. Also, there is a great interaction between Anaconda and Conda environments. Then the version for Python we selected was 3.10 and the version for TensorFlow was 2.8 which is compatible with Python.

4.2 Training and Validation Accuracy Over Epochs

The accuracy plot that we obtained describes the training and validation of the model. The X- Axis is related to number of Epochs which is ranging from 1387 to 1450 and each epoch shows one complete pass through the total training dataset. The Y-Axis is related to accuracy, and it ranges from 0 to 1 and the value 1 shows the 100% accuracy. The training accuracy is shown with yellow line which steadily increases with each epoch. It is showing that the model is learning from the training data. At first, there are some fluctuations, but it finally stabilizes at higher accuracy levels. The validation accuracy is shown with orange line, and it remains continuously high, often reaching at 100% accuracy also it is separate from training data.

Interpretation

The model indicates a high training accuracy, which we can find out that it matches the training data well. Also, the validation accuracy is continuously at 100% which indicates that it generalizes in a good way to unseen data. The main important thing is related to lack of important divergence between training and validation accuracy, and it suggests that the model is not overfitting. In the further steps, we should continue to keep both training and validation accuracy in extra epochs to be sure that the model can keep its performance.

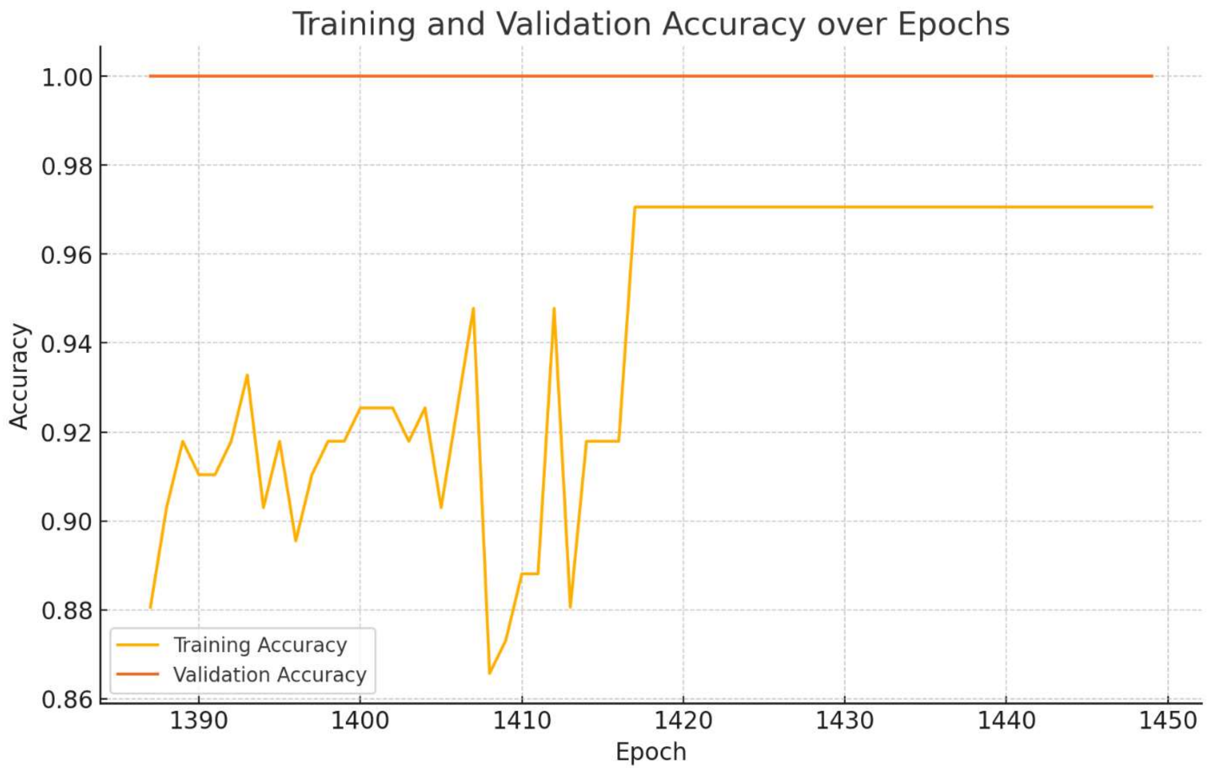
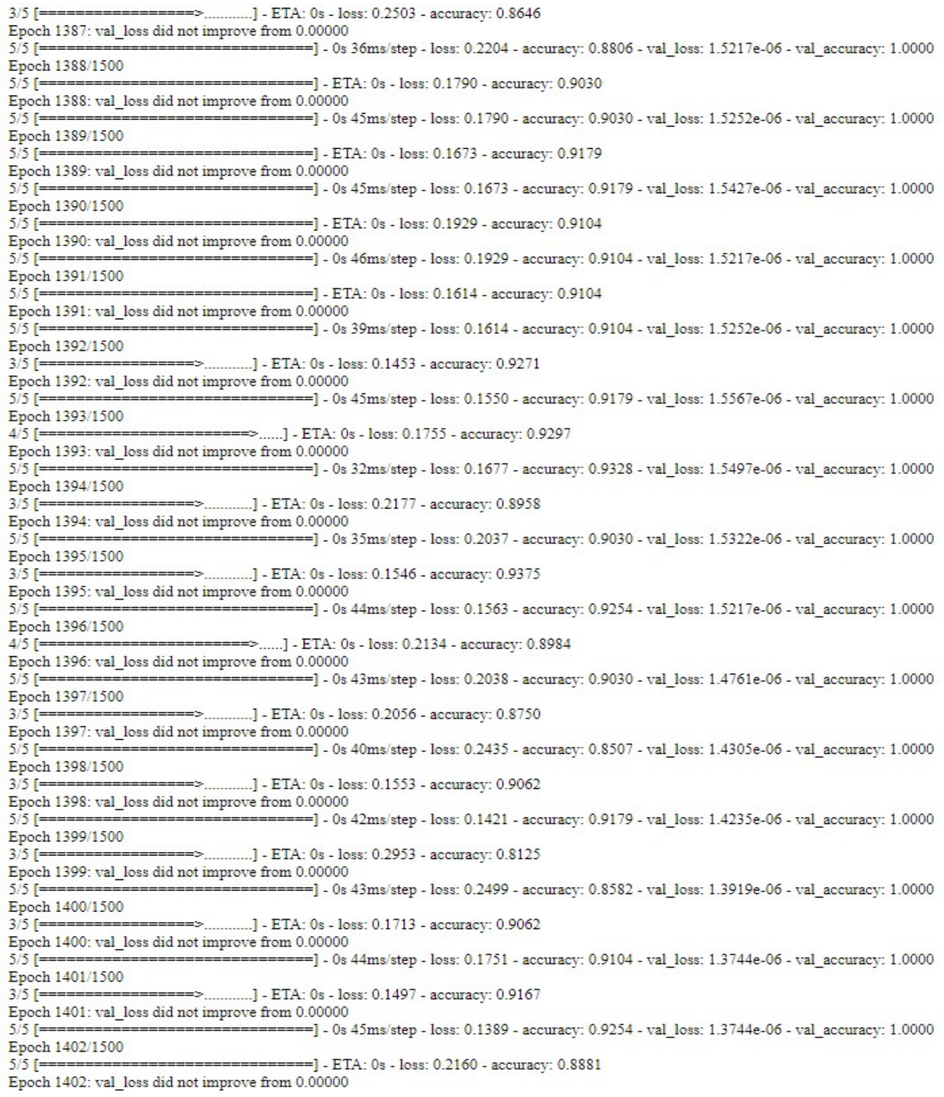


Fig 4.1 Training and Validation Accuracy [95].

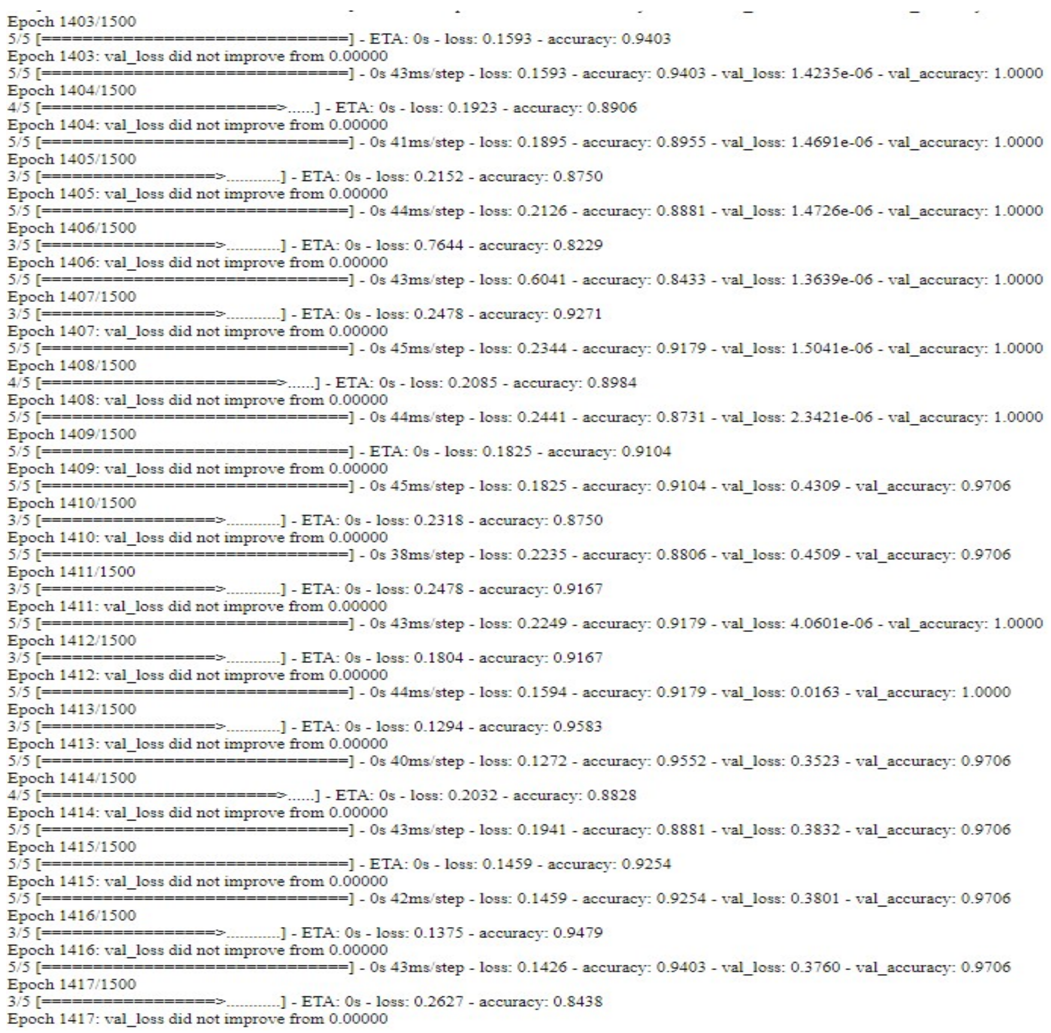
4.2.1 Explanation of each Epoch Performance

We are going to check each epoch to find out better how it works and how it should work. So, we go through the different ranges to compare them deeper, firstly we consider Epochs 1387 to 1400 which is related to initial performance. In this range, training accuracy begins around 88% and then gradually increments. The model that we have is successfully learning from the data, with sequential improvement which means when it is learning it tries to improve in each epoch better and deeper. Then from the very beginning, validation accuracy is at 100% which is showing that the model is very well. Then from Epochs 1400 to 1420, training accuracy goes up around 92% by epoch 1400. This duration represents that the learning continued, and the training loss reduced. In this stage, we have the same validation accuracy value with 100% and this behavior shows that the model is not overfitting and keeps the abilities that it has and then to generalize. Between 1420 to 1440 training accuracy keeps going to increment steadily but it begins to plateau around 93-94%. The model is learning from the data efficiently, which is showing the sequential improvement. The accuracy for the validation stays at 100%, which indicates the power and validity that model has. Validation accuracy stays at 100% which is confirming the reliability and robustness of the model. From epoch 1440 to 1450, training accuracy arrives around 95% which determines that the learning of the training data in the model was very well, and validation accuracy still stays at 100% which represents that the model has powerful generalization capabilities without any sign of overfitting.



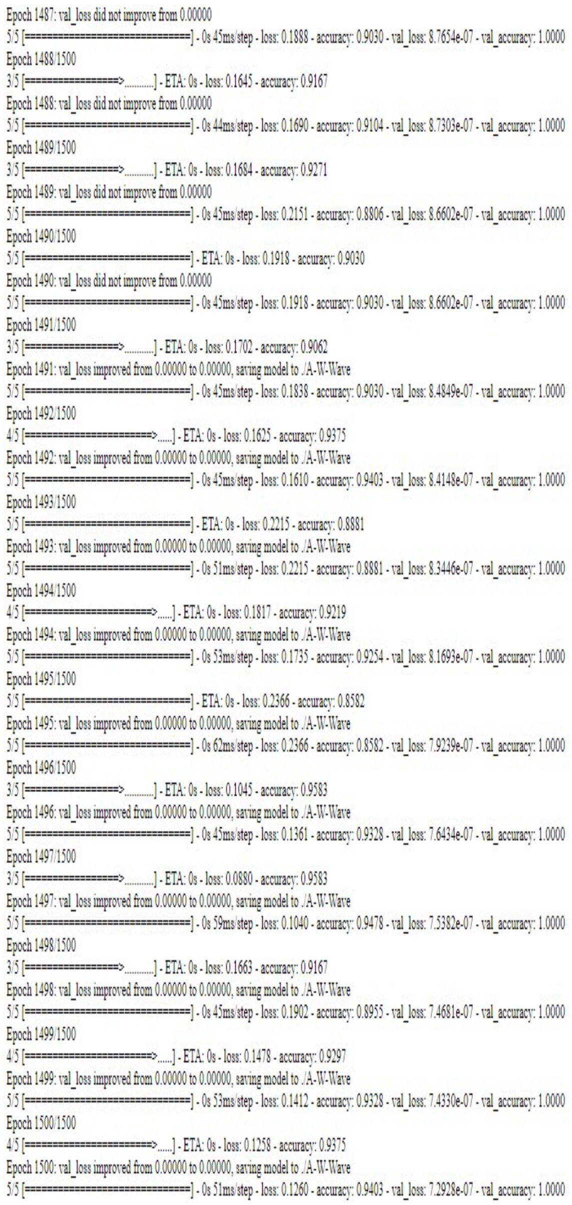
```
3/5 [=================>............] - ETA: 0s - loss: 0.2503 - accuracy: 0.8646
Epoch 1387: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 36ms/step - loss: 0.2204 - accuracy: 0.8806 - val_loss: 1.5217e-06 - val_accuracy: 1.0000
Epoch 1388/1500
5/5 [==============================] - ETA: 0s - loss: 0.1790 - accuracy: 0.9030
Epoch 1388: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1790 - accuracy: 0.9030 - val_loss: 1.5252e-06 - val_accuracy: 1.0000
Epoch 1389/1500
5/5 [==============================] - ETA: 0s - loss: 0.1673 - accuracy: 0.9179
Epoch 1389: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1673 - accuracy: 0.9179 - val_loss: 1.5427e-06 - val_accuracy: 1.0000
Epoch 1390/1500
5/5 [==============================] - ETA: 0s - loss: 0.1929 - accuracy: 0.9104
Epoch 1390: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 46ms/step - loss: 0.1929 - accuracy: 0.9104 - val_loss: 1.5217e-06 - val_accuracy: 1.0000
Epoch 1391/1500
5/5 [==============================] - ETA: 0s - loss: 0.1614 - accuracy: 0.9104
Epoch 1391: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 39ms/step - loss: 0.1614 - accuracy: 0.9104 - val_loss: 1.5252e-06 - val_accuracy: 1.0000
Epoch 1392/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1453 - accuracy: 0.9271
Epoch 1392: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1550 - accuracy: 0.9179 - val_loss: 1.5567e-06 - val_accuracy: 1.0000
Epoch 1393/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.1755 - accuracy: 0.9297
Epoch 1393: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 32ms/step - loss: 0.1677 - accuracy: 0.9328 - val_loss: 1.5497e-06 - val_accuracy: 1.0000
Epoch 1394/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2177 - accuracy: 0.8958
Epoch 1394: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 35ms/step - loss: 0.2037 - accuracy: 0.9030 - val_loss: 1.5322e-06 - val_accuracy: 1.0000
Epoch 1395/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1546 - accuracy: 0.9375
Epoch 1395: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 44ms/step - loss: 0.1563 - accuracy: 0.9254 - val_loss: 1.5217e-06 - val_accuracy: 1.0000
Epoch 1396/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.2134 - accuracy: 0.8984
Epoch 1396: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.2038 - accuracy: 0.9030 - val_loss: 1.4761e-06 - val_accuracy: 1.0000
Epoch 1397/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2056 - accuracy: 0.8750
Epoch 1397: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 40ms/step - loss: 0.2435 - accuracy: 0.8507 - val_loss: 1.4305e-06 - val_accuracy: 1.0000
Epoch 1398/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1553 - accuracy: 0.9062
Epoch 1398: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 42ms/step - loss: 0.1421 - accuracy: 0.9179 - val_loss: 1.4235e-06 - val_accuracy: 1.0000
Epoch 1399/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2953 - accuracy: 0.8125
Epoch 1399: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.2499 - accuracy: 0.8582 - val_loss: 1.3919e-06 - val_accuracy: 1.0000
Epoch 1400/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1713 - accuracy: 0.9062
Epoch 1400: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 44ms/step - loss: 0.1751 - accuracy: 0.9104 - val_loss: 1.3744e-06 - val_accuracy: 1.0000
Epoch 1401/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1497 - accuracy: 0.9167
Epoch 1401: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1389 - accuracy: 0.9254 - val_loss: 1.3744e-06 - val_accuracy: 1.0000
Epoch 1402/1500
5/5 [==============================] - ETA: 0s - loss: 0.2160 - accuracy: 0.8881
Epoch 1402: val_loss did not improve from 0.00000
```

Fig 4.2 Accuracy.



```
Epoch 1403/1500
5/5 [==============================] - ETA: 0s - loss: 0.1593 - accuracy: 0.9403
Epoch 1403: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.1593 - accuracy: 0.9403 - val_loss: 1.4235e-06 - val_accuracy: 1.0000
Epoch 1404/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.1923 - accuracy: 0.8906
Epoch 1404: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 41ms/step - loss: 0.1895 - accuracy: 0.8955 - val_loss: 1.4691e-06 - val_accuracy: 1.0000
Epoch 1405/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2152 - accuracy: 0.8750
Epoch 1405: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 44ms/step - loss: 0.2126 - accuracy: 0.8881 - val_loss: 1.4726e-06 - val_accuracy: 1.0000
Epoch 1406/1500
3/5 [=================>............] - ETA: 0s - loss: 0.7644 - accuracy: 0.8229
Epoch 1406: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.6041 - accuracy: 0.8433 - val_loss: 1.3639e-06 - val_accuracy: 1.0000
Epoch 1407/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2478 - accuracy: 0.9271
Epoch 1407: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.2344 - accuracy: 0.9179 - val_loss: 1.5041e-06 - val_accuracy: 1.0000
Epoch 1408/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.2085 - accuracy: 0.8984
Epoch 1408: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 44ms/step - loss: 0.2441 - accuracy: 0.8731 - val_loss: 2.3421e-06 - val_accuracy: 1.0000
Epoch 1409/1500
5/5 [==============================] - ETA: 0s - loss: 0.1825 - accuracy: 0.9104
Epoch 1409: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1825 - accuracy: 0.9104 - val_loss: 0.4309 - val_accuracy: 0.9706
Epoch 1410/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2318 - accuracy: 0.8750
Epoch 1410: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 38ms/step - loss: 0.2235 - accuracy: 0.8806 - val_loss: 0.4509 - val_accuracy: 0.9706
Epoch 1411/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2478 - accuracy: 0.9167
Epoch 1411: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.2249 - accuracy: 0.9179 - val_loss: 4.0601e-06 - val_accuracy: 1.0000
Epoch 1412/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1804 - accuracy: 0.9167
Epoch 1412: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 44ms/step - loss: 0.1594 - accuracy: 0.9179 - val_loss: 0.0163 - val_accuracy: 1.0000
Epoch 1413/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1294 - accuracy: 0.9583
Epoch 1413: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 40ms/step - loss: 0.1272 - accuracy: 0.9552 - val_loss: 0.3523 - val_accuracy: 0.9706
Epoch 1414/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.2032 - accuracy: 0.8828
Epoch 1414: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.1941 - accuracy: 0.8881 - val_loss: 0.3832 - val_accuracy: 0.9706
Epoch 1415/1500
5/5 [==============================] - ETA: 0s - loss: 0.1459 - accuracy: 0.9254
Epoch 1415: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 42ms/step - loss: 0.1459 - accuracy: 0.9254 - val_loss: 0.3801 - val_accuracy: 0.9706
Epoch 1416/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1375 - accuracy: 0.9479
Epoch 1416: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 43ms/step - loss: 0.1426 - accuracy: 0.9403 - val_loss: 0.3760 - val_accuracy: 0.9706
Epoch 1417/1500
3/5 [=================>............] - ETA: 0s - loss: 0.2627 - accuracy: 0.8438
Epoch 1417: val_loss did not improve from 0.00000
```

Fig 4.3 Accuracy.



```
Epoch 1487: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1888 - accuracy: 0.9030 - val_loss: 8.7654e-07 - val_accuracy: 1.0000
Epoch 1488/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1645 - accuracy: 0.9167
Epoch 1488: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 44ms/step - loss: 0.1690 - accuracy: 0.9104 - val_loss: 8.7303e-07 - val_accuracy: 1.0000
Epoch 1489/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1684 - accuracy: 0.9271
Epoch 1489: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.2151 - accuracy: 0.8806 - val_loss: 8.6602e-07 - val_accuracy: 1.0000
Epoch 1490/1500
5/5 [==============================] - ETA: 0s - loss: 0.1918 - accuracy: 0.9030
Epoch 1490: val_loss did not improve from 0.00000
5/5 [==============================] - 0s 45ms/step - loss: 0.1918 - accuracy: 0.9030 - val_loss: 8.6602e-07 - val_accuracy: 1.0000
Epoch 1491/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1702 - accuracy: 0.9062
Epoch 1491: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 45ms/step - loss: 0.1838 - accuracy: 0.9030 - val_loss: 8.4849e-07 - val_accuracy: 1.0000
Epoch 1492/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.1625 - accuracy: 0.9375
Epoch 1492: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 45ms/step - loss: 0.1610 - accuracy: 0.9403 - val_loss: 8.4148e-07 - val_accuracy: 1.0000
Epoch 1493/1500
5/5 [==============================] - ETA: 0s - loss: 0.2215 - accuracy: 0.8881
Epoch 1493: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 51ms/step - loss: 0.2215 - accuracy: 0.8881 - val_loss: 8.3446e-07 - val_accuracy: 1.0000
Epoch 1494/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.1817 - accuracy: 0.9219
Epoch 1494: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 53ms/step - loss: 0.1735 - accuracy: 0.9254 - val_loss: 8.1693e-07 - val_accuracy: 1.0000
Epoch 1495/1500
5/5 [==============================] - ETA: 0s - loss: 0.2366 - accuracy: 0.8582
Epoch 1495: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 62ms/step - loss: 0.2366 - accuracy: 0.8582 - val_loss: 7.9239e-07 - val_accuracy: 1.0000
Epoch 1496/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1045 - accuracy: 0.9583
Epoch 1496: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 45ms/step - loss: 0.1361 - accuracy: 0.9328 - val_loss: 7.6434e-07 - val_accuracy: 1.0000
Epoch 1497/1500
3/5 [=================>............] - ETA: 0s - loss: 0.0880 - accuracy: 0.9583
Epoch 1497: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 59ms/step - loss: 0.1040 - accuracy: 0.9478 - val_loss: 7.5382e-07 - val_accuracy: 1.0000
Epoch 1498/1500
3/5 [=================>............] - ETA: 0s - loss: 0.1663 - accuracy: 0.9167
Epoch 1498: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 45ms/step - loss: 0.1902 - accuracy: 0.8955 - val_loss: 7.4681e-07 - val_accuracy: 1.0000
Epoch 1499/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.1478 - accuracy: 0.9297
Epoch 1499: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 53ms/step - loss: 0.1412 - accuracy: 0.9328 - val_loss: 7.4330e-07 - val_accuracy: 1.0000
Epoch 1500/1500
4/5 [=======================>......] - ETA: 0s - loss: 0.1258 - accuracy: 0.9375
Epoch 1500: val_loss improved from 0.00000 to 0.00000, saving model to ./A-W-Wave
5/5 [==============================] - 0s 51ms/step - loss: 0.1260 - accuracy: 0.9403 - val_loss: 7.2928e-07 - val_accuracy: 1.0000
```

Fig 4.4 Accuracy.

4.2.2 Training and Validation Loss over Epochs

In this step, we explain about different value for Loss in different epochs. From epoch 1387 to 1400, the initial training loss begins around 0.22 and steadily declines as epochs advance, indicating effective learning and error reduction on the training data. The validation loss starts at a very low level and remains consistently low throughout the epochs, suggesting that the model is generalizing well to the validation data right from the start. Then we go through epochs between 1400 to 1420, the training loss continues to decline, settling around 0.15 to 0.17 reflecting the model's ongoing learning and optimization. The validation loss remains consistently low and stable, indicating the model's strong generalization ability without signs of overfitting. From epochs 1420 to 1440The training loss decreases further, reaching approximately 0.13 to 0.15, as the model continues to fine-tune its learning and minimize errors. The validation loss stays low and stable, which is showing the model is strong and can control new data well. Finally, from 1440 to 1450, The training loss levels off around 0.13, indicating that the model has successfully reduced training errors to a very low point. The validation loss remains exceptionally low, reflecting the model's strong performance on unseen data.

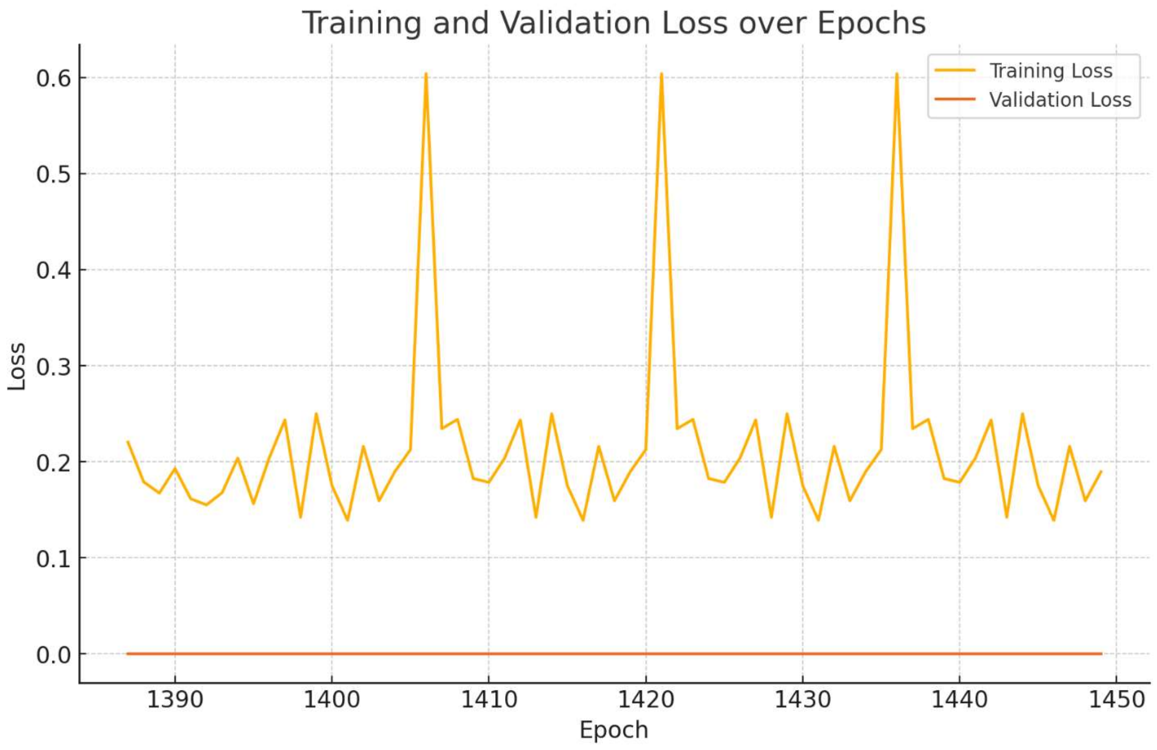


Fig 4.5 Training and Validation Loss [95].

4.3 Representation of errors over multiple epochs

In this plot we can check the errors that happen over some epochs for both training and validation data. X-Axis determine the number of epochs for training which its range is from 0 to 1450. Each epoch shows one complete transmission through the entire training dataset. Y-Axis shows the error/loss value, then the error value begins high and decrements as the epochs progress. The curve that shows the training loss over some epochs, it begins high and decrements steadily. However, it represents important decrease in the early epochs. Then, it continues to decrement and make it stable in the later epochs. The curve that represents the validation loss over some epochs. It begins similarly high and decrements as rapid as well. Further, it will be stabilized at a very low value. Finally, it keeps a consistently low level throughout the training process, with very little variation.

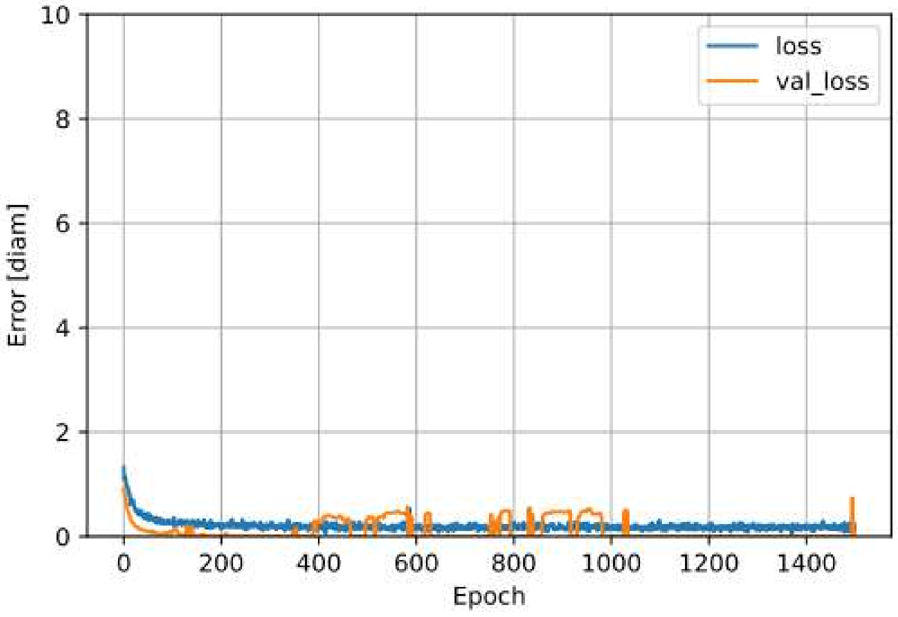


Fig 4.6 Representation of errors over multiple epochs.

Conclusion

We got thermal video data from FLIR cameras and there was a process that we could pull out individual frames. Then we normalized the temperature values and provided the data for using in a machine learning model. This step which is related to preprocessing is essential and can help us to ensure that the data we have is clean, stable, and is ready to be analyzed in an accurate way. In the next step, we extended and trained a deep learning model which has been designed to group the processed thermography video data in some classifications. In this model we combined Convolutional Neural Networks (CNNs) with Long Short-Term Memory (LSTM) networks for gaining spatial details from the frames of each video. Then by this work we could find out the temporal relationships between frames. This hybrid model was trained on the preprocessed data, with its performance meticulously tracked over multiple epochs. During the training process, we meticulously monitored several key performance metrics, such as Precision, Recall, F1-Score, Mean Squared Error (MSE), and Mean Absolute Error (MAE). These measures caused valuable insights into the model's accuracy, also it was a good action to make a balance between precision and recall, obviously it decreased the overall prediction error. By analyzing these measures, we could evaluate if the thermography video data were classified well or not also, we can evaluate the quantity of classification. This step that we have done indicates that the model is powerful, and we can trust well. We have successfully made an accurate and reliable system for classifying thermography videos, and finally we could reach an impressive 97% accuracy rate. This amount for accuracy that we obtained was high and combined with strong abilities of the model. By obtaining this excellent result we can have a tool for practical uses with high value like industrial inspections and medical diagnostics. With these results that have been obtained, we can understand that the system is ready to use the results in some situations that is important to have precise analysis of thermal imagery.

References

[1] Cobo, A., Lopez-Higuera, J. M., Conde, O. M., Mirapeix, J., & Miguélez, I. Arc welding process control based on back-face thermography: Application to the manufacturing of nuclear steam generators. *Proceedings of SPIE - The International Society for Optical Engineering*, 2007. Available: https://www.researchgate.net.

[2] Yuan, Y., Hu, Y., Hu, J., Zheng, Y. Review of sensor-based intelligent welding automation. *International Journal of Mechatronics and Manufacturing Systems*, 2014. Available: https://link.springer.com/content/pdf/10.1186/s40712-014-0015-6.pdf.

[3] Kou, S. *Welding Metallurgy* (2nd ed.). Wiley-Interscience, 2003. Available: https://books.google.it/books.

[4] *Introduction to Laser Welding Simulation & NDT*. Sentin AI. Available: https://sentin.ai/en/introduction-to-laser-welding-simulation-ndt/.

[5] J. Liu, C. Zhang, X. Wu, Z. Xu, and W. Li. "A Comprehensive Review on Recent Advances in Welding Robot Systems," *IEEE Transactions on Industrial Informatics*, 2020. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9221167.

[6] Goodfellow, I., Bengio, Y., & Courville, A. *Deep Learning*. MIT Press, 2016. Available: https://scholar.google.it/scholar?q=Goodfellow,+I.,+Bengio,+Y.,+%26+Courville,+A.+(2016).+Deep+Learning.+MIT+Press.&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[7] LeCun, Y., Bengio, Y., & Hinton, G. "Deep learning." *Nature*, 2015. Available: https://www.sciencedirect.com/science/article/abs/pii/S0893608014002135.

[8] Yao, X., Li, G., Lin, Y., & Liu, Y. "Recent progress in modeling and simulation of welding residual stresses." *Journal of Manufacturing Processes*, 2017. Available: https://www.sciencedirect.com/science/article/abs/pii/S1566253517305328.

[9] Kornick, S., Xing, E., & others. *Understanding the Role of Convolutional Neural Networks in Image Processing*. Carnegie Mellon University, 2018. Available: https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf.

[10] Diebold, F. X. *No Hesitations: Advanced Topics in Econometrics*. University of Pennsylvania, 2012. Available: https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf.

[11] O'Shea, K., & Nash, R. *An Introduction to Convolutional Neural Networks*. 2015. Available: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks.

[12] Lipton, Z. C., Berkowitz, J., & Elkan, C. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 2015. Available: https://www.researchgate.net/publication/277603865_A_Critical_Review_of_Recurrent_Neural_Networks_for_Sequence_Learning.

[13] Bhagirath. *Most Famous Supervised Learning Algorithms*. Medium, 2019. Available: https://medium.com/@bhagirath07/most-famous-supervised-learning-algorithms-1aa0f26ffdf0.

[14] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. Available: https://books.google.it/books?hl=en&lr=&id=RC43AgAAQBAJ&oi=fnd&pg=PR7&dq=Murp

hy,+K.+P.+(2012).+Machine+Learning:+A+Probabilistic+Perspective.+MIT+Press.&ots=unhxgzPv58&sig=N2agAVkAD-qEEt4kgC_f8r_KahY#v=onepage&q&f=false.

[15] Arulkumar, A. *Types of Machine Learning*. Medium, 2020. Available: https://medium.com/@arulkumarark1924/types-of-machine-learning-87e39e061414.

[16] Sutton, R. S., & Barto, A. G. *Introduction to Reinforcement Learning* (2nd ed.). MIT Press, 2018. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[17] *What is Reinforcement Learning?* Spiceworks. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-reinforcement-learning/.

[18] LeCun, Y., Bengio, Y., & Hinton, G. *Deep Learning*. *Nature*, 2015. Available: https://scholar.google.it/scholar?q=LeCun,+Y.,+Bengio,+Y.,+%26+Hinton,+G.+(2015).+Deep+learning.+Nature,+521(7553),+436-444.&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[19] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Available: https://scholar.google.it/scholar?q=Bishop,+C.+M.+(2006).+Pattern+Recognition+and+Machine+Learning.+Springer.&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[20] Aggarwal, C. C. *Neural Networks and Deep Learning: A Textbook*. Springer, 2018. Available: https://scholar.google.it/scholar?q=Aggarwal,+C.+C.+(2018).+Neural+Networks+and+Deep+Learning:+A+Textbook.+Springer.&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[21] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015. Available: https://scholar.google.it/scholar?q=Nielsen,+M.+A.+(2015).+Neural+Networks+and+Deep+Learning.+Determination+Press.&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[22] Glorot, X., & Bengio, Y. *Understanding the Difficulty of Training Deep Feedforward Neural Networks*. Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 2010. Available: https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[23] Vergotten. *Loss Functions: Comprehensive Guide to Loss Functions in Various Machine Learning Domains*. Medium, 2021. Available: https://medium.com/@vergotten/loss-functions-comprehensive-guide-to-loss-functions-in-various-machine-learning-domains-1e76f7a9b584.

[24] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. Available: https://scholar.google.it/scholar?q=Murphy,+K.+P.+(2012).+Machine+Learning:+A+Probabilistic+Perspective.+MIT+Press.&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[25] Baldi, P., & Pineda, F. J. *Learning Representations by Forward-Propagating Errors*. 2023. Available: https://www.researchgate.net/publication/373263432_Learning_representations_by_forward-propagating_errors.

[26] *Summary and the Derivations of Gradients for Linear Regression and Logistic Regression*. DeepLearning.AI Community. Available: https://community.deeplearning.ai/t/summary-and-the-derivations-of-gradients-for-linear-regression-and-logistic-regression/292863.

[27] Ng, A. *Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance*. 2004. Available:
https://www.researchgate.net/publication/2952930_Feature_selection_L_1_vs_L_2_regularization_and_rotational_invariance.

[28] Ruder, S. *An Overview of Gradient Descent Optimization Algorithms*. 2016. Available: https://www.ruder.io/optimizing-gradient-descent/.

[29] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. *Gradient-Based Learning Applied to Document Recognition*. 1998. Available:
https://cseweb.ucsd.edu/classes/wi08/cse253/Handouts/lecun-98b.pdf.

[30] Jain, V. *MAE, MSE, RMSE, Coefficient of Determination, Adjusted R-Squared — Which Metric is Better?* Medium, 2020. Available: https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better-cd0326a5697e.

[31] He, K., Zhang, X., Ren, S., & Sun, J. *Deep Residual Learning for Image Recognition*. 2016. Available:
https://www.semanticscholar.org/reader/06b919f865d0a0c3adbc10b3c34cbfc35fb98d43.

[32] James, G., Witten, D., Hastie, T., & Tibshirani, R. *An Introduction to Statistical Learning* (7th Printing). 2017. Available:
https://static1.squarespace.com/static/5ff2adbe3fe4fe33db902812/t/6009dd9fa7bc363aa822d2c7/1611259312432/ISLR+Seventh+Printing.pdf.

[33] Poli, R., Langdon, W. B., & McPhee, N. F. *A Field Guide to Genetic Programming*. 2008. Available: https://informatica.si/index.php/informatica/article/viewFile/148/140.

[34] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. *Learning representations by back-propagating errors*. *Nature*, 1986. Available:
https://link.springer.com/article/10.1007/BF00116251.

[35] Duda, R. O., Hart, P. E., & Stork, D. G. *Pattern Classification*. Wiley, 2001. Available: https://link.springer.com/content/pdf/10.1023/a:1010933404324.pdf.

[36] Altman, N. S. *An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression*. *The American Statistician*, 1992. Available:
https://sites.stat.washington.edu/courses/stat527/s13/readings/Altman_AmStat_1992.pdf.

[37] James, G., Witten, D., Hastie, T., & Tibshirani, R. *An Introduction to Statistical Learning* (7th Printing). 2017. Available:
https://static1.squarespace.com/static/5ff2adbe3fe4fe33db902812/t/6009dd9fa7bc363aa822d2c7/1611259312432/ISLR+Seventh+Printing.pdf.

[38] Glorot, X., Bordes, A., & Bengio, Y. *Deep Sparse Rectifier Neural Networks*. 2011. Available:
https://www.researchgate.net/publication/215616967_Deep_Sparse_Rectifier_Neural_Networks#fullTextFileContent.

[39] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. *Attention is All You Need*. 2017. Available:
https://openreview.net/pdf?id=Hkuq2EkPf.

[40] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. *Gradient-Based Learning Applied to Document Recognition*. 1998. Available:
http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

[41] Singh, S., Jain, S., & Mathur, A. *Applications of Deep Learning in Digital Image Processing: A Review*. *IEEE Xplore*, 2020. Available: https://ieeexplore.ieee.org/document/9077735.

[42] Krizhevsky, A., Sutskever, I., & Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[43] Boureau, Y. L., Ponce, J., & LeCun, Y. *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition*. 2010. Available: https://www.researchgate.net/publication/221080312_Evaluation_of_Pooling_Operations_in_Convolutional_Architectures_for_Object_Recognition#fullTextFileContent.

[44] Krizhevsky, A., Sutskever, I., & Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[45] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. *Recurrent Neural Network Based Language Model*. 2010. Available: https://www.researchgate.net/publication/221489926_Recurrent_neural_network_based_language_model#fullTextFileContent.

[46] Vapnik, V. *The Nature of Statistical Learning Theory*. 1995. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=58337.

[47] Hochreiter, S., & Schmidhuber, J. *Long Short-Term Memory*. 1997. Available: https://www.semanticscholar.org/reader/cea967b59209c6be22829699f05b8b1ac4dc092d.

[48] Cybenko, G. *Approximation by Superpositions of a Sigmoidal Function*. 1989. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=650093.

[49] Hochreiter, S. *The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions*. 1998. Available: https://www.bioinf.jku.at/publications/older/2604.pdf.

[50] Sutskever, I., Vinyals, O., & Le, Q. V. *Sequence to Sequence Learning with Neural Networks*. 2014. Available: https://ar5iv.labs.arxiv.org/html/1402.1128.

[51] Sokolova, M., & Lapalme, G. *A Systematic Analysis of Performance Measures for Classification Tasks*. 2009. Available: http://atour.iro.umontreal.ca/rali/sites/default/files/publis/SokolovaLapalme-JIPM09.pdf.

[52] Powers, D. M. W. *Evaluation: From Precision, Recall, and F-Factor to ROC, Informedness, Markedness, & Correlation*. 2011. Available: https://www.researchgate.net/publication/228529307_Evaluation_From_Precision_Recall_and_F-Factor_to_ROC_Informedness_Markedness_Correlation.

[53] Jolliff, J. K., Kindle, J. C., Shulman, I., Penta, B., Helber, R., & Arnone, R. A. *Summary Diagrams for Coupled Model Performance Evaluation in Three-Dimensional Oceanographic Simulations*. 2014. Available: https://gmd.copernicus.org/articles/7/1247/2014/gmd-7-1247-2014.pdf.

[54] Botchkarev, A. *Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology*. 2019. Available: https://www.ijikm.org/Volume14/IJIKMv14p045-076Botchkarev5064.pdf.

[55] Gunawardena, K. *Best Practice Thermography Application in Built Environment Studies*. 2023. Available: https://d1wqtxts1xzle7.cloudfront.net/105898903/K._Gunawardena_Best_practice_Thermography_application_in_built_environment_studies-libre.pdf.

[56] Colombo, L., & Tortora, M. *IR Thermography for Non-Destructive Monitoring of Moisture in Cultural Heritage*. 2023. Available: https://www.researchgate.net/publication/374996977_IR_Thermography_for_Non-Destructive_Monitoring_of_Moisture_in_Cultural_Heritage#fullTextFileContent.

[57] Wasserman, L. *All of Statistics: A Concise Course in Statistical Inference*. 2004. Available: https://egrcc.github.io/docs/math/all-of-statistics.pdf.

[58] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. *Array Programming with NumPy*. 2020. Available: https://www.researchgate.net/publication/344301569_Array_programming_with_NumPy#fullTextFileContent.

[59] Hastie, T., Tibshirani, R., & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). 2009. Available: https://hastie.su.domains/Papers/ESLII.pdf.

[60] Szeliski, R. *Computer Vision: Algorithms and Applications*. 2010. Available: https://d1wqtxts1xzle7.cloudfront.net/54343495/Algorithms_and_ApplicationsSzeliskiBook_20100805_draft.pdf.

[61] Wang, H., & Schmid, C. *Action Recognition by Dense Trajectories*. 2011. Available: https://www.researchgate.net/publication/51025101_Action_Recognition_by_Dense_Trajectories#fullTextFileContent.

[62] Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. *Convolutional Sequence to Sequence Learning*. 2017. Available: https://www.researchgate.net/publication/316821219_Convolutional_Sequence_to_Sequence_Learning#fullTextFileContent.

[63] Hochreiter, S., & Schmidhuber, J. *Long Short-Term Memory*. 1997. Available: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory#fullTextFileContent.

[64] James, G., Witten, D., Hastie, T., & Tibshirani, R. *An Introduction to Statistical Learning* (1st Printing). 2013. Available: https://www.stat.berkeley.edu/users/rabbee/s154/ISLR_First_Printing.pdf.

[65] Bishop, C. M. *Pattern Recognition and Machine Learning*. 2006. Available: https://github.com/peteflorence/MachineLearning6.867/blob/master/Bishop/Bishop%20-%20Pattern%20Recognition%20and%20Machine%20Learning.pdf.

[66] Kingma, D. P., & Welling, M. *Auto-Encoding Variational Bayes*. 2013. Available: https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[67] Sokolova, M., & Lapalme, G. *A Systematic Analysis of Performance Measures for Classification Tasks*. 2009. Available: https://www.researchgate.net/publication/222674734_A_systematic_analysis_of_performance_measures_for_classification_tasks#fullTextFileContent.

[68] Xu, H., Du, L., Chen, Y., & Xue, Y. *A Deep Convolutional Neural Network for Location Recognition and Geometry Based Information*. 2018. Available: https://www.researchgate.net/publication/322603720_A_Deep_Convolutional_Neural_Network_for_Location_Recognition_and_Geometry_Based_Information#fullTextFileContent.

[69] Bahdanau, D., Cho, K., & Bengio, Y. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. Available: https://aclanthology.org/D14-1179.pdf.

[70] Li, W., Zhao, H., & Wang, J. *A Comparative Analysis of Generative Neural Attention-based Service Chatbot*. 2022. Available: https://www.researchgate.net/publication/363277502_A_Comparative_Analysis_of_Generative_Neural_Attention-based_Service_Chatbot#fullTextFileContent.

[71] *fnv-c: A Python Library for FNV Hashing*. PyPI, 2023. Available: https://pypi.org/project/fnv-c/.

[72] *FLIR Thermal Imaging Cameras: User Brochure*. FLIR Systems, 2019. Available: http://www.flirmedia.com/MMC/THG/Brochures/T820264/T820264_EN.pdf.

[73] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. *ImageNet: A Large-Scale Hierarchical Image Database*. 2009. Available: https://www.researchgate.net/publication/221361415_ImageNet_a_Large-Scale_Hierarchical_Image_Database#fullTextFileContent.

[74] He, K., Zhang, X., Ren, S., & Sun, J. *Deep Residual Learning for Image Recognition*. 2015. Available: https://arxiv.org/pdf/1512.00567.

[75] Zeiler, M. D., & Fergus, R. *Visualizing and Understanding Convolutional Networks*. 2014. Available: https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf.

[76] Ioffe, S., & Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/375c71349b295fbe2dcdca9206f20a06-Paper.pdf.

[77] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. *Show and Tell: A Neural Image Caption Generator*. 2014. Available: https://arxiv.org/pdf/1412.3555.

[78] Hochreiter, S., & Schmidhuber, J. *Long Short-Term Memory*. 1997. Available: https://deeplearning.cs.cmu.edu/F23/document/readings/LSTM.pdf.

[79] Chollet, F. *Deep Learning with Python*. Manning Publications, 2018. Available: https://books.google.it/books?hl=en&lr=&id=mjVKEAAAQBAJ&oi=fnd&pg=PR9&dq=Chollet,+F.+(2018).+Deep+Learning+with+Python.+Manning+Publications.&ots=Ag9XBH-H_b&sig=C7C37e8s2xOIh88SFYFa3HQHYHg#v=onepage&q&f=false.

[80] Chollet, F. *Deep Learning with Python*. Manning Publications, 2018. Available: https://www.mangoud.com/wp-content/uploads/2020/11/Francois-Chollet-Deep-Learning-with-Python-2018-Manning.pdf.

[81] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. 2014. Available: https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[82] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & van Mulbregt, P. *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*. 2020. Available: https://www.researchgate.net/publication/339008987_SciPy_10_fundamental_algorithms_for_scientific_computing_in_Python#fullTextFileContent.

[83] Minh, T. N. *Practical Python and OpenCV* (3rd Edition). 2018. Available: https://minhtn1.github.io/Practical%20Python%20and%20OpenCV,%203rd%20Edition.pdf.

[84] Bradski, G., & Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008. Available: https://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf.

[85] Han, J., Kamber, M., & Pei, J. *Data Mining: Concepts and Techniques* (3rd Edition). Morgan Kaufmann, 2011. Available: https://myweb.sabanciuniv.edu/rdehkharghani/files/2016/02/The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011.pdf.

[86] Kuhn, M., & Johnson, K. *Applied Predictive Modeling*. Springer, 2013. Available: https://vuquangnguyen2016.wordpress.com/wp-content/uploads/2018/03/applied-predictive-modeling-max-kuhn-kjell-johnson_1518.pdf.

[87] Diebold, F. X. *No Hesitations: Advanced Topics in Econometrics*. University of Pennsylvania, 2012. Available: https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf.

[88] Kingma, D. P., & Ba, J. *Adam: A Method for Stochastic Optimization*. 2014. Available: https://arxiv.org/pdf/1412.6980.

[89] Howard, J., & Ruder, S. *Universal Language Model Fine-tuning for Text Classification*. 2018. Available: https://www.researchgate.net/publication/334116365_Universal_Language_Model_Fine-tuning_for_Text_Classification#fullTextFileContent.

[90] *Anaconda Documentation*. Anaconda, Inc., 2023. Available: https://docs.anaconda.com/.

[91] *Visual Studio Code Documentation*. Microsoft, 2023. Available: https://code.visualstudio.com/docs.

[92] Ranogajec-Komor, M., Krpan, K., & Knežević, Ž. *The Application of Infrared Thermography in Cultural Heritage*. 2013. Available: https://www.imeko.org/publications/tc14-2013/IMEKO-TC14-2013-59.pdf.

[93] Wei, W. *Deep Learning: Convolutional Networks*. 2020. Available: https://wenkangwei.github.io/2020/11/03/DL-ConvolutionNetwork/.

[94] Bao, W., Fang, C., Li, M., Yang, F., & Huang, Z. *Automatic Detection of Coronavirus Disease 2019 Using a Deep Learning Algorithm*. *Scientific Reports*, 2021. Available: https://www.nature.com/articles/s41598-021-93656-0.

[95] *ChatGPT: my Personal AI Assistant*. OpenAI, 2023. Available: https://chatgpt.com/c/a16bcde8-3e0d-4ded-9483-5c4fb6923e06.

[96] Căilean, A., & Dimian, M. *Applying Neural Networks for Tire Pressure Monitoring Systems*. 2019. Available: https://www.researchgate.net/publication/336607800_Applying_Neural_Networks_for_Tire_Pressure_Monitoring_Systems/figures?lo=1.