Warsaw University of Technology

Faculty of
Mathematics and Information Sciences

Master's Diploma Thesis

Computer Science and Information Systems

Specialization: Artificial Intelligence

Deep Learning Model for Emotion Recognition from Facial Expression Images

Andra Umoru

Student I.D.: 324334

Supervisor

dr hab. inż. Jerzy Balicki, prof. ucz.

WARSAW 2023

Deep Learning Model for Emotion Recognition from Facial Expression Images

Abstract

We conducted an empirical study by developing and evaluating five deep learning models, which are the proposed Convolutional Neural Network (CNN) model using (TensorFlow and PyTorch frameworks), the SANet model, the VGG-16 model and the ResNet50 model for emotion recognition. These five (5) deep learning models were applied to four publicly available facial expressions image datasets that are used for emotion recognition – the Extended Cohn-Kanade (CK+) dataset, Facial Expression Recognition (FER2013) dataset, the Karolinska Directed Emotional Faces dataset, and the Natural Human Faces Images dataset.

We explored these datasets; data preprocessing was also carried out, and we set the parameters grid that we used for the hyperparameters tuning. This work aims to measure these models using the following metrics: accuracy, F1 score, AUC ROC, the model with the best training and evaluation time, and the model with the best memory usage. At the end of the experiment, the CK+ dataset recorded the best metric scores across the five CNN models: the SANet achieved an accuracy of +/- 98%, F1 score of +/- 97%, and AUC ROC score of +/- 0.98, TensorFlow obtained an accuracy score of +/- 95%, F1 score of +/- 95%, AUC ROC score of +/- 0.98, the PyTorch model attained an accuracy score of +/- 95%, F1 score of +/- 94% and an AUC ROC score of +/- 0.98 whereas the VGG-16 model accuracy is +/- 90%, F1 Score of +/- 84%, and an AUC ROC score of +/- 0.98 while the ResNet model achieved a peak accuracy +/- 42%, F1 score +/- 26% and an AUC ROC score of +/- 0.79. Comparatively, the SANet model was recorded to have the best accuracy and F1 score while having the same AUC ROC score of +/- 0.98 as the three other models. The SANet model had the best training time of +/- 1388.40 seconds and evaluation time (s) of +/- 277.14 seconds; the PyTorch model recorded the best memory usage of +/- 345.85 MB.

Among the five CNN models, the ResNet model performed poorly across all four selected datasets, while the other four models recorded a fair performance on the three remaining datasets. The study concludes by emphasizing the need for more computing resources and hyperparameter tuning for accurate, faster, and more efficient experimentation. This research provides a broad view of emotion recognition using deep learning models and aims to serve as a foundation for future advancements in the field.

Keywords: classification, deep learning**,** emotion recognition**,** facial expression images.

Modele uczenia głębokiego do rozpoznawania emocji na podstawie zdjęć ekspresji twarzy

Streszczenie

W pracy zaprezentowano wyniki badań empirycznych w oparciu o pięć modeli głębokiego uczenia się, w tym modele konwolucyjnej sieci neuronowej (CNN) wykorzystujące frameworki TensorFlow i PyTorch. Ponadto rozważa się model SANet, model VGG-16 oraz model ResNet50 do rozpoznawania emocji. Skonstruowane modele głębokiego uczą się na wybranych publicznie dostępnych zbiorach danych dotyczących zdjęć ekspresji twarzy, które są wykorzystywane do rozpoznawania emocji. Rozważa się cztery zbiory danych: Extended Cohn-Kanade (CK+), FER2013, Karolinska Directed Emotional Faces i Natural Human Faces Images.

Przeprowadzono wstępną obróbkę danych i wyznaczyliśmy wartości parametrów. Modele porównano pod kątem następujących metryk: dokładność, F1, AUC ROC, czas treningu, czas klasyfikacji oraz wielkość wymaganej pamięci RAM. Dla zbioru danych CK+ model SANet osiągnął dokładność na poziomie +/- 98%, F1 na poziomie +/- 97%, a AUC ROC na poziomie +/- 0,98. Implementacja własnej sieci CNN z wykorzystaniem TensorFlow cechowała się dokładnością na poziomie +/- 95%, F1 +/- 95%, AUC ROC +/- 0,98. Natomiast model CNN w PyTorch charakteryzował się dokładnością +/- 95%, F1 +/- 94% i wynik AUC ROC +/- 0,98. Dokładność modelu VGG-16 wynosi +/- 90%, F1 to +/- 84% i AUC ROC to +/- 0,98. Model ResNet osiągnął maksymalną dokładność +/- 42%, F1 +/- 26% i AUC ROC +/- 0,79.

Model SANet został uznany za model o największej dokładności i najlepszym wyniku F1, mając jednocześnie taki sam wynik AUC ROC +/- 0,98 jak trzy inne modele. Model SANet był modelem o najkrótszym czasie uczenia wynoszącym +/- 1388,40 sekundy i czasie klasyfikacji +/- 277,14 sekundy. Model PyTorch cechuje się najbardziej efektywnym wykorzystaniem pamięci wynoszącym +/- 345,85 MB. Spośród analizowanych modeli CNN model ResNet wypadł słabo dla wszystkich wybranych zbiorów danych. Warto podkreślić, że zalecane są większe zasoby obliczeniowe do dostrajania hiperparametrów w celu zapewnienia dokładnych i wydajniejszych eksperymentów.

Slowa kluczowe: klasyfikacja, głębokie uczenie się, rozpoznawanie emocji, obrazy ekspresji twarzy.

TABLE OF CONTENT

Introduction

Deep learning has found applicability in different domains of human activities; one of such domains is human-computer interaction. This domain has been revolutionized by emotion recognition from facial expressions. Understanding human emotion can never be overemphasized as it is not limited to psychology or sociology, but it has found application in healthcare, security, marketing, and automated vehicle design. Emotion recognition using facial expression images has been a fascinating field of study in computer science, psychology, and artificial intelligence. Since the advent of deep learning, the application of neural networks in emotion recognition has attracted significant attention. These methods promise a solid strategy for reliably and accurately automating identifying human emotions [1].

In medical diagnosis and marketing, understanding emotions plays an essential role in applications such as human-computer interaction [2]. Most conventional techniques or methods for identifying emotions from facial expressions have relied on manually created features and simple machine-learning algorithms, such as Support Vector Machines or Decision Trees [3]. The complexity of patterns and high-level elements in facial expressions, which are frequently necessary for precise emotion recognition, cannot be captured by these approaches despite their merits [4].

Recently, Convolutional Neural Networks (CNNs) have shown exceptional performance in image recognition tasks [5]. CNNs architectures are well-fitted and possess the ability to capture both the low-level and high-level features within images. It is because of this that they are known to be highly effective in recognizing delicate emotional cues that could be missed by conventional machine learning algorithms [6]. However, despite this potential, there are also challenges in implementing deep learning models for emotion recognition. One primary issue is the need for large, annotated datasets to train and evaluate these models effectively [7]. Moreover, the complex architecture of these deep neural networks renders them computationally demanding and expensive, and as a result posing a great challenge for real-time applications [8].

This thesis aimed at producing a deep learning application to recognize emotions based on face images from selected data sets. The multidisciplinary nature of this field of study makes emotion recognition a fertile ground for research, especially in Deep Learning, which has shown remarkable success in pattern recognition-related tasks. Our work aimed to contribute to this ever-changing and ever-growing field by developing and evaluating five deep learning models (TensorFlow model, PyTorch, SANet model, VGG-16 model, and ResNet50 model) for emotion recognition from four facial expression image datasets (CK+, FER2013, Karolinska Directed

Facial Images, and the Natural Human Faces Images). The primary objective of this research is to identify the most effective deep learning models in terms for accuracy, F1 score, AUC ROC best training and evaluation time, and the best memory usage for emotion recognition from facial expression images as described above. The scope of the thesis is wide-ranging, including an in-depth analysis of these four datasets, a comprehensive evaluation of the five deep learning architectures stated above, and a robust comparative analysis based on the above-mentioned performance metrics. We aimed to provide a 360-degree view of these Convolutional Neural Networks (CNNs) models and datasets and their efficacies to serve as a guiding resource for future research in this area.

It is also noteworthy to state that at the end of this experiment, this thesis work seeks to provide answers these important questions that are central to the field of emotion recognition; these questions are: which among the five (5) deep learning models are the most effective deep learning architectures for emotion recognition from facial expression images? Which of these four selected datasets impact the performance of these deep learning models? What metrics are ideal for evaluating the performance of emotion recognition systems? How do these deep learning models compare in terms of computational efficiency, scalability, and accuracy? We intend to employ rigorous and systematic methodology in this thesis work.

In Chapter 1, we provided an exhaustive exploration of the characteristics of the four datasets which include the Extended Cohn-Kanade Image Collection, FER2013 Image Collection, the Karolinska Directed Emotional Faces, and the Natural Human Face Image datasets. Chapter 2 provides an in-depth analysis into the architectures of these three state-of-the-art models (the SANet model, VGG model, and ResNet model) and the proposed model (using TensorFlow and PyTorch framework) by discussing the evolution of these models, their underlying principles, and their impact on the field of emotion recognition. Whereas Chapter 3 will highlight and discuss the metrics for the model's classification. Summarily, the chapter covers the pros and cons of various metrics and provides guidelines for their appropriate usage. In Chapter 4, we explained the implementation details by providing technical details about the five models used in this experiment, data preprocessing, hyperparameter tuning, and the results obtained using these five models on the four selected datasets. Furthermore, Chapter 5 handles the numerical experiments; this chapter provided details about the model with the best accuracy, best F1 score, the best AUC ROC score, the model with the best training and evaluation time, and finally the model with the best memory usage.

In conclusion, we will provide a summary of this thesis work as it will culminate in a comprehensive conclusion that summarizes the key findings and their implications. We will also point to areas to be considered for future research directions, including but not limited to the exploration of more advanced deep learning models and datasets, more hyperparameter tuning activities, the incorporation of multi-modal data, and the development of real-world applications.

CHAPTER 1

CHARACTERISTICS OF THE DATASETS

To successfully develop and train a deep learning model for emotion recognition, there is an unwavering need for image dataset collections. For this experimental work, we selected four different datasets for training and testing the model. The selected datasets are as follows: the Extended Cohn-Kanade (CK+), Facial Expression Recognition 2013 (FER2013) Image Collection, Karolinska Directed Emotional Faces (KDEF), and Natural Human Face Images for Emotion Recognition image collection. We will now briefly explore the characteristics of the image collections selected for this experiment.

1.1 The Extended Cohn-Kanade Image Collection (CK+)

The Cohn-Kanade (CK) image collection was made public in the year 2000 to advance research in automated emotion recognition from facial images [9]. Since this image collection was released, it has proven to be one of the most potent datasets used in testing and training emotion recognition models. However, according to [10], there are several restrictions with the CK dataset when used for scientific research purposes. These limitations include unreliable labels for emotions because they reflect intent rather than actual events. There arise some difficulties with quantifying label validity in such a way that it affects tuning algorithm performance as it opposes the conventional standards, the absence of a consistent metric for assessing the efficacy of novel algorithms for both AU and emotion detection. It is noteworthy that results obtained from some state-of-the-art algorithms serve as a yardstick for comparing the performance of new algorithms, and for managing widely used databases, there are no set rules. In paper [10], it is also opined that there is a known challenge to the quantitative meta-analysis of the database due to this lack of protocols.

As a result, meta-analysis are challenging due to the usage of random portions of the original database and the fact that the CK database has been used for both AU and emotion recognition [10]. Even though the CK image data set collection is one of the most widely used datasets for building and measuring the performance of facial expression algorithms for emotion recognition analysis, the above constraints render benchmarking the performance of different models almost impossible to achieve [10]. The limitations of the CK data set collection proved the need for further work on the data set to improve it. Because of these constraints, further additional work was conducted on the image collection. To handle these known limitations or restrictions or better put constraints, in the year 2010, the researchers introduced a novel version of the CK image

collection. This new image data set collection is called The Extended Cohn-Kanade Image Collection (CK+).

The (CK+) dataset contains 210 adults' facial expressions captured using two hardware-synchronized Panasonic AG-7500 cameras. The participants were between the age range of 18 – 50 years, in which 69% were female, 81% were European American, 13% are African American, and 6% were from other ethnic groups. Participants were directed by an experimenter to perform a series of 23 facial displays; these included single action units and combinations of action units. Each show began and ended in a neutral face with any exceptions specified. Image sequences for frontal views and 30-degree views were converted to digital images with pixel arrays of 640x490 or 640x480 pixels and 8-bit or 24-bit color values [9].

CK+ data set contains 593 video sequences of 123 different subjects where 327 video sequence is labeled with one of seven (7) emotion labels: anger, contempt, disgust, fear, happy, neutral, sadness, and surprise [10]. However, some authors performed their experiments using only six (6) out of the eight (8) emotions by ignoring the neutral and contempt facial emotional expressions [11], this same opinion was also opined by [12]. Another author [13], trained two separate models, one model for the eight features models and the other for the six (6) features model with the aim of obtaining comparative results. The eight (8) class model received a state-of-the-art result accuracy whereas the six (6) class model achieved near state-of-the-art accuracy. The author further stated that other parameters such as regularization, data augmentation, and dropout contributed immensely to increasing the performance of the model to 16.9% [13]. It is worth noting that during the CK image collection was created, 84 subjects smiled at the experimenter on one or more occasions. These smiles were not provided upon request however, since they were comprising the initial pool, they were included in the CK+ image collection. Figure 1 below shows some sample images from the CK+ data sets [14]:

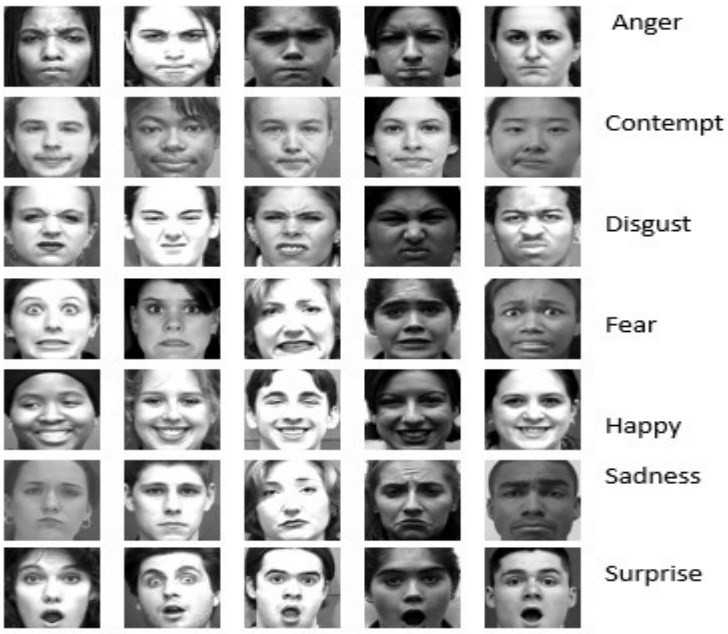

Figure 1: Samples images from CK+ Data sets [14]

For these facial expressions images to be added to the image collection, there are specific criteria were considered as presented by [10]. These criteria are as follows: expression that is relatively neutral at the start of the sequence, there is no sign of a facial expression direction request, before the smile apex, there is no facial occlusion, and the absence of image artifacts. An explorative work performed on the CK+ image dataset can be seen from Figure 2.

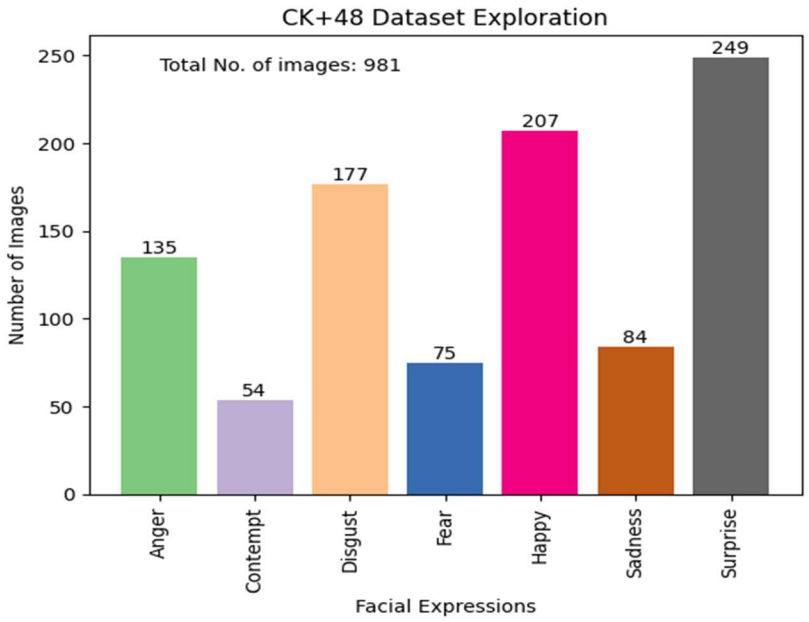

Figure 2: CK+ dataset exploration

1.2 FER2013 Image Collection

The FER2013 image dataset was produced by [15] and made it available for public consumption via a Kaggle challenge competition. The dataset contains 35,887 48x48 pixel grayscale photos of faces that are individually labelled with one of seven (7) emotion categories: anger, contempt, fear, happiness, sorrow, surprise, and neutral [15]. Images within the FER2013 dataset consist of both posed and unposed face images [16]. The dataset is intended for the job of facial emotion recognition. The photographs in this collection reflect a wide range of demographic characteristics, including age, ethnicity, and lighting conditions. The images were downloaded from the internet and manually annotated [15]. Figure 3 shows some samples that were selected from the eight emotions from the FER2013 image collection downloaded from [17].

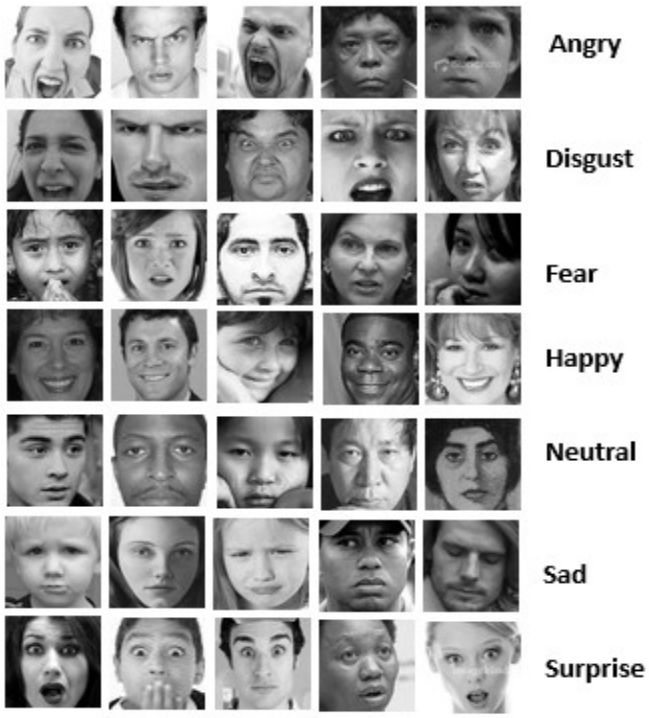

Figure 3: Samples images from FER2013 Data sets [16]

Further work was performed on FER2013 image dataset by [18], they used crowdsourcing to retag the FER2013 image collection and called it FER+ image collection [18]. Taggers were asked to choose one of eight emotions for each image, including neutrality, happiness, surprise, sorrow, anger, disgust, fear, and contempt. Additionally, volunteers had to choose just one emotion for each image. The gold standard approach was used to ensure label quality following the labelling operation [18]. They further stated that in the process of tagging, eight separate attempts were made. Three taggers were used in the initial effort, and the number of taggers was then increased

by one with each subsequent attempt. In the most recent effort, ten separate taggers worked together to minimize noise in the dataset and achieve the target data quality [18]. They conducted a second experiment in which they measured the impact of the number of participants on the labelling process as the number of volunteers increased. It turned out that accuracy is just 46% when there are 3 volunteers for each image. When there were five volunteers, the accuracy rose to about 67%, and when there were seven volunteers, it reached more than 80% as stated by [18]. Thus, one may draw the conclusion that the ultimate quality of the labels is significantly influenced by the number of volunteers for labelling each image. It's important to note that the distribution of the dataset among the classes is unbalanced, with happiness being the primary category. Such imbalances might make it challenging to train a model since the learning algorithm may start to favor the majority class [19].

However, to effectively train the deep learning models that can precisely identify each of the seven emotions, it is crucial to manage the data imbalance. For managing such datasets, [20] suggest a multi-tiered strategy. To begin with, they use the Synthetic Minority Over-sampling Technique (SMOTE) and Random UnderSampling (RUS) techniques to oversample the minority class and undersample the majority class to rebalance the dataset. They show how this strategy lessens overfitting to the dominant class. The second method entails fine-tuning the deep learning model architecture with an emphasis on minority class prediction. They introduce a brand-new loss function called "Rebalanced Cross-Entropy Loss" that gives the minority class more weight during training, making the model pay closer attention to it.

They show that integrating these two approaches considerably boosts the model's functionality, particularly in terms of Precision, Recall, and F1 score for the minority class. Further effort to solve the problem of unbalanced dataset was proposed by [21], they examined cost-sensitive learning, a method for classifying unbalanced data that involves imputing various misclassification costs to multiple classes. They provide a brand-new cost-sensitive loss function that can be incorporated into the architecture of any deep learning model.

The "Adaptive Cost-Sensitive Loss" (ACSL) loss function dynamically modifies the cost of misclassification in accordance with the model's performance at each training session. They [21] provided examples to show how this dynamic modification causes the model to converge more quickly and effectively. They further assert that their suggested approach outperforms conventional resampling techniques since it does not run the danger of overgeneralizing from the minority class (a typical problem with oversampling) or losing important information from the majority class (a problem with conventional resampling techniques).

The marriage between FER2013 images and Convolutional Neural Networks (CNNs) is solid; CNNs are usually used to extract features from FER2013 images. This is because CNNs are known for their remarkable performance in image analysis applications. According to [22], CNNs are deep learning models created to automatically and adaptively learn spatial hierarchies of features from images. Also, [23] and [24] made significant progress in tasks including picture categorization, object identification, and facial recognition. With further focus on FER2013, CNN can be trained to recognize face traits or features connected to various emotions. The CNNs architecture include layers such as the convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for final classification may all be included in the network architecture [25].

Despite CNNs' impressive performance, it is not simple to train a deep learning model to recognize emotions from facial images such as FER2013 collection. The elusiveness of facial expressions, the variation in how people express themselves, and problems with lighting, occlusion, or image quality can all present difficulties in achieving effective results [26]. Several tactics, including data augmentation, transfer learning, and architecture optimisation, can be used with the view to solve these difficulties that often arise during these experiments. Explorative work performed on the FER2013 image dataset can be seen from Figure 4.

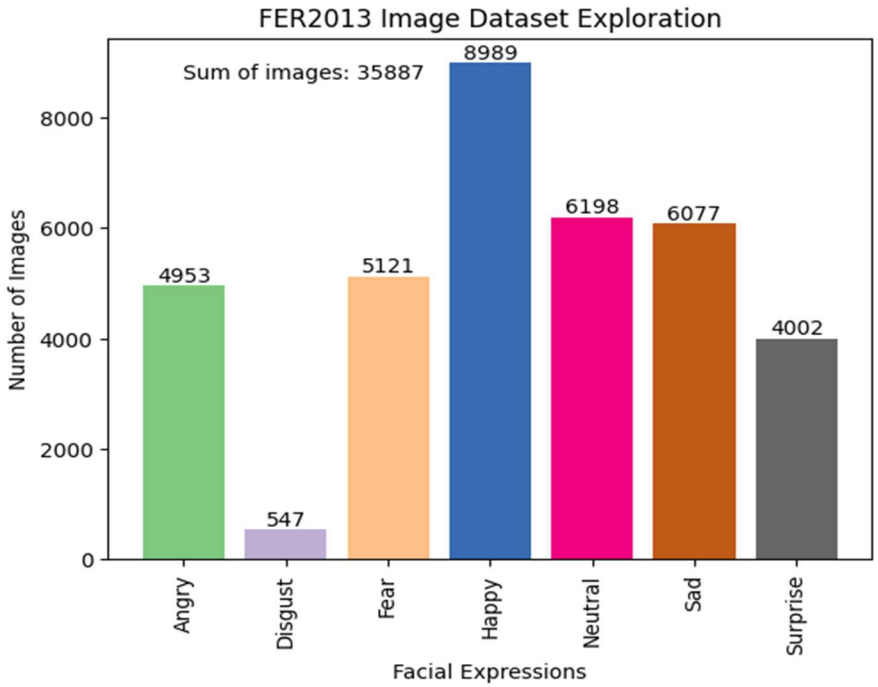

Figure 4: FER2013 dataset exploration

1.3 Karolinska Directed Emotional Faces Image Dataset

The Karolinska Image Dataset (KDEF) is a public-domain facial expression images database developed by the Karolinska Institute in Stockholm, Sweden [27]. The KDEF is known to be one of the widely used human facial expressions database in the domain of emotion recognition [28]. This freely available dataset contains high-resolution pictures of people's faces expressing a variety of emotions, making it an essential resource for psychological and neurological study.

Images of 70 individuals, comprising 35 men and 35 women between the ages of 20 and 30, are included in the KDEF. Seven (7) emotional states are displayed by the participants: neutral, happy, sad, angry, fear, disgusted, and surprised. Five (5) alternative perspectives—full frontal, left profile, right profile, half-left profile, and half-right profile—were used to record each emotional state. The end result is a set of 4900 unique images (70 subjects multiplied by 7 emotions and 5 orientations), which are widely used to train machine learning models [27].

The standardized nature of the images is one outstanding aspect of the KDEF. Each photograph was taken under carefully monitored lighting, camera distance, and technical specifications. In addition, the subjects were told not to use any face accessories or makeup that would mask their expressions [27]. The KDEF image dataset is the most balanced dataset among the selected datasets for this work. Figure 5 shows some samples chosen from the seven emotional classes of the KDEF image collection [29].

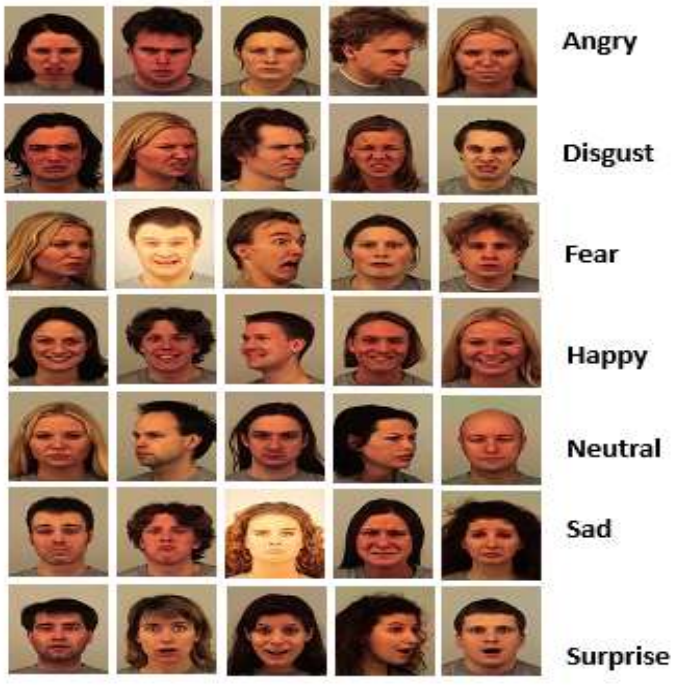

Figure 5: Samples images from KDEF Data sets [29]

Numerous research [30], [31], and [32] have investigated similar facial expression datasets like CK+, FER2013 and other facial expression image datasets. A study by [33] achieved 80.95% average recognition rate using the KDEF image collection. It is also worthy to note that another study was conducted by [32] achieved 82%, with the disgust and the happy classes proved to be the most easily recognizable emotional class while the contrast was observed on the sad emotion class. In 2019, [30] investigated the use of deep learning model (CNN) to identify and categorize emotional states from the KDEF image dataset and achieved 96.43% accuracy. Convolutional neural networks (CNNs) provide remarkable results in identifying emotional states from the KDEF dataset, according to [32] observations.

These models can recognize complex facial expressions and patterns that are connected to various emotional states. The accuracy of the CNN models used by different researchers through the usage of the KDEF image dataset averagely hinged between 80% - 90%, which is a significant improvement over earlier machine learning-based method. This shows that the KDEF can also be utilized in conducting emotion recognition-based studies to classify emotional states from the images.

The effectiveness of deep learning models with the KDEF data, according to these authors, rests in these models' capacity to manage highly dimensional data as well as automatically recognize and extract characteristics from unprocessed pixel values. As a result, there is no longer a need for manually created features, which are time-consuming and frequently miss the true intricacy of the data. The KDEF is a crucial tool for research on emotion recognition, and deep learning models have shown encouraging results when using it. Although the current results are impressive, there is always room for improvement, such as better model topologies or the incorporation of additional emotional indicators like body posture or vocal tone. Combining KDEF’s broad spectrum of emotional responses with cutting-edge computational models provides a viable path for the future of emotion recognition. Also, an explorative work performed on the KDEF image dataset can be seen from Figure 4 given below:

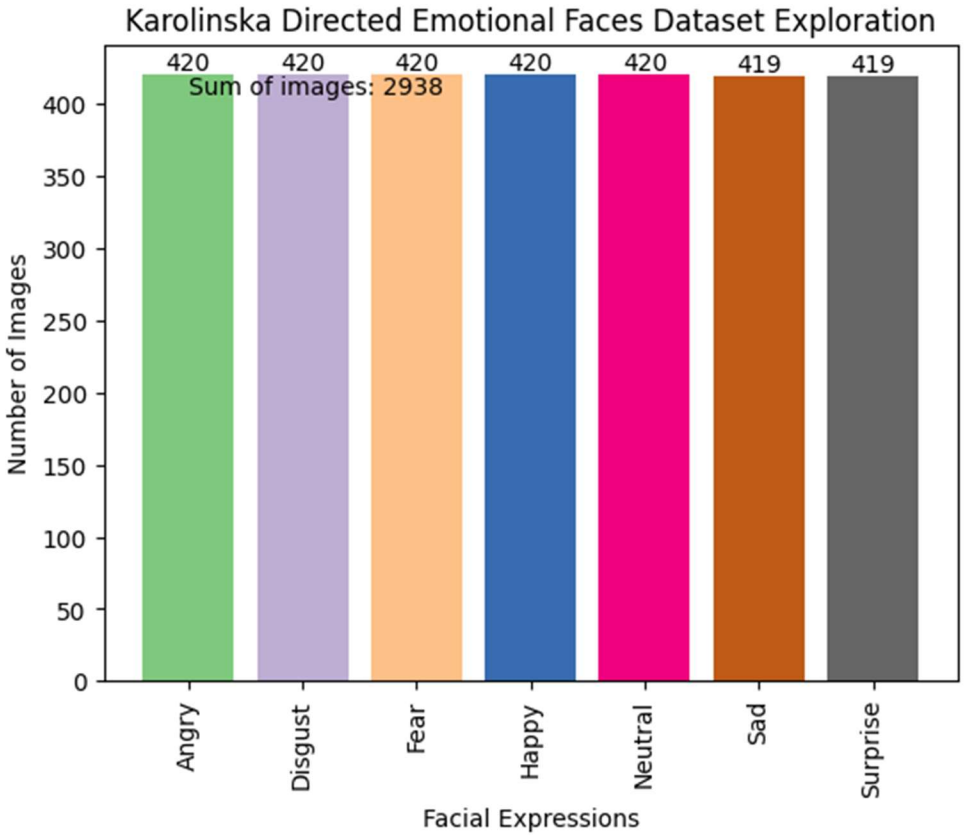

Figure 6: KDEF dataset exploration

1.4 Natural Human Face Images dataset

The Natural Human Face Images Dataset (NHFI) is another image data collection that was created in 2020 by Sudarshan Vaidya and make available to the public on Kaggle [34]. This sole aim of developing this image dataset is to make more annotated data available for the purpose of building, training, and testing deep learning or machine learning models for the purpose of performing emotion recognition [34]. Another motivation for developing this image dataset was the author believed that there would be a definite score for improvement in the manual annotation that was employed in creating the dataset [34]. Just as other standard and popular facial recognition image datasets like FER2013, CK+, and KDEF, Natural Human Face Images Dataset is also being deployed frequently in facial expression detection.

This dataset attempts to add more information to real human faces that have been carefully and manually annotated from the internet [34]. Furthermore, these internet images are downloaded for free from sites like Google, Unsplash, and Flickr, among others [34]. Figure 7 shows some samples selected from the eight emotional classes of the Natural Human Face image collection downloaded from [35].

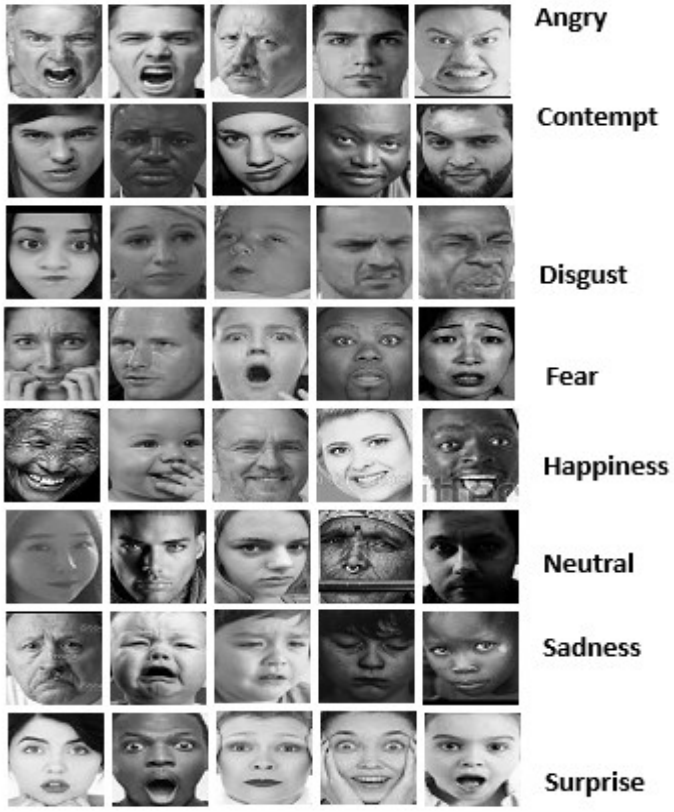

Figure 7: Samples images from Natural Human Face Datasets [35]

The Natural Human Face Images dataset contains more than 5,500 images that are classified into eight (8) emotion categories, including surprise, contempt, fear, disgust, neutral, sadness, surprise, and happiness. These images are grayscale human faces (or sketches). The images are in Portable Network Graphic (PNG) image file format with dimension of 224 x 224 grayscale pixels [34]. Recent studies shed light on the promising outcomes of deep learning models applied to such datasets for emotion recognition tasks. For instance, the Convolutional Neural Networks (CNNs) were used in a study by [36] to perform emotion recognition tasks on the AffectNet dataset. They claimed to have reached an astounding accuracy of 71.7% using sophisticated CNN architecture and data augmentation methods. The importance of a high-quality, diversified dataset for enhancing the model's performance was also highlighted by their research [36]. A bar chart below tagged Figure 8 presents an explorative study performed on the Natural Human Face image dataset showing the eight emotional classes.

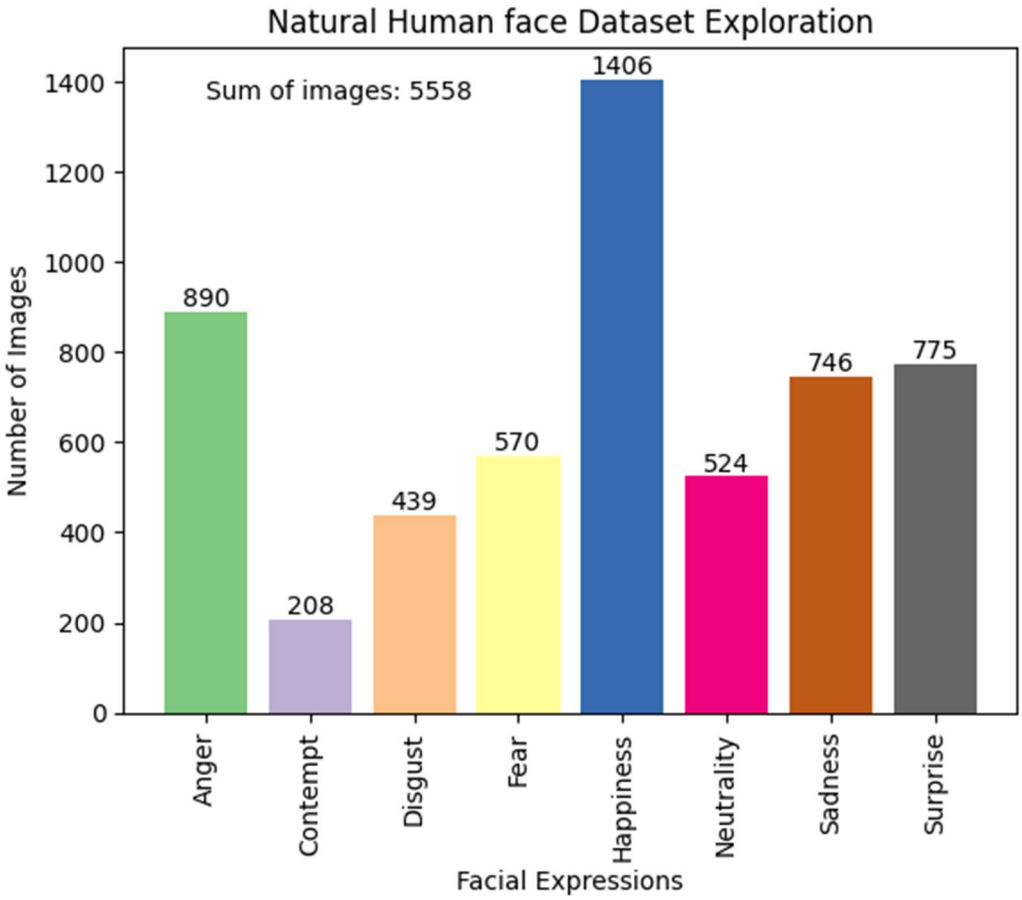

Figure 8: Natural Human faces image dataset exploration

1.5 Remarks and Conclusions

There is a common observation regarding the datasets; all the datasets except the KDEF image dataset are unbalanced in terms of the number of images per emotion category or class. As such, there will be a need for a modality that will bring about balanced test results after training our deep learning model. This is because an unbalanced dataset will significantly affect the performance of the model. Even though some noticeable patterns can be seen from the exploration of these datasets, more profound features will be discovered later when we start the building, training, and testing our deep learning model during the experiment stage.

CHAPTER 2

DEEP LEARNING MODELS

2.1 Residual Network (ResNet) Model

Residual Networks (ResNets) were introduced by [25], this signalled a fundamental change in the methodology used to create deep learning architectures. Deep learning has become a top machine learning methodology because of its ability to extract complicated features from massive datasets. The so-called vanishing or exploding gradients problem has made it impossible to successfully train very deep neural networks, which has been one of the basic problems with this approach. ResNet offered a creative solution in this situation. The term "vanishing or exploding gradients problem" describes a situation in which the gradient of the loss function either becomes extremely small or extremely huge, making it challenging to optimize the weights of the network. As the network's depth increases, this issue becomes more obvious. [25]'s brilliant solution was the creation of a "residual block," which has since served as the foundation of the ResNet architecture.

Each layer in conventional deep networks picks up a fresh representation of the input data. As opposed to this, each layer in a ResNet learns a residual function using the input to the layer. More specifically, if the desired underlying mapping is *H(x)*, the network is instructed to learn the residual function $F(x) = H(x) - x$ rather than attempting to learn this mapping directly. Through a shortcut or skip connection that performs identity mapping, this difference is then transferred back to the original input, allowing the input of a layer to be transmitted to its output [25]. With "shortcut connections," feedforward neural networks can implement the formula $F(x) + x$ as can be seen in Figure 9.

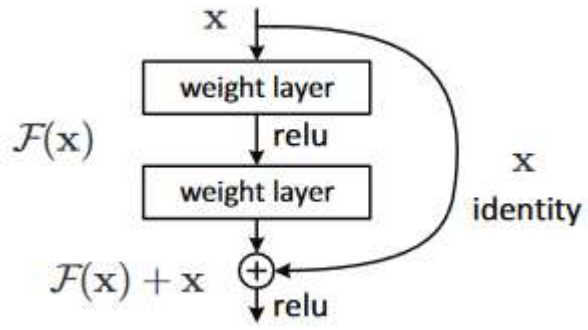

Figure 9: Residual Learning: a building block [37]

The author [25] further argued that the degradation challenge raises the possibility that the solvers may struggle to approximate identity mappings using several nonlinear layers. If identity mappings are the best option, the solvers can push the weights of the numerous nonlinear layers towards zero with the residual learning reformulation to get close to identity mappings.

This residual learning framework's ability to effectively train exceedingly deep networks is one of its primary advantages [26]. In bold terms, [25] showed that networks with 152 layers, much more than earlier architectures, can be trained effectively and efficiently without overfitting. The ResNet model fared better than earlier models on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015, taking first place with its extraordinary depth and creating a new benchmark in the area. ResNet has proven to be efficient in a range of activities and areas, according to subsequent studies. For instance, [26] used ResNet for object detection tasks and outperformed the competition on the Microsoft COCO dataset, which contains over 200k labelled pictures for object detection. ResNet has been used in image analysis for medical purposes and computer vision jobs. An algorithm for detecting pneumonia from chest X-rays was developed by [27] using a modified ResNet model. Their system, CheXNet, outperformed practicing radiologists regarding average precision, demonstrating the potential of ResNet and deep learning more generally in the healthcare industry.

ResNet has achieved great success, yet it has flaws and room for development. ResNets act more like ensembles of relatively shallow networks despite their depth, according to [28]. This result sparked debate about whether ResNets' superior performance is attributable to their depth or this ensemble-like behaviour. Additionally, Wide Residual Networks (WRN), a variation of the ResNet design, was introduced by [29]. Compared to normal ResNets, WRNs are less deep yet have more neurons per layer. The fact that WRNs outperformed ResNets suggests that width might be just as important as depth in achieving good performance.

To sum up, ResNet has been a crucial advancement in deep learning. It introduced a cutting-edge method to overcome deep network constraints, enabling researchers to construct deeper models without problems. Despite some criticism, it has proven adaptable and effective in various applications, and it continues to motivate new ideas and strategies.

2.1.1 Residual Learning

The residual learning framework as proposed by [25] is given as follows: take a function ***H(x)*** as an underneath mapping that can be fitted by a few layers stacked together (it does not have to be the complete net), with x indicating the inputs to the initial of these layers. Suppose the input

and output have the exact dimensions. In that case, it is equivalent to hypothesizing that many nonlinear layers may asymptotically approach the residual functions, i.e., ***H(x) x*** if one believes they can asymptotically approximate complicated functions2. Therefore, we explicitly allow these layers to approach a residual function ***F(x):=H(x) x*** rather than assuming that they will approximate ***H(x).*** Thus, the initial function becomes ***F(x) + x.*** The ease of learning may vary between the two forms, even though both should be able to approximate the target functions asymptotically. The author [25] argued that the impetus behind the reformulation is the illogical phenomenon around the degradation problem. Figure 9 presents a building block to a few layered layers that implement residual learning [25]. They further proposed an identity mapping by shortcuts where it is given by:

$$y = F(x, \{Wi\}) + x \tag{1}$$

As formulated by [37], from the above formula, ***x*** and ***y*** represents the input and output vectors of the layers in consideration. The function ***F(x, {Wi})*** stands for the residual mapping that is to be learned.

2.1.2 The ResNet Architecture

The ResNet architecture is a deep learning architecture extensively employed in many applications, particularly image classification tasks. The introduction of "shortcut connections" or "skip connections," which enable the gradient to be directly backpropagated to prior layers, is the central concept behind ResNet architecture [30]. The ResNet architecture comes with different layers based on the need and configuration of a user.

According to [25] and [31], the layers may be ResNet – 18, ResNet – 34, ResNet – 50, ResNet – 101 or ResNet – 152. It was opined by [31] that the vanilla ResNet-50 architecture, a common and popular variation of the ResNet containing 50 layers, was optimized for maximum performance. It is interesting to note that a ResNet containing very deep pile modules can act as a universal approximator and, in a way, demonstrate the strength of the ResNet architecture as presented by [32]. ResNet architectures have been applied, among other things, to identifying colorectal cancer in medical imaging. Compared to other deep learning architectures, ResNets has produced better outcomes because of its residual mapping and shortcut connections [33]. Figure 10 shows a ResNet architecture that has 34 layers.

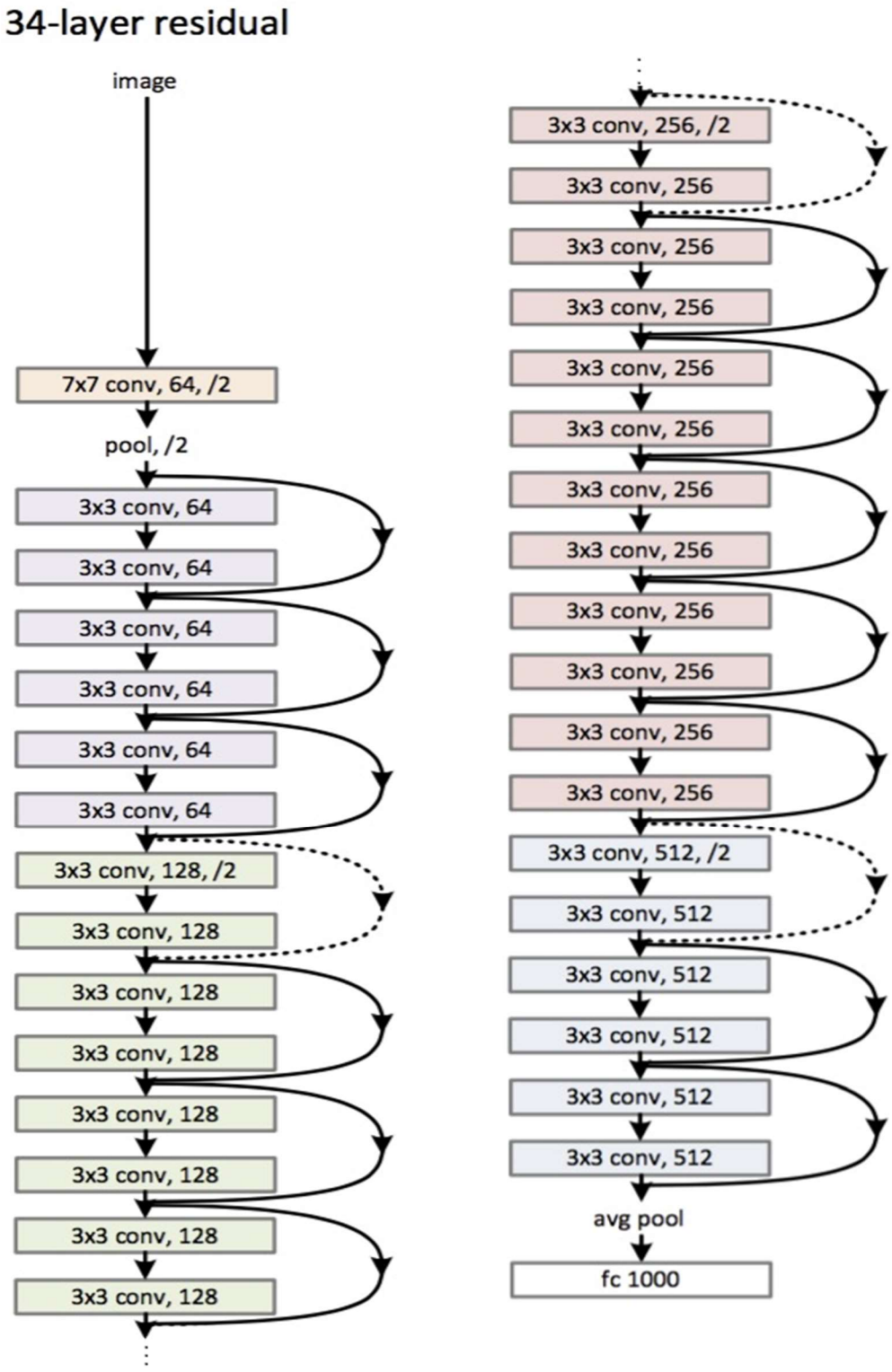

Figure 10: ResNet Architecture with 34 layers (ResNet-34) [38]

2.2 The Visual Geometry Group (VGG) Model

The Visual Geometry Group (VGG) was developed by Karen Simonyan and Andrew Zisserman from the University of Oxford in 2014. It was introduced as a convolutional neural network architecture and, by extension, a deep learning model. The VGG model has two notable variants: the VGG-16 and VGG-19. These two variants have received a great deal of attention in recent times in experimenting with some deep-learning tasks.

The Visual Geometry Group created these VGG models at the University of Oxford and have been used for a variety of tasks, such as the classification of images [35], [36], the diagnosis of diseases [37], and even the analysis of archaeological sites [38]. A comparison analysis of Ground Penetrating Radar (GPR) data gathered from archaeological sites was done in archaeology. The effectiveness of three well-known deep learning architectures—AlexNet, VGG-16, and VGG-19—was assessed in this work [38]. The study aimed to investigate the possibilities of these deep learning models in the processing of GPR data, a non-destructive technique employed in archaeology to view the subsurface. The findings of this research could offer insightful information on how these models might be applied in archaeology, potentially enabling more effective and precise analysis of archaeological sites [38].

2.2.1 VGG Architectures (VGG – 16 and VGG – 19)

The VGG architectures present innovative object identification models within the domain of deep learning—the VGGNet, created as a deep neural network. More specifically, the VGGNets are built upon the fundamental components of convolutional neural networks (CNN), as shown in Figure 12 below. Outperforms benchmarks on a variety of tasks and datasets outside of ImageNet. It remains one of the most often used image recognition architectures today [39]. VGG 16 has a 16-layer architecture with two convolution layers, a layer for pooling, and a fully connected layer at the very end. The concept of a much deeper network with much smaller filters is known as a VGG network.

Compared to AlexNet's eight layers, VGGNet has more layers [40]. Notably, these VGG models consistently use 3 x 3 convolutional filters, which is considerably the minimum convolutional filter size capable of examining some of the nearby pixels. By going through the network, the VGG models maintained this basic structure of 3 x 3 convolutions with periodical pooling [40]. Using the ImageNet dataset, the VGG16 model achieves top-5 test accuracy of about 92.7%. It was among the most well-liked models submitted at ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2014) [39]. It significantly outperforms AlexNet by substituting

a number of 3x3 kernel-sized filters for the huge kernel-sized filters. Nvidia Titan Black GPUs were used to train the VGG16 model over several weeks. The VGGNet-16 has 16 layers and can classify photos into 1000 different object categories, including keyboard, animals, pencil, mouse, and other objects. The model also accepts images with a resolution of 224 by 224 [40], [38], [39].

Just as the VGG-16 has a total of 16 convolutional and fully linked layers in the VGG architecture, the VGG-19 contains 19 convolutional and fully linked layers in the VGG architecture in this instance, thus it is essentially the same architecture with a few extra three (3) convolutional layers added [39], [40], [41]. Figure 11 shows the VGG – 16 architecture:

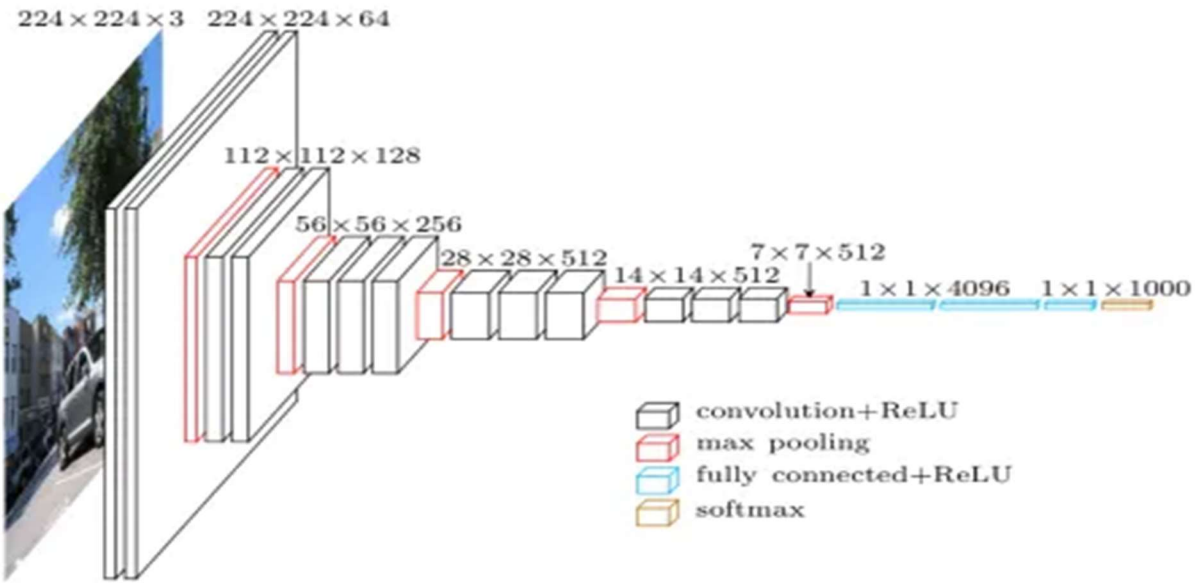

Figure 11: VGG – 16 architecture [42]

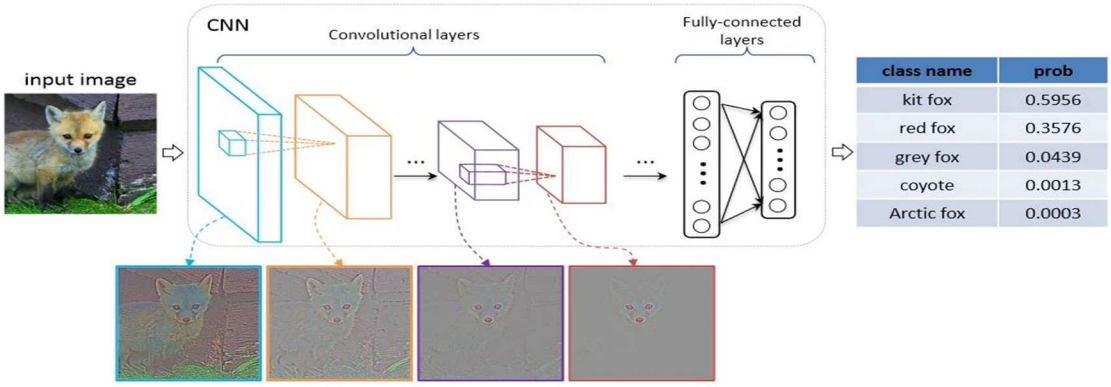

Figure 12: VGGNets CNN Architecture [43]

Furthermore, Figure 13 shows the layers details of the VGG – 16 and the VGG – 19, respectively.

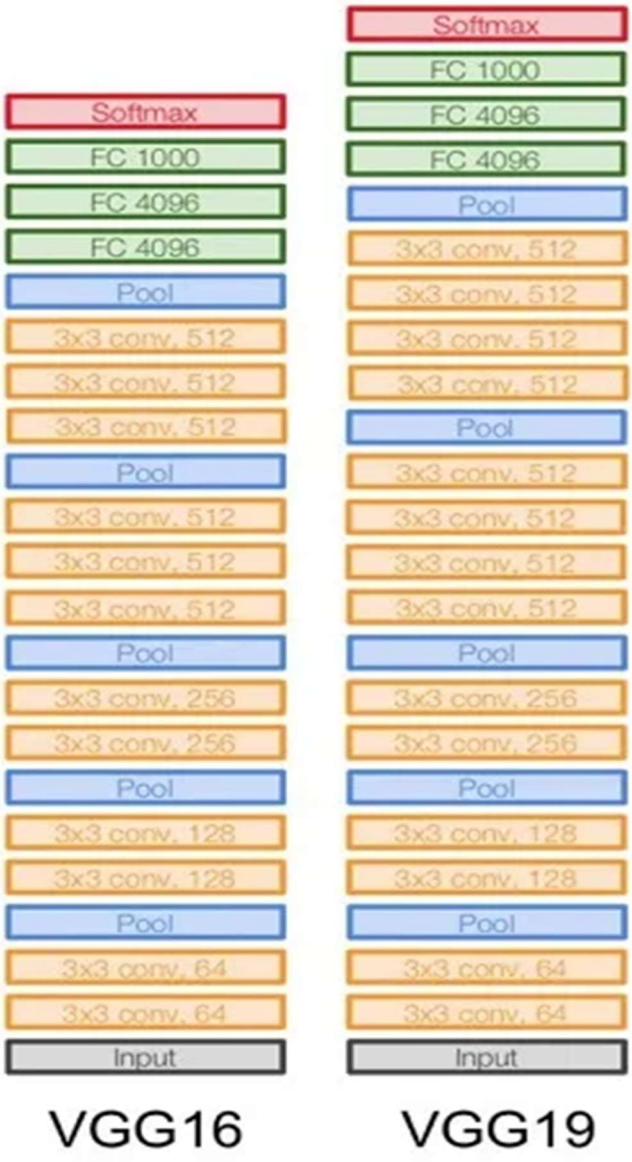

Figure 13: Layers details of VGG-16 and VGG-19 [42]

2.2.2 Some results achieved with VGGNets

The VGGNets have been used in different tasks. Some of these tasks include plant disease identification using the VGG–16 which achieved a 90% accuracy [35], archaeology [38], classification of medical images such as chest X-ray images for COVID-19 diagnosis [43], VGG-SCNet for brain tumors detection on MRI images using transfer learning-based Stacked Classifier (SC) model [37]. The VGG – SCNet through the transfer learning-based Stacked Classifier leveraged on the pre-trained VGG – 16 model.

Other areas in which the VGGNets were also included emotion recognition and sentiment analysis [44]. In conclusion, the VGG model, especially the VGG-16 and VGG-19 variations, has established itself as a flexible and potent tool in deep learning. Its uses include medical imaging, illness detection, archaeological site research, and image classification. Techniques like fine-tuning and transfer learning, as well as incorporating attention mechanisms and stacked classifiers, can further improve the model's performance. To be effective, a model must be carefully chosen and optimised based on the task and dataset, as with every model. This highlights the significance of rigorous model selection and optimisation in deep learning research.

2.3 The SANet Model

The Self-Attention Network (SANet) as a deep learning model was developed in 2017 by a team of researchers [45]. Under the direction of Anfeng He and Huchuan Lu, researchers from China's Dalian University of Technology created the SANet model. Xiangyuan Lan from the Northwestern Polytechnical University in China and Pong C. Yuen from the Hong Kong Baptist University were team members. The SANet model was developed using their combined deep learning and computer vision skills [45]. This model was published in their paper titled "SANet: Structure-Aware Network for Visual Tracking" [45]. The model was developed to deal with the complex problem of visual tracking, a challenge commonly associated with computer vision. Visual tracking is observing an object's movement through a series of still or moving frames. Due to variables including shifting object appearance, occlusions, and camera movements, as a result, this often renders the task to be challenging [45].

2.3.1 SANet Architecture

According to [45], the initial research that brought about the SANet model, as the network's architecture is shown in Figure 14 below, which includes three convolutional layers (each with ReLU and pooling layers), two fully connected layers, and one fully connected classification layer. It takes a 107 x 107 RGB input. A recurrent layer, which simulates the structure of the objects in this level, follows each pooling layer. Additionally, they employ a skip concatenation approach to

combine the features from the pooling and recurrent layers to provide more information to the convolutional layer.

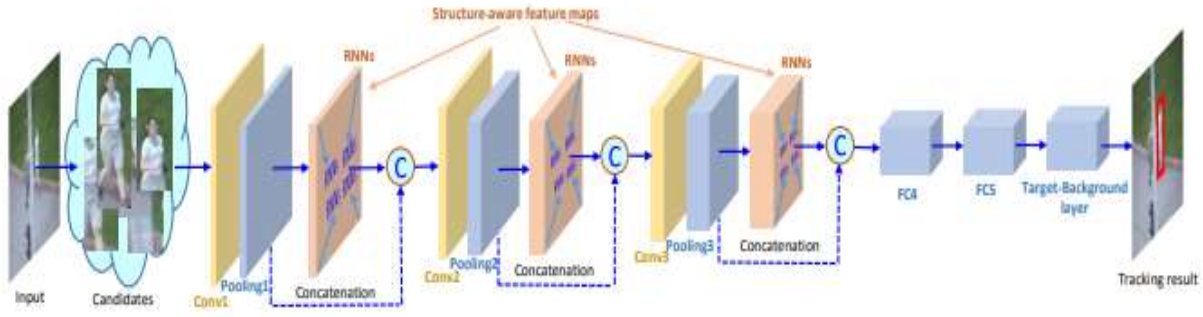

Figure 14: SANet Architecture for visual tracking [44]

It is worth noting that some works were performed using the SANet model but with some distinctive architectural framework. For instance, [46] in their work titled "A deep learning – based method for tooth segmentation on panoramic dental X-ray images." Their architecture was influenced by self-attention modules [47] and vanilla U-Net [48]. These modules included encoder, decoder components, position attention modules, and channel attention modules. In order to adaptively integrate high-level local features with their global dependencies, [46] inserted the self-attention modules after the final encoder layer, which brought about a certain degree of accuracy in predicting the segmentation mask [46]. The SANet architecture described above by [46] is given in Figure 15:

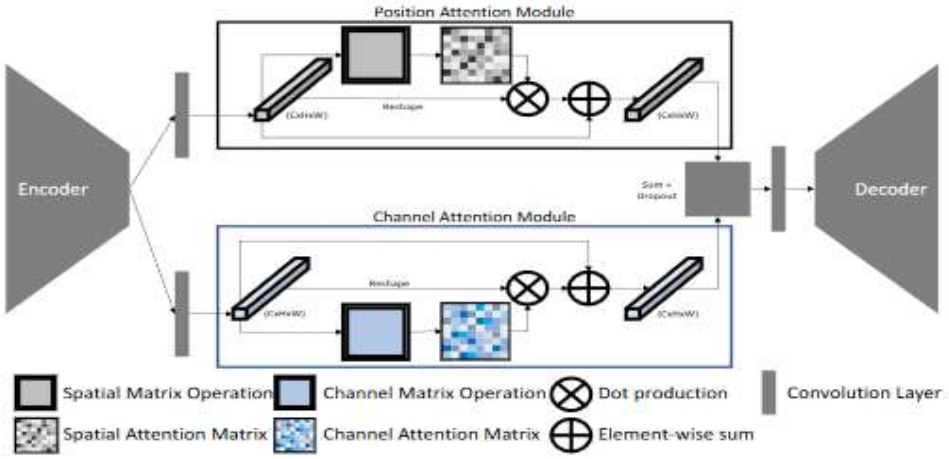

Figure 15: SANet architecture for tooth segmentation on panoramic dental X-ray images [46]

Furthermore, another work by [49] was tagged SANet – SI to identify scripts in natural scene images was conducted. They had text scripts written in Arabic, Bangla, Latin, Symbols, Chinese, Japanese, and Korean languages were used. The architecture they [49] used combines regional and global information to classify these scripts. The architecture has four segmented stages, as shown in Figure 16 ([49]) basic feature extraction, feature fusion, self-attention module, and classification layer. CNN extracted the local features, and a subsequent residual-like fusion module produced multi-scale feature maps. The SRM-Net self-attention model was employed to enhance the classification. The model size was optimized by replacing the conventional fully connected (FC) layers with a convolutional classifier with an average pooling layer.

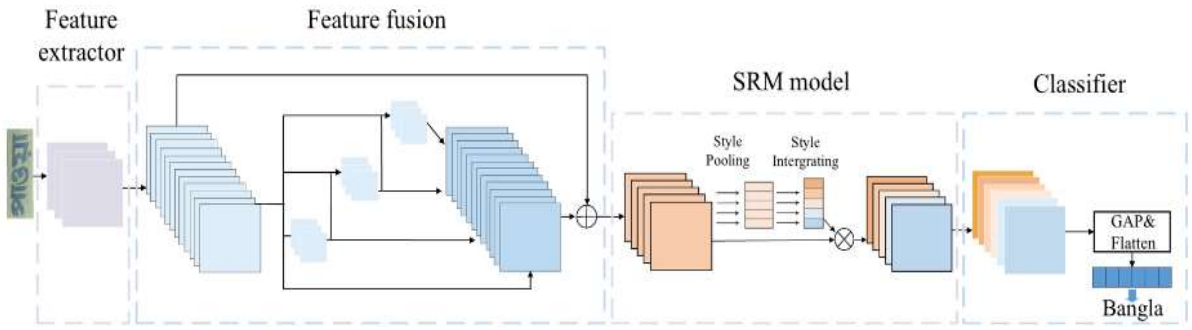

Figure 16: SANet-SI Architecture for Script Identification in scene images [49]

2.3.2 Some results achieved with SANet variants

Different researchers conducted some experiments using the SANet model with differing architectures. Some of these works include [50], where they explained using SANet to Bayesian self-supervised learning through the PyTorch deep learning framework, which produced an improved Compressed Sensing reconstruction quality. Another work was done by [46] on DL based model for segmenting teeth on panoramic dental X-ray images from experiments results which showed a great outing by the SANet in segmenting task with about 94% accuracy. Also, research was conducted by [51] and tagged it *as "a novel segmented attention mechanism and multi-level information fusion network for 6D object pose estimation."* The research discussed the application of SANet for object pose estimation from a single RGB image using deep learning. They used heavy-duty deep models on platforms with constrained resources with about 94% average accuracy rate achieved.

2.4 The proposed model

We will be proposing and implementing a CNN model. We will be using two Python-based deep learning libraries or frameworks (TensorFlow and PyTorch). The proposed model to be developed for this experiment will be developed using convolutional neural networks (CNNs), a family of deep learning models which are known to be effective particularly for image analysis research and experiments. Efforts were put in place to create an accurate and effective model that can perform image classification for emotion recognition from facial expression images. To enhance the speed and the performance of the model, we intend to create a special CNN architecture in conjunction with regularization and optimization techniques. Also, we will be employing the use of hyperparameter tuning with the aim of finding the parameters that will give our model the best performance. These two python-based deep learning libraries we will be using to implement these experiments are briefly discussed below.

2.4.1 Implementation details

For the CNN model to be developed for this implementation, we will be using the Sequential model from ***Keras*** (TensorFlow), a high-level neural networks API, forms the basis of our model. Whereas the PyTorch model defined thus: ***model = nn.Sequential(nn.Linear(in_features, dense layers), nn.ACTIVATION_FUNCTION( ))***. Sequential models are a stack of linear layers with exactly one input tensor and one output tensor in each layer. This makes network design assembly simple because the output of one layer can feed directly into the input. A 2D convolution layer with 32 output filters, each measuring 3x3, is the first layer in our proposed CNN model. The ***Rectified Linear Unit (ReLU),*** which provides non-linearity into the model and enables it to learn more complicated patterns, serves as the activation function for this layer.

The first layer anticipates input images with one or three (RGB) colour channel and dimensions matching *image_size[0]* for height and *image_size[1]* for width. Images in grayscale are usually represented by a single colour channel. We will be using a *2D max pooling layer* to reduce computing complexity and avoid overfitting. The pooling layer reduces the dimensionality of the feature maps by downscaling the input along its spatial dimensions (height and width) using a 2x2 window and the max operation.

To further prevent the possibility of overfitting, we will be adding a Dropout layer, which randomly sets 25% of the input units to 0 at each update during training, for additional regularization. This will make the model more robust and less prone to overfitting because of this type of regularization, which encourages the model to learn redundant representations. Another

convolution layer with a little more complicated structure will be added to our model next. This convolution layer will consist of *64 3x3-sized filters* and the *ReLU activation function*. This convolution layer will allow the model to learn more complex features from the input. To further improve generalization, this layer will be further followed once more by a max pooling layer and a *Dropout layer* with the same configurations as the prior one.

A *Flatten layer* will be added to fill the space between the dense fully connected layers and the convolutional layers. This layer will be used in transforming the 2D output of the preceding layer into a 1D vector so that the fully connected layers can handle it. The *Dense layer* as a fully connected layer will be used which comprised of *128 neurons* with a *ReLU activation function*. This layer will oversee classifying the features that the convolutional layers have extracted and learned. After the Dense layer, another Dropout layer will be added. This *Dropout layer* is different from the previous ones because it has a greater rate (50%), this is aimed at preventing overfitting and making the model more robust.

The last layer to be in the model is another *Dense layer*. This layer will be configured to handle a multi-class classification because the use of *softmax activation function* will be employed, this will be able to outputs a probability distribution over the emotional classes and has several neurons equal to the number of classes. *Categorical cross-entropy* will be used as the loss function for the model's construction because it is suitable for multi-class classification issues. Also, *Adam optimizer*, a *stochastic gradient descent* extension known for its effectiveness, will be used in the model. Finally, we will be using hyperparameter tuning technique to improve the performance of our model. Specifically, we will be using the *GridSearchCV* to determine the ideal *batch size* and the number of *epochs*. The architecture of the proposed CNN model and the workflow diagram of the model using both TensorFlow (Keras) and PyTorch are shown below in Figures 17 – 19.

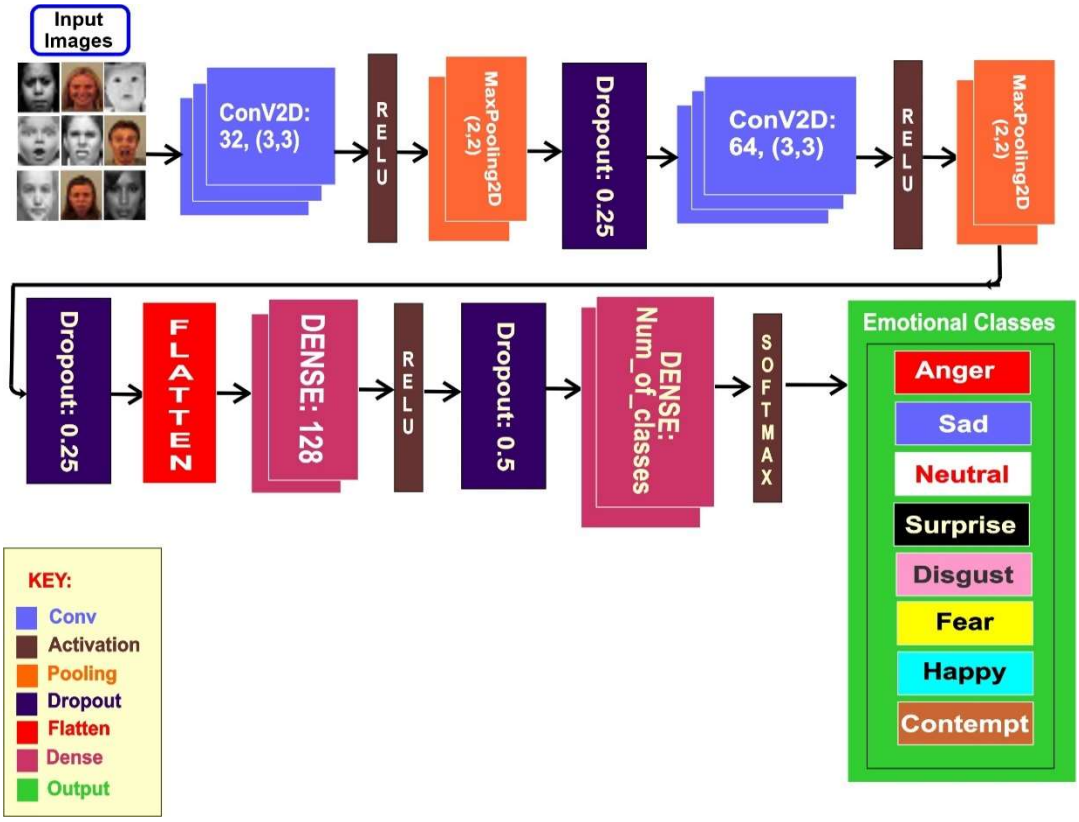

Figure 17: Architecture of the proposed model

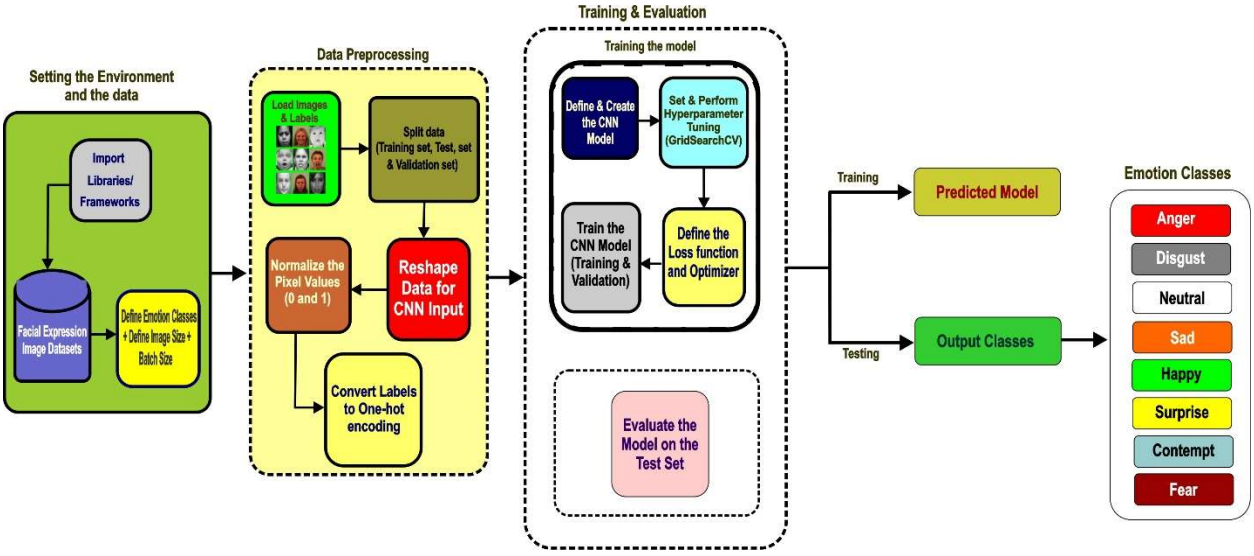

Figure 18: Workflow of the proposed model (using TensorFlow)

Also, the workflow diagram for PyTorch implementation is given in Figure 19.

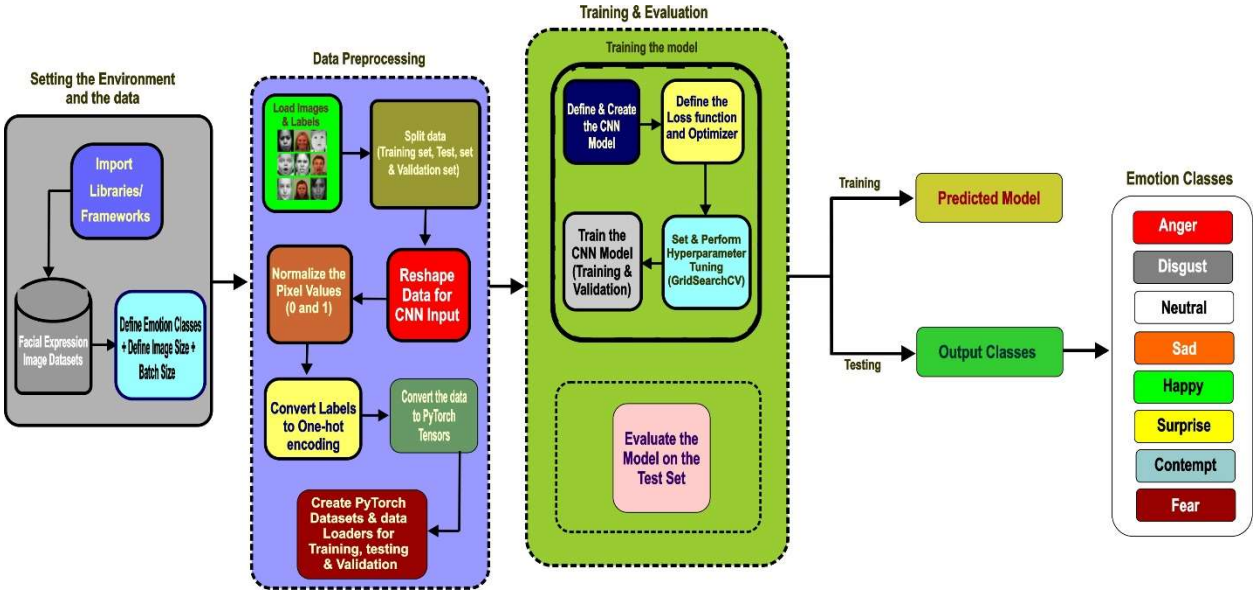

Figure 19: Workflow of the Proposed Model (Using PyTorch)

2.5 Comparison between PyTorch and TensorFlow

In this section, a comparative study will be conducted about PyTorch and TensorFlow. PyTorch and TensorFlow are two powerful and popular deep learning [45] libraries that are used to perform different deep learning experiments. PyTorch and TensorFlow possess some level of similarities as well as some clear-cut differences; we shall therefore be exploring the similarities and the differences between these two libraries more especially as it relates to experiments bordering emotion recognition from facial expression images.

2.5.1 PyTorch

PyTorch is a deep learning library that is embedded into the Python programming language. PyTorch as a package was developed by the Facebook AI Research lab (FAIR) in the year 2016 and was first made available to the public at the beginning of 2017 [46]. It was initially created as a Python-based alternative to the well-known Torch deep learning toolkit by some researchers at the University of California, Berkeley. The Modified BSD license governs the use of the free and open-source project PyTorch. It is widely used in different sectors of human endeavours including academia and business to perform tasks like time series forecasting, natural language processing, and images classification etc. The simplicity of PyTorch's syntax, which

many developers think facilitates quicker prototyping and iterative development, is one of its primary selling points.

Tensors, which are like NumPy's arrays but also permit parallel calculation on graphics processing units (GPUs), are the foundation of PyTorch's capability. Researchers frequently choose PyTorch because it offers a broad range of tensor operations and seamlessly integrates with the Python programming language. The dynamic computation graph used by PyTorch, also known as define-by-run, offers flexibility during runtime. This dynamic graph makes it possible for developers to alter the computation as it is being executed, which improves debugging and makes it possible to create sophisticated dynamic structures.

This contrasts with static graphs, in which the computation graph is determined and specified prior to the execution. Notably, PyTorch presents two primary advantages: it can conduct tensor computations similarly to NumPy, but with the additional capacity of using GPUs for quicker performance, and it can compute gradients automatically, which makes it simpler to train neural networks. Both capabilities are constructed upon C++ backend that utilizes GPUs effectively to carry out the tensor computations. Researchers may utilize the power of numerous GPUs to accelerate model training with PyTorch's distributed training feature, which enables the use of many GPUs across different workstations. This is crucial for deep learning models to scale up and handle enormous datasets.

According to [46], PyTorch is frequently praised for its simple and understandable syntax, which frequently produces more straightforward code. Dynamic computation graphs are provided, enabling runtime flexibility for modifying and improving the model structure. A Convolutional Neural Network (CNN) model for emotion recognition can be created using the PyTorch framework. Thisis because of its dynamism in computing graphs, according to the researchers, made it simpler to debug and improve the model.

They [46] did point out, though, that it occasionally leads to slower training durations than TensorFlow, particularly for big datasets. As argued above, a notable drawback of PyTorch is that it usually takes longer period to train models than TensorFlow, more especially for large datasets. Even though PyTorch offers strong GPU acceleration, its dynamic nature can occasionally result in computations that are not as well-optimized, especially when the model design and training processes are not streamlined.

2.5.2 TensorFlow

TensorFlow is an open-source python-based library for numerical computation that allows for machine learning and deep neural network research activities. It was developed and made available for research purposes by the Google Brain team in 2015 [47]. Its name is derived from the computations that neural networks carry out on tensors, which are multidimensional data arrays. The primary competitive advantage of TensorFlow is its extensive ecosystem, which offers strong capabilities for a range of jobs, from model construction and training through deployment. TensorFlow, in contrast to PyTorch, uses a static computation graph, commonly referred to as define-and-run. This paradigm offers performance benefits for applications that call for repeated execution of the same computation by defining the computation graph only once. TensorFlow additionally facilitates distributed computing, which enables models to be simultaneously trained on several nodes or devices. For large-scale, production-level applications, its extensive toolkit which includes TensorBoard for visualization and TensorFlow Extended (TFX) for production pipelines makes it a desirable option.

TensorFlow on the other hand due to its more complicated syntax, might be having the tendency of a longer learning curve, even though it usually performs well in terms of computing efficiency and scalability [47]. TensorFlow is therefore a well-liked option for commercial applications that include massive datasets and demand excellent speed. Despite being less adaptable than PyTorch's dynamic computation graph, TensorFlow's static computation graph allowed for more effective computations because it could be optimised before execution [47].

TensorFlow has an extensive toolkit including TensorBoard, which is used to track the training procedure and visualize the model's performance. The TensorBoard tool proved very helpful for fine-tuning hyperparameters and locating model bottlenecks. Machine learning pipeline, from data validation and preprocessing to model training, evaluation, and deployment are properly managed through the usage of the TensorFlow Extended (TFX) package. Despite TensorFlow's advantages, one basic thing noted is that TensorFlow takes significantly longer learning curve, mostly because of the syntax increased complexity and the computation graphs static structure [47].

2.5.3 The comparison (PyTorch vs TensorFlow)

Considering the discussion above on PyTorch and TensorFlow above, we will briefly discuss the similarities and differences between PyTorch and TensorFlow:

Simplicity

PyTorch is known for its simplicity when it comes to syntax. It is on this wise that many developers prefer PyTorch to TensorFlow because it facilitates quicker prototyping and iterative development. TensorFlow possesses some level of syntax complexity which in a way impedes quicker prototyping and iterative development.

Parallel Calculation

Parallel computations are easier to be carried using tensors, this allows parallel calculation on graphics processing units (GPUs); PyTorch ability to carried out tensor computations in a speedy manner are the foundation of PyTorch's capability but with the additional capacity of using GPUs for quicker performance. Also, it can compute gradients automatically, which makes training a neural network simpler. In contrast, TensorFlow lacks these capabilities, as such huge difference.

Debuggability

One basic advantage of PyTorch is that it enables the building of dynamic graphs. Developers can therefore quickly spot bugs by inspecting intermediate values in the computation graph. In addition, PyTorch has a number of built-in debuggers (such as *pdb* and *ipdb*) that can be used to examine the execution of code in a more detail. On the other hand, the absence of visualization tools in PyTorch often makes it challenging to debug complicated models. For TensorFlow, it comes with more complete visualization tools pre-installed (such as TensorBoard).

These technologies are very useful for comprehending complicated models and finding bugs. TensorFlow also comes with a variety of helpful debugger hooks that may be utilized to suspend execution at locations in the computation graph. The ability to dynamically adjust the computation graph mid-execution ("rewinding" or "stepping through" code) can be challenging with TensorFlow, which is one of its drawbacks.

Performance Factor

When it comes to performance factors between PyTorch and TensorFlow, this subject lacks a clear-cut solution because it is dependent on a variety of elements, such as the algorithm being used, the hardware in use, and the general effectiveness of the code. In broad terms, TensorFlow may be quicker than PyTorch for some computations, while PyTorch may be quicker for other computations.

Parallel data processing

Both frameworks provide data parallelism capabilities, enabling effective deep learning on huge datasets. Interestingly, data parallelism implementations are varied in these two frameworks. For PyTorch, for data parallelism method, PyTorch divides the dataset into smaller parts before parallelizing the model training on each piece. In general, this is more effective than model parallelism, which involves training the model on the whole dataset at once. However, since PyTorch needs to handle CUDA threads carefully, it can be more challenging to implement data-parallelism. On the other hand, TensorFlow trains models concurrently using a method known as *Asynchronous Stochastic Gradient Descent (ASGD*). Although ASGD is typically less effective than data parallelism, TensorFlow makes it simpler to implement.

Distributable model training

The distribution of calculations over numerous hardware devices such as CPUs and GPUs, and even several computers, is made simpler by TensorFlow. For TensorFlow, there are some built-in capabilities for distributed learning or training in PyTorch, although it is not as robust or user-friendly as TensorFlow's.

2.6 Remarks and conclusions

ResNet is a neural network architecture developed to tackle the challenge of vanishing or exploding gradients and the degradation problem when training the deep networks. It makes use of skip connections, or shortcuts, to allow gradients backpropagation to prior layers. The usage of these skip connections enables the model capability to learn an identity function that guarantees the layer above will perform no less than the lower layer. The ResNet model is known to be efficient with complex deep learning tasks such as image and speech recognition. Applications requiring deep learning can benefit from using ResNet models because they lessen the difficulties associated with developing extremely deep neural networks. They may perform better in some tasks due to their capacity to effectively transmit gradients through their layers and prevent overfitting.

VGG by architectural alignment is a deep convolutional neural network. VGG is known for its lack of complexity and how it is deeper than previous versions since it used numerous layers of 3x3 convolutions. The use of multiple smaller convolutions instead of fewer bigger ones enables the model to learn more complicated characteristics from the data. Contrary to other deep learning models, VGG models demand more computing power and system resources. Notwithstanding, it is known for its excellent performance on different image recognition tasks because of their ability

to learn depth within a given data set. Although VGG models can achieve high accuracy on tasks like image recognition, their resource-intensive nature makes them a poor fit for applications where speed is essential or where resources are scarce. VGG models, however, can perform admirably for applications where the utmost precision is required, and system resources do not pose an issue.

For SANet on the other hand, is a kind of model that has a self-attention mechanism that enables it to concentrate on various aspects of the input when creating output, which can result in better performance on several tasks. Regardless of where the input data are in the sequence, it aids the model in understanding context and interdependencies. When it comes to tasks that call for an awareness of context and long-distance dependencies, SANet models have shown to be quite successful. SANet models have found applicability in many different tasks, including image recognition, natural language processing, etc.

SANet models' self-attention mechanism makes them a useful tool for a wide range of applications. On a variety of tasks, their performance can be enhanced by their capacity to manage context and long-range dependencies in data. SANets, like VGG, can be computationally demanding, especially for bigger inputs, hence they might not be appropriate for applications with limited resources.

For PyTorch and TensorFlow, the interactive computation and straightforward debugging features of PyTorch are well-liked. It seamlessly interacts with Python, making the code more straightforward to comprehend, develop, and debug. It provides excellent support which enables distributable training capabilities. For academics and developers searching for a platform that allows rapid prototyping and iterative approaches, as well as supporting the creation and training of sophisticated deep learning models, PyTorch offers a great choice. In contrast to TensorFlow, it could prove less suited for deployment in real-world settings, mainly if performance is critical.

TensorFlow is known for its scalability and production readiness and can function on a wide range of platforms, including mobile and edge devices. TensorFlow.js for JavaScript environments and TensorFlow Lite for mobile and IoT devices are also available. Although less adaptable than PyTorch's dynamic network, its static computational graph frequently results in speed improvements. When it comes to large-scale, production-level ML/DL jobs, TensorFlow offers a solid option. For complicated applications demanding excellent performance, its reliability, scalability, and variety of deployment choices make it particularly ideal. TensorFlow's steep

learning curve and less user-friendly design compared to PyTorch, however, can be a hurdle for beginners or people who value simplicity and ease of use. In conclusion, the choice of any of these deep learning models and these frameworks largely depends on the task at hand, datasets, computer power and resources availability.

CHAPTER 3

METRICS FOR MODEL CLASSIFICATION

A fundamental component of machine learning and data science is model classification. Model classification is essential to many different applications, from financial forecasting to medical diagnostics [48]. These classification models' performance is assessed using a range of measures, each of which offers a different viewpoint on the model's capacity to correctly and consistently categorize or classify data. Understanding these metrics is essential to efficient model development and evaluation since the choice of metric has a substantial impact on how a model's performance and suitability for a given task is interpreted [49]. Performance indicators are pretty helpful when comparing and evaluating various machine learning or classification models [49].

Numerous metrics can be used to evaluate the performance of any multi-class classification experiments. These metrics are beneficial for comparative analysis of the results of two or more different models, examining the behaviour of the same model after performing hyperparameters tuning. Confusion Matrix serves as the foundation for many metrics because it contains all the necessary data regarding the effectiveness of an algorithm and classification rules [49]. The confusion matrix is a cross table that counts the instances between two raters, the true or actual classification, and the predicted classification. According to [49], the rows shows the classification, whereas the columns represent model prediction. Because the classes are listed in rows and columns in the same order, the correctly categorized elements are found on the major diagonal, from top left to bottom right, corresponding to the percentage of times the two raters agree. Figure 20 shows a sample confusion matrix according to [49] with dummy data.

		PREDICTED classification				
	Classes	a	b	c	d	Total
ACTUAL classification	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	**52**

Figure 20: A sample of confusion matrix with dummy data [49]

The metrics we will be evaluating for this work are accuracy, F1 score, evaluation of the training, evaluation of the testing, memory usage, and brief remarks and conclusion.

3.1 Accuracy

Accuracy is one of the most commonly used metrics in model classification. It is described as the percentage of accurate predictions made by the model. It is mathematically determined by dividing the total number of predictions by the sum of true positives and negatives [50]. Furthermore, when classes are balanced or nearly equal numbers of instances in each class, accuracy can serve as a quick and straightforward indicator of model performance. For example, a model with an accuracy of 0.9 produces accurate predictions 90% of the time. However, in situations with imbalanced classes, where one class vastly outnumbers the other (as with this experiment), accuracy can be a misleading indicator.

Consider a model created to find an illness that only affects 1% of the population. Despite being unable to accurately identify any cases of the disease, a naive model that predicts "no disease" for all patients would have a 99% accuracy rate. In certain circumstances, different metrics like precision, recall, or the F1 score may offer a more accurate representation of model performance [50]. As mentioned above , we will examine the F1 Score, another crucial model classification metric, because of the imbalanced classes at our disposal. Mathematically, as proposed by [50], accuracy is given by the equation 3.1 below:

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP+TN}}{\boldsymbol{TP + TN + FP + FN}} \tag{3.1}$$

where**:**

True Positives (TP) – this indicates the number of cases where both the actual class and the predicted class of the data point are positive. In other words, the model correctly identified a positive instance as positive,

True Negatives (TN) – this indicates the number of cases where both the actual class and the predicted class of the data point are negative. The model correctly identified a negative instance as negative,

False Positives (FP) – this indicates the number of cases where the actual class is negative, but the predicted class is positive,

False Negatives (FN) – this indicates the number of cases where the actual class is positive, but the predicted class is negative.

3.2 F1 Score

As discussed above, in circumstances of imbalanced classes, when accuracy might not be a reliable indicator of model performance, the F1 score is a metric that aims to balance the precision and recall of a model. According to [50], it is described as the harmonic mean of the metrics precision and recall, which are crucial for model classification in and of themselves.

The fraction of accurate positive predictions among all positive predictions generated by the model is known as **precision**, sometimes referred to as the positive predictive value. It is a crucial metric in situations where the cost of false positives is considerable. For instance, a highly precise model in email spam detection would reduce the possibility that crucial communications would be mistakenly classified as spam [50]. Mathematically, precision is given by the equation [50]:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FP}} \tag{3.2}$$

Recall, often called sensitivity or true positive rate, quantifies the percentage of real positives the model accurately recognized. When the cost of false negatives is high, it is essential. For instance, a high recall is preferred in medical diagnosis models to ensure that the greatest number of positive cases are found for additional research [50]. The equation for the recall as presented by [50] is given as below:

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}} \tag{3.3}$$

The above formula can be understood as follows: how many did the model manage to uncover among everything positive? Even though they may occasionally misidentify some negative examples as positive, a model with high recall does an excellent job of locating all the positive cases in the data. All or a significant portion of the positive points in the data cannot be found by a model with low recall.

The F1 score comes in as a single metric that strikes a balance between precision and recall. In addition to making accurate predictions and reducing both types of errors, a model with a high F1 score also excels in recall and accuracy. The F1 score has some limits, much like any measures. It assumes that false positives and negatives cost the same amount, which may not be true in all cases. Therefore, the cost of various sorts of errors should constantly be considered when choosing a measure [50]. F1 score is given by equation 3.4 as presented by [50]:

$$\boldsymbol{F1\ Score} = \frac{2}{\frac{1}{\boldsymbol{Precision}} + \frac{1}{\boldsymbol{Recall}}} = \frac{\mathbf{2\ x\ Precision\ x\ Recall}}{\boldsymbol{Precision + Recall}} \tag{3.4}$$

3.3 AUC – ROC

The Area Under the Receiver Operating Characteristic (AUC-ROC) performance measurement is used for classification issues at various threshold values. The true positive rate (sensitivity) vs the false positive rate (1-specificity) for different threshold levels is plotted on the ROC curve. According to [50], the AUC symbolizes the level or measure of separability, demonstrating how well the model can distinguish between classes. At all threshold values, the model accurately distinguishes between positive and negative cases, achieving an AUC of 1, which denotes perfect classification ability.

An AUC of 0.5, on the other hand, means that the model has no classification capacity and performs no better than chance. AUC values between 0.5 and 1 indicate that a model can classify data, with higher values suggesting improved performance. The ROC curve is a two-dimensional graph whereby TPR stands for the y-axis and FPR represents the x-axis. Furthermore, the ROC curve is a graph that displays how well a classification model performs across all thresholds. This graph shows the two parameters: True Positive Rate (TPR) and False Positive Rate (FPR). Figure 21, as presented by [57], shows a basic ROC Curve showing essential points on the graph.

According to [50], True Positive Rate (TPR) defined by equation, as below:

$$\boldsymbol{TPR} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}} \tag{3.5}$$

Whereas the False Positive Rate (FPR) is also given by:

$$\boldsymbol{FPR} = \frac{\boldsymbol{FP}}{\boldsymbol{TN} + \boldsymbol{FP}} \tag{3.6}$$

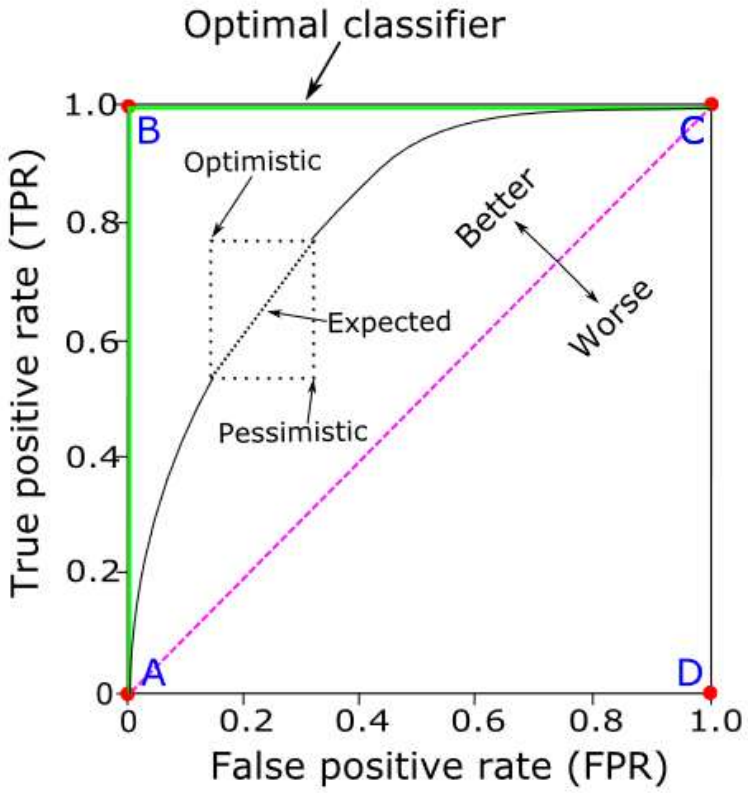

Figure 21: Basic ROC Curve showing important points [57]

The AUC-ROC metric is beneficial when the ideal classification threshold must be chosen based on the trade-off between sensitivity and specificity and is unknown a priori. This makes it a popular option for evaluating models in several applications, from medical diagnosis to credit fraud detection. It is also resilient to imbalanced classes [50].

3.4 Evaluation of the training

Evaluating the training process is a crucial aspect of model development. It entails evaluating the model's performance on the training set of data and adjusting parameters to raise it. The model's performance on the training set of data informs iterative tuning to its parameters [49]. A frequent fundamental problem with model training is overfitting when the model performs well on training data but badly on unobserved data. Overly complicated models that capture noise in training data but do not generalize to new data commonly cause overfitting.

Overfitting can be reduced using strategies like regularization and early halting (stoppage), which encourage the creation of more straightforward models that generalize better to unknown data [49]. The selection of the optimization algorithm is a crucial component of training evaluation. A measure of the difference between the model's predictions and the actual values, the loss function, is what the optimization method oversees minimizing. The optimisation technique chosen can strongly impact the model's performance. Different optimization algorithms, such as stochastic gradient descent and Adam, have different strengths and disadvantages [49].

For this experiment, the training will be evaluated to avoid overfitting, as such, the following measures will be implemented during the model's training. The proposed CNN model will be implemented side by side using TensorFlow and PyTorch frameworks. The training will be carried out using a CPU – based Computer. For TensorFlow, we will use the *Sequential ()* to define the CNN model and *KerasClassifier ()* to create the CNN model. We will use *categorical_crossentropy, adam optimizer* and accuracy for the loss function, optimizer, and metrics.

For hyperparameter tuning, we will be defining a parameter grid for both *epoch* and the *batch size*; for the epoch, we will be having **(10, 20)** as the *epoch* values within the grid and **(32, 64)** as the *batch size* within the grid. ***GridSearchCV*** will perform the hyperparameter tuning during the model's training. The *training set* will take *70%* of the dataset whereas *10%* is for the *validation set* and the remaining *20%* as the *test set*. The PyTorch implementation on the other hand will have the dataset split in the same way as that of the TensorFlow implementation. Further training evaluation details include ***CrossEntropyLoss*** as the loss function, ***adam*** as the optimizer. The hyperparameter tuning is the same as that of TensorFlow. Also, the *accuracy vs epoch* and the *error rate* graph will be plotted to evaluate the model's training.

3.5 Evaluation of the testing

This is a crucial step in developing a deep learning model, it provides a more accurate measurement of the model's predictive capacity. It entails evaluating the model's performance using unobserved or unseen data not used during training. This makes it more likely that the model will learn patterns that transfer to new data rather than only memorizing the training set of data [49]. Cross-validation is a method frequently used in testing evaluation. The data are partitioned into 'k' subsets or 'folds' for cross-validation. The model is then trained '*k*' times, utilizing the leftover fold for testing and 'k-1' folds for training as presented by [49].

To produce a more reliable measure of the model's predictive capacity, the model's performance is then averaged over the '*k*' trials. Cross-validation reduces the risk of overfitting while aiming a more accurate assessment of the model's performance on unobserved or unseen data [49]. The selection of a performance metric is a critical component in testing evaluation. The cost of various types of errors should be considered when choosing a measure because different metrics offer different perspectives on the model's performance. For instance, recall or the F1 score may be more appropriate than accuracy in a medical diagnosis task when false negatives are costly [50]. For this experiment, the testing evaluation will be carried out using 20% of the dataset as will be taken from the complete datasets.

3.6 Memory usage

Memory usage is a crucial factor to consider in model classification, especially when big datasets or few computational resources are available. The number of parameters in a model, the size of the training data, and the complexity of the computations involved in training and prediction can all impact how much memory a model uses [51]. High memory usage may occasionally be a constraint on the development and deployment of models. For instance, a model with many parameters can need more memory than is available, making it impossible to train or use the model for prediction. Also, big datasets might not fit in memory, so several of such as batch processing or online learning are employed to handle such [51].

Memory usage can be optimized using methods like dimensionality reduction without significantly compromising model performance. Data is transformed into a lower-dimensional space using dimensionality reduction techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbour Embedding (t-SNE), reducing the memory needed to store and interpret the data. These techniques can also enhance model performance by reducing the amount of noise and redundant information in the data [51].

For this experiment, the FER2013 dataset is the largest of the datasets we will use. It contains 35,887 images in total, Natural Human Face image dataset has 5,558, Karolinska dataset has 2,938 and the smallest of the four datasets CK+ has 981 images. We will be employing batch processing of the images during the model training; this will help mitigate the challenge of memory usage. Computing the total number of parameters using the proposed CNN model on the four selected images datasets (CK+, FER2013, Karolinska, Natural Human Face image dataset) will give different parameter values due to different image sizes and numbers of emotion classes. The CK+ and the FER2013 dataset have a size of 48x48 and seven (7) emotion classes each; Karolinska

dataset also has seven (7) emotion classes, and the Natural Human face image dataset has eight (8) emotion classes and an image size of 224x224.

The computation below is for an image of size 48x48 and seven (7) emotion classes:

i. **Conv2D Layer (64 filters, (3, 3) kernel size)**: Each filter has a 3x3x1 weight matrix, and there are 64 such filters. Including biases, the total parameters would be:

64x3x3x1 (weights) + 64 (biases) = 640 parameters.

ii. **MaxPooling2D Layer**: No parameters in max-pooling layer.

iii. **Dropout Layer (0.25)**: No parameters in dropout layer.

iv. **Conv2D Layer (128 filters, (3, 3) kernel size)**: The input depth to this layer will be 64 (the number of filters in the previous Conv2D layer). So, the total parameters for this layer would be:

128(filters)3x3x64 (weights) + 128 (biases) = 73,856 parameters.

v. **MaxPooling2D Layer**: No parameters in max-pooling layer.

vi. **Dropout Layer (0.25)**: No parameters in dropout layer.

vii. **Flatten Layer**: This layer just reshapes the input and has no parameters. The output of this layer will be of size:

[48/2/2]x[48/2/2]x128 = 12x12x128 = 18,432

viii. **Dense Layer (128 neurons)**: Each neuron in this layer is fully connected to all outputs from the previous layer. Including biases, the total parameters would be:

128x18,432 (weights) + 128 (biases) = 2,359,424 parameters.

ix. **Dropout Layer (0.5)**: No parameters in dropout layer.

x. **Dense Layer (num_classes = 7 neurons)**: Each neuron in this layer is fully connected to all outputs from the previous layer. Including biases, the total parameters would be:

7x128 (weights) + 7 (biases) = 903 parameters

In total, the model has:

640 (first Conv2D) + 73,856 (second Conv2D) + 2,359,424 (first Dense) + 903 (second Dense) = 2,434,823 parameters.

It is worthy of note that when we perform the same computation for the same image size but with eight emotion classes, we will have a slightly different total parameters value of ***2,434,952.*** This is due to the increase in the number of parameters of the second dense layer to 1,032 as against the 903 parameters value of the seven (7) emotion classes.

In the case where the image size is 224x224 and has seven (7) emotion classes, the total numbers of parameters will be computed summarily thus: In total, the model with a 224x224 image size and seven (7) emotion classes has:

640 (first Conv2D) + 73,856 (second Conv2D) + 51,380,352 (first Dense) + 903 (second Dense) = 51,455,751 parameters and 51,455,880 parameters for eight (8) emotion classes**.**

3.7 Remarks and conclusions

A wide range of metrics are available to assess and compare the performance of various models in the complicated and diverse subject of model classification. Each metric offers a distinct viewpoint on the model's performance, with its advantages and disadvantages. Understanding these metrics is essential to efficient model construction and evaluation since the choice of metric substantially impacts how a model's performance and suitability for a given task are interpreted [49]. Each metric and evaluation technique delivers valuable insights into the model's capacity to classify data reliably and effectively, from accuracy to the F1 score, from AUC-ROC and to memory use. However, the performance of a model cannot be fully captured by a single metric. As a result, when creating and evaluating models, it is crucial to consider a range of metrics and evaluation techniques into account [50].

More study is required to create reliable and understandable metrics for model classification. Metrics that can reliably evaluate machine learning or deep learning models' performance and direct their development are becoming increasingly necessary as they get more complicated and are used in a broader range of scenarios. Additionally, there is a rising demand for scalable and practical approaches for model training and evaluation due to the amount and complexity of datasets [49]. A closer observation of the memory usage shows that due to the increased spatial resolution handled by the dense layer, when the image size is increased from 48x48 to 224x224, the total number of parameters in the CNN model dramatically rises. Additionally, adding more classes has a negligible impact on the model's complexity, resulting in a slight increase in the parameters. As a result, the size of the image affects memory use more significantly than the number of emotion classes. In conclusion, many intriguing potentials exist for further research and improvement in model classification metrics.

CHAPTER 4

IMPLEMENTATION

4.1 Models

For this thesis work, as we mentioned earlier, we will be implementing five (5) deep learning models for the emotion recognition task from facial expression images. These five (5) models are ResNet model, VGG model, SANet model, and the proposed model to be implemented using TensorFlow and PyTorch. In this chapter, we will be providing details on how these models will be implemented for the task at hand and the results obtained from the experiments.

4.2 Data preprocessing

In every DL/ML task, the ability to efficiently load and preprocess data is an essential aspect to be given due attention when developing models. In this section we will discuss the methodology we used to extract and preprocess our dataset, which consists of images representing emotions. To be able to train the model using the image datasets and perform the prediction, certain data processing activities must be carried out. The data processing activities to be performed in this experiment are loading the images and labels, defining the emotion classes, defining the image size, splitting the dataset into training, validation, and test sets, Splitting the dataset, reshape the data for CNN input, normalizing the Pixel values, and converting labels to one-hot encoding. We will be explaining how each of the above-mentioned data preprocessing tasks were carried out during the experiments.

4.2.1 Loading the images and labels

To be able to organize our data, we adopted a directory structure. Each emotion has its directory with the directory name indicating the represented emotion. Within each directory are grayscale or RGB images based on the dataset that exemplify that emotion. To store our image data and corresponding labels in a manner we created two lists: one, for storing the image data and another for aggregating the associated labels.

Our implementation follows a step-by-step process for extracting images:

i. We first defined a function called ***load_data ( )*** within which the loading of images and labels will be carried out.
ii. Starting with each emotion class in our defined list of emotions we locate the directory containing images for that specific emotion using Python's ***os.path.join*** function.
iii. Within each emotion directory we iterate through all the image files. Each image goes through the steps:

a. Determining its path.
b. Using the Python Imaging Library (PIL) we retrieve the image. Convert it to grayscale (if the image is in RGB format).
c. We resize each image to a predetermined size ensuring uniformity in dimensions.
d. After post processing we convert each image into a NumPy array. Add it to our list of images.
e. At the time we add the label to our list of labels based on the index of that emotion class within our predefined list.

iv. To enhance efficiency, we convert both lists of images and labels into NumPy arrays once all images have been loaded and preprocessed. This conversion allows for flexibility in model training phase.

By calling the ***load_data ( )*** function we obtain arrays of images and labels which are ready for further processing in our deep learning framework.

4.2.2 Defining the emotion classes and image size

We further defined the emotion classes to be used for the experiments using a python dictionary. The emotion classes names differ from one dataset to another. More explicitly, with the exception of the natural human face images dataset which has eight (8) emotion classes, the remaining three (3) datasets have seven (7) emotion classes. The eight emotion classes defined are anger, contempt, disgust, fear, happy, neutral, sadness, and surprise. The CK+ dataset is devoid of the disgust emotion class while the FER2013 and the Karolinka directed emotional facial images is without the contempt emotion class. Furthermore, we set a default image size of 48 x 48.

4.2.3 Data partitioning

The evaluation technique is a key component of developing a reliable and generalizable machine learning or deep learning model. To evaluate the model's performance and ensure that it can generalize successfully to new, unseen data, the data must be divided properly into training, validation, and testing sets. The method we used for this partitioning is explained as follows.

Initial data split – Extracting the Test Set

Our main goal was to designate a portion of the data for testing, making sure that this data is kept untouched during the training process. To accomplish this:

i. We used the scikit-learn library's ***train_test_split*** function to split the datasets into the three (3) sets – the training set validation set, and the test set.

ii. The dataset, which consisted of labels as the target data and images as the input data, was divided, with 20% as the test set. For further separation into training and validation sets, the remaining 80% was set aside.
iii. A ***random_state*** of 42 was used to ensure uniformity and reproducibility in our experiments. This ensures that the data is divided consistently each time the code is run.

Separating the Training and Validation Sets in a Refined Data Split:

The training and validation sets were made by splitting the remaining 80% of the data, which had been temporarily kept in ***X_temp*** and ***y_temp***: using the train_test_split function once more, 25% of this subset was given to the validation set, which corresponds to 20% of the original dataset because 0.25 x 0.8 = 0.2. The training set will take the remaining data. To guarantee consistency, a constant random_state of 42 was kept.

By the end of these processes, we had three different datasets:

i. **The training set:** our deep learning models are trained using the training set (X_train and y_train).
ii. **Validation set:** (X_val and y_val): enables hyperparameter tuning and prevents overfitting.
iii. **Test set:** (X_test and y_test): set aside specifically to evaluate the model's performance on unseen data.

This method of structured partitioning not only makes it easier to evaluate our model's capabilities objectively, but it also ensures the accuracy and dependability of the results that follow.

4.2.4 Reshape the data for CNN Input

For Convolutional Neural Networks (CNNs) tasks, the input data must follow a specified shape or format in order for Convolutional Neural Networks (CNNs) to be trained successfully. CNNs commonly demand data in a 4D array, where each dimension corresponds to a batch size, a width, a height, and a channel. Our datasets only have one channel because all the images are in grayscale except for the Karolinska Directed Emotion Facial Images dataset. The techniques taken to ensure that our data assumed this desired shape, making it suitable for CNN-based processing, are described below. We made use of the reshape function for the training dataset, *X_train.* The goal was to make sure the data meet the following criteria:

i. The batch size is ***X_train.shape[0]***, which is equal to the number of training samples.

ii. The image height and width are as defined by our image_size tuple*:* ***image_size[0]*** and ***image_size[1]*** respectively.

iii. For grayscale images, the channel dimension is set to 1 whereas for RGB images, the channel dimension is set to 3.

This is how the reshaped data is shown: ***X_train.shape[0], image_size[0], image_size[1], 1.***

Validation dataset *(X_val)* and test dataset (*X_test*) reshaping:

Using a similar process, the validation dataset (***X_val***) and the test dataset (***X_test***) were both reshaped in the same way. The same four-dimensional structure is embodied in the reshaped data for both sets, guaranteeing uniformity throughout all the subsets.

Expected outcome:

Our datasets (***X_train, X_val, and X_test***) are flawlessly structured for CNN-based analysis at the end of this reshaping operation. This guarantees that the input data perfectly matches the architectural requirements for CNNs, enabling the best performance and effectiveness during both the training and evaluation phases. It is crucial to ensure such rigorous pre-processing since it paves the way for the later stages of model creation and promotes accuracy, effectiveness, and repeatability in our studies.

4.2.5 Normalizing the pixel values

Normalizing the pixel values of images is a vital step in the pipeline for data preprocessing. The images' pixel values are scaled through this procedure to stay within a certain range. We attempted to normalize the pixel values of our images to fall within the range of 0 and 1 for the purpose of our experiment. This normalization speeds up the training process while ensuring that our Convolutional Neural Network (CNN) models operate under more stable conditions. The process and justification for normalizing the pixel values in our dataset are explained below.

To Float32 Data Type Conversion:

Our datasets ***(X_train, X_val, and X_test)*** are initially stored as integers that represent the values of pixels. It is essential to transform these integers into floating-point numbers in order to simplify normalization. The ***astype('float32')*** method is used to convert the data type of our images from integer to floating point (specifically 32-bit floating point), which enables the representation of decimal values.

Pixel Value Normalization

After the conversion from integer to floating point, we proceed to normalize the pixel values. For an 8-bit grayscale image, pixel values are usually in the range of 0 to 255. Normalizing these values involves: Dividing the pixel values of each image in our datasets by 255, the maximum possible value for an 8-bit image. This division scales all pixel values to fall within the [0, 1] range, which is often beneficial when working with neural network architectures, as it ensures that the input features (pixel values in our case) are on a similar scale. The resulting datasets ***(X_train, X_val, and X_test)*** now include images with floating-point images with pixel values ranging from 0 to 1. The normalization process will help us achieve the following:

i. It eases the learning process by providing us with a constrained and reliable input range.
ii. As gradients are effectively scaled and centered around the same range, this often results in faster convergence during the training phase.
iii. Lessens the possibility of running into problems with numerical stability, which are frequently a worry when working with high-dimensional data like images.

In our data preprocessing routine, the rigorous normalization step is a crucial step that lays the groundwork for effective and efficient model training and evaluation in our deep learning pipeline.

4.2.6 Converting Labels to One-Hot Encoding

Converting the categorical labels in deep learning classification tasks to a format that facilitates improved model performance is advantageous. One-hot encoding is a popular example of such a format. We converted the labels from categorical labels to one-hot encoding; below we outlined the method and justification for the one-hot encoding of the labels in our emotion classification dataset in this subsection.

Determining the number of classes

First, we count how many different emotion classes (categories) there are in our dataset. This is crucial since it determines how long the one-hot encoded vectors will be. In our experiment, the length of the list ***emotion_classes***, which contains the names (or identifiers) of all distinct emotion classes in the dataset, is allocated to the variable we named ***num_classes***. The ***np.eye(num_classes)*** creates an identity matrix of shape (***num_classes, num_classes***). In this matrix, the diagonal elements are 1, and all off-diagonal elements are 0.

Mapping Labels to One-Hot Vectors

For each set of labels ***(training, validation, and testing)***, we then map the categorical label indices to their corresponding one-hot encoded representations:

i. For instance, ***y_train = np.eye(num_classes)[y_train]*** maps each label in ***y_train*** to a one-hot encoded vector using the identity matrix as a lookup.
ii. If a particular sample in ***y_train*** has a label ***i***, this operation will map it to the ***i-th*** row of the identity matrix, which is a one-hot encoded vector where the ***i-th*** element is 1 and all other elements are 0.
iii. Similar transformations are applied to the validation and test labels ***(y_val and y_test***, respectively) ensuring uniformity in label format across all data subsets.

Expected Outcome

Instead of integer labels, the output label arrays ***(y_train, y_val, and y_test)*** now include one-hot encoded vectors. The index of the '1' value in each row of these arrays denotes the class of the sample, and each column corresponds to a one-hot vector. For instance, if a sample belongs to class happy and there are three classes (angry, happy , and fear), its one-hot encoded label would be [0, 1, 0]. This conversion serves the following key functions within our pipeline:

i. It clarifies the issue and guarantees that the labels are interpreted by the model as discrete categories rather than as ordinal values.
ii. It is compatible with various loss functions, including categorical cross-entropy, that are used in neural networks to perform classification tasks.
iii. It aids in a clearer and more interpretable evaluation of the model's predictions, simplifying the process of comparing predicted labels with true labels.

This deliberate encoding phase is essential for creating a clear and efficient training and high-precision evaluation pipeline for our emotion classification models.

4.3 Hyperparameter Tuning

Achieving an optimal machine learning/deep learning model's performance often requires tuning a number of hyperparameters. In our emotion classification task, we made use of a systematic approach called Grid Search, a widely used technique for hyperparameter optimization. In this part, we go into detail about how our Convolutional Neural Network (CNN) model implements Grid Search for hyperparameter tuning.

Defining the Parameter Grid

Before starting the hyperparameter tuning procedure, we identify the collection of hyperparameters that we want to optimize. We define:

i. A ***param_grid*** dictionary, where each key is the name of a hyperparameter, and the corresponding value is a list of possible values for that hyperparameter.

ii. The two key hyperparameters ***batch_size*** (with candidate values ***32*** and ***64***) and epochs (with candidate values ***10*** and ***20***) were the ones we concentrated on tweaking in this experiment. Epochs are the number of times the learning algorithm will go through the full training dataset, and ***batch_size*** is the number of samples that will be sent through the network at once.

Grid Search Execution

Following the creation of the parameter grid, the Grid Search procedure is carried out as follows:

i. We make use of the *scikit-learn* ***GridSearchCV*** class, which conducts cross-validated grid search over a parameter grid.

ii. The CNN model that we have already defined is the estimator argument.

iii. We assigned the dictionary of hyperparameters we want to optimize to the ***param_grid*** argument.

iv. The grid search will use a 3-fold cross-validation approach because the ***cv*** *argument* is set to ***3***.

Obtaining and utilizing optimal parameters

The Grid Search results in the identification of the ideal hyperparameter values, i.e., those that produced the best cross-validation performance:

i. The grid search's output, ***grid_result***, contains details on the best hyperparameter combination discovered.

ii. By using the variables ***grid_result.best_score*** and ***grid_result.best_params***, we can output the best cross-validation score as well as the accompanying hyperparameters.

Retraining the Model with Optimal Parameters

With the optimal hyperparameter values identified, the next point of call is to retrain the model using these identified parameters to ensure we achieve the most effective and generalizable model. This was done by:

i. We instantiate a new model using the CNN model function.

ii. We then train this model on our training data ***(X_train and y_train)*** using the optimal *epochs* and *batch_size* identified during Grid Search.

iii. An argument called ***validation_data*** was set for our validation datasets ***(X_val and y_val)***, allowing us to monitor the model's performance on an unseen dataset during the training process.

By using this meticulous and systematic methodology, we make sure that our models are trained with the best hyperparameters, achieving a compromise between the model's capacity to learn from the data and its ability to generalize effectively to new, unexplored data. This hyperparameter tuning step is crucial in the pursuit of achieving the highest possible performance in our emotion classification task. Thus, this implementation reflects a careful and empirical model optimisation strategy, greatly enhancing the validity and effectiveness of our final emotion classification model.

4.4 Results obtained from the models

We implemented emotion recognition on the four (4) facial expressions image datasets using the proposed new CNN model (implemented using TensorFlow and PyTorch), SANet model, ResNet – 50 model, and the VGG – 19 model. In this section, we will be presenting the results obtained from the four (4) deep learning models we implemented using the four (4) facial expressions image datasets. For each of the models, we will be presenting the model's accuracy plot, error rate plot, confusion matrix, and the AUC ROC curve plot. Importantly, we want to state that the results (accuracies) of the five (5) models presented below are the best obtained during the experiments. In general, the results (accuracies) are below the ones presented by a margin of 5% - 10%. The results are therefore presented below:

4.4.1 The proposed CNN Models

For the proposed CNN model, we implemented it using both TensorFlow and PyTorch deep learning framework. We will be presenting the results obtained from the two deep learning frameworks. The results obtained using the TensorFlow framework are presented below:

Model's Accuracy Plots – TensorFlow

The plot below shows the accuracy of the proposed model vs the epoch obtained using the TensorFlow deep learning framework.

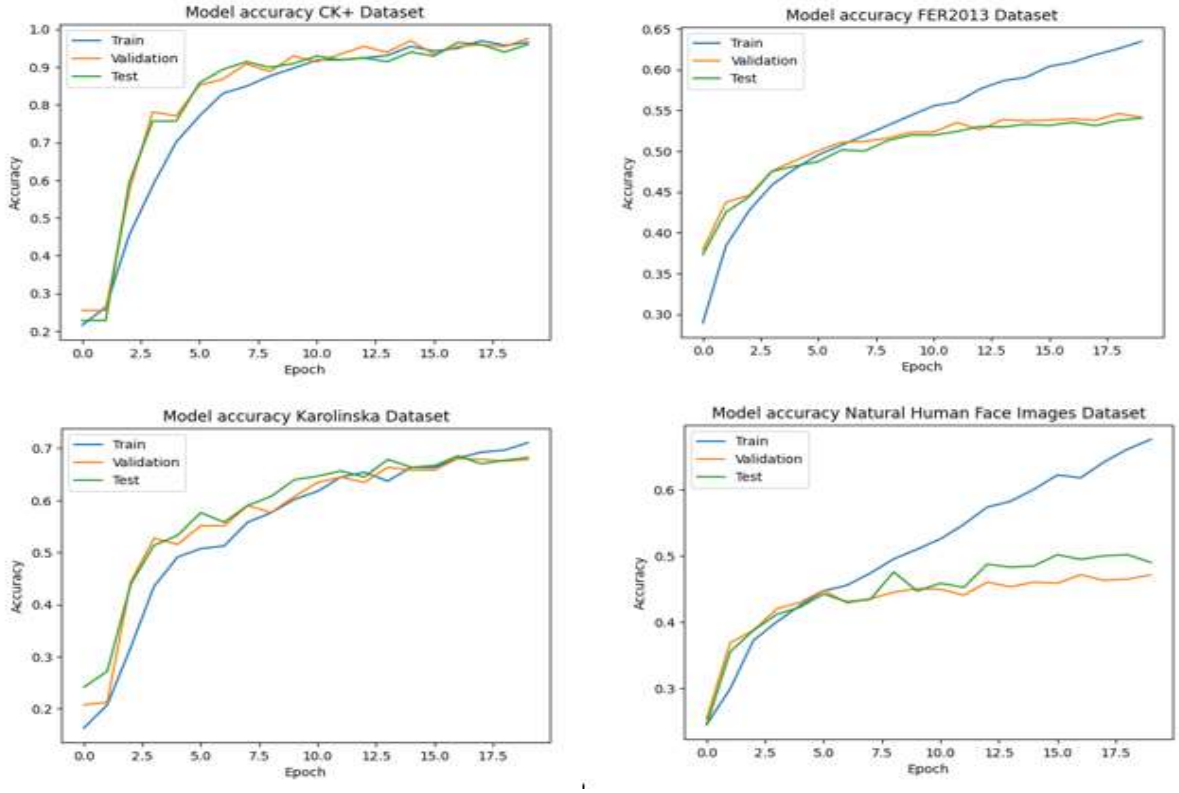

Figure 22: Accuracy plot using TensorFlow Model

Figure 22 above shows the model's accuracy plot of the proposed CNN model using the TensorFlow framework on the four (4) selected datasets. Using the CK+ dataset, the model recorded a training accuracy score of +/-96.4% , validation accuracy score of +/-97.4%, and test accuracy score of +/-95%; the FER2013 had an accuracy score of +/-63%, validation accuracy score of +/-54%, and test accuracy score of +/-54%. Also, the Karolinska dataset recorded an accuracy score of +/-70%, validation accuracy score of +/-68%, and test accuracy score of +/-68%, and the Natural Human Face Image dataset have an accuracy of +/-69%, validation accuracy score of +/-46%, and test accuracy score of +/-50% respectively.

Error Rate vs Epoch Plots – TensorFlow

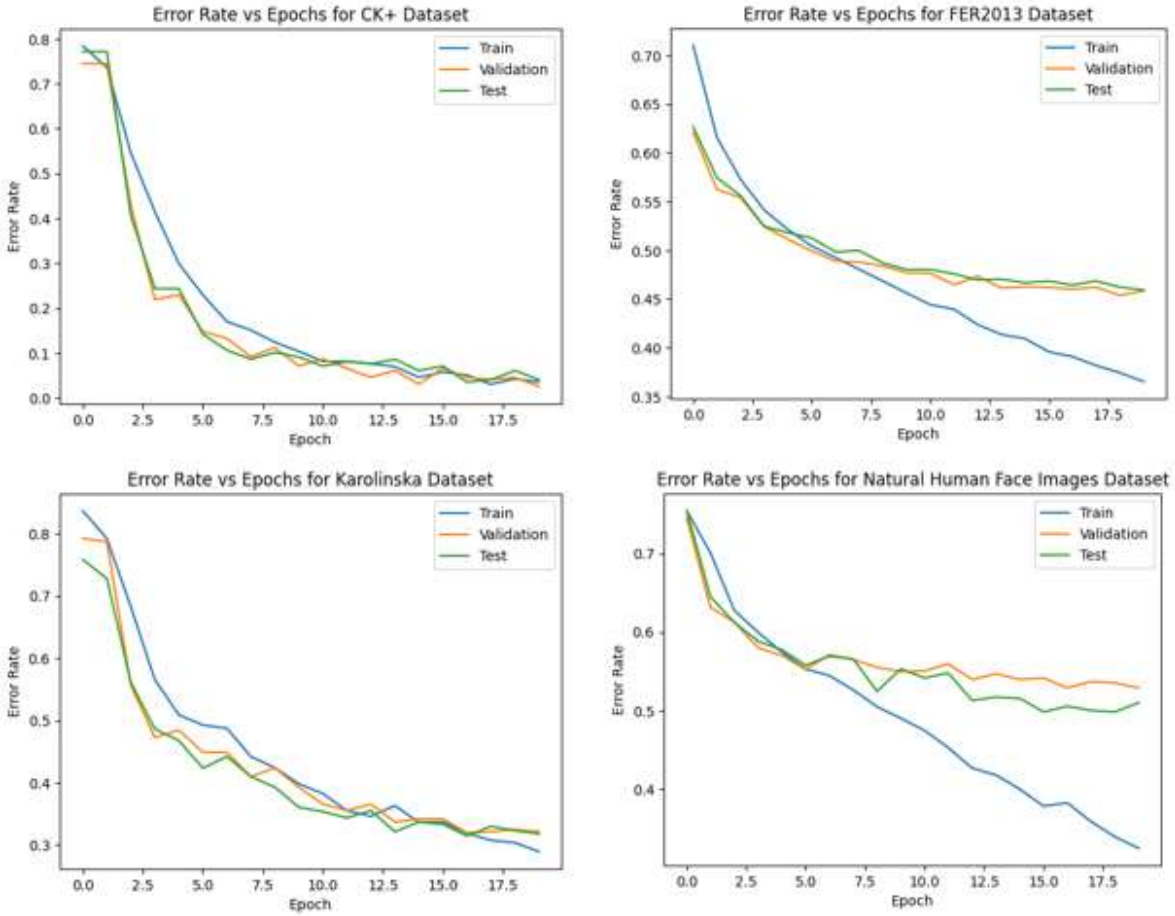

Figure 23: Error Rate vs Epoch Plot – TensorFlow Model

Figure 23 above shows the error rate plots obtained using the TensorFlow model on the four datasets. It can be observed that accuracy, validation, and testing error rate for the CK+ and the Karolinska datasets are within a close range whereas, the training error of both FER2013 and the Natural Human Face Image dataset differs a with that of the validation and testing set.

The confusion matrices – TensorFlow

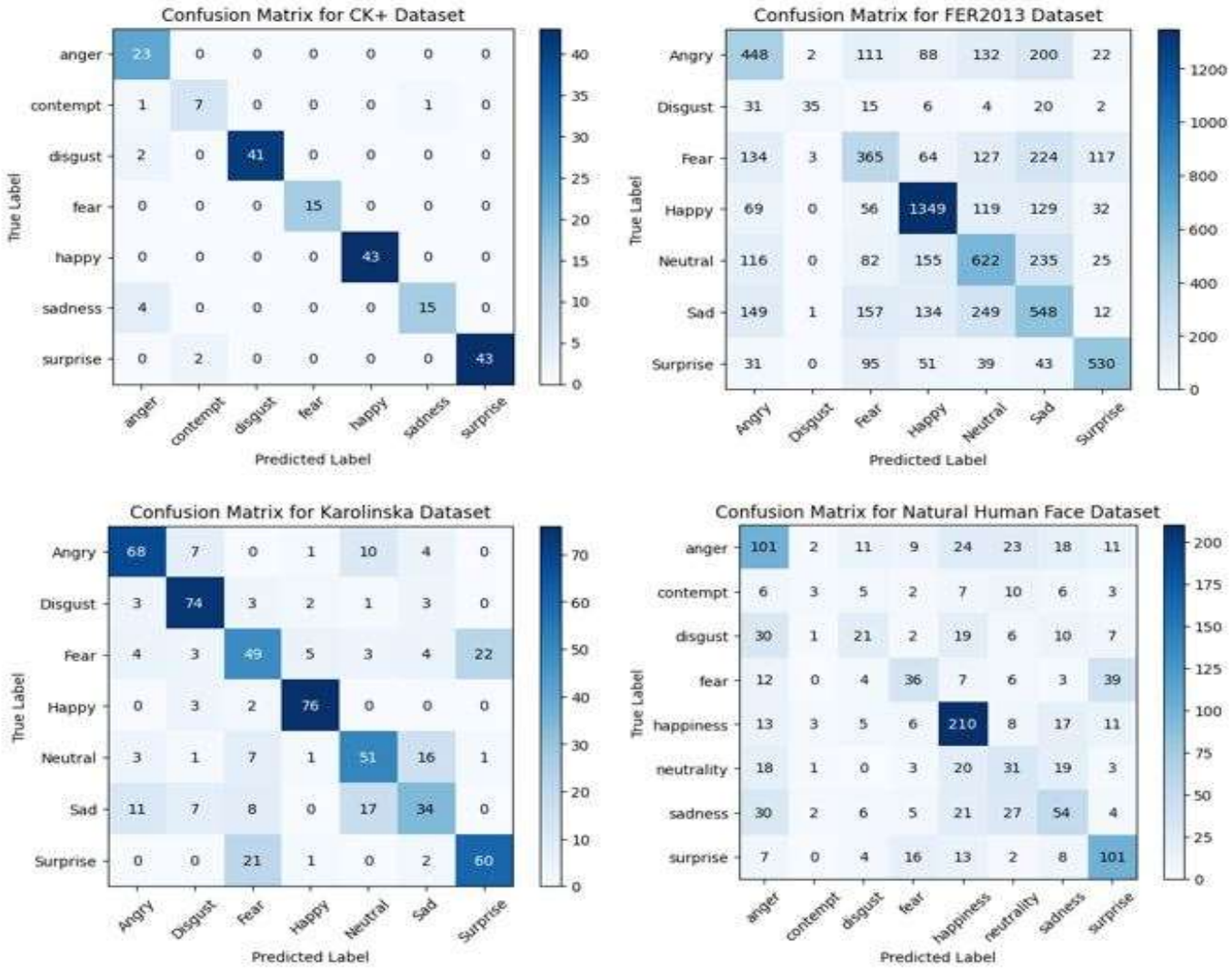

Figure 24: Confusion Matrix (TensorFlow) - for the selected datasets

Figure 24 presents the confusion matrices for the four datasets obtained using the proposed TensorFlow model. It shows the performance of our proposed TensorFlow model across different emotion classes. It can be seen from the matrix that the model shows strength in recognizing the ‘happy’, ‘fear’, and the ‘anger’ emotions as indicated by high values along the corresponding diagonal element, while it struggles with ‘sadness’ emotion, as indicated by lower values and the misclassified values. A near perfect classification was obtained using the CK+ dataset and followed by the Karolinska dataset. Whereas the model obtained a fair prediction on the FER2013 and Natural Human Face image datasets. Also, the model appears to frequently confuse 'Anger' with 'sadness', which suggests that distinguishing between these two emotions is a challenging aspect of our model’s current configuration.

ROC Curve Plots – TensorFlow Model

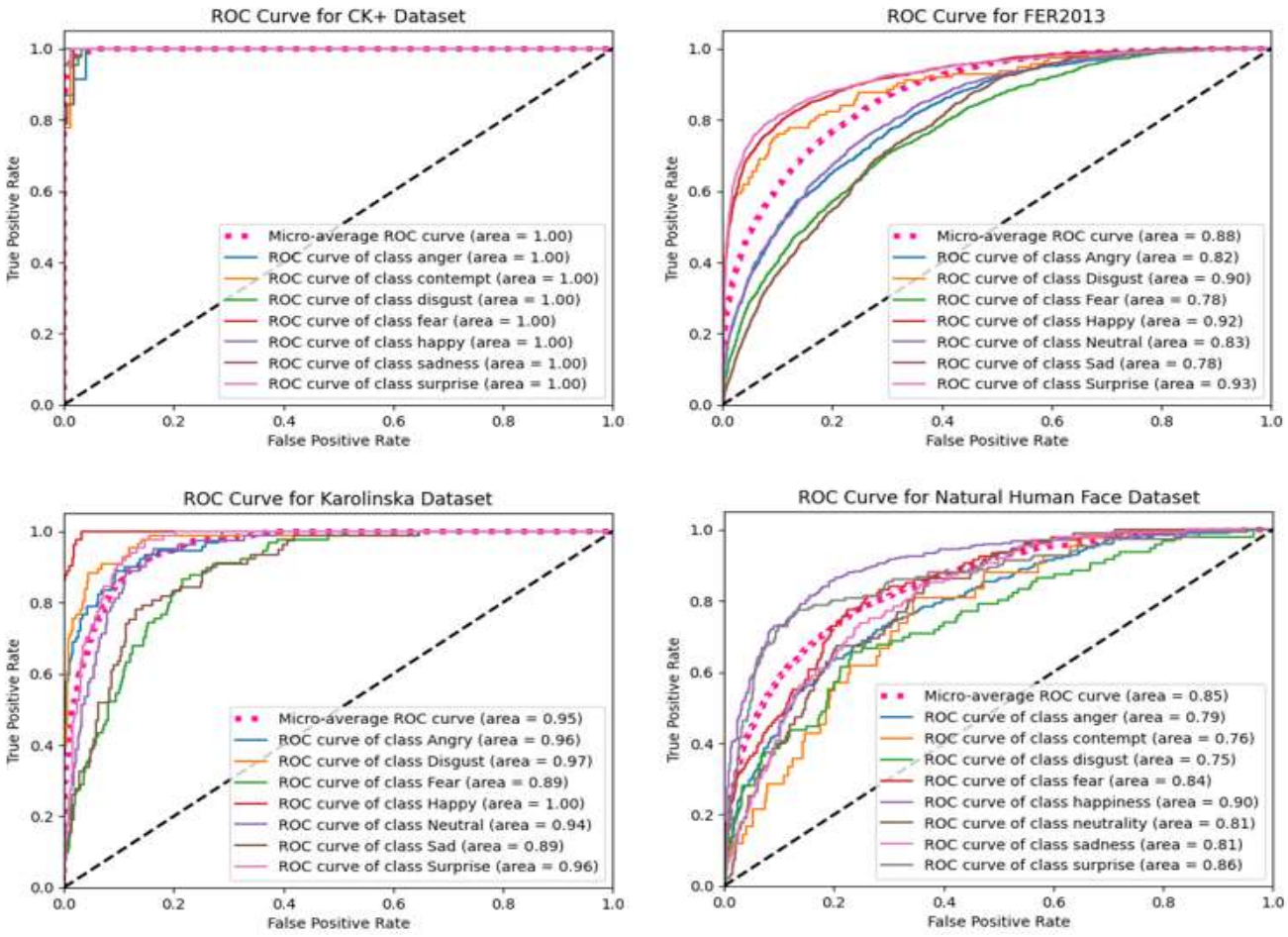

Figure 25: ROC Curve Plot (TensorFlow) – For the four (4) Datasets

From Figure 25, the model shows exceptional performance in recognizing the emotions, achieving a perfect micro-average AUC-ROC score of +/-0.98 for each emotion class on the CK+ image dataset. The FER2013 dataset recorded a micro-average AUC-ROC score of +/-0.88, Karolinska dataset recorded a micro-average AUC-ROC score of +/-0.95, and the Natural Human Face Image dataset obtained a micro-average AUC-ROC score of +/-0.85 respectively. This indicates an extraordinary ability to discriminate between positive and negative cases for varied emotional states, even when they are similar or overlapping. These AUC ROC scores indicate a good performance as recorded by the TensorFlow model.

Comparative analysis – TensorFlow Model

Table 1: Comparative Analysis showing results obtained using the TensorFlow Model

Proposed Model	Dataset	Accuracy	Precision	Recall	F1 Score	AUC ROC
Proposed CNN Model using TensorFlow	CK+	+/- 95%	+/- 96%	+/- 95%	+/- 95%	+/- 0.98
	FER2013	+/- 54%	+/- 55%	+/- 54%	+/- 54%	+/- 0.88
	Karolinska	+/- 70%	+/- 69%	+/- 70%	+/- 70%	+/- 0.95
	NHFI	+/- 50%	+/- 48%	+/- 50%	+/- 49%	+/- 0.85

Table 1 above shows the accuracy, precision, recall, F1 score, and the AUC ROC obtained using the TensorFlow model on the four selected datasets. It can be observed that the CK+ recorded the best score followed by the Karolinska dataset, FER2013 dataset and the Natural Human Face Images dataset recorded the least performance score. Figure 26 shows a plot of these metrics scores obtained using the TensorFlow model on these four (4) datasets.

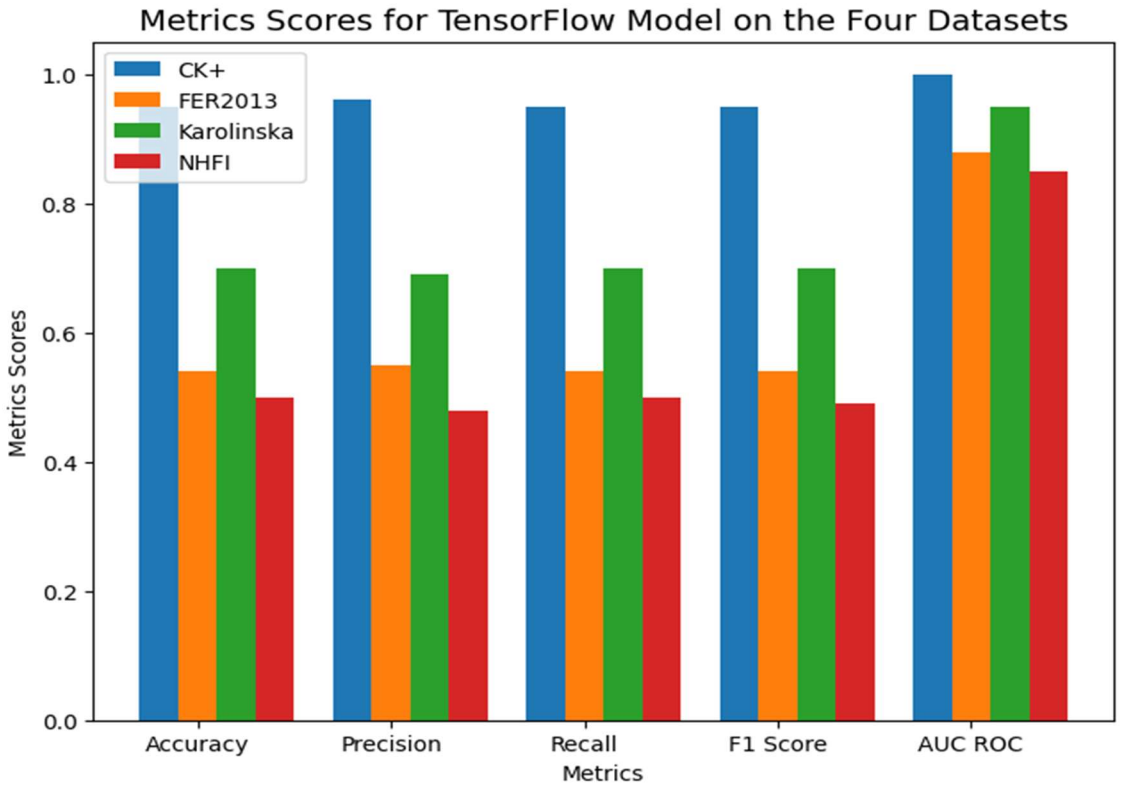

Figure 26: Plot of metrics scores - TensorFlow Model

The results obtained using the model developed using PyTorch framework on the four (4) datasets are given in Figure 27.

Model's Accuracy Plots – PyTorch Model

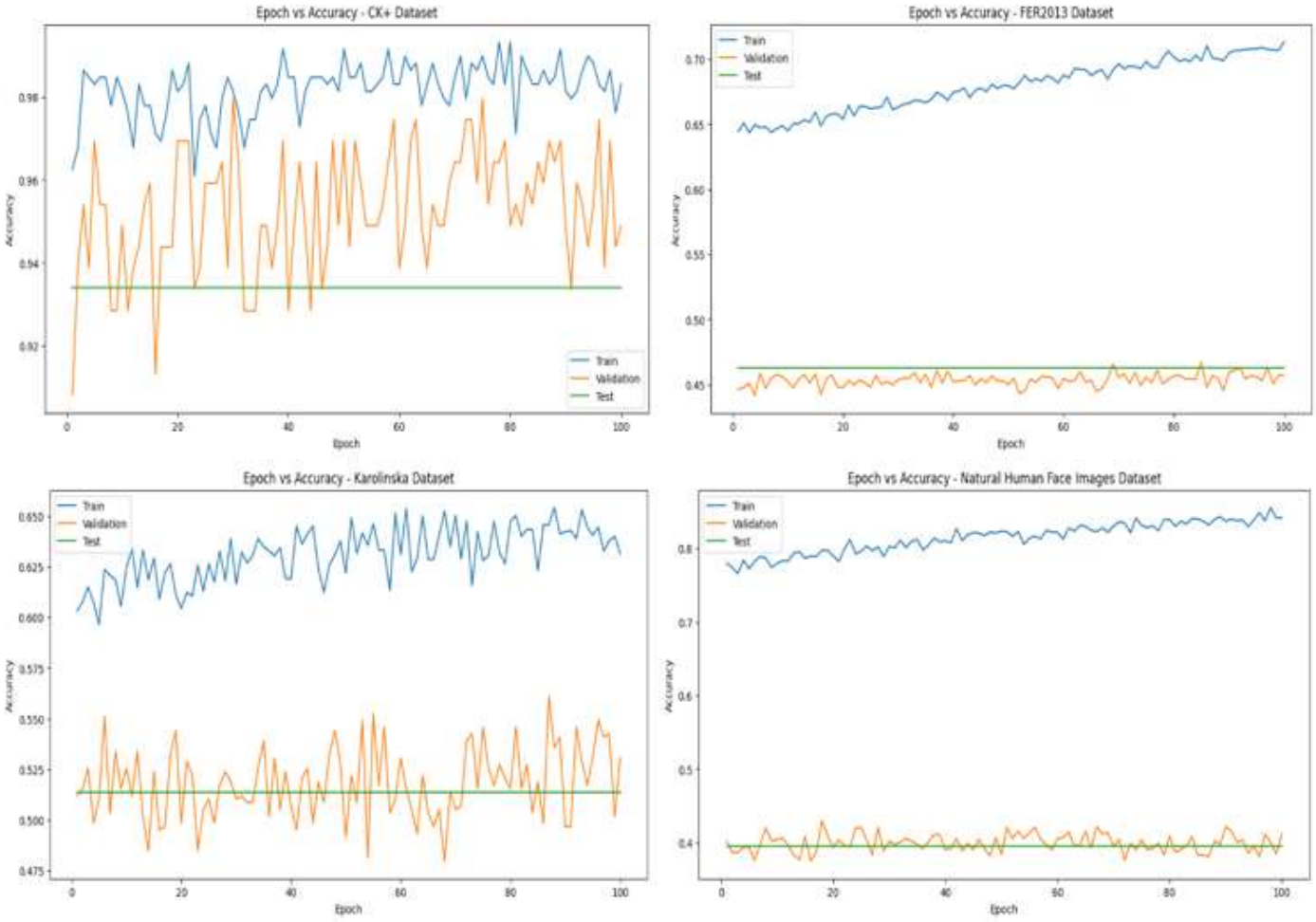

Figure 27: Accuracy Plots for PyTorch Model

Figure 27 shows the model's accuracy plots of the proposed CNN model using the PyTorch framework on the four (4) selected datasets. The CK+ dataset recorded a training accuracy score of +/-98% , validation accuracy score of +/-95.4%, and test accuracy score of +/-93.6%; the FER2013 had an accuracy score of +/-71%, validation accuracy score of +/-45%, and test accuracy score of +/-46%. Also, the Karolinska dataset recorded an accuracy score of +/-63%, validation accuracy score of +/-53%, and test accuracy score of +/-52%, and the Natural Human Face Image dataset have an accuracy of +/-80%, validation accuracy score of +/-42%, and test accuracy score of +/-40% respectively. As it was observed with the TensorFlow model, the best performance of the model was recorded on the CK+ dataset while the worst was recorded on the Natural Human Face Images dataset.

Error Rate vs Epoch Plot – PyTorch

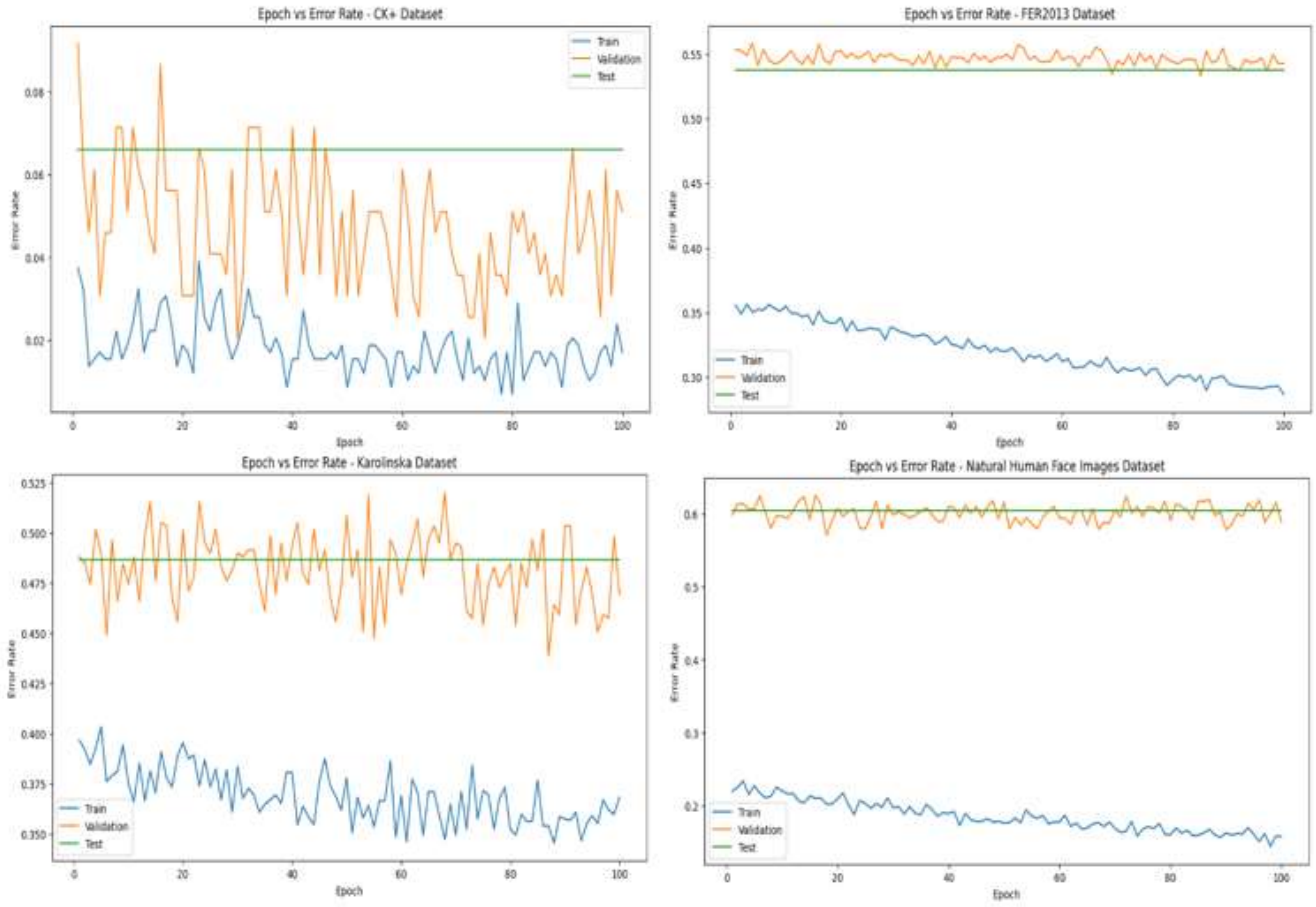

Figure 28: Error Rate using PyTorch Model

Figure 28 shows the error rate plots obtained using the PyTorch model on the four datasets. It can be observed that accuracy, validation, and testing error rate for the CK+ dataset are within a close range of 0.2 to 0.6, the FER2013 dataset recorded the accuracy error rate of +/-0.27, the validation error rate is +/-0.55, and the test error rate is +/-0.54. The Karolinska datasets accuracy, validation, and testing error rate is +/-0.37, +/-0.45, and +/-0.48 respectively, whereas, the training error of the Natural Human Face Image dataset +/-0.2, validation error of +/-0.38 and the test error is +/-0.4.

Confusion Matrix Plot – PyTorch

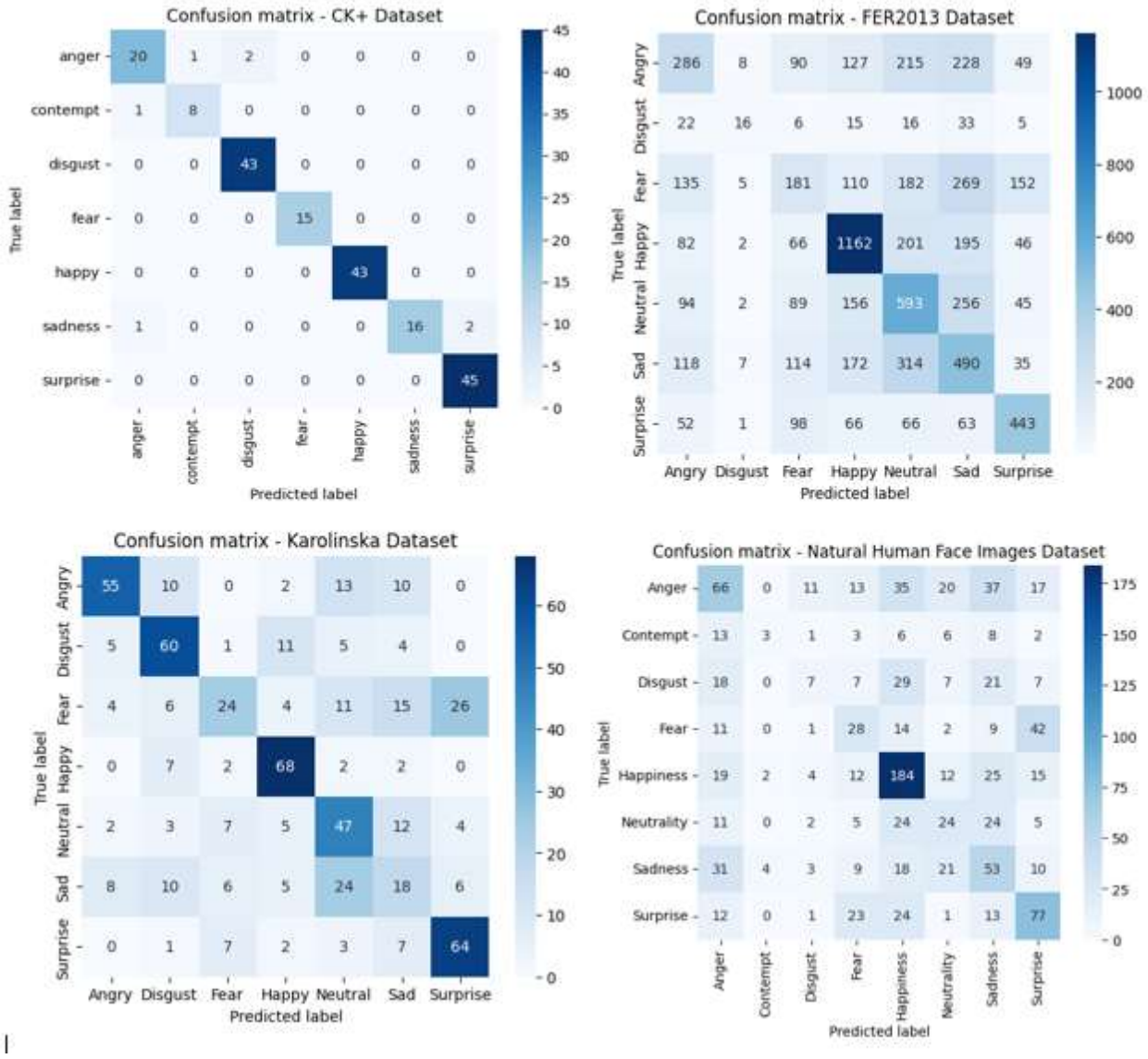

Figure 29: Confusion Matrices for PyTorch Model

Figure 29 above shows the confusion matrices for the four (4) selected datasets. It shows the performance of the PyTorch model in classifying the different emotion classes. It can be seen from the matrix that the model shows strength great strength on the CK+, while the model struggles with recognizing some of the emotion classes from the other three emotional classes: FER2013 dataset, the Karolinska dataset and Natural Human Face image datasets. A near perfect classification was obtained using the CK+ dataset and performed fairly on the other datasets.

ROC Curve Plot – PyTorch

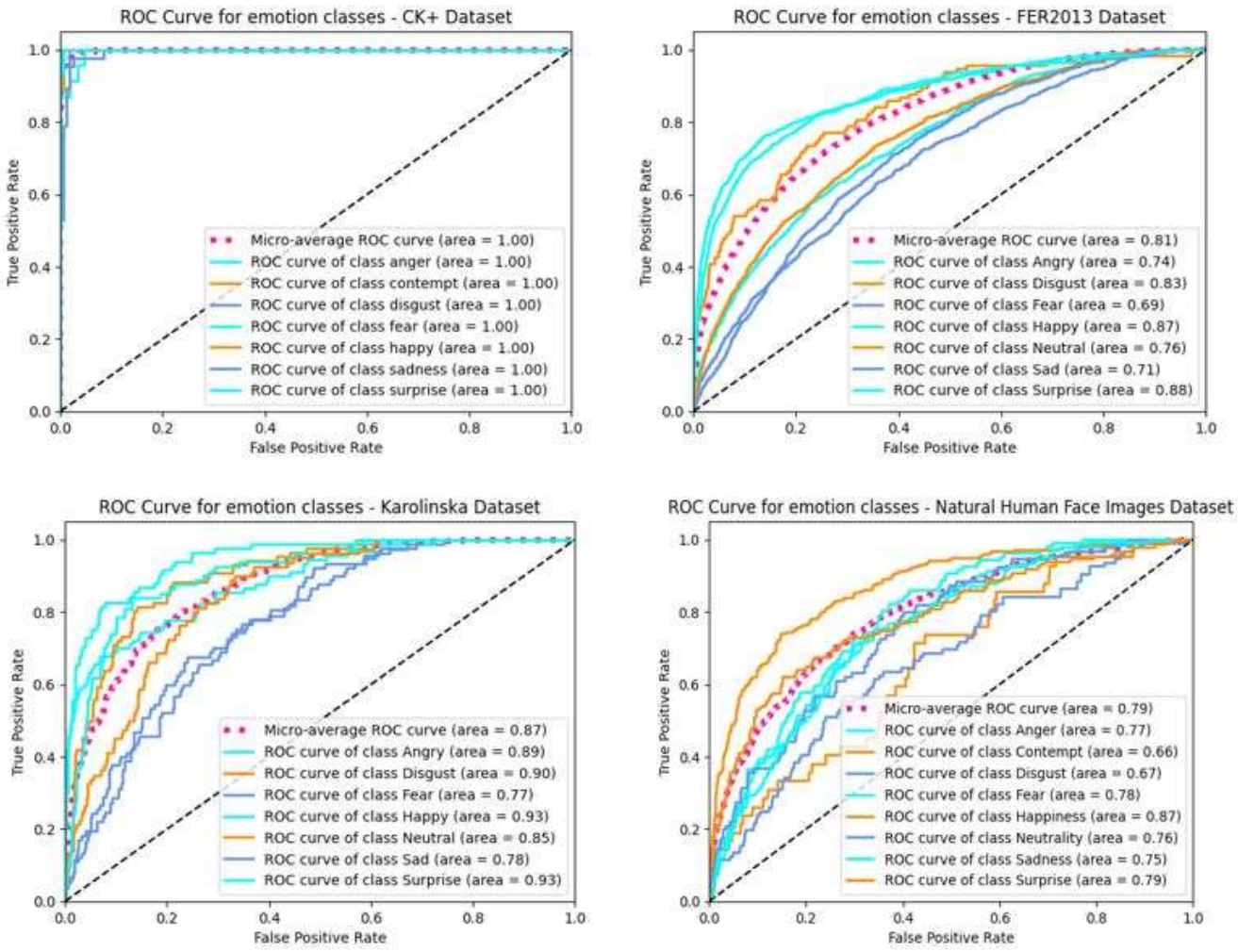

Figure 30: ROC Curve Plot for PyTorch Model

Figure 30 shows the exceptional performance in recognizing the emotions, achieving a perfect micro-average AUC-ROC score of +/- 0.98 for each emotion class on the CK+ image dataset. The FER2013 dataset recorded a micro-average AUC-ROC score of +/- 0.81, Karolinska dataset recorded a micro-average AUC-ROC score of +/- 0.87, and the Natural Human Face Image dataset obtained a micro-average AUC-ROC score of +/- 0.79 respectively. This indicates an extraordinary ability to discriminate between positive and negative cases for varied emotional states, even when they are similar or overlapping. Notably, the lowest micro-average AUC-ROC score was recorded using the Natural Human Face Image dataset.

Comparison on PyTorch CNN Model

Table 2: Comparative Analysis showing results obtained using the PyTorch Model

Proposed Model	Dataset	Accuracy	Precision	Recall	F1 Score	AUC ROC
Proposed CNN Model using PyTorch	CK+	+/- 95%	+/- 94%	+/- 94%	+/- 94%	+/- 0.98
	FER2013	+/- 46%	+/- 45%	+/- 41%	+/- 42%	+/- 0.81
	Karolinska	+/- 55%	+/- 55%	+/- 55%	+/- 55%	+/- 0.87
	NHFI	+/- 42%	+/- 34%	+/- 34%	+/- 34%	+/- 0.79

Table 2 above shows the accuracy, precision, recall, F1 score, and the AUC ROC obtained using the PyTorch model on the four (4) selected datasets. It can be observed that the CK+ recorded the best score followed by the Karolinska dataset, FER2013 dataset and the Natural Human Face Images dataset recorded the least performance score. Figure 31 below shows a plot of these metrics scores obtained using the PyTorch model on these four (4) datasets.

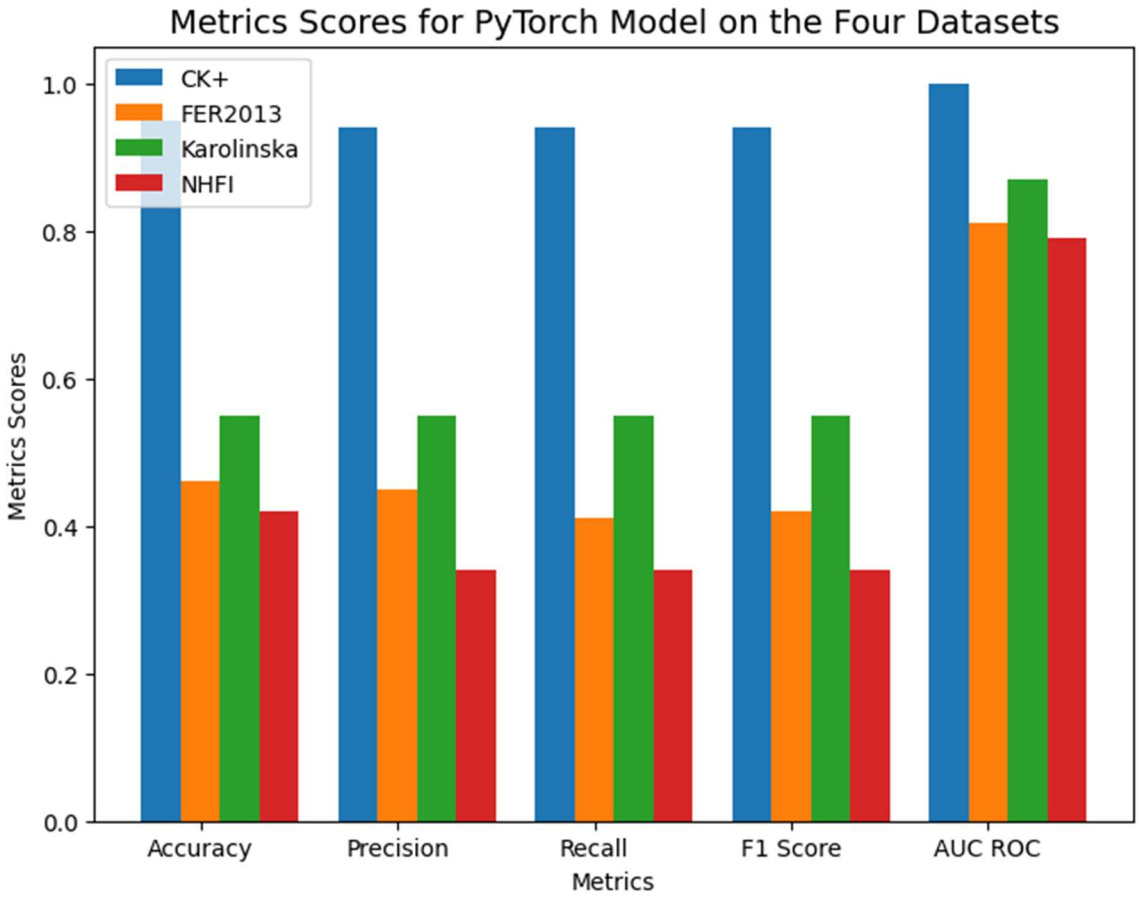

Figure 31: A Plot showing Metrics scores - PyTorch Model

For the proposed CNN model, we implemented during these experiments, we implemented it using both TensorFlow and PyTorch frameworks on the four (4) selected datasets. The results obtained from these frameworks proved that CK+ dataset recorded the best performance using both the TensorFlow and the PyTorch framework. Since we aimed to make a comparative analysis of the performance of these two frameworks, we shall be comparing the best results that were obtained using the CK+ dataset. The metrics we are comparing between these two deep learning Python frameworks are accuracy, precision, recall, F1 score, and AUC ROC. Figure 32 below is a grouped bar chart that shows the best performance of both the TensorFlow and the PyTorch frameworks.

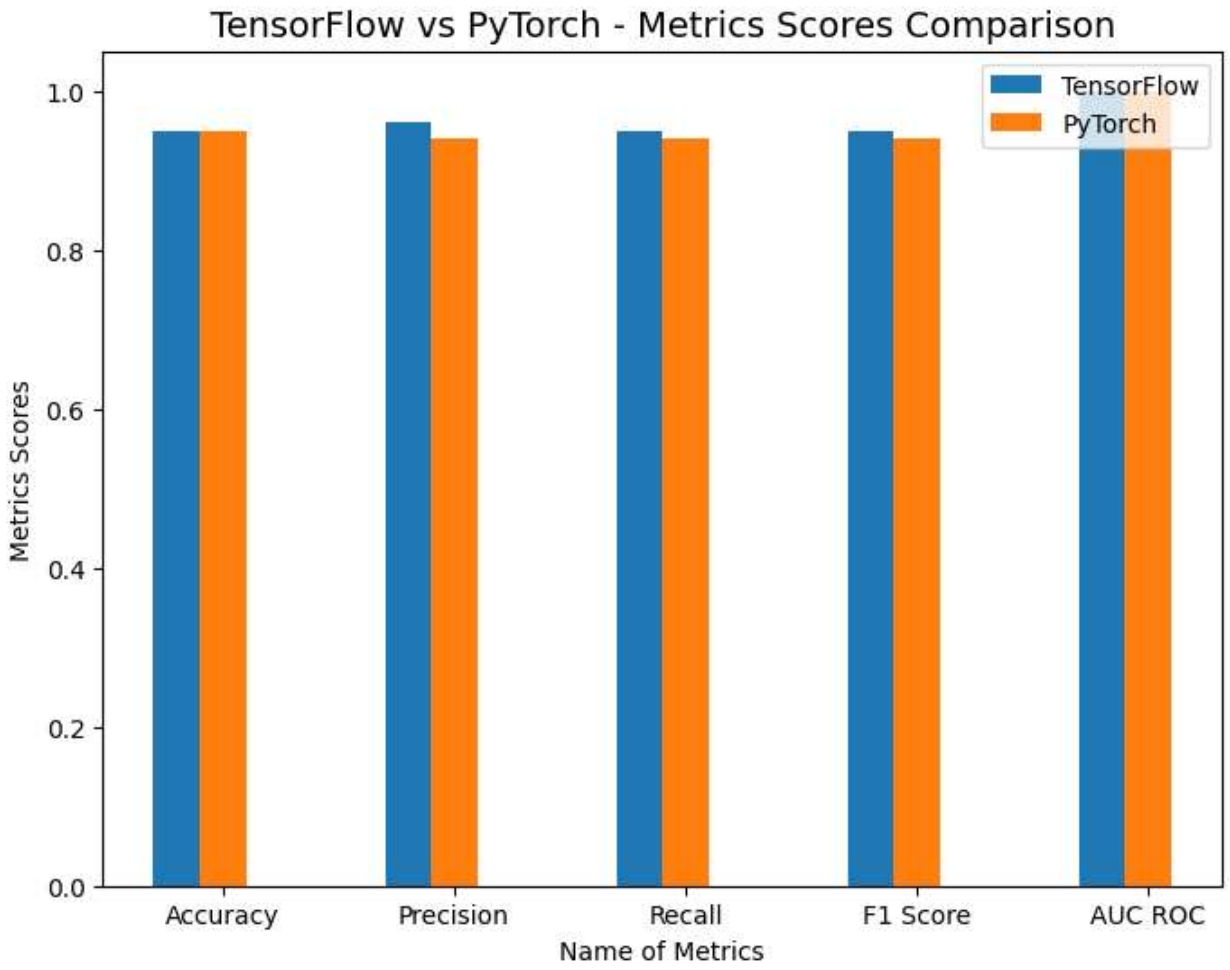

Figure 32: TensorFlow vs PyTorch Metric Comparison

Figure 32 shows both the TensorFlow and PyTorch recorded a +/- 95% accuracy of classification and an AUC ROC of +/- 0.98 each. TensorFlow obtained a precision percentage score of +/- 96%, recall score of +/-95%, and F1 score +/- 95% whereas the PyTorch model recorded a precision percentage score of +/- 94%, recall score of +/-94%, and F1 score +/- 94% respectively. This shows a slight better performance by the TensorFlow model as against the PyTorch model.

4.4.2 Results for SANet Model

As stated above, the experiment was also conducted using the SANet model. The results obtained using the SANet model are presented below:

Model's Accuracy (Accuracy vs Epoch) Plot – SANet

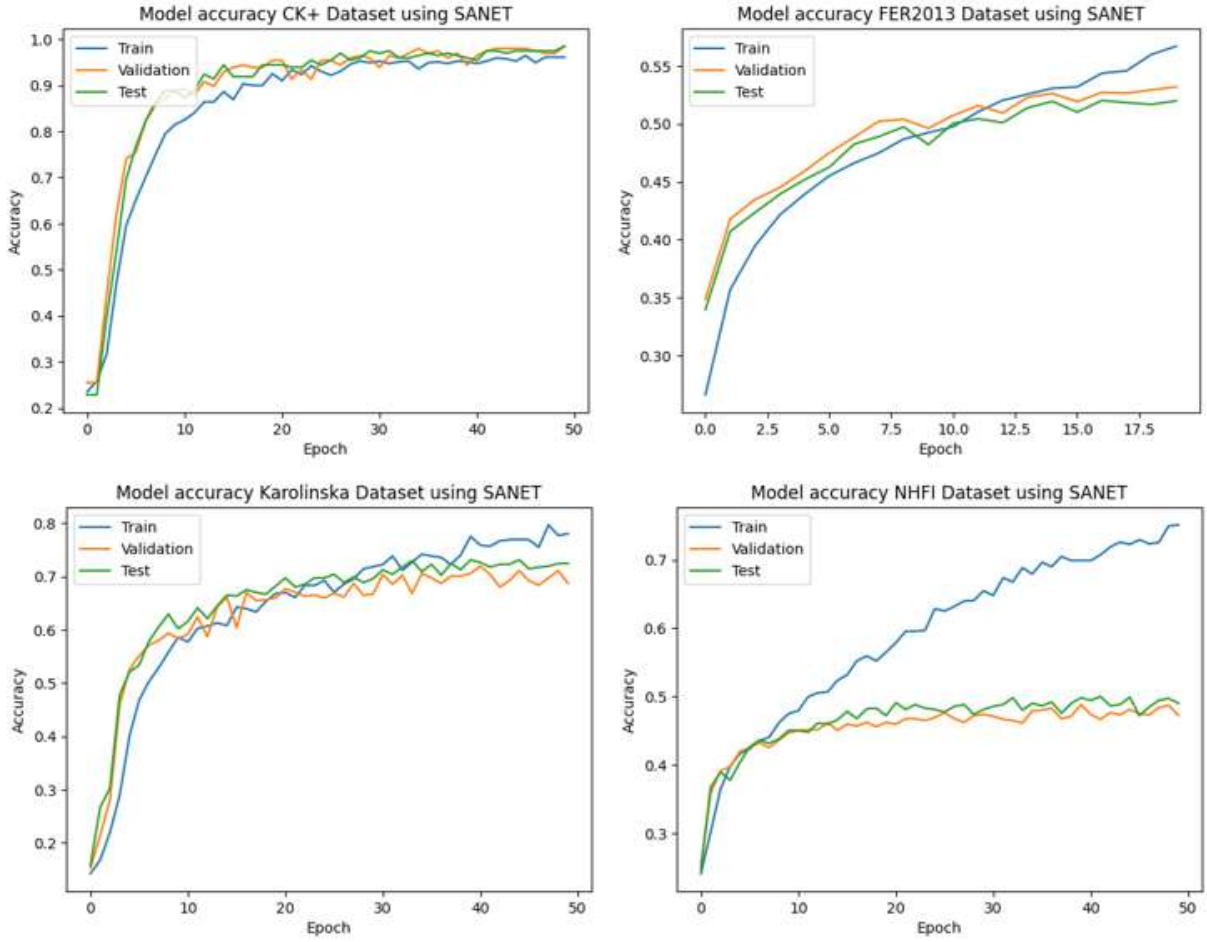

Figure 33: Accuracy Plots for SANet Model

Figure 33 shows the model's accuracy plots of the SANet model on the four selected datasets. The CK+ dataset recorded a training accuracy score of +/- 98% , validation accuracy score of +/- 97%, and test accuracy score of +/- 98%; the FER2013 had an accuracy score of +/- 57%, validation accuracy score of +/- 53%, and test accuracy score of +/- 52%. Also, the Karolinska dataset recorded an accuracy score of +/- 76%, validation accuracy score of +/- 72%, and test accuracy score of +/- 74%, and the Natural Human Face Image dataset have an accuracy of +/- 75%, validation accuracy score of +/- 47%, and test accuracy score of +/- 49% respectively. As it was observed with the TensorFlow model, the best performance of the model was recorded on the CK+ dataset while the worst was recorded on the Natural Human Face Images dataset.

Error Rate vs Epoch Plot – SANet Model

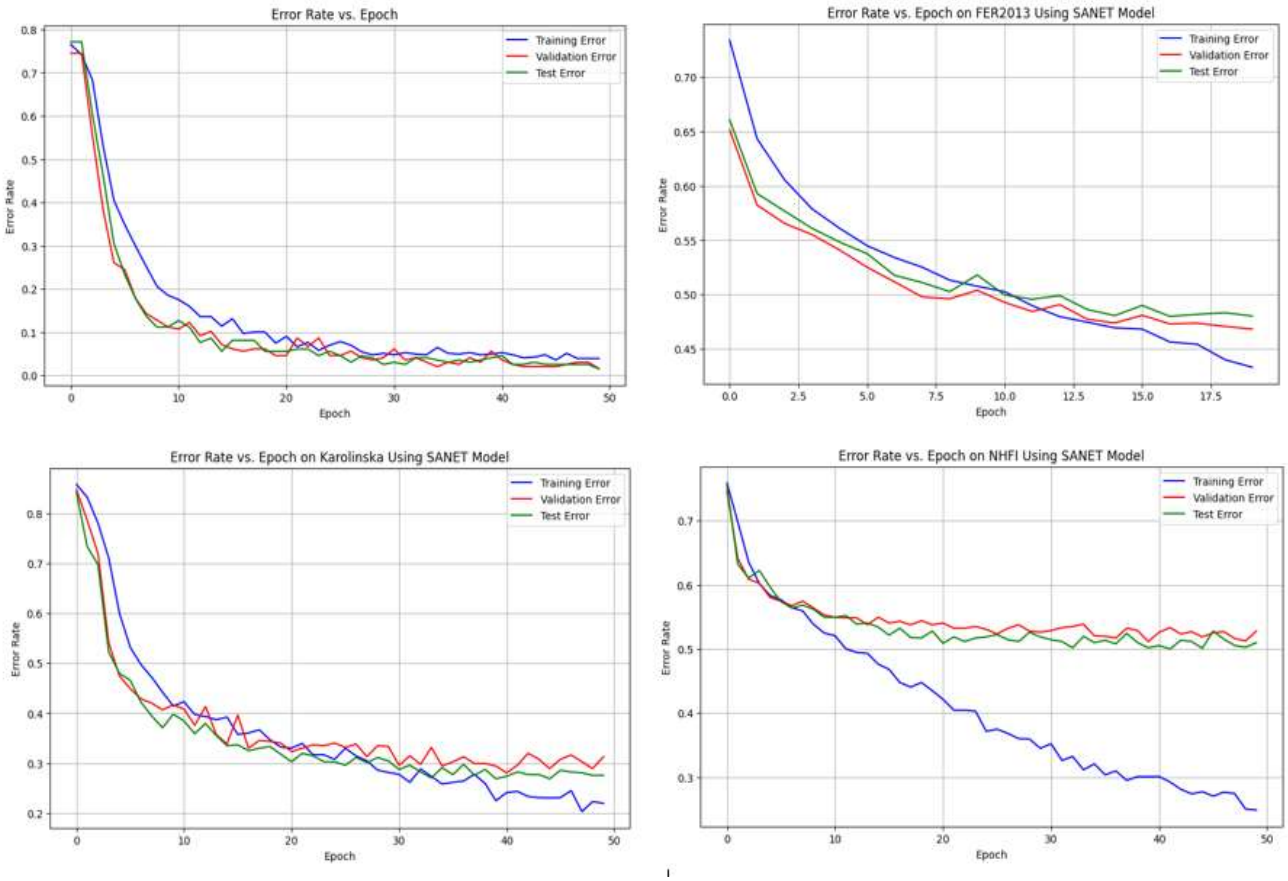

Figure 34: Error rate plots - SANet Model

Figure 34 above shows the error rate plots obtained using the SANet model on the four datasets. It can be observed that accuracy, validation, and testing error rate for the CK+ dataset are within a close range of 0.2 to 0.3, the FER2013 dataset at 50 epochs recorded the accuracy error rate of +/- 0.4, the validation error rate is +/- 0.47, and the test error rate is +/- 0.48. The Karolinska datasets accuracy, validation, and testing error rates are +/- 0.22, +/- 0.29, and +/- 0.31 respectively, whereas the training error of the Natural Human Face Image dataset +/- 0.23, validation error of +/- 0.52 and the test error is +/- 0.51. The plot also shows that the more the number of epochs, the less the error rate across all the four (4) selected datasets.

Confusion Matrices – SANet Model

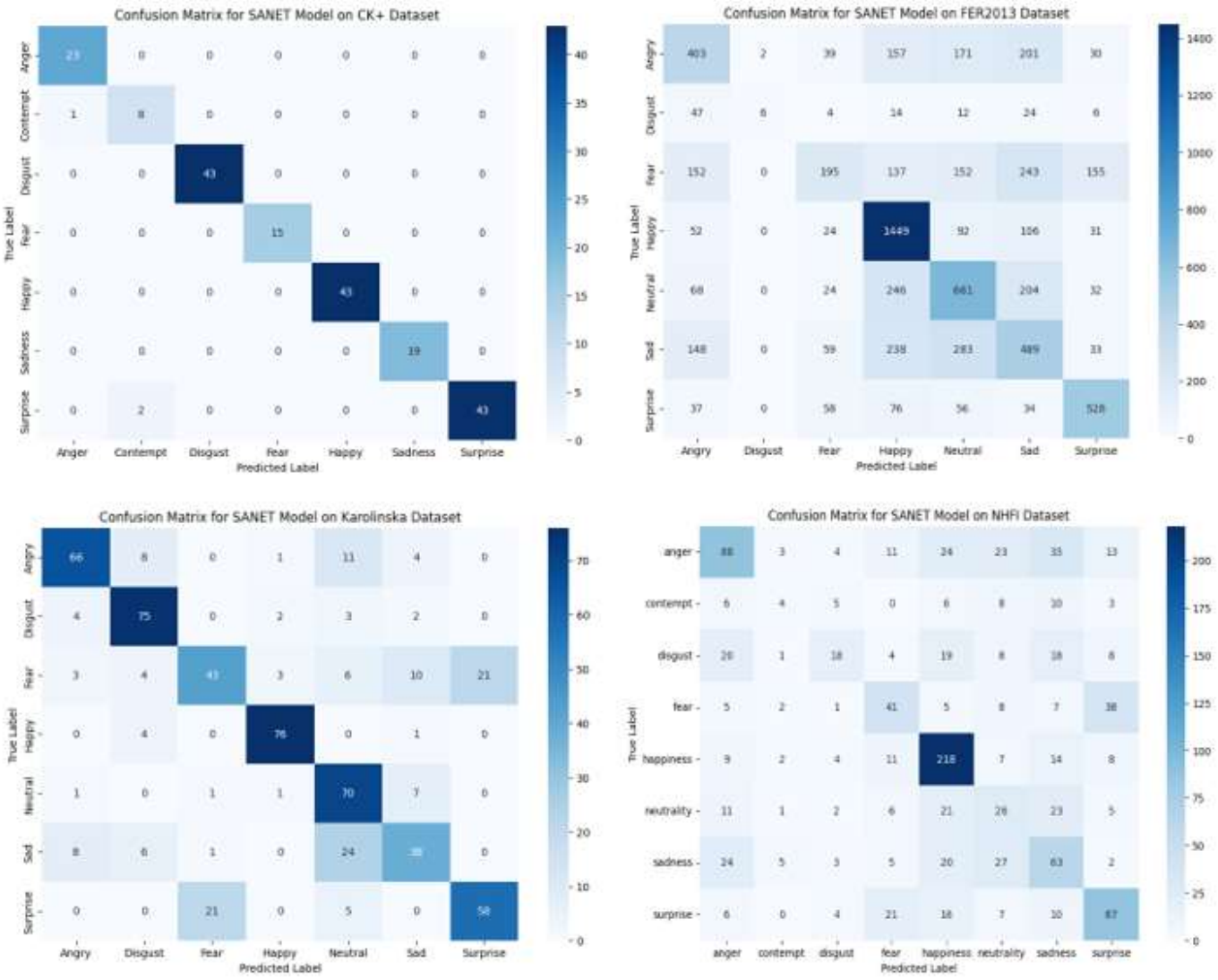

Figure 35: Confusion Matrices - SANet Model

Figure 35 above presents the confusion matrices for the four (4) datasets. It shows the performance of the SANet model on the emotion classes. It can be seen from the matrix that the model shows strength in recognizing the 'happy', and the 'anger' emotions as indicated the confusion matrix obtained using the CK+, while the model struggles with recognizing the 'disgust' emotion class from the FER2013 dataset. A near perfect classification was obtained using the CK+ dataset and performed fairly on the Karolinska dataset and Natural Human Face image datasets.

ROC Curve Plot

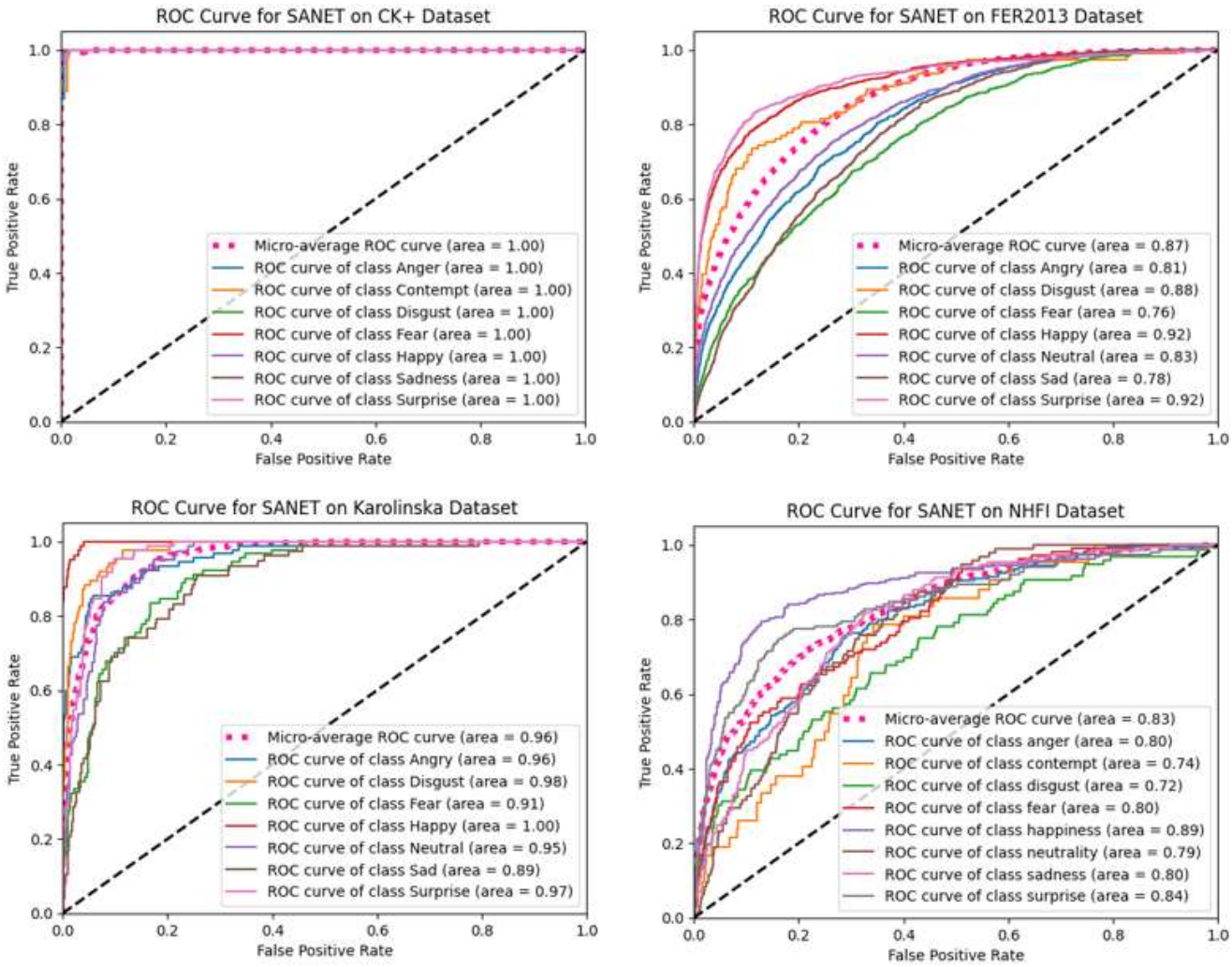

Figure 36: ROC Curve Plots - SANet Model

Figure 36 shows exceptional performance in recognizing the emotions, achieving a near perfect micro-average AUC-ROC score of +/- 0.98 for each emotion class on the CK+ image dataset. The FER2013 dataset recorded a micro-average AUC-ROC score of +/- 0.87, Karolinska dataset recorded a micro-average AUC-ROC score of +/- 0.96, and the Natural Human Face Image dataset obtained a micro-average AUC-ROC score of +/- 0.83 respectively. This indicates an extraordinary ability of the SANet model to discriminate between positive and negative cases for varied emotional classes, however how similar or overlapping they are. Notably, just as was observed with the other models, the lowest micro-average AUC-ROC score was recorded using the Natural Human Face Image dataset.

Comparison on SANet Model

Table 3: Comparative Analysis showing results obtained using the SANet Model

Model	Dataset	Accuracy	Precision	Recall	F1 Score	AUC ROC
SANet Model	CK+	+/- 98%	+/- 97%	+/- 98%	+/- 97%	+/- 0.98
	FER2013	+/- 54%	+/- 55%	+/- 49%	+/- 50%	+/- 0.87
	Karolinska	+/- 74%	+/- 74%	+/- 74%	+/- 74%	+/- 0.96
	NHFI	+/- 49%	+/- 41%	+/- 39%	+/- 39%	+/- 0.83

Table 3 above shows the accuracy, precision, recall, F1 score, and the AUC ROC obtained using the SANet model on the four selected datasets. It can be observed that the CK+ recorded the best score of +/- 98% accuracy, followed by the Karolinska dataset with +/- 74%, FER2013 dataset recorded +/- 54%, and the Natural Human Face Images dataset recorded the least accuracy performance score of +/- 49%. Figure 37 below shows a grouped bar chart plot of these metrics scores obtained using the SANet model on these four datasets.

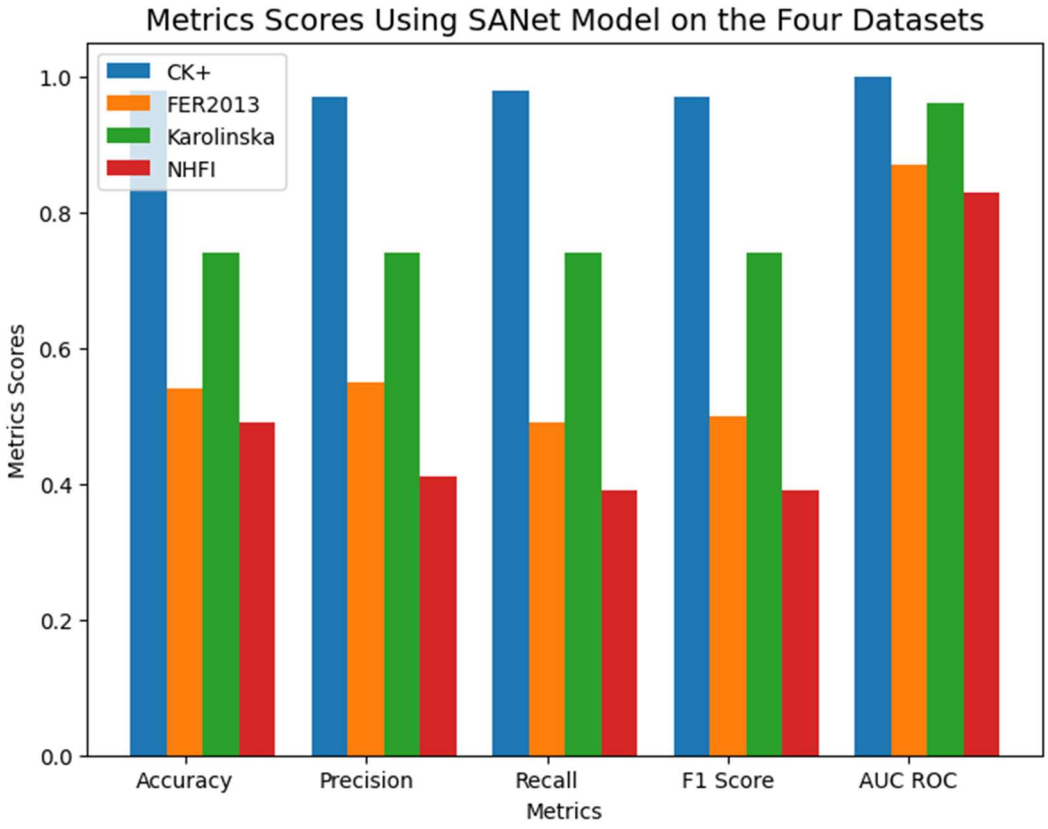

Figure 37: A Plot showing Metrics scores - SANet Model

4.4.3 VGG – 16 Model's Results

For the VGG–16 implementations, the results obtained from using the VGG-16 model is Presented on Figure 38.

Model's Accuracy (Accuracy vs Epoch) Plots: VGG-16 Model

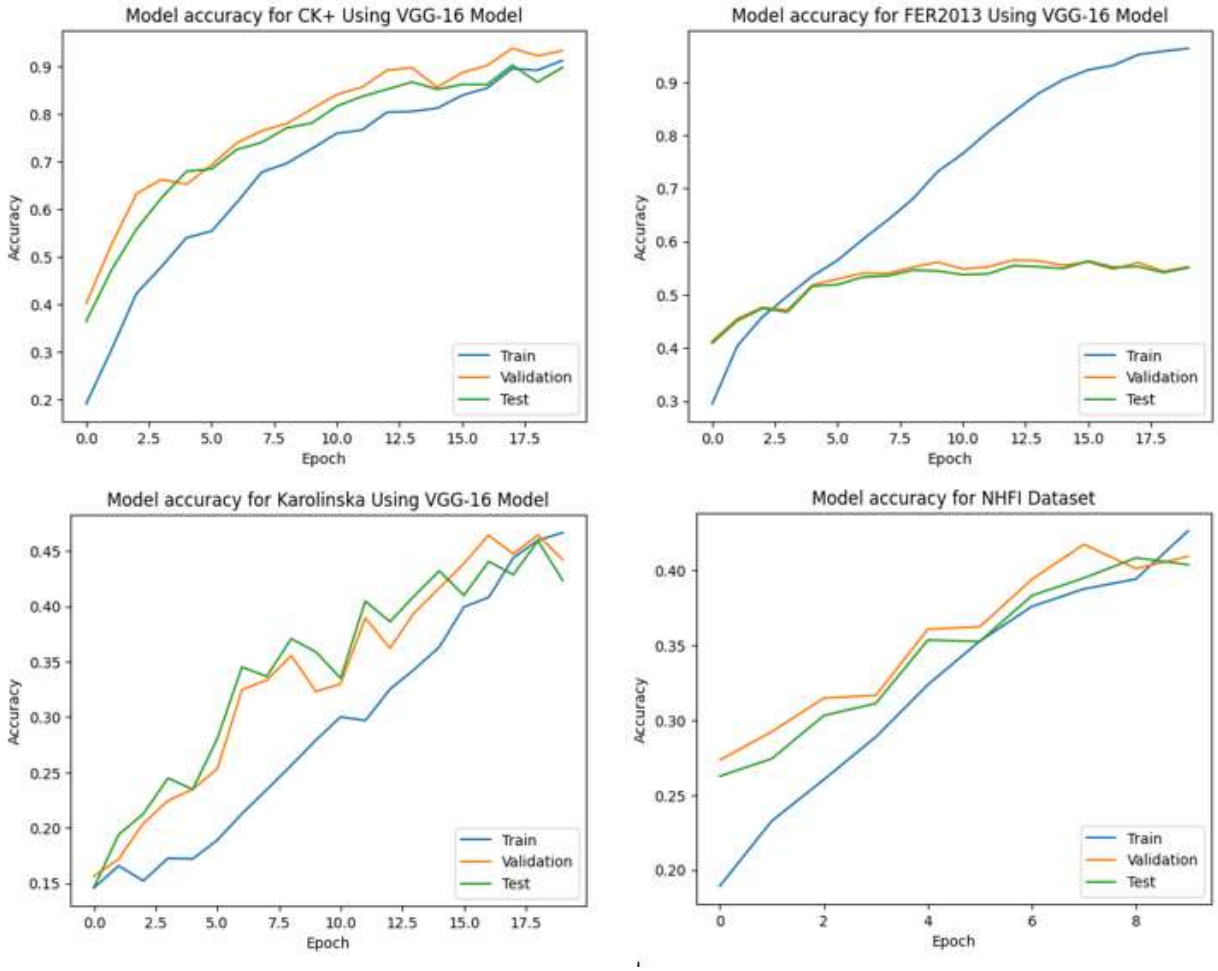

Figure 38: Accuracy Plots: VGG-16 Model

Figure 38 shows the model's accuracy plots of the VGG-16 model on the four (4) selected datasets. The CK+ dataset recorded a training accuracy score of +/- 90% , validation accuracy score of +/- 92%, and test accuracy score of +/- 91%; the FER2013 had an accuracy score of +/- 93%, validation accuracy score of +/- 53%, and test accuracy score of +/- 53%. Also, the Karolinska dataset recorded an accuracy score of +/- 47%, validation accuracy score of +/- 44%, and test accuracy score of +/- 43%, and the Natural Human Face Image dataset have an accuracy of +/- 44%, validation accuracy score of +/- 40%, and test accuracy score of +/- 39% respectively. As it was observed with the TensorFlow PyTorch, and the SANet models, the best performance of the model was recorded on the CK+ dataset while the worst was recorded on the Natural Human Face Images dataset.

Error Rate vs Epoch Plot: VGG-16 Model

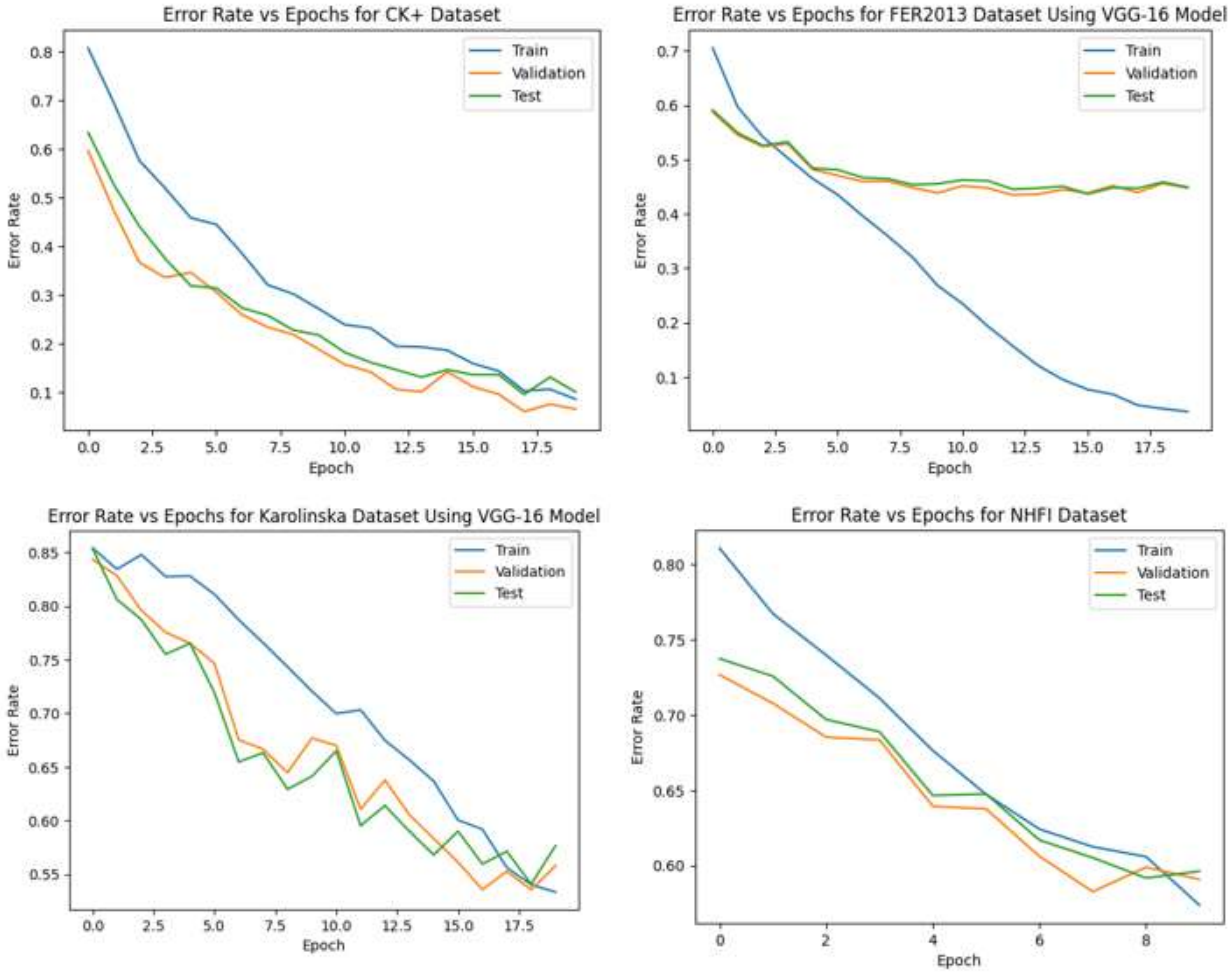

Figure 39: Error rate Plots: VGG-16 Model

Figure 39 shows the error rate plots obtained using the VGG-16 model on the four selected datasets. It can be observed that accuracy, validation, and testing error rate for the CK+ dataset are within a close range of +/- 0.1, the FER2013 dataset at 50 epochs recorded the accuracy error rate of +/- 0.07, the validation error rate is +/- 0.47, and the test error rate is +/- 0.47. The Karolinska datasets accuracy, validation, and testing error rates are +/- 0.53, +/- 0.56, and +/- 0.53 respectively, whereas the training error of the Natural Human Face Image dataset +/- 0.56, validation error of +/- 0.60 and the test error is +/- 0.61.

Confusion Matrix: VGG – 16 Model

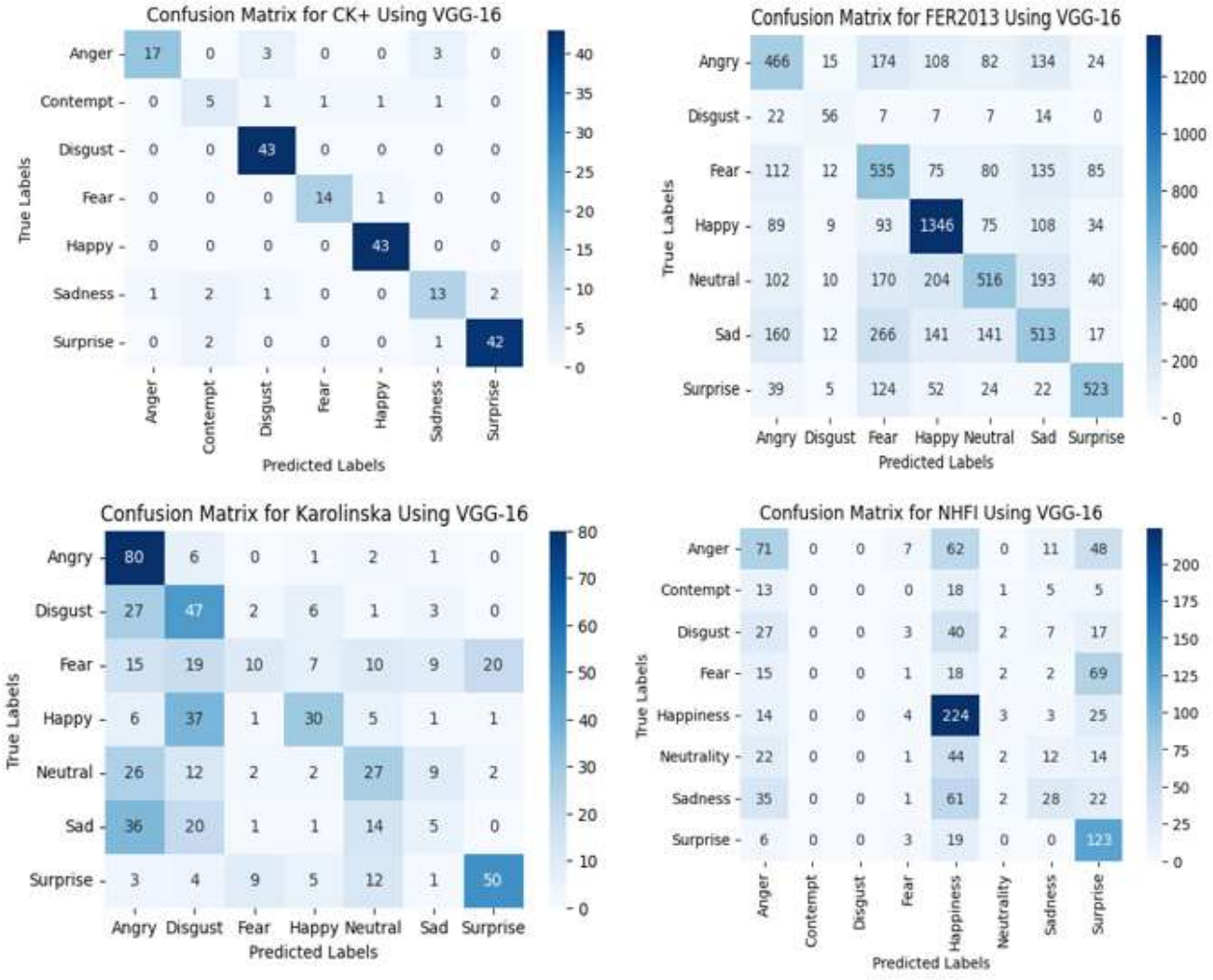

Figure 40: Confusion Matrix - VGG-16 Model

Figure 40 presents the confusion matrices for the four datasets. It shows the performance of the VGG-16 model in classifying the emotion classes. It can be seen from the matrices that the model shows great performance on the CK+ datasets. On the FER2013 dataset and Karolinska dataset recorded a fair performance in classifying the different emotion classes. Notably, the model struggles in identifying and classifying the 'contempt' and 'disgust' emotion classes from the Natural Human Face images dataset. A near perfect classification of about +/- 92% accuracy was obtained using the CK+ dataset and a fair performance on both the FER2013 dataset and the Karolinska dataset while the Natural Human Face image datasets recorded the lowest prediction accuracy.

ROC Curve Plot: VGG – 16 Model

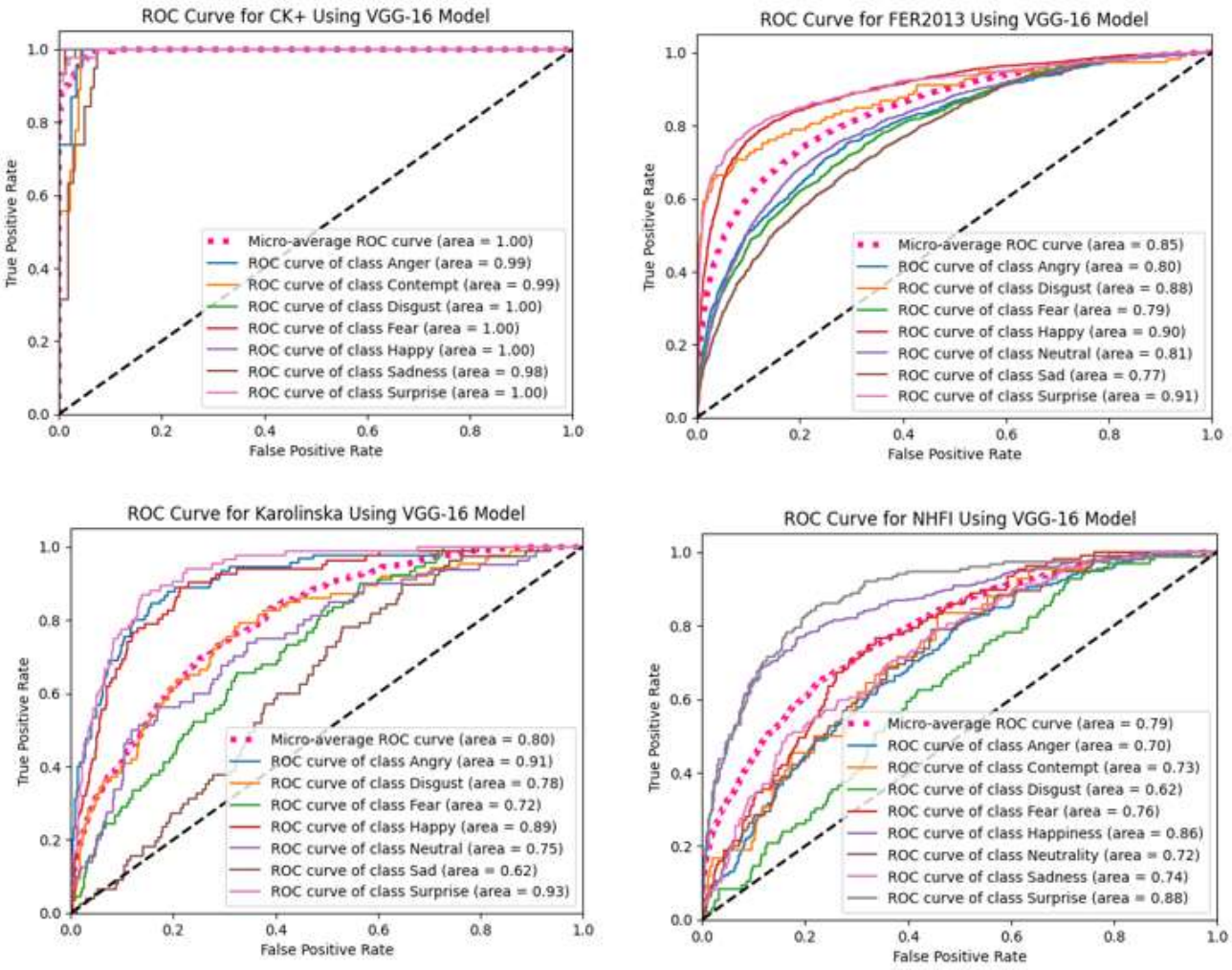

Figure 41: ROC Curve Plots: VGG-16 Model

From Figure 41 the model shows exceptional performance in recognizing the emotions, achieving a near perfect micro-average AUC-ROC score of +/- 0.98 for each emotion class on the CK+ image dataset. The FER2013 dataset recorded a micro-average AUC-ROC score of +/- 0.85, Karolinska dataset recorded a micro-average AUC-ROC score of +/- 0.80, and the Natural Human Face Image dataset obtained a micro-average AUC-ROC score of +/- 0.79 respectively. It shows that the VGG-16 model shows some fair performance in discriminating between positive and negative cases for the varied emotional classes, however how similar or overlapping they are. Notably, just as was observed with the other models, the lowest micro-average AUC-ROC score was recorded using the Natural Human Face Images dataset as was recorded using the other CNN models.

Comparison on VGG-16 Model

Table 4: Comparative Analysis showing results obtained using the VGG-16 model

Model	Dataset	Accuracy	Precision	Recall	F1 Score	AUC ROC
VGG-16 Model	CK+	+/- 90%	+/- 85%	+/- 84%	+/- 84%	+/- 0.98
	FER2013	+/- 55%	+/- 54%	+/- 53%	+/- 53%	+/- 0.85
	Karolinska	+/- 42%	+/- 42%	+/- 42%	+/- 38%	+/- 0.80
	NHFI	+/- 40%	+/- 23%	+/- 28%	+/- 22%	+/- 0.79

Table 4 shows the accuracy, precision, recall, F1 score, and the AUC ROC obtained using the VGG-16 model on the four (4) selected datasets. It can be observed that the CK+ recorded the best score of +/- 90% accuracy, followed by the FER2013 dataset with +/- 55%. The Karolinska dataset recorded +/- 42%, and the Natural Human Face Images dataset recorded the least accuracy performance score of +/- 40%. Figure 42 shows a grouped bar chart plot of these metrics scores obtained using the VGG-16 model on these four datasets.

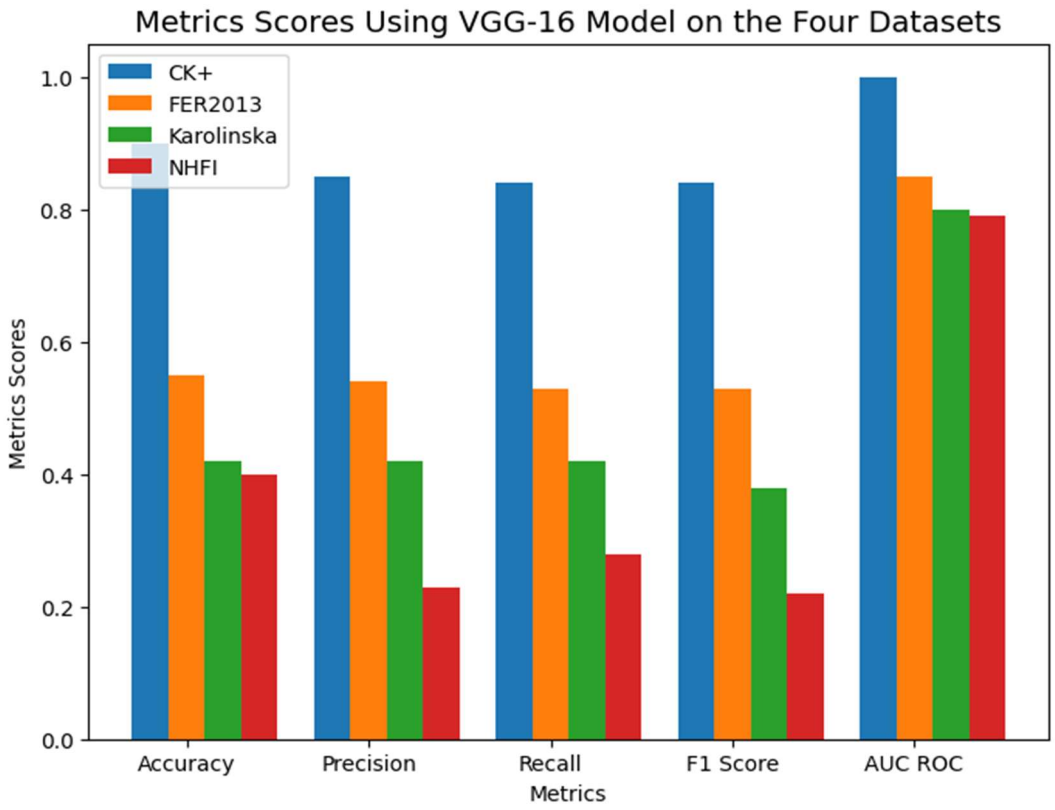

Figure 42: A Plot showing Metrics scores - VGG-16 model

4.4.4 ResNet50 model results

The results obtained using the ResNet50 model are hereby given below.

Accuracy Plot – ResNet50 Model

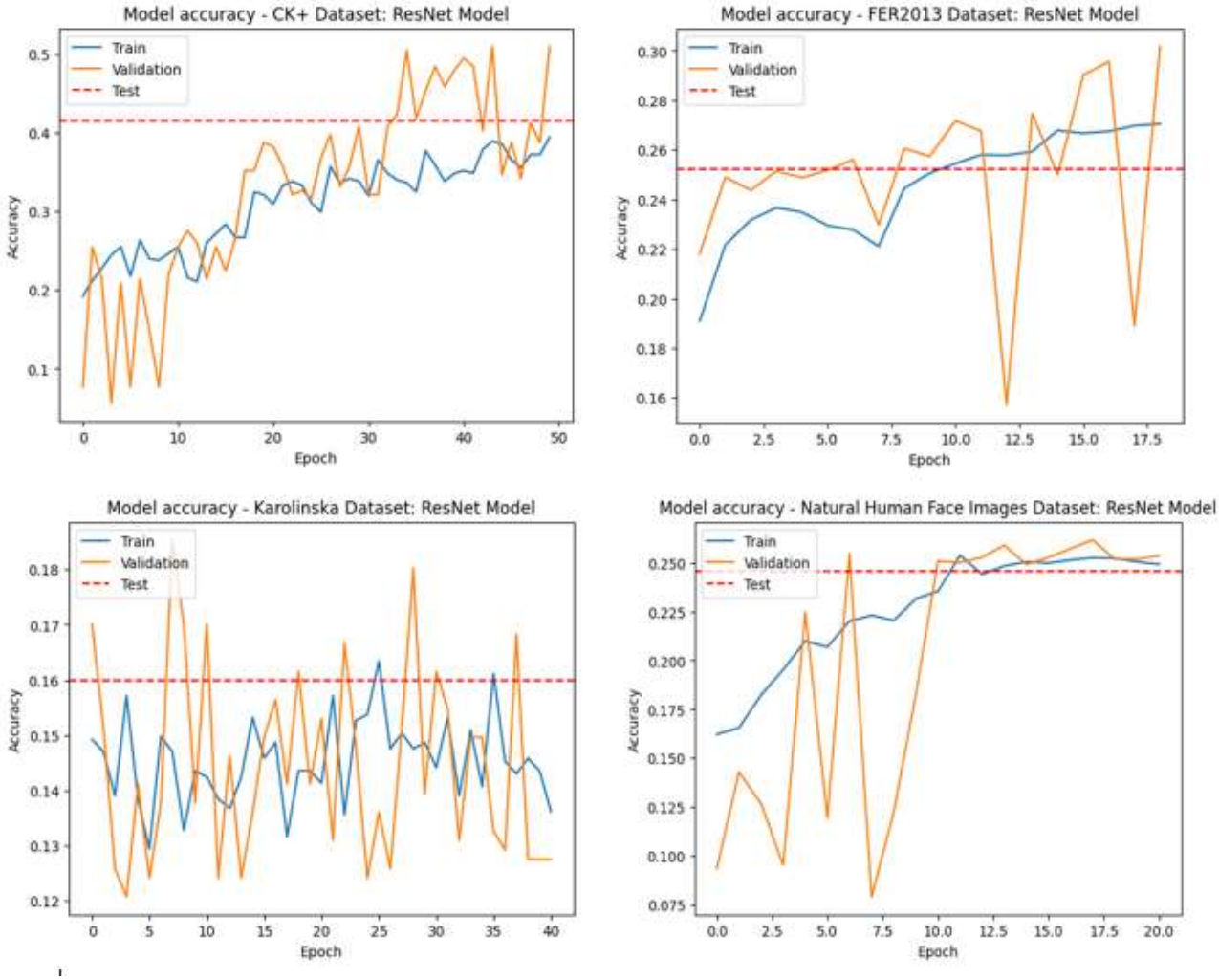

Figure 43: Accuracy Plots - ResNet50 Model

Figure 43 shows the model's accuracy plots of the ResNet50 model on the four (4) selected datasets. The CK+ dataset recorded a training accuracy score of +/- 39% , validation accuracy score of +/- 51%, and test accuracy score of +/- 42%; the FER2013 had an accuracy score of +/- 27%, validation accuracy score of +/- 30%, and test accuracy score of +/- 25%. Also, the Karolinska dataset recorded an accuracy score of +/- 14%, validation accuracy score of +/- 13%, and test accuracy score of +/- 15%, and the Natural Human Face Image dataset have an accuracy of +/- 25%, validation accuracy score of +/- 26%, and test accuracy score of +/- 25% respectively. As it was observed with the TensorFlow PyTorch, and the ResNet50 models, the best performance of the model was recorded on the CK+ dataset while the worst was recorded on the Karolinska Directed Emotional Facial images dataset.

Error Rate Plot – ResNet50 Model

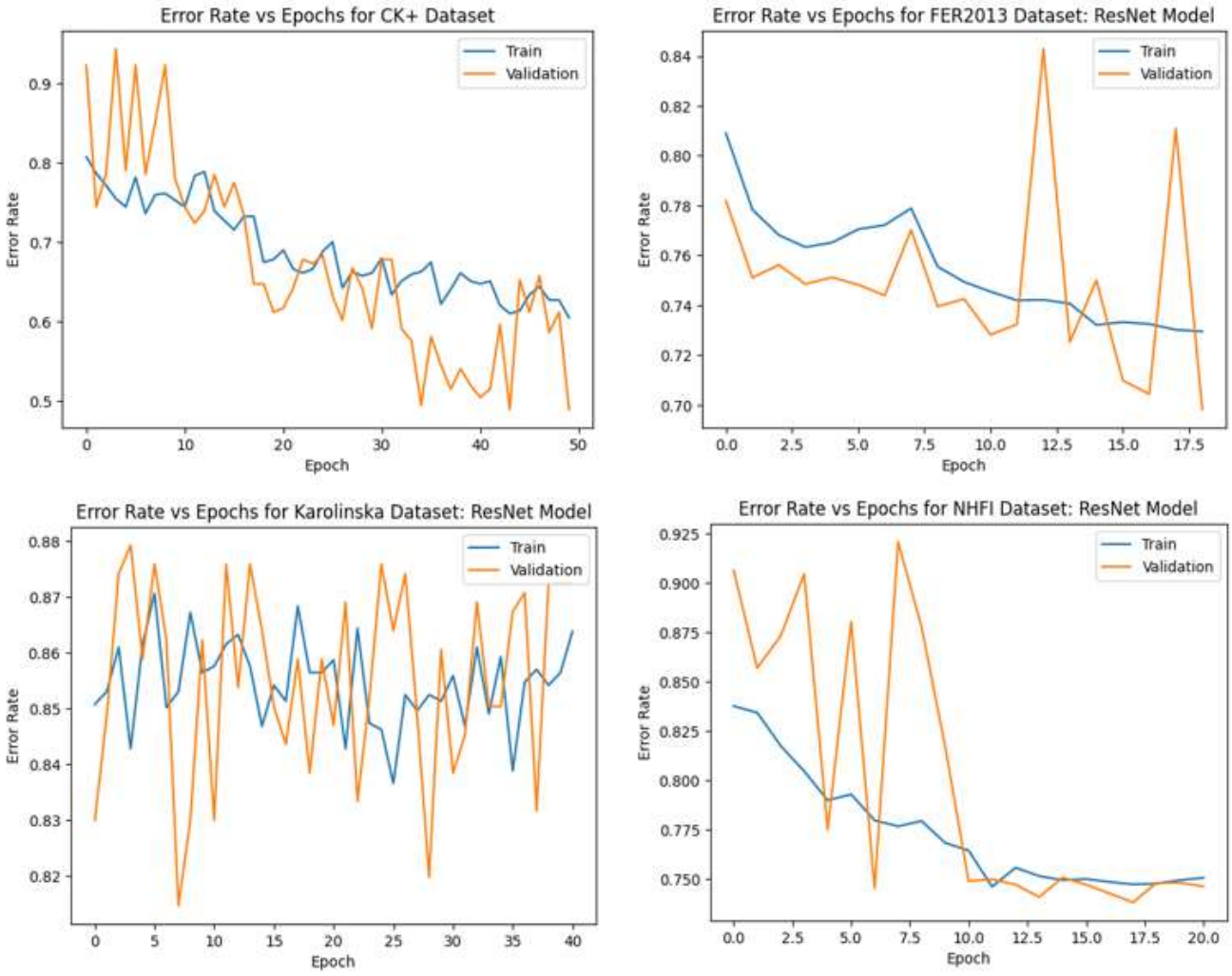

Figure 44: Error rate Plots - ResNet50 Model

Figure 44 shows the error rate plots obtained using the ResNet50 model on the four selected datasets. It can be observed that accuracy, validation, and testing error rate for the CK+ dataset are within a close range of 0.69 to 0.58 along the line of training, validation, and testing. The FER2013 dataset recorded the accuracy error rate of +/- 0.73, the validation error rate is +/- 0.70, and the test error rate is +/- 0.75. The Karolinska datasets accuracy, validation, and testing error rates are +/- 0.86, +/- 0.87, and +/- 0.84 respectively, whereas the training error of the Natural Human Face Image dataset +/- 0.75, validation error of +/- 0.74 and the test error is +/- 0.75. The ResNet50 recorded the highest cumulative error rate when compared to the rest of the CNN models that were used in these experiments.

Confusion Matrix – ResNet50 Model

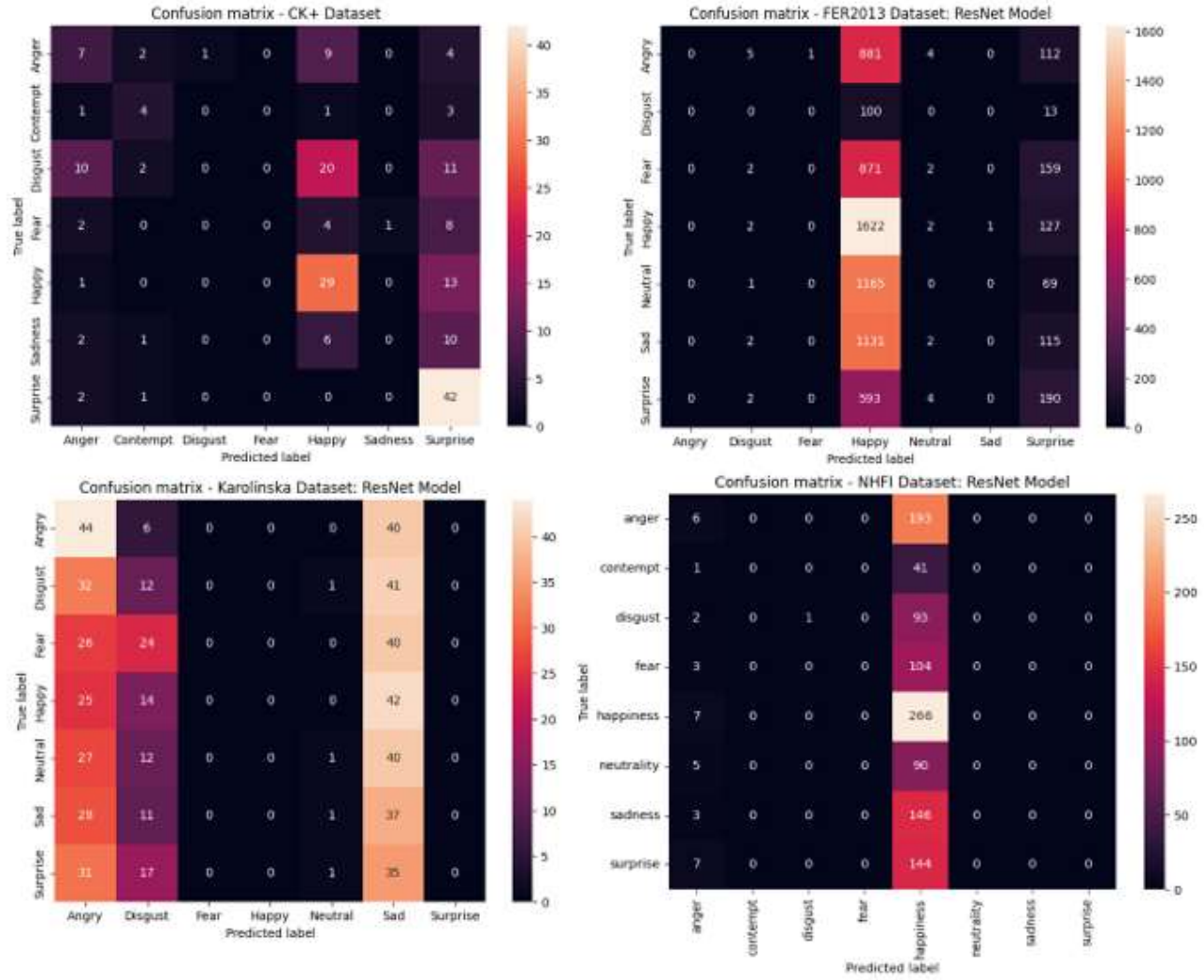

Figure 45: Confusion Matrices - ResNet50 Model

Figure 45 shows the confusion matrices for the four datasets. It shows a very poor performance of the ResNet50 model in classifying the emotion classes. It can be seen from the matrices that the model recorded a great deal of struggles in identifying and predicting the emotional classes from the four (4) selected datasets. The worst performance among of the ResNet50 model was recorded on the Natural Human Face image datasets as indicated by the lowest prediction accuracy. Although different hyperparameter tuning, data augmentation, and model adjustments were conducted on the ResNet50 model, an appreciable prediction performance could not be registered by the model.

ROC Curve Plot – ResNet50 Model

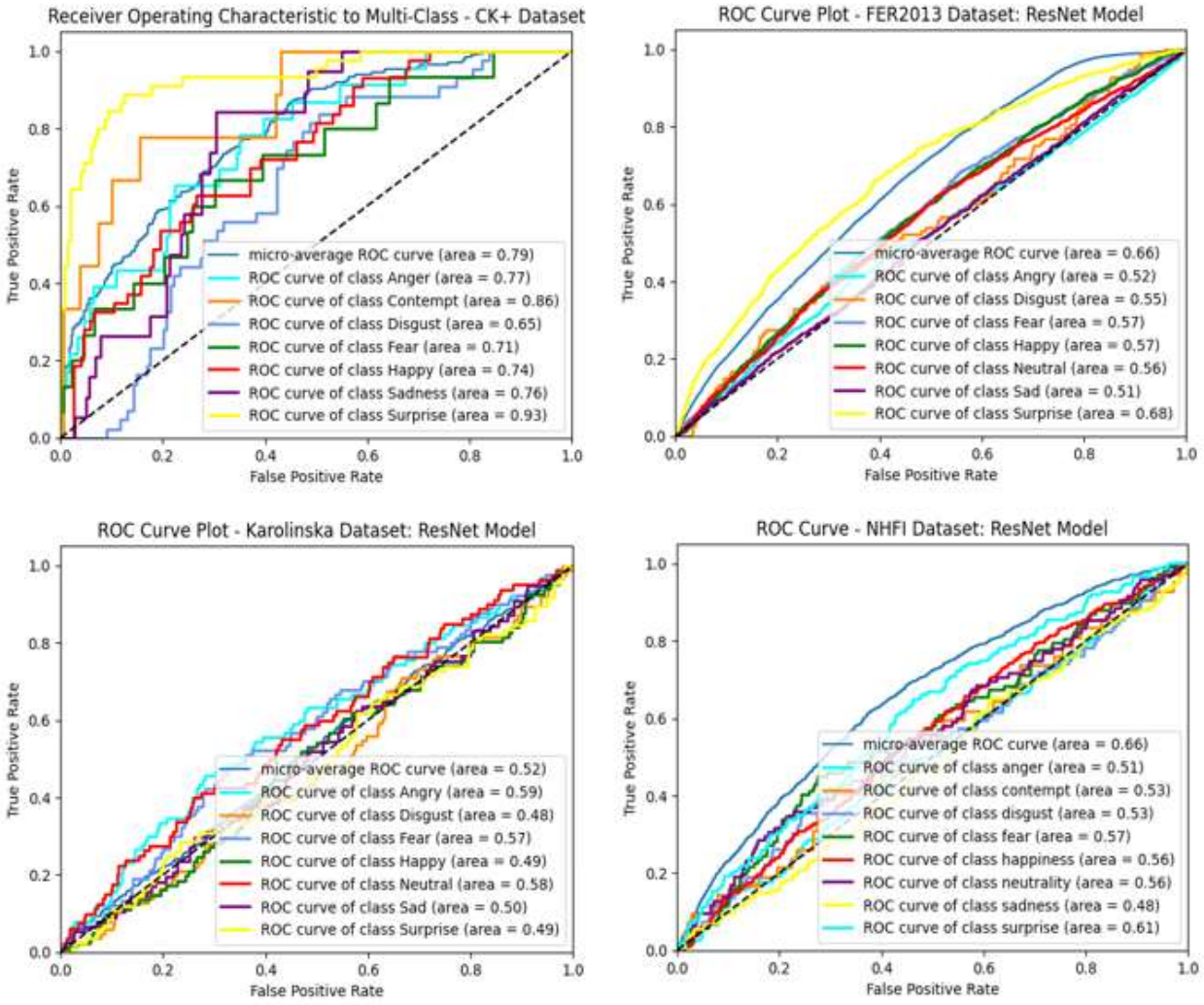

Figure 46: ROC Curve Plots - ResNet50 Model

Figure 46 shows poor performance of the ResNet50 model in recognizing the different emotions classes from the four (4) selected datasets. A micro-average AUC-ROC score of +/- 0.79 was recorded using the CK+ image dataset. The FER2013 dataset recorded a micro-average AUC-ROC score of +/- 0.66, Karolinska dataset recorded a micro-average AUC-ROC score of +/- 0.52, and the Natural Human Face Image dataset obtained a micro-average AUC-ROC score of +/- 0.66 respectively. It shows that the ResNet50 model shows some a very poor performance in discriminating between positive and negative cases for the varied emotional classes. Notably, the lowest micro-average AUC-ROC score was recorded using the Karolinska dataset using the ResNet50 model.

Comparison on ResNet50 Model

Table 5: Comparative Analysis showing results obtained using the ResNet50 Model

Model	Dataset	Accuracy	Precision	Recall	F1 Score	AUC ROC
ResNet50 Model	CK+	+/- 42%	+/- 22%	+/- 34%	+/- 26%	+/- 0.79
	FER2013	+/- 25%	+/- 7%	+/- 17%	+/- 9%	+/- 0.66
	Karolinska	+/- 16%	+/- 10%	+/- 16%	+/- 9%	+/- 0.52
	NHFI	+/- 25%	+/- 18%	+/- 13%	+/- 6%	+/- 0.66

Table 5 shows the accuracy, precision, recall, F1 score, and the AUC ROC obtained using the ResNet50 model on the four selected datasets. It can be observed that the CK+ recorded the best score of +/- 42% accuracy, followed by the FER2013 dataset and the Natural Human Face Images dataset both recorded a +/- 25% each. The Karolinska dataset recorded the least accuracy of +/- 16%, Figure 47 below shows a grouped bar chart plot of these metrics scores obtained using the ResNet50 model on these four datasets.

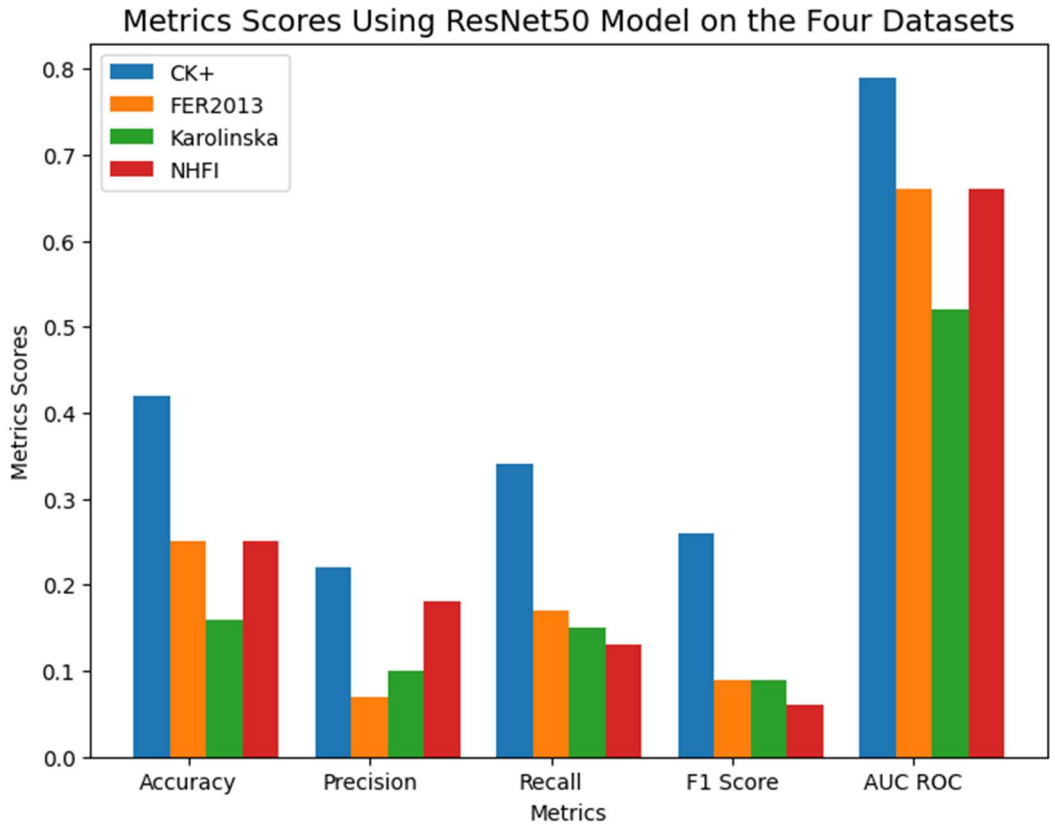

Figure 47: Metrics scores - ResNet50 model

4.5 Remarks and Conclusions

This chapter focuses on the implementation details and results obtained by the five (5) deep learning models developed for facial expression image-based emotion recognition. The chapter acts as a helpful manual by describing the models utilized, data preprocessing methods, and the results obtained. As this chapter outlined the five (5) CNN models that were used for this experiment, they are the proposed model (using TensorFlow and PyTorch frameworks), the SANet model, the VGG-16 model, and the ResNet model. This wide range of models offers a thorough understanding of the many architectures that can be used for emotion recognition tasks on facial expression images. Data preprocessing is an essential activity to be carried out for any deep learning task as was highlighted by this chapter. Effective data preprocessing plays and always has a considerable impact on the performance of the models.

The choice of multiple models and frameworks (TensorFlow and PyTorch) for implementation suggests a thorough and robust approach to tackling the problem. This could offer valuable insights into the pros and cons of each model and framework, thereby serving as a practical guide for those looking to venture into the field of emotion recognition. Hyperparameter tuning was also carried out during these experiments with the view of obtaining an optimized and best set of parameters to be used for the models that can improve the performance of the model. We made use of the *GridSearchCV* function of the scikit-learn library to search for the best set of parameters within the defined parameters grid.

The best results obtained indicate that the SANet model recorded the highest accuracy of 98% accuracy in classifying the various emotional classes within the CK+ dataset, while the F1 score is +/- 97% and the AUC ROC is +/- 0.98; the model also performed fairly on the other three datasets used for this experiment. Both the TensorFlow and the PyTorch models achieved +/- 95% accuracy prediction rate, F1 score of +/- 95%, and the AUC ROC of +/- 0.98 for the TensorFlow model while the PyTorch model achieved a F1 score of +/- 94% and the AUC ROC of +/- 0.98 as well. The VGG-16 model achieved a +/- 90% accuracy, F1 score of +/- 84% and AUC ROC of +/- 0.98, while the ResNet50 model performed poorly in predicting the emotional classes with the highest accuracy pegged at +/- 45%, F1 score of +/- 26% and the AUC ROC of +/- 0.76 using the CK+ dataset. As we stated above, the results (accuracies) of the five (5) models presented above are the best obtained during the experiments. However, in general, the results (accuracies) can be said to be lower than the ones presented above by a margin of 5% - 10%. This is because the results presented above are the best we obtained.

It is noteworthy that the CK+ dataset proves to be the dataset that all the five CNN models achieved higher accuracy scores among the four datasets selected for this task. One key challenge recorded during the experiment is that the training is resource intensive in nature. Although the ResNet50 recorded low accuracy prediction, we also tried other ResNet architectures like the ResNet34 and ResNet101 but yet without achieving a fair model performance.

In conclusion, this chapter provides a guide to a stepwise set of activities carried out in these experiments and the results that were obtained from the five models using four selected emotional facial expression images datasets. It can serve as a guide to interested researchers that want to delve into the practical aspects of developing and implementing deep learning models for emotion recognition. The meticulous approach in detailing the models employed, the datasets used, preprocessing techniques, the hyperparameter tuning techniques, and the results obtained paves the way for further research activity and application in the future.

CHAPTER 5

NUMERICAL EXPERIMENTS

5.1 Model with the best accuracy

This experiment was conducted using five CNN models which proposed CNN model (TensorFlow and PyTorch), SANet model, VGG-16 model, and the ResNet50 model on the four (4) selected image datasets. The best accuracy score was recorded using the SANet model with an accuracy of +/- 98% using the CK+ dataset. The proposed model for this thesis work which was implemented using TensorFlow and PyTorch both recorded an accuracy of +/- 95% using the CK+ dataset, the VGG-16 model recorded an accuracy of +/- 90% classification accuracy using the CK+ dataset. The ResNet50 model recorded the least accuracy of +/- 42% also using the CK+ dataset. Figure 48 shows the accuracies obtained by these models using the four datasets. It is worthy of note that in general, the accuracy is 5% - 10% lower than the best results presented and analysed below.

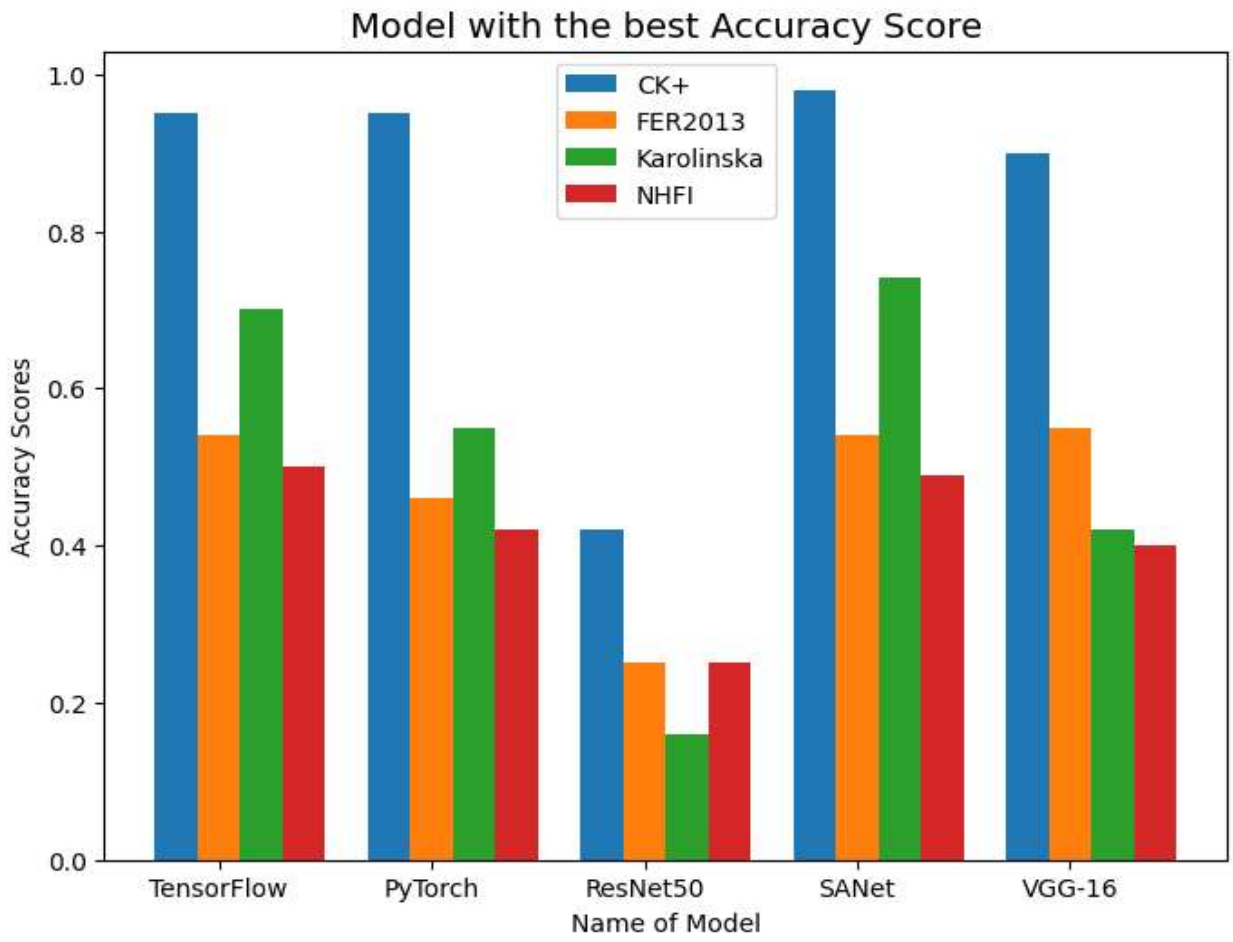

Figure 48: Models accuracies scores plot

We computed the average accuracy of each model by summing the individual accuracy of the four datasets and taking the average. The average accuracy of each model across the four datasets is presented in Table 6.

Table 6: Models' accuracy score for each dataset and their average

Model	CK+)	FER2013	Karolinska	NHFI	Average Accuracy Score
TensorFlow	+/- 0.95	+/- 0.54	+/- 0.70	+/- 0.50	+/- 0.67
PyTorch	+/- 0.95	+/- 0.46	+/- 0.55	+/- 0.42	+/- 0.60
SANet	+/- 0.98	+/- 0.54	+/- 0.74	+/- 0.49	+/- 0.69
VGG – 16	+/- 0.90	+/- 0.55	+/- 0.42	+/- 0.40	+/- 0.57
ResNet50	+/- 0.42	+/- 0.25	+/- 0.16	+/- 0.25	+/- 0.27

The table above can be visualized in Figure 49.

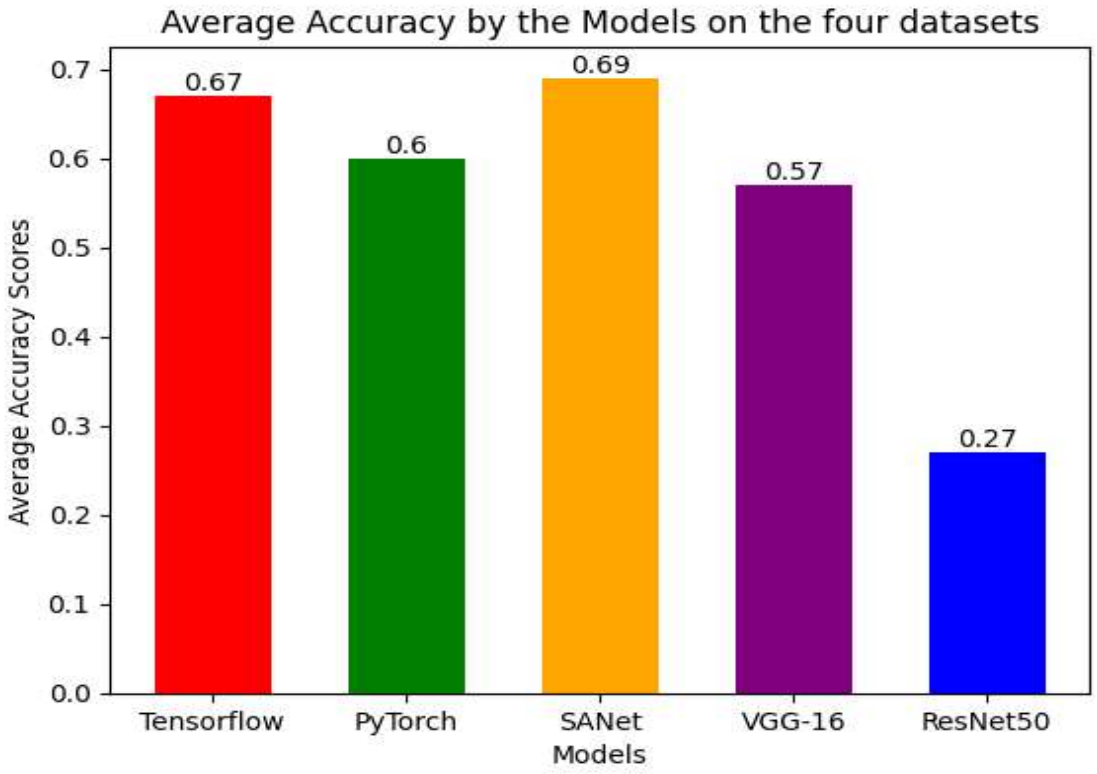

Figure 49: Average Accuracy of the models on the four datasets

From Table 6 and Figure 49, we can observe that the SANet model recorded the best average accuracy of +/- 69% emotion classification, the TensorFlow model recorded the second-best average accuracy score of +/- 67%, +/- 60% for the PyTorch model, while the VGG-16 model and ResNet50 model recorded an average accuracy of 57% and 27% respectively.

5.2 Model with the best F1 Score

While running this experiment, the best F1 score was also recorded. The F1 score of ML/DL classification task is a metric that aims to balance the precision and recall of a model; it is described as the harmonic mean of the metrics precision and recall, which are crucial for model classification. The SANet model recorded the best F1 score of +/- 97% using the CK+ dataset, the TensorFlow model recorded a +/- 95% F1 score on the CK+ dataset, PyTorch obtained a +/- 94% F1 score, the VGG-16 model recorded +/- 84% F1 score while the ResNet50 model recorded the least F1 score of +/- 26% on the CK+ dataset respectively. The plot below (Figure 50) presents a grouped bar chart showing the F1 scores obtained by these models using four datasets.

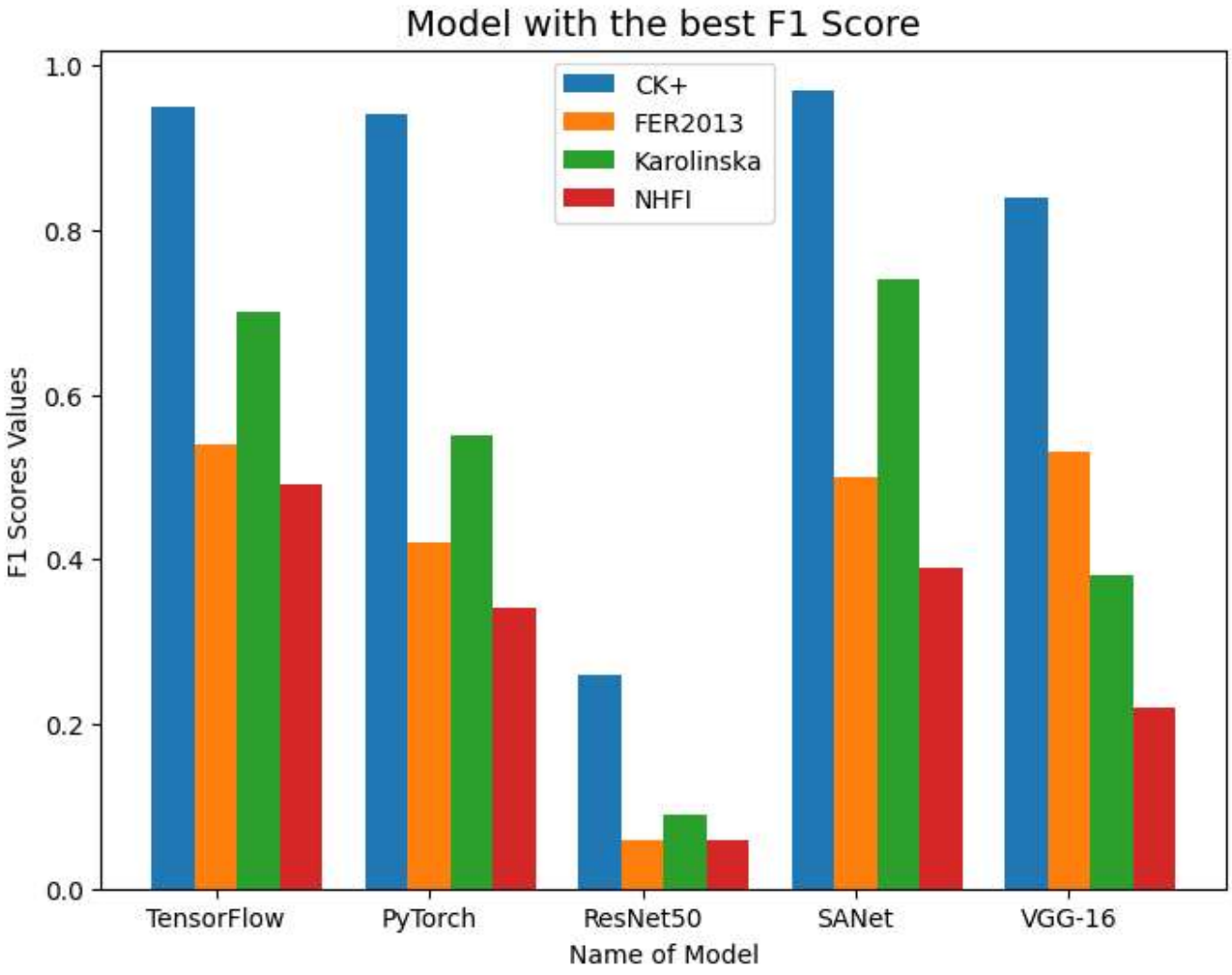

Figure 50: Models' F1 scores plot

5.3 Model with the best AUC ROC

The TensorFlow model, PyTorch model, SANet model and VGG-16 model all recorded an AUC ROC score of +/-98% on the CK+ dataset. The ResNet50 model recorded an AUC ROC score of +/-79% using the CK+ dataset. Using these models except for the ResNet50 model on the Karolinska Directed Emotional Facial Images datasets recorded the second-best AUC ROC score among the four datasets after the CK+ dataset. These AUC ROC scores indicate a good level or measure of separability, it demonstrates how well these models were able to distinguish between these emotional classes.

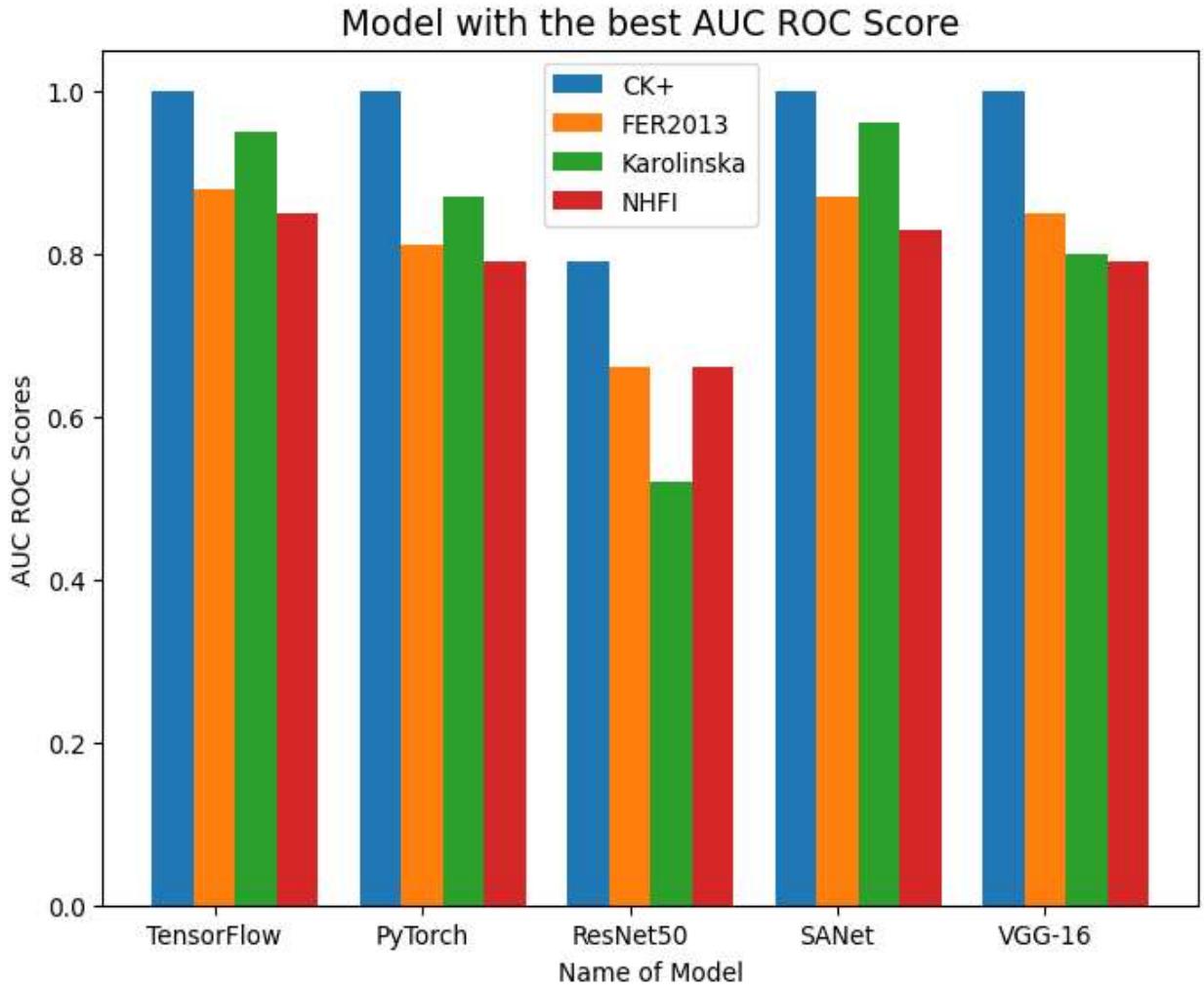

Figure 51: Models' AUC ROC scores plot

5.4 Model with the best time of training and evaluation

When considering the efficiency of deep learning networks, the efficiency of the model in terms of the time it takes to train and evaluate the model is often as crucial as accuracy, especially for real-time applications or resource-constrained environments. In this section, we aim to identify which among the five (5) tested CNNs: TensorFlow, PyTorch, SANet, VGG-16, and ResNet50 has the best time-efficiency in terms of training and evaluation across the four selected datasets: CK+, FER2013, Karolinska, and NHFI.

Methodology for Timing

The timing measurements were conducted on a machine equipped with an Intel Core i7- vPro CPU-based processor with 16GB or RAM memory. Each of these models was run five times on each dataset, and the average training and evaluation times were recorded; also, the standard deviation for both the training and evaluation times (s) were calculated as well. All models were trained and evaluated using the same batch size (32) and the same number of epochs (20) for consistency.

Table 7: Showing the mean training, evaluation times with their standard deviation

Model	Dataset	Mean Training Time (s)	Std Dev Training Time (s)	Avg Evaluation Time (s)	Std Dev Evaluation Time (s)
TensorFlow	**CK+**	+/- 79.28	+/- 2.13	+/- 16.06	+/- 0.99
	FER2013	+/- 4336.2	+/- 37.72	+/- 864.02	+/- 4.39
	Karolinska	+/- 534.02	+/- 6.12	+/- 107.04	+/- 2.56
	NHFI	+/- 988.42	+/- 7.57	+/- 199.04	+/- 1.06
PyTorch	**CK+**	+/- 48.02	+/- 2.34	+/- 9.34	+/- 0.21
	FER2013	+/- 5495.31	+/- 11.37	+/- 1098.47	+/- 5.38
	Karolinska	+/- 1682.86	+/- 14.27	+/- 337.23	+/- 4.39
	NHFI	+/- 2186.46	+/- 45.10	+/- 428.77	+/- 5.69
SANet	**CK+**	+/- 92.92	+/- 1.95	+/- 18.78	+/- 0.53
	FER2013	+/- 3702.87	+/- 76.24	+/- 736.31	+/- 13.20
	Karolinska	+/- 670.50	+/- 25.69	+/- 137.70	+/- 6.42
	NHFI	+/- 1087.32	+/- 54.83	+/- 215.78	+/- 6.27

VGG-16	**CK+**	+/- 395.30	+/- 7.29	+/- 77.55	+/- 5.53
	FER2013	+/- 34853.28	+/- 1297.07	+/- 7078.69	+/- 48.05
	Karolinska	+/- 2660	+/- 84.38	+/- 544.06	+/- 4.23
	NHFI	+/- 4386.56	+/- 153.33	+/- 473.79	+/- 17.19
ResNet50	**CK+**	+/- 355.76	+/- 31.40	+/- 62.76	+/- 2.39
	FER2013	+/- 12586.48	+/- 427.86	+/- 2476.89	+/- 33.09
	Karolinska	+/- 1562.8	+/- 53.61	+/- 319.16	+/- 22
	NHFI	+/- 2650.68	+/- 46.37	+/- 539.13	+/- 23.09

The table can be visualized in Figure 52.

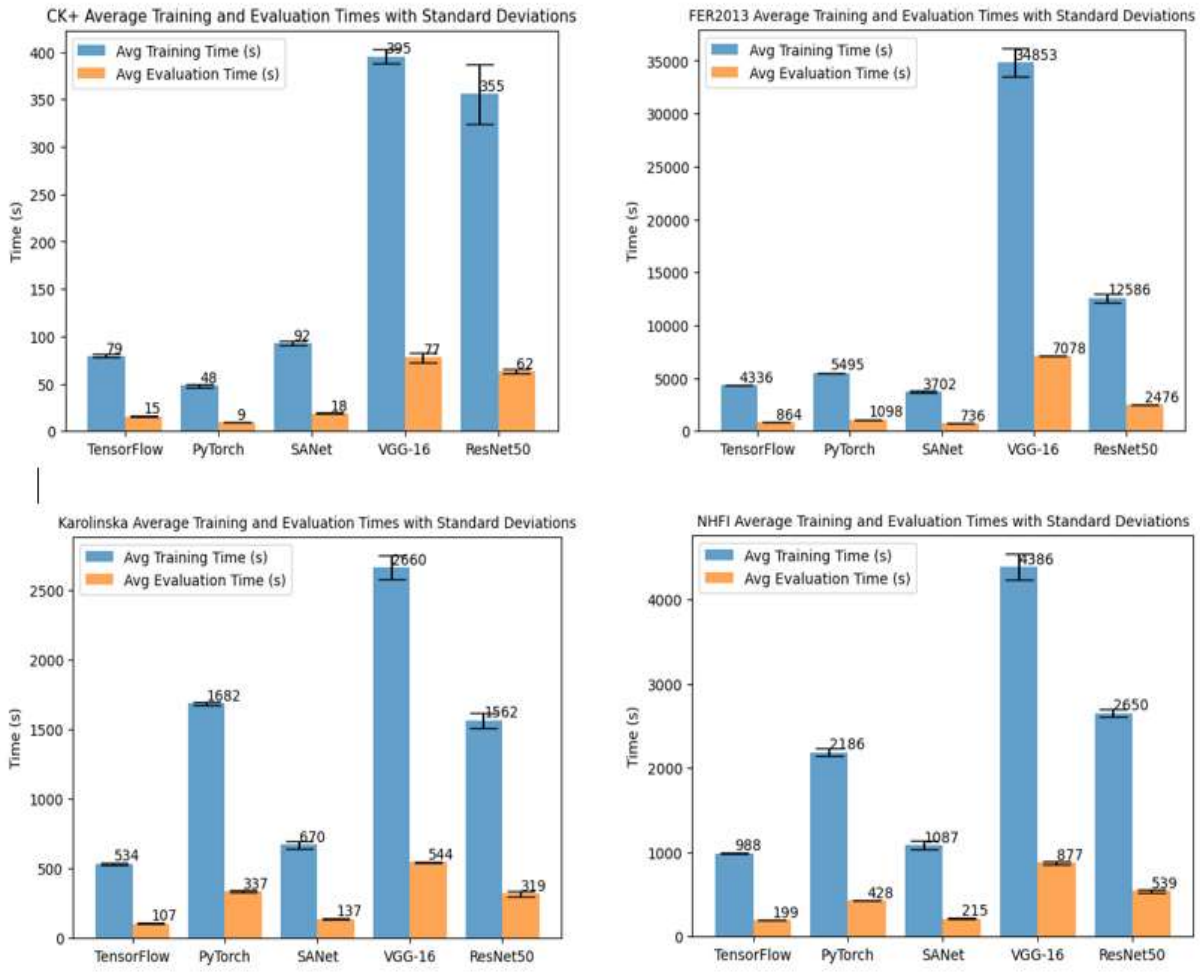

Figure 52: Average training time (s) and evaluation time (s)

Figure 52 shows the average training (s) and evaluation time (s) and the standard deviation for both the training and the evaluation times for the four selected datasets.

The model with the best training and evaluation time

Table 8: The average training and evaluation times (s) for the 5 models

Model	Average Training Time (s)	Average Evaluation Time (s)
TensorFlow	+/- 1484.48	+/- 296.54
PyTorch	+/- 2353.16	+/- 468.45
SANet	+/- 1388.40	+/- 277.14
VGG-16	+/- 10573.79	+/- 2043.52
ResNet50	+/- 4288.93	+/- 849.49

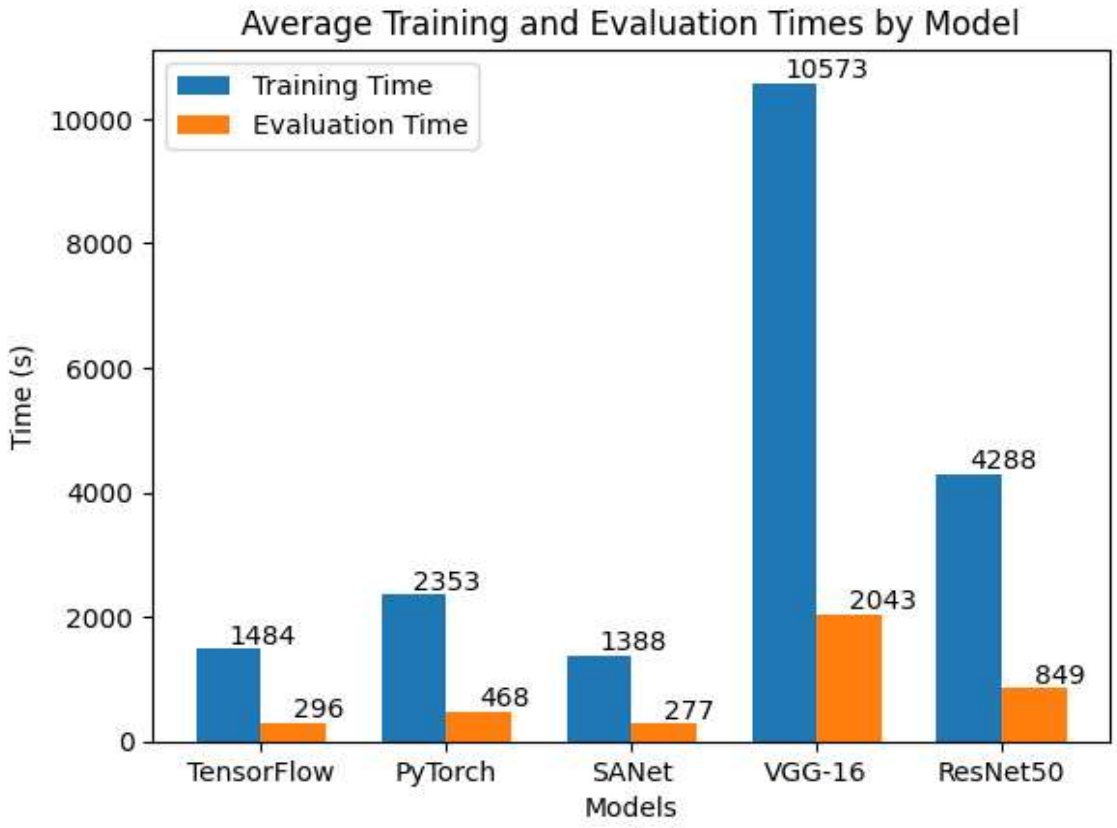

Figure 53: Model with the best (Training and Evaluation)

From figure 53 above, it shows the model with the best training and evaluation time (s) obtained using the four datasets. To obtain the values for this plot, we summed the average training times (s) and summed the evaluation times (s) for the obtained by the four (4) datasets. After summing them, we then computed the average and obtained the values we used to plot the above graph. It can be seen that the SANet model has the best training time (s) of +/- 1388 seconds and evaluation time (s) +/- 277 seconds. The TensorFlow model obtained +/- 1484 seconds and +/- 296 seconds as average training time and evaluation time respectively. The PyTorch model and the ResNet50 model came third and fourth respectively. As can be seen from figure 53, the VGG-16 model recorded the biggest training and evaluation time (s) among the five models used on the four selected datasets.

5.5 Model with the best memory usage

When building deep learning models, especially convolutional neural networks (CNNs), in situations with limited computing resources, memory efficiency is often overlooked yet it is a crucial factor to be considered. The memory usage of the four different facial expressions datasets (CK+, FER2013, Karolinska, and NHFI) that were used in training the five different CNN architectures - TensorFlow, PyTorch, SANet, VGG-16, and ResNet50 is explored in this section with the view of deciphering the model with the best memory usage.

All models were evaluated using the same hardware configuration, which consists of an Intel Core i7- vPro CPU-based processor with 16GB of RAM memory. Standardized software settings were used, including Python 3.9.7 and the most recent library versions available at the time of the experiments. The *psutil* package from Python was used to monitor RAM usage. The metrics employed to measure memory usage are the Peak Memory Usage and the Average Memory Usage during the training and evaluation process. Peak Memory Usage represents the total amount of memory used during the resource-intensive part of the computation. On the other hand, Average Memory Usage provides an overall picture of the memory requirements. The memory usage in (mb) used by the five models on the four selected datasets are hereby presented in the table below:

Table 9: The memory usage of the models on the datasets

Model	CK+ (MB)	FER2013 (MB)	Karolinska (MB)	NHFI (MB)	Average Memory (MB)
TensorFlow	+/- 288.39	+/- 759.11	+/- 454.34	+/- 288.47	+/- 447.58
PyTorch	+/- 303.88	+/- 699.91	+/- 251.87	+/- 127.74	+/- 345.85
SANet	+/- 176.90	+/- 719.55	+/- 225.62	+/- 301.44	+/- 355.88
VGG – 16	+/- 424.43	+/- 471.24	+/- 394.12	+/- 461.06	+/- 437.71
ResNet50	+/- 408.6	+/- 1034.07	+/- 257.72	+/- 258.97	+/- 489.84

The table can be seen visually as Figure 54.

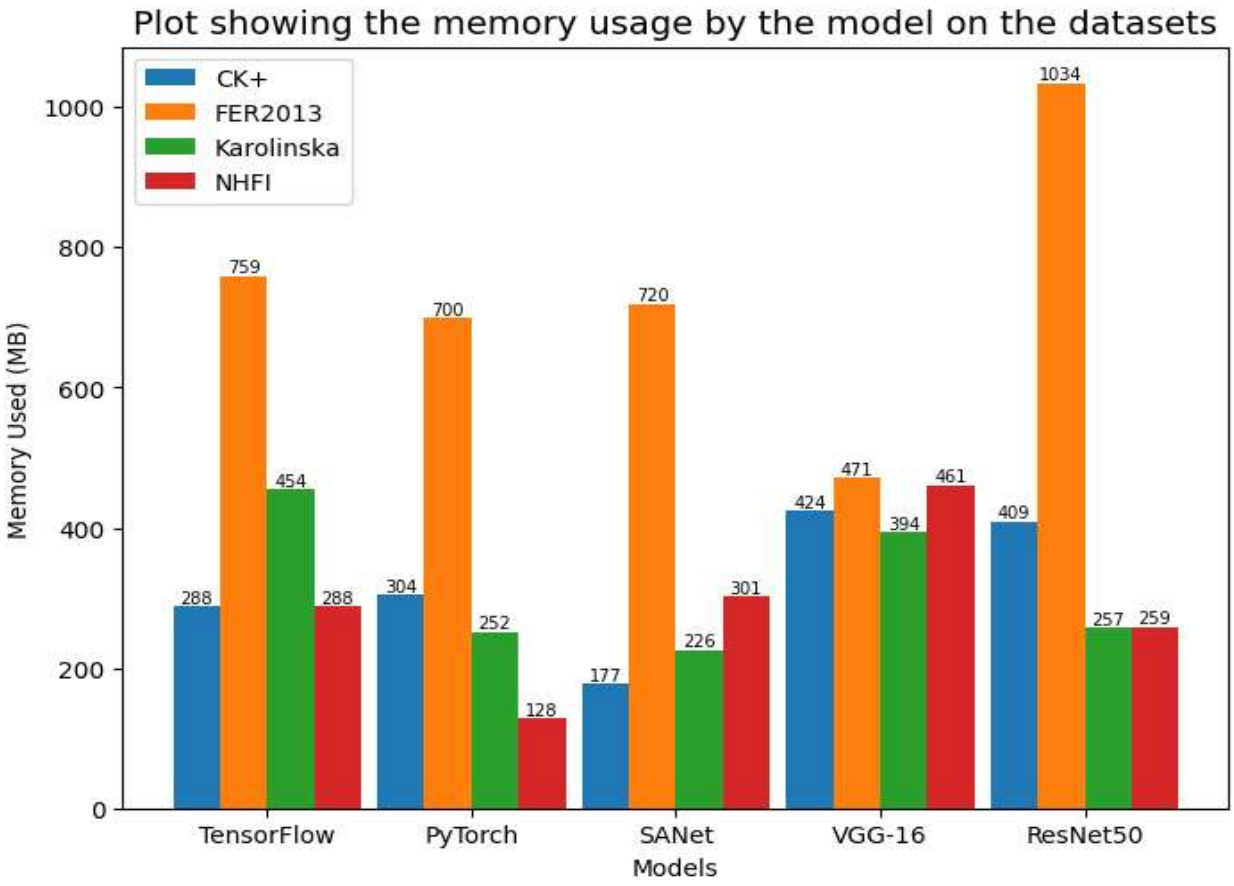

Figure 54: Memory usage of the five models on the four datasets

Figure 54 shows the memory usage of the five models on the four selected datasets. Even though the size of the datasets differs in terms of the number of images in each collection, we can observe from the plot that VGG-16 model recorded a close-ranged uniform memory usage across the four datasets. The SANet model proves to be using less memory across three of the four

datasets. The PyTorch and the TensorFlow models seem to be showing a near range memory usage between the two models. Also, the FER2013 dataset recorded the highest memory usage across the five models, this is not far from the fact that it is the largest of the four datasets that were selected for the experiment.

We computed the average memory usage of each model by summing the individual memory across the four datasets and taking the average. The average memory usage of each model across the four datasets is presented in Figure 55.

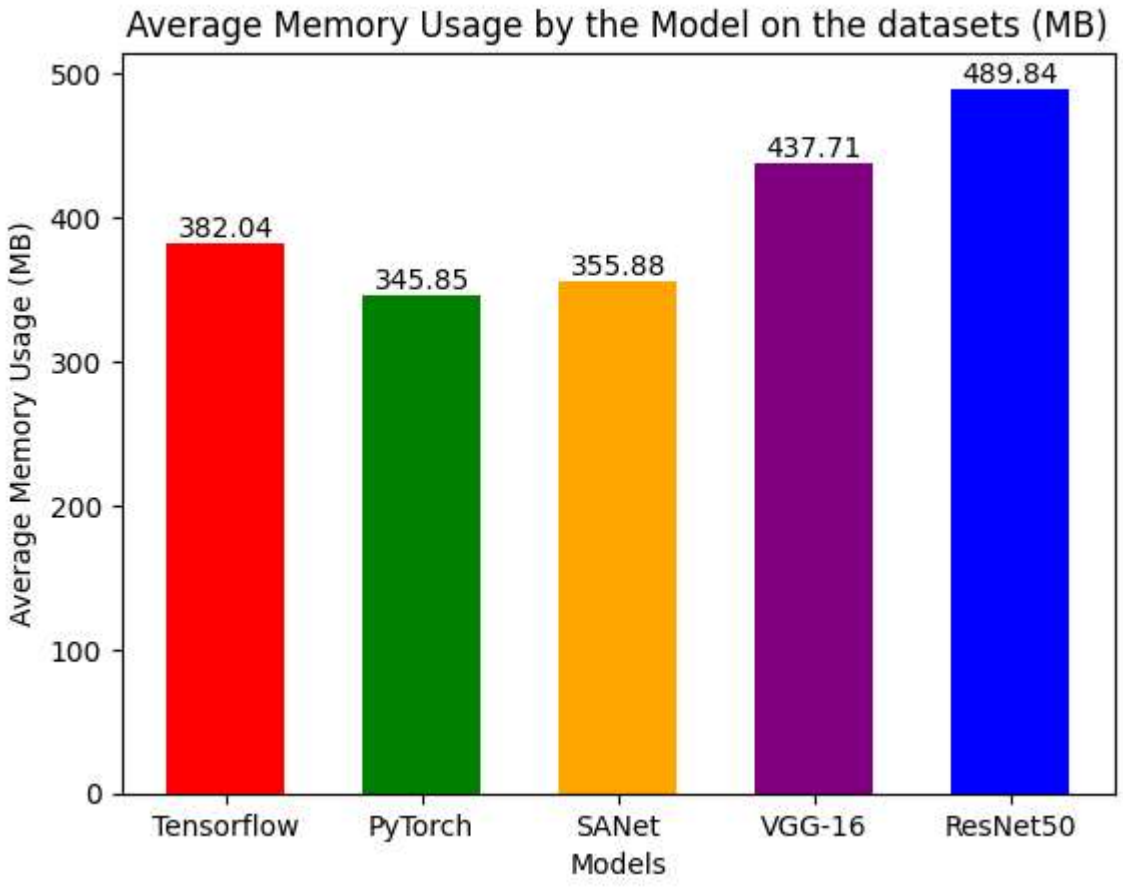

Figure 55: Average memory usage of the models on the four datasets

From Figure 55, we can clearly observe that the PyTorch model recorded the least average memory usage of +/-345.85 MB, this is the least among the five models. The SANet model recorded the second least average memory usage of +/-355.88 MB after the PyTorch model; the TensorFlow model recorded the third average memory usage of +/-382.04 MB while the VGG-16 model came fourth with +/-437.71 MB and the ResNet50 model is in the fifth and the last position

with +/-489.84 MB respectively. It can be deduced that by this observatory average memory usage score in megabytes, the PyTorch model is the model with the best memory usage.

5.6 Remarks and Conclusions

This chapter dived deeper into some comparative analysis, as it is fittingly titled "Numerical Experiments." A thorough set of comparative analysis was conducted by comparing the five (5) models used on the four (4) selected datasets. The objective is to find the model that has the best accuracy, F1 score, AUC ROC, best training, and evaluation time (s), and the model with the best memory usage (MB). These five (5) CNN models are: the SANet model, the VGG-16 model, the ResNet50 model, and a proposed model developed for the thesis and implemented in both the TensorFlow and PyTorch frameworks. The findings are hereby summarily discussed below.

Four distinct facial expressions image datasets (CK+, FER2013, Karolinska Directed Emotional Facial Images, and Natural Human Face Images datasets) were used to extensively evaluate these models, providing a thorough understanding of their strengths and weaknesses. The methodology used in the experimental design gives the findings more validity. It is noteworthy that the SANet outperformed all the other models on the CK+ and Karolinska obtaining an amazing accuracy rate of +/-98% and +/-55%. The VGG-16 model obtained the best accuracy of +/-90% on the CK+ dataset. This is a remarkable accomplishment that demonstrates the SANet model's potential for highly accurate image classification. Additionally, it establishes a standard for competing models and lays the road for additional analysis targeted at improving this model for diverse applications.

While the proposed model (TensorFlow and PyTorch models) could not outperform the SANet model using the CK+ dataset, they both achieved a commendable accuracy rate of +/-95% each. The accuracies of the proposed models demonstrated a good and competitive performance in the emotion recognition tasks from facial expression images and in the domain of image classification in general. Due to its reasonable performance, it has the potential to improve and become an even more reliable solution for a variety of image classification tasks. It is worthy of note that the models' high accuracy rates, especially those of the SANet model and the proposed model, have important ramifications for the field of deep learning and image classification. These models, when modified and fine-tuned, can be used in specialized applications like medical imaging, and autonomous vehicle navigation. These findings could serve as a starting point for further future research aimed at expanding the capabilities of image classification tasks.

It is noteworthy that the impact of the choice model and dataset can impact positively or negatively the accuracy of the results, as such, they must be considered. Although the SANet model performed exceptionally well using the CK+ dataset, the study might benefit from more tests to assess how well it performed in real-world circumstances. This might offer an insight into knowing more explicitly its potential strengths and weaknesses. Even though this study provides insightful information, it would also be beneficial to consider the practicality of implementing these models in real-world applications, considering factors like computational efficiency and scalability. Future studies might concentrate on these elements while also including other CNN models and datasets that can guarantee a more thorough and in-depth comparative study.

The SANet model obtained the best F1 score of +/-97%, the AUC ROC score of +/-0.98 was recorded by four models (TensorFlow, PyTorch, SANet and the VGG-16). Also, the SANet model obtained an average training time (s) of +/-1388.40 seconds and evaluation time (s) of +/-277.14 seconds. When it comes to the model with the best memory usage, the PyTorch model emerged as the best with +/-345.85 MB, and it is followed by the SANet model with +/-355.88 MB. The VGG-16 model used +/-437.71 MB while the TensorFlow and the ResNet50 model used +/-447.58 MB and +/-489.84 MB as the memory usage for these models respectively.

In conclusion, this chapter offers an articulated sequence of numerical experiments that greatly advance our knowledge of the capacities of the CNN models we used in image classification tasks. With its superior performance in these experiments based on the results we obtained, the SANet model stands out as a front-runner in terms of outstanding results, as such, we are recommending it for upcoming study and real-world use. As for the proposed models, they have shown great potential and with additional optimisation attempts, they can be considered for future applicability in other image classification tasks. We therefore opined that the complicated image classification problems could be solved even more creatively in the future by building on the discoveries made from this experiment.

Summary

The experiments were carried out successfully and results obtained show that the SANet model achieved the best accuracy of +/-98%, F1 score of +/-97% and AUC ROC of +/-0.98 using the CK+ dataset. The proposed model that was implemented using the TensorFlow and PyTorch frameworks achieved outstanding results; both of them achieving a +/-95% accuracy score, F1 score of +/-95% for TensorFlow while the PyTorch model scored +/-94% F1 score. The VGG-16 model recorded an accuracy of +/-90% and an F1 score of +/-84%. It is noteworthy that the TensorFlow, PyTorch, SANet, and VGG-16 models achieved an AUC ROC score of +/-0.98. The ResNet50 model recorded a poor performance of +/-45% accuracy on the CK+ dataset, +/-26% F1 score, and an AUC ROC score of +/-0.76.

In terms of training and evaluation time (s), the SANet model obtained the best average training time (s) of +/- 1388.40 seconds and evaluation time (s) of +/-277.14 seconds. The proposed model (TensorFlow and PyTorch models) recorded an average training time (s) of +/- 1484.88 for TensorFlow and +/- 2353.16 for the PyTorch model; the average evaluation time (s) of +/- 296.54 for the TensorFlow model and the PyTorch model's average evaluation time is +/-468.45 seconds respectively. The model with the best memory usage happened to be the PyTorch model with +/- 345.85 MB, and it is followed by the SANet model which recorded +/- 355.88 MB. The VGG-16 model used +/- 437.71 MB while the TensorFlow and the ResNet50 model used +/- 447.58 MB and +/- 489.84 MB as the memory usage for these models respectively. A notable constraint noted is the experiment's need for more computing resources that will facilitate faster and efficient experimentation. In summary, this research provides a holistic view of the current landscape of emotion recognition using deep learning models by presenting both theoretical explorations and empirical findings. We hope this thesis work will serve as a robust foundation for further advancements in the field of emotion recognition technology, image classification, and deep learning in general.

From the results obtained using these five (5) CNN models on the four (4) selected datasets, it can be observed that the SANet model performed better than the remaining other four models. More specifically, it recorded the best accuracy, F1 score, training and evaluation time, and the second-best memory usage after the PyTorch model. On the AUC ROC score, the SANet model recorded a joint-best AUC ROC score of +/- 0.98 together with the TensorFlow Model, PyTorch, Model, and the VGG-16 model. It is on this note that the SANet network is recommended for emotion recognition from facial expression images. On the second and third turn, the PyTorch and the TensorFlow network are recommended when computational cost is of essence.

Direction for further research

Although the five CNN models (TensorFlow model, PyTorch model, SANet model, VGG-16 model, and the ResNet50 model) and four datasets (CK+, FER2013, Karolinska, and NHFI) were considered for this experiment, there are yet other deep learning models and some facial expressions image datasets to be considered for further explorative research in the field of emotion recognition. A deeper dive into exploring hyperparameter tuning should also be considered with the view of achieving more state-of-the-art models' performance. Another area for consideration for further future research should be the model's sensitivity to noisy or varied input data and assess its performance under real-world conditions in situations where the data may be more diverse and less controlled to evaluate its generalizability and robustness further.

References

[1] L. Zhang, S. Wang, and B. Liu, “Deep learning for sentiment analysis: A survey,” *WIREs Data Mining Knowl Discov*, vol. 8, no. 4, Jul. 2018, doi: 10.1002/widm.1253.

[2] S. Afzal and P. Robinson, “Emotion Data Collection and Its Implications for Affective Computing,” in *The Oxford Handbook of Affective Computing*, R. Calvo, S. D’Mello, J. Gratch, and A. Kappas, Eds., Oxford University Press, 2015, p. 0. doi: 10.1093/oxfordhb/9780199942237.013.002.

[3] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, “The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, Jun. 2010, pp. 94–101. doi: 10.1109/CVPRW.2010.5543262.

[4] M. S. Hossain and G. Muhammad, “Emotion recognition using deep learning approach from audio–visual emotional big data,” *Information Fusion*, vol. 49, pp. 69–78, Sep. 2019, doi: 10.1016/j.inffus.2018.09.008.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012. Accessed: Aug. 31, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[6] I. J. Goodfellow *et al.*, “Challenges in Representation Learning: A report on three machine learning contests.” arXiv, Jul. 01, 2013. doi: 10.48550/arXiv.1307.0414.

[7] L. F. Barrett, R. Adolphs, S. Marsella, A. M. Martinez, and S. D. Pollak, “Emotional Expressions Reconsidered: Challenges to Inferring Emotion From Human Facial Movements,” *Psychol Sci Public Interest*, vol. 20, no. 1, pp. 1–68, Jul. 2019, doi: 10.1177/1529100619832930.

[8] K. Zhao, W.-S. Chu, F. De la Torre, J. F. Cohn, and H. Zhang, “Joint Patch and Multi-label Learning for Facial Action Unit and Holistic Expression Recognition,” *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3931–3946, Aug. 2016, doi: 10.1109/TIP.2016.2570550.

[9] T. Kanade, Y. Tian, and J. F. Cohn, “Comprehensive Database for Facial Expression Analysis,” in *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000*, in FG ’00. USA: IEEE Computer Society, Mar. 2000, p. 46.

[10] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, Jun. 2010, pp. 94–101. doi: 10.1109/CVPRW.2010.5543262.

[11] L. Zhong, Q. Liu, P. Yang, B. Liu, J. Huang, and D. N. Metaxas, "Learning active facial patches for expression analysis," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 2562–2569. doi: 10.1109/CVPR.2012.6247974.

[12] P. Liu, S. Han, Z. Meng, and Y. Tong, "Facial Expression Recognition via a Boosted Deep Belief Network," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 1805–1812. doi: 10.1109/CVPR.2014.233.

[13] P. Khorrami, T. L. Paine, and T. S. Huang, "Do Deep Neural Networks Learn Facial Action Units When Doing Expression Recognition?" arXiv, Mar. 15, 2017. doi: 10.48550/arXiv.1510.02969.

[14] "CK+ Datasets." Accessed: Sep. 12, 2023. [Online]. Available: https://www.kaggle.com/datasets/davilsena/ckdataset

[15] I. J. Goodfellow *et al.*, "Challenges in Representation Learning: A report on three machine learning contests." arXiv, Jul. 01, 2013. doi: 10.48550/arXiv.1307.0414.

[16] M. Arul Vinayakam Rajasimman, R. K. Manoharan, N. Subramani, M. Aridoss, and M. G. Galety, "Robust facial expression recognition using an evolutionary algorithm with a deep learning model," *Applied Sciences*, vol. 13, no. 1, p. 468, 2022.

[17] "FER2013 datasets." Accessed: Sep. 12, 2023. [Online]. Available: https://www.kaggle.com/datasets/msambare/fer2013

[18] E. Barsoum, C. Zhang, C. C. Ferrer, and Z. Zhang, "Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution." arXiv, Sep. 23, 2016. doi: 10.48550/arXiv.1608.01041.

[19] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, pp. 249–259, Oct. 2018, doi: 10.1016/j.neunet.2018.07.011.

[20] "2012.02312.pdf." Accessed: Jun. 08, 2023. [Online]. Available: https://arxiv.org/pdf/2012.02312.pdf

[21] S. H. Khan, M. Hayat, M. Bennamoun, F. Sohel, and R. Togneri, "Cost Sensitive Learning of Deep Feature Representations from Imbalanced Data," *arXiv.org*, Aug. 14, 2015. https://arxiv.org/abs/1508.03422v3 (accessed Jun. 08, 2023).

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012. Accessed: Jun. 08, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[23] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 1701–1708. doi: 10.1109/CVPR.2014.220.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587. doi: 10.1109/CVPR.2014.81.

[25] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv, Apr. 10, 2015. doi: 10.48550/arXiv.1409.1556.

[26] A. T. Lopes, E. de Aguiar, A. F. De Souza, and T. Oliveira-Santos, "Facial expression recognition with Convolutional Neural Networks: Coping with few data and the training sample order," *Pattern Recognition*, vol. 61, pp. 610–628, Jan. 2017, doi: 10.1016/j.patcog.2016.07.026.

[27] D. Lundqvist, A. Flykt, and A. Öhman, "Karolinska Directed Emotional Faces." May 11, 2015. doi: 10.1037/t27732-000.

[28] M. V. Garrido and M. Prada, "KDEF-PT: Valence, Emotional Intensity, Familiarity and Attractiveness Ratings of Angry, Neutral, and Happy Faces," *Front Psychol*, vol. 8, p. 2181, Dec. 2017, doi: 10.3389/fpsyg.2017.02181.

[29] "Karolinska Directed Emotional Facial Dataset." Accessed: Sep. 12, 2023. [Online]. Available: https://kdef.se/download-2/7Yri1UsotH

[30] M. Özdemir, B. Elagöz, A. Alaybeyoglu, R. Sadighzadeh, and A. Akan, *Real Time Emotion Recognition from Facial Expressions Using CNN Architecture*. 2019. doi: 10.1109/TIPTEKNO.2019.8895215.

[31] M. G. Calvo, A. Fernández-Martín, G. Recio, and D. Lundqvist, "Human Observers and Automated Assessment of Dynamic Emotional Facial Expressions: KDEF-dyn Database Validation," *Frontiers in Psychology*, vol. 9, 2018, Accessed: Jun. 07, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpsyg.2018.02052

[32] F. N. Azizi, A. Kurniawardhani, and I. V. Paputungan, "Facial Expression Image based Emotion Detection using Convolutional Neural Network," in *2022 IEEE 20th Student*

Conference on Research and Development (SCOReD), Nov. 2022, pp. 157–162. doi: 10.1109/SCOReD57082.2022.9974104.

[33] "Eng_2019_IOP_Conf._Ser.__Mater._Sci._Eng._705_012031.pdf."

[34] "Natural Human Face Images for Emotion Recognition." https://www.kaggle.com/datasets/sudarshanvaidya/random-images-for-face-emotion-recognition (accessed Jun. 10, 2023).

[35] "Natural Human Face Images datasets." Accessed: Sep. 12, 2023. [Online]. Available: https://www.kaggle.com/datasets/sudarshanvaidya/random-images-for-face-emotion-recognition

[36] E. Barsoum, C. Zhang, C. C. Ferrer, and Z. Zhang, "Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution." arXiv, Sep. 23, 2016. Accessed: Jun. 10, 2023. [Online]. Available: http://arxiv.org/abs/1608.01041

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." arXiv, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385.

[38] S. M, "Building Resnet-34 model using Pytorch - A Guide for Beginners," *Analytics Vidhya*, Sep. 14, 2021. https://www.analyticsvidhya.com/blog/2021/09/building-resnet-34-model-using-pytorch-a-guide-for-beginners/ (accessed Jun. 27, 2023).

[39] M. Manataki, N. Papadopoulos, N. Schetakis, and A. Di Iorio, "Exploring Deep Learning Models on GPR Data: A Comparative Study of AlexNet and VGG on a Dataset from Archaeological Sites," *Remote Sensing*, vol. 15, no. 12, Art. no. 12, Jan. 2023, doi: 10.3390/rs15123193.

[40] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv, Apr. 10, 2015. Accessed: Jun. 29, 2023. [Online]. Available: http://arxiv.org/abs/1409.1556

[41] G. Boesch, "VGG Very Deep Convolutional Networks (VGGNet) - What you need to know," *viso.ai*, Oct. 06, 2021. https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/ (accessed Jun. 29, 2023).

[42] K. Patel, "Architecture comparison of AlexNet, VGGNet, ResNet, Inception, DenseNet," *Medium*, Mar. 08, 2020. https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d (accessed Jun. 29, 2023).

[43] W. Yu, K. Yang, Y. Bai, T. Xiao, H. Yao, and Y. Rui, "Visualizing and Comparing AlexNet and VGG using Deconvolutional Layers".

[44] H. Fan and H. Ling, "SANet: Structure-Aware Network for Visual Tracking." arXiv, May 01, 2017. doi: 10.48550/arXiv.1611.06878.

[45] "Deep Learning," *MIT Press*. https://mitpress.mit.edu/9780262035613/deep-learning/ (accessed Jul. 08, 2023).

[46] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library." arXiv, Dec. 03, 2019. doi: 10.48550/arXiv.1912.01703.

[47] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning".

[48] F. Giobergia and E. Baralis, "RECLAIM: Reverse Engineering Classification Metrics," in *2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, Sep. 2022, pp. 106–113. doi: 10.1109/AIKE55402.2022.00024.

[49] M. Grandini, E. Bagli, and G. Visani, "Metrics for Multi-Class Classification: an Overview." arXiv, Aug. 13, 2020. doi: 10.48550/arXiv.2008.05756.

[50] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, vol. 17, no. 1, pp. 168–192, Jan. 2020, doi: 10.1016/j.aci.2018.08.003.

[51] M. Yousuff and R. Babu, "Enhancing the classification metrics of spectroscopy spectrums using neural network based low dimensional space," *Earth Sci Inform*, vol. 16, no. 1, pp. 825–844, Mar. 2023, doi: 10.1007/s12145-022-00917-1.

List of symbols

AUC	Area Under Curve
CK+	Extended Cohn-Kanade
CNN	Convolutional Neural Network
DL	Deep Learning
FER	Facial Expression Recognition
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GPU	Graphic Processing Unit
ML	Machine Learning
NHFI	Natural Human Face Images
RGB	Red, Green, and Blue
ROC	Receiver Operating Characteristic
TN	True Negative
TP	True Positive
TPR	True Positive Rate
VGG	Visual Geometry Group

List of Figures

List of Tables