Emotion Recognition Using Facial Images

Sila Shivanandan

Artificial Intelligence and Machine Learning
Dept of. Computer Science and Engineering

School of Computer Science and Engineering

Bachelor of Technology

ABSTRACT

An important component of human life and behaviour is emotion. Automatic emotion recognition has grown in importance in the domains of affective ciphering and machine-human interaction in recent years. Among the many physiological and active indicators that could be used to identify emotions, taking pictures of facial expressions is one of the most straightforward and affordable methods. Due to architectural, cultural, and environmental variations, developing a universal, cross-subject model for emotion identification from facial expressions is currently difficult. On the other hand, utilizing conventional machine learning techniques would necessitate a sizable collection of labelled examples in order to develop a subject-specific, personal model. Due to these factors, we suggest using transfer learning in this work to create domain-specific models for extracting the valence/arousal dimensions of face image emotional content. Transfer learning enables us to apply the feature extraction functionality in the single-subject situation while reusing the knowledge acquired from a huge multi-subject dataset by a deep convolutional neural network. By doing this, it is possible to use less labelled data when training a customized model than if only subjective data were used.

Chapter 1

Introduction

1.1 EMOTION

Emotions significantly influence how people think and act. Emotional states have an impact on decisions and how actions are taken. In addition, emotions play a significant part in human-to-human communication, and in many circumstances, emotional intelligence—the capacity to accurately assess, express, understand, and regulate one's own and other's emotions—is essential for a productive relationship. Research in affective computing aims to give computers emotional intelligence so they can be truly intelligent and enable authentic human-machine interaction[21]. Emotion recognition is used in many fields, such as marketing, safe and self-driving cars, mental health tracking, brain-computer user interfaces, security for society, and robots. How someone is feeling may be ascertained using a variety of techniques, including physiological markers such as the electroencephalogram, temperature of the body, electrocardiogram, electromyogram, galvanic skin response, the process of respiration talking, and bodily movements including facial expressions[22].

1.2 EFFECT OF EMOTION

The primary means of nonverbal communication that is most natural to employ among these is facial expression. Facial Emotion Recognition (FER) is an acceptable candidate for commercial-grade systems because RGB sensors are used to take pictures of the face are significantly less expensive than alternatives and do not require wearing anything. Traditional FER methods begin with the recognition of faces and landmarks before extracting hand-made features like facial Action Units[20]. Then, methods for machine learning like Support Vector Machines, or SVMs, are taught and put to use based on these attributes. Deep learning techniques, on the other hand, reduce the need for picture pre-processing by attempting to deliver a complete learning mechanism.
Convolutional neural networks are particularly effective at processing face pictures among deep learning models because they greatly reduce the requirement for physics- based models. Additionally, recurring networks, especially Long Short Term Memory, may benefit from temporal information if recognition is done on movies rather than single pictures[17].

FER may be expressed as either a classification or regression issue. The emotional model used to express emotions is what makes the difference. In categorical representations, emotions are shown as discrete entities with labels. Ekman noted that there are six universal signals and physiological characteristics that distinguish the six primary emotions (anger, disgust, fear, happiness, sorrow, and surprise)[16]. Tomkins identified nine biologically based effects, seven of which can be paired to represent the mild and the intense representation. Dimensional models, as opposed to discrete representations, aim to characterize emotions by quantifying some of their characteristics across a continuous range[18]. Russell hypothesized that emotions may be represented as two-dimensional in natural circular space with axes that correspond to the valence as well as arousal dimensions. Axes could be added to the circumplex model to further explain complicated emotions: The Dominance-Submissiveness axis in Mehrabian's Pleasure-Arousal-Dominance (PAD) model represents how much control one feels over an emotion as opposed to how much one feels the emotion is controlling them. Trnka and coworkers created a four-dimensional model to explain the valence, magnitude, management, and utility of emotion[20].

To bridge the gap between discrete and dimensional models, Plutchik and Kellerman proposed an integrated three-dimensional model. The eight basic bipolar emotions are grouped in concentric rings in this paradigm, with the more powerful emotions located in the inner circles[19]. The optimal model for portraying emotions and whether or not facial displays of emotions are universal are topics of continuous debate. From these debates, we step back in our article and concentrate on the utility of the models. Since most datasets that depend on specific feelings do not utilize the same emotional labels, it is challenging to conduct research that uses a variety of datasets. As a result, we choose to use the multidimensional model, more specifically, the Valence-Arousal technique. In contrast, after normalizing values as necessary and using the VA model as their shared denominator, almost all datasets based on dimensional presentations are consistent with one another[23].

1.3 APPLICATIONS OF EMOTION RECOGNITION

There are many potential applications for FER, some of which are described below in groupings according to the application field.

Healthcare:
- Predict psychotic illnesses or depression to identify patients in need of support
- Diagnose autism or neurodegenerative diseases

- Prevent suicide
- Diagnose depression in elderly patients
- Monitor patient conditions during treatment

Employment:

- assist recruiters in making decisions
- spot uninterested job seekers
- keep an eye on workers' tempos and concentration levels

Education:

- create an effective tutoring system
- track student attentiveness
- identify emotional responses of users to educational programs and modify the learningpath

Chapter 2

Literature Survey

A typical deep learning technique, deep belief networks (DBNs) excel in unsupervised feature learning. In this study, Yadan, Chao, and Zhiyong suggested a new deep learning- based technique of face expression detection. It combines DBNs and multi-layer perceptrons (MLPs). The recommended method combines the MLP's better classification skills with DBNs' advantage of unsupervised feature learning. Experimental results on the JAFFE database and the Cohn-Kanade database, two benchmarking facial expression databases, show that the proposed strategy for facial expression recognition surpasses earlier state-of-the-art classification approaches. The combined method of DBN+MLP got 90.9% on JAFFE and 98.5% on CK+ on 64*64 images. The suggested approach of DBNs+ MLP outperforms classification methods including NN, MLP, SVM, NS, as well as SRC, according to experimental data. This is explained by the potent unsupervised feature learning capabilities of DBNs. They overcame their challenges by using high image qualities with different postures, and different ethnic images.[5]

Xiaoming, Xugan, Shiqing suggested a deep-learning-based method employing convolutional neural networks to detect facial emotions from unposed, candid photos (CNNs). In order to evaluate the efficiency of candid facial expression identification in real-time, they created the casual image facial emotion dataset comprising seven distinct types of expression in over ten thousand pictures gathered from the Internet. Two feature- based methods (LBP+SVM and SIFT+SVM) are evaluated on the dataset as baselines.
They applied data augmentation and used 3 convolutional layer structures with an output layer. They achieved 81.5% accuracy on the data. Their ongoing projects involve integrating the designed characteristics with the learned features, as well as quickening online recognition to enable dynamic face expression analysis of live footage. They used good data preparation techniques to remove bias and try to enhance facial expressions[6].

The primary focus of this paper is facial expression recognition using face parsing components (FP). The detectors identify faces first, then nose, eyes, and mouth in a hierarchical order. Using the help of the focused features of components that have been recognized, a deep architecture pre-trained with a stacked autoencoder is used to recognize facial expressions. They used the JAFFE and CK+ datasets. A scaling Algorithm was used on test datasets to reduce human intervention. They trained these images on SVM, DBN, LDP, SAE, etc. Spare AutoEncoder scored 90% in JAFFE and 91% in CK+. Studies on the JAFFE and CK+ databases illustrate the effectiveness of the suggested approach in recognizing expressions and reveal that, in general, the mouth andeyes can distinguish between emotions[4].

In order to recognize human facial expressions, we have created a convolutional neural network. Two well-known facial expression datasets, CFEE and RaFD, which were independently trained and evaluated, yielded test accuracies of 74.79% and 95.71%, respectively, and have been adapted to the convolutional neural network model currently in use. The image identification dataset utilized in the ILSVRC 2012 served as the model's first training ground. The universality of results was shown by training on one set of data and testing on the other. Additionally, the facial expression recognition CNN was fed the picture output of the cropped faces and associated visual saliency maps, which were calculated using a Deep Multi-Layer Network for saliency prediction. In the most widely applicable experiment, we found that the test set's top-1 accuracy was 65.39%[12].

A recent theory of machine learning called "deep learning" emphasizes the depth of the learning model as well as the value of feature learning for network models. Scientific developments in the field of facial expression recognition have been accomplished thanks to deep learning. In order to compare the current research states, the most recent facial expression extraction algorithm and the FER method based on deep learning are used in this work. The datasets that were used are JAFFE, CK+, and MMI. Huilin, Zhiyu, and Biao applied geometric feature extraction, texture feature, and multi-feature fusion methods on the images. CNN and DBN neural networks are used in this survey. The challenges they faced were improving the algorithm robustness of the expression recognition, and the differences in race, geography, and culture, gender differences in expression, and the difficulty of real-time recognition. They overcame their challenges byusing high image qualities with different postures, and different ethnic images[10].

Human-Computer Interaction is becoming increasingly significant as we transition to a digital environment. Over the past ten years, a great deal of study has been conducted in this area. Face expressions are an essential component of non-verbal communication, and they are crucial to HCI. The method of Facial Expression Recognition (FER) presented in this study employs convolutional neural networks (CNN). Face expressions may be recognized in real time using this CNN-created model. They applied pre-trained models like VGG-16 and ImageNet on datasets such as CK+, and JAFFE and applied image transformation techniques like data augmentation and normalization. Their ongoing projects involve integrating the designed characteristics with the learned features, as well as quickening online recognition to enable dynamic face expression analysis of live footage. They employed more data to train their models, feature selection, regularization,and ensembling techniques to combat overfitting[13].

Deep learning approaches have allowed for cutting-edge performance in a number of applications, including recognition of facial features and human second identification. The Deep Neural Networks with Relativity Learning neural network training method is currently presented. This method immediately learns a mapping from the initial images to a Euclidean space, where relative distances correspond to a metric of facial expression resemblance. The model's parameters are changed by DNNRL based on sample significance. This method results in a versatile and trustworthy model. The DNNRL architecture consists of three layers of convolution, four pooling layers, three inception layers, and one fully connected layer. The Inception layers expand the network's breadth and depth without raising computational costs, and they also enhance local feature identification. They used the FER-2013 dataset and obtained a 72 percent accuracy.
DNNRL performs well for FER and is competitive with more established deep neural networks like AlexNet. The difficulties they encountered included overfitting as a result of insufficient training data and interference issues caused by other factors in the real- world environment that have nothing to do with

expression. They used good data preparation techniques to remove bias and try to enhance facial expressions[14].

CNNs haven't been properly tested for identifying facial expressions, though. In this study, we train and evaluate a CNN model for identifying facial expressions. Other pre- trained deep CNN models are assessed against the performance of the CNN model as a benchmark. Inception and VGG, which are pre-trained for object identification, are evaluated for their performance, and their results are contrasted with those of VGG-Face, which is pre-trained for face recognition. All tests are run using the CK+, JAFFE, and FACES face databases, which are all accessible to the general public. VGG-16 got an average accuracy of 88% and VGG-Face with 90% accuracy. They faced a lack of high- quality images, high-volume pressure of data preprocessing, visual privacy etc. The challenge they faced is the pose-variant faces. To overcome overfitting they trained their models with more data, used feature selection, and applied regularization and ensemblingmethods[11].

It is challenging to extract representative features since the bulk of the current deep learning-based FER algorithms does not sufficiently take into account domain expertise. In this study, we propose a novel FER framework (FMPN) called Facial Motion Prior Networks. To focus on the areas where the facial muscles move, we specifically give an additional branch to build a facial mask. By using the typical disparities between neutral faces and the associated expressive expressions as the training director for facial mask learning, they propose adding prior domain knowledge. They used the CK+, MMI, and AffectNet datasets. Their model(FMPN) achieved an accuracy of 98% on CK+, 82% on MMI, and 62% on AffectNet. To further incorporate the learned Spatio-temporal CNN features, they propose to use this in the future with the VGG16 on ImageNet data to separately optimise the spatial CNN network and the time-dependent CNN network on targeted footage-based emotion data[15].

With the recent development and widespread use of machine learning and deep learning methods, it is now possible to develop intelligent systems that accurately comprehend emotions. This project used the CK+, JAFFE, FER-2013, etc datasets for their study.
Daniel and Antonio applied face detection, geometric transformations, and histogram equalization to the images. CNN, SVM, KNN, Naive Bayes, etc models were used and CNN came on top with 98.9% accuracy. The challenge they faced is the pose-variant faces. To overcome overfitting they trained their models with more data, used featureselection, and applied regularization and ensembling methods[2].

One of the areas of symmetry is Facial Expression Recognition (FER), which serves as the main mechanism for understanding non-verbal intentions in the fields of computer vision and artificial intelligence. The image preprocessing techniques applied were noise reduction, face detection, normalization, and histogram equalization. The deep learning approaches were 3D-CNN, DBN, LSTMs, GAN, etc. The datasets used were BU-3DFE, BP4D-Spontaneous, etc. The highest accuracy was obtained on the JAFFE and CK+ datasets. Yunxin and Fei faced a lack of high-quality images, high-volume pressure of data preprocessing, visual privacy, etc[3].

Facial recognition is one of the most potent, inherent, and common ways to express emotions and intentions. Shan Li and Weihong Deng used pre-processing techniques like face alignment, augmentation, face normalization, etc., on CK+ and MMI. AlexNet, VGG, and ImageNet have been investigated in the fields of computer vision and machine learning to encode expression information from facial representations. The challenges they faced are bias and imbalanced distribution of data, the facial expressions not accurate to the emotion, etc. To overcome overfitting they trained their models with more data, used feature selection, and applied regularization and ensembling methods[1].

One of the most essential processes for accurate computations, human-computer interfaces, computer vision, computer game testing, and consumer studies involves facial emotional processing. One kind of indirect expression is facial expressions, which communicate a person's inner feelings and innermost thoughts. As a medium, as the quickest method of information transmission. Comparing modern state-of-the-art technology to more conventional ones, the accuracy rate has significantly increased. This article examines the most recent and current reviews in FER employing Convolution Neural Network methods and gives a quick summary of the key FER application domains and publically accessible datasets utilized in FER. Last but not least, it is shown that all individuals achieved exceptional results, particularly in terms of precision, although employing various methods and data sets, which have an influence on the conclusions[7].

The extraction of discriminative spatiotemporal video characteristics from facial expression images is one of the toughest challenges in facial expression recognition (FER) in video sequences. In this research, Sharmeen and Adnan presented a novel hybrid deep-learning model for FER in video sequences. The suggested method starts off by processing static facial images with one deep convolutional neural network (CNN) and optical flow images with the other. The datasets that were used are BAUM-1s, RML, and MMI. CNN scored an average of 58% and DBN scored 67%. Their next work will utilize this to independently optimize the spatial CNN network and the time-dependent CNN network on target video-based facial expression data using the VGG16 on ImageNet, as well as to extensively integrate the learned Spatio-temporal CNN features[8].

A model for occluded expression recognition is put out in the study and is based on the counteraction network that was created. The model is broken down into two modules: face recognition and obstructed face image restoration. The suggested approach for facial expression recognition outperforms previous state-of-the-art classification methods, like CNN, SVM, and Naive Bayes on CK+, JAFFE, and MMI datasets. The difficulties they encountered included overfitting as a result of insufficient training data and interference issues caused by other factors in the real-world environment that have nothing to do with expression. They used good data preparation techniques to remove bias and try to enhance facial expressions[9].

Chapter 3

Proposed Work

In this paper, we will try and understand the process of image recognition with the AffectNet dataset using many CNN architectures and techniques. The innovative pipeline is suggested and is based on face image processing. The sequences of each picture are first extracted using preprocessing methods including face recognition, enhancing facial characteristics, and others. Next, CNN layers are used to extract emotional features in each frame which is then passed to the Dense layer as the final output.

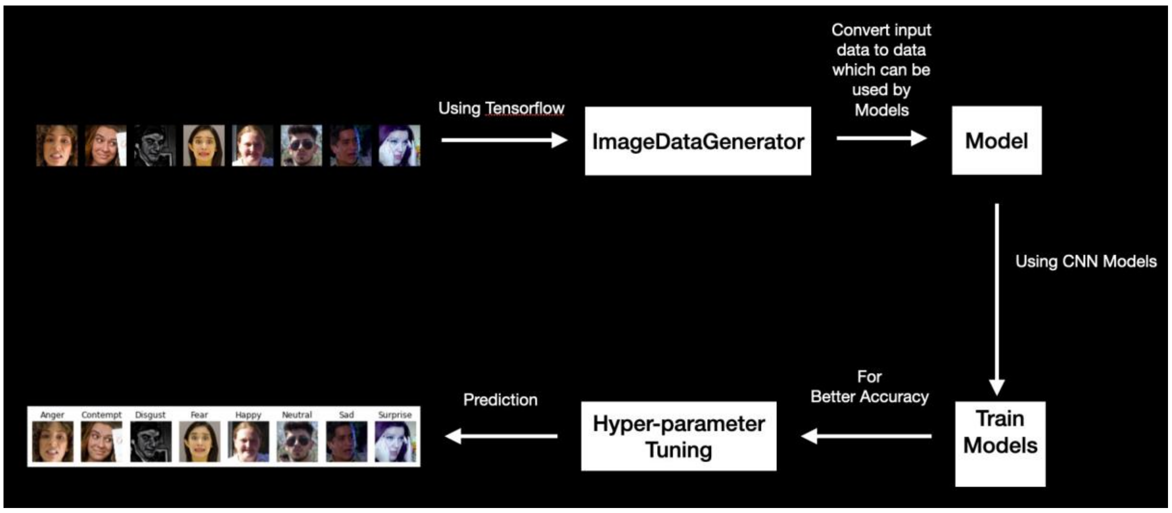


Figure 1

3.1 Dataset

3.1.1 AffectNet

AffectNet has gathered more than 1 million face photos using 1250 emotion- related phrases dispersed over six distinct languages and three significant search engine platforms. A category model was used to meticulously mark the existence of seven distinct facial emotions, in addition to the degree of polarity and emotion, in approximately half of the recovered images (440K). AffectNet, by far the biggest database of facial expressions, emotion valence, and alertness in the wild, enabled the work on computerized facial expression detection in two separate emotion models[24].

3.1.2 Cohn Kanade

Nearly all existing facial expression categorization algorithms make use of the CK+ database, which is considered the gold standard for controlled laboratory analyses of human expressions. CK contained 97 participants' 486 FACS-coded sequences. For the CK+ distribution, we have increased the dataset to 593 sequences from 123 individuals (an additional 107 (22%) sequences and 26 (27%) subjects). The duration of the image sequence ranges from 10 to 60 frames, and it includes the formation of the facial expressions from their commencement (which is also the neutral frame) to their climax[25].

3.1.3 RAF-DB

An expression dataset is the Real-world Affective Faces Database (RAF-DB). It contains 29672 face photographs that have had simple or complex emotions assigned to them by 40 different taggers. The age, gender, and ethnicity of the people, the head postures, the lighting, the occlusions (such as spectacles, facial hair, or self-occlusion), the post-processing processes (such as different filters and special effects), etc., are all very variable in the images in this database.

3.1.4 EMOTIC

The EMOTIC dataset, an abbreviation for EMOTions In Context, is an accumulation of photographs of people in natural settings, each of which has been tagged with an analysis of the viewer's observed emotional state. Alongside the three commonly used continuous dimensions of valence, arousal, and dominance, the photos have been captioned with an expanded description of 26 emotion types.

3.2 TensorFlow

Due to its extensive, adaptable ecosystem of tools, libraries, and community resources, TensorFlow is a completely free-to-download machine learning platform where programmers can quickly develop and use ML-powered products and academicscan advance the latest developments in machine learning research.

3.2.1 ImageDataGenerator

The Keras ImageDataGenerator is used to receive the input of the original data, which is then randomly transformed and generates a result that only includes the newly changed data. Information is not added. To increase the generalization of the whole model, data improvement is also carried out using the Keras picture data generator class. Random data augmentation procedures like shifts, interpretations, shearin, scaling changes, and horizontal flips are carried out using an image data generator.

3.3 CNN

A deep learning algorithm called CNN is capable of comprehending data with an ordered structure, such as images. CNN was created with the animal visual cortex in mind to autonomously and flexibly learn spatial structures of properties, from minimal to excellent patterns. A typical CNN will have three types of layers: convolution, pooling, and fully connected. The framework one and two convolution and pooling layers incorporate features of the extraction process, whereas an order three fully connected layer uses those features in the last step of the process, such as classification. In CNN, which comprises a stack of algorithms, including the specialized linear procedure known as convolution, the convolution layer is crucial.
How CNN Works?

- In order to carry out the convolution process, the pixels of the picture is fed into the convolutional layer.
- It's a path to a convolved map.
- The convolved map is then used to a ReLU function to provide an improved featuremap.
- Multiple convolutional and ReLU layers are applied to the picture in order to locate thefeatures.
- It uses many pooling layers, each with a unique filter, to zero down on certain parts ofthe picture.
- The final result is obtained by feeding a flattened version of the aggregated feature map into a fully linked layer.

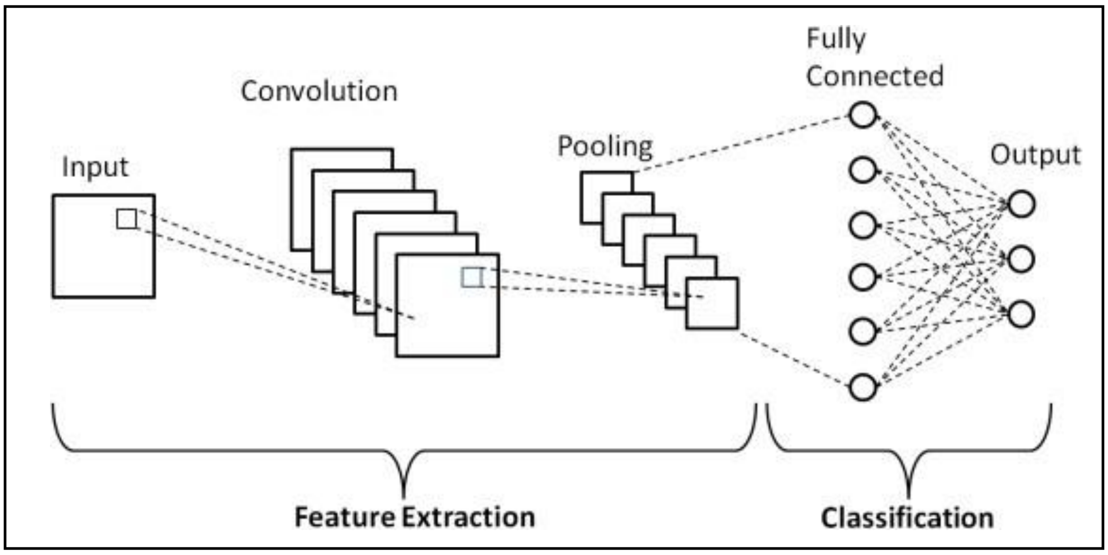


Figure 2

3.3.1 Components of CNN

A convolution neural network can extract information from an image with the aid of several hidden layers. In CNN, there are four significant layers:

• Convolution layer: The bulk of the computation in a CNN occurs in its central component, the convolutional layer. It needs data, an algorithm, and a feature map as inputs. CNN determines if a feature is present by scanning the receptive fields of the picture and applying a detector for features, sometimes referred to as a kernel or filter.The process in question is called convolution.
• ReLU layer: The rectified linear unit is commonly referred to as ReLU. The data maps must then be moved to a ReLU layer after being recovered. ReLU carries out a task, converting every negative pixel to 0 one at a time. A rectified feature map, the end result, provides the network with non-linearity. The rectifier function will make our photos more nonlinear. The rectifier function eliminates all the elements of an image that are black, leaving only those that have positive values (the grey and white colours). The order of the colours in the rectified and unrectified versions of the image is what distinguishes them most significantly. You'll notice that the colours change more suddenly after the image has been corrected. There was no longer a steady change. Thatsuggests that linearity has been abandoned.
• Pooling layer: Downsampling decreases the input's complexity and the number of parameters. Parallel to how the convolutional layer accomplishes it, the pooling operation distributes a kind of filter over the entire input, but this filter lacks features. Instead, the kernel uses a function for aggregation to fill the output array with values from the field that is receptive.
• Max pooling: As it processes the input, the filter selects the pixel with the greatest value and writes it to the resultant array. Furthermore, this method is used far more frequently than traditional pooling.
• Average pooling: It takes the mean value inside the receptive field as it moves over the input and sends it to the resultant array.

• Fully connected layer: The output layer and the pixel values of the input image are not directly coupled in partly connected layers. A node in the layer above it is directly linked to every node in the output layer of the fully-connected layer, in contrast. Based on the output from the previous layers and their corresponding filters, this layer executes the classification process. FC layers frequently utilize a softmax activation function, which generates a probability between 0 and 1, to accurately categorize inputs. In convolutional and pooling layers, ReLu functions are often employed.

3.4 Data Augmentation

Data augmentation is the practice of artificially creating new data from training data that has already been gathered. Cropping, cushioning, flipping, rotating, and resizing are examples of techniques. It improves the performance of the model and deals withissues like overfitting and a lack of data.
Now a well-known and widespread approach employed with CNN, image data augmentation uses operations like flips, rotation (at 90 degrees and smaller angles),translation, scaling, etc.

Utilizing data augmentation has numerous advantages, including:
- Prediction improvement in a model gets more accurate as a consequence of Data Augmentation's aid in recognizing samples the model has never seen before.
- The model has access to sufficient data to understand and train on each of the input parameters. This can be essential in situations where data collection is difficult.
- Increases the variety of the data via data augmentation, which helps prevent modeloverfitting.
- Can speed up processes where gathering additional data takes longer.
- May reduce the price required to gather various sorts of data if data collection is costly.

Different Image Augmentation Techniques
- Image Rotation: One of the most often used augmentation techniques is image rotation. Even after rotation, the image's data remains unchanged. As a consequence, we may use this technique to generate several pictures rotated at different angles, increasing thevolume of training data.
- Image Shifting: Another technique for picture augmentation is image shifting. By switching up the images, it is possible to alter the positions of the items in the image, giving the model more variety. It could eventually result in a model that is more complete. Using the geometric transformation known as image shift, each object's position in the input picture is translated to its new place in the output image. As a result, if an item is at position x,y in the original picture, it will be moved to a new location of X,Y in the new image, as seen below, where the corresponding shifts in both directions are indicated by the letters dx and dy.

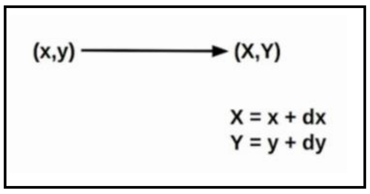


Figure 3

- Image Flipping: The next technique is Image flipping. You may think of flipping as extending rotation. It enables us to flip the image in both the Up-Down and Left-Right directions. In order to avoid overfitting and inject variety into the dataset, this approach is frequently performed randomly to photos during training. When this technique is applied, it is frequently believed that the labels are either transformable or invariant. For instance, segmentation allows for the flipping of the mask.

- Noise Addition: Image Noising is another widely used image augmentation technique in which noise is added to the image. Our model can learn how to distinguish the signal from the noise in an image using this method. Also, this makes our model more resistant to variations in the image. The probability distribution of images, which are frequently quantized in nature, is modeled using density estimation. It is required to convert the images into continuous form since the neural network thinks that the distribution is continuous.
- Image Blurring: The following method uses image blurring. Since photos originate from different places, the quality will differ according to the place. Both very high- quality and extremely poor photos may be included. In these cases, blurring the source photographs will increase the resiliency of our model to the test data's image quality. The dataset is degraded and faults are introduced using blue and other noise addition methods to strengthen the machine learning model's resistance to real-world instances.

3.5 Transfer Learning

The process when a model created for one issue is utilized in some capacity for another problem that is comparable is known as "transfer learning" in general.
Using the method of deep learning, also known as transfer learning, a neural network model is trained on a problem similar to the one that has to be addressed. A new model is then trained on the issue in question using a number of layers from the learned model.

Transfer learning has the benefit of reducing generalization errors and models of neural networks' time for training. Starting the training process using the weights from the earlier utilized layers and altering them to handle the new problem are also options. This application views transfer learning as a particular kind of weight initialization method. When the first related issue contains much more labelled data than the problem of interest, this may be advantageous since the similarity in the problem's structure may be helpful in both settings.

3.5.1 EfficientNet Model

The convolutional neural network building and scaling technique EfficientNet adjusts all dimensions, width, and quality characteristics evenly using an exponential factor. The EfficientNet scaling technique employs a set of fixed scaling coefficients to consistently scale network broadness, dimension, and resolution in contrast to typical practice, which scales these variables randomly. For instance, if and are constant coefficients obtained via a smaller grid search on the initial tiny model, we may simply raise the network depth, breadth, and picture size to consume times more computing resources. To scale the network's width, depth, and resolution evenly, EfficientNet employs an integral coefficient.

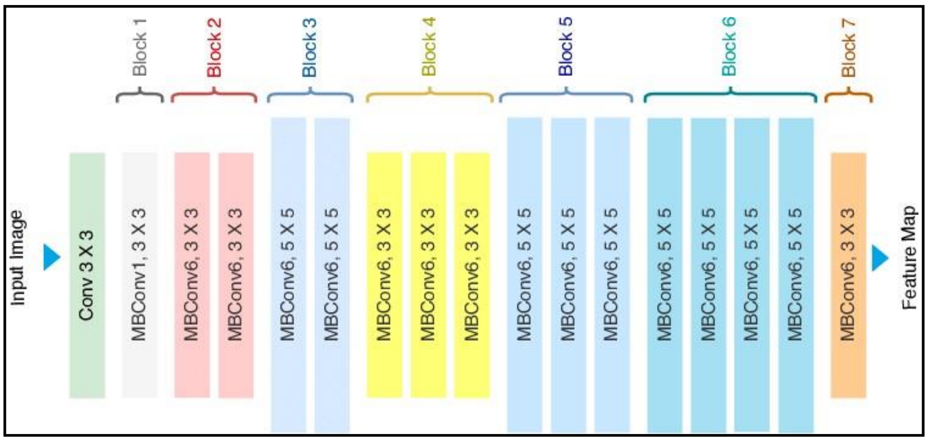


Figure 4

The relationships between various network properties are interconnected. To exploit more information from the image (bigger receptive field), for instance, the depth and width of the model should also be increased when the resolution (input dimensions) is increased in order to catch more fine-grained patterns with fewer pixels.

Let's assume:
Depth = d^{Φ} Width = w^{Φ} Resolution = r^{Φ}

d, w, r are constants, and they are optimized by doing a random grid search while fixing Φ=1, and are constrained such that:
- $d * w^2 * r^2 \approx 2$
- $d \geq 1, w \geq 1, r \geq 1$

Therefore, all that is left to adjust in order to scale up is the value of Φ. No longer are attributes like depth, width, and resolution required to be tuned simultaneously.

The equation was created so that, for any value of, the total FLOPS (floating point operations per second, which measure the training's speed) would roughly rise by 2^{Φ}.

Traditionally, a 3x3 convolution process consists of nothing more complicated than applying a kernel of size (3,3) to an input of depth and creating an output of depth. However, for a typical residual bottleneck block, a 1x1 convolution is used to first lower the depth of the input. Then, the reduced-depth input is subjected to a 3x3 convolution. Finally, the 1x1 convolution is used to expand the depth once more.

The EfficientNet Models are made up of 5 components:
- The building blocks are constructed from here.
- For each of the other seven primary blocks, this serves as the initial building block except the beginning one.
- This has a skip link to each individual building block.
- The skip link between the first few blocks is combined.
- This module is used to combine many sub-blocks by connecting each one to the one before it through a skip connection.

Depth-wise separable convolution is the name of this sophisticated procedure. As a matter of fact, it divides the straightforward 3x3 convolution into the 1x1 compression, 3x3 on the compressed, and 1x1 expansion processes.
The network can then learn more varied features by adding the original and final featuremaps.
This sophisticated method is more computationally efficient and makes use of a smaller number of parameters.

3.5.2 Inception Model

The Inception network was formerly thought of as a cutting-edge deep learning architecture (or model) for problems requiring photo identification and detection. The model is the outcome of multiple ideas that different scholars have developed throughout time. It was built upon Rethinking the Inception Architecture for Computer Vision by Szegedy et al.

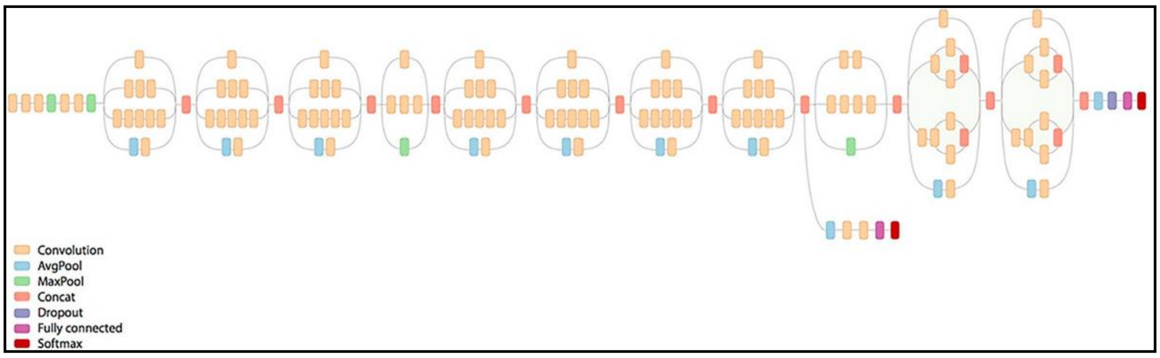


Figure 5

Several independent Inception modules compose the whole Inception model. The Inception V1 model's core component consists of four stacked layers:

- 1×1 convolution
- 3×3 convolution
- 5×5 convolution
- 3×3 max pooling

Convolutions, average pooling, max pooling, concatenations, dropouts, and entirely linked layers are some of the homogeneous and unbalanced building pieces that make up the model. The model extensively employs batch normalization, which is also used for the activation inputs. The loss is computed using Softmax.

Inception V1 sometimes used convolutions like 5*5, which greatly shrink the input dimensions. This reduces the precision of the neural network. This is because the neural network is susceptible to information loss if the input dimension is lowered too much. The two 3*3 convolutions in InceptionV3 replace the 5*5 convolution. TAs a 5*5 convolution costs 2.78 more than a 3*3 convolution, this also reduces calculation time and hence boosts computing speed. The auxiliary classifier additionally makes use of the RMSprop optimizer, batch normalization in the fully connected layer, 7*7 factorized convolution, and label smoothing regularisation, which normalizes the classifier by calculating the impact of label dropout during training. It prevents the classifier from predicting a class with an overly confident tone.

Consequently, the Inception V3 model is just the Inception V1 model that has been improved upon. The Inception V3 model optimized the network using a variety of techniques for better model adaptation.
- The efficiency is improved.
- Its network is more extensive than that of the Inception V1 and V2 models, yet it still performs quickly.
- The computational savings are substantial.
- Regularisation is accomplished by auxiliary Classifiers.

The significant decrease in dimensions was one of the key advantages of the Inception V1 model. The larger Convolutions were broken down into smaller in size Convolutionsto further refine the model.
It includes 55 convolutional layers, which as was already said required costly computing. Thus, two 33 convolutional layers were substituted for the 55 convolutional layers in order to lower the computational cost.
The fewer parameters also result in lower computation costs. A relative increase of 28% was achieved by the factorization of bigger convolutions into smaller convolutions.

Very deep neural networks' convergence is sped up using auxiliary classifiers. The auxiliary classifier plays a major role in solving the vanishing gradient issue in very deepnetworks.
The use of the auxiliary classifiers did not initially lead to any improvement. The network with auxiliary classifiers, however, fared better than the network without auxiliary classifiers in terms of accuracy as the trial went on.

Traditionally, average and maximum pooling was used to reduce the grid size of the feature maps. The Inception V3 model improves the activation dimension of the networkfilters to minimize the grid size.
The total layer count for the Inception V3 model is 42, somewhat higher than for the Inception V1 and V2 models. However, the efficiency of this technique is really astounding.

3.5.3 VGG Model

The term VGG, which stands for Visual Geometry Group, refers to a deep convolutional neural network (CNN) architecture that is common in that it has several layers. The VGG-16 and VGG-19, which are referred to recognized as "deep" models, comprise 16 and 19 convolutional layers, respectively. Models for uniquely identifying objects are developed using the VGG framework. The VGGNet, a deep neural network, outperforms other networks on a variety of tasks and datasets not included in ImageNet. Also, it is still widely used and considered a top picture recognition system.

The VGG16 model ranks in the highest five for test accuracy in ImageNet, at approximately 92.7%. The ImageNet collection comprises about fourteen million images classified into around a thousand different classes. It also performed well in comparison to the other versions entered at ILSVRC-2014. It achieves much better results than AlexNet by replacing the big kernel-size filters with a huge number of 3x3 kernel-size filters. Over the course of many weeks, Nvidia Titan Black GPUs were used to train the VGG16 model.

The VGG network is built using very compact convolutional filters. The VGG-16 comprises 13 convolutional layers and 3 fully linked layers.

- Input: 224*224 pixel pictures may be entered into the VGGNet. The middle 224*224 patch in each picture was removed by the model's creators in order to maintain a constant input size for the ImageNet competition.
- Convolution Layers: The VGG's convolutional layers use a receptive field size of 33, the smallest that still records in the vertical and horizontal directions. Furthermore, 11 convolution filters are used to perform a linear transformation on the input. The next part is a ReLU unit, which is a significant advancement over AlexNet and speeds up the training process. ReLU is a piecewise linear function that returns the input if the input is positive and 0 otherwise. The convolution stride is kept constant at 1 pixel in order topreserve spatial resolution during convolution.

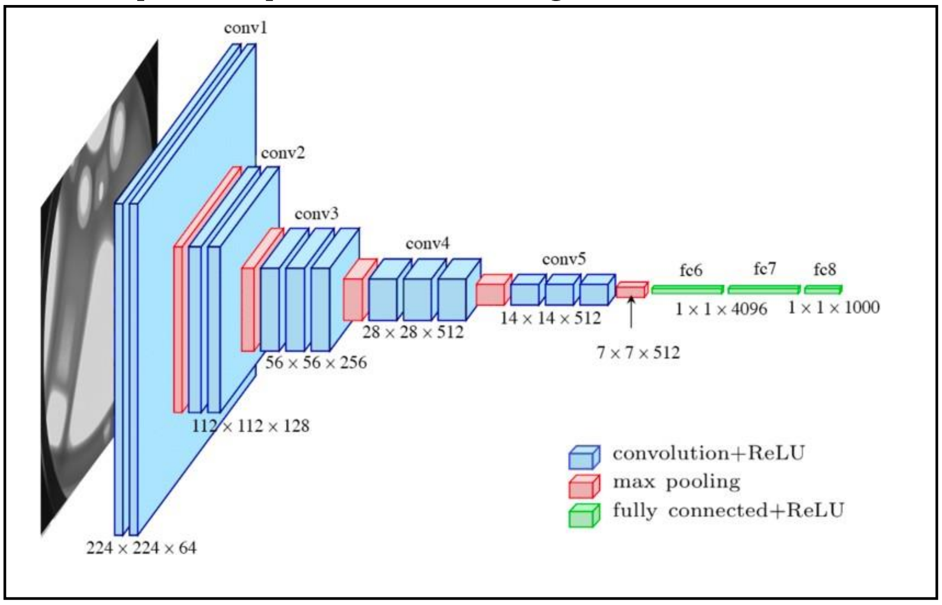


Figure 6

- Hidden Layers: Each hidden layer of the VGG network makes use of ReLU. Since LRN lengthens training times and uses more memory, it is often not utilized with VGG. Additionally, it doesn't improve accuracy in general.
- Fully-Connected Layers: The VGGNet consists of three completely linked layers. Each of the top two layers contains 4096 channels, while the third layer has 1000 channels with one channel for each class.

In the ILSVRC-2012 and ILSVRC-2013 contests, VGG16 significantly outperformed earlier models. Additionally, the VGG16 result performs noticeably better than the ILSVRC-2013 winning submission Clarifai and is in direct competition with Google's 6.7% error for the classification job winner. With external training data, it achieved 11.2%, while without, it reached about 11.7%. The VGGNet-16 model outperforms a single Google Net by roughly 0.9% in terms of single-net performance, with a test error of around 7.0%.

The uses of VGG Models are:

- Since the authors made the models accessible to the public and they may be used without modification for other jobs that are comparable, they can be utilized as a suitable classification architecture for many different datasets.
- Tasks requiring face recognition may also benefit from transfer learning.
- Weights are easily accessible with other frameworks, such as Keras, and may be altered and applied

any way the user sees fit.

- Using the VGG-19 network results in a loss of both style and substance.

The VGG19 model, occasionally known as VGGNet-19, is essentially comparable to the VGG16 model except that it consists of 19 layers rather than 16. We use numbers such as "16" and "19" to designate the model's weight layers. When viewed alongside VGG16,VGG19 contains three more convolutional layers.

3.5.4 ResNet Model

For computer vision applications, the Residual Network (ResNet) deep learning model is used. It is a design for a convolutional neural network (CNN) that can support a large number of convolutional layers—possibly thousands. Performance suffered since earlier CNN designs could only handle a limited number of layers. However, when additional layers were added, researchers ran into the "vanishing gradient" problem.
The backpropagation approach used to train neural networks reduces the loss function and determines the weights that minimize it by using gradient descent. If there are too many layers, performance reaches a saturation point or starts to degrade with each new layer, and further multiplicities will finally cause the gradient to "disappear."

The ResNet "skip connections" function presents a fresh approach to the vanishing gradient problem. Convolutional layers that are initially inactive (many identity mappings; ResNet) are stacked, skipped, and the activations from the previous layer are recycled. Skipping speeds up the initial training process by reducing the number of layersin the network.

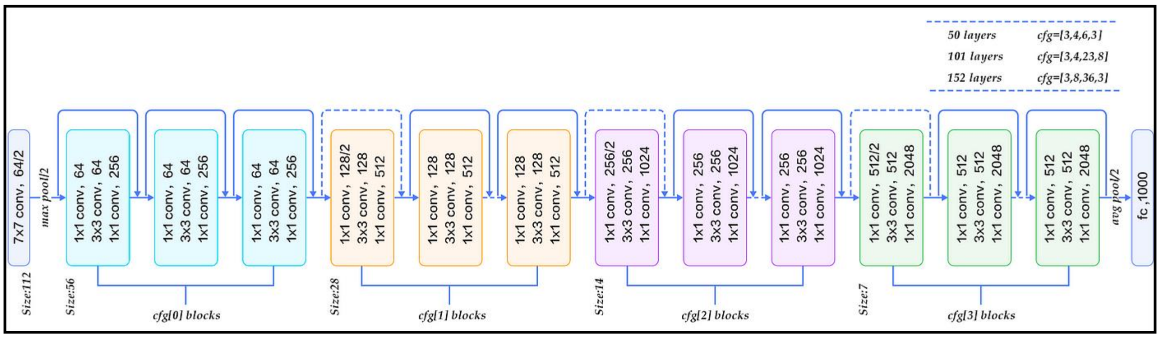

50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 128/2 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
1x1 conv, 256/2 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 512/2 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
avg pool/2
fc, 1000
Size:112
Size:56
cfg[0] blocks
Size:28
cfg[1] blocks
Size:14
cfg[2] blocks
Size:7
cfg[3] blocks

Figure 7
Although adding more layers in a stack to the model can enhance its properties, the issue of degradation might only become apparent at a larger network level. In a nutshell, there is a possibility that accuracy levels would plateau and then steadily drop as the number of layers in the neural network rises. Therefore, the model's performance declines on both its training data and the validation data. Over-fitting did not lead to this decline. Instead, it may be the result of a malfunctioning optimization method, faulty network setup, or, most significantly, vanishing or inflated gradients.

These skip connections operate in two different ways. First, they develop a new shortcut for the gradient to employ in order to fix the issue with the disappearing gradient. They also let the model acquire the capability of learning a unique function. Because of this, the model's higher layers are guaranteed to perform as well as, if not better than, its lowest ones. Finally, the identity functions of the layers may be learned much more quickly thanks to the remaining blocks. ResNet does this by adding more neural layers to deep neural networks, which both reduce error and increase efficiency. In contrast, skip connections enable much more extensive network training than was previously feasible by combining the results of older layers with those of subsequent layers.

The residual network, often known as ResNet, represented a substantial improvement in the training of deep convolutional neural networks for computer vision issues. More advanced models, such as the Resnet50, utilize 3-layer bottleneck blocks to provide greater accuracy and quicker training durations. The original Resnet used 2-layer blocksand had 34 layers.

3.6 Proposed Model

Five convolutional layers make up the neural network, all of which are followed by max-pooling layers with a Flatten and Dense layer. In order to introduce non-linearity, this model makes use of the Rectifier Linear Unit (ReLU) rather than the hyperbolic tangent function (tanh) as activation. We used a simplified neural network and more data to train in order to reduce overfitting. The model was created to perform an 8-class classification (softmax layer produces the output). To estimate valence and arousal separately, CNNs were trained. Images from the AffectNet database were divided into training (75%) and validation (25%) sets.

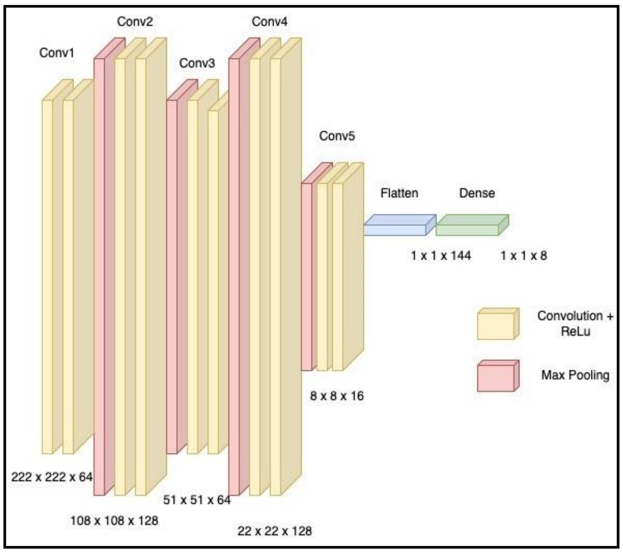


Figure 8

With batch size set to 32 and learning rate set to 0.001, we chose categorical cross- entropy as the loss function and trained the network using Adam as the optimisation. Over the entire training set, the training iterated for 10 epochs. The net obtained validation accuracy values of 0.6574 for validation accuracy and 1.64 for validation lossover the validation set, respectively.

Chapter 4

Results and Discussion

To see how our method affects recognition accuracy, we ran tests under three scenarios:
(1) an untrained CNN was trained on the AffectNet dataset with no transfer learning; (2) an AffectNet dataset was trained using data augmentation techniques on basic CNN models; and (3) an AffectNet dataset was trained using the aforementioned four TransferLearning methods.

For each model, we trained with Adam optimization, categorical cross-entropy as a loss function, and ReLu and Softmax as activation functions to get results. The first condition resulted in an accuracy of 50.6% on a simple CNN model, and the second condition resulted in a bad accuracy of 15%. In the third condition, we ran EfficientNetB7 got an accuracy of 45%, InceptionResNetV2 got 48%, ResNet152V2 got 52%, and VGG16 which got an accuracy of 55%.

Model Name	Accuracy
EfficientNetB0	53.6%
EfficientNetB3	48.8%
EfficientNetB5	51%
EfficientNetB7	49.2%
InceptionV3	45.7%
InceptionResNetV2	45.7%
VGG-16	59.9%
VGG-19	56.8%
ResNet50V2	50.4%
ResNet101V2	51.4%
ResNet152V2	52.8%

Table 1

Model Name	Accuracy
EfficientNetB0	44.4%
EfficientNetB3	43.9%
EfficientNetB5	50.9%
EfficientNetB7	52.3%
InceptionV3	41.8%
InceptionResNetV2	48.2%
VGG-16	48.2%

VGG-19	43.6%
ResNet50V2	51.8%
ResNet101V2	53.3%
ResNet152V2	51.4%

Table 2

A modified inverted bottleneck that is built on top of the depthwise convolution serves as the main building block for EfficientNets. By eliminating all channel connections in the depthwise layer, a type of structured sparsity, this decision drastically lowers the FLOPS and parameters needed. The layer's ability to represent solutions of varying complexity is also diminished. Therefore, more channels are added to increase overall capacity. As a result, there are fewer parameters and FLOPS than with other solutions, but because there are more channels, there is more data movement. We have a neural network architecture with EfficientNets that has significantly less computation and more data movement than comparable networks. Therefore, EfficientNets exhibit poor hardware accelerator performance.

The incorporation of convolutional neural networks greatly improves Inception'sperformance.
Its high-performance output from an Inception network is accomplished with little additional computational burden.
Scalable feature extraction from input data by modifying the size of the convolutional filter used. In order to improve the network's overall ability to extract features, 1x1 convolutional filters learn cross-channel patterns.

With VGG, various structures based on the same concept are introduced. This gives us more options to consider when choosing the architecture that would be most effective forour application.

As the number of layers with smaller kernels increased, non-linearity also increased, which is always advantageous in deep learning.
Both accuracy and speed significantly improved as a result of using VGG. The addition of pre-trained models and the deepening of the model was primarily responsible for this.

Training networks with hundreds or even thousands of layers is straightforward and does not result in a noticeable increase in the training error rate.
ResNets help with the vanishing gradient issue by using identification matching. ResNets are only regular networks with a few tweaks here and there. The design is based on the same functional stages as CNN or other systems but with an additional step added to solve problems like the vanishing gradient. A deeper network typically takes weeks to train, which makes it practically impossible to use in real-world applications.

I tested the proposed model on the Cohn Kanade, RAF-DB, and EMOTIC datasets. On the RAF-DB dataset, the framework performed better, but not as well on the CohnKanade and EMOTIC datasets. Accuracy was 72% for RAF-DB, 57% for CK+, and 54% for EMOTIC. F1 Score for CK+ was 0.65, RAF-DB was 0.78 and 0.61 for EMOTIC.

Using the Cohn Kanade dataset, other models reached an accuracy of approximately 90%, but the RAF-DB dataset only managed around 85%. Many variables, including lighting in the photos, head position, occlusion of facial features, variation in facial emotions for the same person, image quality, etc.,

contributed to the suggested model'sless impressive performance.
Facial expressions may be somewhat different from person to person, can include several emotions experienced at the same moment (such as fear and fury, pleased and sad), or may not communicate an emotion at all, therefore facial expression analysis may not be reliable.

Chapter 5

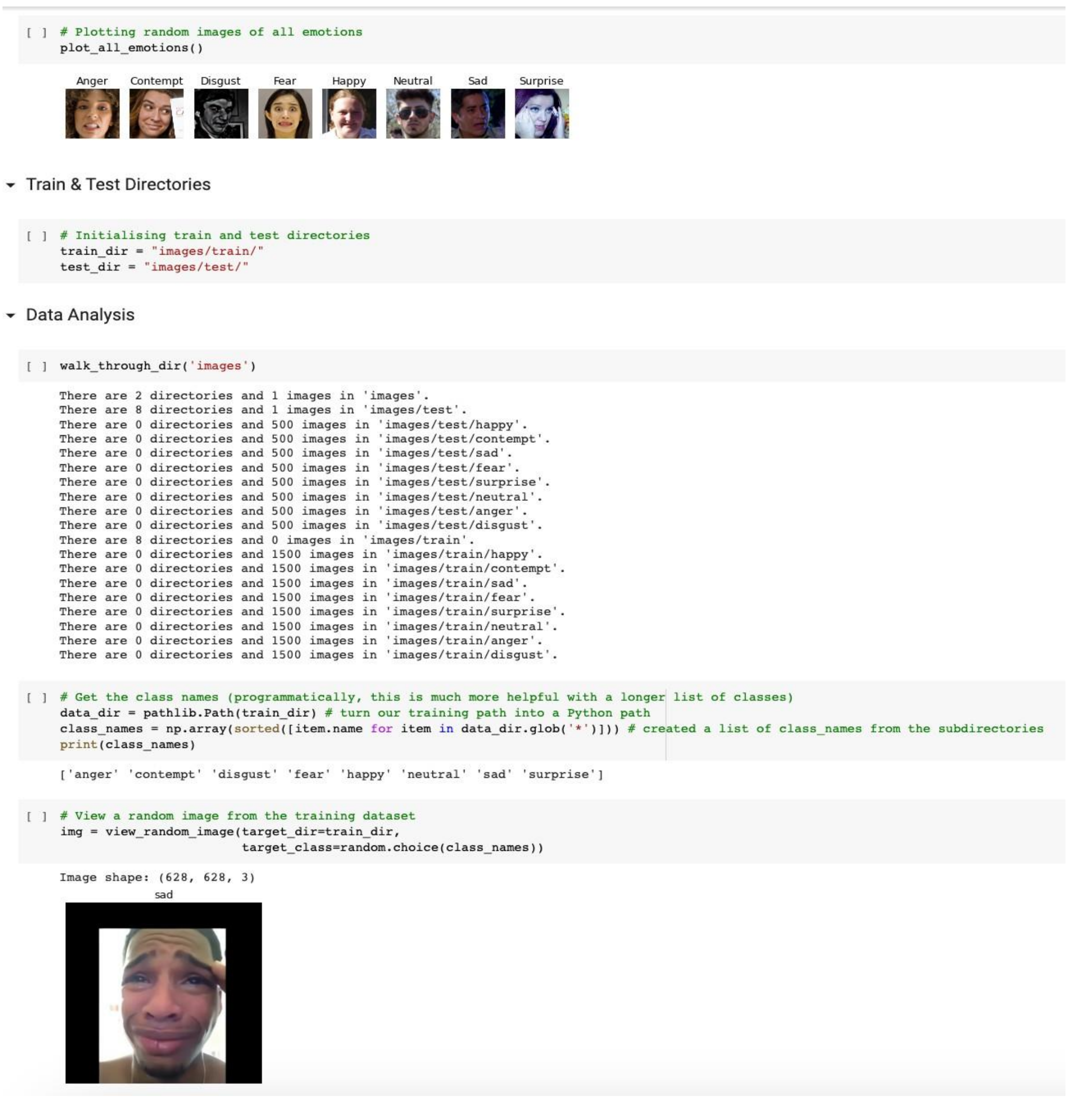
Conclusion and Future Work

We constructed a convolutional neural network to identify emotion from the AffectNet dataset in this project. Conv-ReLU-Pool architecture and softmax loss were employed. We used Adam with learning rate decay to optimise the network.

Using this architecture, we were able to achieve a final validation accuracy of 65.74%. The images are so small that it can sometimes be difficult, even for humans, to tell which emotion is depicted in each one, which makes improving this a challenge. We used saliency maps to identify key areas in the images that the neural net considered significant in order to understand how the neural net classified various images. Even though the majority of the results were quite erratic, some images provided compelling evidence.
Despite the fact that our validation accuracy is fairly high, we think that the network would be further enhanced by the addition of additional layers and filters. It would be interesting to try this strategy in the future, along with creating a classifier using pre- trained nets.

Appendices

Exploratory Data Analysis



```
[ ] # Plotting random images of all emotions
    plot_all_emotions()
```

▾ Train & Test Directories

```
[ ] # Initialising train and test directories
    train_dir = "images/train/"
    test_dir = "images/test/"
```

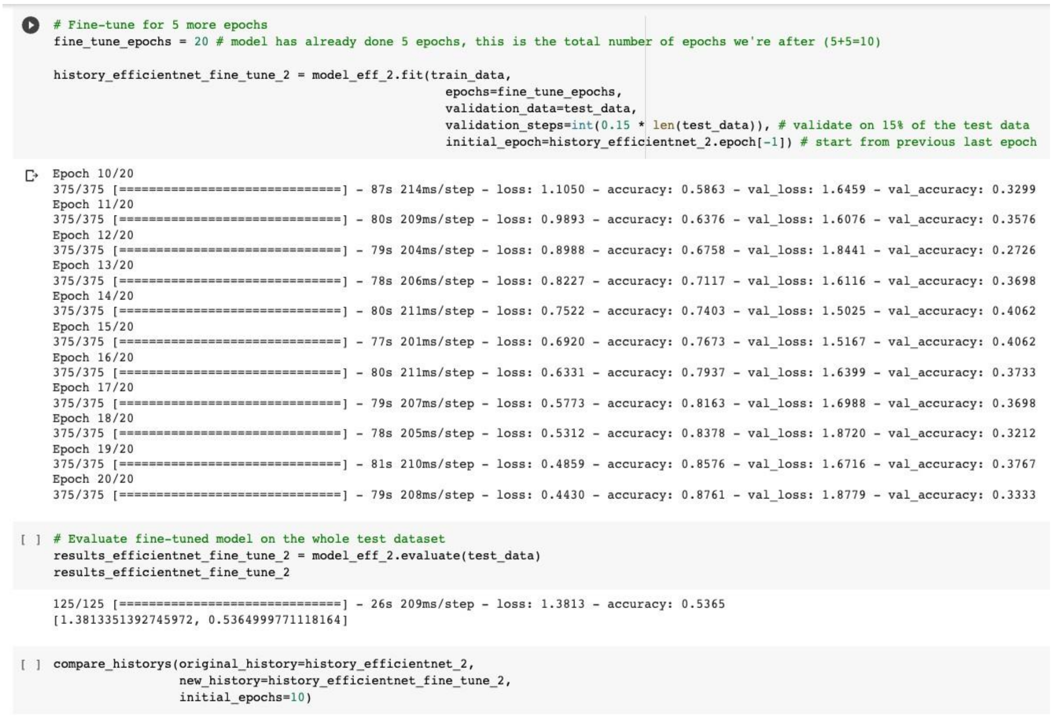
▾ Data Analysis

```
[ ] walk_through_dir('images')

    There are 2 directories and 1 images in 'images'.
    There are 8 directories and 1 images in 'images/test'.
    There are 0 directories and 500 images in 'images/test/happy'.
    There are 0 directories and 500 images in 'images/test/contempt'.
    There are 0 directories and 500 images in 'images/test/sad'.
    There are 0 directories and 500 images in 'images/test/fear'.
    There are 0 directories and 500 images in 'images/test/surprise'.
    There are 0 directories and 500 images in 'images/test/neutral'.
    There are 0 directories and 500 images in 'images/test/anger'.
    There are 0 directories and 500 images in 'images/test/disgust'.
    There are 8 directories and 0 images in 'images/train'.
    There are 0 directories and 1500 images in 'images/train/happy'.
    There are 0 directories and 1500 images in 'images/train/contempt'.
    There are 0 directories and 1500 images in 'images/train/sad'.
    There are 0 directories and 1500 images in 'images/train/fear'.
    There are 0 directories and 1500 images in 'images/train/surprise'.
    There are 0 directories and 1500 images in 'images/train/neutral'.
    There are 0 directories and 1500 images in 'images/train/anger'.
    There are 0 directories and 1500 images in 'images/train/disgust'.
```

```
[ ] # Get the class names (programmatically, this is much more helpful with a longer list of classes)
    data_dir = pathlib.Path(train_dir) # turn our training path into a Python path
    class_names = np.array(sorted([item.name for item in data_dir.glob('*')])) # created a list of class_names from the subdirectories
    print(class_names)

    ['anger' 'contempt' 'disgust' 'fear' 'happy' 'neutral' 'sad' 'surprise']
```

```
[ ] # View a random image from the training dataset
    img = view_random_image(target_dir=train_dir,
                            target_class=random.choice(class_names))

    Image shape: (628, 628, 3)
```

Transfer Learning ModelEfficientNetB0



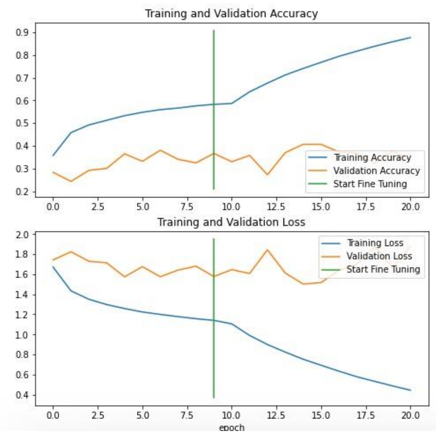
```
# Fine-tune for 5 more epochs
fine_tune_epochs = 20 # model has already done 5 epochs, this is the total number of epochs we're after (5+5=10)

history_efficientnet_fine_tune_2 = model_eff_2.fit(train_data,
                                                   epochs=fine_tune_epochs,
                                                   validation_data=test_data,
                                                   validation_steps=int(0.15 * len(test_data)), # validate on 15% of the test data
                                                   initial_epoch=history_efficientnet_2.epoch[-1]) # start from previous last epoch
```

```
Epoch 10/20
375/375 [==============================] - 87s 214ms/step - loss: 1.1050 - accuracy: 0.5863 - val_loss: 1.6459 - val_accuracy: 0.3299
Epoch 11/20
375/375 [==============================] - 80s 209ms/step - loss: 0.9893 - accuracy: 0.6376 - val_loss: 1.6076 - val_accuracy: 0.3576
Epoch 12/20
375/375 [==============================] - 79s 204ms/step - loss: 0.8988 - accuracy: 0.6758 - val_loss: 1.8441 - val_accuracy: 0.2726
Epoch 13/20
375/375 [==============================] - 78s 206ms/step - loss: 0.8227 - accuracy: 0.7117 - val_loss: 1.6116 - val_accuracy: 0.3698
Epoch 14/20
375/375 [==============================] - 80s 211ms/step - loss: 0.7522 - accuracy: 0.7403 - val_loss: 1.5025 - val_accuracy: 0.4062
Epoch 15/20
375/375 [==============================] - 77s 201ms/step - loss: 0.6920 - accuracy: 0.7673 - val_loss: 1.5167 - val_accuracy: 0.4062
Epoch 16/20
375/375 [==============================] - 80s 211ms/step - loss: 0.6331 - accuracy: 0.7937 - val_loss: 1.6399 - val_accuracy: 0.3733
Epoch 17/20
375/375 [==============================] - 79s 207ms/step - loss: 0.5773 - accuracy: 0.8163 - val_loss: 1.6988 - val_accuracy: 0.3698
Epoch 18/20
375/375 [==============================] - 78s 205ms/step - loss: 0.5312 - accuracy: 0.8378 - val_loss: 1.8720 - val_accuracy: 0.3212
Epoch 19/20
375/375 [==============================] - 81s 210ms/step - loss: 0.4859 - accuracy: 0.8576 - val_loss: 1.6716 - val_accuracy: 0.3767
Epoch 20/20
375/375 [==============================] - 79s 208ms/step - loss: 0.4430 - accuracy: 0.8761 - val_loss: 1.8779 - val_accuracy: 0.3333
```

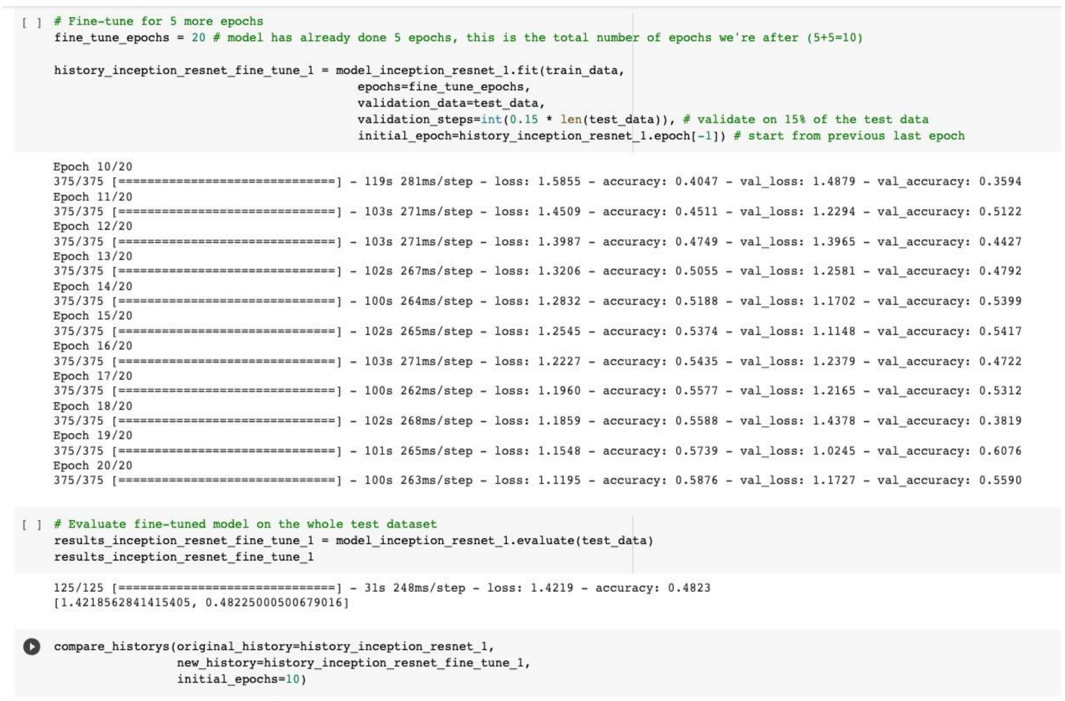
```
[ ] # Evaluate fine-tuned model on the whole test dataset
    results_efficientnet_fine_tune_2 = model_eff_2.evaluate(test_data)
    results_efficientnet_fine_tune_2
```

```
125/125 [==============================] - 26s 209ms/step - loss: 1.3813 - accuracy: 0.5365
[1.3813351392745972, 0.5364999771118164]
```

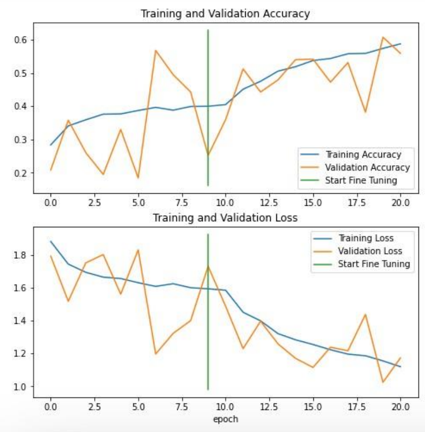
```
[ ] compare_historys(original_history=history_efficientnet_2,
                     new_history=history_efficientnet_fine_tune_2,
                     initial_epochs=10)
```



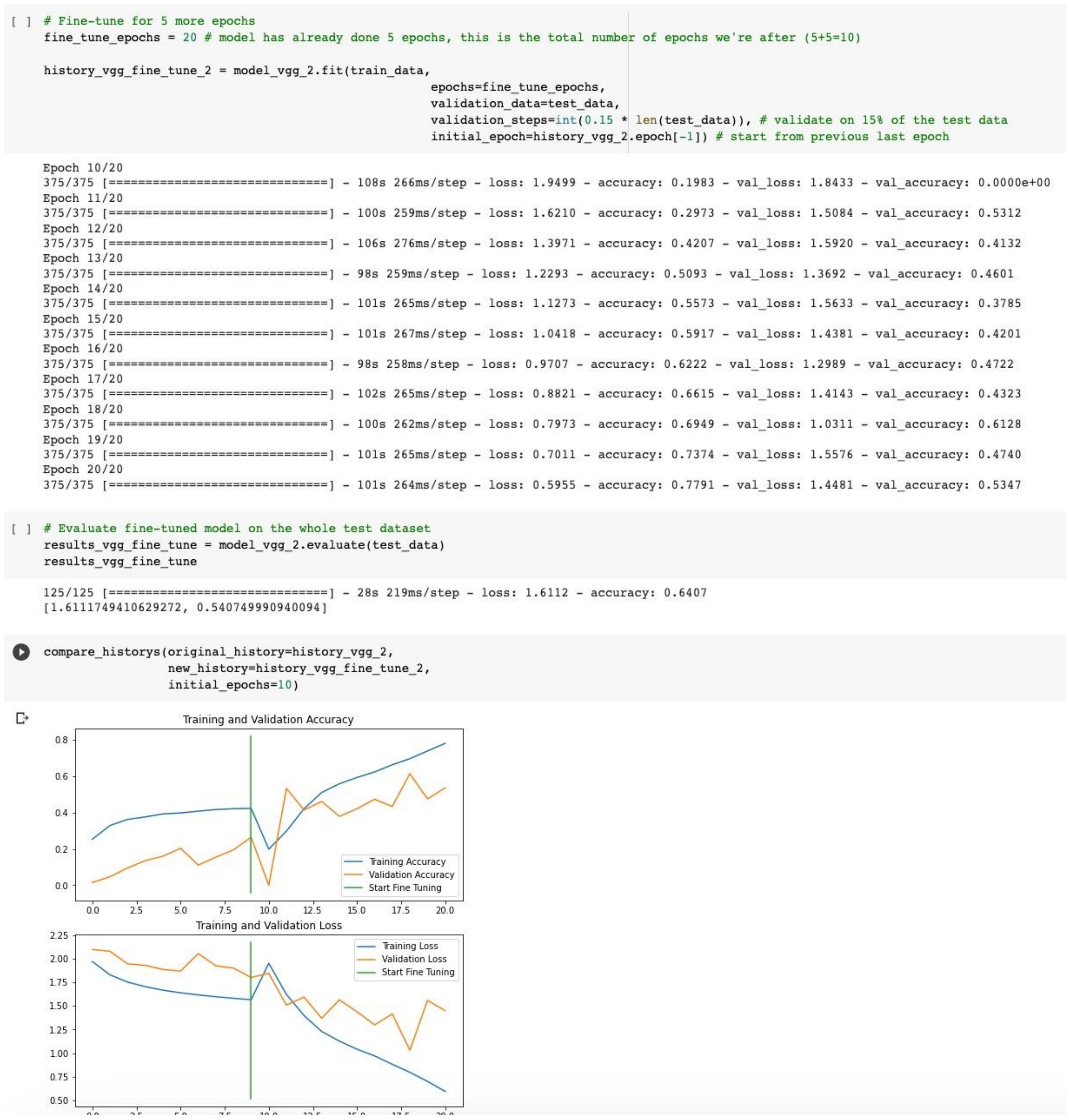
InceptionResNetV2



```
[ ] # Fine-tune for 5 more epochs
    fine_tune_epochs = 20 # model has already done 5 epochs, this is the total number of epochs we're after (5+5=10)

    history_inception_resnet_fine_tune_1 = model_inception_resnet_1.fit(train_data,
                                          epochs=fine_tune_epochs,
                                          validation_data=test_data,
                                          validation_steps=int(0.15 * len(test_data)), # validate on 15% of the test data
                                          initial_epoch=history_inception_resnet_1.epoch[-1]) # start from previous last epoch

    Epoch 10/20
    375/375 [==============================] - 119s 281ms/step - loss: 1.5855 - accuracy: 0.4047 - val_loss: 1.4879 - val_accuracy: 0.3594
    Epoch 11/20
    375/375 [==============================] - 103s 271ms/step - loss: 1.4509 - accuracy: 0.4511 - val_loss: 1.2294 - val_accuracy: 0.5122
    Epoch 12/20
    375/375 [==============================] - 103s 271ms/step - loss: 1.3987 - accuracy: 0.4749 - val_loss: 1.3965 - val_accuracy: 0.4427
    Epoch 13/20
    375/375 [==============================] - 102s 267ms/step - loss: 1.3206 - accuracy: 0.5055 - val_loss: 1.2581 - val_accuracy: 0.4792
    Epoch 14/20
    375/375 [==============================] - 100s 264ms/step - loss: 1.2832 - accuracy: 0.5188 - val_loss: 1.1702 - val_accuracy: 0.5399
    Epoch 15/20
    375/375 [==============================] - 102s 265ms/step - loss: 1.2545 - accuracy: 0.5374 - val_loss: 1.1148 - val_accuracy: 0.5417
    Epoch 16/20
    375/375 [==============================] - 103s 271ms/step - loss: 1.2227 - accuracy: 0.5435 - val_loss: 1.2379 - val_accuracy: 0.4722
    Epoch 17/20
    375/375 [==============================] - 100s 262ms/step - loss: 1.1960 - accuracy: 0.5577 - val_loss: 1.2165 - val_accuracy: 0.5312
    Epoch 18/20
    375/375 [==============================] - 102s 268ms/step - loss: 1.1859 - accuracy: 0.5588 - val_loss: 1.4378 - val_accuracy: 0.3819
    Epoch 19/20
    375/375 [==============================] - 101s 265ms/step - loss: 1.1548 - accuracy: 0.5739 - val_loss: 1.0245 - val_accuracy: 0.6076
    Epoch 20/20
    375/375 [==============================] - 100s 263ms/step - loss: 1.1195 - accuracy: 0.5876 - val_loss: 1.1727 - val_accuracy: 0.5590

[ ] # Evaluate fine-tuned model on the whole test dataset
    results_inception_resnet_fine_tune_1 = model_inception_resnet_1.evaluate(test_data)
    results_inception_resnet_fine_tune_1

    125/125 [==============================] - 31s 248ms/step - loss: 1.4219 - accuracy: 0.4823
    [1.4218562841415405, 0.48225000500679016]

    compare_historys(original_history=history_inception_resnet_1,
                     new_history=history_inception_resnet_fine_tune_1,
                     initial_epochs=10)
```



VGG16



```
[ ] # Fine-tune for 5 more epochs
    fine_tune_epochs = 20 # model has already done 5 epochs, this is the total number of epochs we're after (5+5=10)

    history_vgg_fine_tune_2 = model_vgg_2.fit(train_data,
                                                     epochs=fine_tune_epochs,
                                                     validation_data=test_data,
                                                     validation_steps=int(0.15 * len(test_data)), # validate on 15% of the test data
                                                     initial_epoch=history_vgg_2.epoch[-1]) # start from previous last epoch
```

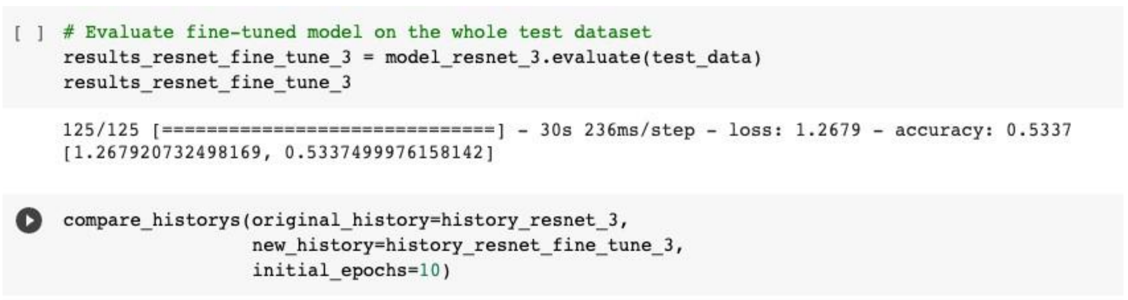
```
Epoch 10/20
375/375 [==============================] - 108s 266ms/step - loss: 1.9499 - accuracy: 0.1983 - val_loss: 1.8433 - val_accuracy: 0.0000e+00
Epoch 11/20
375/375 [==============================] - 100s 259ms/step - loss: 1.6210 - accuracy: 0.2973 - val_loss: 1.5084 - val_accuracy: 0.5312
Epoch 12/20
375/375 [==============================] - 106s 276ms/step - loss: 1.3971 - accuracy: 0.4207 - val_loss: 1.5920 - val_accuracy: 0.4132
Epoch 13/20
375/375 [==============================] - 98s 259ms/step - loss: 1.2293 - accuracy: 0.5093 - val_loss: 1.3692 - val_accuracy: 0.4601
Epoch 14/20
375/375 [==============================] - 101s 265ms/step - loss: 1.1273 - accuracy: 0.5573 - val_loss: 1.5633 - val_accuracy: 0.3785
Epoch 15/20
375/375 [==============================] - 101s 267ms/step - loss: 1.0418 - accuracy: 0.5917 - val_loss: 1.4381 - val_accuracy: 0.4201
Epoch 16/20
375/375 [==============================] - 98s 258ms/step - loss: 0.9707 - accuracy: 0.6222 - val_loss: 1.2989 - val_accuracy: 0.4722
Epoch 17/20
375/375 [==============================] - 102s 265ms/step - loss: 0.8821 - accuracy: 0.6615 - val_loss: 1.4143 - val_accuracy: 0.4323
Epoch 18/20
375/375 [==============================] - 100s 262ms/step - loss: 0.7973 - accuracy: 0.6949 - val_loss: 1.0311 - val_accuracy: 0.6128
Epoch 19/20
375/375 [==============================] - 101s 265ms/step - loss: 0.7011 - accuracy: 0.7374 - val_loss: 1.5576 - val_accuracy: 0.4740
Epoch 20/20
375/375 [==============================] - 101s 264ms/step - loss: 0.5955 - accuracy: 0.7791 - val_loss: 1.4481 - val_accuracy: 0.5347
```

```
[ ] # Evaluate fine-tuned model on the whole test dataset
    results_vgg_fine_tune = model_vgg_2.evaluate(test_data)
    results_vgg_fine_tune
```

```
125/125 [==============================] - 28s 219ms/step - loss: 1.6112 - accuracy: 0.6407
[1.6111749410629272, 0.540749990940094]
```

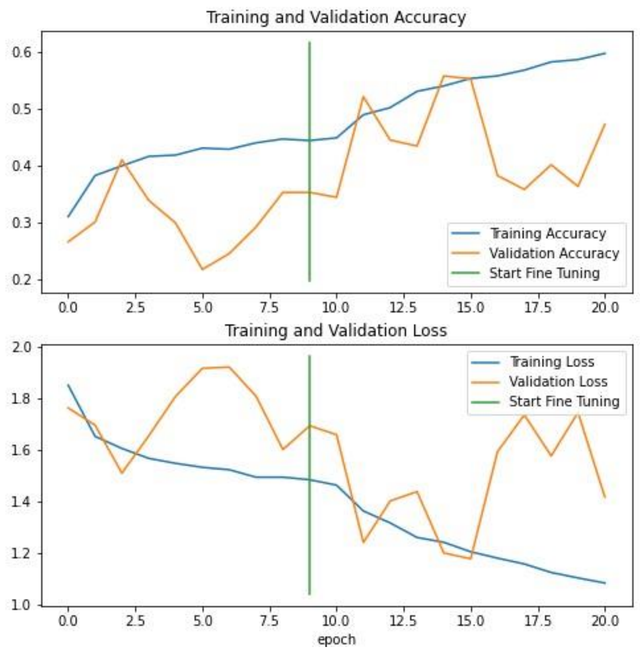
```
compare_historys(original_history=history_vgg_2,
                 new_history=history_vgg_fine_tune_2,
                 initial_epochs=10)
```

ResNet101V2

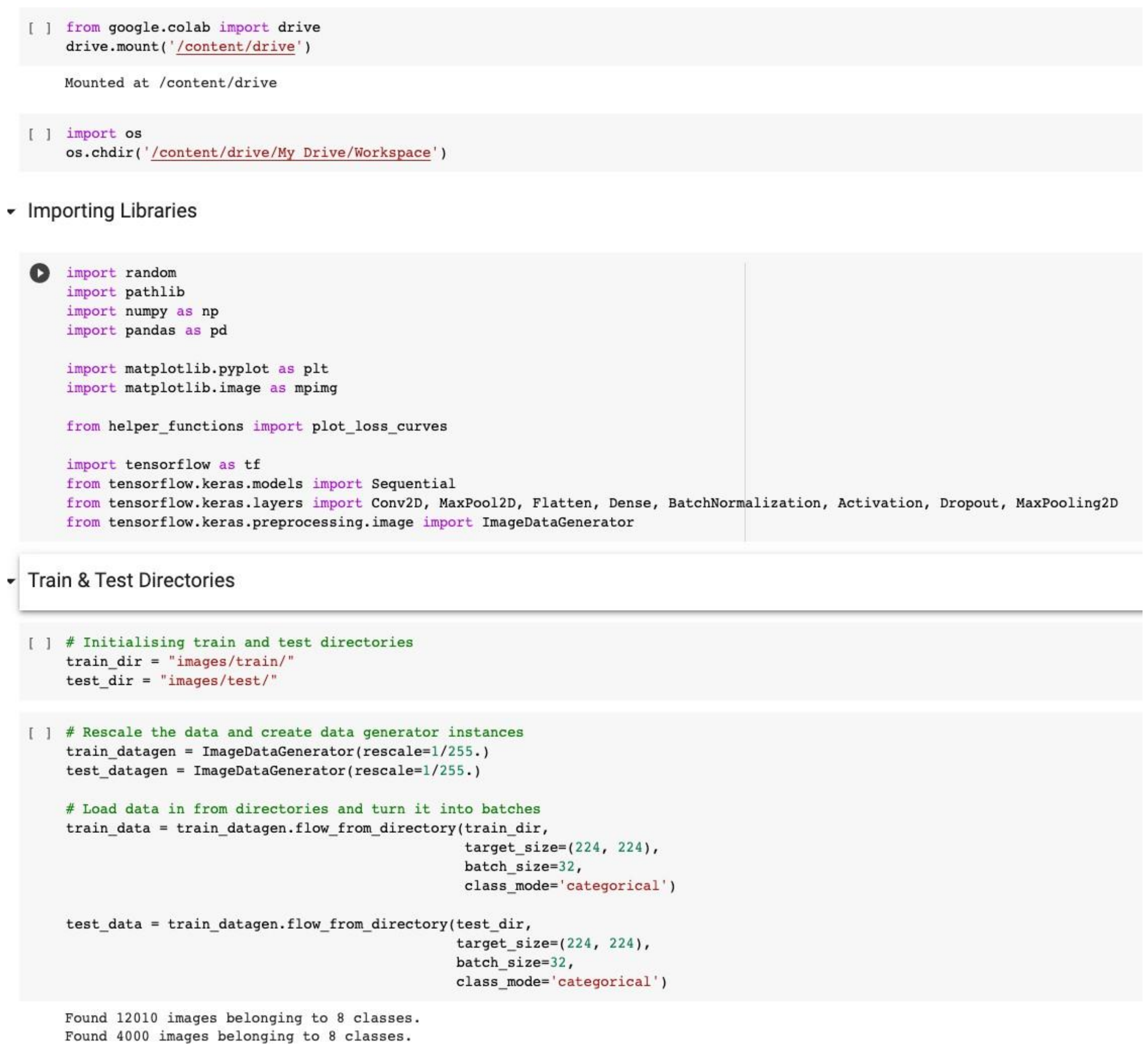


```
[ ] # Evaluate fine-tuned model on the whole test dataset
    results_resnet_fine_tune_3 = model_resnet_3.evaluate(test_data)
    results_resnet_fine_tune_3

    125/125 [==============================] - 30s 236ms/step - loss: 1.2679 - accuracy: 0.5337
    [1.267920732498169, 0.5337499976158142]
```

```
compare_historys(original_history=history_resnet_3,
                 new_history=history_resnet_fine_tune_3,
                 initial_epochs=10)
```



Initialising Basic Models for FER



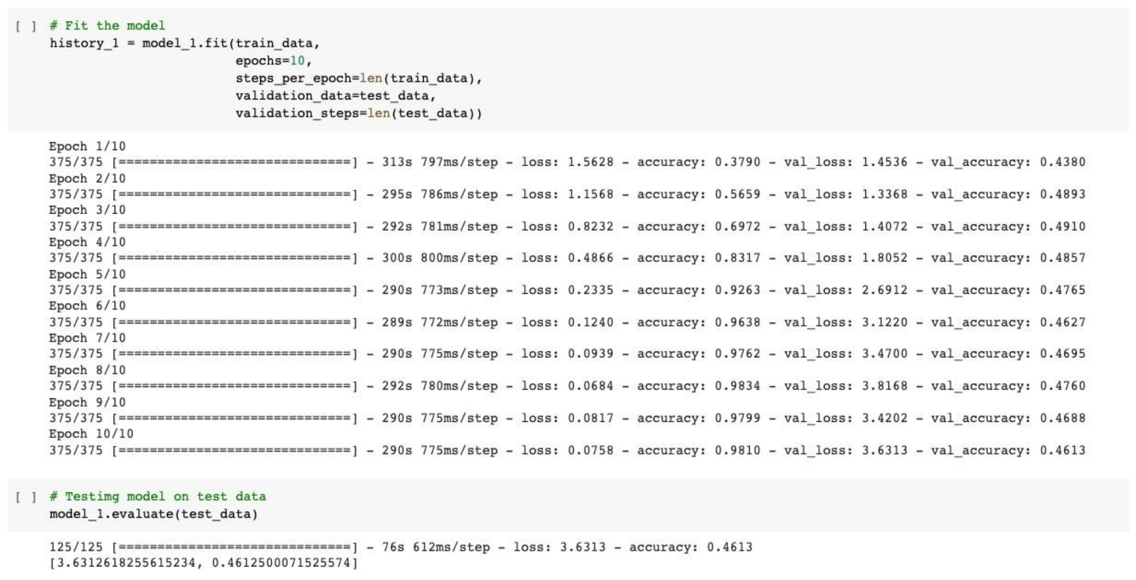
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] import os
    os.chdir('/content/drive/My Drive/Workspace')
```

Importing Libraries

```
import random
import pathlib
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from helper_functions import plot_loss_curves

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, BatchNormalization, Activation, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

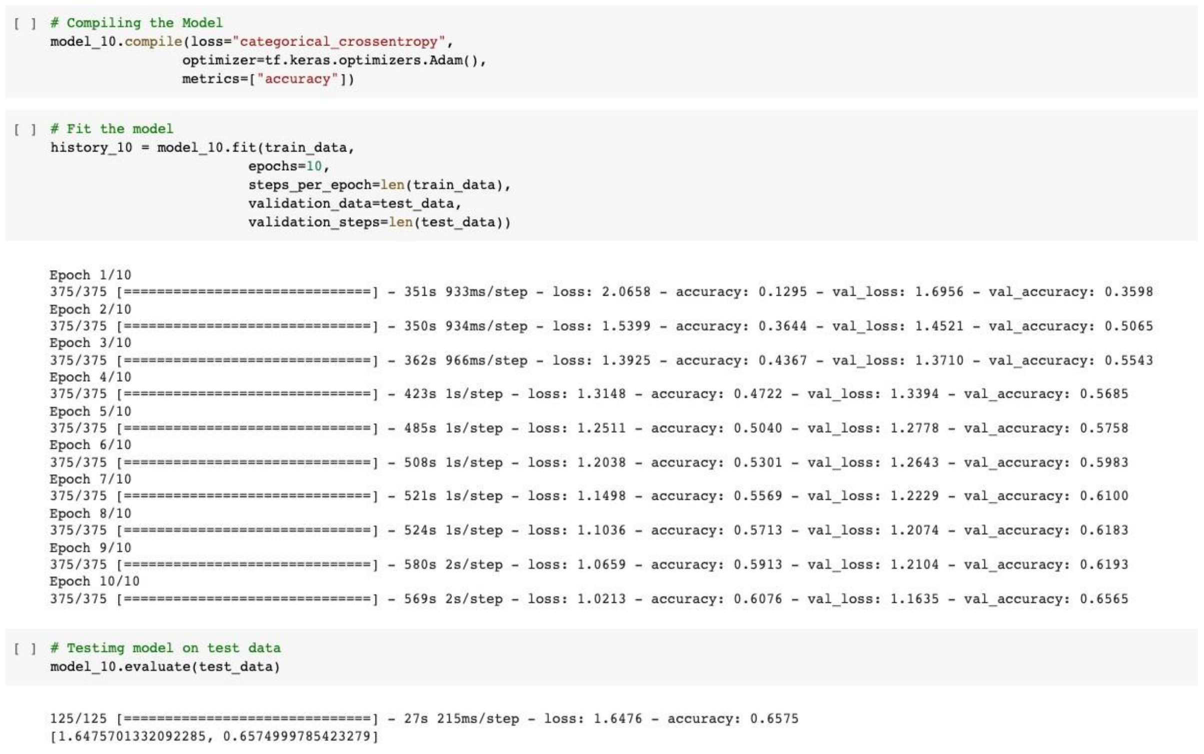
Train & Test Directories

```
[ ] # Initialising train and test directories
    train_dir = "images/train/"
    test_dir = "images/test/"

[ ] # Rescale the data and create data generator instances
    train_datagen = ImageDataGenerator(rescale=1/255.)
    test_datagen = ImageDataGenerator(rescale=1/255.)

    # Load data in from directories and turn it into batches
    train_data = train_datagen.flow_from_directory(train_dir,
                                                   target_size=(224, 224),
                                                   batch_size=32,
                                                   class_mode='categorical')

    test_data = train_datagen.flow_from_directory(test_dir,
                                                  target_size=(224, 224),
                                                  batch_size=32,
                                                  class_mode='categorical')

    Found 12010 images belonging to 8 classes.
    Found 4000 images belonging to 8 classes.
```



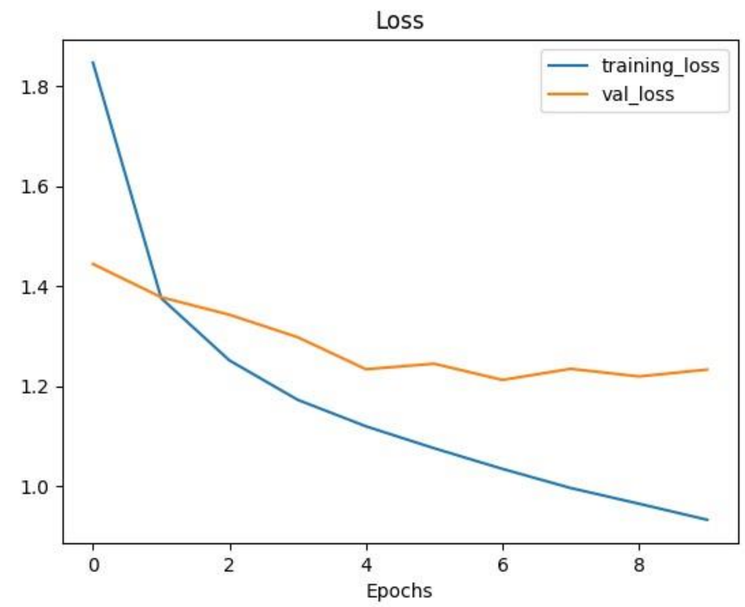
```
[ ] # Fit the model
    history_1 = model_1.fit(train_data,
                            epochs=10,
                            steps_per_epoch=len(train_data),
                            validation_data=test_data,
                            validation_steps=len(test_data))

    Epoch 1/10
    375/375 [==============================] - 313s 797ms/step - loss: 1.5628 - accuracy: 0.3790 - val_loss: 1.4536 - val_accuracy: 0.4380
    Epoch 2/10
    375/375 [==============================] - 295s 786ms/step - loss: 1.1568 - accuracy: 0.5659 - val_loss: 1.3368 - val_accuracy: 0.4893
    Epoch 3/10
    375/375 [==============================] - 292s 781ms/step - loss: 0.8232 - accuracy: 0.6972 - val_loss: 1.4072 - val_accuracy: 0.4910
    Epoch 4/10
    375/375 [==============================] - 300s 800ms/step - loss: 0.4866 - accuracy: 0.8317 - val_loss: 1.8052 - val_accuracy: 0.4857
    Epoch 5/10
    375/375 [==============================] - 290s 773ms/step - loss: 0.2335 - accuracy: 0.9263 - val_loss: 2.6912 - val_accuracy: 0.4765
    Epoch 6/10
    375/375 [==============================] - 289s 772ms/step - loss: 0.1240 - accuracy: 0.9638 - val_loss: 3.1220 - val_accuracy: 0.4627
    Epoch 7/10
    375/375 [==============================] - 290s 775ms/step - loss: 0.0939 - accuracy: 0.9762 - val_loss: 3.4700 - val_accuracy: 0.4695
    Epoch 8/10
    375/375 [==============================] - 292s 780ms/step - loss: 0.0684 - accuracy: 0.9834 - val_loss: 3.8168 - val_accuracy: 0.4760
    Epoch 9/10
    375/375 [==============================] - 290s 775ms/step - loss: 0.0817 - accuracy: 0.9799 - val_loss: 3.4202 - val_accuracy: 0.4688
    Epoch 10/10
    375/375 [==============================] - 290s 775ms/step - loss: 0.0758 - accuracy: 0.9810 - val_loss: 3.6313 - val_accuracy: 0.4613

[ ] # Testimg model on test data
    model_1.evaluate(test_data)

    125/125 [==============================] - 76s 612ms/step - loss: 3.6313 - accuracy: 0.4613
    [3.6312618255615234, 0.4612500071525574]
```

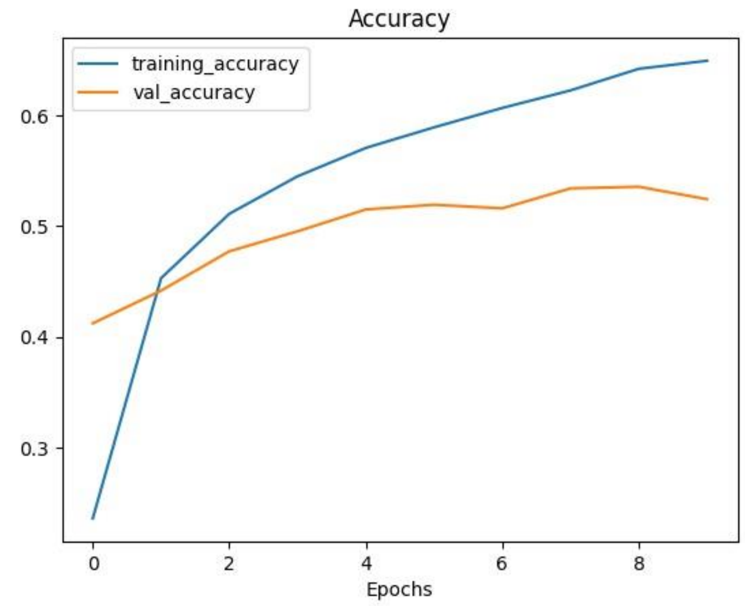
Proposed Model



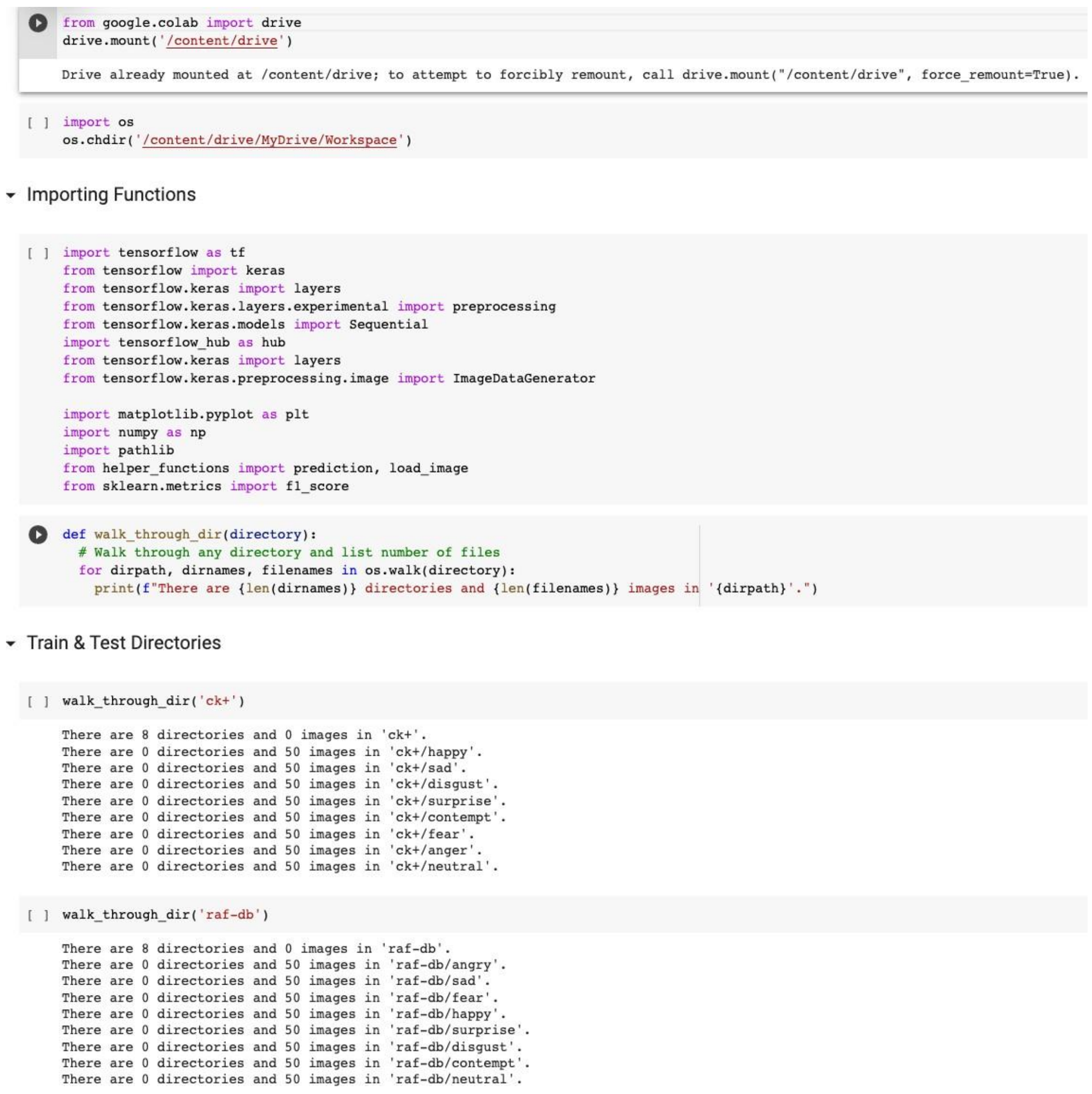
```
[ ] # Compiling the Model
    model_10.compile(loss="categorical_crossentropy",
                     optimizer=tf.keras.optimizers.Adam(),
                     metrics=["accuracy"])

[ ] # Fit the model
    history_10 = model_10.fit(train_data,
                              epochs=10,
                              steps_per_epoch=len(train_data),
                              validation_data=test_data,
                              validation_steps=len(test_data))

    Epoch 1/10
    375/375 [==============================] - 351s 933ms/step - loss: 2.0658 - accuracy: 0.1295 - val_loss: 1.6956 - val_accuracy: 0.3598
    Epoch 2/10
    375/375 [==============================] - 350s 934ms/step - loss: 1.5399 - accuracy: 0.3644 - val_loss: 1.4521 - val_accuracy: 0.5065
    Epoch 3/10
    375/375 [==============================] - 362s 966ms/step - loss: 1.3925 - accuracy: 0.4367 - val_loss: 1.3710 - val_accuracy: 0.5543
    Epoch 4/10
    375/375 [==============================] - 423s 1s/step - loss: 1.3148 - accuracy: 0.4722 - val_loss: 1.3394 - val_accuracy: 0.5685
    Epoch 5/10
    375/375 [==============================] - 485s 1s/step - loss: 1.2511 - accuracy: 0.5040 - val_loss: 1.2778 - val_accuracy: 0.5758
    Epoch 6/10
    375/375 [==============================] - 508s 1s/step - loss: 1.2038 - accuracy: 0.5301 - val_loss: 1.2643 - val_accuracy: 0.5983
    Epoch 7/10
    375/375 [==============================] - 521s 1s/step - loss: 1.1498 - accuracy: 0.5569 - val_loss: 1.2229 - val_accuracy: 0.6100
    Epoch 8/10
    375/375 [==============================] - 524s 1s/step - loss: 1.1036 - accuracy: 0.5713 - val_loss: 1.2074 - val_accuracy: 0.6183
    Epoch 9/10
    375/375 [==============================] - 580s 2s/step - loss: 1.0659 - accuracy: 0.5913 - val_loss: 1.2104 - val_accuracy: 0.6193
    Epoch 10/10
    375/375 [==============================] - 569s 2s/step - loss: 1.0213 - accuracy: 0.6076 - val_loss: 1.1635 - val_accuracy: 0.6565

[ ] # Testimg model on test data
    model_10.evaluate(test_data)

    125/125 [==============================] - 27s 215ms/step - loss: 1.6476 - accuracy: 0.6575
    [1.6475701332092285, 0.6574999785423279]
```

```
plot_loss_curves(history_10)
```





Testing Proposed Model on CK+, RAF-DB and EMOTIC Dataset



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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
os.chdir('/content/drive/MyDrive/Workspace')
```

Importing Functions

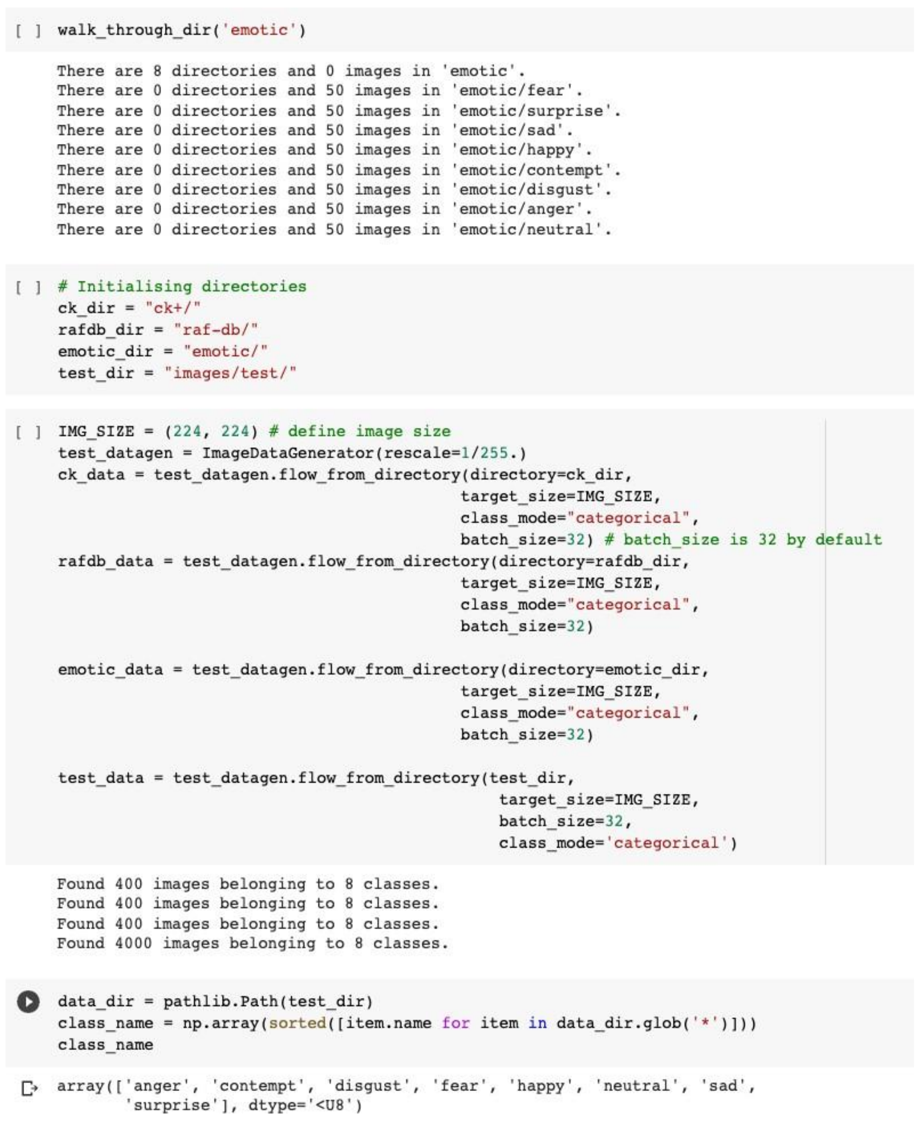
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.models import Sequential
import tensorflow_hub as hub
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt
import numpy as np
import pathlib
from helper_functions import prediction, load_image
from sklearn.metrics import f1_score
```

```
def walk_through_dir(directory):
  # Walk through any directory and list number of files
  for dirpath, dirnames, filenames in os.walk(directory):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

Train & Test Directories

```
walk_through_dir('ck+')

There are 8 directories and 0 images in 'ck+'.
There are 0 directories and 50 images in 'ck+/happy'.
There are 0 directories and 50 images in 'ck+/sad'.
There are 0 directories and 50 images in 'ck+/disgust'.
There are 0 directories and 50 images in 'ck+/surprise'.
There are 0 directories and 50 images in 'ck+/contempt'.
There are 0 directories and 50 images in 'ck+/fear'.
There are 0 directories and 50 images in 'ck+/anger'.
There are 0 directories and 50 images in 'ck+/neutral'.
```

```
walk_through_dir('raf-db')

There are 8 directories and 0 images in 'raf-db'.
There are 0 directories and 50 images in 'raf-db/angry'.
There are 0 directories and 50 images in 'raf-db/sad'.
There are 0 directories and 50 images in 'raf-db/fear'.
There are 0 directories and 50 images in 'raf-db/happy'.
There are 0 directories and 50 images in 'raf-db/surprise'.
There are 0 directories and 50 images in 'raf-db/disgust'.
There are 0 directories and 50 images in 'raf-db/contempt'.
There are 0 directories and 50 images in 'raf-db/neutral'.
```

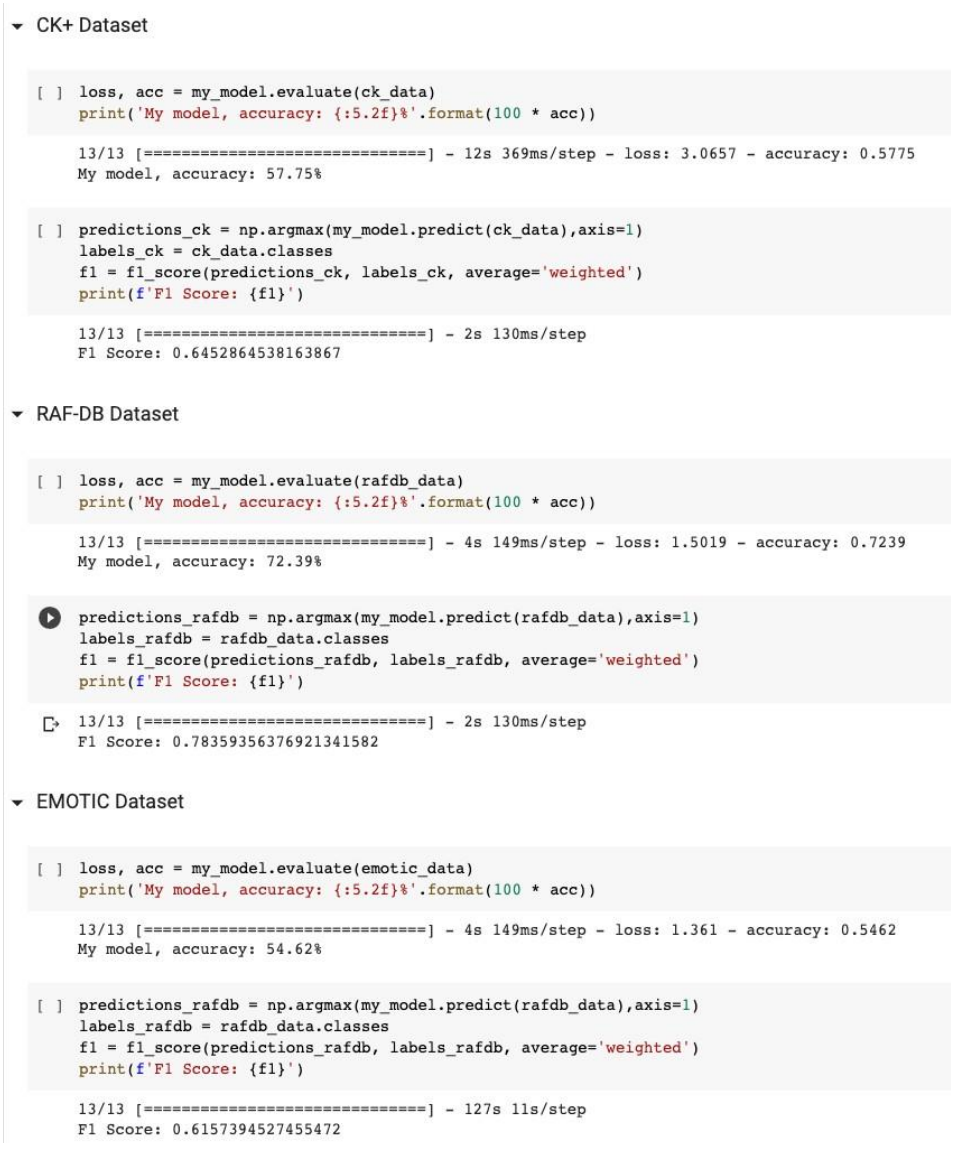


```
[ ] walk_through_dir('emotic')

    There are 8 directories and 0 images in 'emotic'.
    There are 0 directories and 50 images in 'emotic/fear'.
    There are 0 directories and 50 images in 'emotic/surprise'.
    There are 0 directories and 50 images in 'emotic/sad'.
    There are 0 directories and 50 images in 'emotic/happy'.
    There are 0 directories and 50 images in 'emotic/contempt'.
    There are 0 directories and 50 images in 'emotic/disgust'.
    There are 0 directories and 50 images in 'emotic/anger'.
    There are 0 directories and 50 images in 'emotic/neutral'.
```

```
[ ] # Initialising directories
    ck_dir = "ck+/"
    rafdb_dir = "raf-db/"
    emotic_dir = "emotic/"
    test_dir = "images/test/"
```

```
[ ] IMG_SIZE = (224, 224) # define image size
    test_datagen = ImageDataGenerator(rescale=1/255.)
    ck_data = test_datagen.flow_from_directory(directory=ck_dir,
                                              target_size=IMG_SIZE,
                                              class_mode="categorical",
                                              batch_size=32) # batch_size is 32 by default
    rafdb_data = test_datagen.flow_from_directory(directory=rafdb_dir,
                                              target_size=IMG_SIZE,
                                              class_mode="categorical",
                                              batch_size=32)

    emotic_data = test_datagen.flow_from_directory(directory=emotic_dir,
                                              target_size=IMG_SIZE,
                                              class_mode="categorical",
                                              batch_size=32)

    test_data = test_datagen.flow_from_directory(test_dir,
                                                  target_size=IMG_SIZE,
                                                  batch_size=32,
                                                  class_mode='categorical')

    Found 400 images belonging to 8 classes.
    Found 400 images belonging to 8 classes.
    Found 400 images belonging to 8 classes.
    Found 4000 images belonging to 8 classes.
```

```
    data_dir = pathlib.Path(test_dir)
    class_name = np.array(sorted([item.name for item in data_dir.glob('*')]))
    class_name

    array(['anger', 'contempt', 'disgust', 'fear', 'happy', 'neutral', 'sad',
           'surprise'], dtype='<U8')
```



CK+ Dataset

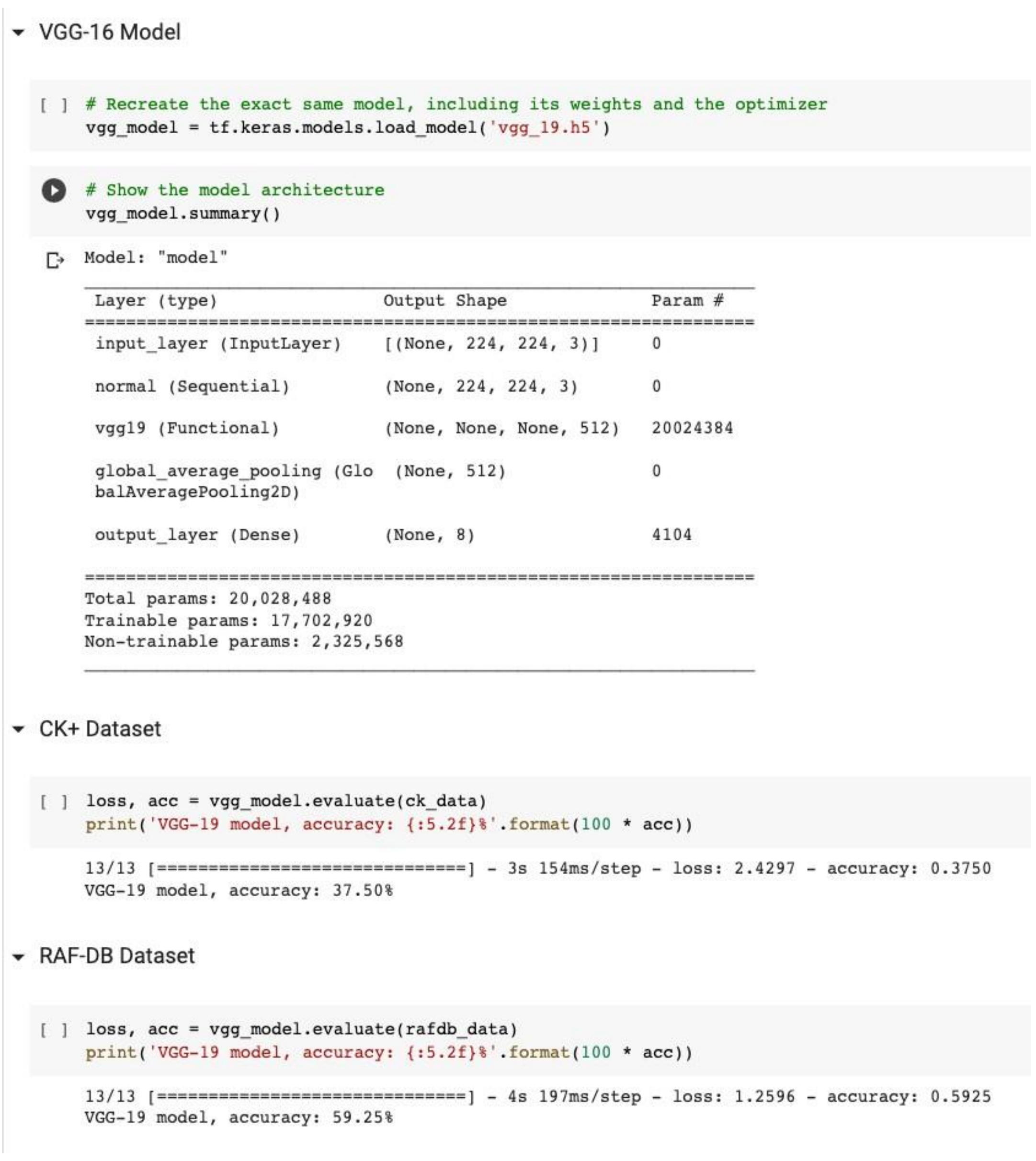
```
[ ] loss, acc = my_model.evaluate(ck_data)
    print('My model, accuracy: {:5.2f}%'.format(100 * acc))

    13/13 [==============================] - 12s 369ms/step - loss: 3.0657 - accuracy: 0.5775
    My model, accuracy: 57.75%

[ ] predictions_ck = np.argmax(my_model.predict(ck_data),axis=1)
    labels_ck = ck_data.classes
    f1 = f1_score(predictions_ck, labels_ck, average='weighted')
    print(f'F1 Score: {f1}')

    13/13 [==============================] - 2s 130ms/step
    F1 Score: 0.6452864538163867
```

RAF-DB Dataset

```
[ ] loss, acc = my_model.evaluate(rafdb_data)
    print('My model, accuracy: {:5.2f}%'.format(100 * acc))

    13/13 [==============================] - 4s 149ms/step - loss: 1.5019 - accuracy: 0.7239
    My model, accuracy: 72.39%

    predictions_rafdb = np.argmax(my_model.predict(rafdb_data),axis=1)
    labels_rafdb = rafdb_data.classes
    f1 = f1_score(predictions_rafdb, labels_rafdb, average='weighted')
    print(f'F1 Score: {f1}')

    13/13 [==============================] - 2s 130ms/step
    F1 Score: 0.78359356376921341582
```

EMOTIC Dataset

```
[ ] loss, acc = my_model.evaluate(emotic_data)
    print('My model, accuracy: {:5.2f}%'.format(100 * acc))

    13/13 [==============================] - 4s 149ms/step - loss: 1.361 - accuracy: 0.5462
    My model, accuracy: 54.62%

[ ] predictions_rafdb = np.argmax(my_model.predict(rafdb_data),axis=1)
    labels_rafdb = rafdb_data.classes
    f1 = f1_score(predictions_rafdb, labels_rafdb, average='weighted')
    print(f'F1 Score: {f1}')

    13/13 [==============================] - 127s 11s/step
    F1 Score: 0.61573945274554472
```



VGG-16 Model

```
[ ] # Recreate the exact same model, including its weights and the optimizer
    vgg_model = tf.keras.models.load_model('vgg_19.h5')
```

```
# Show the model architecture
vgg_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 normal (Sequential)         (None, 224, 224, 3)       0

 vgg19 (Functional)          (None, None, None, 512)   20024384

 global_average_pooling (Glo  (None, 512)              0
 balAveragePooling2D)

 output_layer (Dense)        (None, 8)                 4104

=================================================================
Total params: 20,028,488
Trainable params: 17,702,920
Non-trainable params: 2,325,568
_________________________________________________________________
```

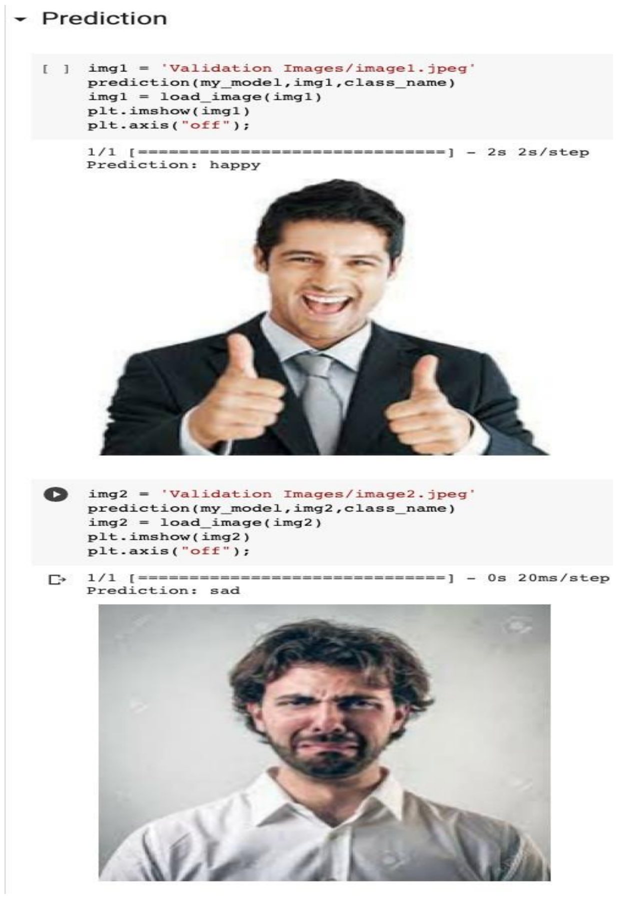
CK+ Dataset

```
[ ] loss, acc = vgg_model.evaluate(ck_data)
    print('VGG-19 model, accuracy: {:5.2f}%'.format(100 * acc))

    13/13 [==============================] - 3s 154ms/step - loss: 2.4297 - accuracy: 0.3750
    VGG-19 model, accuracy: 37.50%
```

RAF-DB Dataset

```
[ ] loss, acc = vgg_model.evaluate(rafdb_data)
    print('VGG-19 model, accuracy: {:5.2f}%'.format(100 * acc))

    13/13 [==============================] - 4s 197ms/step - loss: 1.2596 - accuracy: 0.5925
    VGG-19 model, accuracy: 59.25%
```

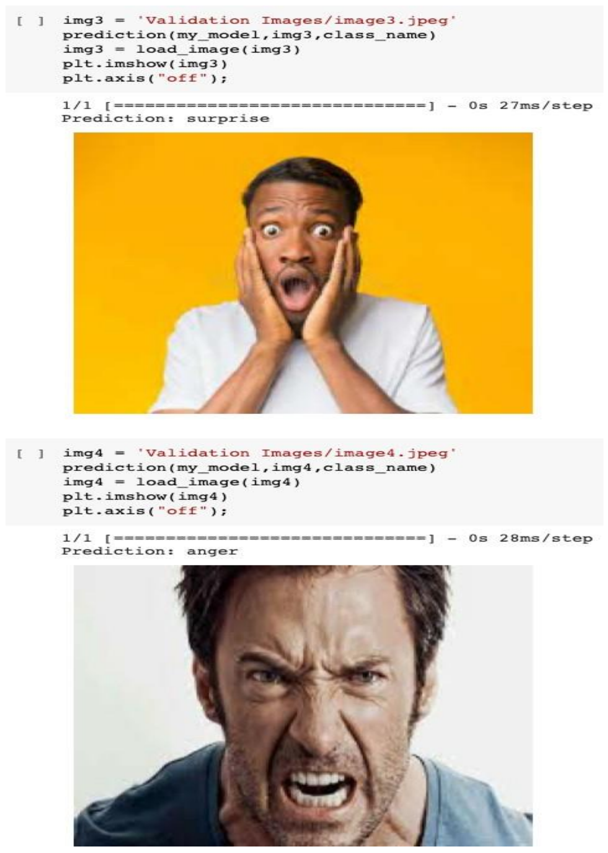
Predicting Image EmotionsUsing Proposed Model



Prediction

```
[ ]  img1 = 'Validation Images/image1.jpeg'
     prediction(my_model,img1,class_name)
     img1 = load_image(img1)
     plt.imshow(img1)
     plt.axis("off");

     1/1 [==============================] - 2s 2s/step
     Prediction: happy
```

```
     img2 = 'Validation Images/image2.jpeg'
     prediction(my_model,img2,class_name)
     img2 = load_image(img2)
     plt.imshow(img2)
     plt.axis("off");

     1/1 [==============================] - 0s 20ms/step
     Prediction: sad
```



```
[ ]  img3 = 'Validation Images/image3.jpeg'
     prediction(my_model,img3,class_name)
     img3 = load_image(img3)
     plt.imshow(img3)
     plt.axis("off");

     1/1 [==============================] - 0s 27ms/step
     Prediction: surprise
```

```
[ ]  img4 = 'Validation Images/image4.jpeg'
     prediction(my_model,img4,class_name)
     img4 = load_image(img4)
     plt.imshow(img4)
     plt.axis("off");

     1/1 [==============================] - 0s 28ms/step
     Prediction: anger
```

References

1. Shan Li ,Weihong Deng (2020). Deep Facial Expression Recognition: A Survey

2. Daniel Canedo and António J. R. Neves(2019). Facial Expression Recognition Using Computer Vision: A Systematic Review

3. Yunxin Huang, Fei Chen, Shaohe Lv and Xiaodong Wang(2019). Facial Expression Recognition: A Survey

4. Yadan Lv, Zhiyong Feng, Chao Xu(2015). Facial expression recognition via deeplearning

5. Xiaoming Zhao, Xugan Shi and Shiqing Zhang(2015). Facial Expression Recognition via Deep Learning

6. Wei Li, Min Li and Zhong Su(2015). A Deep-Learning Approach to Facial Expression Recognition with Candid Images

7. Sharmeen M. Saleem Abdullah and Adnan Mohsin Abdulazeez(2021). Facial Expression Recognition Based on Deep Learning Convolution Neural Network: AReview

8. Shiqing Zhang, Xianzhang Pan, Yueli Cui, Xiaoming Zhao and Limei Liu(2022). Learning Affective Video Features for Facial Expression Recognition via Hybrid DeepLearning

9. Huilin Ge, Zhiyu Zhu, Yuewei Dai, Biao Wang and Xuedong Wu(2022). Facial expression recognition based on Deep Learning

10. Ting Zhang(2017). Facial Expression Recognition Based on Deep Learning: A Survey

11. Atul Sajjanhar, ZhaoQi Wu and Quan Wen(2019). Deep Learning Models for Facial Expression Recognition

12. Viraj Mavani, Shanmuganathan Raman, Krishna P. Miyapuram(2017). Facial Expression Recognition Using Visual Saliency and Deep Learning

13. Isha Talegaonkar, Kalyani Joshi, Shreya Valunj, Rucha Kohok, Anagha Kulkarni(2019). Real-Time Facial Expression Recognition using Deep Learning

14. Yanan Guo, Dapping Tao, Jun Yu, Hao Xiong, Yaotang Li and Dacheng Tao(2019).
Deep Neural Networks with Relativity Learning for Facial Expression Recognition

15. Yuedong Chen, Jianfeng Wang, Shikai Chen, Zhongchao Shi and Jianfei Cai(2019). Facial Motion Prior Networks for Facial Expression Recognition

16. Arriaga O, Valdenegro Toro M, Ploger PG (2019) Real time convolutional neural networks for emotion and gender classification. Proceedings of the 2019 European symposium on artificial neural networks, computational intelligence.

17. Barrett LF, Adolphs R, Marsella S, Martinez AM, Pollak SD (2019) Emotional expressions reconsidered: challenges to inferring emotion from human facial movements.Psychol Sci Public Interest

18. Bartlett MS, Littlewort G, Frank MG, Lainscsek C, Fasel IR, Movellan JR (2006) Automatic recognition of facial actions in spontaneous expressions. J Multimed

19. Chang WY, Hsu SH, Chien JH (2017) FATAUVA-net: an integrated deep learning framework for facial attribute recognition, action unit detection, and valence-arousal estimation. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops

20. Chao L, Tao J, Yang M, Li Y, Wen Z (2015) Long short-term memory recurrent neural network based multimodal dimensional emotion recognition. In: Proceedings of the 5th international workshop on audio/visual emotion challenge

21. Chen J, Liu X, Tu P, Aragones A (2013) Learning person-specific models for facial expression and action unit recognition. Pattern Recogn Lett

22. Chu WS, De la Torre F, Cohn JF (2016) Selective transfer machine for personalized facial expression analysis. IEEE Trans Pattern Anal Mach Intell

23. Dhall A, Ramana Murthy O, Goecke R, Joshi J, Gedeon T (2015) Video and image based emotion recognition challenges in the wild: Emotiw 2015. In: Proceedings of the 2015 ACM on international conference on multi-modal interaction

24. AffectNet Dataset (https://www.kaggle.com/datasets/tom99763/affectnethq)

25. Cohn Kanade Dataset (https://ieeexplore.ieee.org/document/5543262)

26. RAF-DB Dataset (https://www.v7labs.com/open-datasets/raf-db)