

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(DEEMED TO BE UNIVERSITY)

Submitted in partial fulfilment of the requirements for the
award of Bachelor of Engineering Degree in Computer
Science and Engineering

By
SUMANA SAHA
(Reg. No. 38110570)



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU.

MARCH 2022



**SATHYABAMA INSTITUTE
OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

(Established under Section 3 of UGC Act, 1956)

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the bonafide work of **SUMANA SAHA (Reg. No.38110570)** who underwent the professional training in **“REAL TIME EMOTION DETECTION USING TRANSFER LEARNING”** under our supervision from Jan 2022 to Mar 2022.

Internal Guide

DR. A. CHRISTY, M.C.A, Ph.D.

Head of the Department

DR. LAKSHMANAN L, M.E., Ph.D.

DR. S. VIGNESHWARI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **SUMANA SAHA** (Reg. No. 38110570) hereby declare that the Professional Training Report on “**Real Time Emotion Detection using Transfer Learning**” done by me under the guidance of **DR. A. CHRISTY, M.C.A, Ph.D.**, at Sathyabama Institute of Science and Technology is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE:



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **DR. T. SASIKALA M.E., PH. D**, Dean, School of Computing, **DR. Lakshmanan L, M.E., PH.D.** and, **DR. S. VIGNESHWARI, M.E., PH.D.**, Heads of the **Department of Computer Science and Engineering** for providing me necessary support and details at the right time during the progressive views.

I would like to express my sincere and deep sense of gratitude to my Project Guide **DR. A. CHRISTY, M.C.A., PH.D.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-Teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of my project.

ABSTRACT

Emotion is one of the few terms which does not have a precise definition that is also intelligible. It's ethereal. "Emotion" is a term used to describe a subjective condition expressed through social signals. This differs from, for instance, recognizing the emotional content of a multimedia clip, which is concerned with the feelings that the clip may elicit in its audience. Despite this, practically every decision we've ever made has been influenced by emotion. According to marketing studies, correctly forecasting feelings can be a big source of growth for businesses. Also, can be used for safety measures such as drowsiness detection for a car driver. In the computer vision domain, recognizing automated facial expressions from a facial image is a difficult task that seems to have a wide range of applications, covering driving safety, human-computer interactions, health care, behavioural science, teleconferencing, cognitive science, as well as others. This notion falls under the category of cognitive systems in the area of data and machine learning. The goal is to try to segregate the emotions contained in the data using training data. The FER2013 data consists of pixel images of emotions of faces of seven types of emotions which are Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image. The goal is to recognize the emotions in a real-time environment using a webcam. Deep convolutional neural networks have been used to produce numerous advancements in image categorization (CNNs). An automated feature extractor and a classifier are the two major components of these designs. The former generates low-level, mid-level, and high-level characteristics for the item of interest, characterizing basic, moderate, and complicated textures, respectively. In general, a powerful classifier learns the target from a large number of high-level characteristics, hence the network should be trained using a huge amount of data. Deep learning techniques are frequently employed in many picture classification difficulties, such as ImageNet, PASCAL VOC, CIFAR, and the Facial Expression Recognition Challenge 2013. The FER2013 data consists of images of faces of seven types of emotions i.e., Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. Using the dataset, these networks may be leveraged to produce task-specific outcomes. Using Transfer Learning, we utilized our dataset on a pre-trained network. The pre-trained network is used for both feature extraction and

classification. Transfer Learning takes features and weights from previously trained models and applies them to subsequent models, even when there is less information on the most recent job.

TABLE OF CONTENTS

Chapter No.	Title	Page No.
	Abstract	vi
	List of Figures	viii
	List of Tables	ix
1.	INTRODUCTION	
	1.1 Outline of the Project	1
2.	LITERATURE SURVEY	2
3.	Aim And Scope of The Present Investigation	
	3.1 Purpose of the Project	4
	3.2 Project Architecture	5
4.	Experimental Or Materials and Methods Algorithms Used	
	4.1 Introduction	7
	4.2 Working Explanation	8
	4.2.1 Notations	8
	4.2.2 Mathematics behind the project	8
	4.2.3 Dataset	12
	4.2.4 Convolutional Neural Networks	15
	4.2.5 Activation Functions	16
	4.2.6 Improving the model	20
	4.2.7 Transfer Learning	24
5.	Results And Discussion, Performance Analysis	30
6.	CONCLUSION and SUMMARY	34
7.	APPENDIX	
	A. Source Code	35
	B. Output	42
8	REFERENCES	43

LIST OF FIGURES

Figure No.	Title	Page No.
3.1	Architecture of the model	5
3.2	Architecture of the model	5
3.3	Xception Model	6
4.1	FER-2013	13
4.2	Architecture of traditional Neural Network	15
4.3	Convolutional Layer	15
4.4	Fully Connected	16
4.5	Sigmoid Function	17
4.6	ReLU Function	17
4.7	Leaky ReLU Function	17
4.8	Tan h Function	18
4.9	Backpropagation	21
4.10	Updating weights	21
4.11	Dropout	23
4.12	Early Stopping	24
4.13	Resnet 50	26
4.14	Entry flow of Xception model	27
4.15	Middle and Exit flow of Xception model	28
4.16	VGG 16 Model	29
5.1	Training Set	30
5.2	Accuracy and Loss	31
5.3	Confusion Matrix	31
5.4	Test Image 1	32
5.5	Test Image 2	32
5.6	Test Image 3	33
5.7	Test Image 4	33
6.1	Output	37

LIST OF TABLES

Table No.	Title	Page No.
4.1	Data Augmentation using image dimension	14
4.2	Data Augmentation using image colour	14
4.3	ReLU Functions	19
4.4	Transfer Learning	22
4.5	Weight Regularization	23
4.6	Comparison and Analysis between the models	29

CHAPTER 1

INTRODUCTION

1.1 OUTLINE OF THE PROJECT

Artificial Neural Networks (ANN) is inspired by the human brain and it's can be used for machine learning and artificial intelligence. With these networks, various problems can be solved computer-based. The artificial neural network (ANN) is to some extent modelled on the structure of the biological brain. It consists of an abstracted model of interconnected neurons, whose special arrangement and linking can be used to solve computer-based application problems in various fields such as statistics, technology or economics. The neural network is a research subject of Neuro informatics and part of the artificial intelligence. Neural networks must be trained before they can solve problems.

Image recognition is one of the tasks in deep neural networks. Neural networks are computing systems designed to recognize patterns. Their architecture is inspired by the human brain structure, hence the name. They consist of three types of layers: input, hidden layers, and output. The input layer receives a signal, the hidden layer processes it, and the output layer makes a decision or a forecast about the input data. Each network layer consists of interconnected *nodes (artificial neurons)* that do the computation. Transfer Learning takes features and weights from previously trained models and applies them to subsequent models, even when there is less information on the most recent job.

CHAPTER 2

LITERATURE SURVEY

This section summarises all of the previous work on facial expression recognition. They used MLCNN which automatically selected important mid-level and high-level features based on their contribution and also the video-based facial expression recognition, the temporal model whose backbone is the proposed ensemble of MLCNNs also achieved comparable performance and outperformed in Facial Expression Recognition Using a Temporal Ensemble of Multi-level Convolutional Neural Networks. Facial Expression Recognition Using Deep Convolution Neural Networks Based on Local Gravitational Force Descriptors is an article published in the journal Facial Expression Recognition. The suggested technique beats twenty-five baseline methods by considering the average time and achieves average recognition accuracy of 78 percent, 98 percent, 98 percent, 96 percent, and 83 percent for FER2013, JAFFE, CK+, KDEF, and RAF, respectively. Keith Anderson and Peter McCowan's A Real-Time Automated System for the Recognition of Human Facial Expressions was not only trained to distinguish between different expressions but also between expressive and non-expressive sequences. However, this system found the classification of fear and anger expressions particularly difficult, and as a result, the accuracy was reduced. Maja Pantic and Ioannis Patras' Dynamics of Face Expression: Recognition of Facial Actions and Their Temporal Segments from Face Profile Image Sequences automates the segmentation of input image sequences into expressive and expressionless facial activity. Their solution uses a memory-based mechanism that takes into consideration the dynamics of facial expressions to cope with mistakes in facial point tracking, however, the suggested facial expression analyser cannot deal with spontaneously occurring facial behaviour. A Face Emotion Recognition Method Using Convolutional Neural Network and Image Edge Computing, a technique was suggested that can automatically learn pattern features and eliminate the incompleteness caused by artificial design features, as well as direct input the image pixel value via training sample image data. Furthermore, the suggested technique yields a greater recognition rate, although the neural network architecture is extremely complicated in this case. Hyeon-Jung Lee and Kwang-Seok Hong's work on Emotion

Recognition Method and Its Application Using Face Image found that the application performance rate in positive and negative emotions was 72.3 percent, but only 50.7 percent for seven categories, and the neutral category was eliminated. The suggested Human Emotion Recognition using Convolutional Neural Network in the Real-Time model can classify seven different emotions regardless of the number of faces at a time. Although the shallow model, which consists of a single convolution layer, achieves an accuracy of up to 47.94 percent, the model's average performance in identifying the disgust emotion is lower due to a fewer number of pictures in that category in the training dataset. Charvi Jain, Kshitij Sawant, Mohammed Rehman, and Rajesh Kumar's paper Emotion Detection and Characterization Using Facial Features This method only requires two attributes to determine the emotion of a whole face, reducing the amount of storage data required for testing and future applications, but it can only recognize six of the seven emotions. Recognition of Facial Expressions Based on CNN, it provides a method that is very stable and is unaffected by subject-to-subject face physiological changes, but it suffers from overfitting in the training and testing phases due to a lack of facial emotions. Facial Emotion Recognition of Students Using Convolutional Neural Network proposed model by Imane Lasri, Anouar Riad Solh, and Mourad El Belkacemis achieved a 70 percent accuracy rate on the FER 2013 training database, but the actual performance of the model in real-time environment and accuracy on validation dataset is quite low and predicts quite feared faces poorly because it confuses them with sad faces.

CHAPTER 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

3.1 PURPOSE OF THE PROJECT

The main goal of this project is to build a neural network model which gives an accuracy more than what we have received in previous works as mentioned in the literature review on the same dataset (i.e., FER2013) and be able to work dynamically in real-life scenarios. The dataset chosen for the project is FER2013. The pre-trained models from transfer learning will be used to achieve the target. E.g.: Resnet50, MobileNetV2 etc. The model should be able to predict every emotion with similar accuracy and none of the emotions would be excluded. Here we have tried to build a model that should be able to do this on a real-time basis using a webcam or a camera and the time taken to generate the results be minimized as much as possible to facilitate real-life usage.

Disadvantages:

A few types of images the model tends to do poorly on include:

- The human in an unusual position
- Face appears against a background of a similar colour
- Brightness of the picture
- Scale variation (face is very large or small in image)

Advantages:

The model has higher prediction than the basic machine learning model. This is simple algorithm built using simple and open-source packages like TensorFlow and SciPy library.

3.2 PROJECT ARCHITECTURE:

ARCHITECTURE OF THE MODEL:

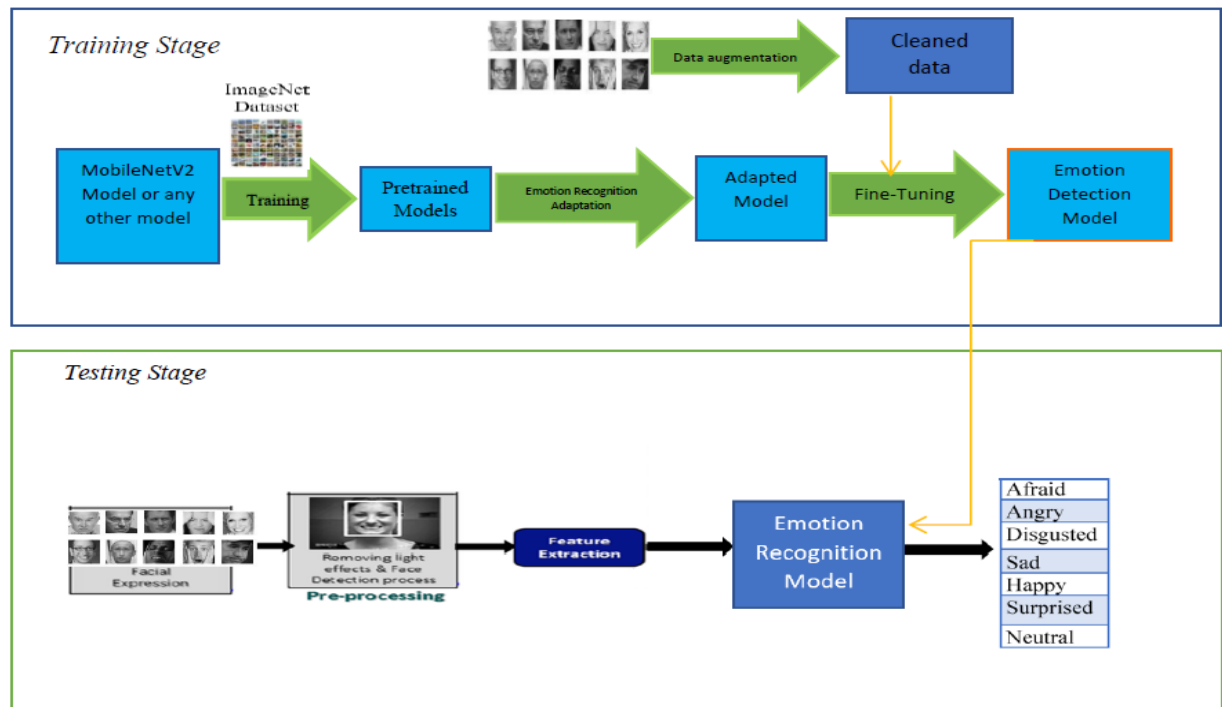


Figure 3.1

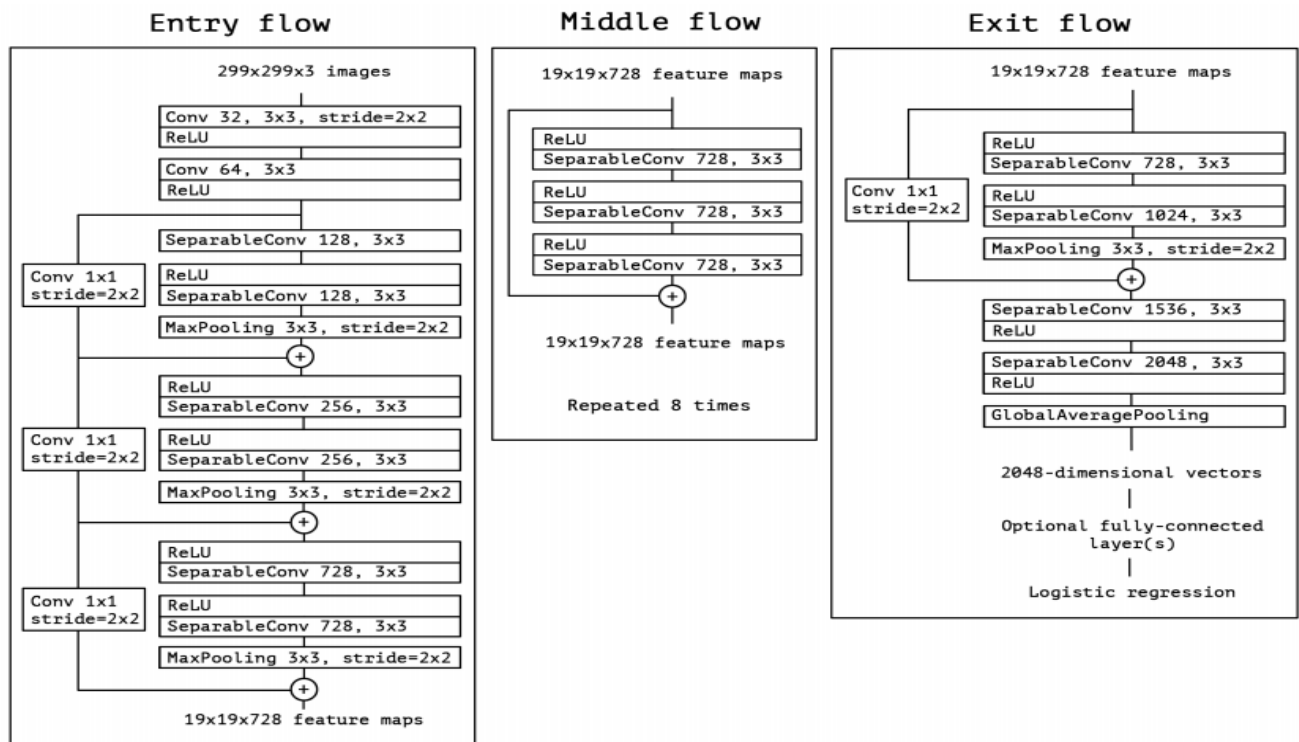


Figure 3.2

Out[23]:

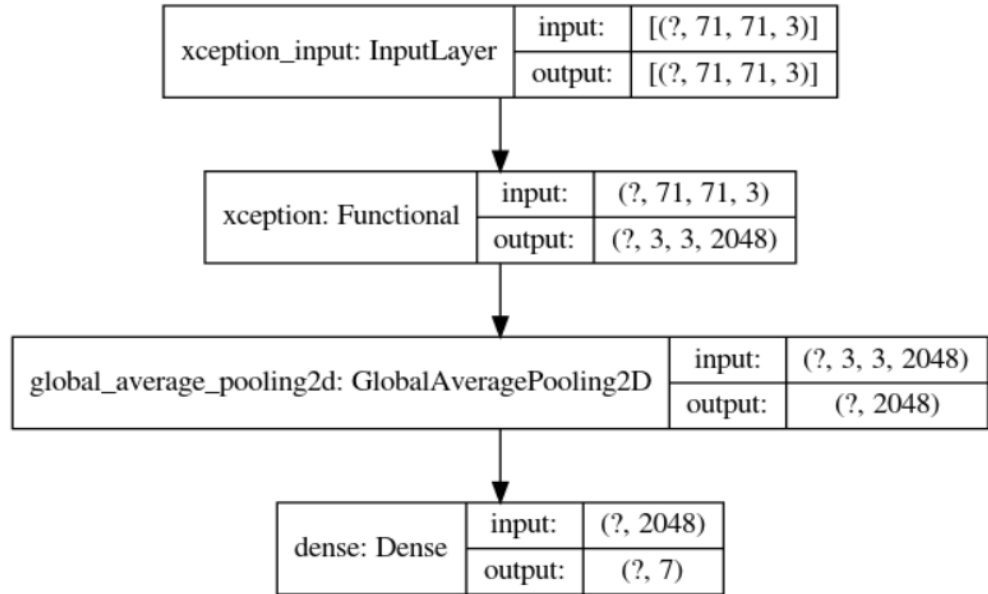


Figure 3. 3

CHAPTER 4

EXPERIMENTAL OR MATERIALS AND METHODS

ALGORITHMS USED

4.1 INTRODUCTION

Image recognition is one of the tasks in deep neural networks. Neural networks are computing systems designed to recognize patterns. Their architecture is inspired by the human brain structure, hence the name. They consist of three types of layers: input, hidden layers, and output. The input layer receives a signal, the hidden layer processes it, and the output layer makes a decision or a forecast about the input data. Each network layer consists of interconnected *nodes (artificial neurons)* that do the computation.

Traditional neural networks have up to three hidden layers, deep networks may contain hundreds of them. So, to be able to recognize faces, a system must learn their features first. It must be trained to predict whether an object is X or Z. Deep learning models learn these characteristics in a different way from machine learning models. That's why model training approaches are different as well. Each layer of nodes trains on the output (feature set) produced by the previous layer. So, nodes in each successive layer can recognize more complex, detailed features visual representations of what the image depicts. Such a "hierarchy of increasing complexity and abstraction" is known as *feature hierarchy*.

So, the more layers the network has, the greater its predictive capability.

4.2 WORKING EXPLANATION:

4.2.1 Notations:

$$(x, y) = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), (X^{(3)}, y^{(3)}), \dots, (X^{(m)}, y^{(m)})\}$$

Where $X \in \mathbb{R}^{n_x}$, $y \in \{0,1\}$

$$X = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}$$

Here, $(X(1), X(2), \dots, X(m))$ are columns

$$y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$$

$$W = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ W^{(1)} & W^{(2)} & \dots & W^{(m)} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}^T$$

Where W is considered as the weight matrix usually initialized to 0.

$$A^{[1]} = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ a^{1} & a^{[1](2)} & \dots & X^{[1](m)} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}$$

Where A represents the Hidden Layer number, $[i]$ represents the i^{th} hidden layer number, and (i) represents the i^{th} element number.

$$Z^{[1]} = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ z^{1} & z^{[1](2)} & \dots & z^{[1](m)} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}$$

Where Z matrix contains the finally calculated result of $[i]^{\text{th}}$ layer.

Dimensions:

Dimension of X : $m \times n_x$

Dimension of Y : $1 \times m$

4.2.2 Mathematics Behind the Project:

Using the Linear Regression Equation:

$$y = ax + c$$

Where a is the slope and c is y intercept

Here:

$$\hat{y} = \sigma(w^T x + b), (\text{where, } \sigma(z) = \frac{1}{1+e^{-z}} \text{ and } z = w^T x + b)$$

$$\text{Or, } \hat{y} = g(w^T x + b), (\text{where } g \text{ is the activation function})$$

Loss or Error Function:

$$\mathcal{L}(\hat{y} - y) = \frac{1}{2} (\hat{y} - y)^2$$

We might use a better loss function here:

$$\mathcal{L}(\hat{y} - y) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y}))$$

Cost Function:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y} - y)$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Derivation of Cost Function:

Using Total Probability theorem:

$$P(\text{Labels in training set}) = \prod_{i=1}^m P(y^{(i)} | x^{(i)})$$

Taking log on both sides (since log changes multiplication to addition and make it easy to work with formulas)

$$\log(P(\text{labels in training set})) = \sum_{i=1}^m \log(P(y^{(i)} | x^{(i)}))$$

$$\log(P(\text{labels in training set})) = -\sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$\text{Therefore, } J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y} - y)$$

Forward Propagation in Neural Networks:

$$z = w^T + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(\hat{y} - y) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y}))$$

Backward Propagation:

$$dz = \frac{d\mathcal{L}(a,y)}{dz} = a - y$$

$$da = \frac{d\mathcal{L}(a,y)}{da} = \frac{-y}{a} + \frac{1-y}{1-a}$$

$$dw = \frac{d\mathcal{L}(a,y)}{dw} = X_1 dz$$

$$db = dz$$

$$w_i = w_i - \alpha dw_i$$

$$b = b - \alpha db_i$$

Where α is called the Learning Rate.

Derivation of $\frac{dL}{dz}$:

If you're curious, here is the derivation for $\frac{dL}{dz} = a - y$

Note that in this part of the course, Andrew refers to $\frac{dL}{dz}$ as dz .

By the chain rule: $\frac{dL}{dz} = \frac{dL}{da} \times \frac{da}{dz}$

We'll do the following: 1. solve for $\frac{dL}{da}$, then

Step 1: $\frac{dL}{da}$

$$L = -(y \times \log(a) + (1 - y) \times \log(a - 1))$$

$$\frac{dL}{da} = -y \times \frac{1}{a} - (1-y) \times \frac{1}{1-a} \times -1$$

We're taking the derivative with respect to a .

Remember that there is an additional -1 in the last term when we take the derivative of $(1-a)$ with respect to a . Also note that the notational conventions are different in the ML world than the math world: here $\log$ always means the natural log.

$$\frac{dL}{da} = \frac{-y}{a} + \frac{1-y}{1-a}$$

We'll give both terms the same denominator:

$$\frac{dL}{da} = \frac{-y \times (1-a)}{a \times (1-a)} + \frac{a \times (1-y)}{a \times (1-a)}$$

Clean up the terms:

$$\frac{dL}{da} = \frac{-y + ay + a - ay}{a \times (1-a)}$$

So now we have:

$$\frac{dL}{da} = \frac{a - y}{a \times (1-a)}$$

Step 2: $\frac{da}{dz}$

$$\frac{da}{dz} = \frac{d}{dz} \sigma(z)$$

The derivative of a sigmoid has the form:

$$\frac{d}{dz} \sigma(z) = \sigma(z) \times (1 - \sigma(z))$$

We can look up why this derivation is of this form.

Recall that $\sigma(z) = a \times(z)=a$, because we defined " a ", the activation, as the output of the sigmoid activation function.

So, we can substitute into the formula to get:

$$\frac{da}{dz} = a(1-a)$$

Step 3: $\frac{dL}{dz}$

We'll multiply step 1 and step 2 to get the result.

$$\frac{dL}{dz} = \frac{dL}{da} \times \frac{da}{dz}$$

From step 1: $\frac{dL}{da} = \frac{a-y}{a(1-a)}$

From step 2: $\frac{da}{dz} = a(1-a)$

$$\frac{dL}{dz} = \frac{a-y}{a(1-a)} \times a(1-a)$$

Notice that we can cancel factors to get this:

$$\frac{dL}{dz} = a - y$$

We refer $\frac{dL}{dz}$ as dz in Neural Networks especially in code

So, we can write:

$$dz = a - y$$

4.2.3 Dataset

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image.

The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

Created in 2013, FER-2013 dataset contains 35,887 grayscale 48x48-pixel images, with the images being stored in a spreadsheet where each image's pixel values are stored in cells per row. The images were obtained utilizing Google search and are later grouped per emotion classes, which are anger, disgust, fear, happiness, neutral, sadness, and surprise. As the dataset was built utilizing Google search, the images are in in-the-wild condition, with even very few images being animated characters.

The dataset originally had data distribution of 28,709 images for training and 3,589 images for public test, but after the competition ended, another 3,589 images which were used for private test were added to the dataset. The usages of FER-2013's data distribution vary among published researches, with each using public test images for different purposes, either as part of training set, validation set, or test set.



Figure 4. 1 FER-2013

Data processing

Data augmentation: Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

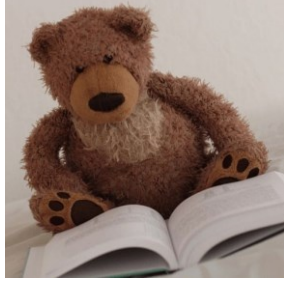
Original	Flip	Rotation	Random crop
			
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

Table 4.1 Data augmentation using dimension

Colour shift	Noise addition	Information loss	Contrast change
			
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

Table 4.2 Data augmentation using colour

4.2.4 Convolutional Neural Networks

Architecture of a traditional CNN: Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:

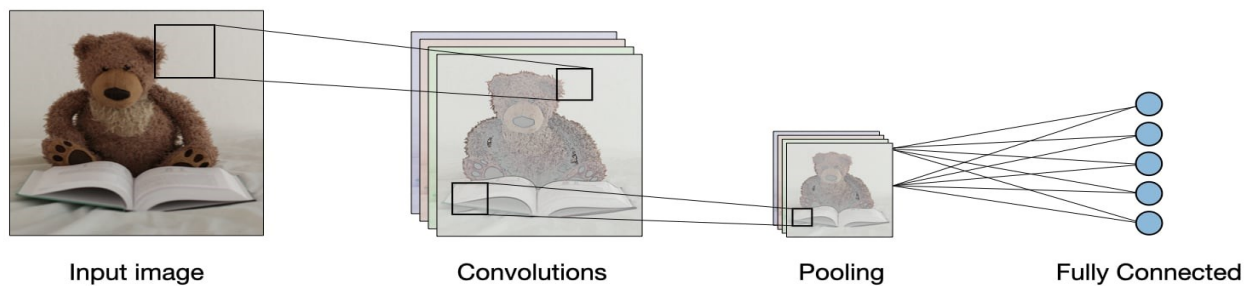


Figure 4. 2 Architecture of traditional Neural Networks

Types of layers:

Convolution layer (CONV) The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size FF and stride SS . The resulting output OO is called *feature map* or *activation map*.

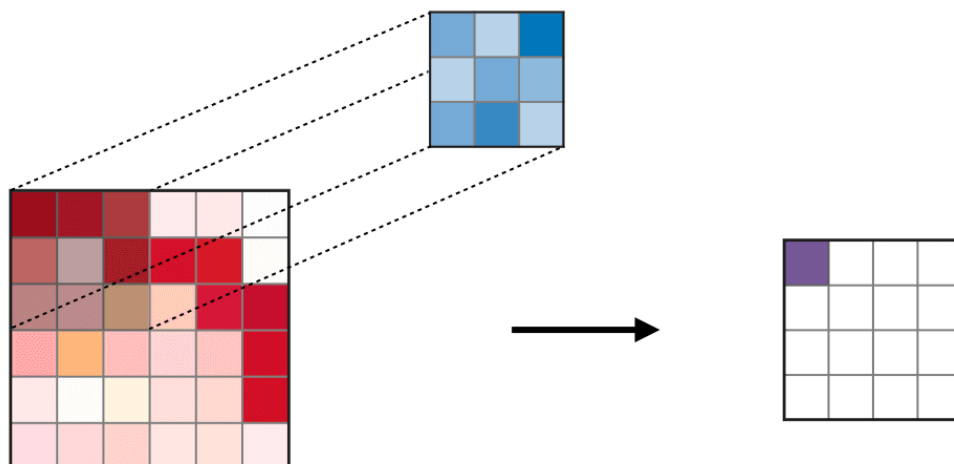


Figure 4. 3 Convolutional Layer

Pooling (POOL) The pooling layer (POOL) is a down sampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

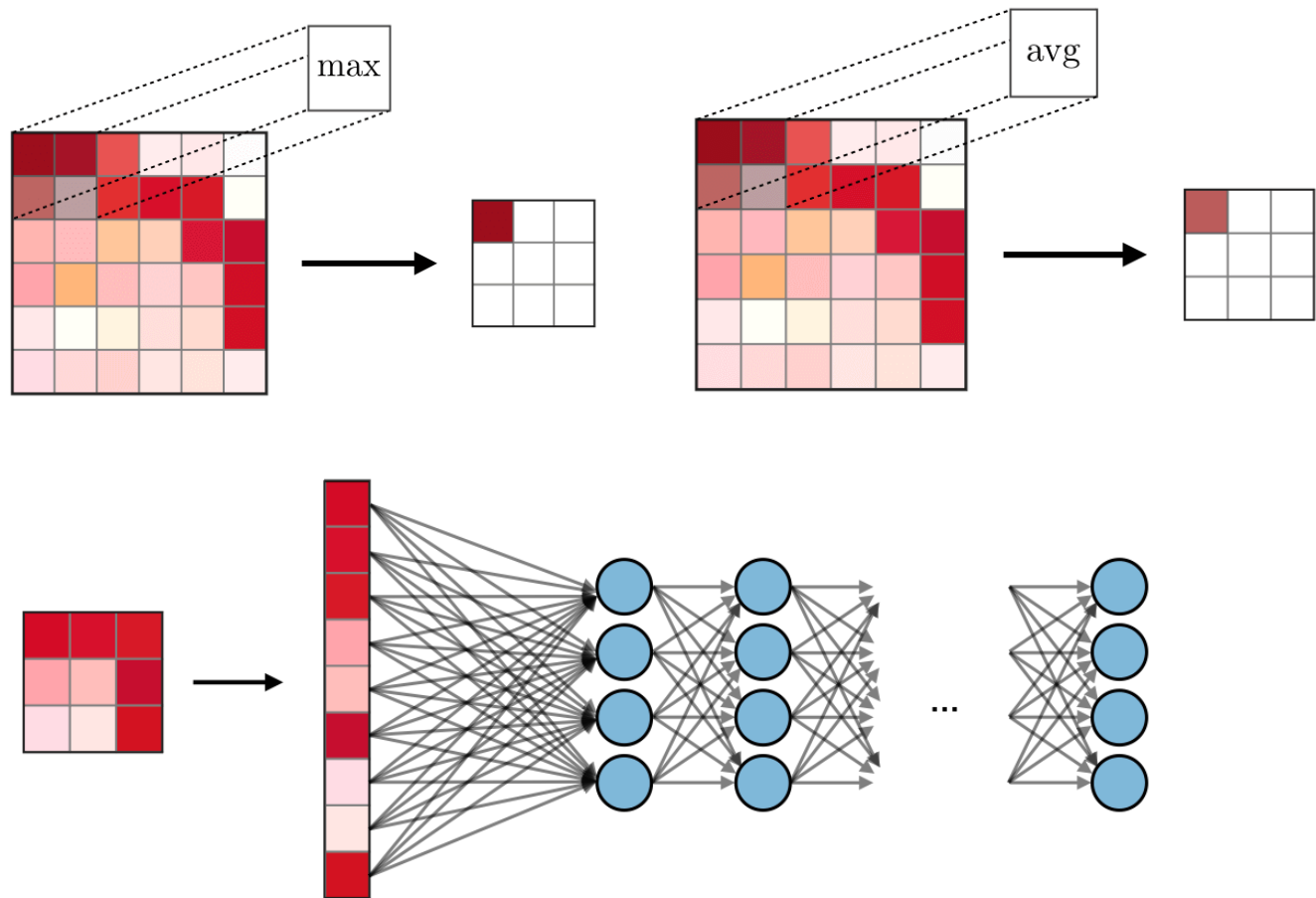


Figure 4. 4 Fully Connected Layer

Fully Connected (FC) The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.

4.2.5 Activation Functions:

Sigmoid function as the Activation Function:

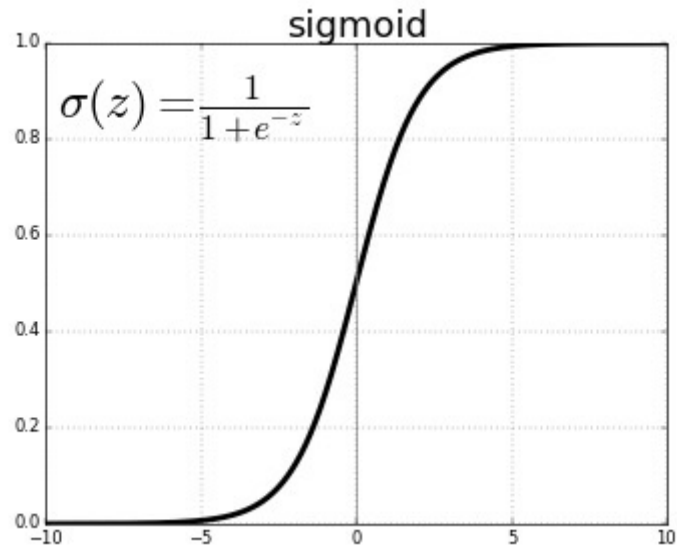


Figure 4. 5 Sigmoid Function

Using this function in 2- Layer Neural Network

ReLU (Rectify Linear Unit) Function as the Activation Function:

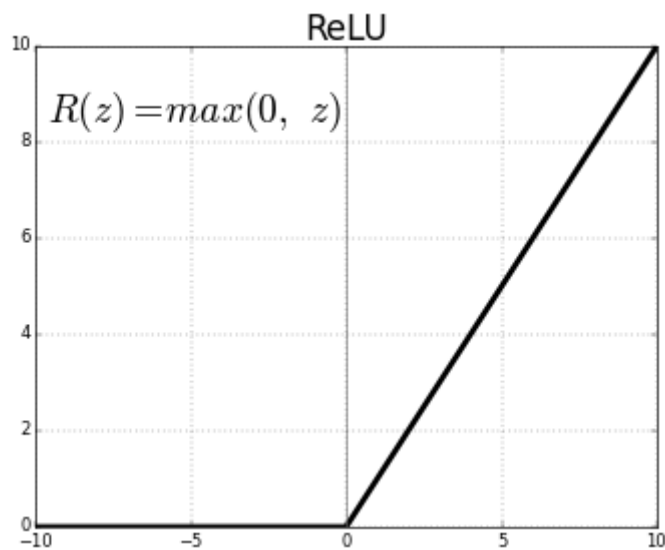


Figure 4. 6 ReLU Function

Usually used for binary prediction models

Leaky ReLU Function as the Activation Function:

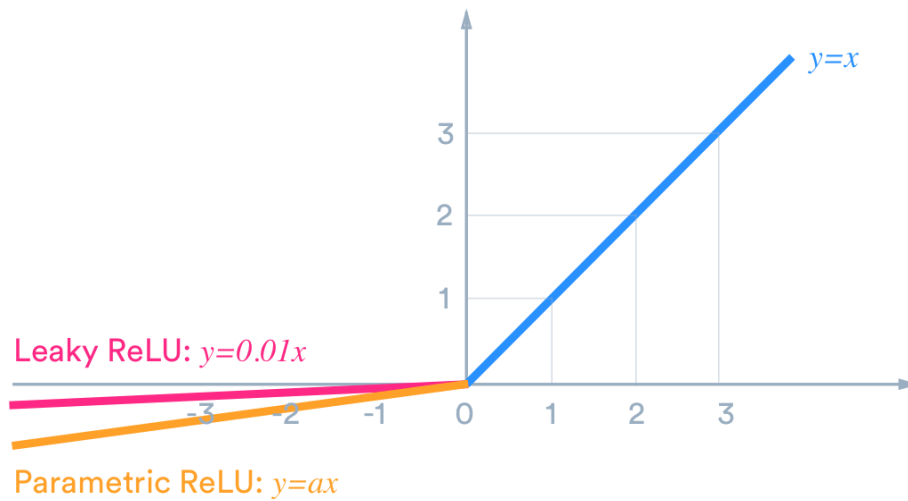


Figure 4. 7 Leaky ReLU Function

Tan h Function as the Activation Function:

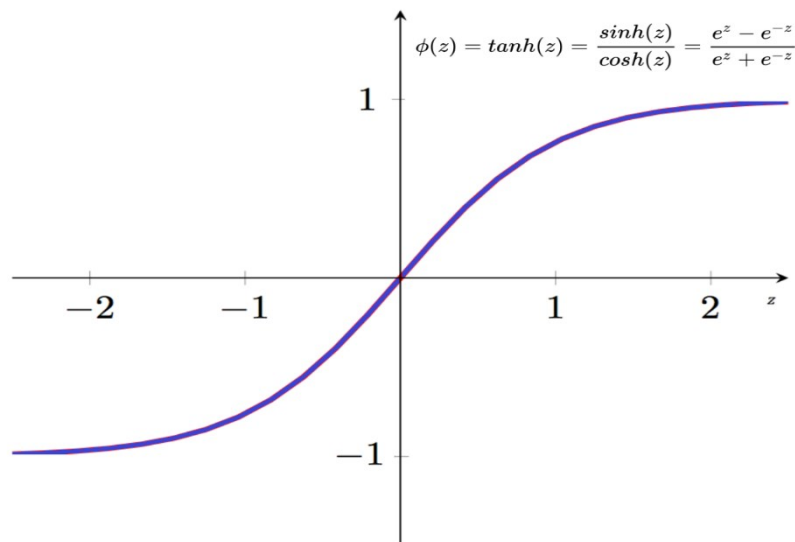


Figure 4. 8 Tan h Function

Rectified Linear Unit The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

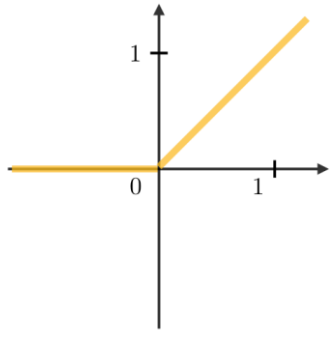
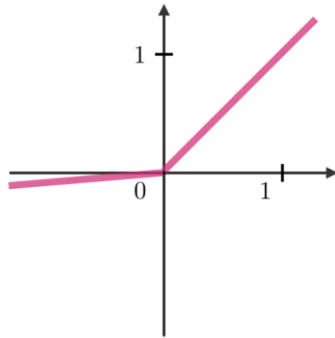
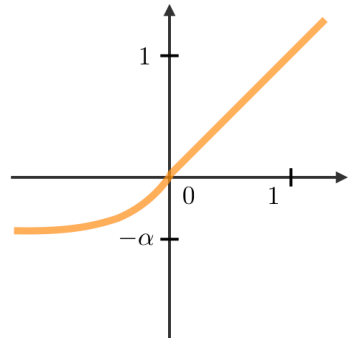
ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

Table 4.3 ReLU Functions

SoftMax The SoftMax step can be seen as a generalized logistic function that takes as input a vector of scores $x \in \mathbb{R}^n$ and outputs a vector of output probability $p \in \mathbb{R}^n$ through a softmax function at the end of the architecture. It is defined as follows:

$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{Where } p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

4.2.6 Improving Model:

Training Neural Network

Epoch In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

Mini-batch gradient descent During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

Loss function: In order to quantify how a given model performs, the loss function LL is usually used to evaluate to what extent the actual outputs y are correctly predicted by the model outputs z .

Cross-entropy loss: In the context of binary classification in neural networks, the cross-entropy loss $L(z, y)$ is commonly used and is defined as follows:

$$L(z, y) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

Finding optimal weights

Backpropagation: Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.

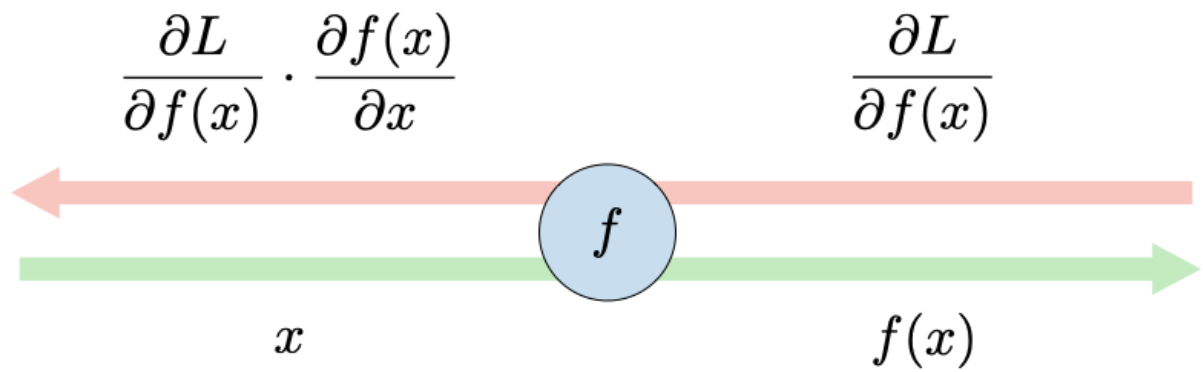


Figure 4. 9 Backpropagation

Updating weights: In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.

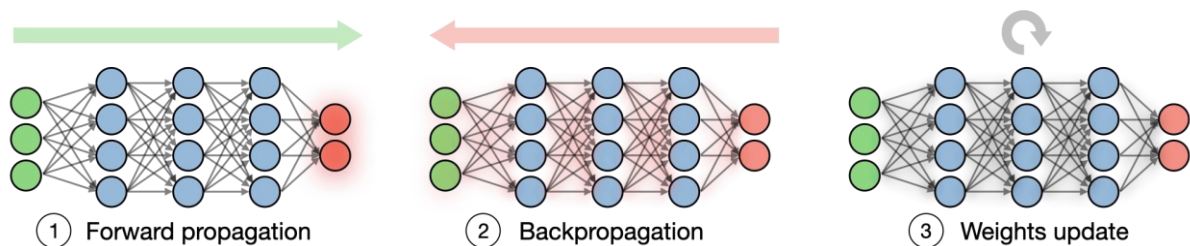


Figure 4. 10 Updating weights

Parameter tuning

Weights initialization

Xavier initialization: Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

Transfer learning: Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights

on huge datasets that took days/weeks to train, and leverage it towards our use case.

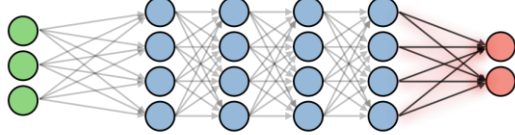
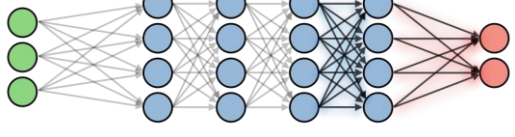
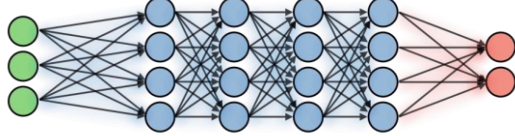
Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on SoftMax
Medium		Freezes most layers, trains weights on last layers and SoftMax
Large		Trains weights on layers and SoftMax by initializing weights on pre-trained ones

Table 4.4 Transfer Learning

Optimizing convergence

Learning rate: The learning rate, often noted α or sometimes η , indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

Adaptive learning rates: Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful.

Regularization

Dropout: Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.

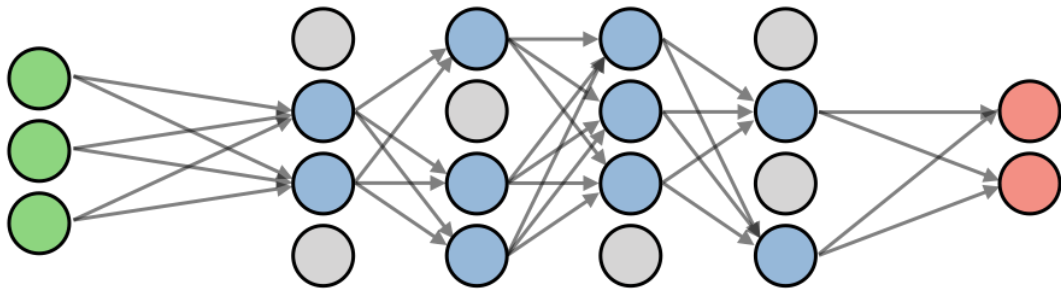


Figure 4. 11 Dropout

Weight regularization: In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Trade-off between variable selection and small coefficients
$ \theta _1 \leq 1$	$ \theta _2 \leq 1$	$(1 - \alpha) \theta _1 + \alpha \theta _2^2 \leq 1$

Table 4.5 Weight Regularization

Early stopping: This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.

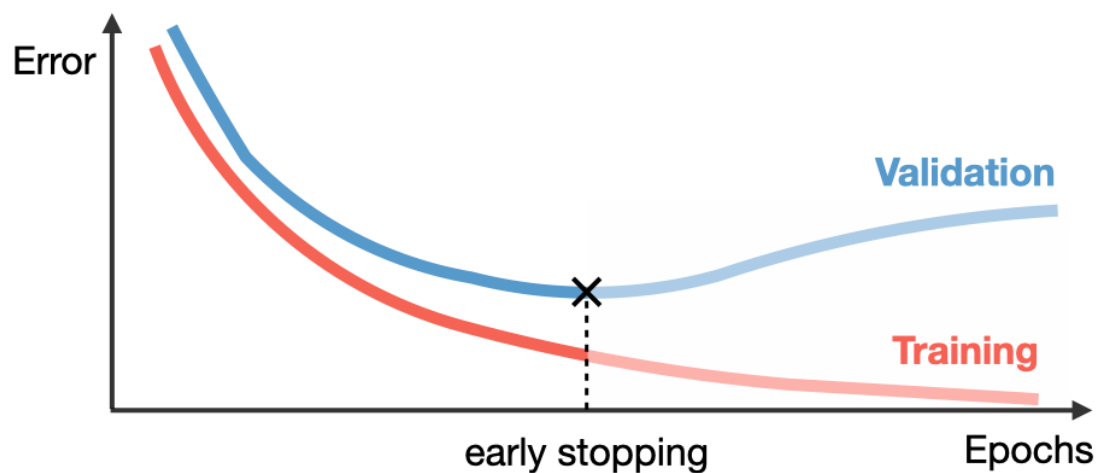


Figure 4. 12 Early stopping

Overfitting small batch: When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

Gradient checking: Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

4.2.7 Transfer Learning

Transfer learning is a machine learning technique in which a model created for one problem is employed as the basis for a model on a different task. Given the vast compute and time resources required to develop neural network models on these problems, as well as the huge jumps in a skill that they provide on related problems, it is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks.

Pretrained Models:

ResNet:

AlexNet took first place in the LSVRC2012 classification challenge in 2012, and ResNet became the most fascinating thing to happen in the computer vision and deep learning worlds after that. It was able to train super deep neural networks using ResNets' architecture, which means that a network may have hundreds or thousands of layers and still function well. ResNets were first employed for image identification, but the framework may also be utilized for non-computer vision jobs to improve accuracy, according to the report.

- We get one layer from convolution with a kernel size of $7 * 7$ and 64 distinct kernels, all with a stride of size 2.
- Following that, we have max pooling with a stride size of 2.
- There is a $1 * 1,64$ kernel in the next convolution, followed by a $3 * 3,64$ kernel, and finally a $1 * 1,256$ kernel. These three layers are repeated three times in total, giving us nine layers in this phase.
- This phase was repeated four times, giving us 12 layers. Next, we observe a kernel of $1 * 1,128$ followed by a kernel of $3 * 3,128$ and finally a kernel of $1 * 1,512$.
- Then there's a $1 * 1,256$ kernel, followed by $3 * 3,256$ and $1 * 1,1024$ kernels, which are repeated six times for a total of 18 layers.
- Then a $1 * 1,512$ kernel was added, followed by two additional $3 * 3,512$ and $1 * 1,2048$ kernels, for a total of nine layers.
- After that, we run an average pool and finish with a fully linked layer with 1000 nodes, followed by a SoftMax function, giving us one layer.
- The activation functions and the max/average pooling layers are not counted.
- As a result, we get a Deep Convolutional network with $1 + 9 + 12 + 18 + 9 + 1 = 50$ layers.

ResNet, short for Residual Networks is a classic neural network used as a backbone for many computers vision tasks. This model was the winner of

ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+ layers successfully. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients. ResNet first introduced the concept of skip connection.

The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.

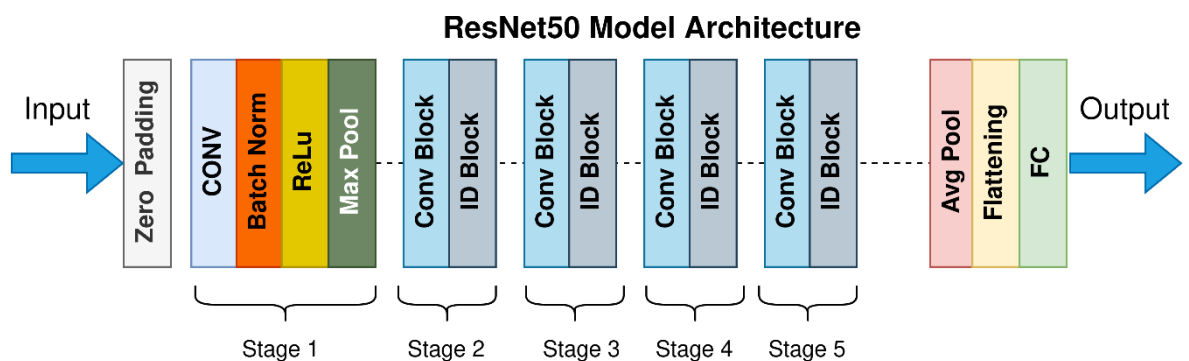


Figure 4.13 Resnet 50

Xception:

Xception by Google, stands for Extreme version of Inception, is reviewed. With a modified depth wise separable convolution, it is even better than Inception-v3. Depth wise convolution is the channel-wise $n \times n$ spatial convolution. Suppose in the figure above, we have 5 channels, then we will have $5 n \times n$ spatial convolution. Pointwise convolution actually is the 1×1 convolution to change the dimension. The modified depth wise separable convolution is the pointwise convolution followed by a depth wise convolution. This modification is motivated by the inception module in Inception-v3 that 1×1 convolution is done first before any $n \times n$ spatial convolutions. Thus, it is a bit different from the original one. The Xception without any intermediate activation has the highest accuracy compared with the ones using either ELU or ReLU.

The entry flow has two blocks of convolutional layer followed by a ReLU activation. The diagram also mentions in detail the number of filters, the filter size (kernel size), and the strides.

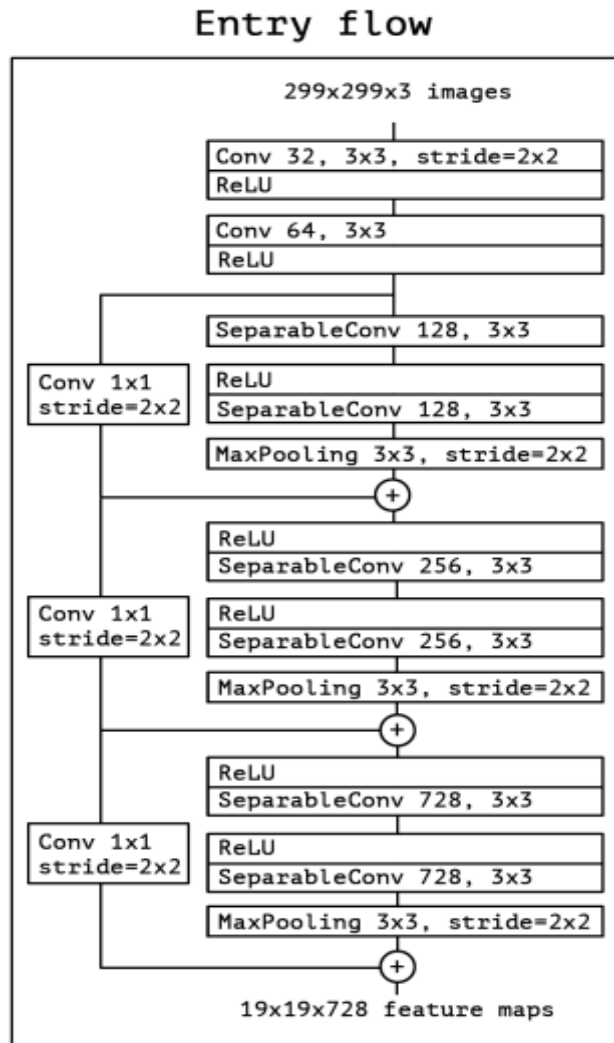


Figure 4.14 Entry flow of Xception model

There are also various Separable convolutional layers. There are also Max Pooling layers. When the strides are different than one, the strides are also mentioned. There are also Skip connections, where we use 'ADD' to merge the two tensors. It also shows the shape of the input tensor in each flow. For example, we begin with an image size of

299x299x3, and after the entry flow, we get an image size of 19x19x728.

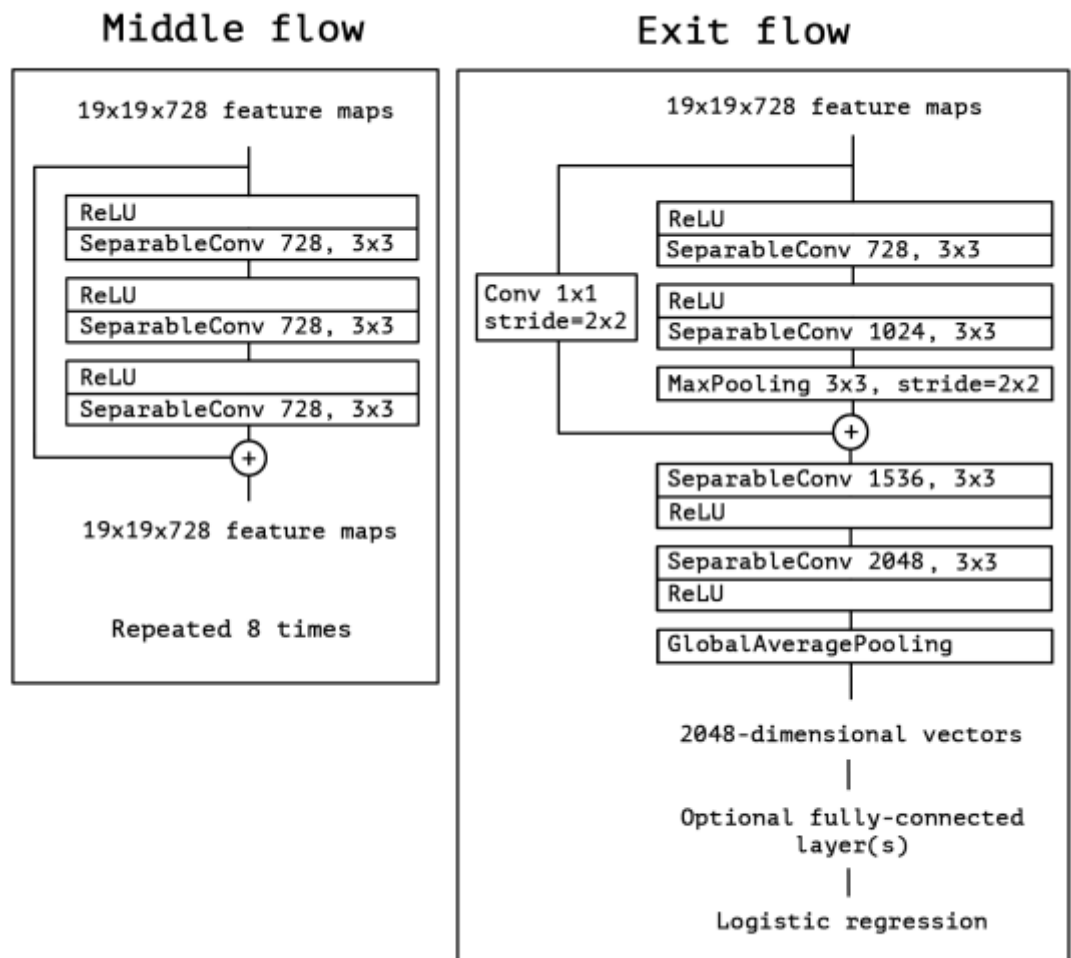


Figure 4.15 Middle and Exit flow of Xception model

For the Middle flow and the Exit flow, this diagram clearly explains the image size, the various layers, the number of filters, the shape of filters, the type of pooling, the number of repetitions, and the option of adding a fully connected layer in the end.

VGG16:

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(ImageNet) competition in 2014. It is considered to be one of the excellent vision model architectures till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and

always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC (fully connected layers) followed by a SoftMax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx.) parameters.

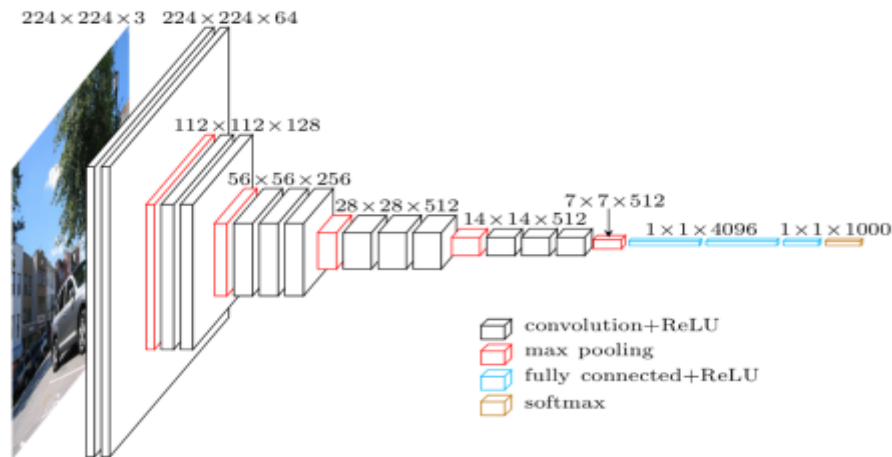


Figure 4.16 VGG 16 Model

Comparison and Analysis between the used Models:

Xception Model	Resnet 50	VGG 16
Validation accuracy:83%	Validation accuracy:85%	Validation accuracy:85%
Training Accuracy: 70%	Training Accuracy: 85%	Training Accuracy: 85%
Real Time Accuracy on Images: mostly 90%	Real Time Accuracy on Images: only 40-50% for some expressions	Real Time Accuracy on Images: only 30-40% for some expressions
Predicts all the expression equally well	Predicts only happy and neutral expression properly.	Predicts only neutral expression properly

Table 4.6 Comparison and Analysis between the models

CHAPTER 5

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

CHECKING IMAGE FROM TRAINING SET:

```
In [5]: plt.figure(figsize=(14,22))
i = 1
for expression in os.listdir(train_dir):
    img = load_img((train_dir + expression + '/' + os.listdir(train_dir + expression)[5]))
    plt.subplot(1,7,i)
    plt.imshow(img)
    plt.title(expression)
    plt.axis('off')
    i += 1
plt.show()
```



Figure 5. 1 Training Set

MODEL:LOSS AND ACCURACY PLOT

```
plt.plot(hist.history['val_loss'])
plt.title('model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

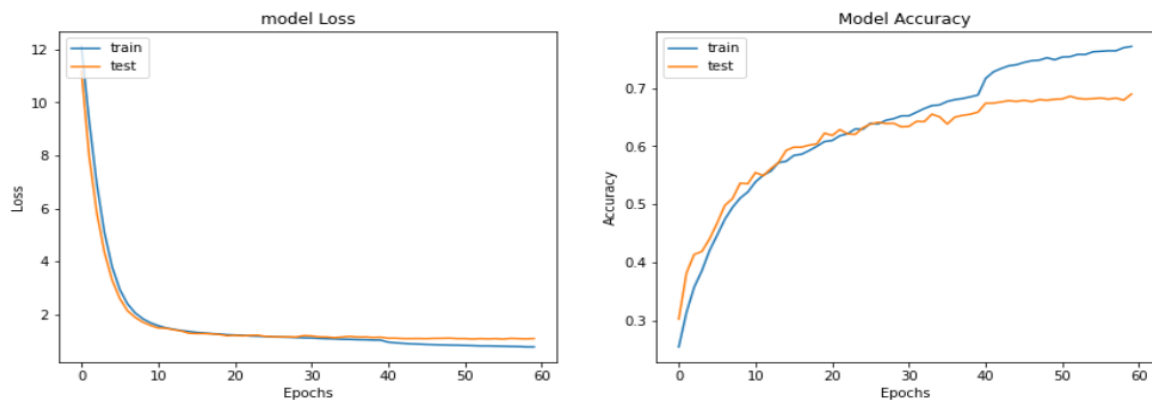


Figure 5. 2 Accuracy and Loss

CONFUSION MATRIX

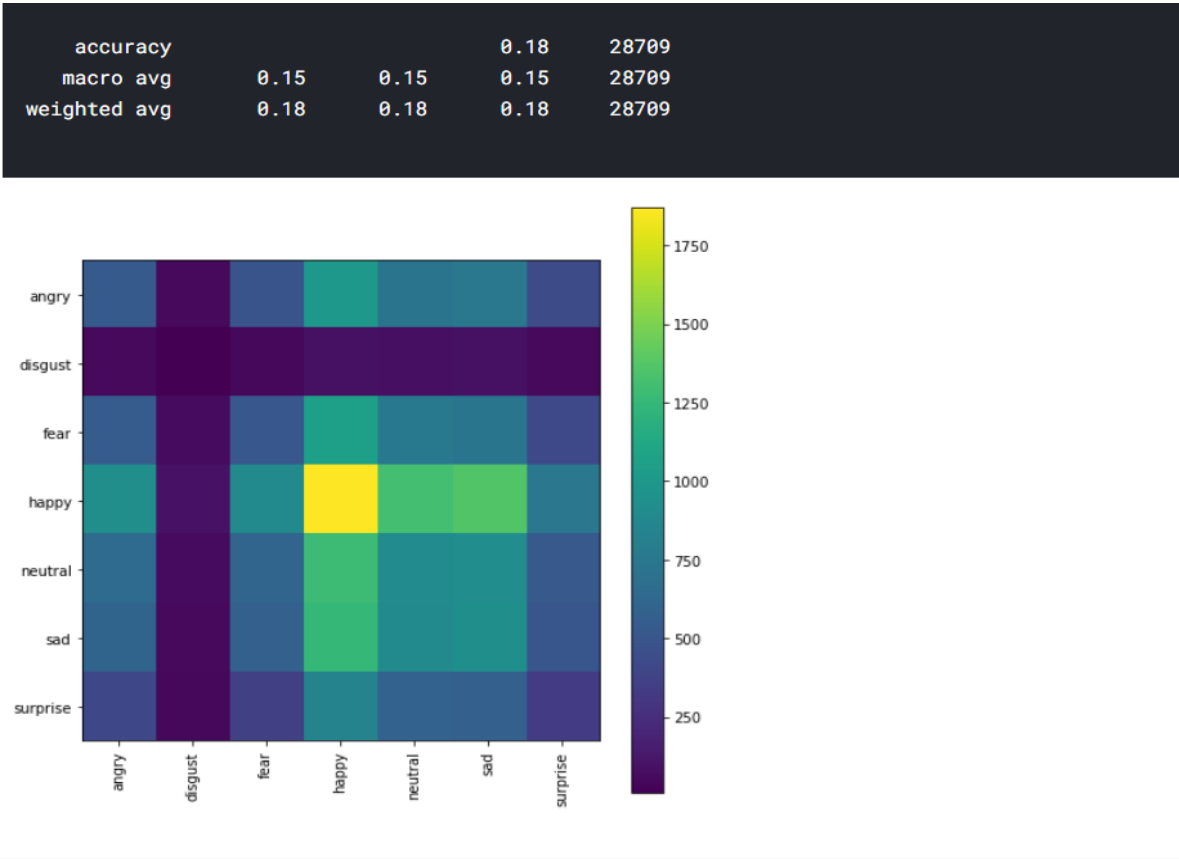


Figure 5. 3 Confusion matrix

USER INTERFACE:

TEST IMAGE1:

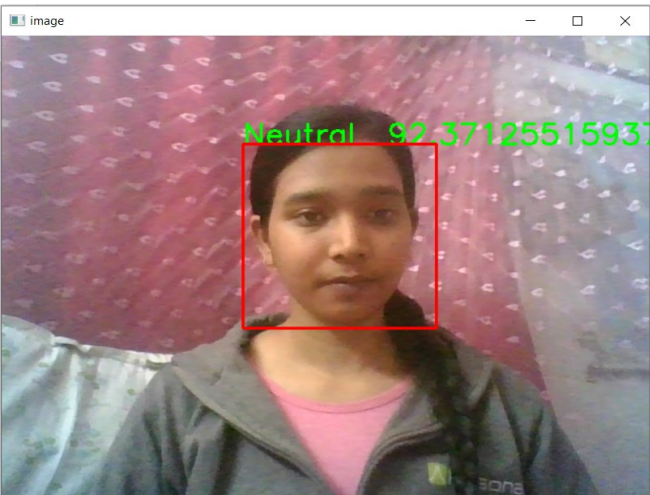


Figure 5. 4

TEST IMAGE 2:

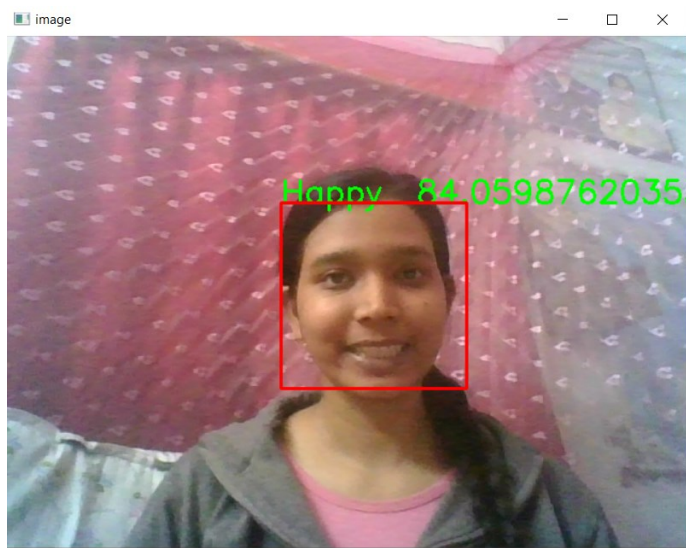


Figure 5. 5

TEST IMAGE 3:

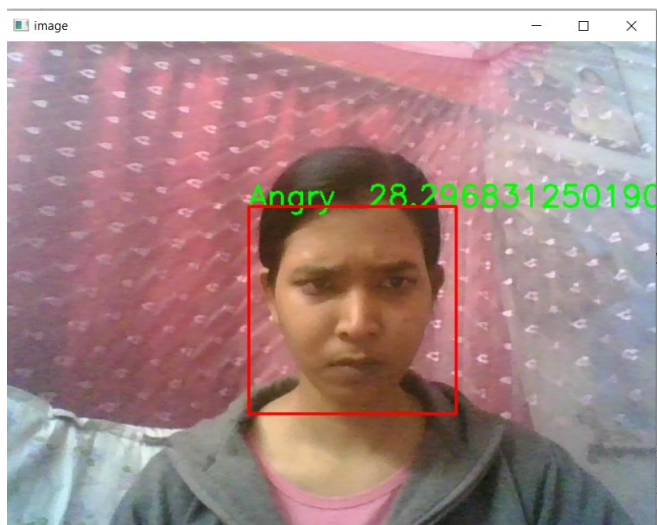


Figure 5. 6

TEST IMAGE 4:

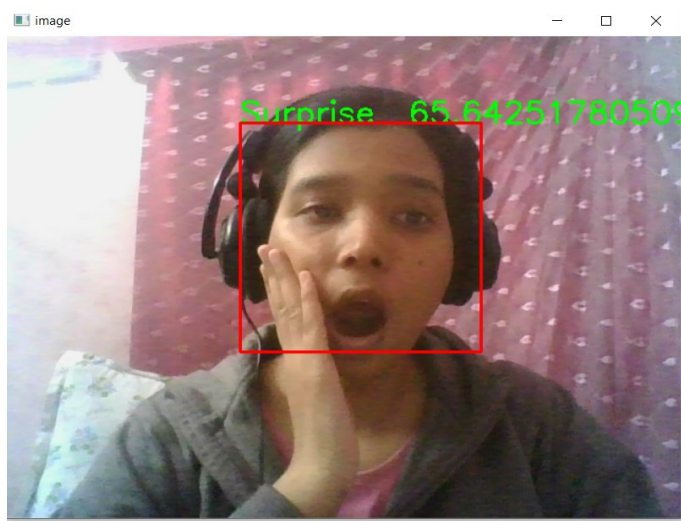


Figure 5. 7

CHAPTER 6

SUMMARY AND CONCLUSION

In this project, research to classify facial emotions over static facial images using deep learning techniques was developed. This is a complex problem that has already been approached several times with different techniques. While good results have been achieved using feature engineering, this project focused on feature learning, which is one of DL promises. While the results achieved were not state-of-the-art, they were slightly better than other techniques including feature engineering. It means that eventually DL techniques will be able to solve this problem given an enough number of labelled examples. While feature engineering is not necessary, image pre-processing boosts classification accuracy. Hence, it reduces noise on the input data. Nowadays, facial emotion detection software includes the use of feature engineering. A solution totally based on feature learning does not seem close yet because of a major limitation: the lack of an extensive dataset of emotions. For instance, ImageNet contest uses a dataset containing 14,197,122 images. By having a larger dataset, networks with a larger capability to learn features could be implemented. Thus, emotion classification could be achieved by means of deep learning techniques.

APPENDIX

SOURCE CODE:

```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.layers import Conv2D, Dense, BatchNormalization, Activation, Dropout,
MaxPooling2D, Flatten
from keras.optimizers import Adam, RMSprop, SGD
from keras import regularizers
from keras.callbacks import ModelCheckpoint, CSVLogger, TensorBoard,
EarlyStopping, ReduceLROnPlateau
import datetime
import matplotlib.pyplot as plt
from keras.utils import plot_model
train_dir = '../input/fer2013/train/'
test_dir = '../input/fer2013/test/'

row, col = 48, 48
classes = 7

def count_exp(path, set_):
    dict_ = {}
    for expression in os.listdir(path):
        dir_ = path + expression
        dict_[expression] = len(os.listdir(dir_))
    df = pd.DataFrame(dict_, index=[set_])
    return df
train_count = count_exp(train_dir, 'train')
```

```

test_count = count_exp(test_dir, 'test')
print(train_count)
print(test_count)
train_count.transpose().plot(kind='bar')
test_count.transpose().plot(kind='bar')
plt.figure(figsize=(14,22))
i = 1
for expression in os.listdir(train_dir):
    img = load_img((train_dir + expression + '/' + os.listdir(train_dir + expression)[5]))
    plt.subplot(1,7,i)
    plt.imshow(img)
    plt.title(expression)
    plt.axis('off')
    i += 1
plt.show()

```

CREATING TRAINING AND TEST SETS

```

train_datagen = ImageDataGenerator(rescale=1./255,
                                   zoom_range=0.3,
                                   horizontal_flip=True)

training_set = train_datagen.flow_from_directory(train_dir,
                                                batch_size=64,
                                                target_size=(48,48),
                                                shuffle=True,
                                                color_mode='grayscale',
                                                class_mode='categorical')

test_datagen = ImageDataGenerator(rescale=1./255)
test_set = test_datagen.flow_from_directory(test_dir,
                                           batch_size=64,
                                           target_size=(48,48),

```

```

        shuffle=True,
        color_mode='grayscale',
        class_mode='categorical')

training_set.class_indices

DEFINING MODEL

def get_model(input_size, classes=7):
    #Initialising the CNN
    model = tf.keras.models.Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu',
input_shape =input_size))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same',
kernel_regularizer=regularizers.l2(0.01)))
    model.add(Conv2D(256, kernel_size=(3, 3), activation='relu',
kernel_regularizer=regularizers.l2(0.01)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(512, kernel_size=(3,3), activation='relu', padding='same',
kernel_regularizer=regularizers.l2(0.01)))
    model.add(Conv2D(512, kernel_size=(3,3), activation='relu', padding='same',
kernel_regularizer=regularizers.l2(0.01)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

```

```

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(classes, activation='softmax'))

#Compiling the model
model.compile(optimizer=Adam(lr=0.0001, decay=1e-6),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

return model

fernet = get_model((row,col,1), classes)
fernet.summary()

plot_model(fernet, to_file='fernet.png', show_shapes=True,
show_layer_names=True)

chk_path = 'ferNet.h5'

log_dir = "checkpoint/logs/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

checkpoint = ModelCheckpoint(filepath=chk_path,
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             monitor='val_loss')

earlystop = EarlyStopping(monitor='val_loss',
                           min_delta=0,
                           patience=3,
                           verbose=1,
                           restore_best_weights=True)

```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',  
                               factor=0.2,  
                               patience=6,  
                               verbose=1,  
                               min_delta=0.0001)
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,  
                                                       histogram_freq=1)
```

```
csv_logger = CSVLogger('training.log')
```

```
callbacks = [checkpoint, reduce_lr, csv_logger]
```

Training Model

```
steps_per_epoch = training_set.n // training_set.batch_size
```

```
validation_steps = test_set.n // test_set.batch_size
```

```
hist = fernet.fit(x=training_set,  
                  validation_data=test_set,  
                  epochs=60,  
                  callbacks=callbacks,  
                  steps_per_epoch=steps_per_epoch,  
                  validation_steps=validation_steps)
```

LOSS AND ACCURACY PLOT

```
plt.figure(figsize=(14,5))  
plt.subplot(1,2,2)  
plt.plot(hist.history['accuracy'])  
plt.plot(hist.history['val_accuracy'])  
plt.title('Model Accuracy')  
plt.xlabel('Epochs')  
plt.ylabel('Accuracy')
```

```
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.subplot(1,2,1)
```

```
plt.plot(hist.history['loss'])
```

```
plt.plot(hist.history['val_loss'])
```

```
plt.title('model Loss')
```

```
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
```

```
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.show()
```

MODEL EVALUATION

```
train_loss, train_accu = fernet.evaluate(training_set)
```

```
test_loss, test_accu = fernet.evaluate(test_set)
```

```
print("final train accuracy = {:.2f} , validation accuracy =  
{:0.2f}".format(train_accu*100, test_accu*100))
```

```
fernet.save_weights('fernet_bestweight.h5')
```

Confusion Matrix and Classification on training set

```
y_pred = fernet.predict(training_set)
```

```
y_pred = np.argmax(y_pred, axis=1)
```

```
class_labels = test_set.class_indices
```

```
class_labels = {v:k for k,v in class_labels.items()}
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
cm_train = confusion_matrix(training_set.classes, y_pred)
```

```
print('Confusion Matrix')
```

```
print(cm_train)
```

```
print('Classification Report')
```

```
target_names = list(class_labels.values())
```

```
print(classification_report(training_set.classes, y_pred,  
target_names=target_names))
```

```

plt.figure(figsize=(8,8))
plt.imshow(cm_train, interpolation='nearest')
plt.colorbar()
tick_mark = np.arange(len(target_names))
_ = plt.xticks(tick_mark, target_names, rotation=90)
_ = plt.yticks(tick_mark, target_names)

```

CONFUSION MATRIX AND CLASSIFICATION ON TEST SET

```

y_pred = fernet.predict(test_set)
y_pred = np.argmax(y_pred, axis=1)
class_labels = test_set.class_indices
class_labels = {v:k for k,v in class_labels.items()}

#from sklearn.metrics import classification_report, confusion_matrix
cm_test = confusion_matrix(test_set.classes, y_pred)
print('Confusion Matrix')
print(cm_test)
print('Classification Report')
target_names = list(class_labels.values())
print(classification_report(test_set.classes, y_pred, target_names=target_names))

```

```

plt.figure(figsize=(8,8))
plt.imshow(cm_test, interpolation='nearest')
plt.colorbar()
tick_mark = np.arange(len(target_names))
_ = plt.xticks(tick_mark, target_names, rotation=90)
_ = plt.yticks(tick_mark, target_names)

```

USER INTERFACE

```

pip install gradio
from PIL import Image

```

```

import gradio as gr

def landclass(img):
    img = np.array(img)
    img = np.expand_dims(img,axis = 0)
    img = img.reshape(-1,48,48,1)
    result = fernet.predict(img)
    result = list(result[0])
    img_index = result.index(max(result))
    return(label_dict[img_index])

label_dict = {0:'Angry',1:'Disgust',2:'Fear',3:'Happy',4:'Neutral',5:'Sad',6:'Surprise'}

im = gr.inputs.Image()
iface = gr.Interface(
    fn = landclass,
    inputs = im,
    outputs = 'text'
)
iface.launch(share=True)

```

OUTPUT:

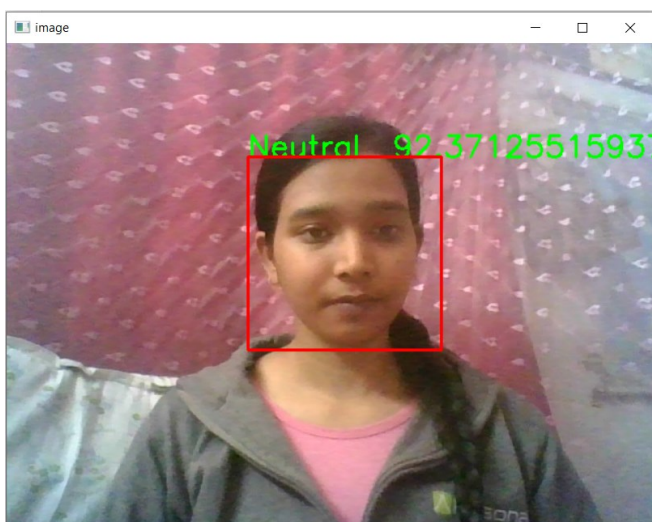


Figure 6. 1 Output Prediction

References

- D. H. Nguyen, S. Kim, G. Lee, H. Yang, I. Na and S. H. Kim, "Facial Expression Recognition Using a Temporal Ensemble of Multi-level Convolutional Neural Networks," in *IEEE Transactions on Affective Computing*, doi: 10.1109/TAFFC.2019.2946540.
- K. Mohan, A. Seal, O. Krejcar and A. Yazidi, "Facial Expression Recognition Using Local Gravitational Force Descriptor-Based Deep Convolution Neural Networks," in *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1-12, 2021, Art no. 5003512, doi: 10.1109/TIM.2020.3031835.
- K. Anderson and P. W. McOwan, "A real-time automated system for the recognition of human facial expressions," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 1, pp. 96-105, Feb. 2006, doi: 10.1109/TSMCB.2005.854502.
- M. Pantic and I. Patras, "Dynamics of facial expression: recognition of facial actions and their temporal segments from face profile image sequences," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 2, pp. 433-449, April 2006, doi: 10.1109/TSMCB.2005.859075.
- H. Zhang, A. Jolfaei and M. Alazab, "A Face Emotion Recognition Method Using Convolutional Neural Network and Image Edge Computing," in *IEEE Access*, vol. 7, pp. 159081-159089, 2019, doi: 10.1109/ACCESS.2019.2949741.
- H. Lee and K. Hong, "A study on emotion recognition method and its application using face image," *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, 2017, pp. 370-372, doi: 10.1109/ICTC.2017.8191005.
- R. Pathar, A. Adivarekar, A. Mishra and A. Deshmukh, "Human Emotion Recognition using Convolutional Neural Network in Real Time," *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, 2019, pp. 1-7, doi: 10.1109/ICIICT1.2019.8741491.
- C. Jain, K. Sawant, M. Rehman and R. Kumar, "Emotion Detection and Characterization using Facial Features," *2018 3rd International Conference*

and Workshops on Recent Advances and Innovations in Engineering (ICRAIE), 2018, pp. 1-6, doi: 10.1109/ICRAIE.2018.8710406.

- S. Liu, D. Li, Q. Gao and Y. Song, "Facial Emotion Recognition Based on CNN," *2020 Chinese Automation Congress (CAC)*, 2020, pp. 398-403, doi: 10.1109/CAC51589.2020.9327432.
- I. Lasri, A. R. Solh and M. E. Belkacemi, "Facial Emotion Recognition of Students using Convolutional Neural Network," *2019 Third International Conference on Intelligent Computing in Data Sciences (ICDS)*, 2019, pp. 1-6, doi: 10.1109/ICDS47004.2019.8942386.