UNIVERSITY OF
BIRMINGHAM
School of Computer Science

MSc Computer Science

Human Emotion Detection using Convolutional Neural Network

Divesh Sharma

Supervisor: Dr Harish Tayyar Madabushi

University of Birmingham

September 2020

ABSTRACT

As technology advances, deciphering the human brain, especially its emotional centre, is becoming the most important task to improve human-computer interaction. Detecting human emotion is a challenge, since the way people display their emotions vary each time. In this report, we propose a modified architecture for facial expression recognition that uses a convolutional neural network to address this challenge. We compared our model with the state-of-the-art results, while keeping the image database and execution format the same for fairness. Our model achieved an accuracy of 98.125%, the best result to our knowledge.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr Harish Tayyar Madabushi for his advice and guidance throughout this project. I would also like to thank Prof. Xiaolan Fu for providing access to the CASME II database.

Contents

Acronyms

ANN Artificial Neural Network. 6

API Application Programming Interface. 3

AdaGrad Adaptive Gradients. 13

Adam Adaptive Moment Estimation. 13

CASME Chinese Academy of Sciences Micro-Expression Database. 18

CK+ Cohn-Kanade Dataset. 18

CNN Convolutional Neural Network. 5

ELU Exponential Linear Unit. 8

JAFFE Japanese Female Facial Expressions. 18

MUG Multimedia Understanding Group. 18

OOM Out of Memory. 28

RMSprop Root Mean Squared Propagation. 13

ReLU Rectified Linear Unit. 7

SGD Stochastic Gradient Descent. 11

Chapter 1: Introduction and Motivation

As we enter a new era of human-machine interaction, a world where humans are able to enter machine space, a place that connects humans and machines virtually with a sense of physical touch, it is important to establish a network between the two. This network would allow machines to understand humans in a manner similar to how humans are learning the operations of a machine. Only then would we be able to develop a successful bi-directional communication between the human brain and a machine.

To achieve this, we would need the machine to learn more about humans, and the way a human brain works. One of the most difficult tasks in learning about humans is to understand human emotion.

Emotion is the key component of our daily lives. From planning to decision making, everything is influenced by emotions. Emotions are also the most crucial part of any human interaction and detecting human emotion is challenging; even from a human point of view, we make mistakes in recognising the emotions that other people display.

With time we learn to hide our emotions, and detecting a person's true emotion becomes all the more difficult. We would require the machine to correctly detect our emotions in order to give an appropriate response. With the increase in facial expression database, we can now make machines learn to detect parts of our emotion displayed by our facial features. However, to detect a human emotion accurately, we would not only require the visual data, but also the speech data along with the vitals such as the heart rate, breathing rate among others. But gaining access to such data is not possible given the current pandemic situation. So we decided to use the available facial data instead.

This report focuses on detecting emotion using facial recognition techniques. We use a Convolutional Neural Network to extract features from the images, and then an Artificial Neural Network would help classify the features. There are other state-of-the-art methods that detect and classify human emotions by using facial recognition techniques, but we aim to improve the performance of these methods.

The report is organised in the following chapters:

- Chapter 2 introduces background knowledge about convolutional layers, types of activation functions, optimizers, augmentation methods and key architectures that will give an insight into the project
- Chapter 3 explores related literature and gives an overview of the current state-of-the-art methods
- Chapter 4 introduces the project plan, explains the architecture used along with the implemented methods. It also gives a brief overview of the thought process while selecting the hyper-parameters
- Chapter 5 displays the experiments that were done and compares our results with the current state-of-the-art results
- Chapter 6 summarises the project and explores possible future work

Chapter 2: Background

2.1 APIs

2.1.1 TensorFlow distributed Keras

According to surveys, it is known that Keras is the number one API used for deep learning problems.(Chollet et al., 2015) This is because of few key reasons viz. Keras provides APIs that are simple and consistent which makes it easy to read and understand. Easy to read code increases productivity.

Also, Keras offers to build custom functions which makes it effortless to build our own workflows along with simple readable code. Simple readable code is a huge boost for beginners as it is easily understood and does not need extra learning hours.(Chollet et al., 2015)

TensorFlow DistributionStrategy API (Martıén Abadi et al., 2015b) supports multi-GPU training, which comparatively uses less time to train vis-a-vis training time needed by a CPU. The benefit of using Keras include the add-ons that it offers, like various callback options, batch normalization function, etc.

Other APIs that support the convolutional neural network are LightGBM, XGBoost, PyTorch, SciKit-learn, Fastai, Caffe, etc.

However, TensorFlow distributed Keras offers two major benefits over others, viz. simplicity without losing the complex nature of the program and flexible coding styles, so it is a clear winner of choice. (Chollet et al., 2015)

2.2 Models

2.2.1 The Sequential Model

The sequential model is the most common model that is used because of its simple layout. It comprises of a stack of layers, where each layer has a single input and a single output tensor. (Martıén Abadi et al., 2015d)

2.2.2 The Functional Model

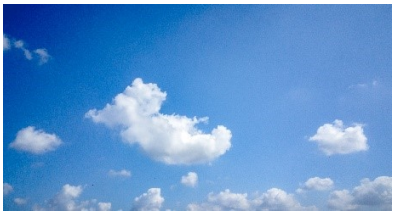
The functional model allows us to create more flexible models when compared to the sequential model. In the functional model we can build a multi-connected layered framework, i.e., when we need layer-sharing, or a non-linear topology and have multiple inputs or multiple outputs. (Martıén Abadi et al., 2015d)

2.3 Image Data Augmentation

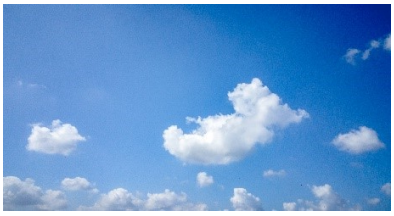
Image augmentation is used to increase the number of images present in our training set, this in turn helps us increase the diversity of our training data and helps the model learn better. (Martıén Abadi et al., 2015c)

Image augmentation is the process to apply realistic transformations on an image, like flipping an image or moving or rotating the image by a certain degree, within a given range. (Martıén Abadi et al., 2015c) For example, consider the image of the sky as shown in Figure: 2.1.

The first is an actual image of the sky. But in reality, there could be multiple variations of the same sky. Variation 1: Figure: 2.1b, is horizontally flipped but it is still identified as a sky, this increases variance in the image database. Variation 2: Figure: 2.1c is a slightly tilted version of a sky. Variation 3: Figure: 2.1d is converted to grey-scale.



(a) Normal Image



(b) Horizontally Flipped Image

(c) Rotated Image

(d) Grey-scale Image

Figure 2.1: Image transformations to increase variance of the training data

A sky might not be grey, but if we use a grey-scale input, by incorporating colour/intensity transformation on our images, we can further enhance the learning of our model.

2.4 Layers

2.4.1 Convolutional Layer

Convolutional layers, also known as the "conv" layer in Keras, is a layer that uses a feature detector (viz. kernel or feature extractor) which is convolved with the input to produce an output matrix of numbers or a tensor. (Martıén Abadi et al., 2015e) For grey-scale images, the output is a 2-dimensional tensor, where the images are converted into a 2-dimensional matrix of numbers, in the range of 0 to 255. (Martıén Abadi et al., 2015e) While in the case of a colour image, the output is a 3-dimensional tensor, which consists of three 2-dimensional matrices, for red, green and blue colours. (Martıén Abadi et al., 2015e)

In mathematics, the convolution between two functions, say f, g $\mathbb{R}^d \to R$ is defined as

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz \tag{2.1}$$

That is, we measure the overlap between f and g when one function is "flipped" and shifted by x. (Martıén Abadi et al., 2015e)

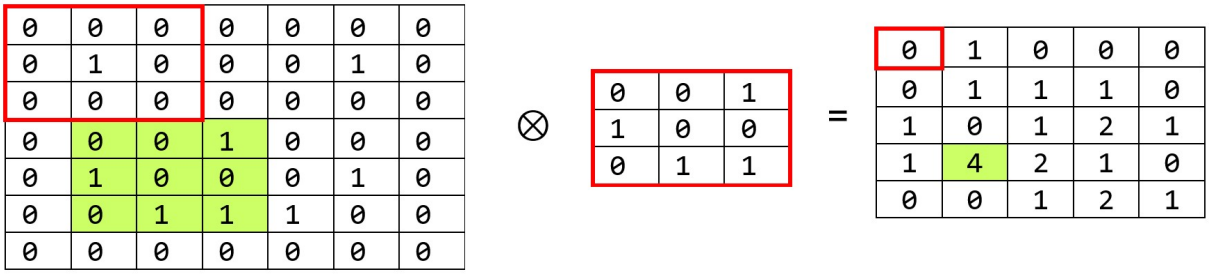


Figure 2.2: Example of a convolution operation (adapted from superdatascience.com, 2020)

The example in Figure: 2.2, shows convolution of a 7x7 smiley face using a feature detector of size 3x3, which in turn gives an output of size 5x5. The feature detector window slides over the entire input image and creates an output matrix which states the number of features matched.

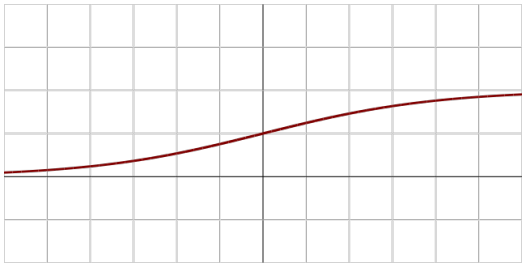
2.4.2 Activation Function

An activation function decides the output of that particular node in an artificial neural network. (Tayyar Madabushi, 2019) When stated in neuro-scientific terms, an activation function is a representation of the action potential "firing" of a "neuron". (Hodgkin and Huxley, 1952)

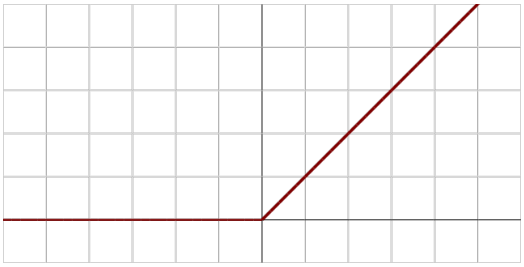
When the input, of that particular node, exceeds the set threshold, the neuron is said to "fire", in binary terms this would be a 1 (ON). Else, it "does not fire" or a 0 (OFF). (Zhang et al., 2020)

As stated by Zhang et al., 2020, "Activation functions decide whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it."

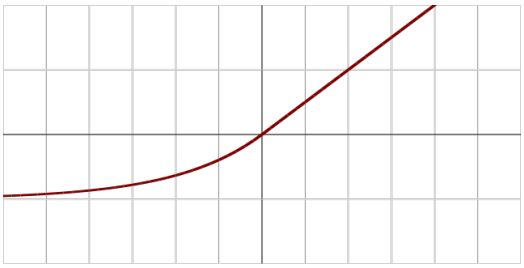
There are several activation functions available, like *Identity Function*, *Hyperbolic Tangent (TanH)*, *Sigmod*, *Rectified Linear Unit (ReLU)*, *Exponential Linear Unit (ELU)*, among others, we will however focus only on a few key functions for this project .



(a) Sigmoid (Garnier and Quetelet, 1838)



(b) ReLU (Nair and Hinton, 2010)



(c) ELU (Clevert, Unterthiner, and Hochreiter, 2015)

Figure 2.3: Activation Functions

Sigmoid

The Sigmoid function, as stated by Zhang et al., 2020, "is also known as the *squashing* function, because it *squashes* any input in the range (-inf, inf) to a value in the range of (0, 1)."

$$sigmoid(x) = \frac{1}{1 + exp(-x)} \tag{2.2}$$

As shown in Figure: 2.3a, it is non-linear with a smooth gradient within the range of 0 and 1 and the activation would be analogous in nature as compared to the step function. (Garnier and Quetelet, 1838)

However, as we move to the extreme ends of the graphs, we can notice that there is very little change in y-axis values, while values on the x-axis change drastically. This is known as the "vanishing gradient" problem and results in slower models. (Garnier and Quetelet, 1838)

ReLU

Rectified Linear Unit (ReLU) is the most used activation function due to its simplicity and its performance.

Given the element x, the function is defined as the maximum of that element and 0. (Zhang et al., 2020)

$$ReLU(x) = max(x, 0) \tag{2.3}$$

As shown in Figure: 2.3b, the ReLU function retains only positive elements and discards all negative elements (setting the corresponding activations to 0). (Zhang et al., 2020)

Although ReLU has a better gradient proportion when compared to Sigmoid function, it is unbounded and cannot be derived when the input is exactly 0. (Nair and Hinton, 2010)

Furthermore, when the learning rate is set too high, ReLU nodes can sometimes become inactive or in a dead like state for almost all inputs. This causes the node to get stuck in an inactive state and it eventually "dies". This is known as the dying ReLU problem and is a form of the vanishing gradient. (Nair and Hinton, 2010)

ELU

Exponential Linear Unit (ELU) is very similar to ReLU, but it fixes few problems that ReLU possesses. (Clevert, Unterthiner, and Hochreiter, 2015)

ELU can be mathematically states as following (Clevert, Unterthiner, and Hochreiter, 2015),

$$ELU(x) = \begin{cases} x & if \quad x > 0 \\ \alpha(e^x - 1) & if \quad x < 0 \end{cases} \tag{2.4}$$

This means when the input is greater than 0, the output (value on the y-axis) will be equal to the input (value on the x-axis). But when the input (value on the x-axis) is less than 0, the corresponding output will be slightly below 0. We can see the graphical representation for ELU in Figure: 2.3c. (Clevert, Unterthiner, and Hochreiter, 2015)

The ELU function produces non-zero activations instead of zero and avoids the "dead" ReLU problem. Based on the experiments carried out by Clevert, Unterthiner, and Hochreiter, 2015, ELUs have lower noise when compared to other activation functions like ReLU or Leaky ReLU.

2.4.3 Pooling Layer

Let us consider an example of Figure: 2.1, and assume that we have a CNN which detects clouds in an image. Here we have four different images of clouds. The clouds are differently positioned and in one of the image, it is tilted at an angle. We want our neural network to recognise all four images, that contain cloud. But if we strictly train our neural network on two of the images while testing the other two, it will keep looking for the exact same feature in other images and would never detect it as a cloud even if there was a slight difference in the features.

This is why we must make sure that our neural network is spatially invariant.(Scherer, Müller, and Behnke, 2010) This means that even if the features are fuzzy or distorted, our model must have enough flexibility to detect those features. For this reason we use a pooling layer in our model. Pooling also helps reduce the size and the number of parameters required to train. Removing information would also avoid over-fitting. (Cireşan, Meier, Masci, et al., 2011)

The different types of pooling are Max Pooling, Sum Pooling, Average Pooling also known as sub-sampling. In Figure: 2.4, here we perform the Max Pooling operation.

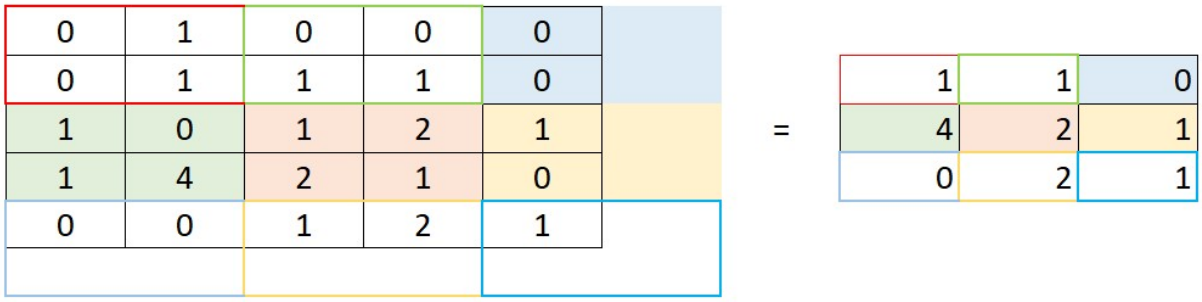


Figure 2.4: Example of a Max Pooling operation (adapted from superdatascience.com, 2020)

We have taken a stride of 2, where we move the 2x2 filter by 2 spaces. Stride of 2 is very common, but it's not compulsory to always take the same stride size, we can also keep the stride size as 1 or 3 or change it as per our needs. (Scherer, Müller, and Behnke, 2010) Now the maximum value is taken from the first window (with red borders). The maximum value here is 1, which was then given to the output, as shown. If this was average pooling, then we would have taken the average of all the values present in the window. (Scherer, Müller, and Behnke, 2010)

Since we are taking the maximum values, we are eliminating any distortion or fuzziness in the image. In doing so, we are preserving all the features and account for any textural or spatial noise. Even if we rotate the image a little, the maximum values taken would still be the same. (Cireşan, Meier, Masci, et al., 2011)

2.4.4 Normalization Layer

Batch normalization was first proposed by Sergey Ioffe and Christian Szegedy in 2015. It is a method which mitigates the problem of internal covariate shift. (Ioffe and Szegedy, 2015)

As Ioffe and Szegedy, 2015 describes the phenomenon *"internal covariate shift"*, is when the training of deep neural networks is slowed down, due to the fact that the parameters of layers change and this causes the distribution of next layer's input while the model undergoes training.

As Ioffe and Szegedy, 2015 stated Batch Normalization normalises each training mini-batch. This relaxes the tension of parameter initialization by allowing the model architecture to use higher learning rates.

2.4.5 Flattening Layer

In the flattening layer, we take the pooled output, which is in a 2-dimensional tensor and "flatten" it. This means that we convert the 2-dimensional matrix to a single dimensional matrix. This is because we want to take the output from the pooling layer and feed it to an Artificial Neural Network (ANN) or a Fully Connected Layer, given that the ANN takes a single dimensional matrix as its input.

2.4.6 Fully Connected Layer

The fully connected layer is also known as the dense layer in Keras. (Chollet et al., 2015) Similar to a regular artificial neural network, neurons are connected to the activations in the previous layers. This layer processes all the high level reasoning in the model.

2.4.7 SoftMax

SoftMax function is used for normalising the the output to a probability distribution over predicted classes. (Goodfellow, Bengio, and Courville, 2016) It is mathematically expressed as shown in equation 2.5, (Goodfellow, Bengio, and Courville, 2016)

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2.5}$$

2.5 Optimizers

2.5.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD), calculates the gradient of loss function by performing a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$: (Ruder, 2017)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.6}$$

where,

θ is a parameter for the neural network

η is the learning rate

∇ is the gradient

J is the loss function

As denoted by Ruder, 2017, the psuedo code for SGD can be stated as following,

```
for i in range(nb_epochs)
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function,example,params)
        params = params - learning_rate * params_grad
```

The main problem with SGD is that it can easily get stuck in a local minima. (Ruder, 2017) As Ruder, 2017 states, learning rates must be carefully chosen, as a very high learning rate would lead to fluctuations in the loss function and can hinder convergence.

2.5.2 Momentum

Momentum is very similar to SGD, but with just one extra term. As the name suggests, it adds momentum to the equation. As said by Qian, 1999, system near the local minima behaves similar to a set of damped and coupled harmonic oscillators.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) + \gamma v_t \tag{2.7}$$

where,

γ is the momentum constant,

and v_t is the last update to θ

(a) SGD with momentum (b) SGD without momentum

Figure 2.5: Source: (Orr, Schraudolph, and Cummins, 1999)

Optimizing the loss can be thought of as a vertical oscillation. If the vertical displacement is large, progress along towards the local minima would be slower. Dampening the oscillation would help us reach the local minima quicker, this can be done by adding momentum to the oscillations. (Ruder, 2017)

2.5.3 Adam

Adaptive Moment Estimation (Adam) makes use of adaptive learning rate and momentum (2.5.2) to converge quicker than most other optimisers. (Ruder, 2017)

AdaGrad, AdaDelta, RMSprop and Adam use adaptive learning rate, however the most relevant optimisers to understand Adam are RMSprop and AdaGrad.

Adaptive Gradients (AdaGrad) allows us to change the learning rate over time. With this we can have variable updates. As Ruder, 2017 states, "It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters."

Root Mean Squared Propagation (RMSprop) has a lower convergence rate than AdaGrad, but it still is faster than most other optimizers. This is because RMSprop provides us with an exponentially decaying average when compared to AdaGrad's sum of gradients. (Ruder, 2017)

Adam has a similar decaying average as that of RMSprop, but instead of having just one decay term like RMSprop, Adam makes use of two such decay terms, given as $\beta 1$ and $\beta 2$ in the algorithm shown as Figure: 2.6. (Kingma and Ba, 2015)

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 2.6: Adam, algorithm for stochastic optimization by Kingma and Ba, 2015

2.6 Architectures

2.6.1 AlexNet

AlexNet, as shown in Figure: 2.7, was created by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton for the ImageNet Large-Scale Visual Recognition Challenge in 2012.

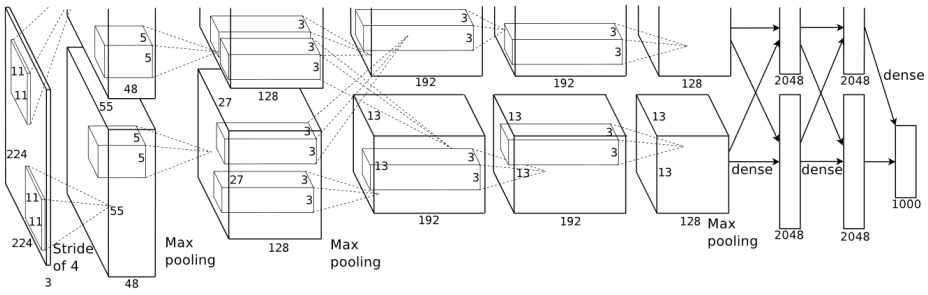


Figure 2.7: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000. (Krizhevsky, Sutskever, and Hinton, 2012)

They created a convolutional neural network which consists of 5 convolutional layers and 3 fully connected layers, for classifying 1000 possible categories with ReLU as the activation function. (Krizhevsky, Sutskever, and Hinton, 2012) ImageNet was the database used on this CNN, which consists of over 15 million labelled high-resolution images which belongs to more than 22,000 categories.

2.6.2 LeNet5

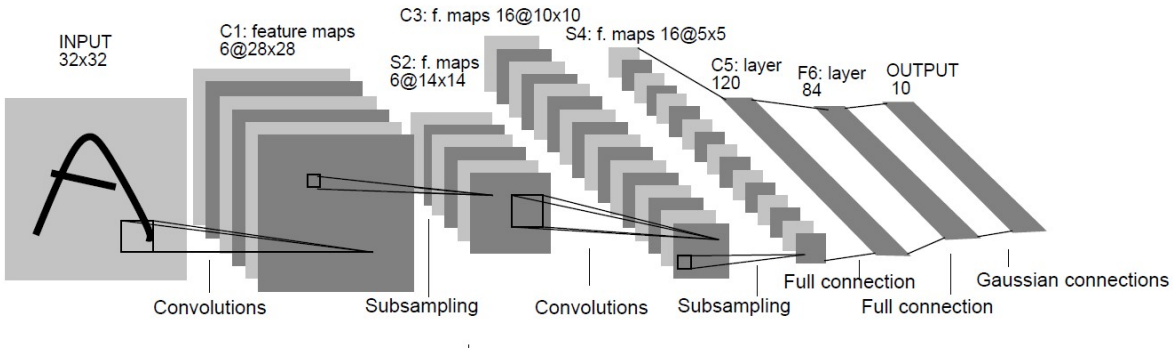


Figure 2.8: Lecun et al., 1998's LeNet-5

Lecun et al., 1998's LeNet-5, as shown in Figure: 2.8, consists of 7 layers, excluding the input layer. The input to the model is a 32x32 pixel image. Which is then sent to the first convolutional layer with 6 feature maps of size 28x28 with a kernel of size 5x5. The output from C1 is then sent to a sub-sampling layer, S2, with 6 feature maps of size 14x14 and a kernel of size 2x2. S2 is then connected to another convolutional layer C3, of size 10x10, with 16 feature maps and a kernel of size 5x5. Output of C3 is connected to a sub-sampling layer S4, with with 16 feature maps of size 5x5 and a kernel size of 2x2. S4 is connect to the C5 layer with a kernel size of 1x1, i.e., it flattens out the image so as to create a full connection with the artificial neural network in the following layers. Layer F6 consists of 84 units which are then connected to the output layer. (Lecun et al., 1998)

2.6.3 VGG Net

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.9: Simonyan and Zisserman, 2014's VGG Architecture

The VGG architecture, as shown in Figure: 2.9, consists of 3x3 filters with a stride of 1 with MaxPooling layers of stride 2. All hidden layers use ReLU as the activation function. It has 3 fully connected layers and a SoftMax layer in the end to normalize the output. (Simonyan and Zisserman, 2014) The increasing filter units suggests an increase in the depth and shrink of spatial dimensions.

Chapter 3: Related Work

Over the years, there have been several research papers published, that have focused their research on detecting emotions by recognising various facial expressions. Most of the papers available online have used support vector machines and nearest neighbour classifier to detect and classify micro-expressions into their respective emotion category.

Some of the papers have also used image processing techniques to reduce the dimensionality of the image while maintaining its spatial structure information, along with classifiers such as Extreme Learning Machine (ELM) to classify various micro-expressions. Wang, Chen, et al., 2013 uses Discriminant Tensor Subspace Analysis (DTSA) and ELM, to reduce the dimensionality of the input. DTSA uses two sided transformation and evaluates the input as a second order tensor while maintaining the spatial structure.

While few papers have used colour information as a fourth dimension to enhance the performance of the classifier along with creating Regions Of Interest (ROIs) by partitioning the facial area. (Wang, Yan, Li, Zhao, Zhou, et al., 2015) The two most used combinations are Discriminant Tensor Subspace Analysis (DTSA) with Extreme Learning Machine (ELM) and the other being Main Directional Mean Optical-flow (MDMO) with Support Vector Machines (SVMs) (Y. Liu et al., 2015) or nearest neighbour classifiers.

P. Liu et al., 2014 presented a novel Boosted Deep Belief Network (BDBN), to perform feature learning, feature selection and classifying the features.

However, the best results that we found were of Lopes, de Aguiar, and Oliveira-Santos, 2015, who had the highest accuracy rate 93.74%. Their architecture comprises of five layers. Two convolutional layers and two sub-sampling layers and one fully connected layer of 256 units. The two convolutional layers they used were of sizes 32 and 64 units, as shown in Figure: 3.1. Their final output was of 6 nodes representing each one of the six expressions for classifying. (Lopes, de Aguiar, and Oliveira-Santos, 2015)

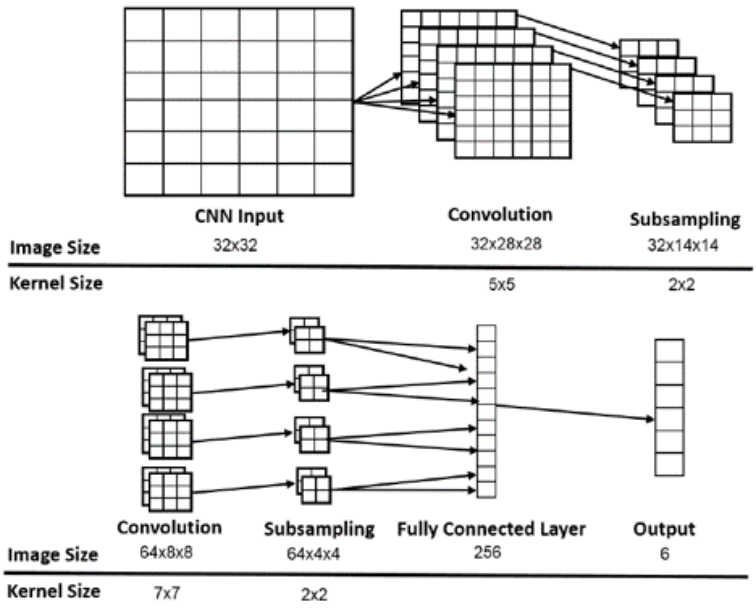


Figure 3.1: Lopes, de Aguiar, and Oliveira-Santos, 2015's architecture model

Convolutional Neural Networks were introduced by Lecun et al., 1998. The major advantage of using a CNN is its ability to detect features from an image, without the need to manually code or define any features. Lecun et al., 1998 uses the LeNet-5, as described in section: 2.6.2, to recognise handwritten text.

The most used databases for spontaneous micro-expressions are SMIC, CASME and CASME II, while other micro-expression databases include USF-HD, Polikovsky's database. Also, to train the model for facial recognition few databases used are CK+ (Cohn-Kanade Dataset), MUG (Multimedia Understanding Group), MMI, JAFFE (Japanese Female Facial Expressions), Multi-PIE dataset.

Chapter 4: Methodology

4.1 Course of Action

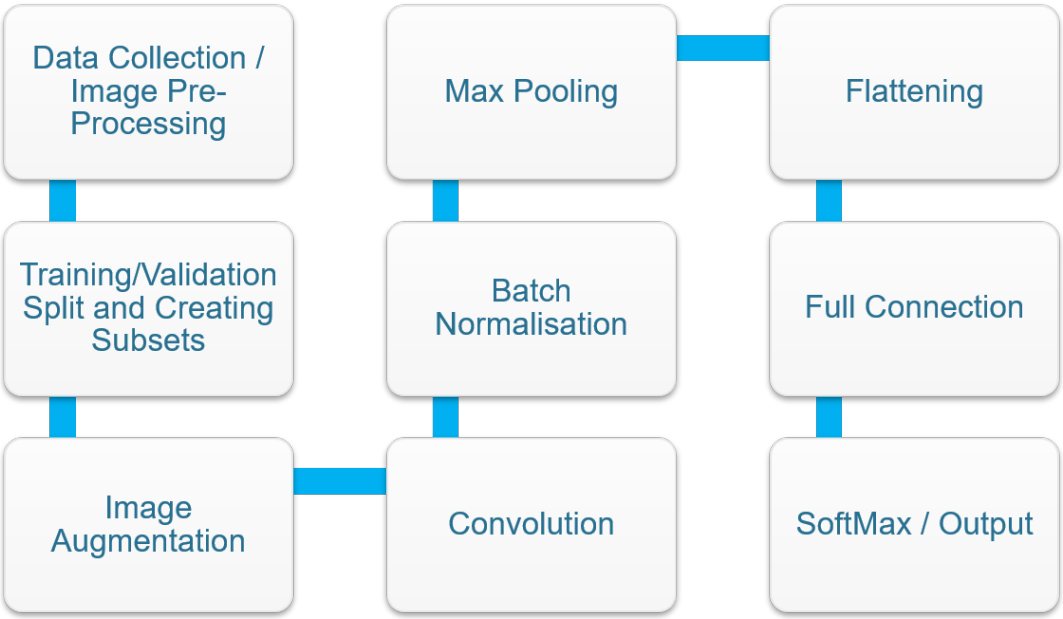


Figure 4.1: Basic layout of the plan

First, we searched and collected relevant databases, that would be used in our project. The different databases that we looked into were, CASME II, CK+, JAFFE, SASE-FE and SMIC databases. Once we collected the data, we pre-processed it, i.e., if the database had only videos, then we split the videos into image frames and defined the apex frame, where the expression was captured.

Once we had our frames ready, we then cropped the images, such that, only the face was visible. Some databases like CK+ already had cropped images. However, the images were not ordered into directories as was required. We then arranged all files into appropriate directories, in a format such that each emotion had its own folder, and so there were 8 folders in total. Next we split our database into training set and validation set. After splitting the database, we augmented our images, as explained in section: 2.3.

Later we trained our model, by applying convolution to our input images, which were sent in batches. Once our model completed training, we applied batch normalisation and pooled our data using Max Pooling. We then passed it through another round of convolution, normalisation and pooling layers as presented in the model architecture in Figure: 4.2. Once we were done with our convolutional part of the architecture, we flattened the output so that we could send it to the artificial neural network section, i.e. the fully connected layers. Here the ANN decides the category of the image and sends it to the Soft-Max layer, which normalises the output, so that it can be correctly classified.

4.2 Architecture

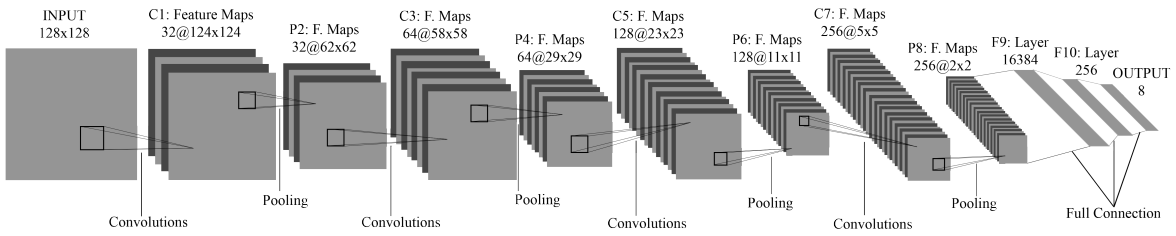


Figure 4.2: Model Architecture adapted from Lecun et al., 1998

This architecture, as shown in Figure: 4.2, is loosely based on Lecun et al., 1998's LeNet-5 architecture, but our model uses a combination of 5x5 and 7x7 kernels and has 11 layers rather than the 7 layers with only a 5x5 kernel of a standard LeNet-5. While LeNet-5 used sub-sampling, we have used Max Pooling (refer section: 2.4.3).

Layer C1 is a convolutional layer with 32 feature maps of size 124x124. Each unit of this feature map is connected to 5x5 neighborhood in the input. C1 contains 832 trainable parameters.

Layer P2 is a pooling layer with 32 feature maps of size 62x62. Each unit in the feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The number of rows in and columns in P2 is half of those in C1 because the receptive fields are non-overlapping. P2 contains 128 trainable parameters.

Layer C3 is a convolutional layer with 64 feature maps of size 58x58. Each unit of this feature map is connected to 5x5 neighborhood in the input. C3 contains 51264 trainable parameters.

Layer P4 is a pooling layer with 64 feature maps of size 29x29. Each unit in the feature map is connected to a 2x2 neighborhood in the corresponding feature map in C3. P4 contains 256 trainable parameters.

Layer C5 is a convolutional layer with 128 feature maps of size 23x23. Each unit of this feature map is connected to 7x7 neighborhood in the input. C5 contains 401536 trainable parameters.

Layer P6 is a pooling layer with 128 feature maps of size 11x11. Each unit in the feature map is connected to a 2x2 neighborhood in the corresponding feature map in C5. P6 contains 512 trainable parameters.

Layer C7 is a convolutional layer with 256 feature maps of size 5x5. Each unit of this feature map is connected to 7x7 neighborhood in the input. C7 contains 1605888 trainable parameters.

Layer P8 is a pooling layer with 256 feature maps of size 2x2. Each unit in the feature map is connected to a 2x2 neighborhood in the corresponding feature map in C7. P8 contains 1024 trainable parameters.

Layer F9 is a fully connected layer, with 16384 nodes. This layer contains 16793600 trainable parameters.

Layer F10 is a fully connected layer, with 256 nodes. This layer contains 4194560 trainable parameters.

The OUTPUT layer/Soft-Max layer contains 8 nodes (because we must classify into 8 different classes). This layer contains 2056 trainable parameters. Soft-Max is used to normalize the output between 0 and 1, so that the image can be correctly classified.

Layer Parameters				
Layer	Units	Feature Detector	Stride	Activation
conv2D	32	5x5	1x1	ELU
maxpool	32	2x2	2x2	
conv2D	64	5x5	1x1	ELU
maxpool	64	2x2	2x2	
conv2D	128	7x7	1x1	ELU
maxpool	128	2x2	2x2	
conv2D	256	7x7	1x1	ELU
maxpool	256	2x2	2x2	
Full Connection				
FC	16384	1x1	1x1	ELU
FC	256	1x1	1x1	ELU
softmax				

Table 4.1: Layer Parameters in Detail

4.3 Data Collection

To create a good facial expression detector, a large amount of data is needed. To meet our objective of expression detection, we assumed that high quality data was required. In order to do so, we requested the CASME II database, as their videos were shot at 200fps and their micro-expression samples were spontaneous and dynamic. This consists of 26 subjects with a total of 256 videos and 7 emotions. However, after experimenting with the database (as given in section: 5.1) it was concluded that the database did not have enough subject diversity to provide good results.

We then came across the CK+ database with 7 emotions, which were later converted to 8 emotion categories, as described in Section: 4.4.

4.4 Data Split

While working with the CAMSE II database, we used the inbuilt split function provided by Keras. We kept 20% of the data for validation while the rest was used in training.

VAR3															
Subset1		*Subset2*		*Subset3*		*Subset4*		*Subset5*		*Subset6*		*Subset7*		*Subset8*	
anger	30	anger	12	anger	12	anger	9	anger	12	anger	18	anger	21	anger	21
contempt	3	contempt	3	contempt	3	contempt	6	contempt	3	contempt	3	contempt	9	contempt	24
disgust	15	disgust	30	disgust	24	disgust	30	disgust	24	disgust	27	disgust	18	disgust	9
fear	6	fear	15	fear	9	fear	6	fear	6	fear	12	fear	3	fear	18
happy	24	happy	24	happy	36	happy	27	happy	33	happy	24	happy	30	happy	9
neutral	39	neutral	41	neutral	42	neutral	40	neutral	40	neutral	45	neutral	40	neutral	40
sadness	12	sadness	3	sadness	9	sadness	6	sadness	6	sadness	15	sadness	12	sadness	21
surprise	27	surprise	36	surprise	33	surprise	36	surprise	36	surprise	36	surprise	27	surprise	18
Total	**156**	**Total**	**164**	**Total**	**168**	**Total**	**160**	**Total**	**160**	**Total**	**180**	**Total**	**160**	**Total**	**160**

Figure 4.3: CK+ split into 8 subsets - 3^{rd} variation

The CK+ (Cohn-Kanade) database originally consisted of 7 emotions, i.e., anger, contempt, disgust, fear, happiness, sadness and surprise. The eighth emotion, "neutral" was created by taking the first frame from each video and placing it in the neutral folder.

The CK+ database consists of 327 videos in total, which were then divided into the 7 emotions mentioned above. Considering the last frame as the apex frame, last 3 frames were taken from each video. This totaled upto 981 images. The neutral emotion set added another 327 images, which made it into a total of 1308 images.

As shown in Figure: 4.3, we divided our database of 1308 images into 8 subsets. It took us 3 variations to normalise the contents of each subset, so that the subsets did not have any overlapping subjects.

Out of the 8 subsets, 1 subset was used for validation and the other 7 subsets were used for training the model. We ran our model 8 times, so that each subset was tested upon.

4.5 Image Augmentation

For augmenting our image data, we generated synthetic samples which were rotated in a 30deg range. The width and height of these images were also shifted in the range of 0-30% along with horizontal flips. We also used a shear range and a zoom range of 20% on the images. All images were converted to grey-scale, since we were not working with the colour dimension, and using the third dimension would only increase the processing load.

4.6 Convolution Section

While working with the CASME II database, we first used the VGG net - A architecture, as described in Section: 2.6.3. We made a few changes to the weights, i.e., instead of using 64 units as the starting layer, we used a convolution layer of 32 units. Also, VGG Net was designed using conv3 layers, instead we used conv2 layers, since we converted our images to grey-scale. But, the results with the CASME II database were not upto the mark, as shown in section: 5.1.

We came across a paper (Lopes, de Aguiar, and Oliveira-Santos, 2015), which showed results of over 90% while using the CK+ database. This motivated us to use the CK+ database as a focal point, given its coverage of diverse subject data. Once we had our goal in mind, we began searching for other papers that would display better results than Lopes, de Aguiar, and Oliveira-Santos, 2015, but found this to be the best. For the CK+ dataset, we first experimented using the VGG architecture. Initially we were also unable to produce great results with the VGG architecture, as shown in section 5.2. After multiple attempts at improving the VGG network, we were left with some good hyper-parameters that included, the learning rate and the activation function, but we still were unable to produce the desired results.

We then decided to explore further into the origins of the CNN and discovered that it started from Lecun et al., 1998's paper that had LeNet-5 at its core. This revelation encouraged us to base our model on the LeNet-5 architecture, with some variations to it, because the basic model was already tested by Lopes, de Aguiar, and Oliveira-Santos, 2015 and it did not give the best results.

4.7 Choosing the Hyper-parameters

After multiple attempts at optimising the LeNeT-5 architecture, we decided to divide the model into two sections. The first section or the first half would use a smaller feature detector, while the second half would use a comparatively larger feature detector.

Since the original LeNeT-5 architecture used feature detectors of size 5x5, the first half of the architecture preserved that detail, while the second half used a feature detector of size 7x7.

With the architecture now in place, the next key decision point was finalising the type of sub-sampling layer. Lecun et al., 1998's model used average sampling, or sub-sampling between its convolutional layers. Since we were using two different feature detectors, we decided to use Max Pooling, simply because it preserves the maximum features of the image, as described in section: 2.4.3. However we observed certain limitations in Max Pooling during the analysis stage, discussed in detail in section: 5.4

4.8 Fully Connected Layers

After the pooling stage, we flattened our image and the first full connection layer was of size 16384 units. We decided on this specific number of units because, the input to the model was a 128x128 pixel image, when multiplied the value was 16384. The formula we applied for choosing this particular node was to multiply the width of the image to its height.

The next fully connected layer was chosen similarly. Instead of multiplying, we added the width of the image to its height, which gave us the value 256.

The final node was the Soft-Max node, since we had 8 classes to categorise into, this layer consisted of 8 units. The Soft-Max layer normalises the output, for easy classification as explained in section: 4.2.

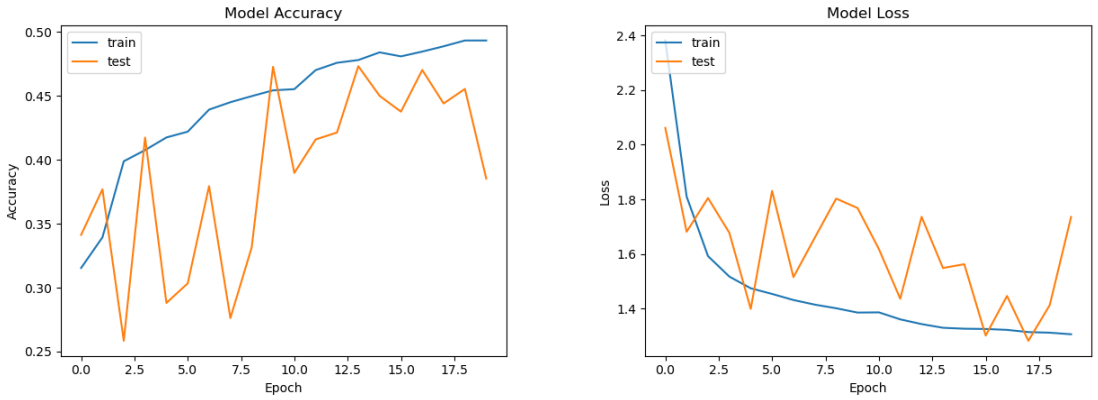
4.9 Classifying Individual Emotions

We tried to analyse the predictive performance for each emotion using a confusion matrix. The confusion matrix usually works well with binary classification. Since we had used classical cross-entropy with 8 classes, the confusion matrix was not working as expected and would throw "out of axis" error.

The other alternate that we could have used to analyse the predictive performance for each emotion was by setting up a binary classifier and by classifying each emotion one by one. However, this method of classification would significantly require more time, as we would have to first move all the emotion files into a directory except the ones that were going to be tested upon. Then we would need to modify the code appropriately. Given the limited available time, this alternate option was not pursued as it had very limited material impact on the overall result.

Chapter 5: Experiments and Results

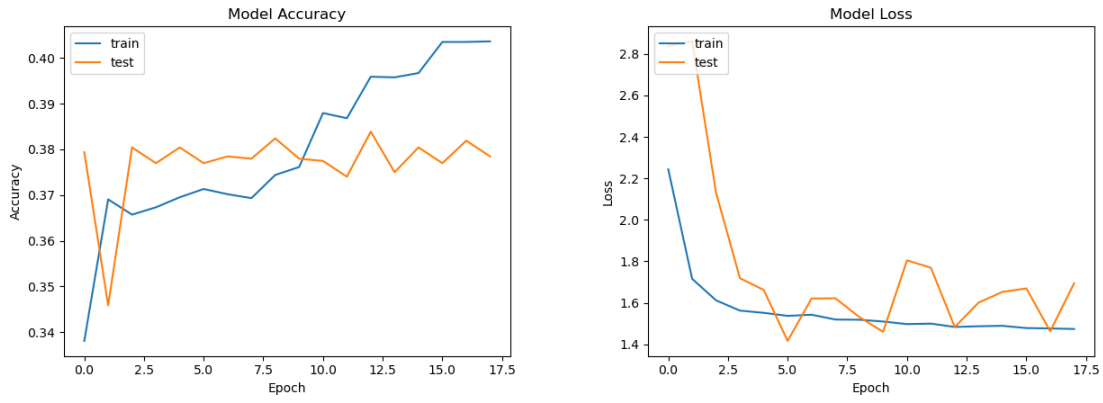
5.1 Experiments with CASME II Database



(a) Accuracy (Max at 47%)

(b) Loss (Min at 1.2)

Figure 5.1: VGG with ELU with 32, 64, 128 and 256 layer units units, with 0.2 and 0.5 dropout rates, and 0.0001 learning rate



(a) Accuracy (Max at 38.2%)

(b) Loss (Min at 1.439)

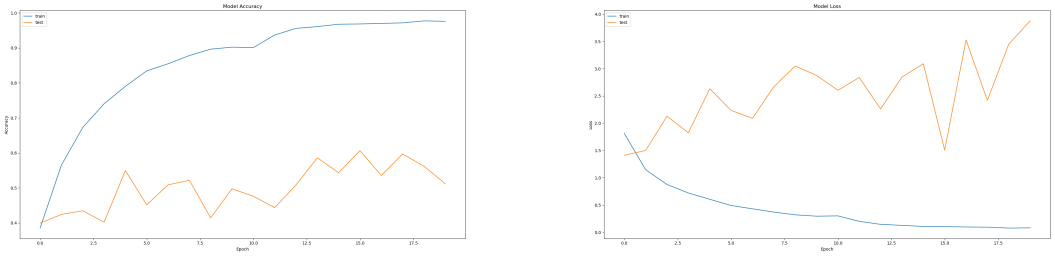
Figure 5.2: VGG with ELU with 32, 64, 128, 256 and 512 layer units units, with 0.2 and 0.5 dropout rates, and 0.0001 learning rate

As you can see from the graph results in Figure: 5.1 and Figure: 5.2, the maximum accuracy our VGG model reaches was that of 47%. When compared with other state-of-the-art results, this reflected a very low accuracy level. Even though the model performance was note-worthy, the overall accuracy was very low.

5.2 Experiments with CK+ Database

5.2.1 VGG with ReLU

The VGG model provided better results with the CK+ database, at 60% accuracy when compared to that of CAMSE II database, which was at 47% accuracy.



(a) Accuracy (Max at 60%)

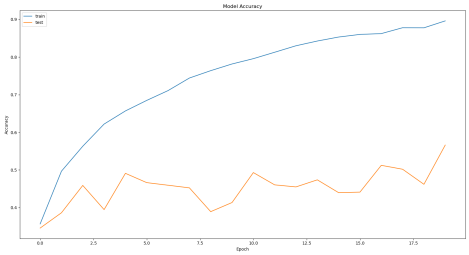
(b) Loss (Min at 1.4)

Figure 5.3: VGG with ReLU with 32, 64, 128 and 256 layer units units, with 0.2 and 0.5 dropout rates, two dense layers of 64 units each and 0.001 learning rate

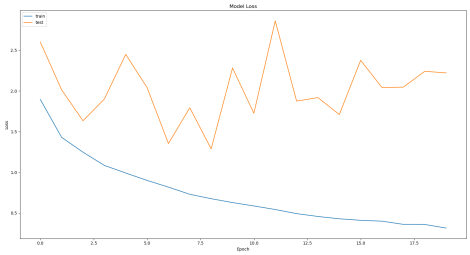
5.2.2 VGG with ELU

When we compared ELU to ReLU, ELU seemed to perform better when the hyper-parameters were kept the same, in Figure: 5.7 when compared to ReLU in Figure: 5.3.

However, VGG as a model performed poorly when compared with other state-of-the-art results, as given by Lopes, de Aguiar, and Oliveira-Santos, 2015.

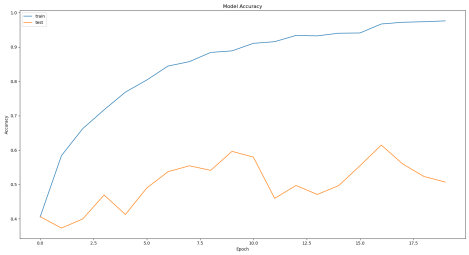


(a) Accuracy (Max at 57%)

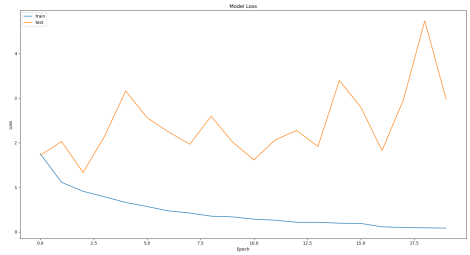


(b) Loss (Min at 1.4)

Figure 5.4: VGG with ELU with 32, 64, 128 and 256 layer units units, with 0.2 and 0.3 dropout rates, two dense layers of 64 and 32 units and 0.0001 learning rate

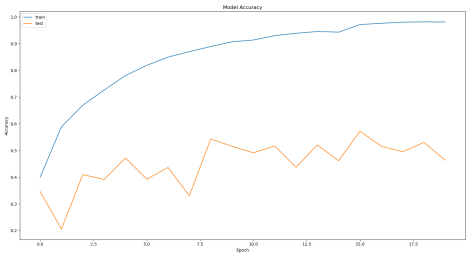


(a) Accuracy (Max at 60%)

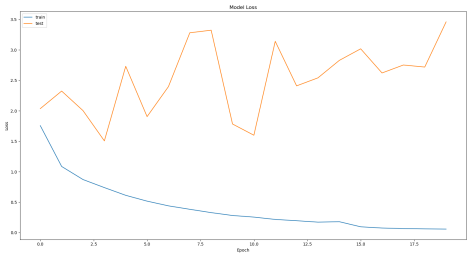


(b) Loss (Min at 1.4)

Figure 5.5: VGG with ELU with 32, 64, 128 and 256 layer units units, with 0.2 and 0.5 dropout rates, two dense layers of 64 and 32 units and 0.001 learning rate

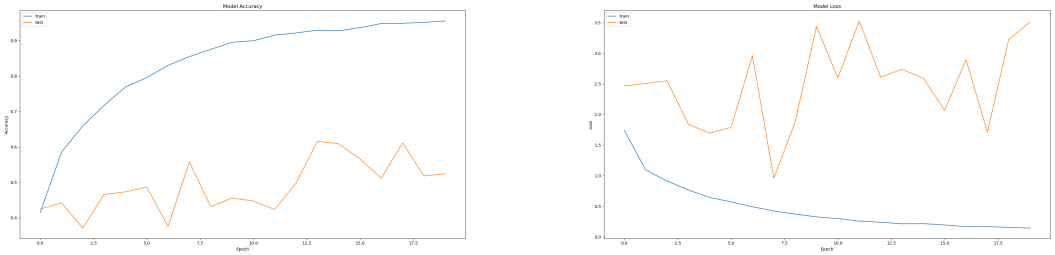


(a) Accuracy (Max at 60%)



(b) Loss (Min at 1.6)

Figure 5.6: VGG with ELU with 32, 64, 128 and 256 layer units units, with 0.2 and 0.5 dropout rates, two dense layers of 64 units each and 0.001 learning rate



(a) Accuracy (Max at 63%)

(b) Loss (Min at 1.1)

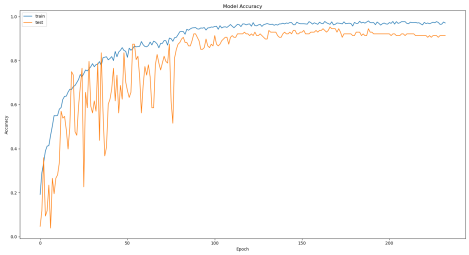
Figure 5.7: VGG with ELU with 32, 64, 128 and 256 layer units units, with 0.3 and 0.5 dropout rates, two dense layers of 64 units each and 0.001 learning rate

5.3 Experiments on the CK+ Subsets

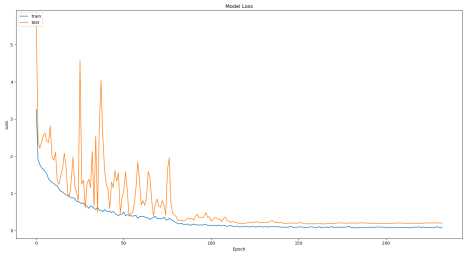
Since the CK+ database was overfittng on the VGG architecture and the results were not satisfactory, we decided to experiment with other architectures. This was when we decided to explore the origins of the Convolutional Neural Network and modified the LeNet-5 architecture. We first tried with an input image of 48x48 pixels, but since the loss was high we decided to take the input of 128x128 pixels. But when we used the input as 128x128, we found out that we needed to use pooling layers after every convolution layer, this was to prevent overloading the RAM and GPU, as without the pooling layers the system displayed "Out Of Memory" (OOM) errors.

The following graphs in Figure: 5.8 and Figure: 5.9, are the results of training the model (shown in Figure: 4.2), using 7 of the 8 subsets that were split (shown in Figure: 4.3), while the remaining 1 subset was used for testing the model.

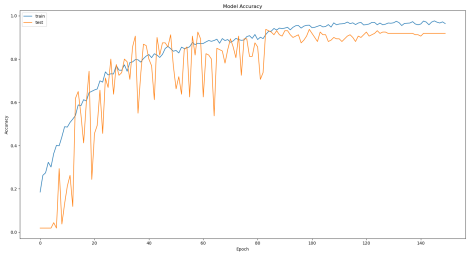
As it is visible from the graphs, the model performs very well, in terms of high accuracy and low loss rates. However, the model that was tested on subset 8 performed poorly when compared to other subsets. This was because in the data split (Figure: 4.4), subset 8 had majority of the "contempt" data and a bit more of "sadness" than others. This was one of the major reasons for the model to under perform.



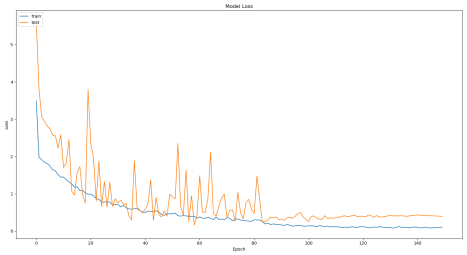
(a) Accuracy tested on subset 1



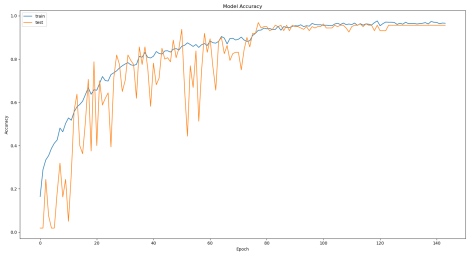
(b) Loss tested on subset 1



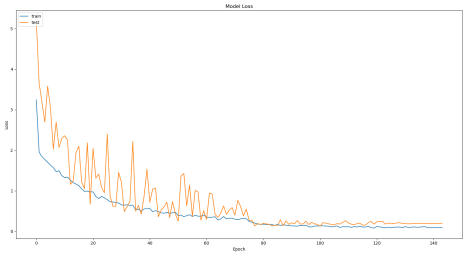
(c) Accuracy tested on subset 2



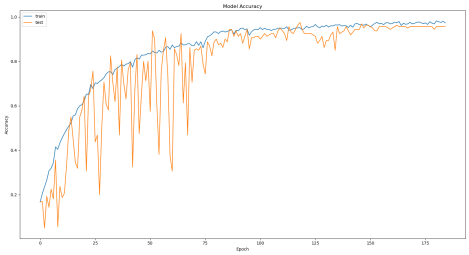
(d) Loss tested on subset 2



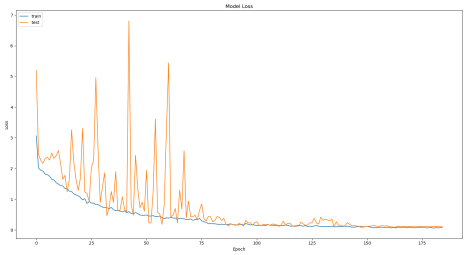
(e) Accuracy tested on subset 3



(f) Loss tested on subset 3

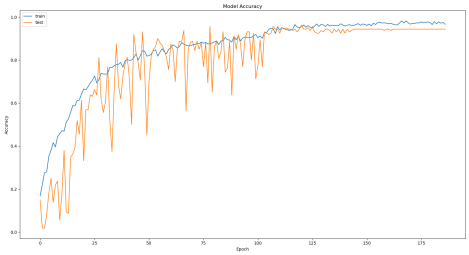


(g) Accuracy tested on subset 4

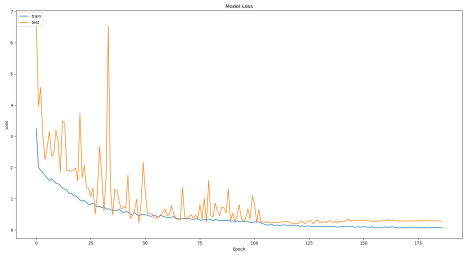


(h) Loss tested on subset 4

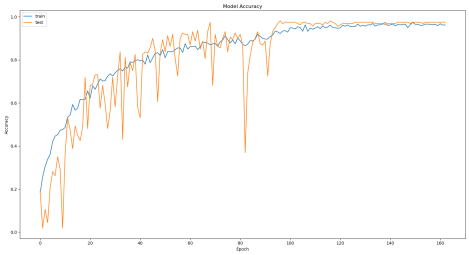
Figure 5.8: Accuracy and Loss on the CK+ data subsets (1-4)



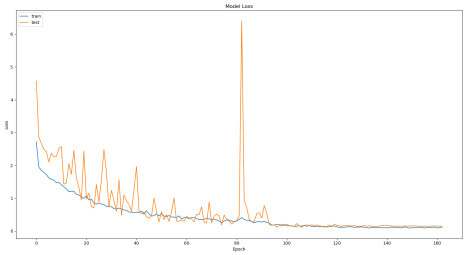
(a) Accuracy tested on subset 5



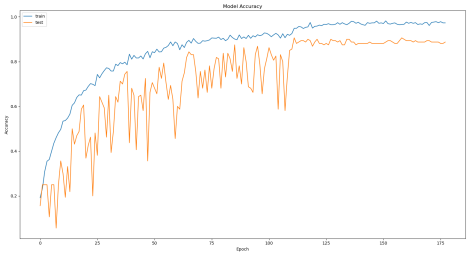
(b) Loss tested on subset 5



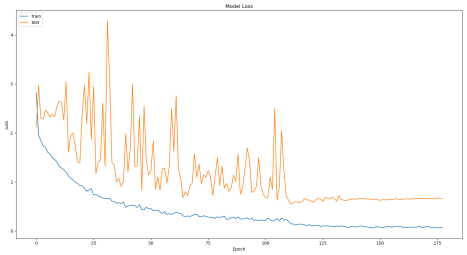
(c) Accuracy tested on subset 6



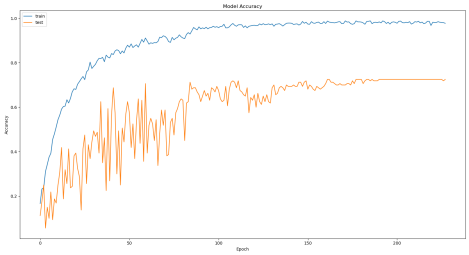
(d) Loss tested on subset 6



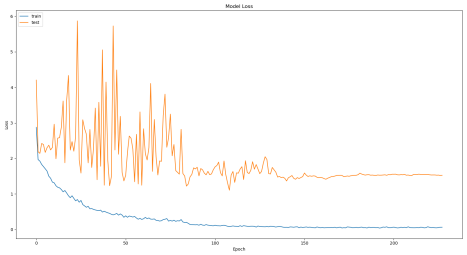
(e) Accuracy tested on subset 7



(f) Loss tested on subset 7



(g) Accuracy tested on subset 8



(h) Loss tested on subset 8

Figure 5.9: Accuracy and Loss on the CK+ data subsets (5-8)

5.4 Final Results and Observations

The table in figure: 5.10 shows the complete analysis of results. The model was run 5 times for each subset to ensure that it was not stuck in a local minima (Tayyar Madabushi, 2019), and the

best result of that subset was chosen, which is highlighted in green. The results in the Subset 1 column, are the ones when the model was tested on subset 1, while subsets 2 to subset 8 were used for training. Similarly all other subsets were run 5 times and the best result was chosen.

Accuracy								
	Subset 1	**Subset 2**	**Subset 3**	**Subset 4**	**Subset 5**	**Subset 6**	**Subset 7**	**Subset 8**
Run 1	0.95312	0.9375	0.96875	0.925	0.95625	0.96875	0.90625	0.725
Run 2	0.9375	0.91875	0.96875	0.95625	0.9625	0.98125	0.875	0.775
Run 3	0.89844	0.90625	0.95625	0.95	0.95625	0.975	0.8875	0.72
Run 4	0.92188	0.9375	0.95	0.975	0.95625	0.9625	0.9	0.73125
Run 5	0.90625	0.925	0.96875	0.95	0.95625	0.975	0.875	0.7625

(a) Accuracy

Loss								
	Subset 1	**Subset 2**	**Subset 3**	**Subset 4**	**Subset 5**	**Subset 6**	**Subset 7**	**Subset 8**
Run 1	0.185	0.2486	0.1322	0.1838	0.2158	0.1026	0.5462	1.5218
Run 2	0.2539	0.2537	0.197	0.1292	0.198	0.1189	0.7414	1.0133
Run 3	0.3885	0.5845	0.1656	0.2534	0.1431	0.0942	0.7653	1.634
Run 4	0.3382	0.314	0.136	0.1193	0.2784	0.1455	0.5919	1.5332
Run 5	0.3519	0.4375	0.1149	0.1274	0.2032	0.1479	0.7488	1.0532

(b) Loss

Figure 5.10: Accuracy and Loss results when tested on each subset. Best results highlighted in green

Figure: 5.11 shows the comparison of the state-of-the-art results to the results we achieved. To maintain fairness of this result, we divided the database in 8 subsets in a similar manner, as described in their research paper, Lopes, de Aguiar, and Oliveira-Santos, 2015. In addition, we used the same format to execute our program. The average accuracy rate of their model was 91.46±1%, while we got an average accuracy rate of 93.24±1%. Their best accuracy rate was at 93.74% and ours topped at 98.12%. This shows that, our model performed better when using spatial normalization and generating synthetic samples.

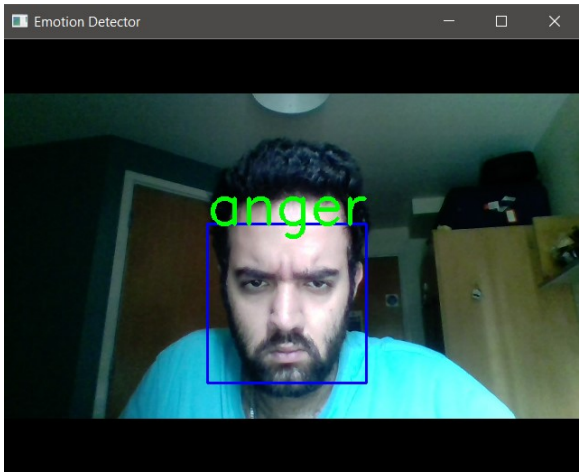
Preprocessing	Average	Best
None	56.93±5%	61.70%
Spatial Normalization	86.44±3%	90.00%
Intensity Normalization	82.39±1%	86.10%
Both Normalizations	84.68±2%	87.81%
Spatial Normalization and Synthetic Samples	87.54±1%	90.83%
Both Normalizations and Synthetic Samples	91.46±1%	93.74%
Spatial Normalization and Synthetic Samples (our model)	93.24±1%	98.12%

Figure 5.11: Comparison of our results with state-of-the-art results by Lopes, de Aguiar, and Oliveira-Santos, 2015

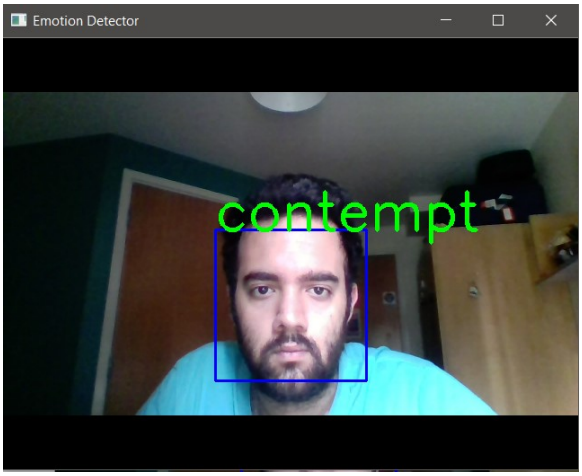
The screenshots in Figure: 5.12 were tests carried out in real time, on the best model, i.e., the model tested on subset 6, which got an accuracy of 98.125%. The tests were carried out in different lighting conditions, to notice any significant differences in the result.

The major difference in result was when the face of the subject was over exposed to light. The model was not able to accurately define the emotion that was shown on the subject's face. Since we used Max Pooling, as explained in section: 2.4.3, it is possible that the features were not very visible due to the over exposure that the image was undergoing.

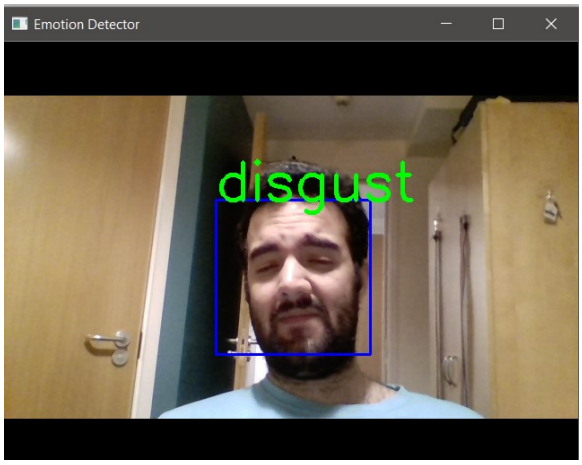
One possible solution to this is to normalise the image in real time, while making predictions, this would possibly cause the detector to lag and the time taken to predict the emotion would be very large. The other possible solution is that during the pre-processing stage or the data augmentation stage, we could deliberately increase the exposure of the image, so that the model could learn about the intensity changes accordingly.



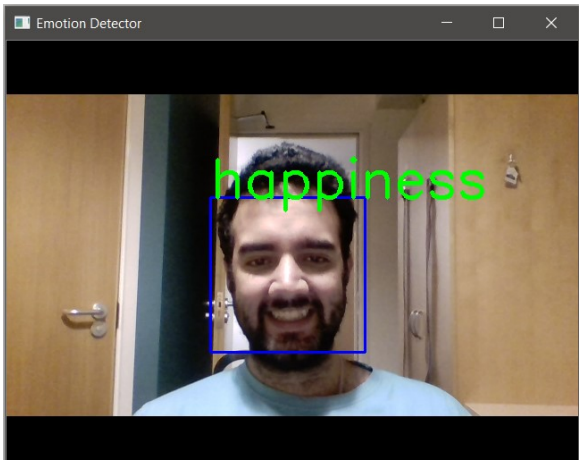
(a) Detected Emotion: Anger



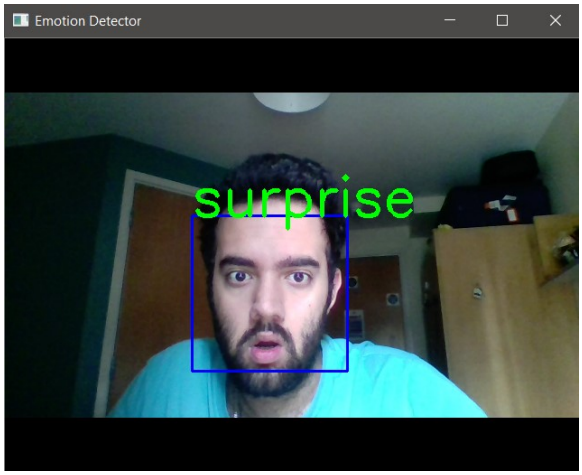
(b) Detected Emotion: Contempt



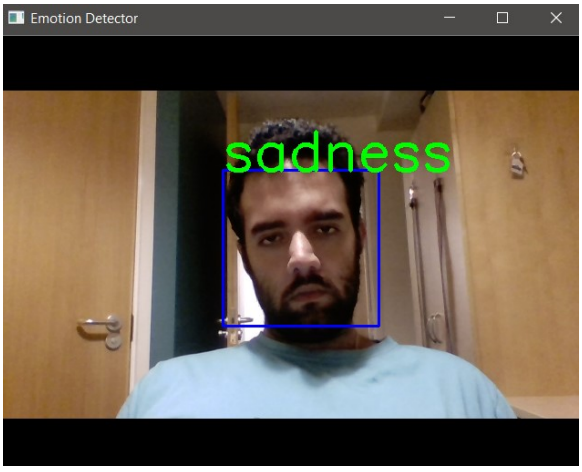
(c) Detected Emotion: Disgust



(d) Detected Emotion: Happiness



(e) Detected Emotion: Surprise



(f) Detected Emotion: Sadness

Figure 5.12: Screenshots of real time results of the model that was tested on subset 6.

Chapter 6: Conclusion and Future Work

In this report, we have proposed a modified architecture for facial expression recognition that uses a convolutional neural network. We compared our model with the state-of-the art results while keeping the database and execution format the same for fairness. After a detailed analysis we found our model achieving a higher accuracy rate of 98.125%.

While training our CNN model, we ensured the database was split into 8 subsets, without any overlapping subjects and we ran our models for five runs on each subset and saved the best out of the five runs.

We tested our best performing model, by using it in a real time detector. The detector worked as expected, but there were few important observations that were made, the detector didn't work well when there was a major change in the lighting condition.

There are several ways to extend this project, one of them being to work on improving the model's sensitivity to light and to stabilise the detector without slowing down the real time detection. Alternatively, setting up a binary classifier to classify each emotion one by one to analyse their individual predictive performance.

Another way to extend this project would be to detect fake emotions from the real emotions. However, the fake emotions would not work just by adding a fake emotion database, this is because every person has a different expression to relay their fake or real emotions. For this the model must learn their real expressions in real time, and then compare their other expressions based on the situation, using a predictive model.

References

Chollet, François et al. (2015). *Keras*. https://keras.io.

Cireşan, Dan, Ueli Meier, Jonathan Masci, et al. (2011). *Flexible, High Performance ConvolutionalNeural Networks for Image Classification*. URL: http://people.idsia.ch/~juergen/ijcai2011.pdf.

Cireşan, Dan, Ueli Meier, and Juergen Schmidhuber (2012). *Multi-column Deep Neural Networks for Image Classification*. arXiv: 1202.2745 [cs.CV]. URL: https://arxiv.org/pdf/1202.2745.pdf.

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2015). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. arXiv: 1511.07289 [cs.LG].

Garnier, J.G. and A. Quetelet (1838). *Correspondance mathématique et physique*. v. 10. Impr. d'H. Vandekerckhove. URL: https://books.google.co.uk/books?id=8GsEAAAAYAAJ.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *6.2.2.3 Softmax Units for Multinoulli Output Distributions*. URL: https://www.deeplearningbook.org/contents/mlp.html.

Hodgkin, A. L. and A. F. Huxley (1952). *A quantitative description of membrane current and its application to conduction and excitation in nerve*. DOI: 10.1113/jphysiol.1952.sp004764. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/.

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. URL: https://arxiv.org/abs/1502.03167v3.

Kingma, Diederik P. and Jimmy Lei Ba (2015). "Adam: A Method for Stochastic Optimization". In: arXiv: 1412.6980v9 [cs.LG]. URL: https://arxiv.org/pdf/1412.6980v9.pdf.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). *ImageNet Classification with Deep ConvolutionalNeural Networks*. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Liu, Ping et al. (2014). "Facial Expression Recognition via a Boosted Deep Belief Network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, Y. et al. (2015). "A Main Directional Mean Optical Flow Feature for Spontaneous Micro-Expression Recognition". In: *IEEE Transactions on Affective Computing*. DOI: 10.1109/TAFFC.2015.2485205.

Lopes, A. T., E. de Aguiar, and T. Oliveira-Santos (2015). "A Facial Expression Recognition System Using Convolutional Networks". In: *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pp. 273–280.

Martıén Abadi et al. (2015a). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

— (2015b). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/tutorials/distribute/keras.

— (2015c). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/tutorials/images/data_augmentation.

— (2015d). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/guide/keras/sequential_model.

— (2015e). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D.

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, pp. 807–814. ISBN: 9781605589077.

Orr, Genevieve B., Nici Schraudolph, and Fred Cummins (1999). *Momentum and Learning Rate Adaptation*. URL: https://www.willamette.edu/~gorr/classes/cs449/momrate.html.

Qian, Ning (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1, pp. 145–151. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. URL: http://www.sciencedirect.com/science/article/pii/S0893608098001166.

Ruder, Sebastian (2017). *An overview of gradient descent optimization algorithms*. URL: https://arxiv.org/pdf/1609.04747.pdf.

Scherer, Dominik, Andreas Müller, and Sven Behnke (2010). "Evaluation of Pooling Operations in Convolutional Architectures for ObjectRecognition". In: *20th International Conference on Artificial Neural Networks (ICANN)*. URL: http://ais.uni-bonn.de/papers/icann2010_maxpool.pdf.

Simonyan, Karen and Andrew Zisserman (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: 1409.1556 [cs.CV].

superdatascience.com (2020). URL: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-2-max-pooling/.

Tayyar Madabushi, Harish (2019). "Lectures: MSc Introduction to Artificial Intelligence (06-27112)". The University of Birmingham.

Wang, S., H. Chen, et al. (2013). "Face Recognition and Micro-expression Recognition Based on Discriminant Tensor Subspace Analysis Plus Extreme Learning Machine". In: *Neural Process Lett*. DOI: 10.1007/s11063-013-9288-7.

Wang, S., W. Yan, X. Li, G. Zhao, and X. Fu (2014). "Micro-expression Recognition Using Dynamic Textures on Tensor Independent Color Space". In: *22nd International Conference on Pattern Recognition*, pp. 4678–4683.

Wang, S., W. Yan, X. Li, G. Zhao, C. Zhou, et al. (2015). "Micro-expression Recognition Using Color Spaces". In: *IEEE Transactions on Image Processing*. DOI: 10.1109/TIP.2015.2496314.

Wu, Q., X. Shen, and X. Fu (2011). "The Machine Knows What You Are Hiding: An Automatic Micro-expression Recognition System". In: *D'Mello S., Graesser A., Schuller B., Martin JC. (eds) Affective Computing and Intelligent Interaction* 6975, pp. 152–162. DOI: https://doi.org/10.1007.

Yan, W. et al. (2014). *For micro-expression recognition: Database and suggestions*. URL: www.elsevier.com/locate/neucom.

Zhang, Aston et al. (2020). "Dive into Deep Learning". In: pp. 135–139.

Appendix A

GitLab repository:

https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2019/dxs956

The main program files to look at:

- PythonPrograms/EmotionClassifier.py
- PythonPrograms/WebcamTest.py

Outputs to look at:

- ckPlusSubsetOutpus

Note: if there are two files Acc_testedOn4.png and Acc_testedOn4-4.png, then Acc_testedOn4-4.png is the latest edition to look at since the "-4" represents that its the 4^{th} run.

Subset division:

- Directory: ckplus subsets contains all the image file names split up in set of csv files

PPT presentation:

- UoB-HumanEmotionDetectionUsingCNN.pptx

Analysis and Results:

- filenames.xlsm