**Bilkent University, Department of Computer Science**
**CS 464: Introduction to Machine Learning**
**Section 02**


# Emotion Recognition in Still Images

*GROUP 21*

**Team Members:**

- Hanzallah Azim Burney
- Sayed Abdullah Qutb
- Balaj Saleem
- Mehmet Alper Genç
- Burak Mutlu

**Course Professor:** Aysegul Dundar

**TA:** Furkan Ozden

**GROUP 21** 1

**1. Introduction** 3

**2. Problem Description** 3

**3. Data and Methods** 3
- **3.1 Dataset Used** 3
- **3.2 Deep Learning Models** 3
  - **3.2.1 ResNet50V2 and ResNet101V2** 3
  - **3.2.2 InceptionV3 and InceptionResNetV2** 4
  - **3.2.3 VGG16 and VGG19** 5
  - **3.2.4 DenseNet121 and DenseNet169** 7
  - **3.2.5 NasNetLarge and NasNetMobile** 8

**4. Results** 9
- **4.1 Training Results** 10
  - **4.1.1 Epochs** 10
  - **4.1.2 Learning Rate** 11
  - **4.1.3 Optimizers** 12
  - **4.1.4 Activation Functions** 13
  - **4.1.5 Data Normalization** 14
- **4.2 Test Evaluation Results** 15
  - **4.2.1 Epochs** 15
  - **4.2.2 Learning Rate** 16
  - **4.2.3 Optimizers** 16
  - **4.2.4 Activation Functions** 18
  - **4.2.5 Data Normalization** 19

**5. Discussion** 20
- **5.1 Effect of Epochs and Layers** 20
- **5.2 Effect of Learning Rate** 20
- **5.3 Effect of Optimizers** 21
- **5.4 Effect of Activation Functions** 21
- **5.5 Effect of Data Normalization** 22
- **5.6 Layer Visualizations** 22
  - **5.6.1 ResNet50V2** 22
  - **5.6.2 InceptionV3** 23
  - **5.6.3 DenseNet121** 24
  - **5.6.4 VGG16** 25
  - **5.6.5 NasNetMobile** 26
- **5.7 Issues Faced** 26

**6. Conclusion** 27

**7. References** 28

**8. Appendix** 30
- **8.1. Contributions** 30

# 1. Introduction

The project involves recognizing various predetermined emotions expressed by subjects in still images. In order to achieve this, we will be using various deep neural network based learning models and comparing their performances on the same dataset using several multi-class classification and accuracy metrics. The dataset being used is called the “Facial Recognition Dataset” supplied by Gotam Dahiya on Kaggle. To evaluate the various models under different conditions we will primarily use Top-1 accuracy metric. Under this metric, we match the true class with the most probable classes predicted by our model.

# 2. Problem Description

The project aims to predict a range of different emotions such as happiness, sadness, anger, fear, disgust, and surprise that are expressed by subjects in still images. The image data will be obtained from the “Facial Recognition Dataset” described further in section 3.1. The project will explore the performance of different machine learning and deep learning models on the dataset in order to determine which sort of a model provides the best classification of emotions in still images.

# 3. Data and Methods

## 3.1 Dataset Used

The project will use the “Facial Recognition Dataset” supplied by Gotam Dahiya on Kaggle. The dataset consists of a training set and a testing set. The training set contains 28,079 image samples which are classified into emotions including happiness, sadness, anger, fear, disgust, neutral, and surprise. The testing set includes 7,178 sample images. Each image is a 48x48 grayscale image that has already been centered and occupies the same amount of space as other images in the dataset.

The dataset can be found at https://www.kaggle.com/apollo2506/facial-recognition-dataset.

## 3.2 Deep Learning Models

### 3.2.1 ResNet50V2 and ResNet101V2

Residual Networks or ResNets are a kind of a neural network whose construct is based on pyramidal cells in the cerebral cortex [4]. These neural networks introduce the concept of an identity shortcut connection and include residual building blocks which allow forward and backward signals to be directly propagated from one block to any other block after the activation function is added by using identity mappings as the skip connections [2]. These residual neural networks solve the vanishing gradient problem that conventional CNN architectures face as the number of layers is increased [3]. This can lead to the performance of deep CNN architectures degrading rapidly [3]. Residual networks work around this problem by reusing the activations generated by the previous layers until the adjacent layer has managed to learn its weight [4].

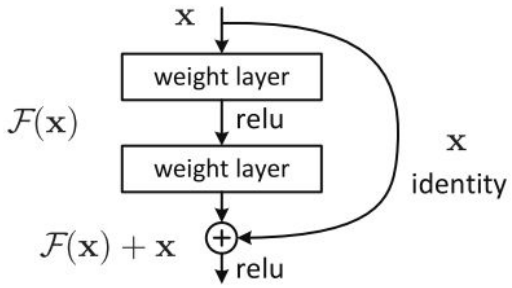


**Fig 1. A Residual Building Block [3]**

The stacked residual building blocks can be represented mathematically by,

$$y_l = h(x_l) + F(x_l, W_l) \quad [2]$$
$$x_{l+1} = f(y_l) \quad [2]$$

where $x_l$ and $x_{l+1}$ are the input and output of the l-th unit, and F is the residual function [2]. In deep residual networks, $h(x_l) = x_l$ is an identity mapping and f is the ReLu function [2].

In terms of emotion recognition detection, we will be using the ResNet50V2 (50 layer network) and the ResNet101V2 (101 layer network) from the Keras library. The Keras ResNet50V2 model trained on the ImageNet dataset gives a Top-1 accuracy of 0.749 and a Top-5 accuracy of 0.921 [1]. The ResNet101V2 model by Keras trained on the ImageNet dataset gives a Top-1 accuracy of 0.764 and a Top-5 accuracy of 0.928 [1].

### 3.2.2 InceptionV3 and InceptionResNetV2

InceptionV3 is the third edition of the Inception architecture of GoogLeNet, based on the paper "Rethinking the Inception Architecture for Computer Vision" by Szegedy et al., and reportedly was able to achieve "greater than 78.1% accuracy on the ImageNet dataset" [5]. As shown in the diagram below, the Inception architecture consists of many building blocks that perform different functions, such as filter concatenation ("Concat" in the diagram) and Softmax for loss calculation [5].

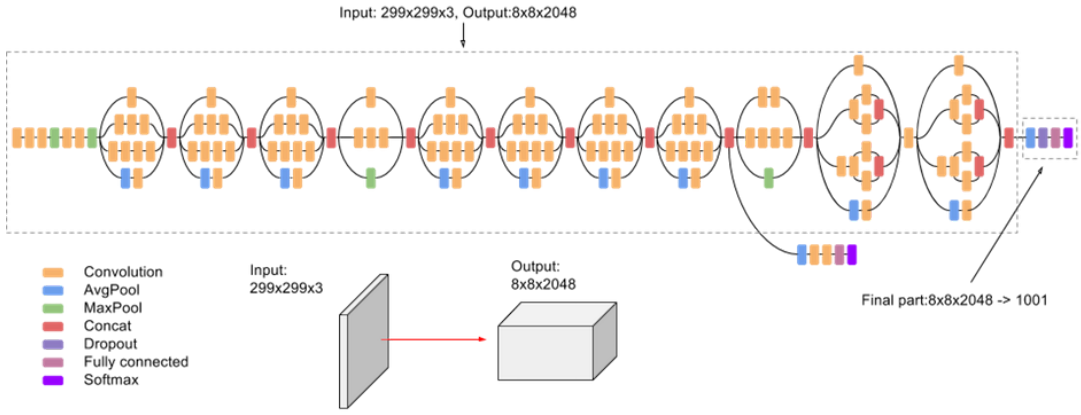


**Fig. 2: High Level Diagram of the Inception architecture [5]**

Models belonging to the Inception family of architectures factorize their convolutions, which according to Szegedy et al., is done so to "end up with more disentangled parameters and therefore with faster training" [6]. This is done by reducing the size of the convolution grids, as shown in Fig. 3 below [6]. They also argue that "the computational cost saving increases dramatically as n grows" [6].

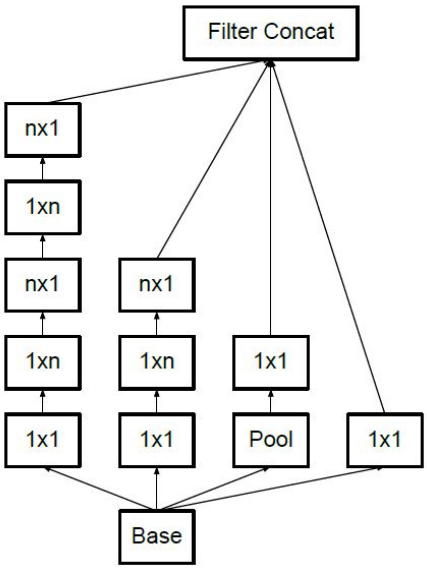


**Fig. 3: "Inception modules after the factorization of the nxn convolutions" [6]**

We have decided to use this model as part of our project as it has a top-1 accuracy of 0.779 and a top-5 accuracy of 0.937 according to the Keras Applications webpage [1]. Furthermore, according to Szegedy et al., the Inception architecture can "perform well even under strict constraints on memory and computational budget" [6].

InceptionResNetV2 is also a member of the Inception family of architectures, but rather than using filter concatenation like the "pure" Inception architectures, it utilizes residual connections, which are explained in section 1.1 [7]. According to the Keras Applications webpage, this CNN architecture has a top-1 accuracy of 0.803 and a top-5 accuracy of 0.953 [1].

### 3.2.3 VGG16 and VGG19

VGG is a post cursor of Alex-net and similarly a convolutional neural network (CNN) considered to be state of the art for Image Classification[13]. It was first introduced by Simonyan and Zisserman in 2014 in a paper titled: Very Deep Learning Convolutional Neural Networks for Large-Scale Image Recognition [14]. Modern implementations of network include 16 or 19 network layers using an architecture of very small 3x3 convolution filters (smallest possible to capture left/right , top/bottom). Detailed configurations of the ConvNet on a 224x224 RGB image dataset used by the authors of the paper is given below [14]:

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

| 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 | | | | | |

Table 2: **Number of parameters** (in millions).

| Network | A,A-LRN | B | C | D | E |
|---|---|---|---|---|---|
| Number of parameters | 133 | 133 | 134 | 138 | 144 |

With regards to emotion recognition i.e. our project the network will be loaded from Keras [1] using pre-trained weights from Image-Net trained more than a million images. The dimensions of the images. Since the images are 48x48, while the model is trained on 224x224 some modifications must be made to either the input layer or the data must be scaled up.

The following image elaborates the architecture of VGG-16 in further detail and delineates the layers involved:

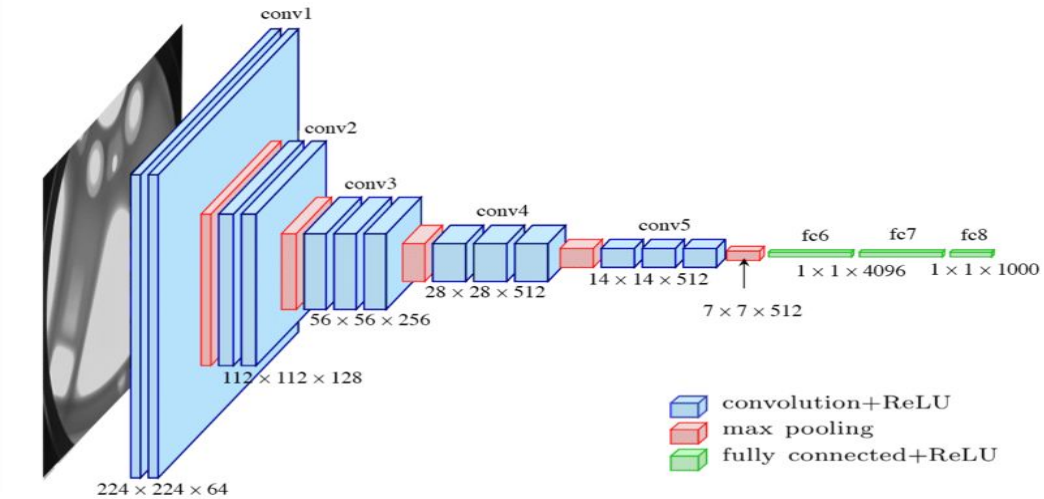


**Fig. 4: "Architecture of VGG-16" [15]**

### 3.2.4 DenseNet121 and DenseNet169

Dense Convolutional Network (DenseNet), as its name suggests, represents a dense connectivity between the layers of a network. Using feed forward approach, each layer on a DenseNet passes its own feature maps to all subsequent layers, that is, a layer obtains all feature maps from all preceding layers as input additionally [8].

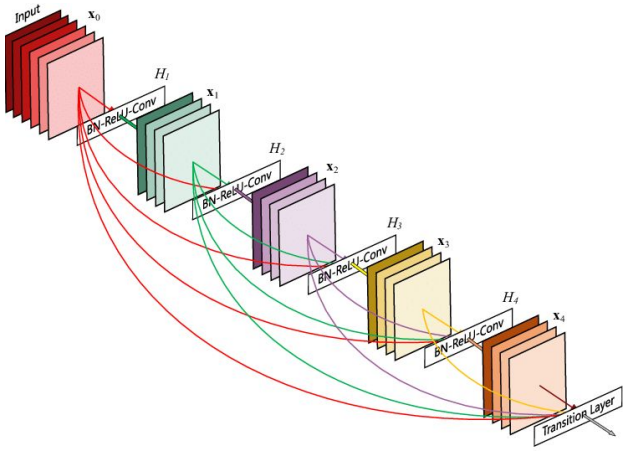


**Fig. 5: A 5-layer dense block in which each layer takes all preceding feature maps as input [8].**

As seen Fig. 5, such dense connectivity structure provides maximum information flow between layers. By transferring all information among the network, it eliminates the need for re-learning redundant feature maps. As an intuitive effect, such a dense connectivity structure requires fewer parameters than traditional CNNs [8]. In Keras Applications webpage, DenseNet is the one of models using the least number of parameters [1]. It overcomes the vanishing gradient problem in increasingly deep CNNs. Actually, there are many other approaches addressing this problem like ResNets by using identity connections to bypass input from one layer to the next layer. All these approaches share a key idea which is creating short paths to later layers. Similarly in DenseNet, it provides all these short paths to ensure maximum information flow. Traditional CNN with *L* layers includes overall *L* connections between layers. In DenseNet, this connection count is $L(L+1) / 2$ since a layer at level $l^{th}$ includes *l* connections [8].

Providing high accuracy on various vision tasks requires multi-level features, that is, network structure with cross-layer connections. Many other approaches provide this functionality through bypassing paths which eases to train very deep networks like ResNet. However, in its characteristic, DenseNet fits the data into the network model with wider layers instead of deeper layers. In the end, it has much less layers compared with others but the layers are much wider. That is, the final classifier has access to all feature maps in the network and it can base its decision regardingly. Another advantage of DenseNet in Huang et al. is that "dense connections create many short paths in the network, which have a strong regularizing effect and reduce overfitting on smaller training sets [8]."

In DenseNet architecture, the network includes *L* layers, all of which applies non-linear transform $H_l(.)$, where *l* denotes the layer level. "Specifically, each $H_l(.)$ correspondence to the sequence: BN(Batch normalization) - ReLU - Conv(1x1) - BN - ReLU - Conv(3x3)". Each $H_l(.)$ function

produces $k$ feature maps and $l^{th}$ layer contains $k(l - 1) + k_0$ input feature maps. That is, hyper-parameter $k$ represents the growth rate of a network. Illustrating the general structure where $x_l$ denotes the output for $l^{th}$ layer:

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$$ [8].

So, we will be using DenseNet121 (L = 121) and DenseNet169 (L = 169) models from the Keras library for emotion recognition tasks. According to Keras Applications webpage, these models scored (Top-1, Top-5) accuracy as (0.750, 0.923) for DenseNet121 and (0.762, 0.932) for DenseNet169 on ImageNet dataset, which are very promising results for image recognition tasks [1]. That's why we decided to involve these models in our methodologies by imposing various preprocessed data and experimenting their performance.

### 3.2.5 NasNetLarge and NasNetMobile

Short for Neural Architecture Search Network, is a convolutional neural network, part of pretrained deep neural networks, used for image classification. It is usually used to train on a large number of image datasets, approx. more than a million and can classify images into more than 1000 object categories. It initially tries to search for the best convolutional layer or cell in a smaller dataset, such as CIFAR10, and then apply this layer or cell on a bigger dataset, such as ImageNet. [9]

As it can be seen in Figure 5, the given blocks are not defined by authors, but rather searched by reinforcement learning search methods. The number of initial convolutional filters and the value N can be used as parameters for scaling the architecture. A Normal Cell, Figure 5, is a convolutional cell which returns the feature map with the same dimension, but a Reduction Cell reduces the width and height of the feature map by a factor of 2. These blocks of convolutional cells are generated by reinforcement search methods, which is called Controller RNN and it recursively finds and predicts the rest of the structure. [10]

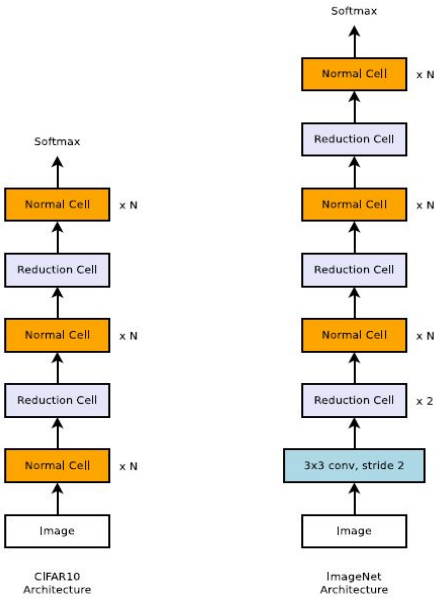


**Fig. 6: Nasnet Scalable Architecture for ImageNet and CIFAR10 datasets**

NasNet Mobile is a smaller version of NasNet and is designed to be smaller in complexity and model size. NasNet mobile is preferable for devices of lower computational power, such as mobile devices. However NasNet mobile performance is better compared to any other similar-size models in mobile platforms. It contains 12 cells with 5.3 million parameters. [11]

## 4. Results

Till date, we have managed to read and analyse the Facial Recognition Dataset. The analysis provided insights into the number of samples for each emotion in the training and test sets. This is summarized below,

| | Train | Test |
|---|---|---|
| **Angry** | 3995 | 958 |
| **Happy** | 7215 | 1774 |
| **Neutral** | 4965 | 1233 |
| **Fear** | 4097 | 1024 |
| **Sad** | 4830 | 1247 |
| **Surprise** | 3171 | 831 |

**Table 1. Number of samples in the training and test sets**

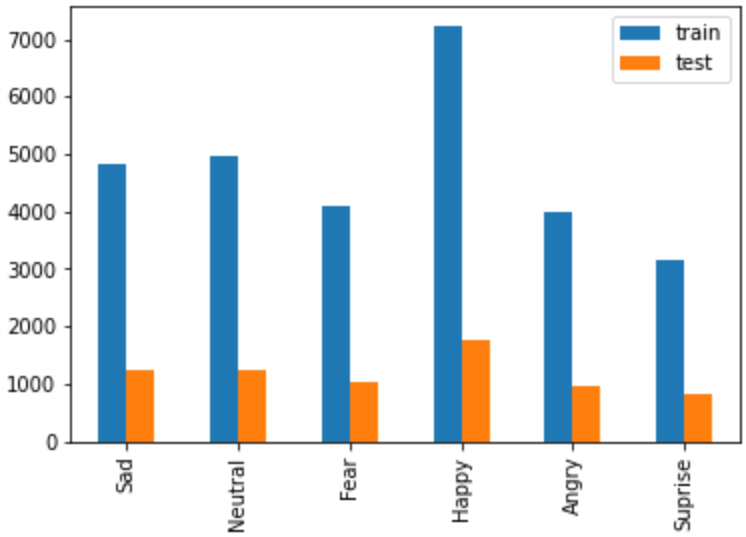


**Fig 7. Visualization of the training and test sets**

There is very little class imbalance in the training and testing datasets and the proportion of samples in each class is proportional in the training and test datasets. Therefore, we did not have to take class imbalance correction measures. Using the keras ImageDataGenerator class

we performed ‘on the fly’ image preprocessing and conversion to a pixel feature set. The preprocessing techniques included mean-centering, normalization, and standardization of images. We also performed ZCA whitening which is similar to dimensionality reduction using PCA but has the advantage of retaining the same spatial arrangement of pixels thereby retaining the ‘look’ of the original image which PCA whitening does not manage to do. It removes redundant pixels through a linear algebra based operation. This type of reduction is therefore advantageous for neural networks such as convolutional neural networks.

We performed training and evaluation on some of our models. In this report we managed to study the effect of changing the epochs, the learning rate, the optimizer, the activation function, and data normalization on the accuracy of the predictions. The results, and their analysis is provided in the following sections.

## 4.1 Training Results

### 4.1.1 Epochs

| Model | Epochs | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | 10 | 1.039 | 0.602 |
| ResNet101V2 | 10 | 1.052 | 0.590 |
| InceptionV3 | 10 | 1.205 | 0.524 |
| InceptionResNetV2 | 10 | 1.183 | 0.535 |
| DenseNet121 | 10 | 1.033 | 0.602 |
| DenseNet169 | 10 | 1.002 | 0.617 |
| VGG16 | 10 | 1.758 | 0.255 |
| VGG19 | 10 | 1.755 | 0.259 |
| NasNetMobile | 10 | $6x10^5$ | 0.5374 |
| ResNet50V2 | 15 | 0.797 | 0.702 |
| ResNet101V2 | 15 | 0.895 | 0.658 |
| InceptionV3 | 15 | 0.920 | 0.648 |
| InceptionResNetV2 | 15 | 0.964 | 0.632 |
| DenseNet121 | 15 | 0.773 | 0.703 |
| DenseNet169 | 15 | 0.824 | 0.689 |

| | | | |
|---|---|---|---|
| VGG16 | 15 | 1.761 | 0.251 |
| VGG19 | 15 | 1.758 | 0.257 |
| NasNetMobile | 15 | 550 | 0.5641 |

**Table 2. Training Epoch Results**

### 4.1.2 Learning Rate

| Model | Learning Rate | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | 0.01 | 1.039 | 0.602 |
| ResNet101V2 | 0.01 | 1.052 | 0.590 |
| InceptionV3 | 0.01 | 1.205 | 0.524 |
| InceptionResNetV2 | 0.01 | 1.183 | 0.535 |
| DenseNet121 | 0.01 | 1.033 | 0.602 |
| DenseNet169 | 0.01 | 1.002 | 0.617 |
| VGG16 | 0.01 | 1.758 | 0.255 |
| VGG19 | 0.01 | 1.755 | 0.259 |
| NasNetMobile | 0.01 | $6x10^5$ | 0.5374 |
| ResNet50V2 | 1 | 1.057 | 0.590 |
| ResNet101V2 | 1 | 1.906 | 0.201 |
| InceptionV3 | 1 | 1.906 | 0.200 |
| InceptionResNetV2 | 1 | 1.907 | 0.202 |
| DenseNet121 | 1 | 1.909 | 0.204 |
| DenseNet169 | 1 | 1.910 | 0.198 |
| VGG16 | 1 | 1.960 | 0.251 |
| VGG19 | 1 | 1.761 | 0.253 |
| NasNetMobile | 1 | 6.943 | 0.1943 |

**Table 3. Training Learning Rate Results**

### 4.1.3 Optimizers

| Model | Optimizer | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | RMSprop | 1.039 | 0.602 |
| ResNet101V2 | RMSprop | 1.052 | 0.590 |
| InceptionV3 | RMSProp | 1.205 | 0.524 |
| InceptionResNetV2 | RMSProp | 1.183 | 0.535 |
| DenseNet121 | RMSProp | 1.033 | 0.602 |
| DenseNet169 | RMSProp | 1.002 | 0.617 |
| VGG16 | RMSprop | 1.758 | 0.255 |
| VGG19 | RMSprop | 1.758 | 0.255 |
| NasNetMobile | RMSprop | $6x10^5$ | 0.5374 |
| ResNet50V2 | SGD | 0.850 | 0.685 |
| ResNet101V2 | SGD | 0.986 | 0.631 |
| InceptionV3 | SGD | 1.124 | 0.588 |
| InceptionResNetV2 | SGD | 0.163 | 0.952 |
| DenseNet121 | SGD | 0.950 | 0.648 |
| DenseNet169 | SGD | 1.021 | 0.620 |
| VGG16 | SGD | 1.759 | 0.251 |
| VGG19 | SGD | 1.755 | 0.259 |
| NasNetMobile | SGD | 1.173 | 0.552 |
| ResNet50V2 | Adam | 1.064 | 0.594 |
| ResNet101V2 | Adam | 1.258 | 0.505 |
| InceptionV3 | Adam | 1.304 | 0.459 |
| InceptionResNetV2 | Adam | 1.282 | 0.486 |
| DenseNet121 | Adam | 1.045 | 0.599 |
| DenseNet169 | Adam | 1.109 | 0.574 |

| | | | |
|---|---|---|---|
| VGG19 | Adam | 1.756 | 0.260 |
| VGG16 | Adam | 1.761 | 0.251 |
| NasNetMobile | Adam | 0.218 | 0.5202 |

**Table 4. Training Optimizer Results**

### 4.1.4 Activation Functions

| Model | Activation Function | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | Softmax | 1.051 | 0.596 |
| ResNet101V2 | Softmax | 1.115 | 0.566 |
| InceptionV3 | Softmax | 1.102 | 0.571 |
| InceptionResNetV2 | Softmax | 1.218 | 0.519 |
| DenseNet121 | Softmax | 0.952 | 0.634 |
| DenseNet169 | Softmax | 1.079 | 0.590 |
| VGG16 | Softmax | 1.758 | 0.255 |
| VGG19 | Softmax | 1.755 | 0.259 |
| ResNet50V2 | ReLu | 6.672 | 0.310 |
| ResNet101V2 | ReLu | 2.294 | 0.455 |
| InceptionV3 | ReLu | 6.193 | 0.442 |
| InceptionResNetV2 | ReLu | 7.608 | 0.293 |
| DenseNet121 | ReLu | 4.857 | 0.460 |
| DenseNet169 | ReLu | NaN | 0.460 |
| VGG16 | ReLu | 5.579 | 0.253 |
| VGG19 | ReLu | 10.451 | 0.250 |
| ResNet50V2 | Sigmoid | $1.192x10^{-7}$ | 0.141 |
| ResNet101V2 | Sigmoid | $1.192x10^{-7}$ | 0.135 |
| DenseNet121 | Sigmoid | 0.988 | 0.622 |

| | | | |
|---|---|---|---|
| DenseNet169 | Sigmoid | 1.070 | 0.589 |
| InceptionV3 | Sigmoid | 1.791 | 0.141 |
| InceptionResNetV2 | Sigmoid | 1.710 | 0.294 |
| VGG16 | Sigmoid | 5.708 | 0.256 |
| VGG19 | Sigmoid | 1.758 | 0.255 |

**Table 5. Training Activation Function Results**

### 4.1.5 Data Normalization

| Model | Normalization | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | True | 1.039 | 0.602 |
| ResNet101V2 | True | 1.052 | 0.590 |
| InceptionV3 | True | 1.205 | 0.524 |
| InceptionResNetV2 | True | 1.183 | 0.535 |
| DenseNet121 | True | 1.033 | 0.602 |
| DenseNet169 | True | 1.002 | 0.617 |
| VGG 16 | True | 1.754 | 0.258 |
| VGG 19 | True | 1.755 | 0.259 |
| NasNetMobile | True | 1.176 | 0.5374 |
| ResNet50V2 | False | 1.069 | 0.591 |
| ResNet101V2 | False | 1.112 | 0.569 |
| InceptionV3 | False | 1.247 | 0.498 |
| InceptionResNetV2 | False | 1.123 | 0.559 |
| DenseNet121 | False | 1.000 | 0.615 |
| DenseNet169 | False | 1.070 | 0.596 |
| VGG 16 | False | 1.758 | 0.255 |
| VGG 19 | False | 1.758 | 0.255 |

| | | | |
|---|---|---|---|
| NasNetMobile | False | 1.19 | 0.531 |

**Table 6. Training Normalization Results**

## 4.2 Test Evaluation Results

### 4.2.1 Epochs

| Model | Epochs | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | 10 | 1.465 | 0.477 |
| ResNet101V2 | 10 | 1.346 | 0.479 |
| InceptionV3 | 10 | 2.824 | 0.462 |
| InceptionResNetV2 | 10 | 1.482 | 0.403 |
| DenseNet121 | 10 | 1.262 | 0.502 |
| DenseNet169 | 10 | 1.299 | 0.519 |
| VGG16 | 10 | 1.757 | 0.251 |
| VGG19 | 10 | 1.760 | 0.251 |
| NasNetMobile | 10 | 1.176 | 0.251 |
| ResNet50V2 | 15 | 1.333 | 0.535 |
| ResNet101V2 | 15 | 3.394 | 0.509 |
| InceptionV3 | 15 | 1.283 | 0.538 |
| InceptionResNetV2 | 15 | 14.220 | 0.514 |
| DenseNet121 | 15 | 1.614 | 0.534 |
| DenseNet169 | 15 | 10.230 | 0.516 |
| VGG16 | 15 | 1.761 | 0.251 |
| VGG19 | 15 | 1.760 | 0.251 |
| NasNetMobile | 15 | 1.176 | 0.239 |

**Table 7. Test Epoch Results**

### 4.2.2 Learning Rate

| Model | Learning Rate | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | 0.01 | 1.639 | 0.411 |
| ResNet101V2 | 0.01 | 1.346 | 0.479 |
| InceptionV3 | 0.01 | 2.824 | 0.462 |
| InceptionResNetV2 | 0.01 | 1.482 | 0.403 |
| DenseNet121 | 0.01 | 1.262 | 0.502 |
| DenseNet169 | 0.01 | 1.299 | 0.519 |
| VGG16 | 0.01 | 1.758 | 0.255 |
| VGG19 | 0.01 | 1.760 | 0.251 |
| NasNetMobile | 0.01 | 1.176 | 0.251 |
| ResNet50V2 | 1 | 1.272 | 0.521 |
| ResNet101V2 | 1 | 34.510 | 0.177 |
| InceptionV3 | 1 | 1.918 | 0.176 |
| InceptionResNetV2 | 1 | 1.841 | 0.251 |
| DenseNet121 | 1 | 1.900 | 0.135 |
| DenseNet169 | 1 | 134.255 | 0.252 |
| VGG16 | 1 | 1.960 | 0.251 |
| VGG19 | 1 | 1.762 | 0.251 |
| NasNetMobile | 1 | 2.54 | 0.144 |

**Table 8. Test Learning Rate Results**

### 4.2.3 Optimizers

| Model | Optimizer | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | RMSprop | 1.639 | 0.411 |
| ResNet101V2 | RMSprop | 1.346 | 0.479 |

| | | | |
|---|---|---|---|
| InceptionV3 | RMSprop | 2.824 | 0.462 |
| InceptionResNetV2 | RMSprop | 1.482 | 0.403 |
| DenseNet121 | RMSprop | 1.262 | 0.502 |
| DenseNet169 | RMSprop | 1.299 | 0.519 |
| VGG16 | RMSprop | 1.758 | 0.251 |
| VGG19 | RMSprop | 1.758 | 0.251 |
| NasNetMobile | RMSprop | 1.176 | 0.251 |
| ResNet50V2 | SGD | 1.592 | 0.440 |
| ResNet101V2 | SGD | 1.481 | 0.466 |
| InceptionV3 | SGD | 1.629 | 0.472 |
| InceptionResNetV2 | SGD | 1.996 | 0.527 |
| DenseNet121 | SGD | 0.530 | 0.444 |
| DenseNet169 | SGD | 1.534 | 0.438 |
| VGG16 | SGD | 1.759 | 0.251 |
| VGG19 | SGD | 1.760 | 0.251 |
| NasNetMobile | SGD | 2.04 | 0.177 |
| ResNet50V2 | Adam | 1.534 | 0.624 |
| ResNet101V2 | Adam | 1.420 | 0.453 |
| InceptionV3 | Adam | 1.436 | 0.416 |
| InceptionResNetV2 | Adam | 2.149 | 0.288 |
| DenseNet121 | Adam | 0.530 | 0.444 |
| DenseNet169 | Adam | 1.380 | 0.458 |
| VGG16 | Adam | 1.761 | 0.251 |
| VGG19 | Adam | 1.756 | 0.251 |
| NasNetMobile | Adam | 2.510 | 0.251 |

**Table 9. Test Optimizer Results**

### 4.2.4 Activation Functions

| Model | Activation Function | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | Softmax | 1.307 | 0.504 |
| ResNet101V2 | Softmax | 1.694 | 0.380 |
| InceptionV3 | Softmax | 2.118 | 0.517 |
| InceptionResNetV2 | Softmax | 1.373 | 0.471 |
| DenseNet121 | Softmax | 1.215 | 0.531 |
| DenseNet169 | Softmax | 1.505 | 0.497 |
| VGG16 | Softmax | 1.758 | 0.255 |
| VGG19 | Softmax | 1.760 | 0.251 |
| ResNet50V2 | ReLu | 8.735 | 0.219 |
| ResNet101V2 | ReLu | 4.222 | 0.343 |
| InceptionV3 | ReLu | 6.255 | 0.426 |
| InceptionResNetV2 | ReLu | 7.521 | 0.275 |
| DenseNet121 | ReLu | 5.055 | 0.404 |
| DenseNet169 | ReLu | NaN | 0.419 |
| VGG16 | ReLu | 5.708 | 0.251 |
| VGG19 | ReLu | 1.761 | 0.251 |
| ResNet50V2 | Sigmoid | $1.192*10^{-7}$ | 0.135 |
| ResNet101V2 | Sigmoid | $1.192*10^{-7}$ | 0.135 |
| InceptionV3 | Sigmoid | 1.791 | 0.136 |
| InceptionResNetV2 | Sigmoid | 1.732 | 0.264 |
| DenseNet121 | Sigmoid | 1.492 | 0.457 |
| DenseNet169 | Sigmoid | 1.320 | 0.486 |

| | | | |
|---|---|---|---|
| VGG16 | Sigmoid | 1.759 | 0.251 |
| VGG19 | Sigmoid | 1.760 | 0.251 |

**Table 10. Test Activation Function Results**

### 4.2.5 Data Normalization

| Model | Normalization | Loss | Accuracy |
|---|---|---|---|
| ResNet50V2 | True | 1.639 | 0.411 |
| ResNet101V2 | True | 1.346 | 0.479 |
| InceptionV3 | True | 2.824 | 0.462 |
| InceptionResNetV2 | True | 1.482 | 0.403 |
| DenseNet121 | True | 1.262 | 0.502 |
| DenseNet169 | True | 1.299 | 0.519 |
| VGG 16 | True | 1.754 | 0.258 |
| VGG 19 | True | 1.760 | 0.251 |
| NasNetMobile | True | 1.176 | 0.251 |
| ResNet50V2 | False | 1.475 | 0.433 |
| ResNet101V2 | False | 1.302 | 0.497 |
| InceptionV3 | False | 1.394 | 0.441 |
| InceptionResNetV2 | False | 4.707 | 0.459 |
| DenseNet121 | False | 1.245 | 0.525 |
| DenseNet169 | False | 1.853 | 0.525 |
| VGG 16 | False | 1.758 | 0.255 |
| VGG 19 | False | 1.758 | 0.251 |
| NasNetMobile | False | 1.89 | 0.251 |

**Table 11. Test Normalization Results**

# 5. Discussion

## 5.1 Effect of Epochs and Layers

The major issues when tuning epochs for a deep learning model are overfitting and underfitting where a low epoch number can cause underfitting whereas a very high epoch number can lead to overfitting of the data on the model.

The evaluation results show that generally as the epochs are increased we are able to train more with less loss, and higher accuracies for almost all models. This is because in one epoch the network sees the whole dataset and increasing the epochs allows the network to optimise the learning that takes place via the RMSprop optimizer that we are using by updating the weights repeatedly. This allows the accuracy to increase with each epoch as the weights are optimized. Increasing the epochs drastically can lead to overfitting but we believe that the general trend of our performance metrics do not indicate that this has occurred. The best performing models with respect to 10 epochs were the DenseNet121 and DenseNet169 models which gave the best accuracy. They hit the sweet spot both in terms of layers, epochs, and their architecture. For 15 epochs however, models including ResNet50V2, ResNet101V2, InceptionV3, InceptionResNetV2, DenseNet121, and DenseNet169 outperformed the other deep learning models and gave the best accuracies.

A particular point of interest was the NasNet algorithms. The algorithms keep searching for the best convolutional layer or cell and recursively finds new ones, so the number of layers are not fixed in the start, which is why it takes longer to train and use. The NasNetMobile layer image contains thousands of layers. NasnetLarge takes much longer to train and even GPU takes 30mins to train with 10 epochs.

Increasing the network layers for ResNet allows us to increase the accuracy since it might allow us to extract more features. This, however, does not happen with the Inception models since InceptionV3 with 159 layers performs better than InceptionResNetV2 which has 572 layers which may be due to overfitting with the deeper network. Both the ResNet and Inception models perform relatively similarly, but for both 10 and 15 epochs the InceptionV3 model gives the best performance. The DenseNet models with 121 and 169 layers perform better consistently for both epochs. The data also shows underfitting in shallow networks such as VGG16 and VGG19 which contain 16 and 19 layers respectively. These models give the lowest accuracies for both 10 and 15 epochs compared to other models because of the few layers they have and their primitive architecture.

## 5.2 Effect of Learning Rate

The learning rate is a hyperparameter that determines how much the model should adjust when the weights are updated with respect to the encountered error. Too small of a value can lead to a long training phase that may get stuck, while a value that is too high can lead to learning a non-optimal set of weights too quickly or an inconsistent training process.

This effect has been observed in general in all the models. A lower learning rate generally gave higher training times, and always gave higher or equal accuracy as a higher learning rate. We can observe this in for instance, ResNet, VGG, and the Inception architecture. As an example, observe that ResNet101V2 had an accuracy of 0.479 on the test set at a learning rate of 0.01. However as the Learning Rate was increased to 1 the accuracy dropped to 0.177. This outlines

that a suboptimal version of the weights of the models was reached. Perhaps having a learning rate even lower than 0.01 would provide an even more optimal set of weights however, this would increase training times by a considerable amount and hence for the scope of this project was ignored.

One thing should be noted here: the library that was used for the models, Keras, uses methods to update learning rate according to the model performance, and function call backs are used for this. Features such as "ReduceLROnPlateau" [16] allow the model to adjust the learning rate slightly as a constant value for weights is reached in order to perhaps improve further.

## 5.3 Effect of Optimizers

Optimizers in machine learning models are useful in minimizing the loss function. They guide the loss function and make sure the loss function is going in the right direction by tweaking and changing the parameters (weights) of the model.

We tested three different optimizers and each of them with 10 epochs and a learning rate of 0.01, so that the global minima is not missed while using these optimizers. The optimizers are RMSprop, SGD, and Adam. All three of these models are variants of gradient descent and move in both horizontal and vertical directions. RMSprop or Root Mean Square Prop works by keeping an exponentially weighted average of the squares of past gradients. The convergence is sped up after diving the learning rate by this given average. SGD or Stochastic Gradient Descent is calculated by taking a sample batch from the whole of the dataset, this batch is usually random. This works best when the data is random and a SGD picks a random sample in each iteration. Adam or Adaptive Moment Estimation combines ideas of both momentum (past gradients to smooth out) and RMSprop. The way it works is that it takes the weighted average of past gradients, squares them and puts a bias to avoid biases which are directed towards zero.

As per the data in the table in section 4.2.3, the RMSprop optimizer gives better accuracy rates overall compared to the other two. The reason behind that is that models such as NasNet or DenseNet which make use of recurrent neural networks in finding convolutional layers, RMSprop performs pretty well. Other models which RMSprop proves to be more accurate are ResNet101V2 NasnetMobile. The SGD optimizer computes gradients over a smaller and random part of the dataset at each iteration, rather than going over the whole dataset. Such behaviour can be seen in InceptionV3, and InceptionResNetV2 models. Adam optimizer does pretty well on models such as ResNet50V2 and DenseNet169, because it combines both momentum and RMSprop, which is basically making use of previously calculated gradients.

## 5.4 Effect of Activation Functions

Activation functions rank the features up or down based on their correlation with the prediction result. That's why it is important to pick the right activation function in the model regarding the machine learning problem being solved. In emotion recognition, basically we deal with an image classification problem which consists of six classes. For a classification problem, it is best to use softmax activation function in general since it generates predictions with specific classes. As seen in our results in Table 5 and Table 10, softmax activation function is the only one which gives desirable accuracy results for all algorithms (except VGG). Also, it is the best resulting activation function for all algorithms. Picking up softmax activation function is the most reasonable choice in our emotion recognition task.

Comparing the training and test results of softmax, it has a %15 accuracy drop in test results which indicates softmax is not an overfitting activation function. For sigmoid activation function, it gives considerable results only for DenseNet models and it has a %25 accuracy drop in test results which show sigmoid much overfit than softmax. Relu activation function gets intermediate level results for all algorithms which makes it the second optimal activation function for our task after softmax.

Regarding loss values, Relu function tends to have much higher loss values since it is a linear type of activation function. In softmax and sigmoid, they have smooth gradients which reduces the loss value in classification problems. The lower loss values on these functions shows that they have well-fitting smooth gradient property for an image classification task.

In activation function implementation, we could not come up with a code. In ResNet and Inception applications, keras provides a parameter to set activation function in the models. However, this capability is not provided in NasNet. So, we do not have any activation function results for NasNet models.

DenseNet models built-in provide a max pooling and a linear activation function at the last two layers. That's why we need a different implementation to observe the effects of activation functions (relu - softmax - sigmoid) on DenseNet models. For this, we extracted the last two layers of DenseNet121 and DenseNet169 models which are GlobalMaxPooling2D and linear activation, and, we inserted two new layers as GlobalAveragePooling2D and specific activation function. Training the models with default parameters gave reasonable results to analyze the effects of activation functions on DenseNet models.

## 5.5 Effect of Data Normalization

The evaluation results show that ResNet50V2, ResNet101V2, InceptionResNetV2 and both DenseNet architectures obtain higher accuracies if the data is not feature-wise normalized beforehand. Since these models utilize different types of connections in which a layer forwards its output to another layer that comes after it, we think that the reason behind this decrease in accuracy could be that these connections, along with the feature-wise normalization, might be causing overfitting.

An unexpected result of comparing models with normalized and non-normalized data was that the VGG19 and NasNetMobile architectures resulted in a very small difference in accuracy value in both trials (a difference that is smaller than $10^{-3}$). Thus, we can say that these architectures exhibit negligible changes when feature-wise normalization is used.

All other architectures that are not mentioned earlier obtain higher accuracy values when feature-wise normalization is used. This is understandable, since normalization lowers the effects of features that exist in a higher percentage of the dataset, leaving only the differences in the dataset for the model to learn.

## 5.6 Layer Visualizations

### 5.6.1 ResNet50V2

In order to understand the operations and manipulation performed by the deep learning models in different layers, we visualized the image matrix for the first seven layers For the ResNet50V2

model these include the input layer, zero padding layer, 2D convolution layer, another zero padding layer, 2D max pooling layer, batch normalization layer, and the ReLU activation layer. The structure for these first 7 layers is shown below,

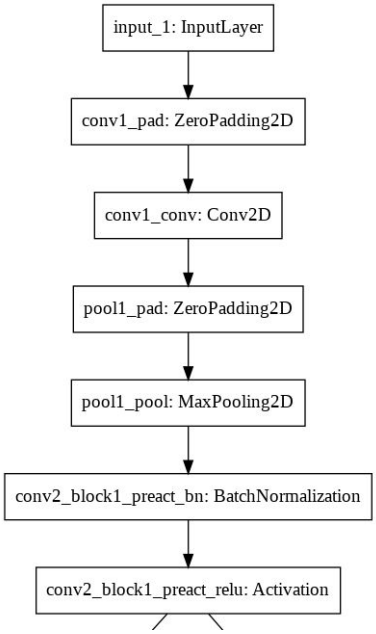


**Fig. 8. The first 7 layers of the ResNet50V2 model**

The outputs for these are shown below in their respective order from left to right,

**Fig 9. ResNet50V2 first 7 layers for an image**

The outputs from each layer show the manipulation done by the layer. For example, the zero padding layer whose output is second from left, pads the image on all sides with zeros in order to preserve the original input size by the end. The output image shows this padding as it looks smaller than the image before. The output of the 2D convolution layer shows the features that were extracted by the convolution and we can clearly make out the important facial features required for emotion detection such as the mouth, and the eye outlines. The 2D max pooling layer takes the maximum from each filter-sized matrix in the convolution matrix and its output (fifth from the left) shows the image after its work is done. This further narrows the number of features, extracting the most important ones from the previous convolution layer.

### 5.6.2 InceptionV3

Regarding the InceptionV3 model, the first seven layers of the model are illustrated below:

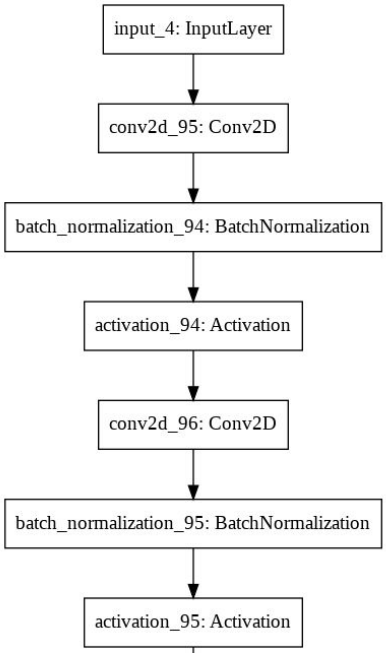


**Fig. 10. The first 7 layers of the InceptionV3 model**

InceptionV3's visualization is given below,

**Fig. 11. The image matrix visualizations of InceptionV3**

Firstly, the images are taken through an input layer in which they receive no change. Then, the images go through the convolution layer, of which the outputs with the selected features can be seen in the second element of the visualizations. Then, the images are taken through a batch normalization layer to be centered with the other images in the batch, and then an activation layer. This convolution-batch normalization-activation layer sequence is then repeated in the fifth to seventh layers.

### 5.6.3 DenseNet121

In DenseNet121, the first seven layers can be listed as input, zero padding, convolution2D, batch normalization, relu activation, another zero padding and max pooling. The illustration for this structure given in Fig. 12 below.

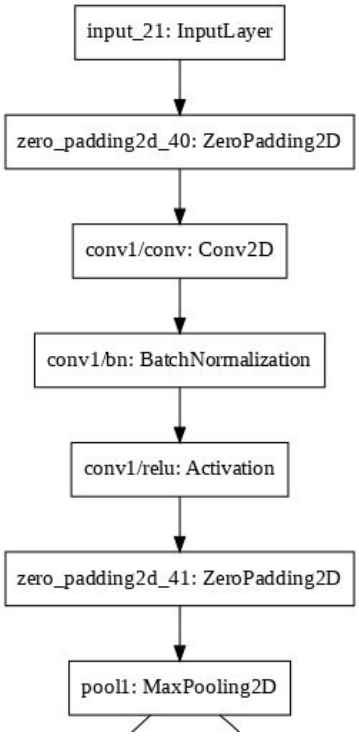


**Fig. 12. The first 7 layers of the DenseNet121 model**

Applying layer visualization, we can observe the effects of such operations; giving the visuals of layer outputs for an image given below.

**Fig. 12. The Visualized Layers of DenseNet121 for an image**

First image is basically the input image. Second layer applies zero padding which fills the sides of the image with zero bits (black in gray-scaling). The most important layer is the convolution layer which extracts the critical features for model prediction, that is, the facial lines in given images. Batch normalization layer boosts the model performance by normalizing the feature values. Activation layer eliminates the features which are not correlated much with the prediction and ranks the features for prediction. Zero padding layer again pads the image from the sides and Max Pooling layer formats the image matrix for output.

### 5.6.4 VGG16


**Fig. 13. The first 7 layers of the VGG16 model**

**Fig. 14. Visualization of a test image from VGG16**

As observed above apart from the image input layer the architecture is divided into 2 sets of convolution / max pooling variants. Each two convolution layers are followed by a max pooling layer. However as the architecture is made for RGB images, it does a poor job at distinguishing features from the grayscale image and hence loses a lot of information. This can explain the poor comparative accuracy of the VGG model in comparison to the others.

### 5.6.5 NasNetMobile

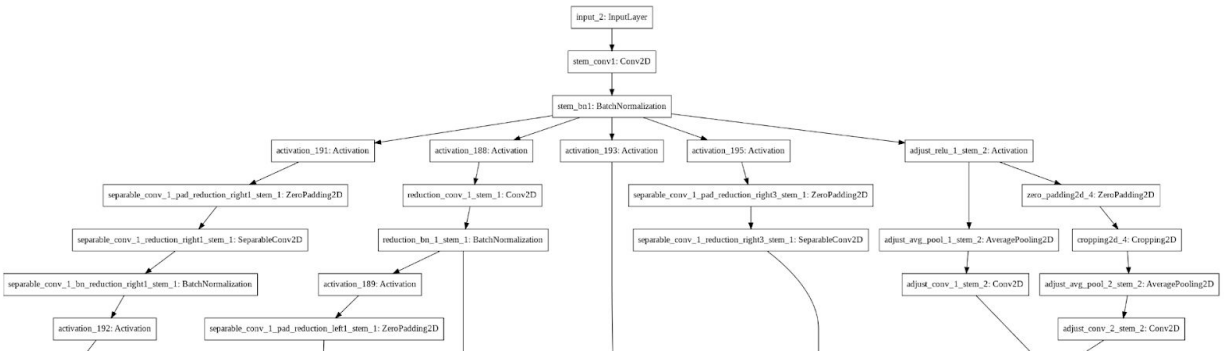


**Fig. 15: Part of the layers created by NasNetMobile model**

**Fig. 16. Visualization of a test image by NasNetMobile model**

As explained before, NasNet models such as NasNetLarge and NasNetMobile do not have predefined convolutional layers, rather they try to find a new layer at each stage using reinforcement learning methods. New layers are found, and later those layers are combined back together. As per the visualization, the model did not achieve preferable results. The first reason is that NasNetMobile requires image size 224x224 pixels, while our images were much smaller comparatively. The other reason is that NasNetMobile works on RGB images, but our dataset had only grayscale images. NasNetLarge is expected to have better results with better accuracy, however it takes hours to complete a single epoch in Nasnet large, despite using GPU on google colab.

## 5.7 Issues Faced

A problem we faced while implementing the Inception models was that the smallest image size they accepted was 75x75, while our dataset contained images of size 48x48. For this reason, we re-sized the images inside the code before training and testing the Inception models.

Similarly, using NasNetLarge would also be difficult to implement in our project, since the image input size is 331x331 pixels, while our dataset includes images of size 48x48 pixels. However we will search techniques to scale down the input image size or scale up our images to see if we can use NasNet network

Converting gray scale images to their RGB versions was also an issue faced with VGG and Nasnet models since they are meant strictly for 3 channel images, i.e. RGB and hence the grayscale matrix values were simply stacked in order to attain a similar result.

# 6. Conclusion

In conclusion, the results clearly show that generally models with deep layers between 100 and 200, trained with 15 epochs, with a low learning rate of 0.01, having an SGD optimizer, and using a softmax activation would give higher accuracies for emotion recognition in still images. We believe that the normalization data seemed to be anomalous when compared to our intuition since it did not show improvement with some model. Therefore, this can be looked into further for future work. The results indicate that there needs to be more development with regards to deep learning architectures in order to increase the accuracy of prediction to an even higher level than our results suggest. However, given that the probability of random guessing was ~16.67%, our models performed relatively well. Therefore, deep learning models are suitable for recognizing emotions in still images, and by extension are a viable tool for image recognition and classification.

# 7. References


[1] Team, K., 2020. *Keras Documentation: Keras Applications*. [online] Keras.io. Available at: <https://keras.io/api/applications/> [Accessed 15 November 2020].

[2] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Identity Mappings in Deep Residual Networks. *Computer Vision – ECCV 2016*, pp.630-645.

[3] Fung, V., 2020. *An Overview Of Resnet And Its Variants*. [online] Medium. Available at: <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035> [Accessed 15 November 2020].

[4] En.wikipedia.org. 2020. *Residual Neural Network*. [online] Available at: <https://en.wikipedia.org/wiki/Residual_neural_network> [Accessed 15 November 2020].

[5] "Advanced Guide to Inception v3 on Cloud TPU | Google Cloud", Google Cloud, 2020. [Online]. Available: https://cloud.google.com/tpu/docs/inception-v3-advanced. [Accessed: 15- Nov- 2020].

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1512.00567. [Accessed: 15- Nov- 2020].

[7] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv.org, 2016. [Online]. Available: https://arxiv.org/abs/1602.07261. [Accessed: 15- Nov- 2020]

[8] Huang, G., Liu, Z., van der Maaten, L. and Weinberger, K., 2016. Densely Connected Convolutional Networks. [online] arXiv.org. Available at: <https://arxiv.org/abs/1608.06993> [Accessed 16 November 2020].

[9] “DAGNetwork,” Pretrained NASNet-Large convolutional neural network - MATLAB. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nasnetlarge.html. [Accessed: 16-Nov-2020]..

[10] S.-H. Tsang, “Review: NASNet-Neural Architecture Search Network (Image Classification),” *Medium*, 01-Aug-2019. [Online]. Available: https://sh-tsang.medium.com/review-nasnet-neural-architecture-search-network-image-classification-23139ea0425d. [Accessed: 16-Nov-2020].

[11] Y. Zheng, H. Wang, and Y. Hao, “CNN study of convolutional neural networks in classification and feature extraction applications,” *Search the world's largest collection of optics and photonics applied research.*, 21-Apr-2020. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11395/113950K/CNN-study-of-convolutional-neural-networks-in-classification-and-feature/10.1117/12.2560372.full?SSO=1. [Accessed: 16-Nov-2020].

[12] K. Team, “Keras documentation: NasNetLarge and NasNetMobile,” *Keras*. [Online]. Available: https://keras.io/api/applications/nasnet/. [Accessed: 16-Nov-2020].

[13] J. Wei, “VGG Neural Networks: The Next Step After AlexNet,” *Medium*, 04-Jul-2019. [Online]. Available: https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c. [Accessed: 16-Nov-2020].

[14] K Simonyan, A Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 16- Nov- 2020].

[15] V. Khandelwal, “The Architecture and Implementation of VGG-16,” *Medium*, 18-Aug-2020. [Online]. Available: https://medium.com/towards-artificial-intelligence/the-architecture-and-implementation-of-vgg-16-b050e5a5920b. [Accessed: 16-Nov-2020].

[16] K. Team, “Keras documentation: ReduceLROnPlateau,” Keras. [Online]. Available: https://keras.io/api/callbacks/reduce_lr_on_plateau/. [Accessed: 20-Dec-2020].

# 8. Appendix

## 8.1. Contributions

The work was divided based on the deep learning models that we have all researched upon. For each member their completed share of work is listed as follows,

- **Hanzallah Azim Burney:** Initial analysis and, performing and testing with different preprocessing techniques on the data. Researching about deep residual networks, specifically ResNet50V2 and ResNet101V2. Fitting the data into the Keras implementations of these models and then evaluating the results using the metrics described in section 3.2.1. Analysed the results, and understood the loss and accuracy metrics from the models in order to gauge performances and derive conclusions. Specifically, analyzed the effect of epochs on the model performance, and worked on interpreting layer visualizations.

- **Sayed Abdullah Qutb:** Researched about different image processing algorithms, mostly convolutional neural networks to be used on the datasets. Of those algorithms, one is NasNet, Neural Architecture Search Network. NasNet has 2 types, Large and Mobile. NasNetLarge contains more parameters and is good for comparatively big datasets with input size 331 by 331, while NasNetMobile is for smaller versions. The implementation of NasNet models are explained in Keras and will be researched further [12]. Analyzed the effect of optimizers on the model performance, and worked on interpreting layer visualizations.

- **Balaj Saleem:** Performed different preprocessing techniques on the data. Researched very deep convolutional networks, specifically VGG16 and VGG19. Fitting the data into the Keras implementations of these models and then evaluating the results using the metrics described in section 3.2.3. Analyzed the effect of learning rate on the model performance, and worked on interpreting layer visualizations.

- **Mehmet Alper Genç:** Performed different preprocessing preprocessing techniques on the data. Researching about the inception architecture for computer vision, specifically InceptionV3 and InceptionResNetV2. Fitting the data into the Keras implementations of these models and then evaluating the results using the metrics described in section 3.2.2. Analyzed the effect of normalization on the model performance, and worked on interpreting layer visualizations.

- **Burak Mutlu:** Researching about preprocessing techniques on image recognition and densely connected convolutional networks, specifically DenseNet121 and Densenet169. Fitting the data into the Keras implementations of these models and then evaluating the results using the metrics described in section 3.2.4. Analyzed the effect of activation functions on the model performance, and worked on interpreting layer visualizations.