TU Dublin, Tallaght Campus

Bachelor of Honours in Computer Science with Data Analytics

Emotion Detection through Image Classification

Leigh Warner

Department of Computing

Supervised by

Keith Nolan

Department of Computing

16 May 2021

Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Science, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Leigh Warner
Student Name
16 May 2021

List of Figures

Contents

Abstract

As the IT sector is making amazing moves in today's world, facial recognition in machines seems to be on the rise. Human's are well capable of calculating a rough estimate of the age and gender of a specific person from a glance. Although for humans, today's machines have improved immensely in the past decade for so, giving them the ability to automatically predict these things from images. This project will try set out, if it can identify the seven basic human emotions from a given face within an image. The tools that this I will utilize within this project will involve a Convolutional Neural Network along side the programming language Python and along with libraries such like Open-CV, TensorFlow and Keras for the construction and development of my Convolutional Neural Network. I will also use existing models such like VGG16, RESNET50 and Google's Inception models to compare against my own CNN to see if it can compete against pre-existing models.

1 Introduction

I have a big interest in Machine Learning and Artificial Intelligence. It has been something I wanted to do since I started my degree back in 2017 and for me that the idea that we can program a computer to generate accurate predictions on the seven basic human emotions is groundbreaking. How we can teach a Convolutional Neural Network and fine tune the hyper-parameters to allow the CNN to learn at it's best ability. In my thesis I will demonstrate my understandings of my work and how I developed my Convolutional Neural Network to predictions on the seven basic human emotions. I will also demonstrate the predictions of existing models that are widely used in Machine Learning, models like VGG16, RESNET50, and Inception3 models that can be used for image classification. These are high end models trained on imagenet weights, which are the best weights that can be used. Imagenet is an image database organized according to the WordNet hierarchy, in which each node of the hiearchy is depicted by hundreds and thousands of images. The project has been instrumental in advancing computer vision and deep learning reseach. This data is free to researchers for non-commerical use.

2 Background Research

Within this section we will hopefully be able to discuss some existing models that currently exist on the web. From researching literature on google scholar we have come to learn that the very first method for age estimation focussing on geometric features of the face that determine the ratios among different dimensions of facial features. These geometric features can separate babies from adults successfully but are in fact incapable of distinguishing between young adults and senior adults (Agbo-Ajala and Viriri 2020). Advancing to the early 2000's, classification and regression methods are ideal to conduct and classify the age and gender based of facial images. Classification methods such as Support Vector Machine (SVM) based methods are common methods for age and gender classification (Sunil 2017). SVM is a supervised machine learning algorithm that can analyse data which can be used for classification or regression methods, but mostly used in classification methods (Agbo-Ajala and Viriri 2020),(Sunil 2017). Methods for age and gender predictions consist of Linear Regression, Support Vector Regression (SVR) is characterized by the use of kernels sparse solution, and VC control of the margin and the number of support vectors (Awad and Rahul 2015). Canonical Correlation Analysis (CCA) is a statistical method that is employed to investigate the relationship of two or more variable sets, that contains two or more variables (Thompson 2005), and Partial Least Square (PLS) are in fact the most common methods of regression for age and gender predictions. In more recent times, more and more researches have moved away from these early stages and advanced to using a Convolutional Neural Network (CNN) for the prediction of gender and age classification. The advantages of using such neural network have allowed researchers to advance the classification of facial features from images relying on its good feature extraction technique. For this project, A Convolutional Neural Network will be developed with the primary programming languages being Python and availing of the libraries it has to offer. The libraries that will be used in this project will consist of OpenCV, TensorFlow, Keras, Pillow and Matplotlib. While the project is not entirely novel, there are some existing tools available on the web. Starting with Microsoft Azure, they're tool which is a Face Client Library allowing you to use within C, Go, and Python. To use Microsoft's library, you will need an Azure subscription to gain access to

their facility. Microsoft offer code examples for implementing their cognitive services, as well as class names and interface descriptions (Microsoft 2021). Moving onto Google's Vision API, Google offer a powerful pre-trained machine learning model through REST and RPC API's, allowing you assign labels to images and quickly classify them into millions of predefined and handwritten texts. They use AutoML Vision to help train your data. They allow you to upload images and train them using the AutoML using the easy-to-use graphical interface. There are more options to optimize your model for accuracy, latency, and size (**Google**). As much as these two models have their own unique way of creating a prediction system for gender and age through an image, this is not something I will implement within my project. I want to focus on using the Convolutional Neural Network alongside Python and OpenCV.

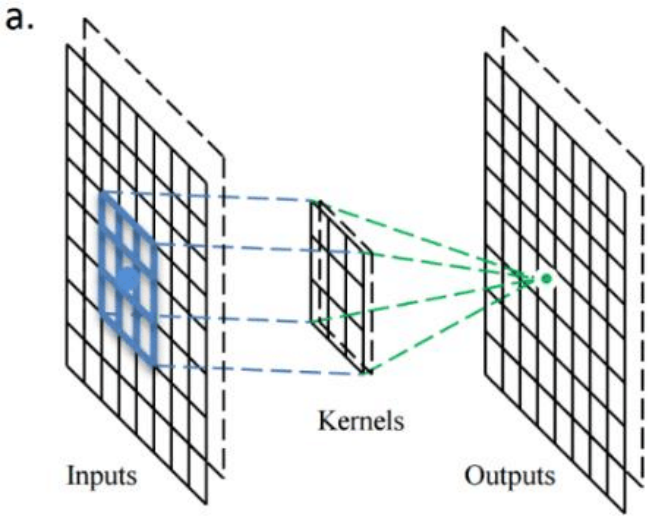
3 Model Overview

As stated in previous sections, I discussed that my project would use a Convolutional Neural Network for the purpose of predicting the seven basic human emotions. A Convolutional Neural Network, which is also known as a CCN or ConvNet for short, is a class of neural networks that concentrates in the processing of data that has a grid-like topology, such as an image (Mishra 2020). An image to a human is a visual representation of something that we can see but to a machine it is a binary representation of that image (ibid.). It contains a series of pixels that are arranged in a grid-like trend that contain pixelated values to help indicate how bright and what specific colour each pixel should be in the image. A human brain can understand an enormous amount of information just from a glance of an image. The brain itself contains 100 billion neurons, each neuron working in its own receptive field being fully connected to other neurons that they cover the whole visual field (ibid.). By developing a Convolution Neural Networks, we are enabling sight to computers just like humans (ibid.).

3.1 Convolutional Layer

If we look deeper into a CNN, it typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer (Mishra 2020). The convolutional layer is the first layer within a CNN, this layer extracts features from an input image (ibid.). Convolution maintains the relationship between pixels by learning image features using small squares of input data (ibid.). This layer preforms a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field (ibid.). The kernel is spatially smaller than an image but is more in-depth (ibid.). This means that if an image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels. During the forward pass, the kernel essentially slides across the height and width of an image, producing the image representation of that receptive region (ibid.). The result of this returns a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of that image (ibid.). The sliding size of the kernel is called a stride.

Figure 1: Below image is a diagram explaining the operation of a convolutional kernel.



If we have an input impage of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following forumla (Mishra 2020):

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Figure 2: Formula for Kernel.

If we take the above formula for the Convolution Layer, it will produce an output volume of size Wout x Wout x Dout (ibid.).

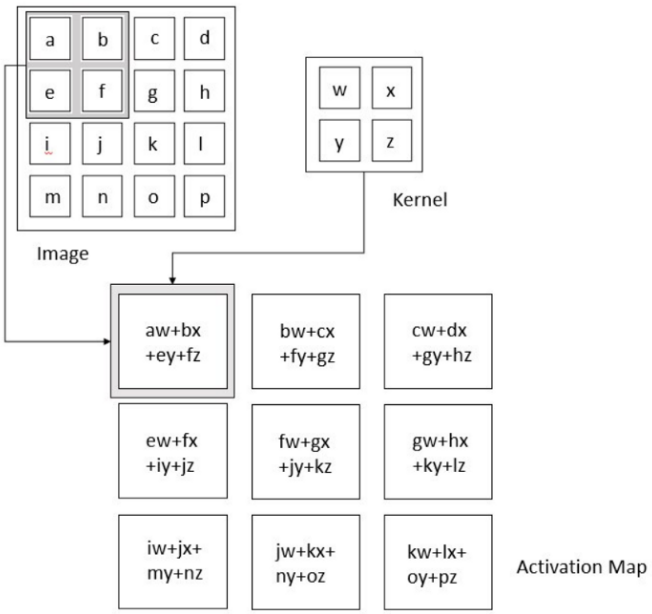


Figure 3: Output volume.

3.2 Pooling Layer

After the Convolution Layer comes the Pooling Layer. The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs (Mishra 2020). This is used to help reduce the spatial size of the representation, which in-turn decreases the required amount of computation and weights (ibid.). The pooling operation is processed on every slice of the representation individually. There is three different pooling types within this layer. The Global pooling, Max pooling, and Average pooling. Max pooling and Average pooling are the most popular among the three types of pooling (ibid.). All of these pooling types can be imported using the Keras library in python.

1. Global Pooling is used to reduce each channel in the feature map to a singular value. If we had a feature map of **nh x nw x nc** it is then reduced to **1 x 1 x nc** feature map (Savyakhosla 2019).

2. Max Pooling operation selects the maximum value from the specific region that the filter is covering on the feature map (ibid. Therefore, the output of the max-pooling layer would consist of a feature map that contains the most distinguished feature of the prior feature map (ibid.).

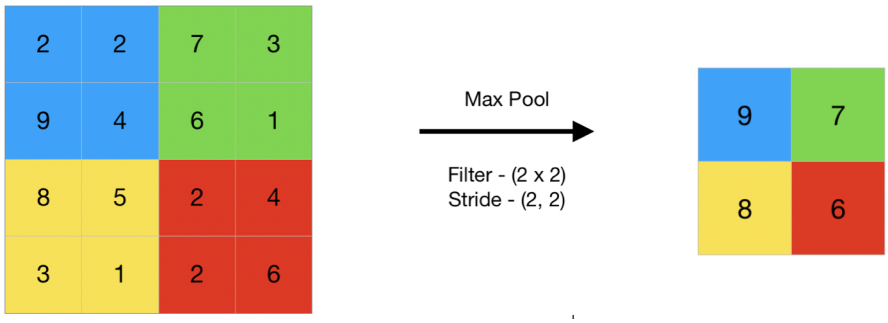


Figure 4: Max Pooling Operation.

3. Average Pooling calculates the average of the elements that are currently present in that region of the feature map covered by the filter at that time (ibid.). While max pooling gives the most distinguished feature in that

particular region of the feature map, average pooling calculates the average of features present in that specific region (Savyakhosla 2019).

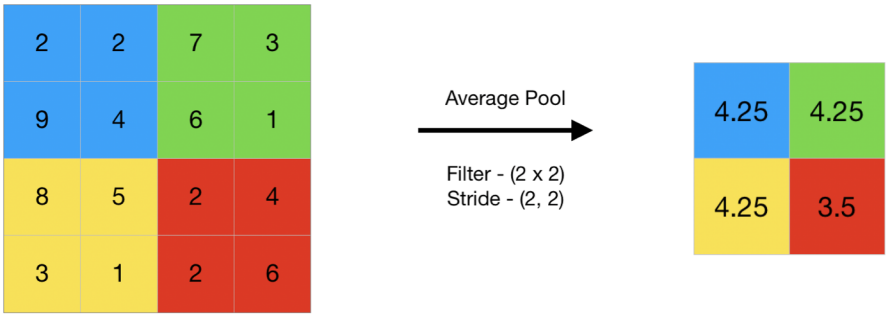


Figure 5: Average Pooling Operation.

3.3 Fully Connected Layer

The Fully Connected Layer is the next-to-last layer within a CNN. The fully connected layer outputs a vector of K dimensions where K is the number of possible classes that the convolutional neural network will be able to predict (Arc 2018. This vector contains the probabilities for each class of any unclassified image that is being classified (ibid.). Below is an image displaying a Fully Connected Network.

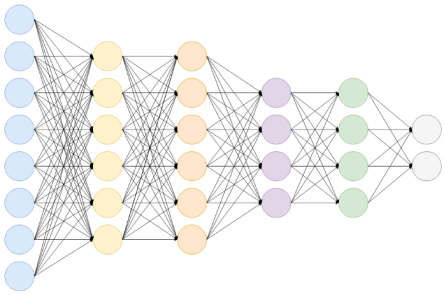


Figure 6: Fully Connected Layer

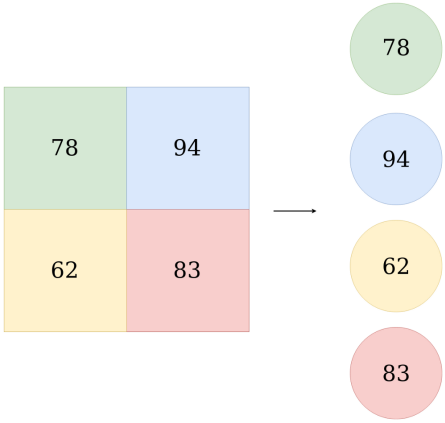


Figure 7: Flattening Layer

The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is then flattened and passed into the fully connected layer (ibid.). Flattening is the process in which we can transform a two-dimensional matrix that consists of features into a vector.(ibid.). We flatten the output of the convolutional layers to create a single long feature vector and is then connected to the final classification model, which is called a fully-connected layer (ibid.).

4 Model Development

For the development of my project, I availed of Google's Colaboratory, or Colab for short. Colab allows anybody to write and execute arbitrary python code through a web browser and is well suited for machine learning and data analysis. Since my project was based on the development and fine tuning of a Convolutional Neural Network, I wanted to make use of a graphics card (GPU) as they can process multiple computations simultaneously. GPU's have a large number of cores which allows for better computation of multiple parallel processes. Since I wanted to make use of a GPU, Google's Colab offers a free GPU service along with Colab and allows for anyone to access this with a Google account. Of course, there are other services out there that would allow me to avail of a GPU like Azure, a Data Science Virtual Machine and even AWS but Google's Colab suited me perfectly. To begin the construct and implementation of a Convolutional Neural Network I had to import the TensorFlow framework in Python. TensorFlow is an open-source end-to-end platform for creating Machine Learning applications. It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks. TensorFlow has an architecture that allows for all the development to work, its broken into three parts. Preprocessing the data, Building the model, and Training and estimate the model. It is called TensorFlow because it takes input as a multi-dimensional array, also known as tensors. It allows you to construct a sort of flowchart of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as an output.

4.1 Data Collection

During the process of identifying a suitable dataset for the proposed Data Analytics project, there was some other options available before myself and my supervisor came to a decision. The first option was to manually collect the data myself, obtaining images from family and friends as my source for my dataset. This would require a huge range of ages, ethnic backgrounds and gender to give my dataset the full opportunity of achieving its potential with the Convolutional Neural Network. Unfortunately, this idea collapsed before it even took off, when asking my peers through the local chat we use, the response I got were very childlike. Containing jokes and completely different answers to the question I had put in the chat. Obtaining the photos from family members also did not work to my hope, due to my other college work and work outside college meeting up with family members was quite difficult, they also had work to attend just like myself. Due to the current global state with the Coronavirus, this also made an impact on some family members who felt unease at the thought of meeting up, which was acceptable as I too was uneasy. The second most identifiable dataset was a dataset my supervisor stumbled across; this dataset contained a suitable number of respectable images that I could use for my Convolutional Neural Network but required me to submit a contact sheet in order for the images to be released to me. I submitted the form on grounds for it solely being used for my Data Analytics project, but this fell through as the time pushed on and no response was made back to me. The last option was a dataset that consisted of 72 images. Even though this dataset contained too little images that would not allow my Convolutional Neural Network to get the best result it could possibly get, this was the best I had, so myself and my supervisor agreed that this dataset would suit. In order to have a suitable dataset I had to enlarge the image count to an ideal requirement for the CNN to effectively work. The format for the data was to be kept as .JPG across all the images, its best I keep all the images under the one format to avoid confusion when interacting with the data at a later stage in the data pre-processing stage.

4.2 Data Augmentation

The data pre-processing section of the project has been the most interesting thus far. As my data set is very small, consisting of 72 base images which is not enough alone to train the model and have a good accuracy of that model. Taking this issue into account, I done some research and considered the option of using libraries Python has to offer. Image Data Generator which is a sub library of Keras. Image Data Generator allowed me to augment my images in real-time, allowing me to change some aspects of these images and apply augmentation to these images. The function offers many augmentation techniques, but I only availed of a handful of these. I applied a vertical and horizontal flip to my images, sheared my images in a counterclockwise direction along with applying a random zoom value to my images. Below is the script I used to allow me to do such augmentation to my images.

```
# imports
import numpy as np
import os
from PIL import Image

#  uisng the image data generator to augment my data
datagen = ImageDataGenerator(
        horizontal_flip = True)

# empty list
dataset = []

# initialzing the directory where images are stored
image_directory = r'C:\Users\leigh\Desktop\Project\images\train\surprise/'
SIZE = 256
dataset = []

# listing the images from the directory
my_images = os.listdir(image_directory)

for i, image_name in enumerate(my_images):
    # taking the images that are .jpg
    if(image_name.split('.')[1]=='jpg'):
        # reading the images
        image = io.imread(image_directory + image_name)
        image = Image.fromarray(image, 'RGB')
        # resizing image to 256 x 256
        image = image.resize((SIZE,SIZE))
        # appending to numpy array
        dataset.append(np.array(image))

x = np.array(dataset)

# saving the images to a specific folder with prefix and format
# eg. hf = horizontal flip
i = 0
for batch in datagen.flow(x, batch_size=32,
                          save_to_dir=r'C:\Users\leigh\Desktop\Project\images\train\surprise/',
                          save_prefix='hf',
                          save_format='jpg'):
    i += 1
    if i > 5:
        break
```

I also applied rotation to my data set, going in rotates of 15 all the way to 360 allowing me to grow my data set substantially overtime. The Pillow library allowed me to open, manipulate and save images when working with them throughout the data augmentation stage. Below is a working script of how I was able to rotate my images and save them using the Pillow library.

```
# imports
import os
import glob
from PIL import Image
os.chdir("./")

# going through each file
for file in glob.glob('images/r330/*.jpg'):
    # opening file
    image = Image.open(file)
    # rotating the new images by 15 degrees incrementally
    image_rot_15 = image.rotate(330)
    # saving the new rotated image
    image_rot_15.save(file.replace(".jpg", "r330.jpg"))
```

After the data augmentation stage that allowed me to grow my data set that would give my model a wide range of data, I started on some pre-processing before running my model. To begin with, my data is stored in 7 classes, each identifying as a human emotion i.e., angry, disgust, fear, happy, neutral, sad, and surprised. I then opened the images and converted them to grey scale using Pillows Image.convert() function, which allowed me to convert the images into grey scale. From there I resized all my images to 28 by 28 and appended them to an array while appending the class value to another array. I done this step for every class within my data set then reshaped my data before normalizing the data and randomizing the data set.

```
# loading images from directory using glob
for filename in glob.glob('/content/drive/MyDrive/images/training/anger/*.jpg'):
    # opening image and converting to greyscale
    im=Image.open(filename).convert('L')
    # resizing the image
    im=im.resize((28, 28))
    # appending images to arry
    arr = np.array(im)
    # appening arr to X
    X.append(arr)
    # appening the class value i.e. 1 = Anger to Y list
    Y.append(1)  # anger class
```

In terms of attribute selection for my model, the only attributes that were needed for my model were the class identifiers, that being the emotions. I placed the images into their right folder that would allow the model to associate the class value to that image then appending that class value to an array. After I done all this to my model, I normalized my data, the goal in normalizing my data is to change the values of numeric columns in my data set. After normalizing my data I then went onto shuffle both the X and Y lists. The reason why we shuffle the data before training is that it helps the training converge fast.

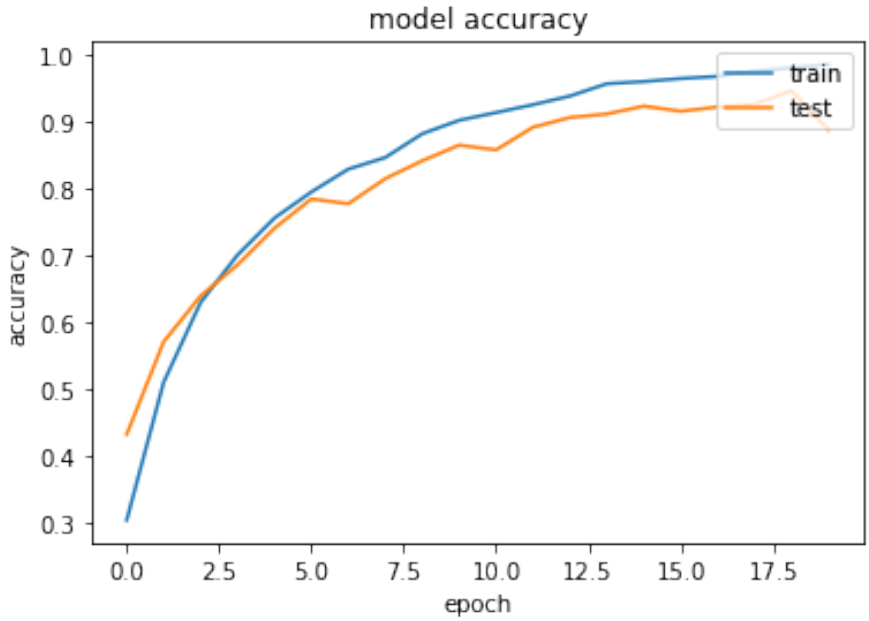
4.3 Developing CNN

During the development of my CNN, I decided to develop four models that had different parameters, hyper parameters and architecture to experiment with different models that would return the best accuracy. When fitting the four models I used a validation split of 0.33. This will allow for my models to train on 66 percent of the data while the validation trains on the remaining 33 percent of the data. All models consist of valid padding in that all models assume that the dimensions of all in the input images are valid and they get fully covered by a filter and the stride I specify. All model activation functions consist of the Relu. The reason being is that during my time in the Applied Artifical Intelligence and Deep Learning class with Keith Quille we discovered that Relu was in fact the best activation function to use.

1. **Model A** consisted of two Convolution Layers, beginning with a filter of 32 then 64. Both kernel sizes are set to (3, 3) along with valid padding for all the layers in this model. I have in input shape of (1, 28, 28) as I have convert all my images to greyscale and resized them in the pre-processing stage. I also have MaxPooling after each convolutional layer so that I can get the maximum value from the feature map. After the Convolution layer and Max Pooling layer, I have Flatten my data into a 1-dimensional array to allow me to input it into my Dense layers. This is then connected to my Fully-Connected layer which allows me do make classifications based on my model with the activation function of Softmax. The reason why I use softmax here is that I have more than one classes within my model, thus I use categorical crossentroy as my loss. Within this model and every other model I avail of model checkpoint and early stopping. Model checkpoint allows me to make a checkpoint on the models weights for the validation accuracy and Early stopping allows me to stop the models training early when it reaches its maximum validation accuracy and stops the model early to avoid over fitting of the model. For my first investigation I gave a training exposure of 20 epochs with a batch size of 64.

```
modelA = Sequential()
modelA.add(Conv2D(32, (3, 3), strides=1, padding="valid", input_shape=(1, 28, 28), activation='relu', data_format
modelA.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelA.add(Conv2D(64, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelA.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelA.add(Flatten())
modelA.add(Dense(128, activation='relu'))
modelA.add(Dense(num_classes, activation='softmax'))


modelA.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

checkpoint = ModelCheckpoint("CNNModelA_1.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_on

early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')
```

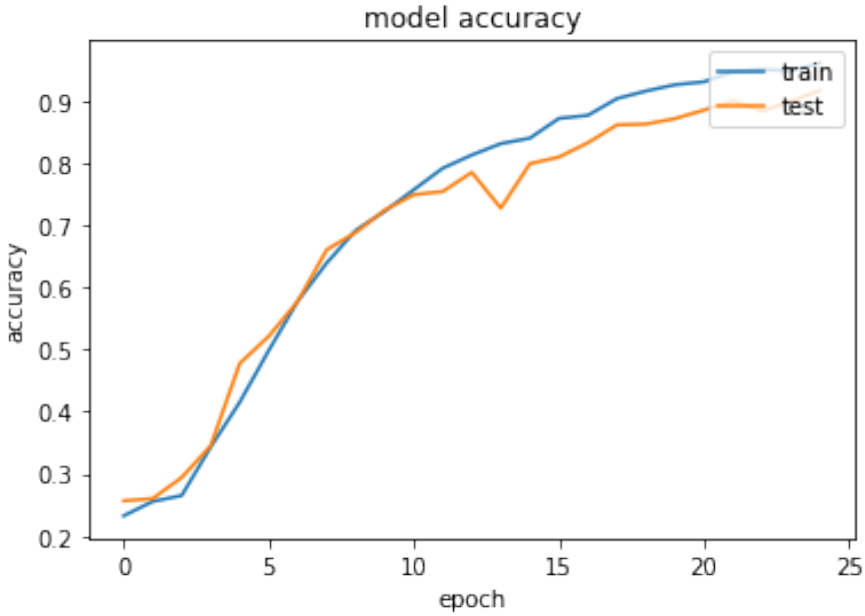
From interpreting the graph we can see that the first investigation went quiet well, the model has a slight bit of over fitting but good none the less. For the first model validation accuracy I have an accuracy of 88.90 percent for the validation set.



2. **Model B** differed from my first model, in that I added an extra convolution layer with a kernel size of 128 and a new dense layer with a kernel size of 512. I also changed that batch size of the model to 256 to see if this will cause any affect to my models accuracy, along with a new batch size I also gave model B 25 epochs to train on. This is an increase of 5 epochs from the last model. Below is a code snippet of the model B's architecture for reference.

```
modelB = Sequential()
modelB.add(Conv2D(32, (3, 3), strides=1, padding="valid", input_shape=(1, 28, 28), activation='relu', data_format
modelB.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelB.add(Conv2D(64, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelB.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelB.add(Conv2D(128, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelB.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelB.add(Flatten())
modelB.add(Dense(128, activation='relu'))
modelB.add(Dense(512, activation='relu'))
modelB.add(Dense(num_classes, activation='softmax'))


modelB.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

#implementating model checkpoint and early stopping
checkpoint = ModelCheckpoint("CNNModelB_1.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_on

early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')


# Fit the model
history = modelB.fit(X, Y, validation_split=0.33, epochs=25, batch_size=256, verbose=1, callbacks=[checkpoint,ear
```

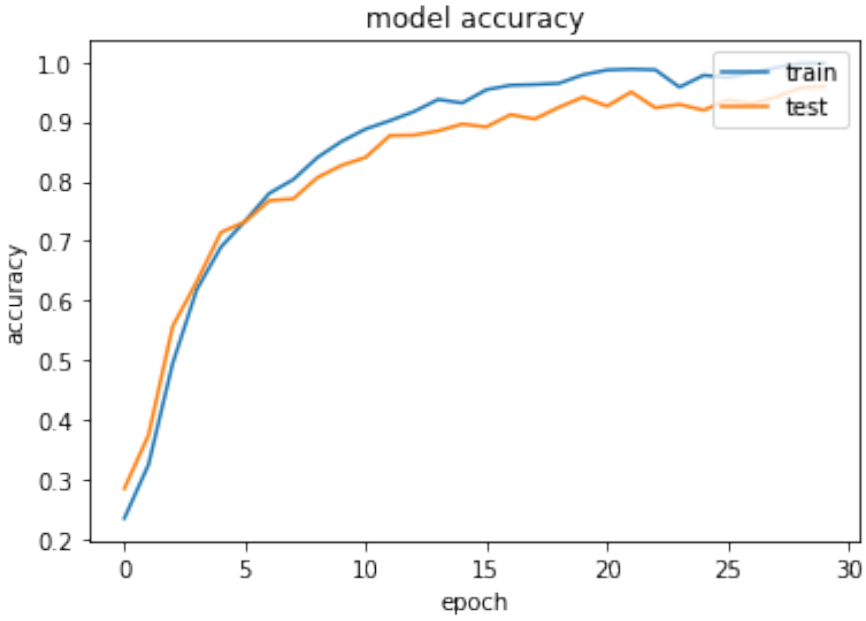
Model B trained quite well in terms of accuracy, hitting a respectful 91.75 percent. An overall increase of 2.85 percent from my first model investigation. The below image represents the training of the model, showing the training and testing sets model accuracy.


model accuracy
train
test
accuracy
epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
5
10
15
20
25

3. **Model C** For my Model C, I decided to develop my network even deeper than before, adding in an extra convolutional layer with a filter size of 256. I also gave my network an extra 5 epochs to expand the amount of training exposure that my model would get resulting in 30 epochs for the model. With the image below, we can see the extra ConV2d layer along with the 30 epochs of training. These small changes to fine tuning the model have increased the models accuracy.

```
modelC = Sequential()
modelC.add(Conv2D(32, (3, 3), strides=2, padding="valid", input_shape=(1, 28, 28), activation='relu', data_format
modelC.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelC.add(Conv2D(64, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelC.add(MaxPooling2D(pool_size=(2, 2), padding="valid"))

modelC.add(Conv2D(128, (2, 2), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelC.add(MaxPooling2D(pool_size=(1, 1), padding="valid"))

modelC.add(Conv2D(256, (1, 1), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelC.add(MaxPooling2D(pool_size=(1, 1), padding="valid"))

modelC.add(Flatten())
modelC.add(Dense(256, activation='relu'))
modelC.add(Dense(128, activation='relu'))
modelC.add(Dense(num_classes, activation='softmax'))


modelC.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

checkpoint = ModelCheckpoint("CNNModelC_1.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_on

early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')

# Fit the model
history = modelC.fit(X, Y, validation_split=0.33, epochs=30, batch_size=64, verbose=1, callbacks=[checkpoint,earl
```

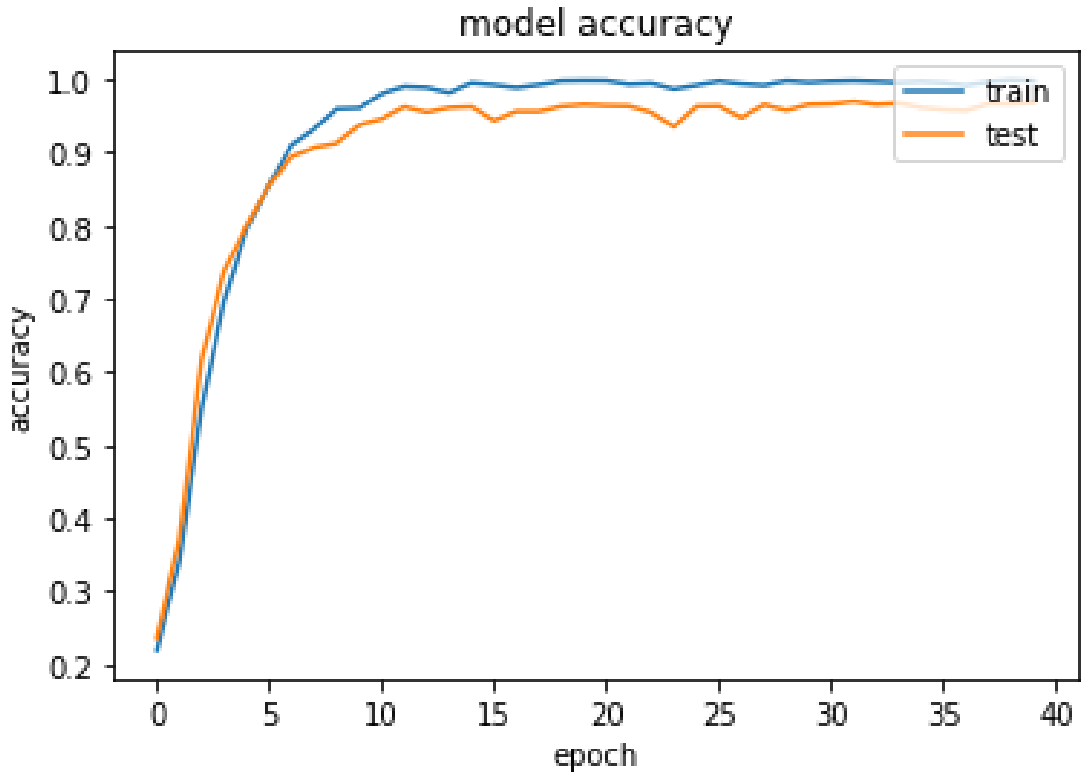
This model seemed to be the best model thus far with an overall accuracy of 95.916 percent. Comparing model C to previous models A and B we can see that this model seems to good. Both the training and testing data set have learned at a good rate, a slight margin of over fitting still occurring but a really good model none the less. The diagram below represents the models accuracy .


model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30

4. **Model D** has been quite different from the previous three models in that I have left the same convolution structure as model C but have added more dense layers after the Flatten layer. I have two more fully connected layers at 1024 and 512 kernel size with the activation function of Relu. I have also given the model 40 epochs to further expose the model to more training then before with a batch size of 128.

```
modelD = Sequential()
modelD.add(Conv2D(32, (3, 3), strides=2, padding="valid", input_shape=(1, 28, 28), activation='relu', data_format
modelD.add(MaxPooling2D(pool_size=(1, 1), padding="valid"))

modelD.add(Conv2D(64, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelD.add(MaxPooling2D(pool_size=(1, 1), padding="valid"))

modelD.add(Conv2D(128, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelD.add(MaxPooling2D(pool_size=(1, 1), padding="valid"))

modelD.add(Conv2D(256, (3, 3), strides=1, padding="valid", activation='relu', data_format='channels_first'))
modelD.add(MaxPooling2D(pool_size=(1, 1), padding="valid"))

modelD.add(Flatten())
modelD.add(Dense(1024, activation='relu'))
modelD.add(Dense(512, activation='relu'))
modelD.add(Dense(256, activation='relu'))
modelD.add(Dense(128, activation='relu'))
modelD.add(Dense(num_classes, activation='softmax'))


modelD.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
modelD.save_weights("modelD.hdf5")

checkpoint = ModelCheckpoint("CNNModelD_1.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_on
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')

# Fit the model
history = modelD.fit(X, Y, validation_split=0.33, epochs=40, batch_size=128, verbose=1, callbacks=[checkpoint,ear
```

Model D has surpassed model C in accuracy score, scoring an accuracy of 97.014 percent. It is marginally highly then the previous model but overall I am happy with how this model turned out. We can see that both the training and testing sets learned a good steady increase, perhaps the training exposure could be small as both the training and testing set hit a straight line where about 10 epochs seems to be sufficient training exposure.

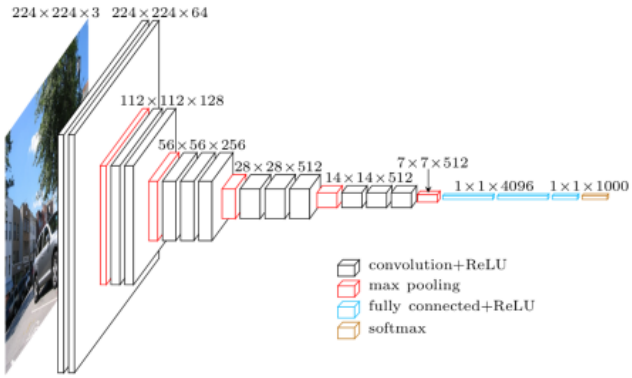

model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30
35
40

4.4 Architecture of Existing Models

The models that I have used in this project are VGG16, RESNET50 and Google's Inception 3. The reasoning behind this is that I wanted to test pre-existing models against my own model that I have developed and try to see if my own Convolutional Neural Network is as good as pre-existing models.

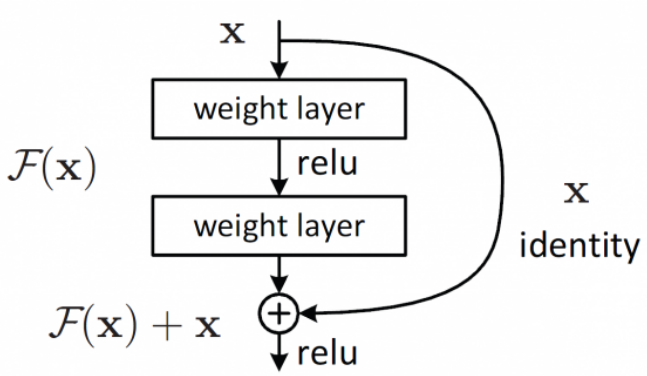
4.5 VGG16

VGG16 is a convolution neural network that was used to win ILSVR (Imagenet) competition back in 2014 (Rohit 2019). It is still considered to be one of the best vision models to this date. VGG16 is quite unique, in that instead of consisting of a large number of hyper-parameters they focused on the model having convolution layers of 3x3 filter with a stride 1 and consistantly using same padding and maxpool layer of 2x2 filter of stride 2 throughout the architecture of the VGG16 model (ibid.). After all these layers, the model is completed with 2 fully connected layers which is then followed by softmax for output (ibid.). The reasoning behind the 16 in the VGG name is that the vgg model is made of 16 layers that have weights. This is considered to be a large network and it contains about 138 million parameters approxmly (ibid.).



4.6 RESNET50

ResNet50 is a form of the ResNet model which is made up of 48 Convolution layers accompanied by 1 MaxPool and 1 AveragePool layer (OpenGenus n.d.). ResNet makes it possible to train up to hundreds or even thousands of layers and still manages to achieve captivating results (ibid.). ResNet was originally applied to the image recognition task but it can also be used for non computer vision tasks to also achieve better accuracy (ibid.). When developing and training a neural network, there comes a point in time where an overall increase in depth will cause accuracy to saturate, then deteriorate swiftly (Kurama 2020). This problem with deep neural networks is called the degradation problem. This brings attention to the fact that not all neural network architectures are likely easy to optimize. Within the ResNet model, it utilizes a method called residual mapping to overcome this problem (ibid.). Within in this method the ResNet model explicitly lets the layers fit a residual mapping instead of hoping that every couple stacked layers directly fit a desired underlying mapping (ibid.). Below is a diagram that shows the building blocks of a Residual Network, where F(x)+x can be realized by feedforward neural networks with shortcut connections.



4.7 Inception-v3

The Inception-v3 model is a convolutional neural network architecture that comes from the Inception family. This v3 model makes serval improvements inclduing Label Smoothing, Factorized 7 x 7 convolutions, and uses an auxiliary classifier to generate the label information lower in the network (PapersWithCode n.d.). The Inception-v3 model is comprised of symmetric and asymmetric building blocks. These building blocks include convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers (ibid. The model also applies Batchnorm laregly throughout the architecture and applies it to the activation inputs (Kurama 2020). The loss for the model is computed through Softmax. The architecture of the v3 model is continuously built in step-by-step pattern (ibid.).

1. **Factorized Convolutions:** this aids to lessen the computational efficiency as it reduces the number of parameters involved. It can also keep a check on the overall network efficiency (ibid.).

2. **Smaller convolutions:** After the factorized convolutions comes smaller convolutions which help lead to faster training. The reasoning behind this is to allow for less parameters in the network (ibid.).

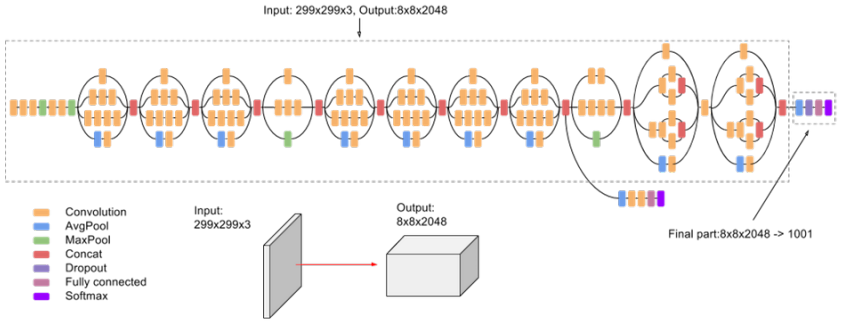
3. **Asymmetric convolutions:** If we take a 3 x 3 convolution, this can be replaced by a 1 x 3 convolution which can be followed by a 3 x 1 convolution but if we take the 3 x 3 convolution and replace it by a 2 x 2 convolution, the overall number of parameters would be marginally higher than the asymmetric convolution proposed (ibid.).

4. **Auxiliary classifier:** This is a small CNN that is essentially inserted directly between layers during the training process, which takes the loss and adds to the main network loss (ibid. In the Inception-v3 model the auxiliary classifier acts as a regularizor for the network (ibid.).

5. **Grid size reduction:** This is where feature maps are connected with a convolution layer of stride two and a max-pooling later in parallel for progression (ibid.).

If we take all of the above concepts and consolidate them into one architecture,

the given result would be the Inception-v3 model show below.

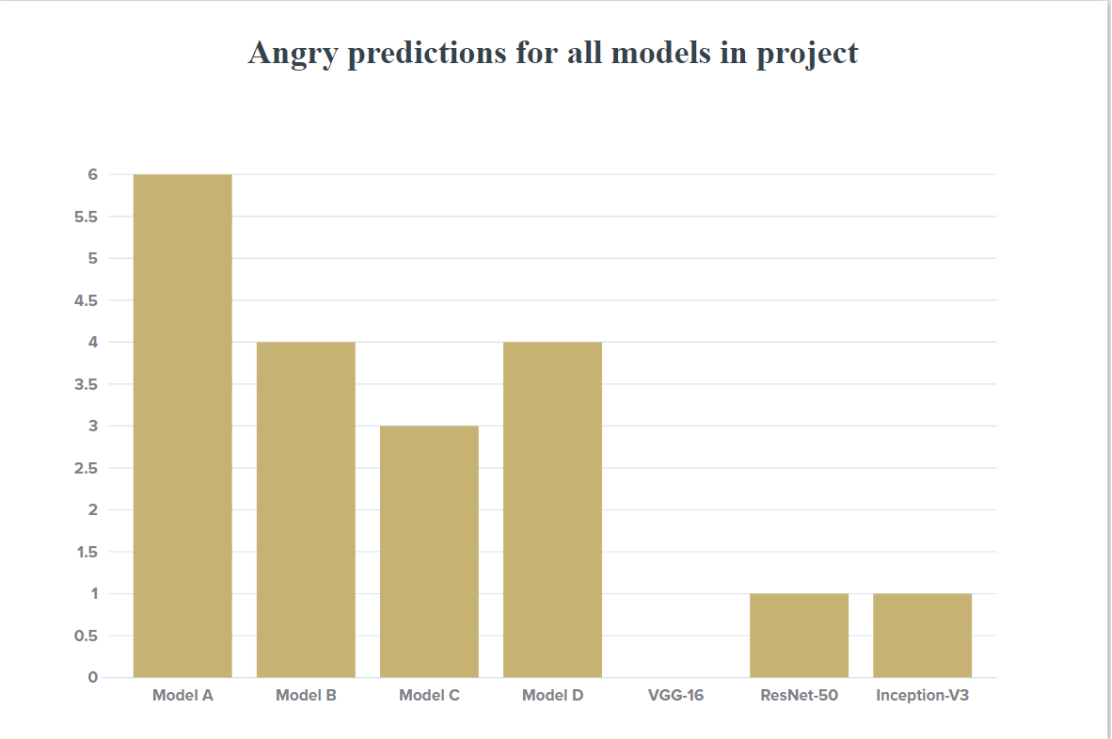


5 Results

Every model that was developed in this project had to be tested on unclassified images. The images where sourced online and hand picked one by one. Overall I picked 140 images assuming that each model would run 20 predictions per class. Every prediction followed the same class value, in that I started with the first class value predictions being 1 i.e. Angry then making my value through the remaining emotions. I have accompanied charts for each prediction on the model to gain a better understanding of which model had the highest predictions of each emotion starting with Anger.

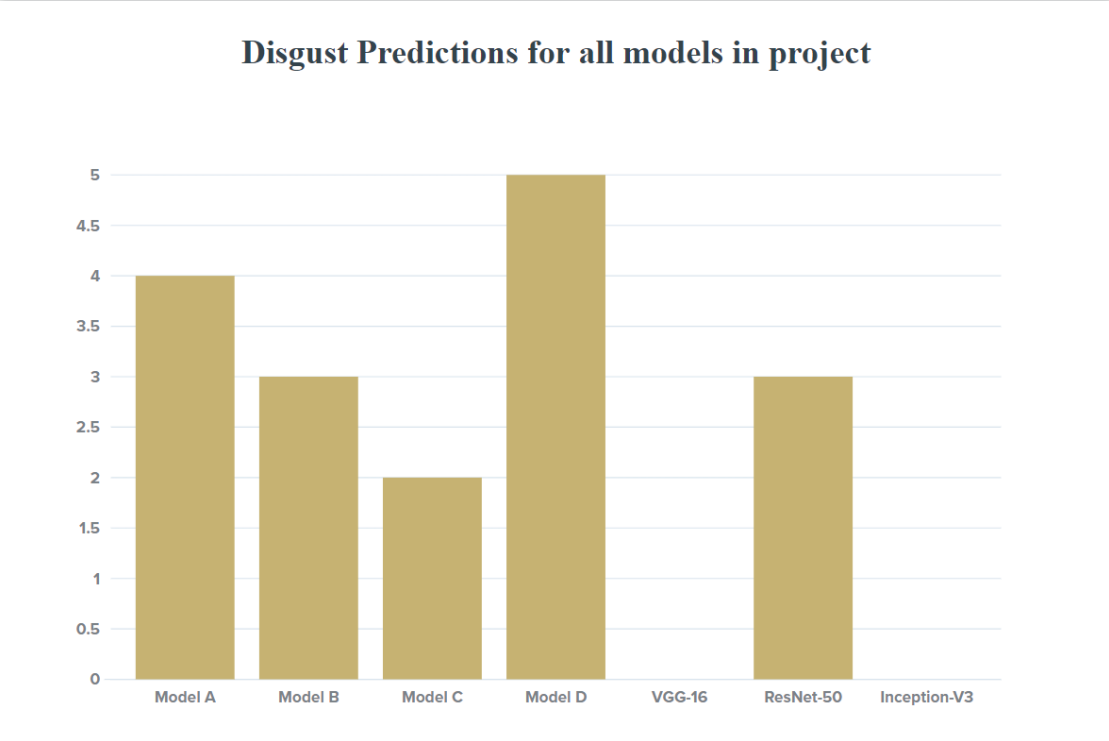
5.1 Anger

If we study the bar chart we can clearly see that all the models that I developed had a higher prediction rate than those that are pre-trained models. In fact model A had the highest result in predictions for the angry emotion and VGG-16 had the least amount of predictions. Next is the Disgust emotion, again 20 unclassified images were predicted across all seven models to see which model had the highest prediction score for the disgust emotion.



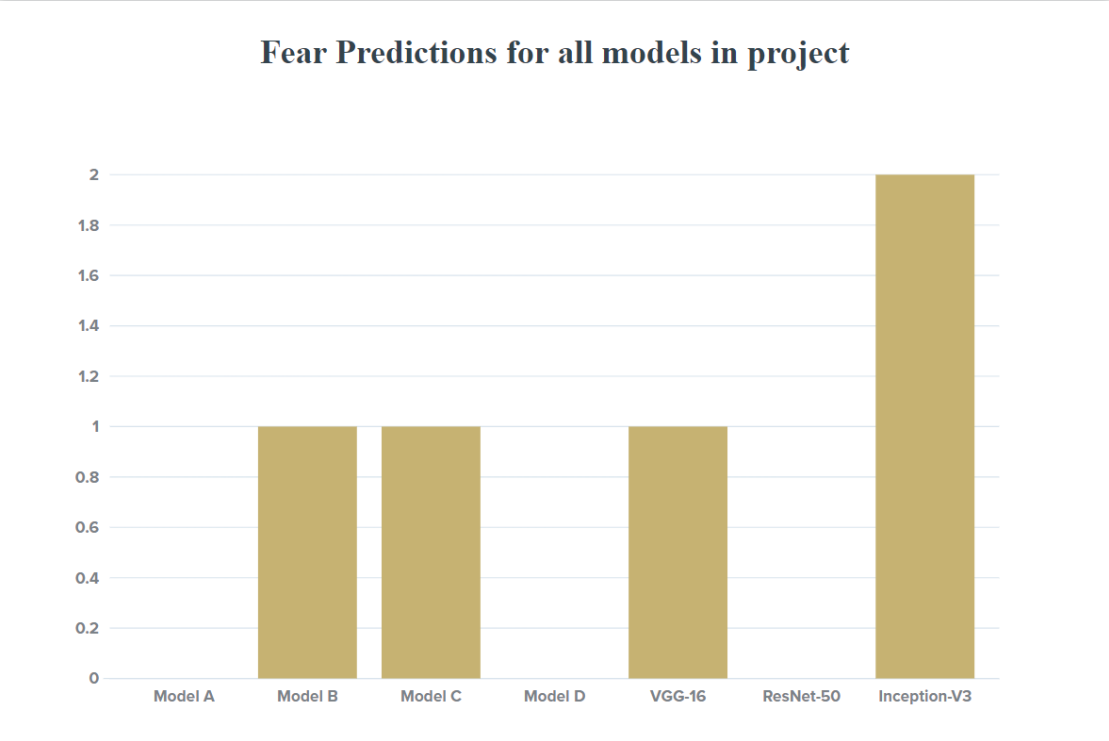
5.2 Disgust

Again we can see that Model D had the highest predictions of 5 out of 20 unclassified images with both the pre-trained models VGG-16 and Inception-V3 predicting 0 out of 20 unclassified images. Model A comes in at second with 4 predictions whereas Model B and ResNet-50 have a tied score of 3 out of 20 predictions. Model C has 2 out of 20 leaving VGG-16 and Inception-V3 with the least amount of correct predictions. The next emotion to predict is the Fear emotion.



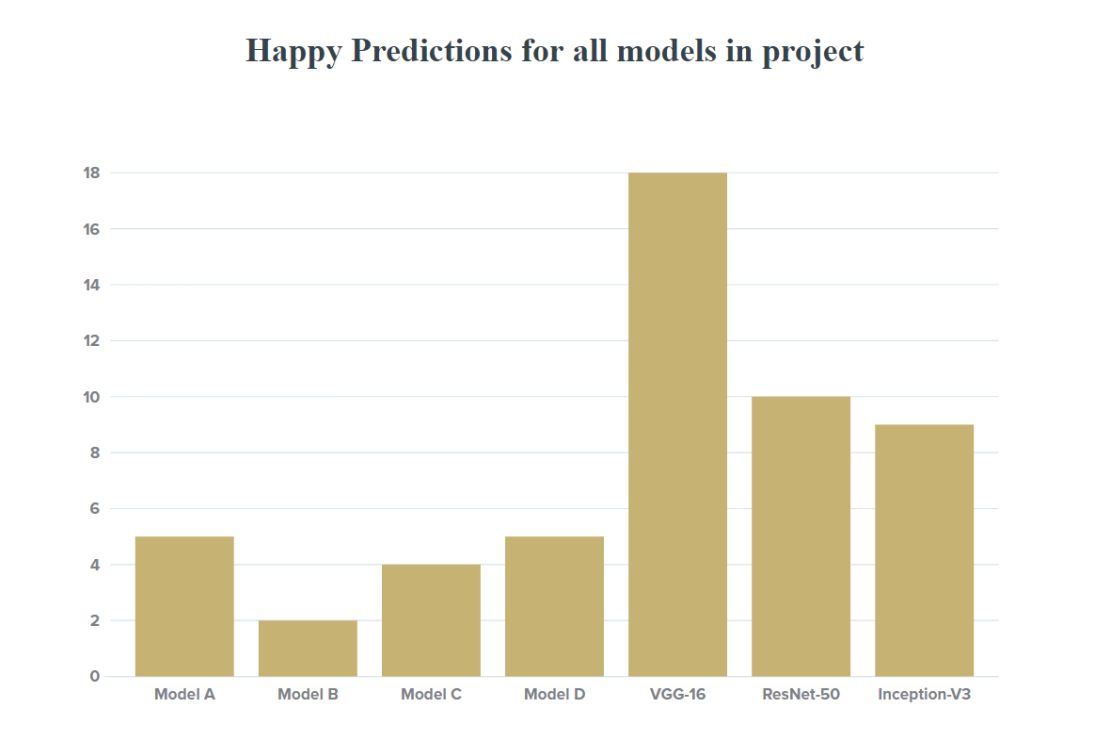
5.3 Fear

The Fear predictions did not go so well right across every model, I do have my conclusions to why this happened with alot of the models that I will discuss in a later section of my thesis. We can see that the Inception-V3 model has made the highest predictions of 2 out of 20. Model B, C, and VGG-16 making 1 correct prediction out of 20 and Model A and D making no correct predictions at all. Moving onto the next prediction, we have the Happy emotion.



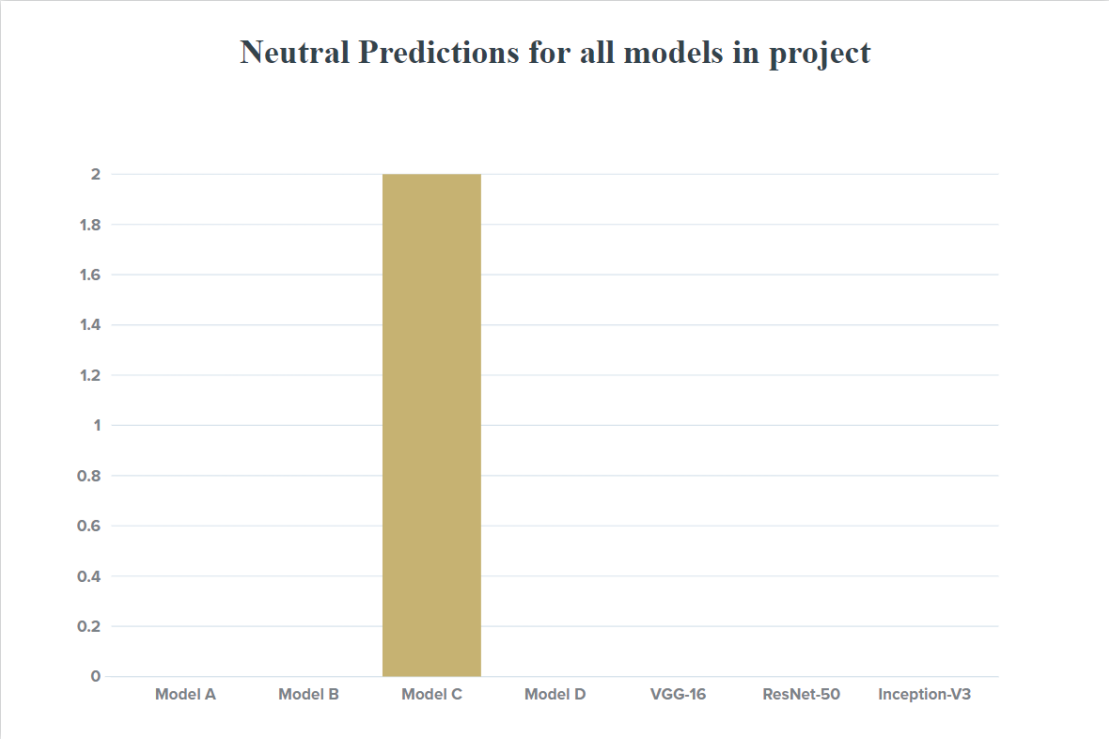
5.4 Happy

The Happy emotion seems to have preformed quite well for the pre-trained models VGG, ResNet-50, and the Inception-V3 model. VGG-16 making 18 out of 20 correct predictions, the best score we've seen thus far.ResNet and Inception marginally apart with ResNet making 10 out of 20 predictions and Inception making 9 correct predictions. Models A, B, C, and D still making correct predictions but just not as good as the pre-trained models. Model A and D making 5 correct predictions, Model C making 4 correct predictions and Model B only making 2 correct predictions. Even though the models I developed did not have good scores like the pre-trained models but they still made a handful of correct predictions which is good none the less. After the happy emotion, we come to the Neutral emotion prediction's.



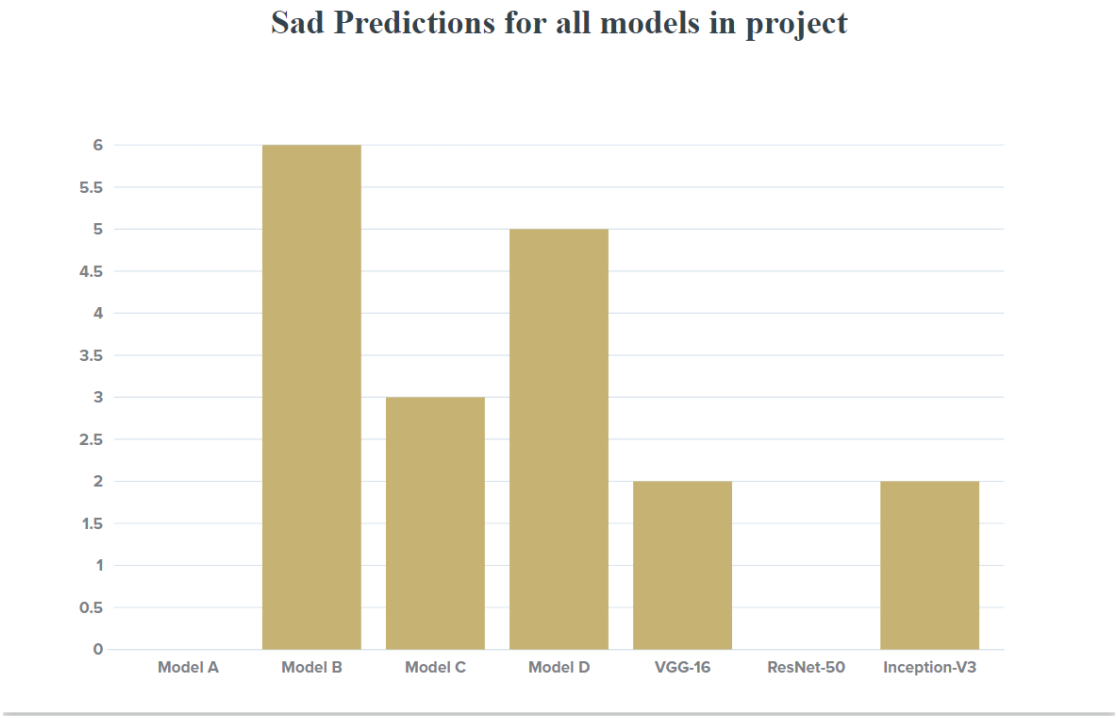
5.5 Neutral

As we can see from the graph detailing all the model predictions for the Neutral emotion, straight away we can see that all models except for model C made no correct predictions on the unclassified images for the Neutral emotion. Model C being the only model that made any correct predictions only made 2 correct predictions, the Neutral emotion predictions have been the poorest predictions thus far. After the Neutral prediction's, we have the Sad predictions.



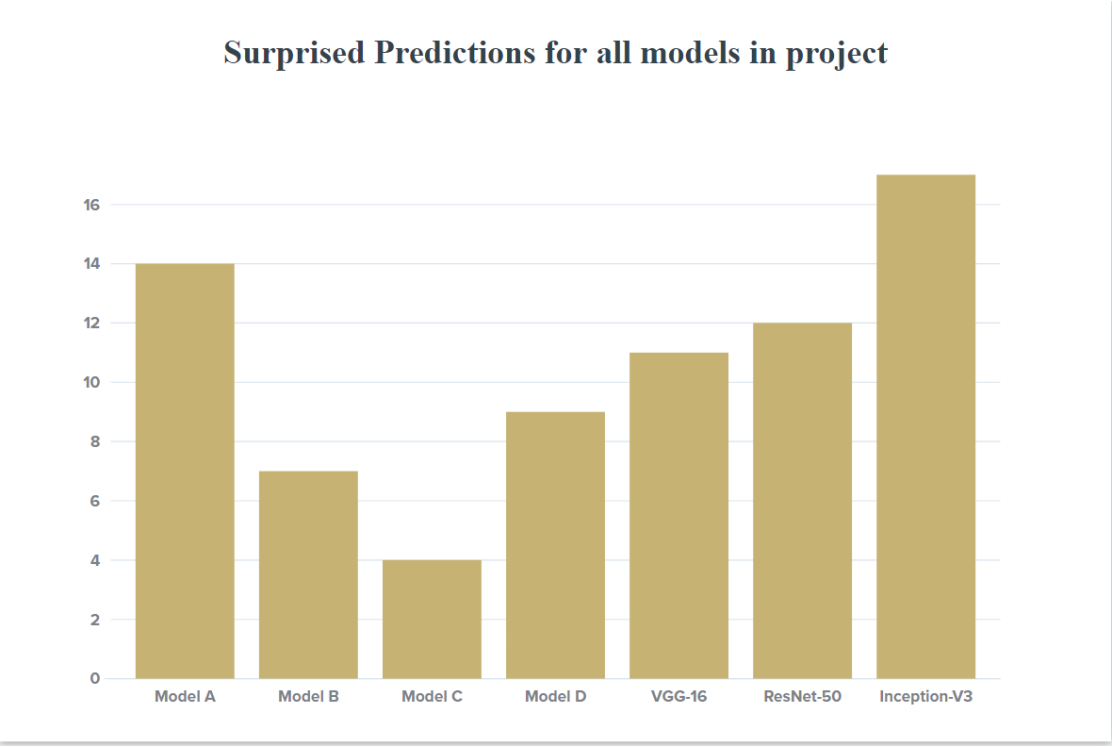
5.6 Sad

The predictions for the Sad emotions were not great either with the highest score only being 6 which was made by Model B. We see two models, Model A and ResNet-50 models make 0 correct predictions out of 20. We see Model D only make 5 correct predictions and Model C make 3 correct predictions out of 20 thus coming after these models is the VGG and Inception models, these pre-trained models only managed to make 2 correct predictions out of 20 unclassified images. Overall the Sad predictions have not been the greatest but we can seen worse scores on other emotions.



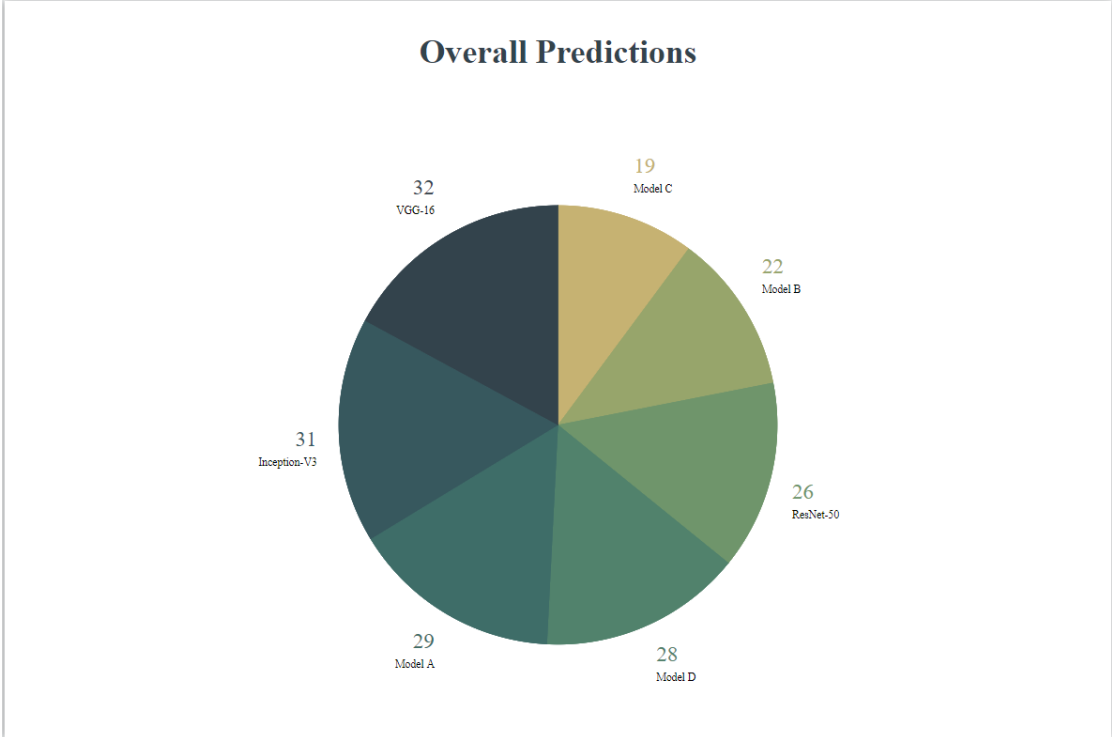
5.7 Surprised

The Surprised emotion is the last emotion to run predictions across all the models. Just like the Happy predictions we can see that the predictions for the Surprised emotion have seen some good results. Starting with the Inception model, we can see that this model made 17 correct predictions followed by Model A with 14 out of 20 predictions made successfully. After that we have the ResNet model with 12, the VGG model with 11 and Model D with 9 correct predictions. Next we have the lower scores of Model B and C. Model B making 7 predictions correctly and Model C only making 4 predictions correct.



5.8 Overall Results

If we look at the results as a whole for all the models we can see that not one model made 50 percent of the 140 predictions correctly. The model with the most amount of predictions made right was the VGG-16 model, making a total of 32 predictions correct across the board. Following the results of the VGG model we have the Inception model making 31 out of the 140 predictions correct. After that we have my own model, Model A coming in with 29 correct predictions overall. After Model A, comes Model D with 28 correct predictions out of the 140 unclassified images. Following that we have 26 predictions made correct by the ResNet model. Last but not least we can see that Model B made 22 predictions and Model C made only 19 predictions correct.



6 Discussions and Conclusions

Overall this has been quite an interesting project. I have expanded my knowledge into the Artificial Intelligence side of computing thus leaving me wanting to take my own career path down it. This project has in lightened me more on the fact that computers are able to learn the basic human emotion and even make predictions on it. This just shows how advanced computers have come in the last decade or so, making amazing advancements in technology. Overall I am happy with how the project went but I do want to discuss some elements of the project. We have seen in the Results section of my thesis that all models did not make exceptional results. When doing the predictions on the models I had found that the majority of the models would return a happy or surprised prediction no matter the prediction. Thus leaving me to believe that the data for the project was not sufficient in that the rest of the data for the other emotions was not good. We've seen that the highest prediction for the fear emotion was 2 with the rest of the models being one or zero. The Neutral, Disgust, Angry, and Sad predictions were also not good in that the highest out of all the predictions for those emotions was 6. This leaves me to believe that the data for the emotions Angry, Disgust, Fear, Neutral and Sad was not detailed enough in that all the models had a tough time trying to make correct predictions on the unclassified images for these emotions. If we turn our attention to the Happy and Surprised emotions we can see that almost every model had a good or decent amount of correct predictions on the unclassified images, this leads me to believe that the data for the Happy and Surprised emotions was good whereas the rest of the data may not have been. If given the chance to redo this project, I would invest more time into the data collection process of the project as it really defines the learning for the models. I do believe that with better data that the current models would preform better than the results given within the thesis.

7 References

References

Agbo-Ajala, Olatunbosun and Serestina Viriri (2020). 'Deeply Learned Classifiers for Age and Gender Predictions of Unfiltered Faces'. In: *The Scientific World Journal* 1. DOI: 10.1155/2020/1289408. URL: https://doi.org/10.1155/2020/1289408.

Arc (2018). 'Convolutional Neural Network'. In: *Towards Data Science* 1. URL: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05#:~:text=Fully%20Connected%20Layer%20is%20simply,into%20the%20fully%20connected%20layer..

Awad, Mariette and Khanna Rahul (2015). *Support Vector Regression*. Vol. 1. SpringerLink. DOI: https://doi.org/10.1007/978-1-4302-5990-9_4. URL: https://link.springer.com/chapter/10.1007/978-1-4302-5990-9_4.

Kurama, Vihar (2020). *A Review of Popular Deep Learning Architectures: ResNet, InceptionV3, and SqueezeNet*. URL: https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/.

Microsoft, Azure (2021). *Quickstart: Use the Face client library*. URL: https://docs.microsoft.com/en-us/azure/cognitive-services/face/quickstarts/client-libraries?tabs=visual-studio&pivots=programming-language-csharp.

Mishra, Mayank (2020). 'Convolutional Neural Networks Explained'. In: 1. URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20Convolutional%20Neural%20Network%2C%20also,topology%2C%20such%20as%20an%20image.&text=Each%20neuron%20works%20in%20its,cover%20the%20entire%20visual%20field..

OpenGenus (n.d.). *Understanding ResNet50 architecture*. URL: https://iq.opengenus.org/resnet50-architecture/#:~:text=Machine%20Learning%20(ML),-More%20Less%20Go&text=ResNet50%20is%20a%20variant%20of,explored%20ResNet50%20architecture%20in%20depth..

PapersWithCode (n.d.). *Inception-v3*. URL: https://paperswithcode.com/method/inception-v3#:~:text=Inception%2Dv3%20is%20a%20convolutional,use%20of%20batch%20normalization%20for.

Rohit, Thakur (2019). 'Step by step VGG16 implementatio in Keras for beginners'. In: *Towards Data Science* 1. URL: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c#:~:text=VGG16%20is%20a%20convolution%20neural,vision%20model%20architecture%20till%20date.&text=It%20follows%20this%20arrangement%20of,consistently%20throughout%20the%20whole%20architecture..

Savyakhosla (2019). 'CNN | Introduction to Pooling Layer'. In: *GeeksforGeeks* 1. URL: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/.

Sunil, Ray (2017). 'Understanding Support Vector Machine (SMV)'. In: *Analytics Vidhhya* 1. URL: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

Thompson, Bruce (2005). 'Canonical Correlation Analysis'. In: *Encyclopedia of Statistics in Behavioral Science*. American Cancer Society. ISBN: 9780470013199. DOI: https://doi.org/10.1002/0470013192.bsa068. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470013192.bsa068. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470013192.bsa068.