Artificial Intelligence & Machine learning 1 – SEM2.

Machine Learning System Development

Daniel Kavanagh

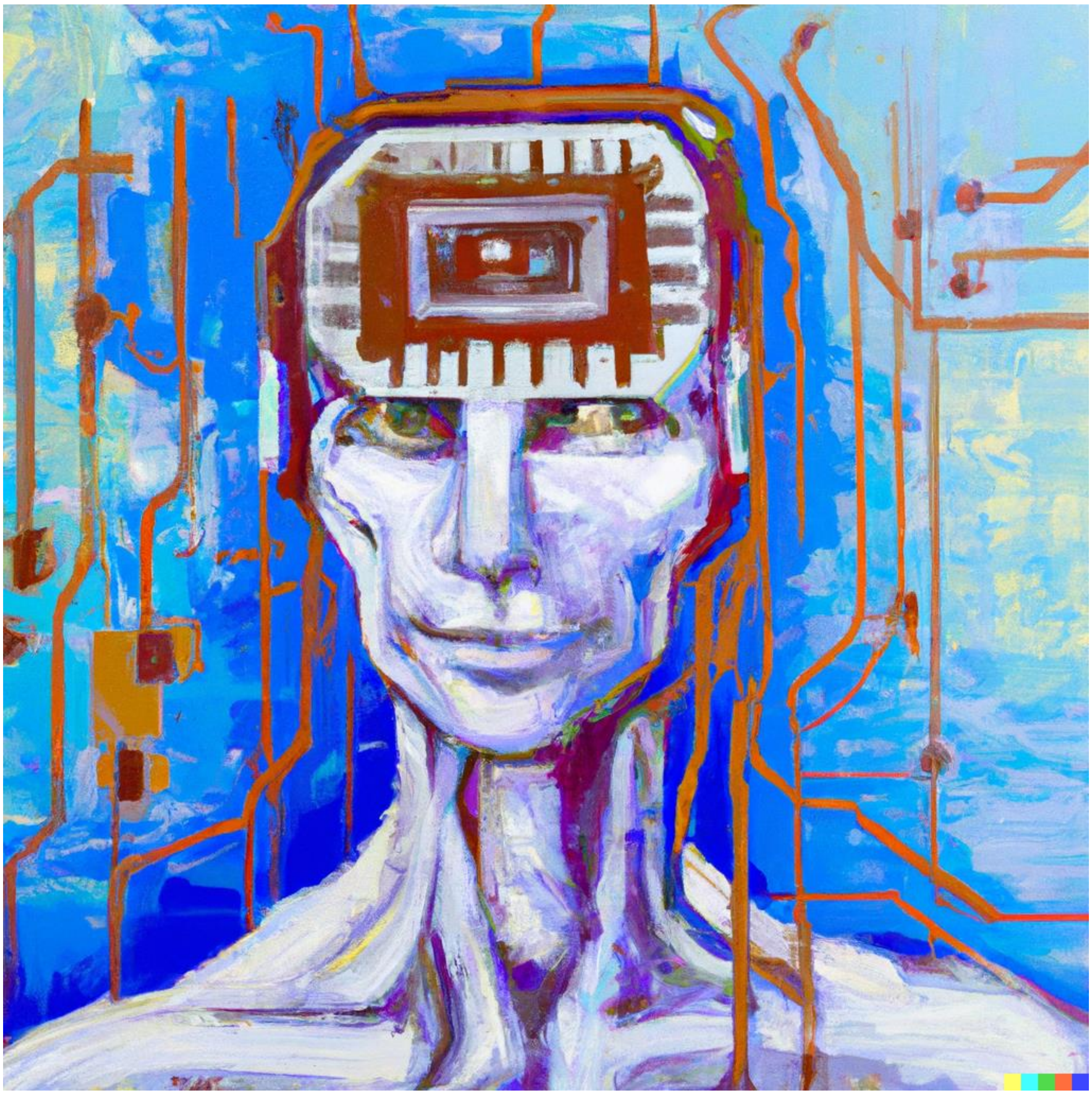


Figure 1 - Created using Open Ai's DALL:E 2: "An oil painting of artificial intelligence"

Table of Contents

Introduction/ Project Plan

This project aims to develop a machine learning image/ object recognition system whereby it can distinguish single or multiple faces within a given image. The system will be evaluated on different images, including its own test set from the main dataset straight after training.

The project mainly focuses on data exploration, pre-processing, feature analysis, different machine learning algorithms, and finally, performance evaluation.

I intend to use the provided dataset to solve this assignment, which consists of 2 .npy (saved NumPy array) files: one consisting of images of faces, the second consisting of images not containing faces. Both datasets will be explored in more detail throughout this report.

I intend to also use a Convolutional Neural Network (CNN) to solve this problem.

System Design/ Development

Below is a flow chart highlighting the development process for both training the model and using the model to make predictions once training is completed.

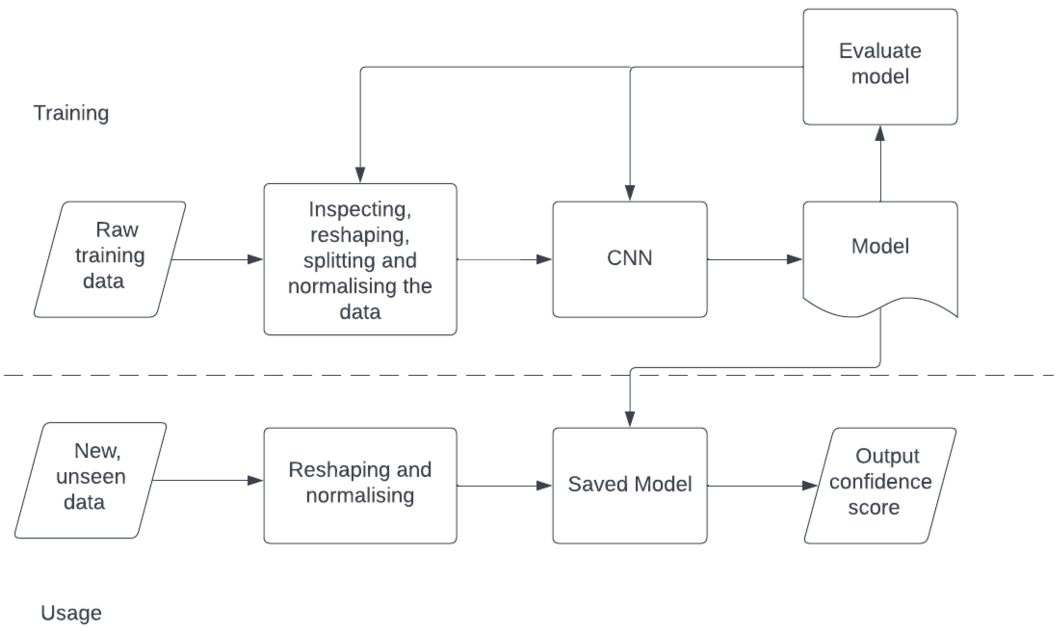
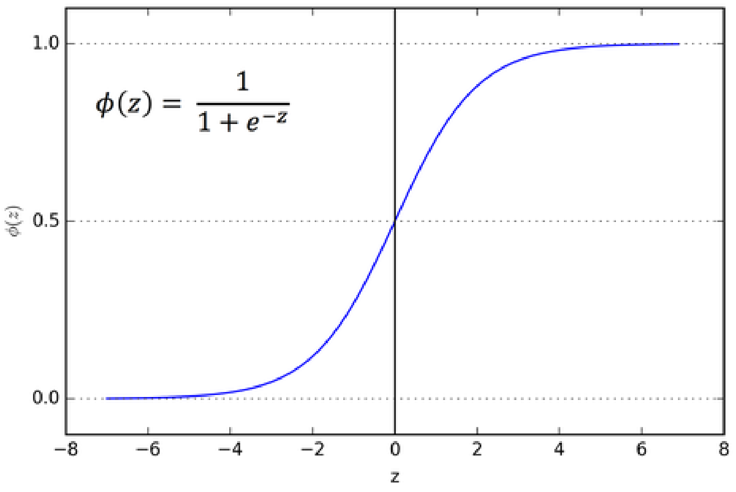


Figure 2 – Development process of a CNN

The training section of the flow chart demonstrates how it has been tweaked after model evaluation, this includes changing activation functions, changing the number of epochs, and adjusting the number of layers. This is to prevent overfitting, underfitting and to produce a lower loss and higher accuracy.



The chosen activation function for this CNN is the Sigmoid activation function, whereby weights from the nodes/ neurons are between 0-1. This is important as the output of the model is binary (face detected or not).

System implementation strategies

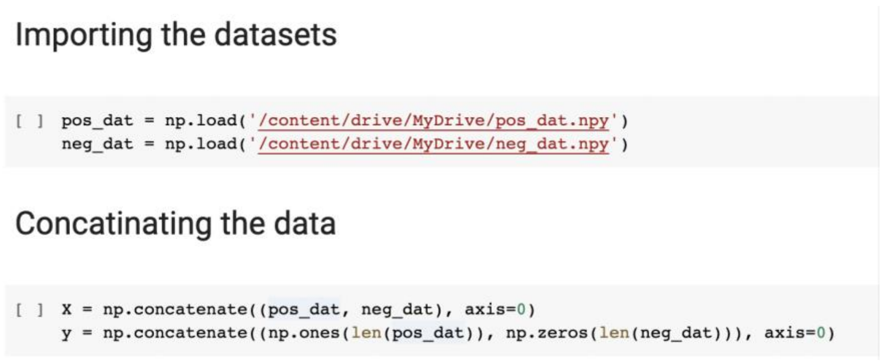
- ## Data collection methods

As mentioned before, I will be using the provided dataset to solve this problem.

- ## Data pre-processing methods

In this section, I will cover the entire data pre-processing pipeline for training the model.

The first step of data pre-processing is to split the data into X and y values. Once this has been done, the data can be reshaped and split as shown below.



Importing the datasets

```
[ ] pos_dat = np.load('/content/drive/MyDrive/pos_dat.npy')
    neg_dat = np.load('/content/drive/MyDrive/neg_dat.npy')
```

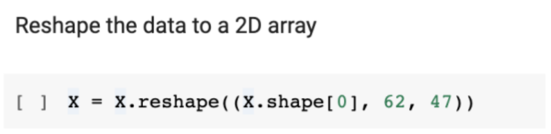
Concatinating the data

```
[ ] X = np.concatenate((pos_dat, neg_dat), axis=0)
    y = np.concatenate((np.ones(len(pos_dat)), np.zeros(len(neg_dat))), axis=0)
```

Figure 3 - Importing the datasets and splitting into X and y values.

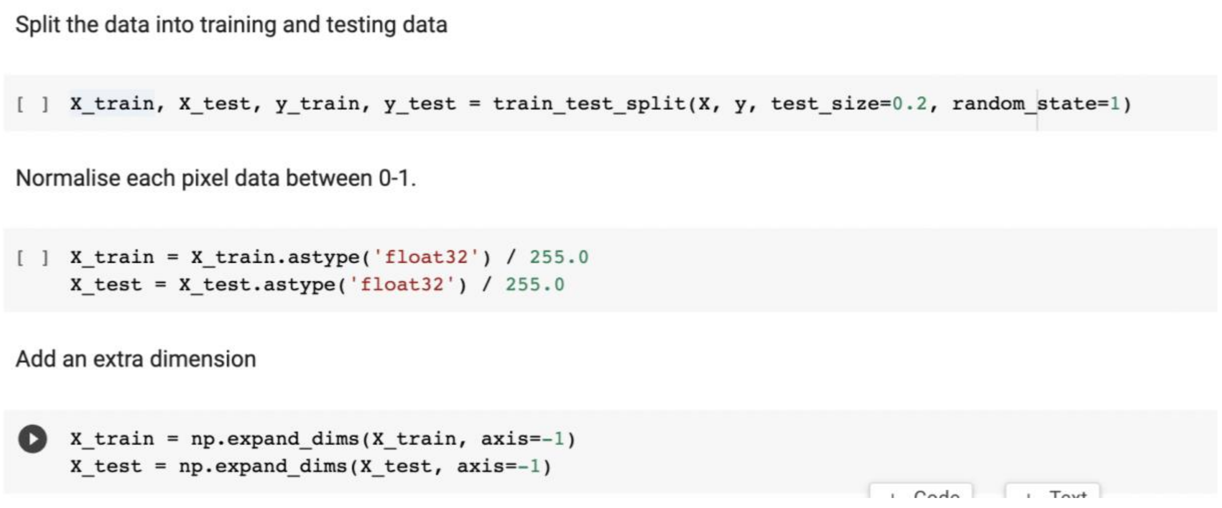
The figure above demonstrates how the data was imported using NumPy's 'load' method. Once both the positive and negative datasets (*pos_dat*, *neg_dat*) have been loaded into the environment, they can be labeled and concatenated to create one labelled dataset.

To label the data, the length of each dataset is taken using the 'len' method, therefore that number of zeros or ones are added to the y variable to correspond with the images in X.



Reshape the data to a 2D array

```
[ ] X = X.reshape((X.shape[0], 62, 47))
```


Split the data into training and testing data

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Normalise each pixel data between 0-1.

```
[ ] X_train = X_train.astype('float32') / 255.0
    X_test = X_test.astype('float32') / 255.0
```

Add an extra dimension

```
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)
```

Figure 4 - Reshaping, train test split, normalizing and adding an extra dimension.

When feeding the images into a CNN, they all must be the same size since the model needs to know how many pixels to expect. In this case, the images are resized to 64x47.
The data is then split into 4 variables using SciKitLearn's 'train_test_split' method: *X_train*, *X_test*, *y_train*, *y_test*. The advantage of train test split is that it shuffles the dataset using a random seed. In this case, I have chosen random seed 1. I've also chosen to opt for a 20% test split, meaning that 20% of the entire dataset will never be used to train the model; instead, it will be used to evaluate the model's performance.

After the split, the data is then normalised, meaning the pixel values are all scaled between 0-1. This is always done after the train test split to prevent data leakage and creating an incorrect evaluation score. Neural networks are sensitive to data normalisation; therefore, all data must be scaled.

Finally, an extra dimension is then added to the data. In this case, it is the colour channel. The model expects 3 types of features: width, height and colour. 1 is greyscale, 3 is RGB.

- ## Machine learning algorithms

When it comes to image recognition tasks like this one, CNNs have proved to be extremely effective compared to other machine learning models such as Support Vector Machines (*SVMs*).

Firstly, CNNs primary purpose is to analyse image data. The architecture of a CNN includes several convolutional layers that process the image and extract features such as edges and textures. These extracted features are then passed through pooling layers to reduce dimensionality, and finally through fully connected layers to make a prediction. On the other hand, SVMs aren't optimised to extract features from images, therefore requiring extensive feature engineering to achieve the desired result.

Secondly, CNNs can learn complex patterns with ease in comparison to SVMs. This is thanks to their ability to recognise hierarchical patterns. They work by learning to recognize more simple patterns, such as edges and corners, and then combing them into more complex faces – in this case, faces. On the other hand, SVMs rely soley on the features provided to them. Meaning extensive feature engineering is often required to achieve the desired result with an acceptable accuracy score on the test set.

Lastly, and perhaps most impressive about CNNs, they have the ability to learn special relationships between pixels in an image. In face detection, this is especially useful where the location and arrangement of facial features are important. In contrary to this, SVMs treat each feature independently and don't have the same level of spatial awareness.

Overall, using a CNN for facial detection seems like the sensible choice for powerful face detection on images within a timely and efficient manner. This has additionally lead to wide spread use of neural networks in many fields such as computer vision, medical imaging and robotics.

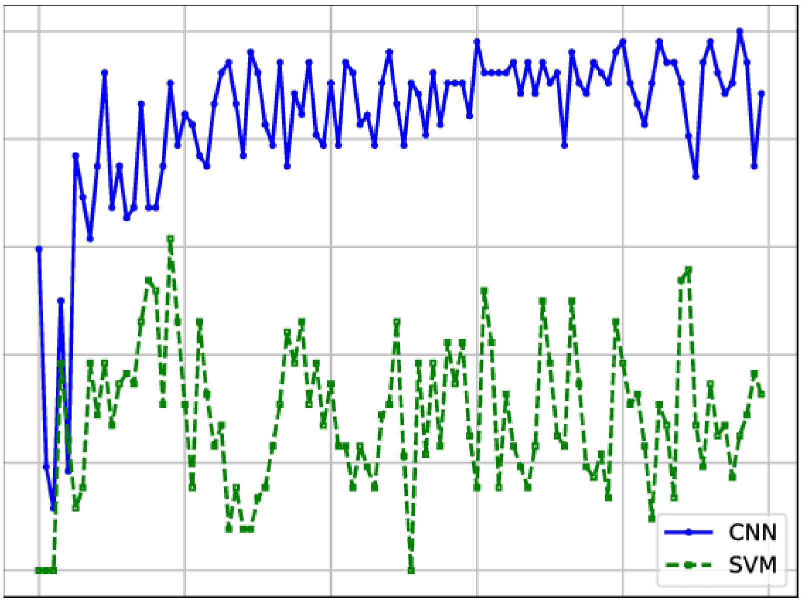
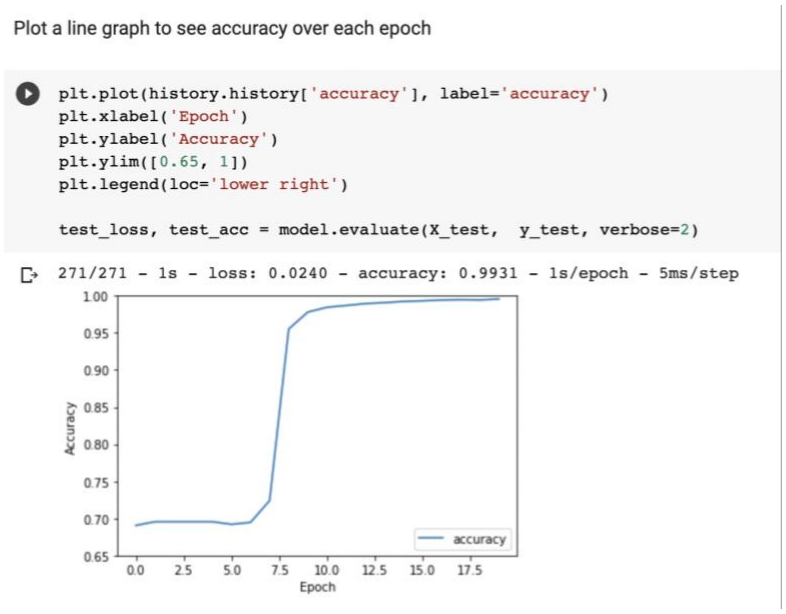


Figure 5 - Performance comparison (Accuracy) of CNN and SVM

Experiment Design/ Results Analysis

After training the model, each epoch was recorded into a variable called '*history*'. This can then be plotted using MatPlotLib to see how the accuracy increases over each epoch. An epoch is a training cycle whereby all the training data is passed through the model to improve the accuracy of the model. 20 epochs seemed to be the sweet spot whereby the model wasn't overfitted, but had a respectable enough accuracy to make the model functional for these circumstances.


Plot a line graph to see accuracy over each epoch

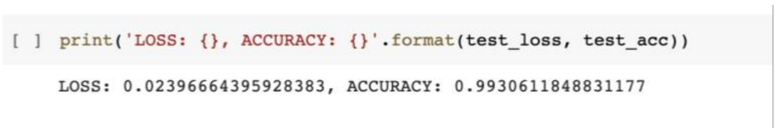
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.65, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(X_test,  y_test, verbose=2)
```

```
271/271 - 1s - loss: 0.0240 - accuracy: 0.9931 - 1s/epoch - 5ms/step
```

The graph suggests that after around epoch 5-6, the accuracy significantly increased until epoch 8. After this, it was a steady increase to the final 99.31% accuracy score.

This graph was primerally used to fine tune the amount of epochs needed for the amount of layer and neurons.



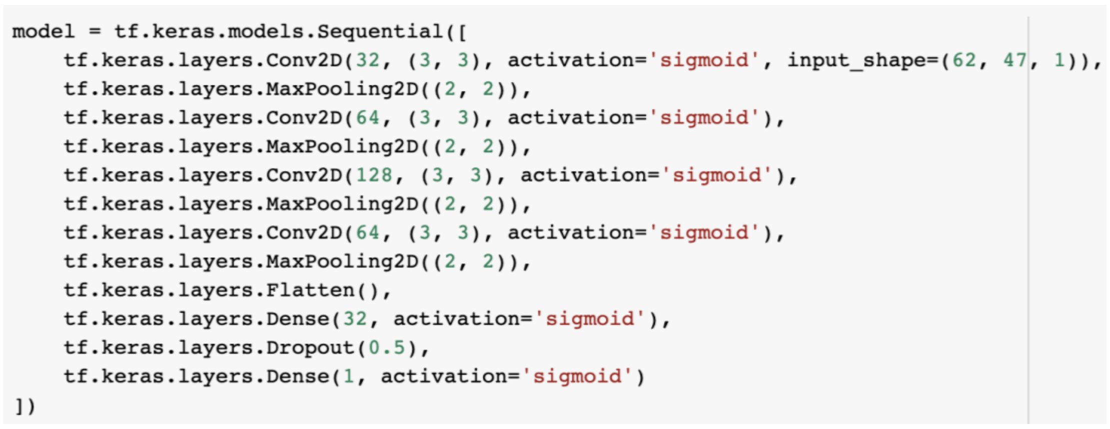
```
[ ] print('LOSS: {}, ACCURACY: {}'.format(test_loss, test_acc))

    LOSS: 0.02396664395928383, ACCURACY: 0.9930611848831177
```

As the dataset is small for machine learning standards, 20 epochs proved to be sufficient to get a respectable loss and accuracy. This is known as *fine tuning.*

Model fine tuning

Model hyperparameters are an important aspect of building CNN models. Tweaking these settings is known as *model fine tuning*. The choice of hyperparameters can significantly impact the performance of the CNN and tuning these hyperparameters offer after initial training is necessary to achieve the best accuracy score.



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='sigmoid', input_shape=(62, 47, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='sigmoid'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='sigmoid'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='sigmoid'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation='sigmoid'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Figure 6 - CNN model architecture used for this solution

Listed below are the hyperparameters that were tweaked to achieve a good accuracy and loss:

- **Number of epochs-** As mentioned above, an epoch is the number of times the model sees the training data. Seeing the data too much will result in an excellent accuracy score on the training data, but poor performance on new data after training – known as overfitting.
- **Number of filters-** The number of filters determines the number of feature maps generated by each convolutional layer in the CNN. A larger number of filters can lead to better feature extraction but can also increase the computational complexity of the model.
- **Kernel size-** The kernel size determines the size of the convolutional filter used in each convolutional layer. A larger kernel size can capture more complex features, but can also increase the computational complexity of the model. This increases accuracy on images where the face isn’t the main feature of the image.
- **Activation function-** Used to apply a non-linear transformation to the output of each layer. The function used for this specific model is Sigmoid as the ouputs need to be between 0 (Low confidence) and 1 (High confidence).