Deliverable 2

1. **Problem Statement**:

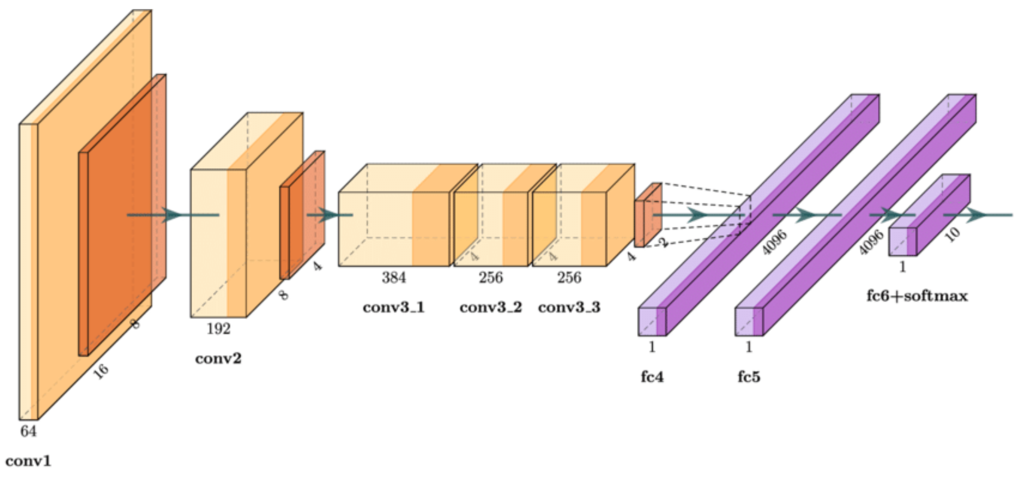
   Emotion Detection using Convolutional Neural Networks.
   We want to build a model that takes images as data and predicts the emotion that matches closely with the image provided.

2. **Data Preprocessing**:

   ➔ Dataset : https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer
   ➔ The dataset contains 35,685 examples of 48x48 pixel gray scale images of faces divided into train and test dataset. Images are categorized based on the emotion shown in the facial expressions (happiness, neutral, sadness, anger, surprise, disgust, fear).
   ➔ For the first test, we are just loading the dataset as is. However, we plan on using data augmentation (random image shifts) in further iterations.
   ➔ The data preprocessing methods we plan to use are: resizing, face detection, cropping, adding noises, and data normalization. Face detection alone can achieve very high accuracy and combined with other preprocessing techniques we hope to boost our accuracy further.

3. **Machine Learning Model**:

   For our model, we are using the *AlexNet DCNN*.
   A. The model has a total eight layers: the first five are convolutional layers, some of which are followed by max-pooling layers. The last three layers are fully connected layers. We are using Tensorflow and Keras libraries for our code.



   B. To avoid overfitting, we are using dropout as our regularization technique. The results obtained using the dropout model have low losses and the accuracy of the model is higher than other techniques like L1/L2 regularization.

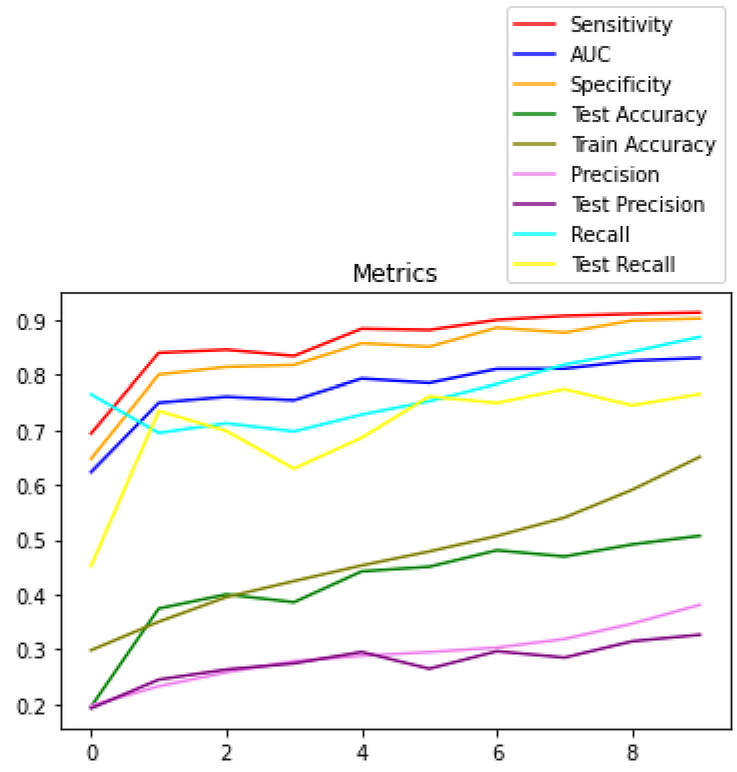
The parameters learning rate, batch size, training ratio, number of epochs, image preprocessing method and optimizer were chosen after testing with different values. We chose Adam as our optimizer and a batch size of 32. Our image dimensions were chosen to be 227x227

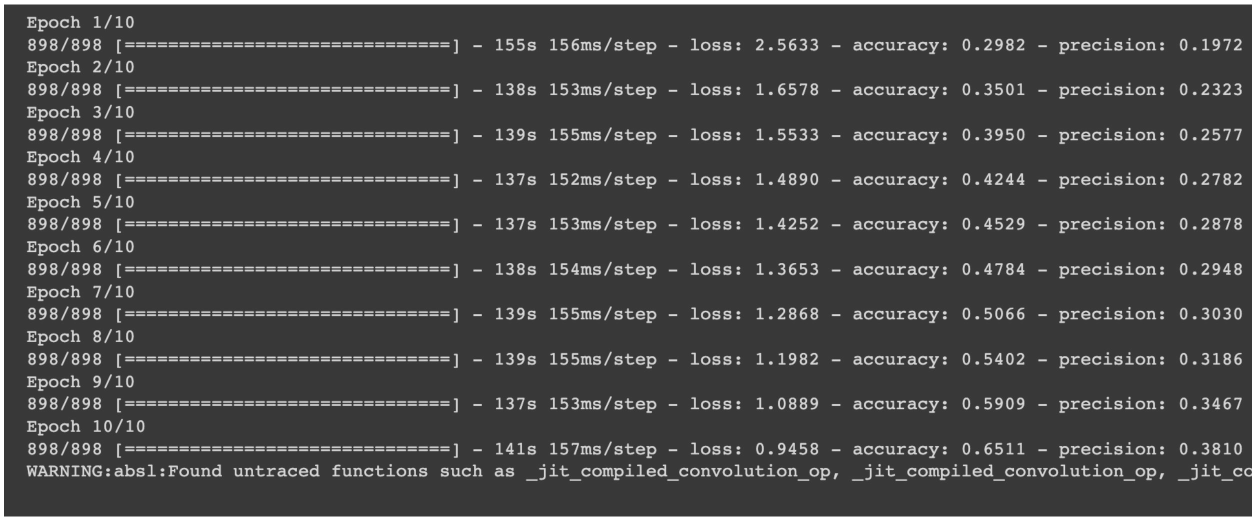
C. For the validation step, we used the basic train and test split. We used an 80/20 split for our model. Our model was overfitting and, hence, we used dropout as our regularization technique.
Training data = 28709 images belonging to 7 classes.
Testing data = 7178 images belonging to 7 classes.

D. The main challenge we faced is to reduce the training time. It was taking too long.

4. **Preliminary Results:**

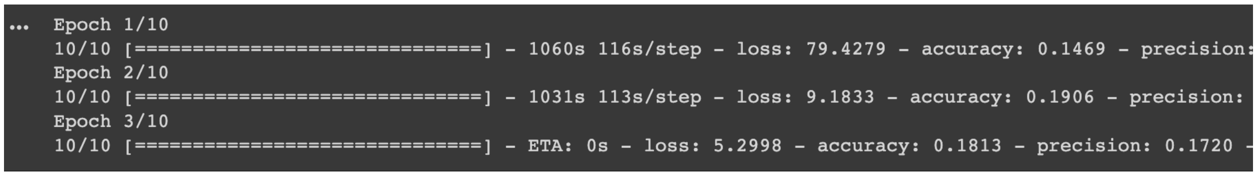
The metrics we are using are confusion matrix, accuracy, precision, recall, and loss. (see below for details)





```
Epoch 1/10
898/898 [==============================] - 155s 156ms/step - loss: 2.5633 - accuracy: 0.2982 - precision: 0.1972
Epoch 2/10
898/898 [==============================] - 138s 153ms/step - loss: 1.6578 - accuracy: 0.3501 - precision: 0.2323
Epoch 3/10
898/898 [==============================] - 139s 155ms/step - loss: 1.5533 - accuracy: 0.3950 - precision: 0.2577
Epoch 4/10
898/898 [==============================] - 137s 152ms/step - loss: 1.4890 - accuracy: 0.4244 - precision: 0.2782
Epoch 5/10
898/898 [==============================] - 137s 153ms/step - loss: 1.4252 - accuracy: 0.4529 - precision: 0.2878
Epoch 6/10
898/898 [==============================] - 138s 154ms/step - loss: 1.3653 - accuracy: 0.4784 - precision: 0.2948
Epoch 7/10
898/898 [==============================] - 139s 155ms/step - loss: 1.2868 - accuracy: 0.5066 - precision: 0.3030
Epoch 8/10
898/898 [==============================] - 139s 155ms/step - loss: 1.1982 - accuracy: 0.5402 - precision: 0.3186
Epoch 9/10
898/898 [==============================] - 137s 153ms/step - loss: 1.0889 - accuracy: 0.5909 - precision: 0.3467
Epoch 10/10
898/898 [==============================] - 141s 157ms/step - loss: 0.9458 - accuracy: 0.6511 - precision: 0.3810
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_co
```

5. **Next Steps:**
   Problems
   1. We tried to train the model but it was taking an unbelievably long time to train. The model was stuck at Epoch 1/10. At first we were using number_of_steps_per_epoch = image_count/batch_size which gave us 898 steps per epoch. For that each epoch had an ETA of 6 hours.
   2. Next we tried to reduce the steps to see if it would reduce the ETA



```
...  Epoch 1/10
     10/10 [==============================] - 1060s 116s/step - loss: 79.4279 - accuracy: 0.1469 - precision:
     Epoch 2/10
     10/10 [==============================] - 1031s 113s/step - loss: 9.1833 - accuracy: 0.1906 - precision:
     Epoch 3/10
     10/10 [==============================] - ETA: 0s - loss: 5.2998 - accuracy: 0.1813 - precision: 0.1720 -
```

   It still gave us a very high ETA per Epoch for 10 steps. So for 897 steps per epoch, it would take a couple of days to train.

   Solution
   We realized that GPU wasn't enabled in our notebook settings which gave us a very high runtime. After enabling it we were able to train our model in 30-45 minutes.

   Link to our colab file:
   https://colab.research.google.com/drive/1O2rBfc86MZi2a3Zumk0x9ElZujRpBgJm#scrollTo=RkEpk7IjUntj