Homework-3

CSCE 633: Machine Learning

Manav Gurumoorthy

830000011

Introduction:

In this HW, we looked at the use of Artificial Neural Networks to perform image recognition.
The dataset consists of labelled images of humans depicting different facial expression. The recgonition task was to accurately discern the facial expression being depicted in the image.

The dataset was consisted of several 48X48 grayscale images. i.e. each image was considered as a vector of length 2304, with each element representing the intensity of that pixel.

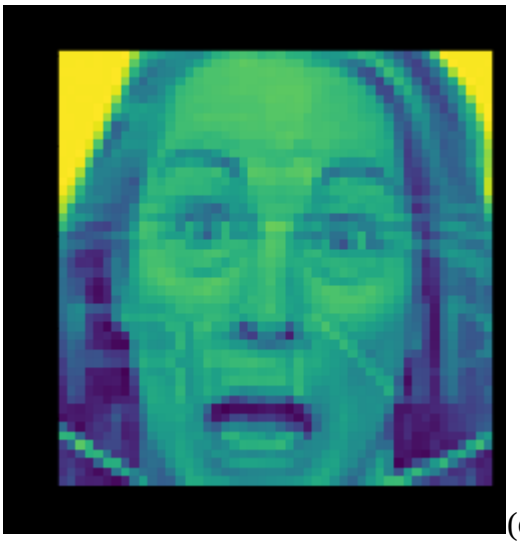
We look at two main approaches here

1. The use of a Feed Forward Network
2. Use of a Convolutional Neural Network.

While these are the two broad approaches in this study, each method had their own nuances and required careful manipulation of hyperparameters.

Data Exploration:

In this part, we explored the data that we were going to model. The first step was to plot the images from the data that we had. This was done using inbuilt functions in python.

 (one such example) more in the code section.

We also looked at how many exhibits from each of the labels were present in the training dataset.

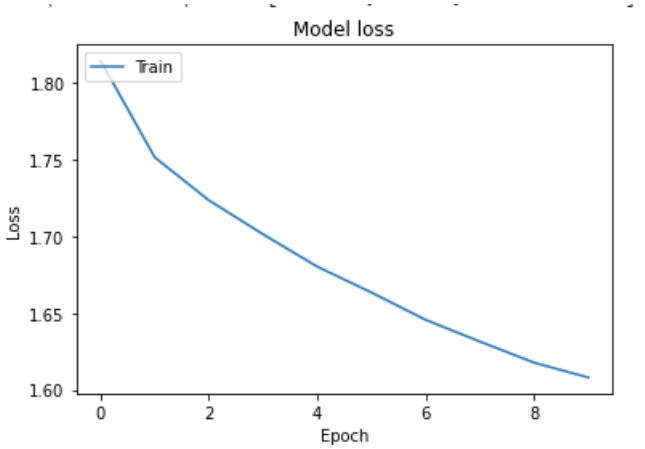
```
Angry:    3995
Disgust:   436
Fear:     4097
Happy:    7215
Sad:      4830
Surprise: 3171
Neutral:  4965

_______
Total:   28709
```

Feedforward Neural Network:

In this section we construct a FNN using the keras library and test the performance of the model with the validation data by playing around with hyperparameter values. I liked the hyperopt library, so I used this to find optimal hyperparameter values. My search space consisted of the following parameters:

```
space = {
    # The activation_fn choices:
    'activation_fn':hp.choice('activation_fn',['relu']),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.5),
    # Choice of optimizer
    'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
}
```

We also plotted the loss as a function of the number of iterations (epochs?). One such example is show below. The rest are shown as a part of the code section.



Some stats that are printed with every change in hyperparameters: (This was done using Prafulla's code)

```
Total Training Time is (s):
7.4364213943481445
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.25336800158266903, 'optimizer': 'sgd'}
Accuracy:
0.39091668988991396
--------------------------------------------------
  4%|▌          | 1/25 [00:07<03:11,  8.00s/it, best loss: -0.39091668988991396]
                    Model loss
```

The best hyperparams were then used and the model was tested on the test set.

```
================================================================================
Best Hyperparameters {'activation_fn': 0, 'dropout_prob': 0.2476540471215125, 'optimizer': 0}
3589/3589 [==============================] - 0s 49us/step
==================================
Test Accuracy:  0.45528002229863535
```

In an earlier test I also tried 3 different activation functions namely ReLu, Sigmoid and tanh, but since the other 2 gave me extremely poor results I defaulted to just using ReLu.

Convolution Neural Networks:

Like the FNN case, we create a CNN model using the keras library APIs. Similar to the FNN case, we try to optimise the model using the validation set and different combinations of hyperparameters to optimise the model.

Firstly I used a simple for loop which used the following params and trained 10 different models, the best hyperparameter pair from this was used on the test set.

```
1 dropout_range = [0.1,0.2,0.3,0.4,0.5]
2 kernels = [2,3]
```

```
Best Feature Pair is: 3 and 0.3
Best performance from training: 0.45082195598489894
Kernel Size: 3
Dropout: 0.3
```

```
3589/3589 [==============================] - 2s 522us/step
Accuracy on Test samples: 0.2449150181191982
```

Unfortunately, this was not yielding very good results because I was only using a small set of values. So once again I used HyperOpt to find a better set of hyperparameters.

```
----------------------------------------------------
100%|██████████| 25/25 [12:46<00:00, 30.65s/it, best loss: -0.5937587071939842]
==================================
Best Hyperparameters {'conv_kernel_size': 1, 'dropout_prob': 0.2890305052408365, 'optimizer': 0}
3589/3589 [==============================] - 0s 102us/step
==================================
Test Accuracy:  0.5851212036862085
```

Fine Tuning:

I was a little unclear about this part so I have done 2 things.
The first thing is to train a model on the MNIST dataset and modify the FER dataset suitably to work with the trained model. Doing this got an accuracy of 40%

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_31 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 64)        18496
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_32 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_32 (Dense)             (None, 10)                5130
=================================================================
Total params: 594,922
Trainable params: 594,922
Non-trainable params: 0
```

Then adding another layer for the outputs of FER,

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_31 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 64)        18496
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_32 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_32 (Dense)             (None, 10)                5130
_________________________________________________________________
dense_35 (Dense)             (None, 7)                 77
=================================================================
Total params: 659,914
Trainable params: 594,922
Non-trainable params: 64,992
```

The final model accuracy:

```
3589/3589 [==============================] - 1s 232us/step
Accuracy on Test samples: 0.4093062134299248
```

The second thing was to create a combined model of FER and MNIST.

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 28, 28, 1)    0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 48, 48, 1)    0
__________________________________________________________________________________________________
sequential_2 (Sequential)       multiple             98272       input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
MNIST (Dense)                   (None, 10)           81930       sequential_2[1][0]
__________________________________________________________________________________________________
FER (Dense)                     (None, 7)            290311      sequential_2[2][0]
==================================================================================================
Total params: 470,513
Trainable params: 470,513
Non-trainable params: 0
```

This yielded the following results,

```
===
MNIST Accuracy: 0.9875731395127282
FER Accuracy: 0.8625562360739103
```

Data Augmentation:

In this section we look at different types of data augmentation using the ImageDataGenerator from keras. I tried Standardisation, ZCA Whitening and Random Rotation, the results are shown below in the same order.

```
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.21040384851779753, 'optimizer': 0}
3589/3589 [==============================] - 0s 92us/step
==================================
Test Accuracy:  0.6032320980816108
```

```
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.14865966844802445, 'optimizer': 0}
3589/3589 [==============================] - 0s 112us/step
==================================
Test Accuracy:  0.5918083031568131
```

```
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.19430122651964926, 'optimizer': 0}
3589/3589 [==============================] - 0s 123us/step
==================================
Test Accuracy:  0.5770409584925612
```

Feature Design:

In this section I tried implementing HOG on the images because it seemed like it would give good results for the task we had intended. Unfortunately, I guess either there is an error in my implementation or HOG is not a method that is suitable for this sort of task. But I attribute it to the former rather than the latter.

```
--------------------------------------------------
Total Training Time is (s):
26.25912308692932
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.10813003008178762, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
21.570691347122192
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.27555616085888457, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
100%|██████████| 25/25 [10:50<00:00, 26.00s/it, best loss: -0.24937308442878273]
=================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.15049653462509077, 'optimizer': 1}
3589/3589 [==============================] - 0s 87us/step
=================================
Test Accuracy:  0.2449150181191982
```

APPENDIX A: COLAB Workbook