Omer Moussa

900171920

PMDL Assignment #2

Methodology and preprocessing:

- Download the dataset, read each image and resize it to 32 *32* 3, 64 *64* 3, 64 *64* 3 to make them all of the same shape. Link to the data: https://drive.google.com/drive/u/1/folders/1Gffr71bnA-GZToa9rlfoJcc3PRFGw5hB
- Constructed the test set and the training set
- Shuffled the data
- All random values generated using np.random.seed(0)
- Made the ANN class with functions train(), backward_pass(), forward_pass, predict(), save_model, etc. that can be used to classify any input X using ANN with specified num layers and num nodes per layers.
- Ran Hyperparameters optimization to find best lambda, learning rate, architecture of the network.
- Predicted on the test set using the best hyperparameters, and got the accuracy per class and the average accuracy.

1. All implementations and code can be found in the attached notebook.
2. Data Preprocessing:

- resize it to 32 *32* 3, 64 *64* 3, 128 *128* 3 to different sizes & find the best for training.
- Normalized all the data by dividing by 255. To make pixels between them between 0&1
- Shuffled the data.

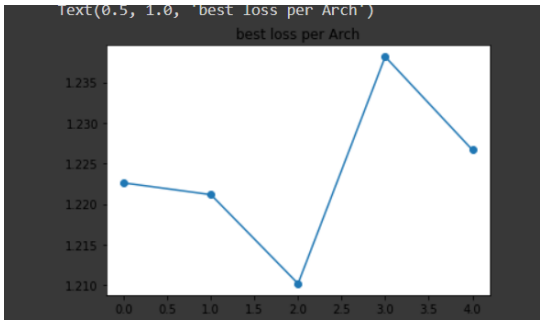
3. All hyperparameters were searched for using validation loss as the judging metric.

3.1. Image sizes (tried data image sizes 32 , 64, 128). The best one was 32 by 32 by 3 images.

3.2. Choosing Architecture:

Tried different num layers and plotted the loss per architecture to find the Best architecture.

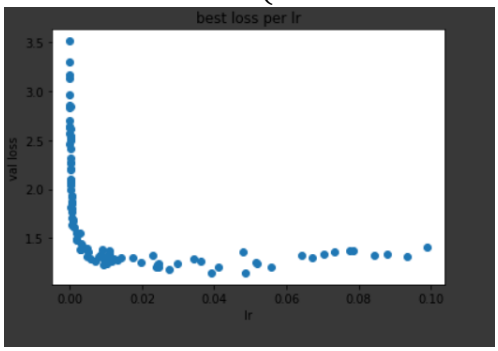
```
archs=[]
archs.append([Xt.shape[1],256,5]) # 1 layers Architecture

archs.append([Xt.shape[1],512,256,5]) # 2 hiddenlayers Architecture
archs.append([Xt.shape[1],256,256,5]) # 2 hiddenlayers Architecture

archs.append([Xt.shape[1],256,256,128,5]) # 3 hidden layers Architecture
archs.append([Xt.shape[1],512,256,128,5]) # 3 hidden layers Architecture
```



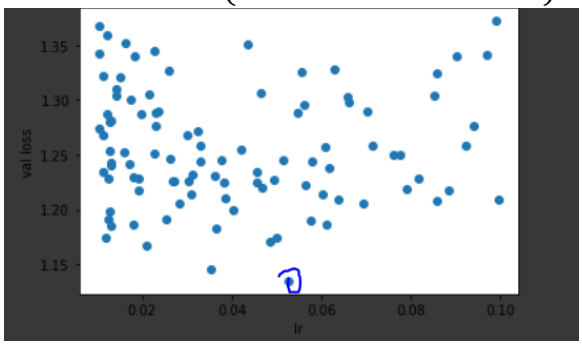

Best found architecture is with 2 hidden layers 256 nodes each.

3.2. Best Learning rate:

- Choose lr limits (0.1 – .0001)
- Run coarse search (best lr around 10^-2)

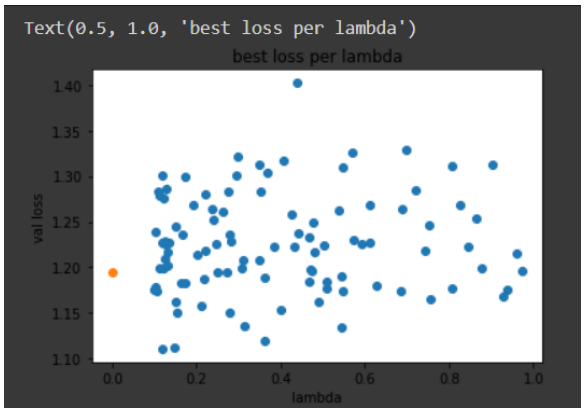



- Run fine search (best lr is around 0.055)

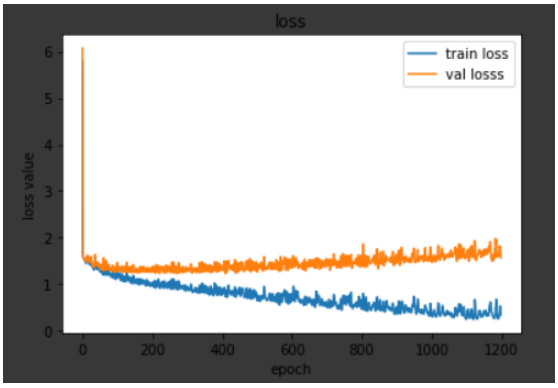



3.3. Choosing Lambda of L2 reg.

- Tried limits of lambda (> 10^-1)
- Run search scale from 1 to 0.1
- Best one was around 0.1

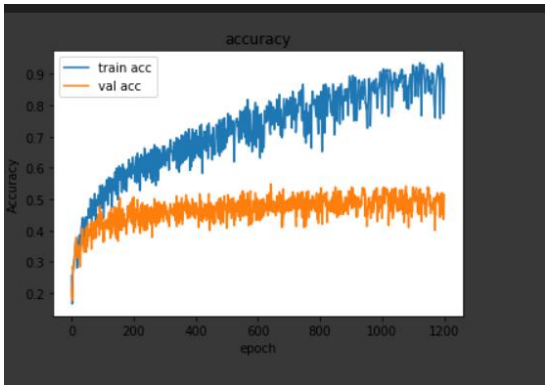



4. Stopping the training used validation accuracy as the judging metric.



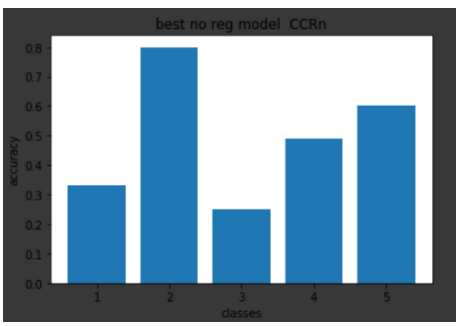

4.2. Loss plots

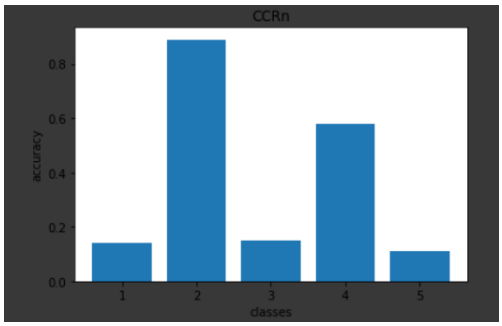
4.3. Accuracy plots




4.4. Stopping the training: the training was stopped when the validation accuracy started to decrease (with patience for some epochs), and num of epochs was decided using this technique. So the graphs above were plotted against the appropriate number of epochs that validation accuracies started to decrease or saturate after (with some patience of course.)

5. The CCRn of the NN (left )vs CCRn of KNN (right). The Classification rate for classes (3 and 1 ) are the lowest of all (probably are the hardest to predict or they are similar somehow to other classes.).






6. Below is the report of ACCR for two models (one with L2 reg ld=0.1 and the other with no regularization). The heights ACCR achieved was 54.2% which was the model with no L2 reg.

```
Model Arch and parameters (lr, lambda):  [3072, 256, 256, 5] 0.055 0
model Average Accuracy  0.542
model  Accuracy on training set  0.9015772870662461
```

```
Best reg Model Arch and parameters (lr, lambda):  [3072, 256, 256, 5] 0.055 0.1
Best reg model Average Accuracy  0.538
```

**Note: both best performing models are attached and can be loaded using ANN.load_model() to reproduce the results on the data. The link to the data is provided above.**

6.2. I tried to compile a TF model on the same data with the same architecture to verify the outputs. I then evaluated it on the same test data. It actually gave less but close accuracy.

```
1  ty=keras.utils.to_categorical(cp.asnumpy(y_test).reshape(-1,1))
2  ty=ty[:,1:]
3  print(" TF Avg Acc", model.evaluate(cp.asnumpy(X_test),ty))
```

```
16/16 [==============================] - 0s 2ms/step - loss: 1.9222 - accuracy: 0.4900
 TF Avg Acc [1.9221936464309692, 0.49000000953674316]
```