

CMPT 762

Computer Vision

Assignment 2

Gokul Mohanarangan
gma56@sfu.ca

301436162

Free-Late day: 3 late days given exclusively for
Assignment 2 are intended to be used here.

1 free late day from the pool of 5 free late days is
also used here.

The name on Kaggle : Naruzu
Best accuracy : 0.68500

BASENET ARCHITECTURE:

Layer number	Layer type	Kernel size (For conv layers)	Input Output Dimensions	Input Output Channels (For conv layers)
1	Conv2d	3	32 32	3 64
2	BatchNorm2d		32 32	
3	Relu		32 32	
4	Conv2d	3	32 32	64 128
5	BatchNorm2d		32 32	
6	Relu		32 32	
7	Pool		32 16	
8	Conv2d	3	16 16	128 128
9	BatchNorm2d		16 16	
10	Relu		16 16	
11	Conv2d	3	16 16	128 128
12	BatchNorm2d		16 16	
13	Relu		16 16	
14	Conv2d	3	16 16	128 256
15	BatchNorm2d		16 16	
16	Relu		16 16	
17	Pool		16 8	
18	Conv2d	3	8 8	256 512
19	BatchNorm2d		8 8	
20	Relu		8 8	
21	Conv2d	3	8 8	512 512
22	BatchNorm2d		8 8	
23	Relu		8 8	
24	Conv2d	3	8 8	512 512
25	BatchNorm2d		8 8	
26	Relu		8 8	
27	Pool		8 4	
28	Conv2d	3	4 4	512 1024
29	BatchNorm2d		4 4	
30	Relu		4 4	
31	Conv2d	3	4 4	1024 1024
32	BatchNorm2d		4 4	
33	Relu		4 4	
34	Conv2d	3	4 4	1024 1024
35	BatchNorm2d		4 4	
36	Relu		4 4	

37	Linear (FC)		1024 * 4 * 4 100	
38	BatchNorm1d		100	
39	Relu		100	
40	Linear (FC)		100 100	
41	BatchNorm1d		100	

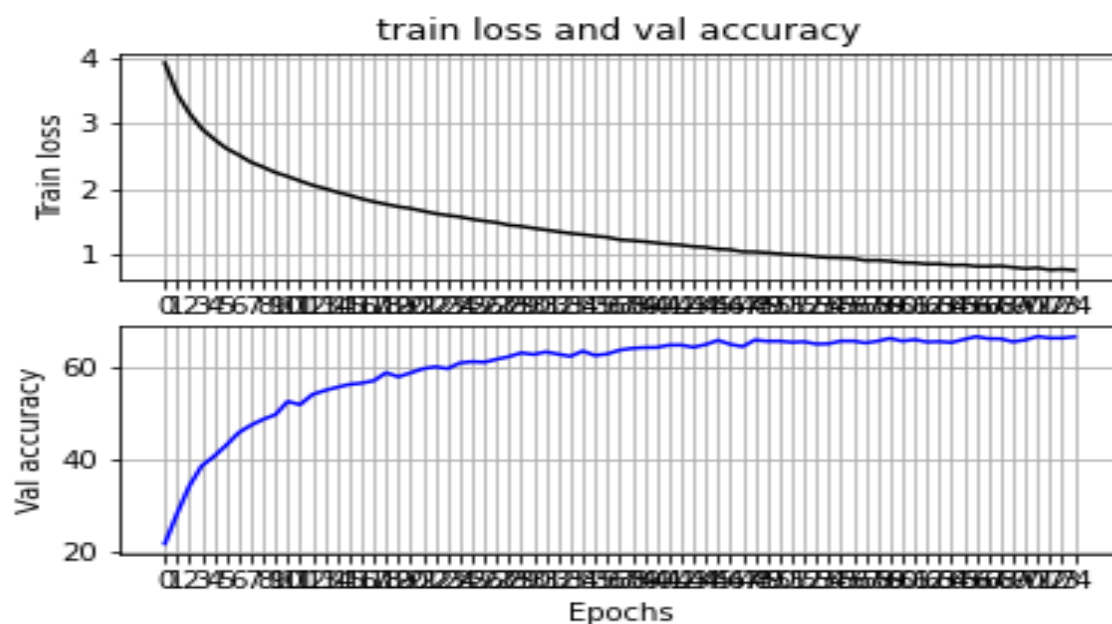
MODEL OVERVIEW:

To decrease the variance of the model, many data augmentations have been performed as part of the training process. Example : Random resized crop, random horizontal flip, random rotation have been included in the pipeline. The required mean and std values for the entire dataset has been calculated and all data has been normalized before model building. The model has 2 main components to it, the CONV-RELU-POOL and the linear fully connected layers. Every CONV layer is also followed by a BatchNorm layer for normalization and helping the training better. The conv layer has been padded to not reduce dimensions at every step and this enables the model to be deeper. Not every conv layer is followed by a pool layer to prevent loss of information, especially in the deeper layers. Hyperparameters like learning rate, epochs, momentum, and batch sizes have been experimented on. All kernel sizes are fixed at 3 and at every conv layer, the number of channels are increased, usually by a factor of 2. Some residual layers have been mixed in which have the same input and output channels but the previous layer being added to the next layer. This enables the model to 'retain' some information from the previous layers.

Best validation accuracy : 68%

Best testing accuracy on Kaggle : 0.68500

Loss plot and validation accuracy plot:



Ablation study:

Model edits :

- 1) Add residual layers. Since the model does not have a means to “remember” the previous layers, adding residual layers where the output of a certain layer is also added with the a previous layer output mimics the retention capabilities of the model and increases model accuracy.
- 2) Adding more conv layers. To create a deeper network to better capture the features, several new conv layers were added. Some with same input/output channels to enable residual layers.
- 3) It also means that conv layers have to be padded so that we don't lose the dimensions too quickly, thus enabling us to create deeper networks. The padding has been set to 1 from all conv layers.

Before making these series of changes, the accuracy was validation accuracy was 61% and the test accuracy was 0.615. After making the series of changes, the validation accuracy was 68% and the test accuracy was 0.68500. The test accuracies can be verified on Kaggle. To address memory needs, the batch size was also reduced to 32 but it did not significantly affect the accuracies that much, but was kept for ease of computation when memory was occupied in the GPU RAM.

PART 2 :

HYPERPARAMETERS :

Fine tune whole network :

Batch size : 16

Epochs : 50

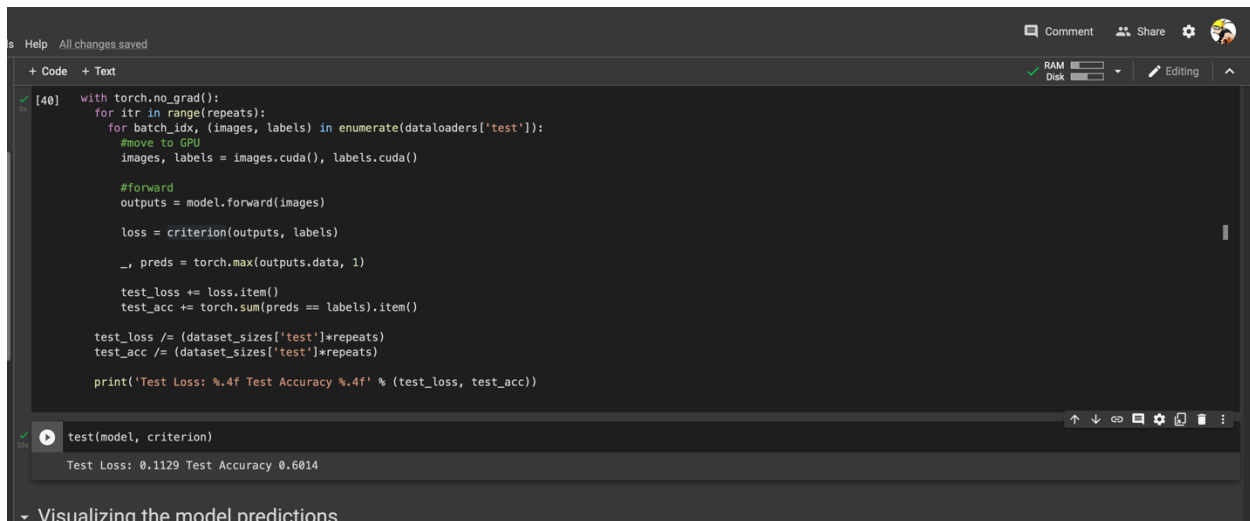
Learning rate : 0.001

Momentum : 0.9

Restnet last only : False

Training accuracy : 0.8097

Test accuracy : 0.6014



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains a function `test(model, criterion)` that iterates over a dataset to calculate test loss and accuracy. The output cell shows the result of the function call.

```
[40] with torch.no_grad():
    for itr in range(repeats):
        for batch_idx, (images, labels) in enumerate(dataloaders['test']):
            #move to GPU
            images, labels = images.cuda(), labels.cuda()

            #forward
            outputs = model.forward(images)

            loss = criterion(outputs, labels)

            _, preds = torch.max(outputs.data, 1)

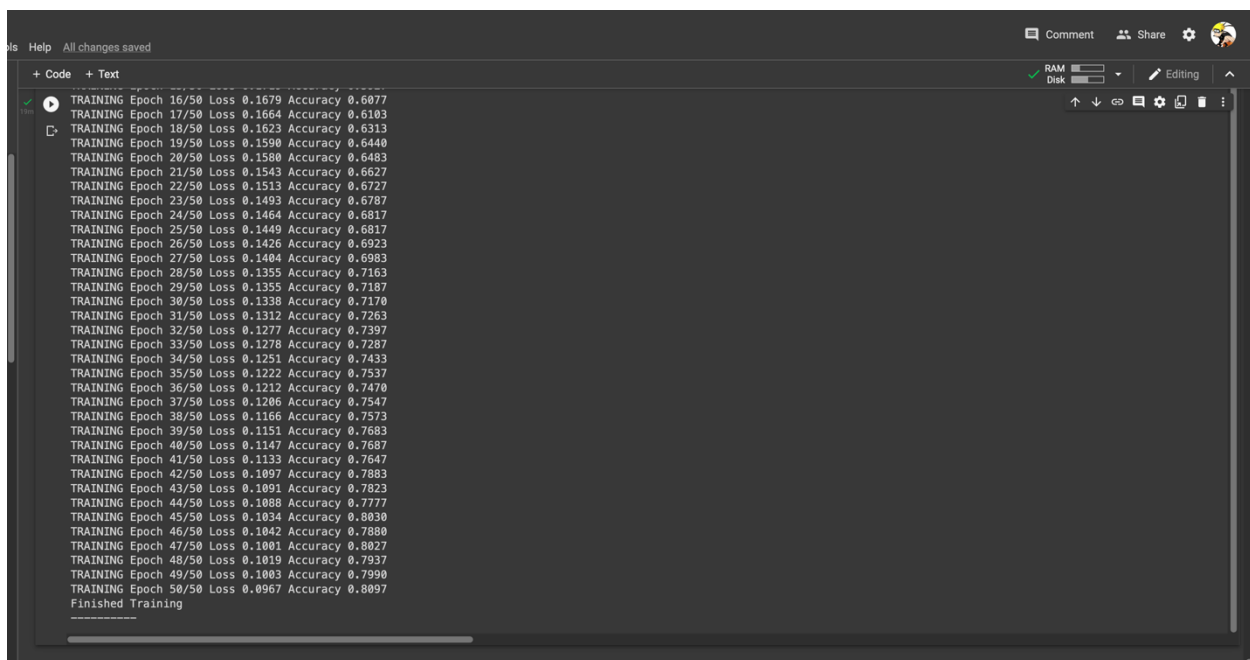
            test_loss += loss.item()
            test_acc += torch.sum(preds == labels).item()

        test_loss /= (dataset_sizes['test']*repeats)
        test_acc /= (dataset_sizes['test']*repeats)

    print('Test Loss: %.4f Test Accuracy %.4f' % (test_loss, test_acc))
```

```
test(model, criterion)
Test Loss: 0.1129 Test Accuracy 0.6014
```

Visualizing the model predictions



The screenshot shows a Jupyter Notebook interface with a code cell containing a list of training progress logs. The logs show the loss and accuracy for each epoch from 16 to 50.

```
TRAINING Epoch 16/50 Loss 0.1679 Accuracy 0.6077
TRAINING Epoch 17/50 Loss 0.1664 Accuracy 0.6103
TRAINING Epoch 18/50 Loss 0.1623 Accuracy 0.6313
TRAINING Epoch 19/50 Loss 0.1590 Accuracy 0.6440
TRAINING Epoch 20/50 Loss 0.1580 Accuracy 0.6483
TRAINING Epoch 21/50 Loss 0.1543 Accuracy 0.6627
TRAINING Epoch 22/50 Loss 0.1513 Accuracy 0.6727
TRAINING Epoch 23/50 Loss 0.1493 Accuracy 0.6787
TRAINING Epoch 24/50 Loss 0.1464 Accuracy 0.6817
TRAINING Epoch 25/50 Loss 0.1449 Accuracy 0.6817
TRAINING Epoch 26/50 Loss 0.1426 Accuracy 0.6923
TRAINING Epoch 27/50 Loss 0.1404 Accuracy 0.6983
TRAINING Epoch 28/50 Loss 0.1355 Accuracy 0.7163
TRAINING Epoch 29/50 Loss 0.1355 Accuracy 0.7187
TRAINING Epoch 30/50 Loss 0.1338 Accuracy 0.7170
TRAINING Epoch 31/50 Loss 0.1312 Accuracy 0.7263
TRAINING Epoch 32/50 Loss 0.1277 Accuracy 0.7397
TRAINING Epoch 33/50 Loss 0.1278 Accuracy 0.7287
TRAINING Epoch 34/50 Loss 0.1251 Accuracy 0.7433
TRAINING Epoch 35/50 Loss 0.1222 Accuracy 0.7537
TRAINING Epoch 36/50 Loss 0.1212 Accuracy 0.7470
TRAINING Epoch 37/50 Loss 0.1206 Accuracy 0.7547
TRAINING Epoch 38/50 Loss 0.1166 Accuracy 0.7573
TRAINING Epoch 39/50 Loss 0.1151 Accuracy 0.7683
TRAINING Epoch 40/50 Loss 0.1147 Accuracy 0.7687
TRAINING Epoch 41/50 Loss 0.1133 Accuracy 0.7647
TRAINING Epoch 42/50 Loss 0.1097 Accuracy 0.7883
TRAINING Epoch 43/50 Loss 0.1091 Accuracy 0.7823
TRAINING Epoch 44/50 Loss 0.1088 Accuracy 0.7777
TRAINING Epoch 45/50 Loss 0.1034 Accuracy 0.8030
TRAINING Epoch 46/50 Loss 0.1042 Accuracy 0.7880
TRAINING Epoch 47/50 Loss 0.1001 Accuracy 0.8027
TRAINING Epoch 48/50 Loss 0.1019 Accuracy 0.7937
TRAINING Epoch 49/50 Loss 0.1003 Accuracy 0.7990
TRAINING Epoch 50/50 Loss 0.0967 Accuracy 0.8097
Finished Training
```

[The GPU quota ended and I had to switch to a local machine for fixed feature extraction part.]

Fixed feature extraction:

Batch size : 16

Epochs : 50

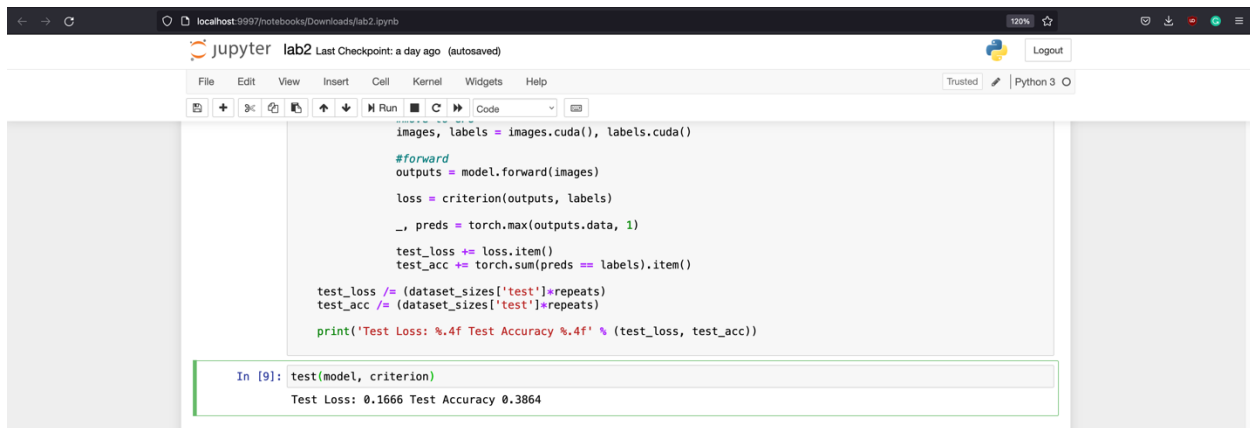
Learning rate : 0.001

Momentum : 0.9

Restnet last only : True

Training accuracy : .05417

Test accuracy : 0.3864



```
images, labels = images.cuda(), labels.cuda()

#forward
outputs = model.forward(images)

loss = criterion(outputs, labels)

_, preds = torch.max(outputs.data, 1)

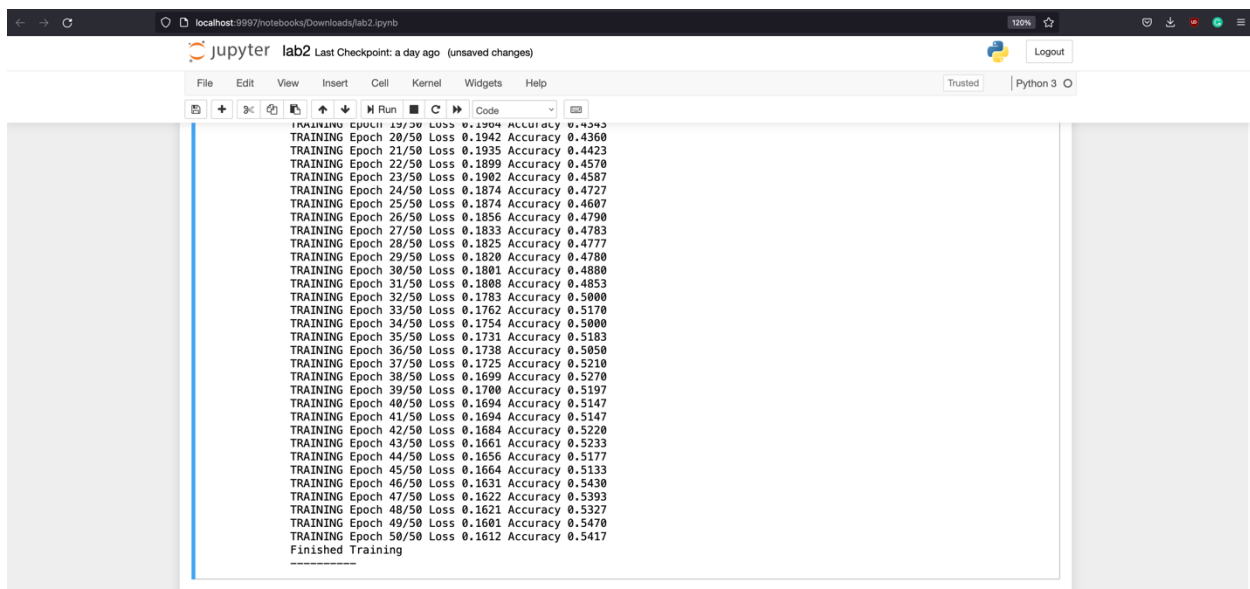
test_loss += loss.item()
test_acc += torch.sum(preds == labels).item()

test_loss /= (dataset_sizes['test']*repeats)
test_acc /= (dataset_sizes['test']*repeats)

print('Test Loss: %.4f Test Accuracy %.4f' % (test_loss, test_acc))
```

In [9]: test(model, criterion)

Test Loss: 0.1666 Test Accuracy 0.3864



```
TRAINING Epoch 19/50 Loss 0.1904 Accuracy 0.4343
TRAINING Epoch 20/50 Loss 0.1942 Accuracy 0.4360
TRAINING Epoch 21/50 Loss 0.1935 Accuracy 0.4423
TRAINING Epoch 22/50 Loss 0.1899 Accuracy 0.4570
TRAINING Epoch 23/50 Loss 0.1902 Accuracy 0.4587
TRAINING Epoch 24/50 Loss 0.1874 Accuracy 0.4727
TRAINING Epoch 25/50 Loss 0.1874 Accuracy 0.4607
TRAINING Epoch 26/50 Loss 0.1856 Accuracy 0.4790
TRAINING Epoch 27/50 Loss 0.1833 Accuracy 0.4783
TRAINING Epoch 28/50 Loss 0.1825 Accuracy 0.4777
TRAINING Epoch 29/50 Loss 0.1820 Accuracy 0.4780
TRAINING Epoch 30/50 Loss 0.1801 Accuracy 0.4880
TRAINING Epoch 31/50 Loss 0.1808 Accuracy 0.4853
TRAINING Epoch 32/50 Loss 0.1783 Accuracy 0.5000
TRAINING Epoch 33/50 Loss 0.1762 Accuracy 0.5170
TRAINING Epoch 34/50 Loss 0.1754 Accuracy 0.5000
TRAINING Epoch 35/50 Loss 0.1731 Accuracy 0.5183
TRAINING Epoch 36/50 Loss 0.1738 Accuracy 0.5050
TRAINING Epoch 37/50 Loss 0.1725 Accuracy 0.5210
TRAINING Epoch 38/50 Loss 0.1699 Accuracy 0.5270
TRAINING Epoch 39/50 Loss 0.1700 Accuracy 0.5197
TRAINING Epoch 40/50 Loss 0.1694 Accuracy 0.5147
TRAINING Epoch 41/50 Loss 0.1694 Accuracy 0.5147
TRAINING Epoch 42/50 Loss 0.1684 Accuracy 0.5220
TRAINING Epoch 43/50 Loss 0.1661 Accuracy 0.5233
TRAINING Epoch 44/50 Loss 0.1656 Accuracy 0.5177
TRAINING Epoch 45/50 Loss 0.1664 Accuracy 0.5133
TRAINING Epoch 46/50 Loss 0.1631 Accuracy 0.5430
TRAINING Epoch 47/50 Loss 0.1622 Accuracy 0.5393
TRAINING Epoch 48/50 Loss 0.1621 Accuracy 0.5327
TRAINING Epoch 49/50 Loss 0.1601 Accuracy 0.5470
TRAINING Epoch 50/50 Loss 0.1612 Accuracy 0.5417
Finished Training
```