Name: Krishnakumar Chavan UBIT: kchavan

Name: Nikhil Gupta

UBIT: ngupta22

**Assignment 1:**

# Checkpoint

**PART 1:**

1. Here, we have implemented VGG13 version B which consists of 13 weight layers (10 convolutional layers and 3 fully connected layers). Other than that, it also consists of Maxpool and Softmax layer which is applied at the end for multiclass classification.

You can find the whole summary of the model below:

==========================================================================================

Layer (type:depth-idx) Output Shape Param #

==========================================================================================

VGG13 [32, 3] --

├─Sequential: 1-1 [32, 64, 32, 32] --

│ └─Conv2d: 2-1 [32, 64, 64, 64] 1,792

│ └─ReLU: 2-2 [32, 64, 64, 64] --

│ └─Conv2d: 2-3 [32, 64, 64, 64] 36,928

│ └─ReLU: 2-4 [32, 64, 64, 64] --

│ └─MaxPool2d: 2-5 [32, 64, 32, 32] --

├─Sequential: 1-2 [32, 128, 16, 16] --

│ └─Conv2d: 2-6 [32, 128, 32, 32] 73,856

│ └─ReLU: 2-7 [32, 128, 32, 32] --

│ └─Conv2d: 2-8 [32, 128, 32, 32] 147,584

│ └─ReLU: 2-9 [32, 128, 32, 32] --

│ └─MaxPool2d: 2-10 [32, 128, 16, 16] --

├─Sequential: 1-3 [32, 256, 8, 8] --

│ └─Conv2d: 2-11 [32, 256, 16, 16] 295,168

│ └─ReLU: 2-12 [32, 256, 16, 16] --

│ └─Conv2d: 2-13 [32, 256, 16, 16] 590,080

│ └─ReLU: 2-14 [32, 256, 16, 16] --

│ └─MaxPool2d: 2-15 [32, 256, 8, 8] --

├─Sequential: 1-4 [32, 512, 4, 4] --

│ └─Conv2d: 2-16 [32, 512, 8, 8] 1,180,160

│ └─ReLU: 2-17 [32, 512, 8, 8] --

│ └─Conv2d: 2-18 [32, 512, 8, 8] 2,359,808

│ └─ReLU: 2-19 [32, 512, 8, 8] --

│ └─MaxPool2d: 2-20 [32, 512, 4, 4] --

├─Sequential: 1-5 [32, 512, 2, 2] --

│ └─Conv2d: 2-21 [32, 512, 4, 4] 2,359,808

│ └─ReLU: 2-22 [32, 512, 4, 4] --

│ └─Conv2d: 2-23 [32, 512, 4, 4] 2,359,808

│ └─ReLU: 2-24 [32, 512, 4, 4] --

│ └─MaxPool2d: 2-25 [32, 512, 2, 2] --

├─AdaptiveAvgPool2d: 1-6 [32, 512, 7, 7] --

├─Sequential: 1-7 [32, 4096] --

│ └─Linear: 2-26 [32, 4096] 102,764,544

│ └─ReLU: 2-27 [32, 4096] --

│ └─Identity: 2-28 [32, 4096] --

├─Sequential: 1-8 [32, 4096] --

│ └─Linear: 2-29 [32, 4096] 16,781,312

│ └─ReLU: 2-30 [32, 4096] --

│ └─Identity: 2-31 [32, 4096] --

├─Linear: 1-9 [32, 3] 12,291

==========================================================================================

Total params: 128,963,139

Trainable params: 128,963,139

Non-trainable params: 0

Total mult-adds (G): 33.08

==========================================================================================

Input size (MB): 1.57

Forward/backward pass size (MB): 257.95

Params size (MB): 515.85

Estimated Total Size (MB): 775.38

### A. Model with Regularization:

Here, L2 Regularization helps prevent the model from memorizing training data too closely and encourages it to generalize better to unseen data which reduces overfitting and the model learns to fit the noise in training data which is why we get a high accuracy of 99.17% for training data and 93.63 and 93.83% for validation and testing data.

A screenshot of a graph

Description automatically generated

.

Loss Graph for Dropout:

We can see loss for training data is the least as it has the highest accuracy and testing and validation loss are close

as their accuracies are close.

A graph showing a line of a graph

Description automatically generated with medium confidence

Confusion Matrix for Dropout:

A screenshot of a computer

Description automatically generated

Precision, Recall and F1 Score for Dropout:

Precision = 0.94

Recall = 0.94

Fscore = 0.94

### Model with Regularization and Dropout:

As during training, at each epoch dropout method randomly select a subset of units to be dropped according to the value it’s been given. This method is preventing our model to overfit and is encouraging it to learn more on certain features.

Due to regularization and dropout applied together, training accuracy has risen upto 98.75% whereas validation and testing accuracy is 93.57 and 94.67 respectively.

A screenshot of a graph

Description automatically generated

Loss Graph for Dropout:

A graph showing a line of a graph

Description automatically generated with medium confidence

As training accuracy is higher, loss is the least for training, while testing loss is not stable, it still gives us a good accuracy value and low loss.

Confusion Matrix for Dropout:

A screenshot of a computer

Description automatically generated

Precision, Recall and F1 Score for Dropout:

Precision = 0.95

Recall = 0.95

Fscore = 0.95

### Model with Regularization, Dropout, and Early Stopping:

Early stopping acts whenever validation performance improves and compares the current performance with it. If the performance does not improve for a certain no of epochs, that is our patience value which we’ve set to 5 which is a significant value for 40 epochs we ran. Our model stops at 17.5 epochs and compared with L2 Regularization and dropout, we get a pretty good training accuracy of 94.01%. Validation and testing accuracies are 92.53 and 92.13% respectively.

A screenshot of a computer

Description automatically generated

Loss Graph for Dropout:

We can see that validation loss is not stable and drops up and down multiple times. So, as our performance improves, without degrading our validation loss, we stop the validation at 17.5 epochs and get a good accuracy.A graph with different colored lines

Description automatically generated

Confusion Matrix for Dropout:

A screenshot of a computer

Description automatically generated

Precision, Recall and F1 Score for Dropout:

Precision = 0.92

Recall = 0.92

Fscore = 0.92

### Model with Regularization, Dropout, Early Stopping and Image Augmentation:

We have also applied Image Augmentation to our model with L2 Regularization, Dropout and Early Stopping. Image Augmentation helps our model prevent from certain attacks such as adversarial attacks by transforming, rotating, scaling, cropping, filter adjustments, etc. Due to Image Augmentation, our model learns to recognize images regardless of their orientation.

But we can see our model has not yet fully learnt from Image Augmentation due to which it gives us low accuracy than Regularization, Dropout and Early Stopping.

We get training accuracy of 82.91%, validation accuracy of 82.73% and testing accuracy of 82.13%.

### A screenshot of a computer Description automatically generated

Loss Graph for Dropout:

As accuracies for all training, validation and testing are the same, the loss graph are at the same range. While training loss is stable, validation and testing losses are changing per epoch.

A graph of a graph

Description automatically generated with medium confidence

Confusion Matrix for Dropout:

A screenshot of a computer

Description automatically generated

Precision, Recall and F1 Score for Dropout:

Precision = 0.83

Recall = 0.82

Fscore = 0.82

**Part 2:**

* + 1. We have constructed a Resnet-18 model using Residual Blocks. Overall, this model consists of 17 convolutional layers and 1 fully-connected layers. It also has Relu, Maxpool, Batch Normalization layers and Average Adaptive Pool layers which are non-constructive layers.

We have constructed a Residual Block which appears for 8 times with different input channels and output channels. Each Residual Block has 2 convolutional layers, 2 Batch Normalization layers, 2 Relu layers and 1 sequential layers.

One basic convolutional layer is in the initial phase before the first residual block and a fully connected layer appears at the end. You can find the whole summary of the model below:

==========================================================================================

Layer (type:depth-idx) Output Shape Param #

==========================================================================================

ResNet18 [32, 3] --

├─Conv2d: 1-1 [32, 64, 32, 32] 9,408

├─BatchNorm2d: 1-2 [32, 64, 32, 32] 128

├─ReLU: 1-3 [32, 64, 32, 32] --

├─MaxPool2d: 1-4 [32, 64, 16, 16] --

├─Sequential: 1-5 [32, 64, 16, 16] --

│ └─BasicBlock: 2-1 [32, 64, 16, 16] --

│ │ └─Conv2d: 3-1 [32, 64, 16, 16] 36,864

│ │ └─BatchNorm2d: 3-2 [32, 64, 16, 16] 128

│ │ └─ReLU: 3-3 [32, 64, 16, 16] --

│ │ └─Conv2d: 3-4 [32, 64, 16, 16] 36,864

│ │ └─BatchNorm2d: 3-5 [32, 64, 16, 16] 128

│ │ └─ReLU: 3-6 [32, 64, 16, 16] --

│ └─BasicBlock: 2-2 [32, 64, 16, 16] --

│ │ └─Conv2d: 3-7 [32, 64, 16, 16] 36,864

│ │ └─BatchNorm2d: 3-8 [32, 64, 16, 16] 128

│ │ └─ReLU: 3-9 [32, 64, 16, 16] --

│ │ └─Conv2d: 3-10 [32, 64, 16, 16] 36,864

│ │ └─BatchNorm2d: 3-11 [32, 64, 16, 16] 128

│ │ └─ReLU: 3-12 [32, 64, 16, 16] --

├─Sequential: 1-6 [32, 128, 8, 8] --

│ └─BasicBlock: 2-3 [32, 128, 8, 8] --

│ │ └─Conv2d: 3-13 [32, 128, 8, 8] 73,728

│ │ └─BatchNorm2d: 3-14 [32, 128, 8, 8] 256

│ │ └─ReLU: 3-15 [32, 128, 8, 8] --

│ │ └─Conv2d: 3-16 [32, 128, 8, 8] 147,456

│ │ └─BatchNorm2d: 3-17 [32, 128, 8, 8] 256

│ │ └─Sequential: 3-18 [32, 128, 8, 8] 8,448

│ │ └─ReLU: 3-19 [32, 128, 8, 8] --

│ └─BasicBlock: 2-4 [32, 128, 8, 8] --

│ │ └─Conv2d: 3-20 [32, 128, 8, 8] 147,456

│ │ └─BatchNorm2d: 3-21 [32, 128, 8, 8] 256

│ │ └─ReLU: 3-22 [32, 128, 8, 8] --

│ │ └─Conv2d: 3-23 [32, 128, 8, 8] 147,456

│ │ └─BatchNorm2d: 3-24 [32, 128, 8, 8] 256

│ │ └─ReLU: 3-25 [32, 128, 8, 8] --

├─Sequential: 1-7 [32, 256, 4, 4] --

│ └─BasicBlock: 2-5 [32, 256, 4, 4] --

│ │ └─Conv2d: 3-26 [32, 256, 4, 4] 294,912

│ │ └─BatchNorm2d: 3-27 [32, 256, 4, 4] 512

│ │ └─ReLU: 3-28 [32, 256, 4, 4] --

│ │ └─Conv2d: 3-29 [32, 256, 4, 4] 589,824

│ │ └─BatchNorm2d: 3-30 [32, 256, 4, 4] 512

│ │ └─Sequential: 3-31 [32, 256, 4, 4] 33,280

│ │ └─ReLU: 3-32 [32, 256, 4, 4] --

│ └─BasicBlock: 2-6 [32, 256, 4, 4] --

│ │ └─Conv2d: 3-33 [32, 256, 4, 4] 589,824

│ │ └─BatchNorm2d: 3-34 [32, 256, 4, 4] 512

│ │ └─ReLU: 3-35 [32, 256, 4, 4] --

│ │ └─Conv2d: 3-36 [32, 256, 4, 4] 589,824

│ │ └─BatchNorm2d: 3-37 [32, 256, 4, 4] 512

│ │ └─ReLU: 3-38 [32, 256, 4, 4] --

├─Sequential: 1-8 [32, 512, 2, 2] --

│ └─BasicBlock: 2-7 [32, 512, 2, 2] --

│ │ └─Conv2d: 3-39 [32, 512, 2, 2] 1,179,648

│ │ └─BatchNorm2d: 3-40 [32, 512, 2, 2] 1,024

│ │ └─ReLU: 3-41 [32, 512, 2, 2] --

│ │ └─Conv2d: 3-42 [32, 512, 2, 2] 2,359,296

│ │ └─BatchNorm2d: 3-43 [32, 512, 2, 2] 1,024

│ │ └─Sequential: 3-44 [32, 512, 2, 2] 132,096

│ │ └─ReLU: 3-45 [32, 512, 2, 2] --

│ └─BasicBlock: 2-8 [32, 512, 2, 2] --

│ │ └─Conv2d: 3-46 [32, 512, 2, 2] 2,359,296

│ │ └─BatchNorm2d: 3-47 [32, 512, 2, 2] 1,024

│ │ └─ReLU: 3-48 [32, 512, 2, 2] --

│ │ └─Conv2d: 3-49 [32, 512, 2, 2] 2,359,296

│ │ └─BatchNorm2d: 3-50 [32, 512, 2, 2] 1,024

│ │ └─ReLU: 3-51 [32, 512, 2, 2] --

├─AdaptiveAvgPool2d: 1-9 [32, 512, 1, 1] --

├─Linear: 1-10 [32, 3] 1,539

==========================================================================================

Total params: 11,178,051

Trainable params: 11,178,051

Non-trainable params: 0

Total mult-adds (G): 4.74

==========================================================================================

Input size (MB): 1.57

Forward/backward pass size (MB): 103.81

Params size (MB): 44.71

Estimated Total Size (MB): 150.09

### A. Model with Regularization

Here, L2 Regularization helps prevent the model from memorizing training data too closely and encourages it to generalize better to unseen data which reduces overfitting and the model learns to fit the noise in training data which is why we get a high accuracy of 96.15% for training data and 91.63 and 91.77% for validation and testing data.

A screenshot of a graph

Description automatically generated

Loss Graph for Dropout:

We can see loss for training data is the least as it has the highest accuracy and testing and validation loss are close as their accuracies are close.

A graph showing a line graph

Description automatically generated with medium confidence

Confusion Matrix for Dropout:

A screenshot of a computer

Description automatically generated

Precision, Recall and F1 Score for Dropout:

Precision = 0.92

Recall = 0.92

Fscore = 0.92

### B. Model with Regularization and Dropout:

As during training, at each epoch dropout method randomly select a subset of units to be dropped according to the value it’s been given. This method is preventing our model to overfit and is encouraging it to learn more on certain features.

Due to regularization and dropout applied together, training accuracy has risen upto 96.65% whereas validation and testing accuracy is 92.57 and 93.00 respectively.

A screenshot of a graph

Description automatically generated

Loss Graph for L2 Regularization:

As training accuracy is higher, loss is the least for training, while testing loss is not stable, it still gives us a good accuracy value and low loss.

A graph of a graph

Description automatically generated with medium confidence

Confusion Matrix for L2 Regularization:

A screenshot of a computer

Description automatically generated

Precision Recall F1 Score for L2 Regularization:

Precision = 0.93

Recall = 0.93

Fscore = 0.93

### C. Model with Regularization, Dropout and Early Stopping:

Early stopping acts whenever validation performance improves and compares the current performance with it. If the performance does not improve for a certain no of epochs, that is our patience value which we’ve set to 5 which is a significant value for 40 epochs we ran. Our model stops at 27 epochs and compared with L2 Regularization and dropout, we get a pretty good training accuracy of 95.52%. Validation and testing accuracies are 91.23 and 92.40% respectively.

A screenshot of a graph

Description automatically generated

Loss Graph for Early Stopping:

We can see that validation loss is not stable and drops up and down multiple times. So, as our performance improves, without degrading our validation loss, we stop the validation at 27 epochs and get a good accuracy.

A graph showing a line graph

Description automatically generated with medium confidence

Confusion Matrix for Early Stopping:

A screenshot of a game

Description automatically generated

Precision Recall F1 Score for Early Stopping:

Precision = 0.93

Recall = 0.92

Fscore = 0.92

### C. Model with Regularization, Dropout, Early Stopping and Image Augmentation:

We have also applied Image Augmentation to our model with L2 Regularization, Dropout and Early Stopping. Image Augmentation helps our model prevent from certain attacks such as adversarial attacks by transforming, rotating, scaling, cropping, filter adjustments, etc. Due to Image Augmentation, our model learns to recognize images regardless of their orientation.

But we can see our model has not yet fully learnt from Image Augmentation due to which it gives us low accuracy than Regularization, Dropout and Early Stopping.

We get training accuracy of 84.40%, validation accuracy of 84.07% and testing accuracy of 82.60%

A screenshot of a computer screen

Description automatically generated

Loss Graph for Early Stopping:

As accuracies for all training, validation and testing are the same, the loss graph are at the same range. While training loss is stable, validation and testing losses are changing per epoch.

A graph showing a line graph

Description automatically generated with medium confidence

Confusion Matrix for Early Stopping:

A screenshot of a computer

Description automatically generated

Precision Recall F1 Score for Early Stopping:

Precision = 0.83

Recall = 0.83

Fscore = 0.83

**Part 3:**

1. Width = 32

Height = 28

Padding = 0

Stride = 1

Filter size = 5 \* 5

Output width = ((Input width - Filter size + 2 \* Padding) / Stride) + 1

= ((32 – 5 + 2\* 0) / 1) + 1

= 28

Output height = ((Input height - Filter size + 2 \* Padding) / Stride) + 1

= ((28 – 5 + 2\* 0) / 1) + 1

= 24

Activation Maps = 10

So output size after first layer = 28\*24\*10

1. Total Parameters = No of input layers \* Filter size \* Filter size \* No of output layers = 3 \* 5 \* 5\* 10 = 750
2. Width = 32

Height = 28

Padding = 1

Stride = 1

Filter size = 5 \* 5

Output width = ((Input width - Filter size + 2 \* Padding) / Stride) + 1

= ((32 – 5 + 2\* 1) / 1) + 1

= 30

Output height = ((Input height - Filter size + 2 \* Padding) / Stride) + 1

= ((28 – 5 + 2\* 1) / 1) + 1

= 26

Activation Maps = 10

So output size after first layer = 30\*26\*10

1. If image is grayscale, no of input layers = 1

Total Parameters = No of input layers \* Filter size \* Filter size \* No of output layers = 1 \* 5 \* 5\* 10 = 250

1. Softmax activation function is useful for multiclass classification and for requirement of probabilistic outputs.

Formula for softmax activation function is based on probability -> exi / ∑i=1n exi.

1. Here, xi  is the input value.

Adding a constant term = xi + c

Equation becomes

= exi + c  / ∑i=1n exi + c.

= exi . ec  / ∑i=1n exi. ec

e c  will get cancelled out.

= exi / ∑i=1n exi.

Therefore, we get the same equation as the initial.

So softmax activation function is unaffected by adding constant values in input values.