Project 1 – Implementing CNN Architectures

Hemant Koti
Department of Computer Science
University at Buffalo
hemantko@buffalo.edu

Abstract

This report aims to detail a comparison study between three popular CNN architectures – VGG16, ResNet18, and InceptionV3 networks on CIFAR 100 dataset to perform the task of image classification for 100 different labels. In this report, we list out the models' performance with SGD and ADAM optimizers along with pre-defined variations in the regularization techniques. Additionally, we will explore various hyperparameters that finetune a model to improve performance.

1 Introduction

Convolutional neural networks are a state-of-the-art algorithms that are proven to work extremely well for image recognition and analysis. Over the past years, several CNN architectures were introduced to solve distinct problems related to image classification. These are very deep convolutional neural networks that gained special traction in ImageNet competitions for achieving benchmark results as shown in Figure 1.

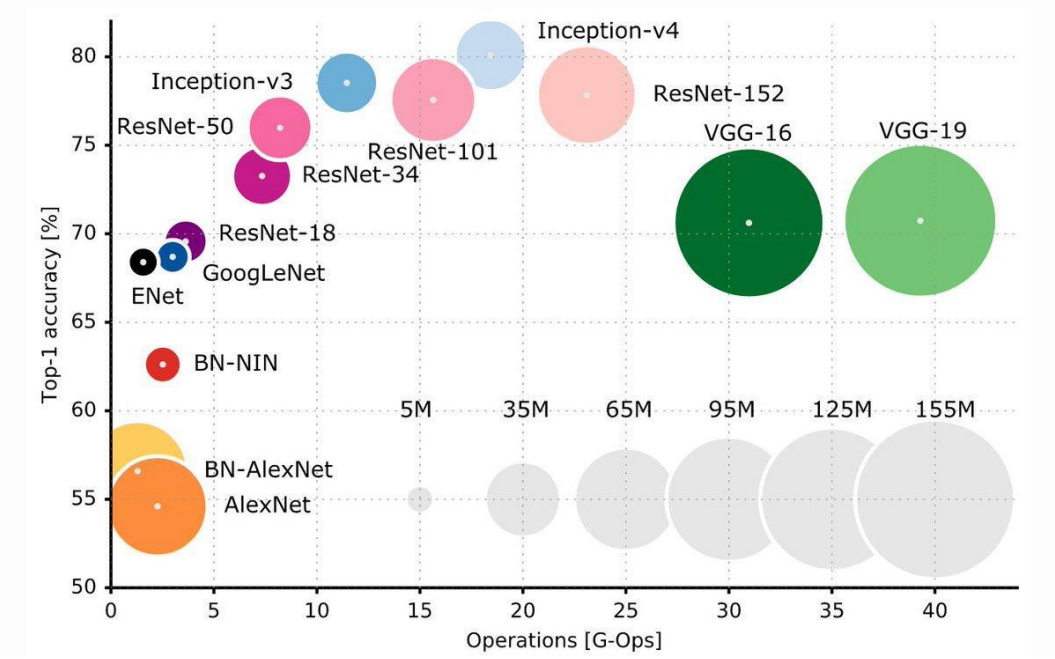

Figure 1: Comparison of CNN Architectures on ImageNet Dataset [1]

Our motivation is to explore some of these architectures over different network settings to not only understand the process of image recognition but also to analyze and compare each models' performance, memory usage, and computation cost. This comparison is especially important for narrowing down the best-suited model on CIFAR 100 dataset.

2 Dataset

The CIFAR-100 dataset consists of 60000 32x32 color images in 100 classes, with 600 images per class. There are 500 training images and 100 testing images per class [3]. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a *fine* label (the class to which it belongs) and a *coarse* label (the superclass to which it belongs).

3 VGG-16 Network

The VGGNet architecture explores the idea of having small convolutional filters with deeper networks instead of large filters with shallow networks. VGGNet contains two variants one with 16 layers and the other with 19 layers. For our implementation, we choose VGG-16 architecture which consists of the following layers.

- 13 3 x 3 convolutional layers with stride 1 and padding 1.
- 5 2 x 2 max-pooling layers with of stride 2.
- 3 fully connected layers.
- The total memory used by a single image in a forward pass ~ 96 MB.
- The total number of parameters ~ 138 Million.

These layers are stacked in the following manner [2].

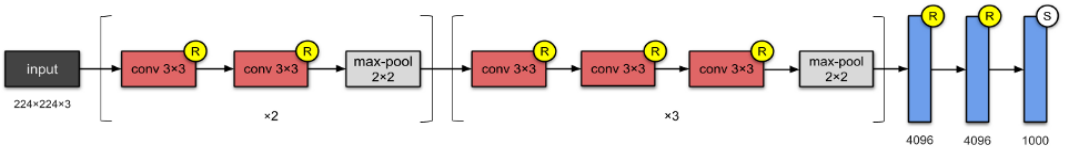

Figure 2: Visualizing VGG-16 architecture

3.1 Implementation and Results

The implementation follows the VGG-16 network architecture [4] with 16 layers as described above. The code consists of two main classes for the network architecture – VGG() and make_layers(). The make_layers() method accepts a configuration parameter that constructs the convolution and max-pooling layers, whereas the VGG method contains the classifier with fully connected layers, ReLU activation, and an output layer with the number of classes (100 in this case).

For our training, we use 200 epochs (derived from experimentation) to attain the best accuracy. In some cases, the training loss starts to increase gradually after the model is converged which results in overfitting. To halt the training at the right point, we used an early-stopping technique with a patience value of 10.

As for the training itself, we iterate over the trainloader containing a batch size of 200 with

images and labels. We then calculate the loss in each epoch along with the training accuracy. In the same epoch, we turn off the gradient for validation and calculate the loss and accuracy of the testloader dataset. We then print the training loss, training accuracy, best training accuracy, testing loss, testing accuracy, and best testing accuracy parameters.

The following are some of the network combinations used to train the model.

3.1.1 SGD with Batch Normalization

The optimizer used here is Stochastic gradient descent with a learning rate of 0.001, a momentum of 0.9, and a weight decay of 0.00001 (L2 regularizer).

We add a batch normalization layer in this combination after every convolutional layer without any dropout.

The accuracy for this combination is ~ *64%* with precision and recall ~ *58%* as a weighted average of all the classes.

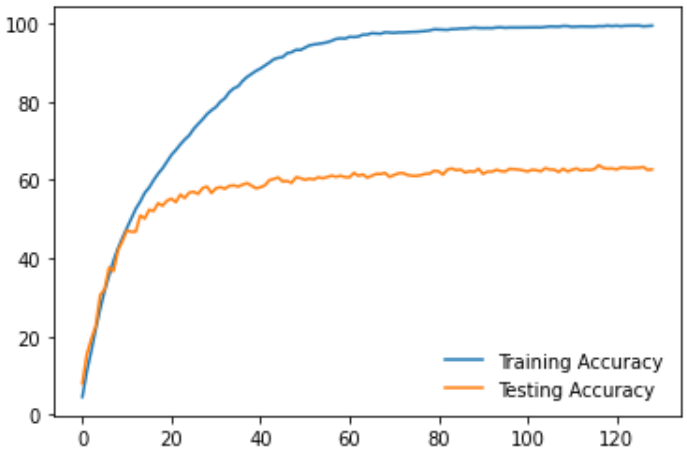

Figure 3: Training vs Testing accuracy for VGG16_SGD_BatchNormalization

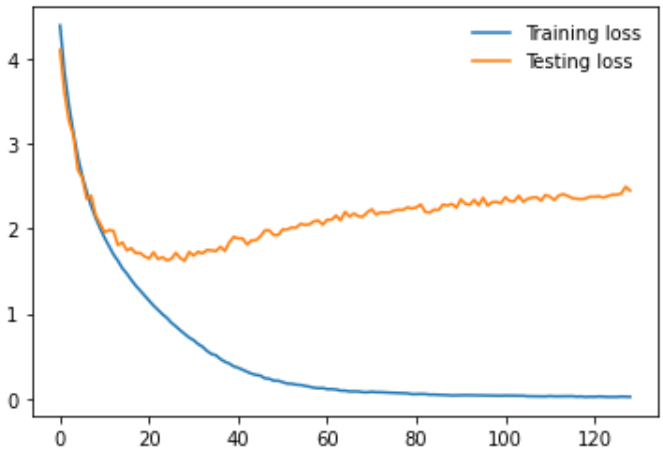

Figure 4: Training vs Testing loss for VGG16_SGD_BatchNormalization

3.1.2 SGD with Dropout

The optimizer used here is Stochastic gradient descent with a learning rate of 0.001, a momentum of 0.9, and a weight decay of 0.0005 (L2 regularizer).

We add a dropout with a probability of 0.3 for fully connected layers and a 2d dropout with a probability of 0.05 for the convolutional layers. There is no batch normalization applied in this case.

The accuracy for this combination is ~ *61.45%* with precision and recall ~ *48%* as a weighted average of all the classes

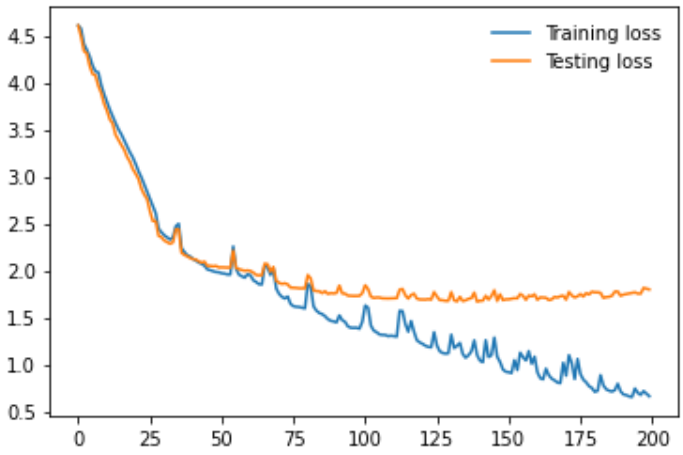

Figure 5: Training vs Testing loss for VGG16_SGD_Dropout

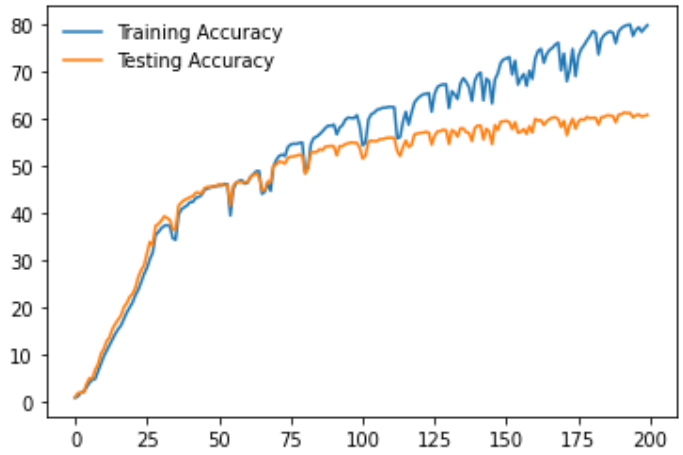

Figure 6: Training vs Testing accuracy for VGG16_SGD_Dropout

3.1.3 SGD without Regularization

The optimizer used here is Stochastic gradient descent with a learning rate of 0.001, a momentum of 0.9. We also use a scheduler that dynamically updates the learning rate after 0,

10, 30, 80, 180 epochs to avoid vanishing gradients. Additionally, we also implement gradient clipping to avoid the exploding gradient problem when dealing with high training losses.

We don't use any dropout or batch normalization or L2 regularization in this case.

The accuracy for this combination is ~ *36%* with precision and recall ~ *22% and 24%* as a weighted average of all the classes. The low accuracy is a consequence of not using any regularization parameters.

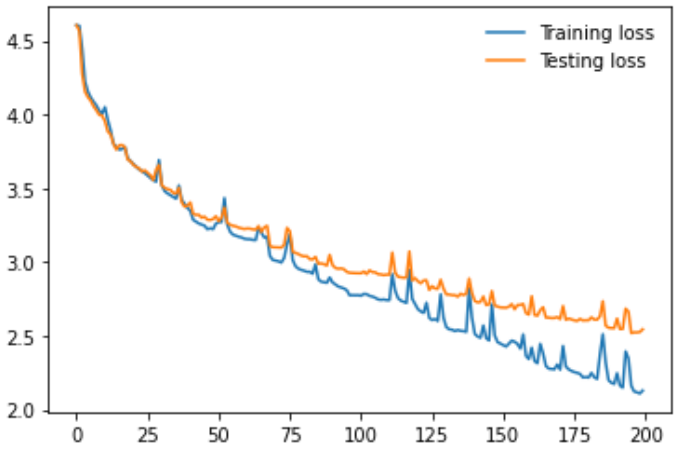

Figure 7: Training vs Testing loss for VGG16_SGD_NoRegularization

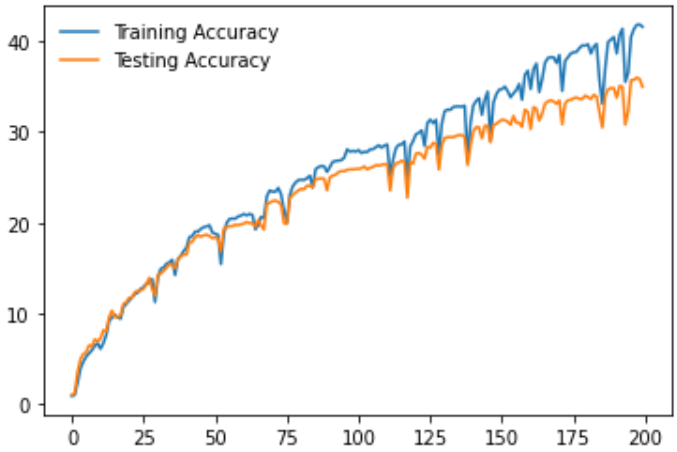

Figure 8: Training vs Testing accuracy for VGG16_SGD_NoRegularization

3.1.4 ADAM with Batch Normalization

The optimizer used here is Adam with a learning rate of 0.0001 and an L2 regularization value of 0.00001. In the case of adam optimizer, we *dynamically change the learning rate* parameter using the update_lr() method to adaptively train our model when the testing accuracy decreases for a prolonged time.

We add a batch normalization layer in this combination after every convolutional layer without any dropout.

The accuracy for this combination is ~ *61.38%* with precision and recall ~ *56%* as a weighted average of all the classes.

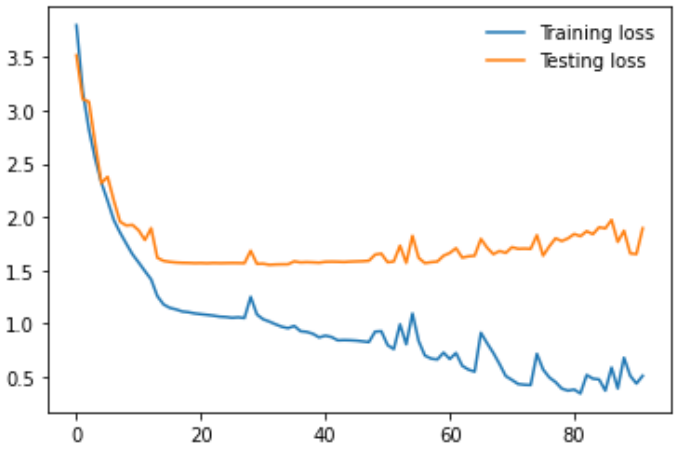

Figure 9: Training vs Testing loss for VGG16_ADAM_BatchNormalization

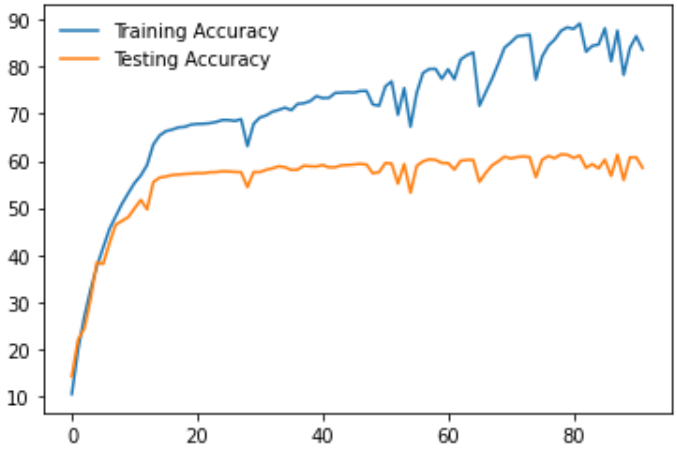

Figure 10: Training vs Testing accuracy for VGG16_ADAM_BatchNormalization

3.1.5 ADAM with Dropout

The optimizer used here is Adam with a learning rate of 0.0001.

We add a dropout with a probability of 0.3 for fully connected layers and a 2d dropout with a probability of 0.02 for the convolutional layers. There is no batch normalization applied in this case.

The accuracy for this combination is ~ *56.62%* with precision and recall ~ *46%* as a weighted average of all the classes.

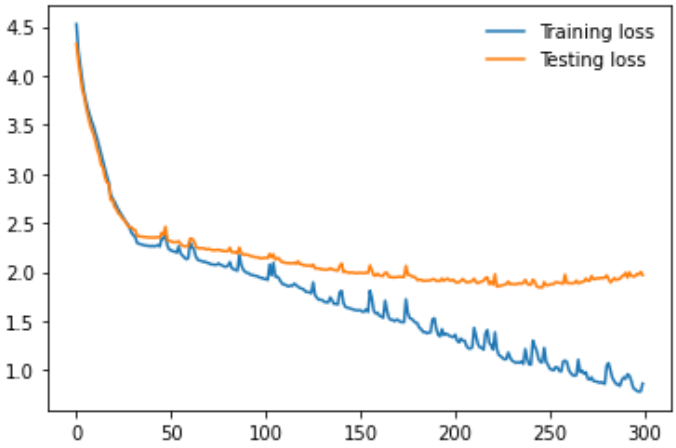

Figure 11: Training vs Testing loss for VGG16_ADAM_Dropout

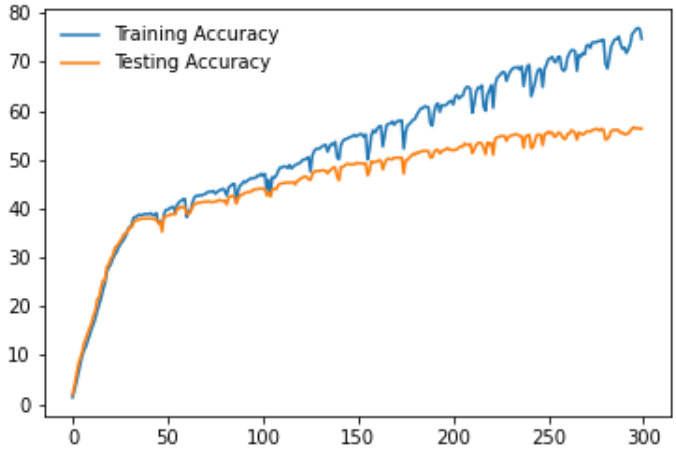

Figure 12: Training vs Testing accuracy for VGG16_ADAM_Dropout

3.1.6 ADAM without Regularization

The optimizer used here is Adam with a learning rate of 0.001. We implement gradient clipping to avoid the exploding gradient problem when dealing with high training losses.

We don't use any dropout or batch normalization or L2 regularization in this case.

The accuracy for this combination is ~ *41.92%* with precision and recall ~ *33%* as a weighted average of all the classes. The low accuracy is a consequence of not using any regularization parameters.

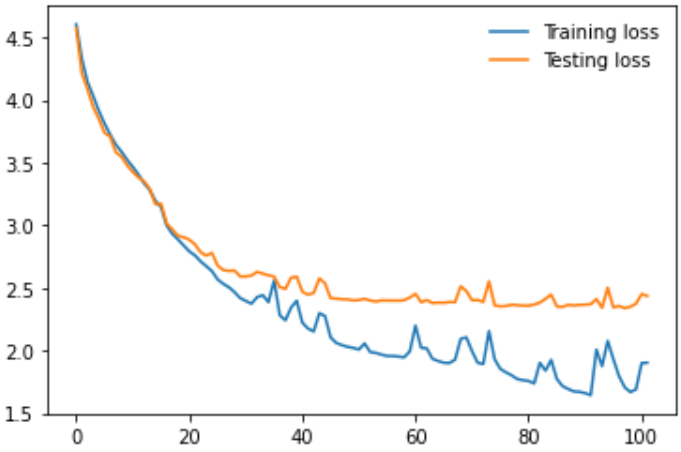

Figure 13: Training vs Testing loss for VGG16_ADAM_NoRegularization

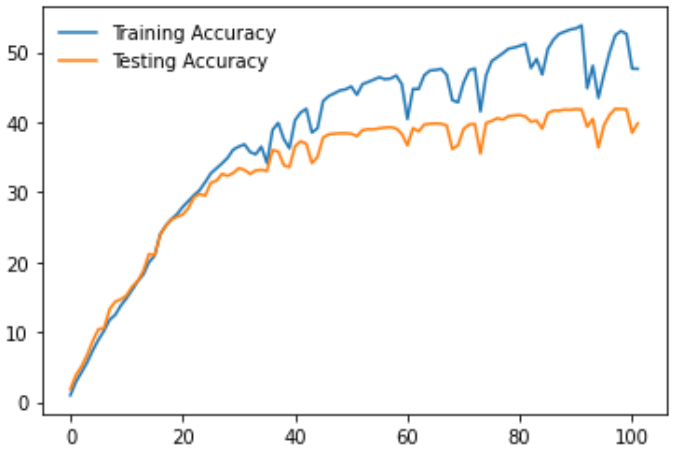

Figure 14: Training vs Testing accuracy for VGG16_ADAM_NoRegularization

4 ResNet18 Network

The deeper models face a huge barrier to training neural networks due to the vanishing gradients problem. Very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow. The core idea of ResNet is using *identity shortcut connections (residual connections)* that skips one or more layers.

In the case of VGG, every layer is connected to its previous layer, from which it is getting its inputs. This makes sure that upon propagation from layer to layer, more and more useful features are carried and less important features are dropped out [5]. This is not the best way since the later layers cannot see what the former layers have seen. ResNet addresses this problem by connecting not just the previous layer to the current layer but also a layer behind the previous layer. By incorporating this, now each layer can see more than just its previous layer's observations.

The architecture consists of the following layers

- 2 3 x 3 conv2_x layers with output size 56 x 56.

- 2 3 x 3 conv3_x layers with output size 28 x 28.
- 2 3 x 3 conv4_x layers with output size 14 x 14.
- 2 3 x 3 conv5_x layers with output size 7 x 7.
- 1 global average pooling layer after the last convolution layer.
- 1 fully connected layer.

These layers are stacked in the following manner [2].

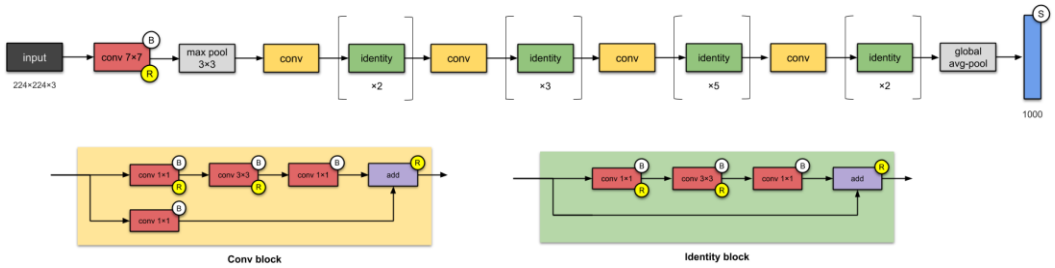

Figure 15: Visualizing ResNet50 architecture

4.1 Implementation and Results

The implementation follows the ResNet network architecture [6] with 18 layers as described above. The code consists of two main classes for the network architecture – BasicBlock() and ResNet18(). The ResNet18() accepts a block size parameter that constructs the conv2_x, conv3_x, conv4_x, and conv5_x layers along with one global max-pooling layer and ReLU activation function to the output layer. The BasicBlock() contains the classifier with one fully connected layer, 2 3 x 3 convolution layers, downsample parameter for residual connections.

For our training, we use 200 epochs (derived from experimentation) to attain the best accuracy (sometimes 300 for no regularization cases). In some cases, the training loss starts to increase gradually after the model is converged which results in overfitting. To halt the training at the right point, we used an early-stopping technique with a patience value of 20.

As for the training itself, we iterate over the trainloader containing a batch size of 200 with images and labels. We then calculate the loss in each epoch along with the training accuracy. In the same epoch, we turn off the gradient for validation and calculate the loss and accuracy of the testloader dataset. We then print the training loss, training accuracy, best training accuracy, testing loss, testing accuracy, and best testing accuracy parameters.

The following are some of the network combinations used to train the model.

4.1.1 SGD with Batch Normalization

The optimizer used here is Stochastic gradient descent with a learning rate of 0.01, a momentum of 0.9.

We add a batch normalization layer in this combination after every convolutional layer without any dropout.
The accuracy for this combination is ~ *66.26%* with precision and recall ~

62% as a weighted average of all the classes.

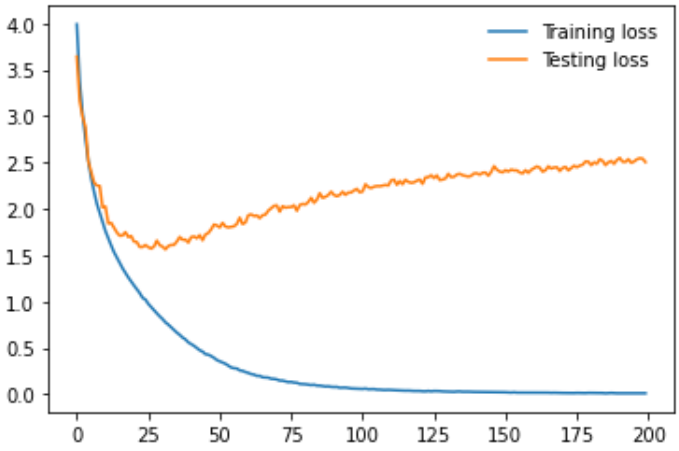

Figure 16: Training vs Testing loss for ResNet18_SGD_BatchNormalization

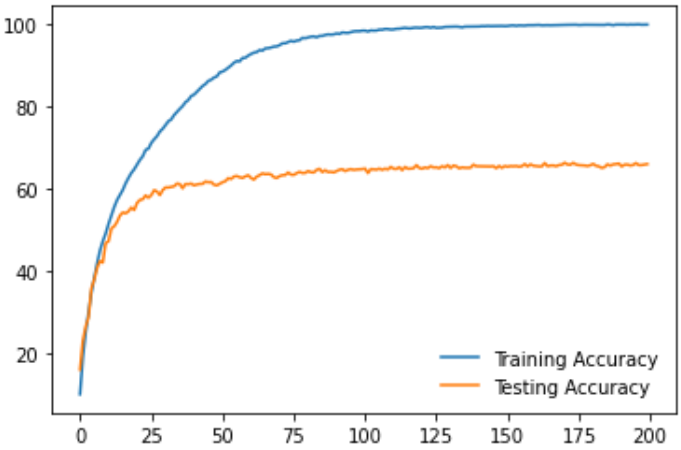

Figure 17: Training vs Testing accuracy for ResNet18_SGD_BatchNormalization

4.1.2 SGD with Dropout

The optimizer used here is Stochastic gradient descent with a learning rate of 0.001, a momentum of 0.9, and a weight decay of 0.001 (L2 regularizer).

We add a dropout with a probability of 0.5 for fully connected layers and a 2d dropout with a probability of 0.01 for the convolutional layers. There is no batch normalization applied in this case.

The accuracy for this combination is ~ *50.75%* with precision and recall ~ *41%* as a weighted average of all the classes

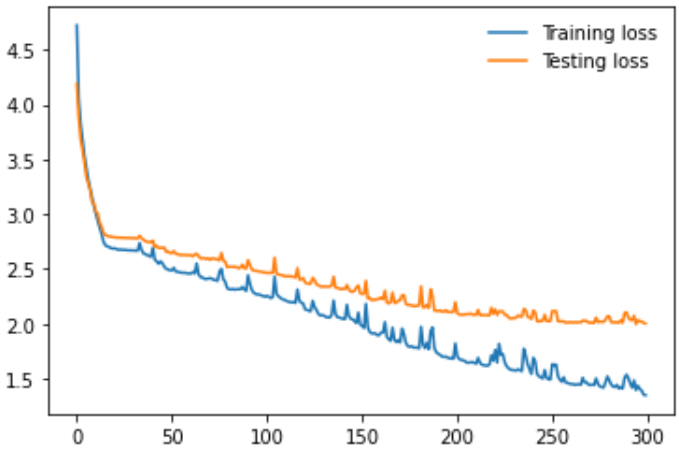

Figure 18: Training vs Testing loss for ResNet18_SGD_Dropout

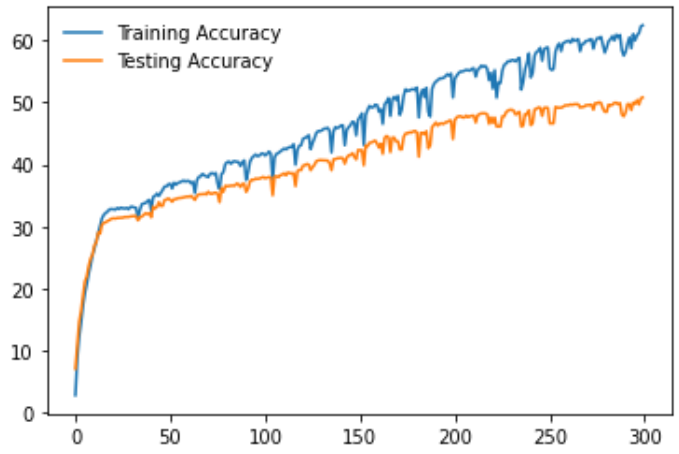

Figure 19: Training vs Testing accuracy for ResNet18_SGD_Dropout

4.1.3 SGD without Regularization

The optimizer used here is Stochastic gradient descent with a learning rate of 0.001, a momentum of 0.9.

We don't use any dropout or batch normalization or L2 regularization in this case.

The accuracy for this combination is ~ *50.62%* with precision and recall ~ *44%* as a weighted average of all the classes. Since we use skip connections in ResNet18 there won't be an issue for vanishing gradients and hence the high accuracy even without any regularization.

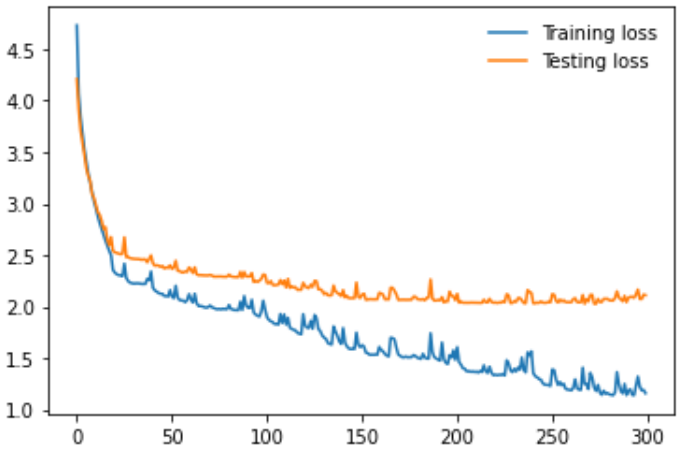

Figure 20: Training vs Testing loss for ResNet18_SGD_NoRegularization

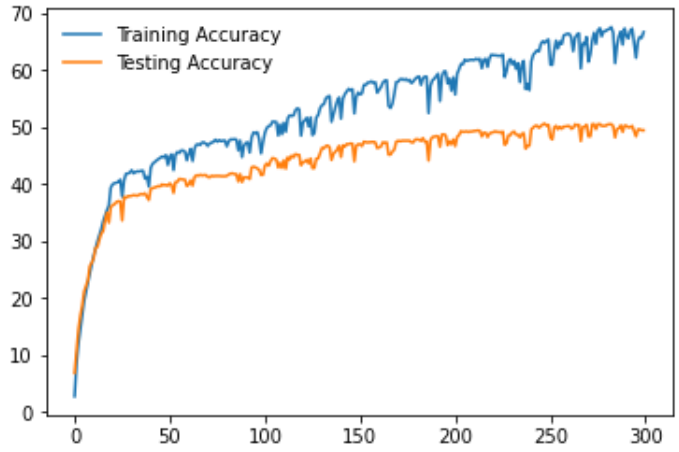

Figure 21: Training vs Testing accuracy for ResNet18_SGD_NoRegularization

4.1.4 ADAM with Batch Normalization

The optimizer used here is Adam with a learning rate of 0.001 and an L2 regularization value of 0.00001. In the case of adam optimizer, we *dynamically change the learning rate* parameter using the update_lr() method to adaptively train our model when the testing accuracy decreases for a prolonged time.

We add a batch normalization layer in this combination after every convolutional layer without any dropout.

The accuracy for this combination is ~ *68.01%* with precision and recall ~ *64%* as a weighted average of all the classes.

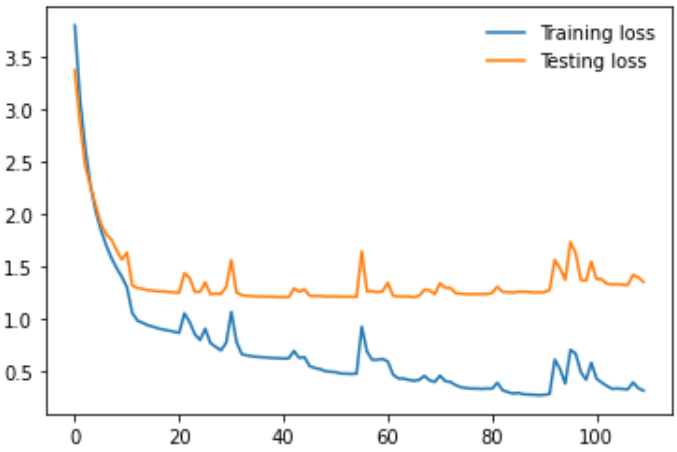

Figure 22: Training vs Testing loss for ResNet18_ADAM_BatchNormalization

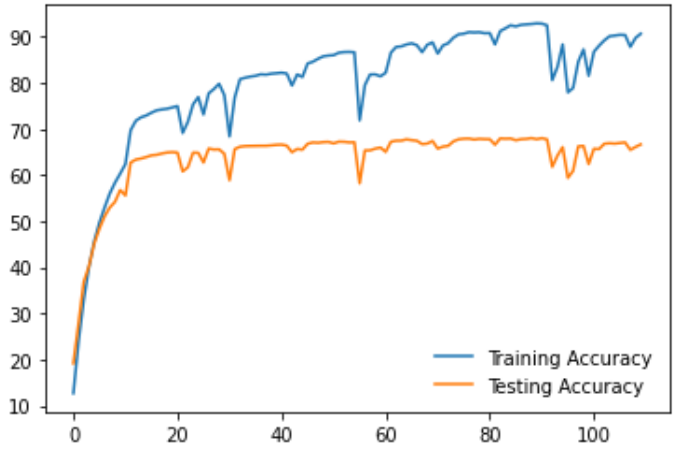

Figure 23: Training vs Testing accuracy for ResNet18_ADAM_BatchNormalization

4.1.5 ADAM with Dropout

The optimizer used here is Adam with a learning rate of 0.0001 and an L2 regularization value of 0.00001.

We add a dropout with a probability of 0.5 for fully connected layers and a 2d dropout with a probability of 0.05 for the convolutional layers. There is no batch normalization applied in this case.

The accuracy for this combination is ~ *62.72%* with precision and recall ~ *56%* as a weighted average of all the classes.

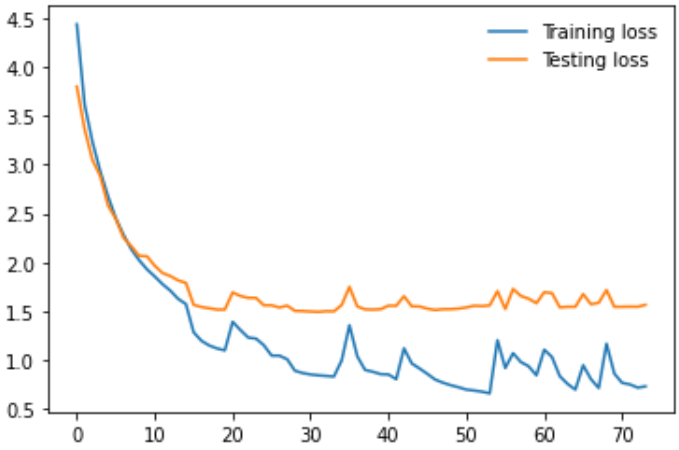

Figure 24: Training vs Testing loss for ResNet18_ADAM_Dropout

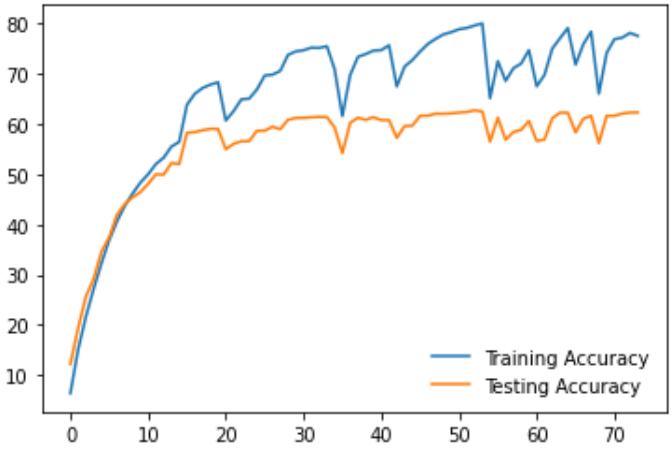

Figure 25: Training vs Testing accuracy for ResNet18_ADAM_Dropout

4.1.6 ADAM without Regularization

The optimizer used here is Adam with a learning rate of 0.001.

We don't use any dropout or batch normalization or L2 regularization in this case.

The accuracy for this combination is ~ *61.73%* with precision and recall ~ *57%* as a weighted average of all the classes. Since we use skip connections in ResNet18 there won't be an issue for vanishing gradients and hence the high accuracy even without any regularization.

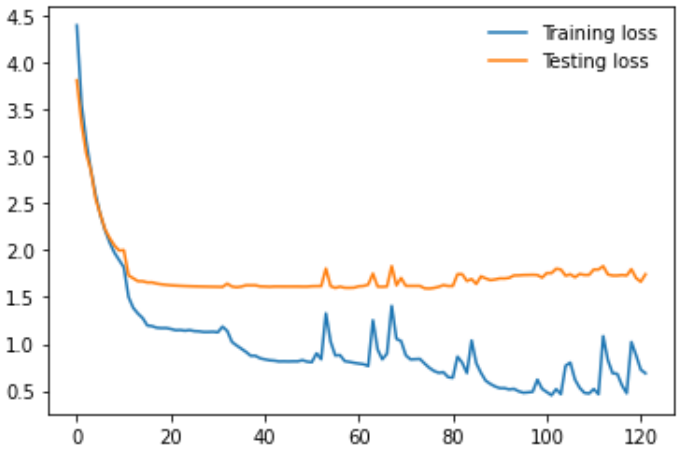

Figure 27: Training vs Testing loss for ResNet18_ADAM_NoRegularization

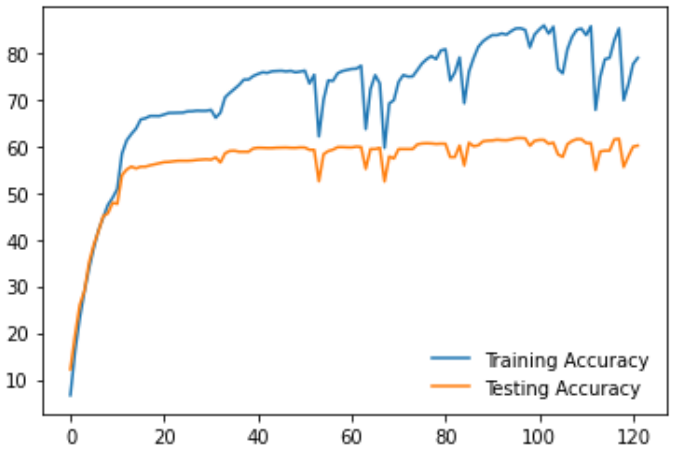

Figure 28: Training vs Testing accuracy for ResNet18_ADAM_NoRegularization

5 InceptionV3 Network

InceptionNet contains deeper networks with the best computation efficiency. It contains an efficient inception module and avoids the expensive fully connected layers in a network. The idea behind InceptionV3 is to avoid *representational bottlenecks* (this means drastically reducing the input dimensions of the next layer) and have more efficient computations by using factorization methods [2].

The architecture consists of the following layers

- Factorized Convolutions: Reduces the number of parameters.
- Smaller Convolutions (Inception Module A): Two 3 × 3 filters replace a 5 × 5 convolution filter.
- Factorization into Asymmetric Convolutions (Inception Module B and C): A 3 × 3 convolution is replaced by a 1 × 3 convolution followed by a 3 × 1 convolution.
- Auxiliary classifier: An auxiliary classifier is a small CNN inserted between layers

during training, and the loss incurred is added to the main network loss.

These layers are stacked in the following manner [2].

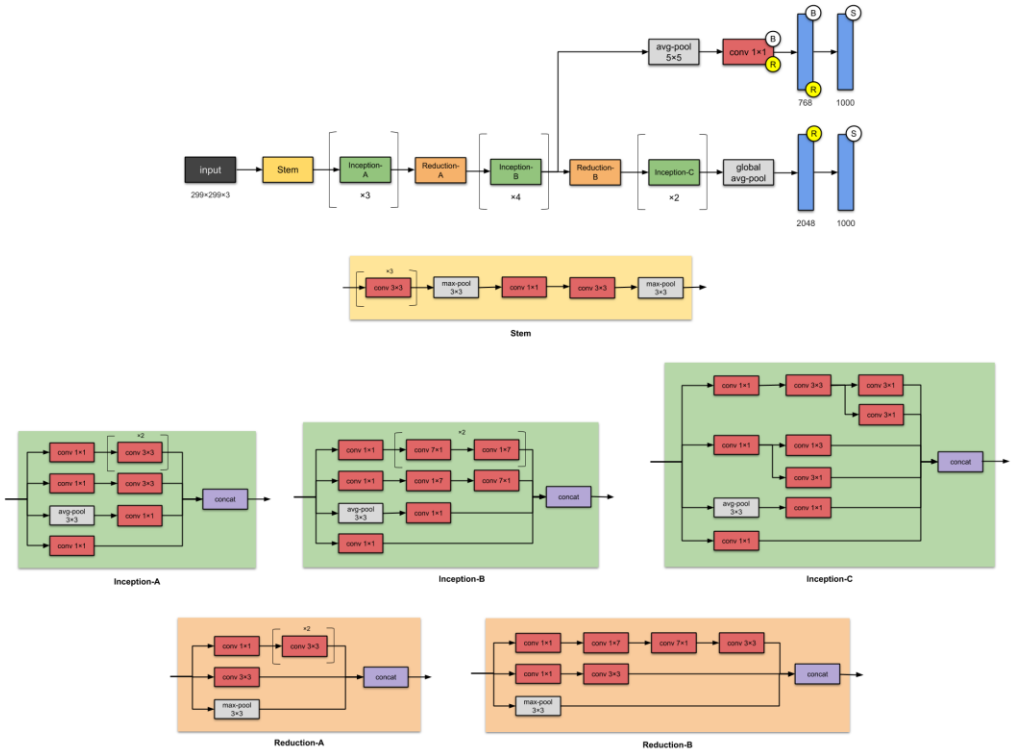

Figure 29: Visualizing InceptionV3 architecture

5.1 Implementation and Results

The implementation follows the InceptionV3 network architecture [7] as described above. The code consists of various classes for the network architecture – InceptionA(), InceptionB(), InceptionC(), InceptionD(), InceptionE() blocks corresponding to the image above.

For our training, we use 100 epochs (derived from experimentation) to attain the best accuracy. Usually, for InceptionNet training, a single epoch takes a lot of time (~ 180 seconds) In some cases, the training loss starts to increase gradually after the model is converged which results in overfitting. To halt the training at the right point, we used an early-stopping technique with a patience value of 10.

As for the training itself, we iterate over the trainloader containing a batch size of 200 with images and labels. We then calculate the loss in each epoch along with the training accuracy. In the same epoch, we turn off the gradient for validation and calculate the loss and accuracy of the testloader dataset. We then print the training loss, training accuracy, best training accuracy, testing loss, testing accuracy, and best testing accuracy parameters.

The following are some of the network combinations used to train the model.

5.1.1 SGD with Batch Normalization

The optimizer used here is Stochastic gradient descent with a learning rate of 0.001, a momentum of 0.9, L2 regularization of 0.0005.

We add a batch normalization layer in this combination after every convolutional layer without any dropout.

The accuracy for this combination is ~ *67.49%* with precision and recall ~ *59%* as a weighted average of all the classes.

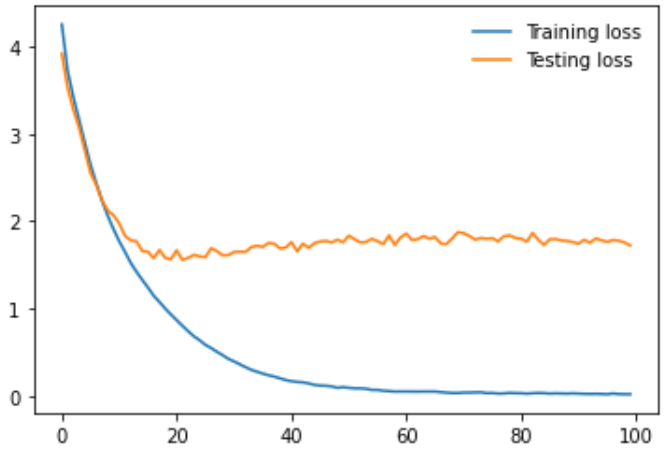

Figure 30: Training vs Testing loss for InceptionV3_SGD_BatchNormalization

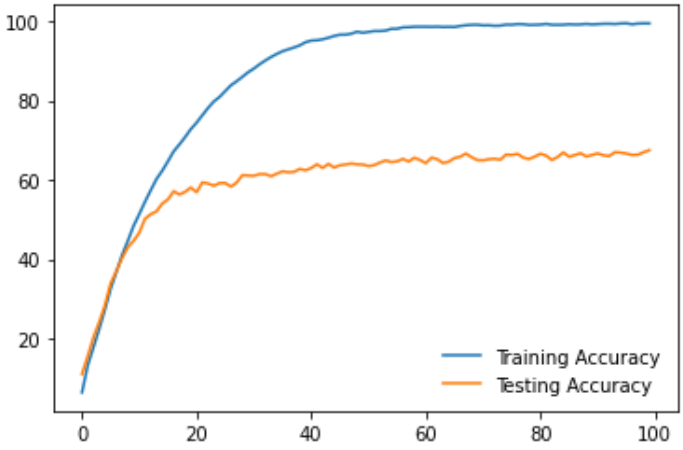

Figure 31: Training vs Testing accuracy for InceptionV3_SGD_BatchNormalization

5.1.2 SGD with Dropout

The optimizer used here is Stochastic gradient descent with a learning rate of 0.01, a momentum of 0.9, and a weight decay of 0.0005 (L2 regularizer).

We add a dropout with a probability of 0.5 for fully connected layers and a 2d dropout with a probability of 0.05 for the convolutional layers. There is

no batch normalization applied in this case.

The accuracy for this combination is ~ *52.27%* with precision and recall ~ *45%* as a weighted average of all the classes

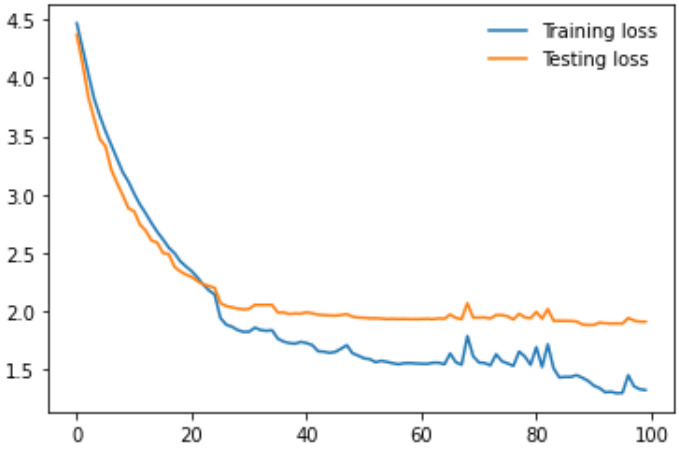

Figure 32: Training vs Testing loss for InceptionV3_SGD_Dropout

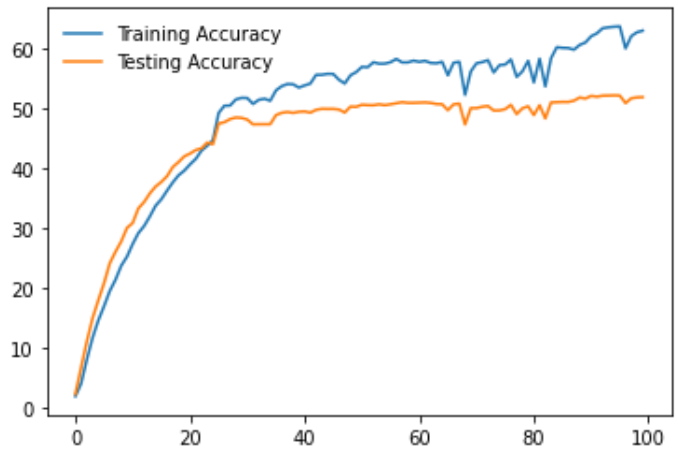

Figure 33: Training vs Testing accuracy for InceptionV3_SGD_Dropout

5.1.3 SGD without Regularization

The optimizer used here is Stochastic gradient descent with a learning rate of 0.002, a momentum of 0.9, and an L2 regularization value of 0.0005.

We don't use any dropout or batch normalization or L2 regularization in this case.

The accuracy for this combination is ~ *12.14 %* with precision and recall ~ *7%* as a weighted average of all the classes. The low accuracy is a consequence of not using any regularization parameters.

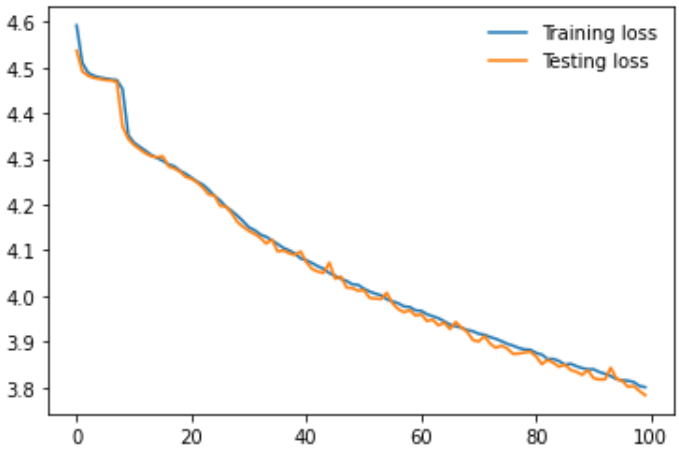

Figure 34: Training vs Testing loss for InceptionV3_SGD_NoRegularization

Figure 35: Training vs Testing accuracy for InceptionV3_SGD_NoRegularization

5.1.4 ADAM with Batch Normalization

The optimizer used here is Adam with a learning rate of 0.001. In the case of adam optimizer, we *dynamically change the learning rate* parameter using the update_lr() method to adaptively train our model when the testing accuracy decreases for a prolonged time.

We add a batch normalization layer in this combination after every convolutional layer without any dropout.

The accuracy for this combination is ~ *71.33%* with precision and recall ~ *63%* as a weighted average of all the classes.

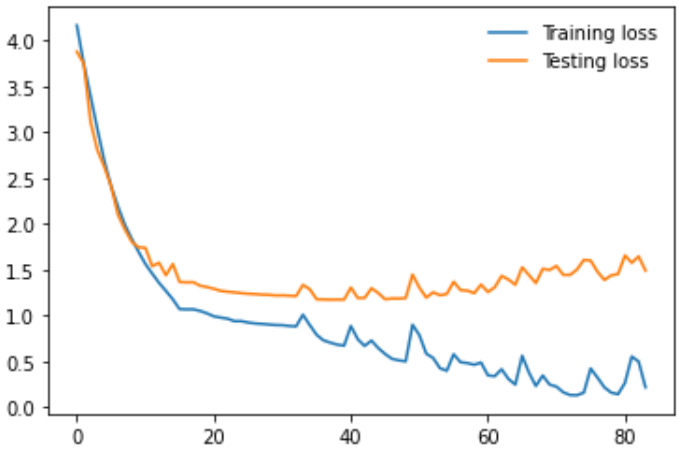

Figure 36: Training vs Testing loss for InceptionV3_ADAM_BatchNormalization

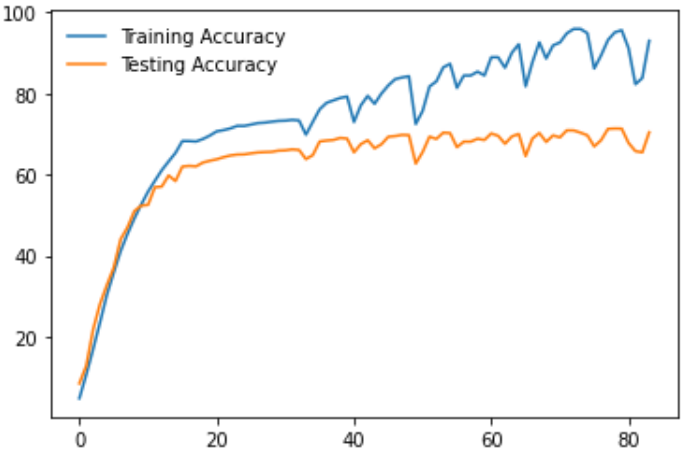

Figure 37: Training vs Testing accuracy for InceptionV3_ADAM_BatchNormalization

5.1.5 ADAM with Dropout

The optimizer used here is Adam with a learning rate of 0.001.

We add a dropout with a probability of 0.5 for fully connected layers and a 2d dropout with a probability of 0.05 for the convolutional layers. There is no batch normalization applied in this case.

The accuracy for this combination is ~ *52.1%* with precision and recall ~ *44%* as a weighted average of all the classes.

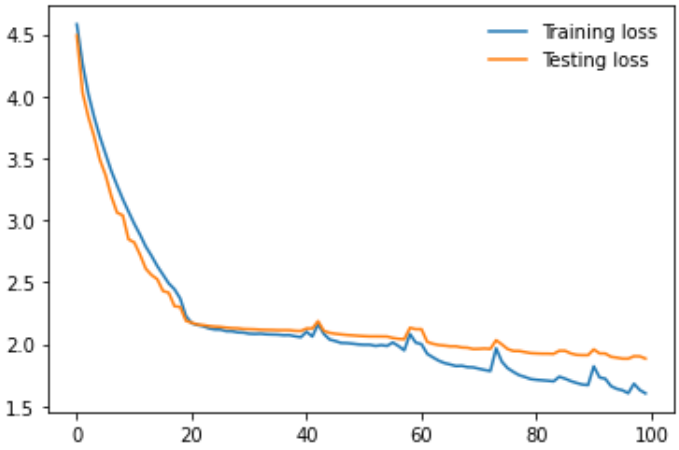

Figure 38: Training vs Testing loss for InceptionV3_ADAM_Dropout

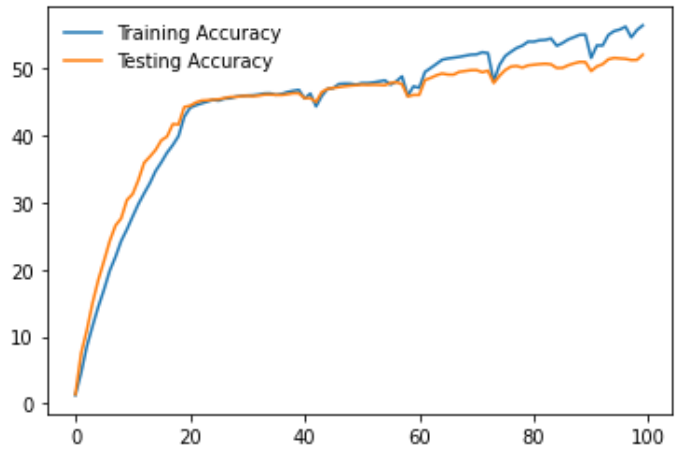

Figure 39: Training vs Testing accuracy for InceptionV3_ADAM_Dropout

5.1.6 ADAM without Regularization

The optimizer used here is Adam with a learning rate of 0.001.

We don't use any dropout or batch normalization or L2 regularization in this case.

The accuracy for this combination is ~ *46.67%* with precision and recall ~ *38%* as a weighted average of all the classes. The low accuracy is a consequence of not using any regularization parameters.

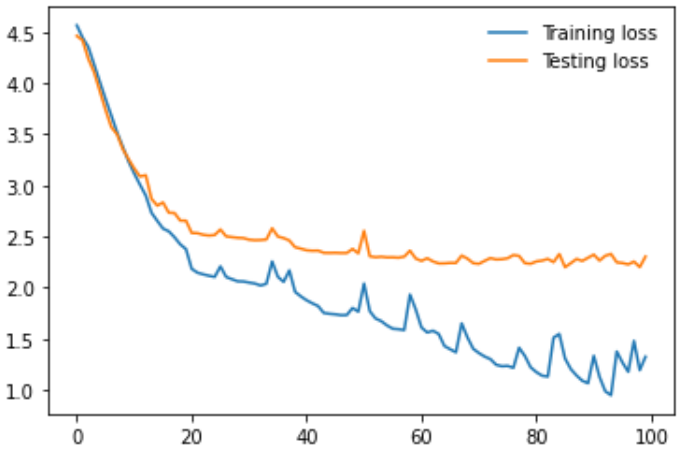

Figure 40: Training vs Testing loss for InceptionV3_ADAM_NoRegularization

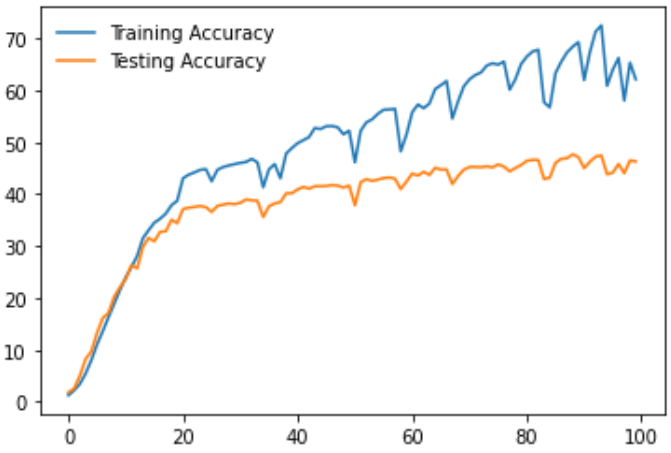

Figure 41: Training vs Testing accuracy for InceptionV3_ADAM_NoRegularization

6 Results and Comparison

	Arch	VGG-16 (in %)			ResNet18 (in %)			InceptionV3 (in %)		
Optimizer	Score / Setting	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy
SGD	Batch Norm	58	58	64	62	62	66.26	59	59	67.49
	Dropout	48	48	61.45	41	41	50.75	45	45	52.27
	No Regularizer	22	24	36	44	44	50.62	7	7	12.14
ADAM	Batch Norm	56	56	61.38	64	64	68.01	63	63	71.33
	Dropout	46	46	56.62	56	56	62.72	44	44	52.1
	No Regularizer	33	33	41.92	57	57	61.73	38	38	46.67

References

[1] https://arxiv.org/pdf/1605.07678.pdf

[2] https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#c5a6

[3] https://www.cs.toronto.edu/~kriz/cifar.html

[4] https://arxiv.org/pdf/1409.1556.pdf

[5] https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049

[6] https://arxiv.org/pdf/1512.03385.pdf

[7] https://arxiv.org/pdf/1512.00567v1.pdf