Assignment 1 - Report

EE5178: Modern Computer Vision

Vishnu Vinod

Contents

1 Introduction

Image Classification is a fundamental task in the field of machine learning whereby we attempt to train a machine learning model to identify and classify images into various classes. During recent years, with the advent of deep learning and the increasing use of **Deep Neural Networks** (DNNs) there has been swift progress in the efficiency and robustness of the SOTA (state-of-the-art) models used for this task. Of the most common image classification competitions is the **ImageNet Large Scale Visual Recognition Challenge** or **ILSVRC** with the winners of the challenge innovating newer and better models. Many of the erstwhile SOTA models like **AlexNet**, **GoogLeNet**, **VGG**, **ResNet** have been innovated in the pursuit of this challenge.

In this assignment we start from the very basics and first train a simple **MLP** (Multi-Layer Perceptron) model to classify images from the **CIFAR10** dataset. Then we build and train the **VGG11** (Visual Graphics Group - 11) model from scratch and perform experiments on both these models.

2 Dataset

The **CIFAR-10** dataset (Canadian Institute For Advanced Research) is a collection of **60,000 images** that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 **32x32** color images in **10 classes**. The 10 different classes represent **airplanes**, **cars**, **birds**, **cats**, **deer**, **dogs**, **frogs**, **horses**, **ships**, and **trucks**. There are 6,000 images of each class.

The main reason for the popularity of the CIFAR10 dataset is that because of the small size of the images and the low number of classes it is an ideal dataset for researchers to test out new algorithms and models for image classification in a computationally inexpensive manner. It has generally been seen that CNN based networks perform better on the CIFAR10 dataset when compared to other models such as MLPs etc.

2.1 Data Preprocessing

The 60,000 images of the CIFAR10 dataset have been split into 50000 images of the training dataset with 5000 images per class and 10000 images of the test dataset with 1000 images per class. We further split the training images into a training and validation set with 5000 images **(10%)** going into the **validation set** and the remaining 45000 images (90%) going to the training set. This is done randomly using the `SubsetRandomSampler()` function.

We also perform a **normalization transform** on the images when we load the images to ensure that the pixel values get re-normalized with a mean $\mu = 0$ and a standard deviation of $\sigma = 1.0$.

3 Multi-Layer Perceptron (MLP) Model

A **Multi-Layer Perceptron** (MLP) is a type of **feedforward ANN** (Artificial Neural Network) which consists of all the neurons in a particular layer being fully connected to the preceding layer. It is a type of dense network which was initially used for machine learning tasks because of its ability to learn decision boundaries for data which is not linearly separable. The MLP follows a **backpropagation** algorithm for training.

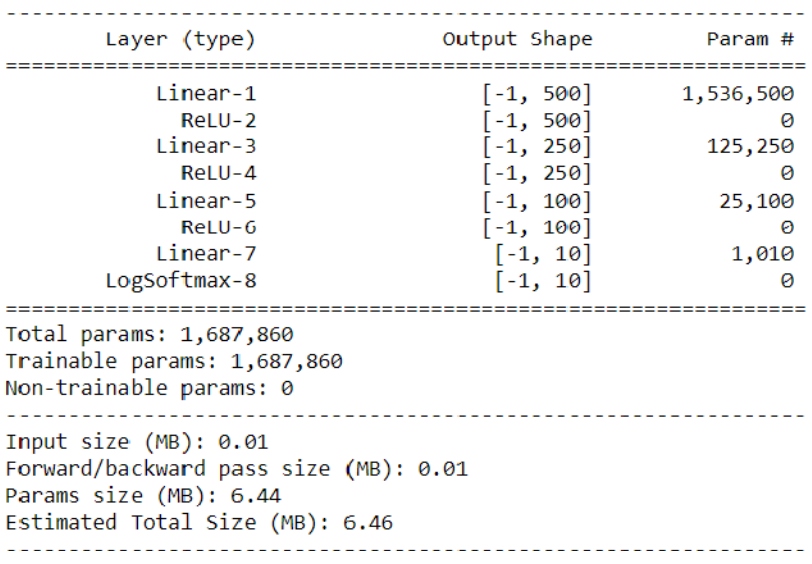
Here we discuss two MLP based models. Both have the same skeleton with the only difference being the use of batch-normalization in one and the absence of the same in the other.

Batch normalization (also known as batch norm) is a method used to make training of artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling. It was proposed by Sergey Ioffe and Christian Szegedy in 2015. Batch normalization also has a slight regularizing effect and greatly helps combat against the phenoenon of **Internal Covariate Shift**.

3.1 MLP Model 1: Without Batch Normalization

This is a simple multi-layer perceptron model with an input layer, 3 hidden layers and an output layer with softmax activation. The MLP takes in one-dimensional input which consists of a 3072 pixel flattened vector of the CIFAR10 images. The number of neurons in the 3 hidden layers are 500, 250 and 100 respectively.

The activation function used in all the linear (fully connected layers) is the **ReLU** (Rectified Linear Unit) activation function. The summary of the model obtained using the `torchsummary.summary()` function is given below:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 500]       1,536,500
              ReLU-2                  [-1, 500]               0
            Linear-3                  [-1, 250]         125,250
              ReLU-4                  [-1, 250]               0
            Linear-5                  [-1, 100]          25,100
              ReLU-6                  [-1, 100]               0
            Linear-7                   [-1, 10]           1,010
        LogSoftmax-8                   [-1, 10]               0
================================================================
Total params: 1,687,860
Trainable params: 1,687,860
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.01
Params size (MB): 6.44
Estimated Total Size (MB): 6.46
----------------------------------------------------------------
```

Figure 1: MLP Without Batch Normalization

3.1.1 Model Training

The model training is done with the a batch size of 64 images per batch. The initial learning rate was experimented with. The model was trained for 15 epochs.

We use a SGD optimizer for the training with a standard momentum value of 0.9 which has not been further tuned. We have also used an exponential learning rate decay scheduling where the learning rate decays by a factor of 0.5 after every 8 epochs. This was done to aid training and allow the model to descend further into the minima.

3.1.2 Hyperparameter Tuning

We carried out hyperparameter tuning on the learning rate and the results comprising of the training validation and test accuracies is given in the following table:

Initial Learning Rate	Train-Accuracy	Val-Accuracy	Test-Accuracy
0.01	9.73	9.72	10.00
0.005	93.87	52.56	53.30
0.001	**99.78**	**53.52**	**54.73**
0.0005	99.01	53.36	54.13

As we can see the **learning rate of 0.001** gives the best results out of all the rest in terms of validation and test accuracy. The metrics plotted belong to the trial with this learning rate.

3.1.3 Plotting Model Metrics

We plot the loss and accuracy curves for the training and the validation sets first:

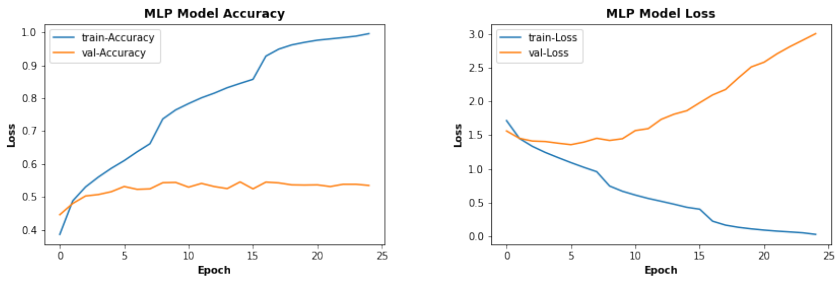


Figure 2: Accuracy and Loss History

Next we plot the confusion matrices for the training, validation and test datasets:

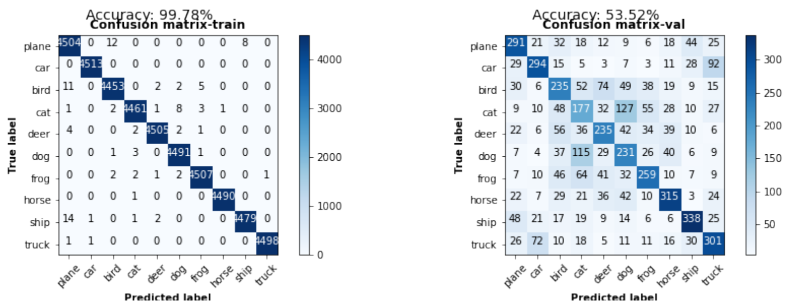


Figure 3: Confusion Matrix - Train Val

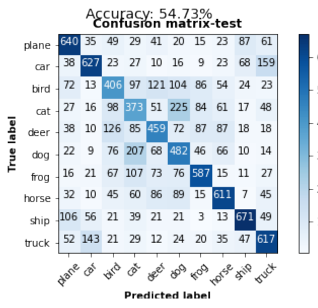


Figure 4: Confusion Matrix - Test

3.1.4 Visualizing the Classification

Here we output the predictions, true labels and the images for a few random samples of the test dataset.

```
True Labels
cat ship ship plane frog frog car frog
Predictions
cat ship ship plane frog frog cat frog
```

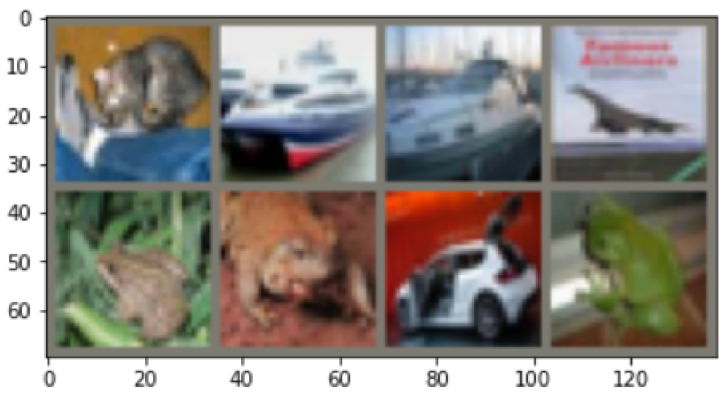
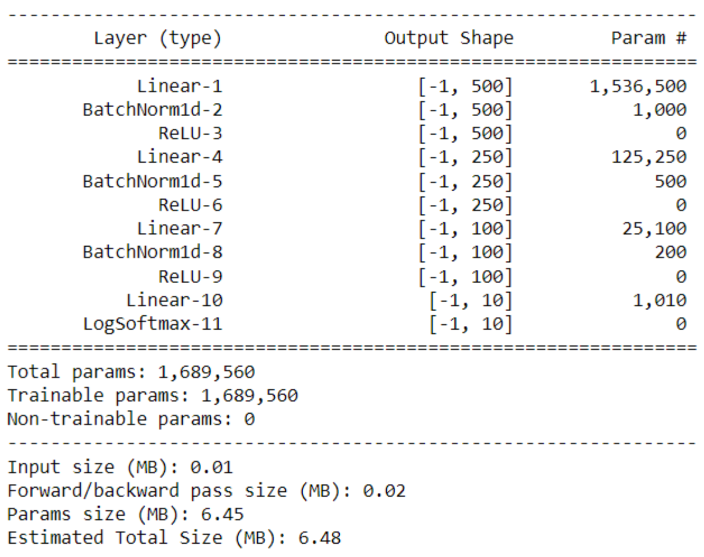


Figure 5: Sample Predictions

3.2 MLP Model 2: With Batch Normalization

This is a simple multi-layer perceptron model with an input layer, 3 hidden layers and an output layer with softmax activation. The MLP takes in one-dimensional input which consists of a 3072 pixel flattened vector of the CIFAR10 images. The number of neurons in the 3 hidden layers are 500, 250 and 100 respectively.

The activation function used in all the linear (fully connected layers) is the **ReLU** (Rectified Linear Unit) activation function. The model also contains a batch normalization applied before the activation function in each layer except the output layer. The summary of the model obtained using the `torchsummary.summary()` function is given below:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 500]       1,536,500
       BatchNorm1d-2                  [-1, 500]           1,000
              ReLU-3                  [-1, 500]               0
            Linear-4                  [-1, 250]         125,250
       BatchNorm1d-5                  [-1, 250]             500
              ReLU-6                  [-1, 250]               0
            Linear-7                  [-1, 100]          25,100
       BatchNorm1d-8                  [-1, 100]             200
              ReLU-9                  [-1, 100]               0
           Linear-10                   [-1, 10]           1,010
       LogSoftmax-11                   [-1, 10]               0
================================================================
Total params: 1,689,560
Trainable params: 1,689,560
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.02
Params size (MB): 6.45
Estimated Total Size (MB): 6.48
```

Figure 6: MLP With Batch Normalization

3.2.1 Model Training

The model training is done with the a batch size of 64 images per batch. The initial learning rate was experimented with. The model was trained for 15 epochs.

We use a SGD optimizer for the training with a standard momentum value of 0.9 which has not been further tuned. We have also used an exponential learning rate decay scheduling where the learning rate decays by a factor of 0.5 after every 8 epochs. This was done to aid training and allow the model to descend further into the minima.

3.2.2 Hyperparameter Tuning

We carried out hyperparameter tuning on the learning rate and the results comprising of the training validation and test accuracies is given in the following table:

Initial Learning Rate	Train-Accuracy	Val-Accuracy	Test-Accuracy
0.01	74.01	55.34	55.88
0.005	**75.06**	**55.10**	**56.65**
0.001	75.08	55.38	55.46
0.0005	74.26	55.24	55.12

Here the results are not so straightforward. We have opted to choose **0.005** as the best learning rate over the value of 0.001 because of a significantly higher test accuracy. Furthermore we can observe that the overall effect of hyperparamter tuning has decreased as a result of batch normalisation.

3.2.3 Plotting Model Metrics

We plot the loss and accuracy curves for the training and validation sets first:

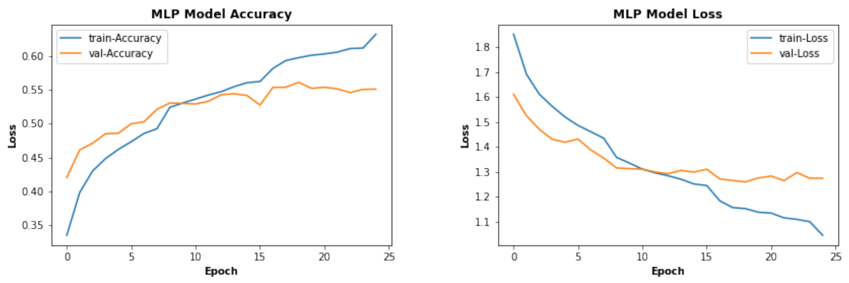


Figure 7: Accuracy and Loss History

Next we plot the confusion matrices for the training, validation and test datasets:

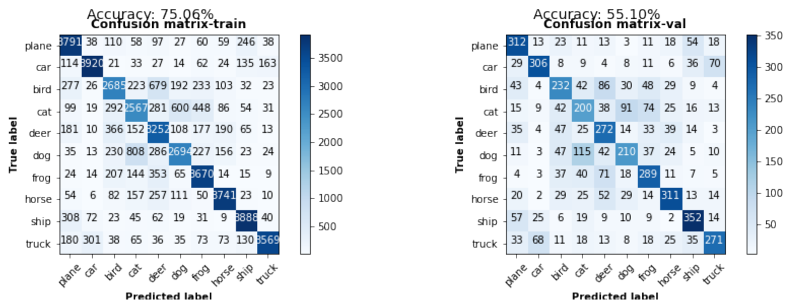


Figure 8: Confusion Matrix - Train Val

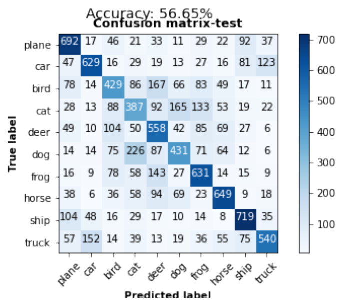


Figure 9: Confusion Matrix - Test

3.2.4 Visualizing the Classification

Here we output the predictions, true labels and the images for a few random samples of the test dataset.

```
True Labels
cat car plane truck dog horse truck ship
Predictions
dog car plane car dog horse truck ship
```

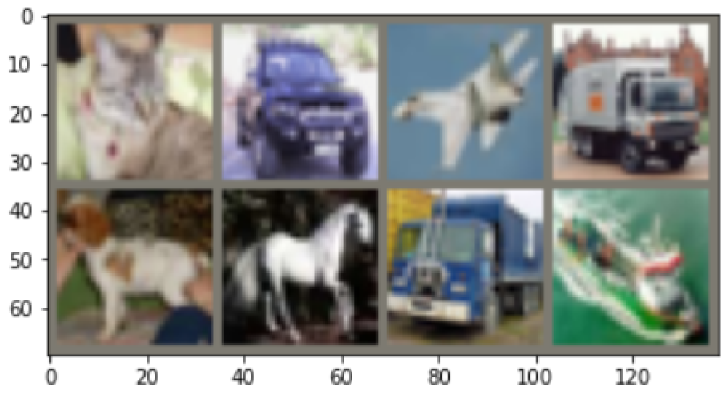


Figure 10: Sample Predictions

3.3 Comparison

The obvious comparison would be that there is an overall regularising effect when we use batch normalization. This can be seen in a significant increase in the test and validation set accuracies and a decrease in the training accuracy for the same number of epochs of training. Furthermore in the initial case of learning rate being set to 0.01, the model without batch normalization collapses completely yielding a test accuracy identical to that of a random classifier. The model with batch normalization doesn't suffer from this issue. Thus we can say that batch normalization somehow stabilizes the training process.

Furthermore adding batch normalization increases the number of trainable parameters by a small amount and similarly the training time for a training schedule of 25 epochs is increased from 9 minutes in the case of model 1 to 11 minutes in the case of model 2 across a number of combinations of hyperparameters

4 CNN Based Model: VGG 11

The VGG11 architecture was proposed by the Visual Geometry Group at Oxford University in the paper: https://arxiv.org/pdf/1409.1556.pdf and is a CNN based deep neural network that performs extremely well on image classification and object localisation tasks. The structure of the image is as given below:

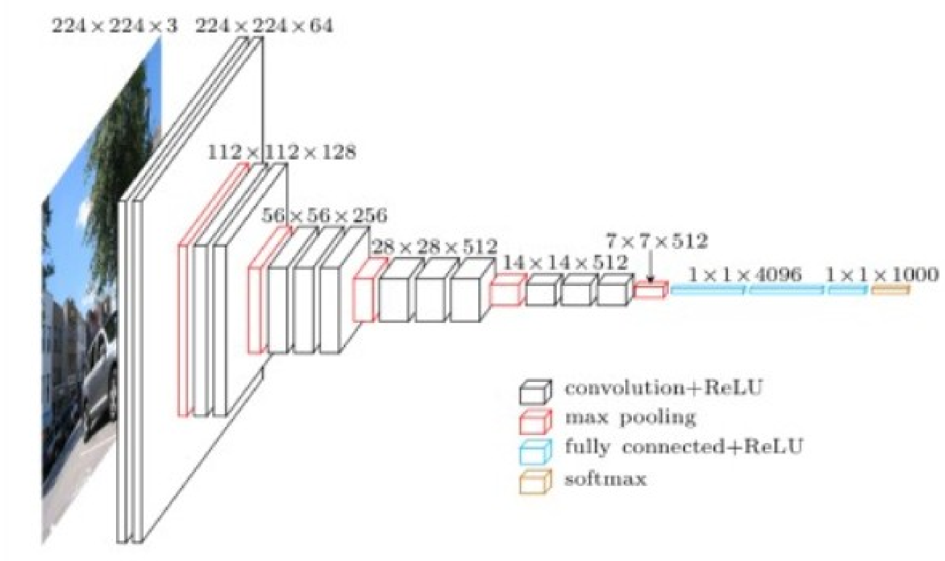


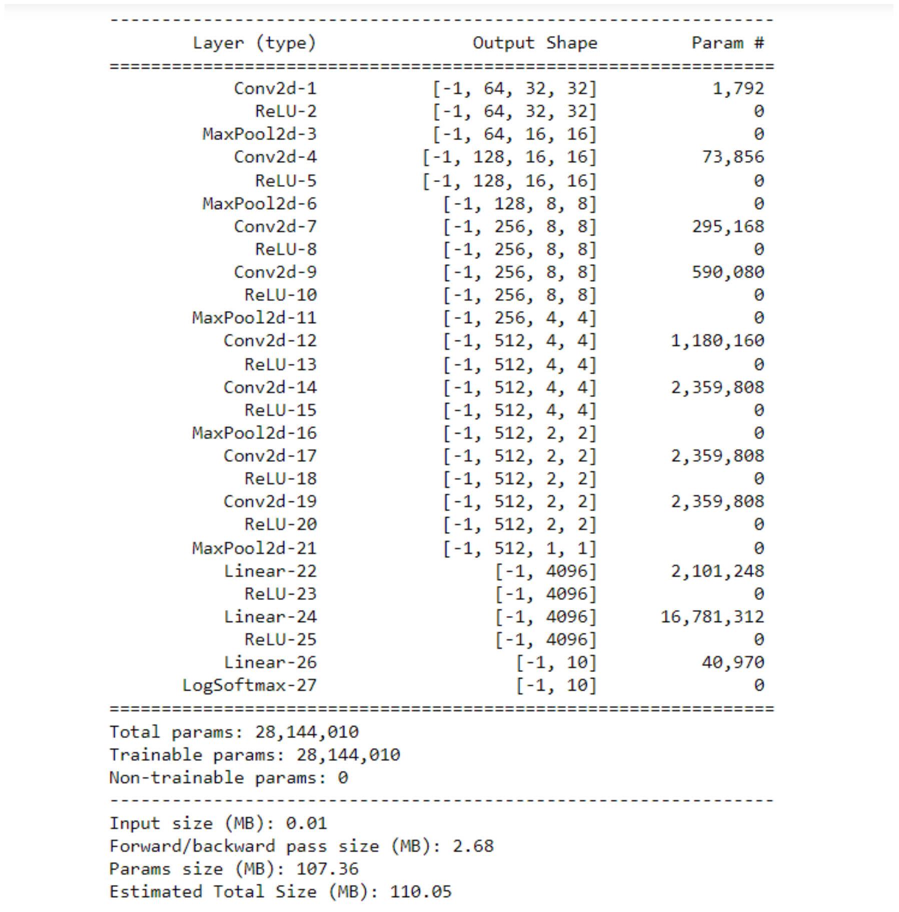
Figure 11: VGG11 Model on a 224x224 image

The VGG11 model makes use of a large number of convolutional layers with a very low convolutional filter size of `3x3` to achieve astounding results in image classification tasks. The major innovation used by group was to, contrary to the LeNet, use these small convolutional filters stacked together along with maxpooling layers to get the best results.

Here we first try to use the VGG11 prepared from scratch to train on the CIFAR10 training set directly. However the results discussed in the next section were slightly disappointing. This led us to modify the model using batch normalization to get a working model with a satisfactory performance.

4.1 VGG11 Without Batch Normalization

This is the simple VGG11 network with the structure given in above. The convolutional filters are all the same. However, since the size of images in the CIFAR dataset (32x32) is different from the expected input of the VGG11 network (224x224) the size of the flattened feature map after the convolutional layers is smaller. This leads to the VGG network having much lesser parameters than when it is used on 224 pixel images. The model is shown below:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
              ReLU-2           [-1, 64, 32, 32]               0
         MaxPool2d-3           [-1, 64, 16, 16]               0
            Conv2d-4          [-1, 128, 16, 16]          73,856
              ReLU-5          [-1, 128, 16, 16]               0
         MaxPool2d-6            [-1, 128, 8, 8]               0
            Conv2d-7            [-1, 256, 8, 8]         295,168
              ReLU-8            [-1, 256, 8, 8]               0
            Conv2d-9            [-1, 256, 8, 8]         590,080
             ReLU-10            [-1, 256, 8, 8]               0
        MaxPool2d-11            [-1, 256, 4, 4]               0
           Conv2d-12            [-1, 512, 4, 4]       1,180,160
             ReLU-13            [-1, 512, 4, 4]               0
           Conv2d-14            [-1, 512, 4, 4]       2,359,808
             ReLU-15            [-1, 512, 4, 4]               0
        MaxPool2d-16            [-1, 512, 2, 2]               0
           Conv2d-17            [-1, 512, 2, 2]       2,359,808
             ReLU-18            [-1, 512, 2, 2]               0
           Conv2d-19            [-1, 512, 2, 2]       2,359,808
             ReLU-20            [-1, 512, 2, 2]               0
        MaxPool2d-21            [-1, 512, 1, 1]               0
           Linear-22                 [-1, 4096]       2,101,248
             ReLU-23                 [-1, 4096]               0
           Linear-24                 [-1, 4096]      16,781,312
             ReLU-25                 [-1, 4096]               0
           Linear-26                   [-1, 10]          40,970
       LogSoftmax-27                   [-1, 10]               0
================================================================
Total params: 28,144,010
Trainable params: 28,144,010
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.68
Params size (MB): 107.36
Estimated Total Size (MB): 110.05
```

Figure 12: VGG11 Without Batch Normalization

4.1.1 Model Training

The model training is done with the a batch size of 64 images per batch. The initial learning rate was experimented with. This model was trained for 15 epochs. We use a SGD optimizer for the training with a learning rate of **0.001** and standard momentum value of 0.9 which has not been further tuned. We opted against using LR decay for the VGG model in order to offer a better comparison of the performance of each learning rate.

However as we observed in the previous case of MLP Model 1 with a learning rate of 0.01, the absence of batch normalization somehow caused the model to completely collapse in the first few training epochs after which the training picked up and reached an overall accuracy of **93.64%** on training set, **77.02%** on validation set and **77.11%** on test set. The results are given below.

4.1.2 Plotting Model Metrics

We plot the loss and accuracy curves for the training and validation sets first:

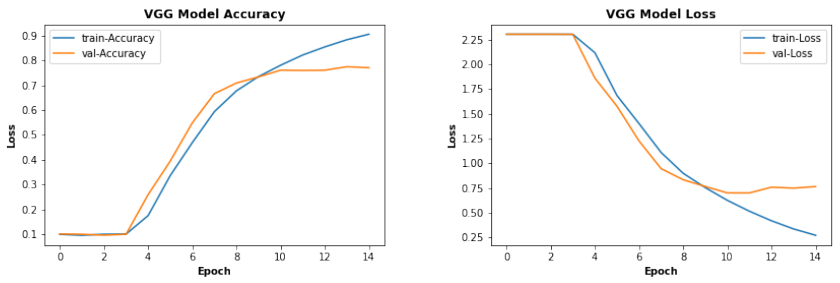
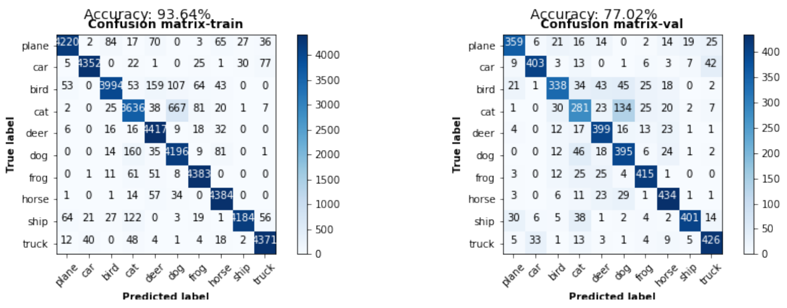


Figure 13: Accuracy and Loss History

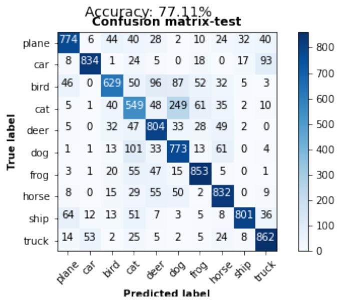
Next we plot the confusion matrices for the training, validation and test datasets:



	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	4220	2	84	17	70	0	3	65	27	36
car	5	4352	0	22	1	0	25	1	30	77
bird	53	0	3994	53	159	107	64	43	0	0
cat	2	0	25	3636	38	667	81	20	1	7
deer	6	0	16	16	4417	9	18	32	0	0
dog	0	0	14	160	35	4196	9	81	0	1
frog	0	1	11	61	51	8	4383	0	0	0
horse	1	0	1	14	57	34	0	4384	0	0
ship	64	21	27	122	0	3	19	1	4184	56
truck	12	40	0	48	4	1	4	18	2	4371

	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	359	6	21	16	14	0	2	14	19	25
car	9	403	3	13	0	1	6	3	7	42
bird	21	1	338	34	43	45	25	18	0	2
cat	1	0	30	281	23	134	25	20	2	7
deer	4	0	12	17	399	16	13	23	1	1
dog	0	0	12	46	18	395	6	24	1	2
frog	3	0	12	25	25	4	415	1	0	0
horse	3	0	6	11	23	29	1	434	1	1
ship	30	6	5	38	1	2	4	2	401	14
truck	5	33	1	13	3	1	4	9	5	426

Figure 14: Confusion Matrix - Train Val



	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	774	6	44	40	28	2	10	24	32	40
car	8	834	1	24	5	0	18	0	17	93
bird	46	0	629	50	96	87	52	32	5	3
cat	5	1	40	549	48	249	61	35	2	10
deer	5	0	32	47	804	33	28	49	2	0
dog	1	1	13	101	33	773	13	61	0	4
frog	3	1	20	55	47	15	853	5	0	1
horse	8	0	15	29	55	50	2	832	0	9
ship	64	12	13	51	7	3	5	8	801	36
truck	14	53	2	25	5	2	5	24	8	862

Figure 15: Confusion Matrix - Test

4.1.3 Visualizing the Classification

Here we output the predictions, true labels and the images for a few random samples of the test dataset.

```
True Labels
dog bird deer plane truck frog frog dog
Predictions
deer bird frog plane bird frog frog bird
```

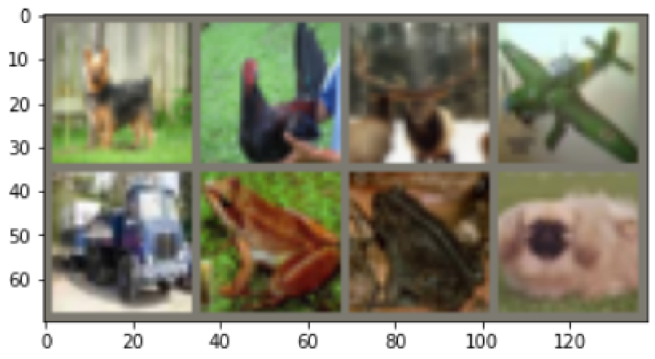


Figure 16: Sample Predictions

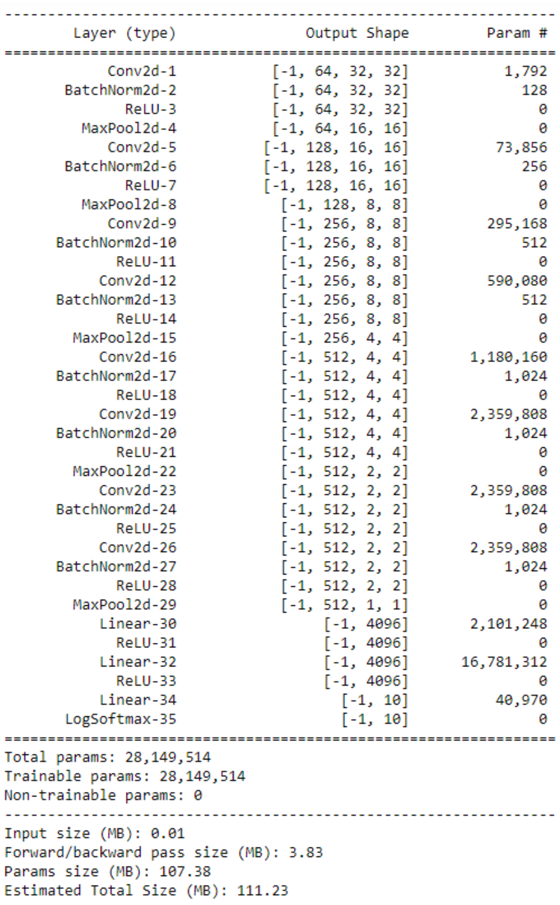
4.2 VGG11 With Batch Normalisation

This is the same VGG11 network that was used in the above example except that each layer in the model has been modified with a batch normalization layer after every convolutional layer. When we do this we can see that the model doesn't collapse for the first few training iterations now. An important point to note is that the VGG model offered in the Pytorch library has batch normalization included. The model is shown in the following page:

4.2.1 Model Training

The model training is done with the a batch size of 64 images per batch. The initial learning rate was experimented with. This model was trained for 15 epochs.

We use a SGD optimizer for the training with a standard momentum value of 0.9 which has not been further tuned. We opted against using LR decay for the VGG model in order to offer a better comparison of the performance of each learning rate.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
         MaxPool2d-4           [-1, 64, 16, 16]               0
            Conv2d-5          [-1, 128, 16, 16]          73,856
       BatchNorm2d-6          [-1, 128, 16, 16]             256
              ReLU-7          [-1, 128, 16, 16]               0
         MaxPool2d-8            [-1, 128, 8, 8]               0
            Conv2d-9            [-1, 256, 8, 8]         295,168
      BatchNorm2d-10            [-1, 256, 8, 8]             512
             ReLU-11            [-1, 256, 8, 8]               0
           Conv2d-12            [-1, 256, 8, 8]         590,080
      BatchNorm2d-13            [-1, 256, 8, 8]             512
             ReLU-14            [-1, 256, 8, 8]               0
        MaxPool2d-15            [-1, 256, 4, 4]               0
           Conv2d-16            [-1, 512, 4, 4]       1,180,160
      BatchNorm2d-17            [-1, 512, 4, 4]           1,024
             ReLU-18            [-1, 512, 4, 4]               0
           Conv2d-19            [-1, 512, 4, 4]       2,359,808
      BatchNorm2d-20            [-1, 512, 4, 4]           1,024
             ReLU-21            [-1, 512, 4, 4]               0
        MaxPool2d-22            [-1, 512, 2, 2]               0
           Conv2d-23            [-1, 512, 2, 2]       2,359,808
      BatchNorm2d-24            [-1, 512, 2, 2]           1,024
             ReLU-25            [-1, 512, 2, 2]               0
           Conv2d-26            [-1, 512, 2, 2]       2,359,808
      BatchNorm2d-27            [-1, 512, 2, 2]           1,024
             ReLU-28            [-1, 512, 2, 2]               0
        MaxPool2d-29            [-1, 512, 1, 1]               0
           Linear-30                 [-1, 4096]       2,101,248
             ReLU-31                 [-1, 4096]               0
           Linear-32                 [-1, 4096]      16,781,312
             ReLU-33                 [-1, 4096]               0
           Linear-34                   [-1, 10]          40,970
       LogSoftmax-35                   [-1, 10]               0
================================================================
Total params: 28,149,514
Trainable params: 28,149,514
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.83
Params size (MB): 107.38
Estimated Total Size (MB): 111.23
```

Figure 17: VGG11 With Batch Normalization

4.2.2 Hyperparameter Tuning

We carried out hyperparameter tuning on the learning rate and the results comprising of the training validation and test accuracies is given in the following table:

Learning Rate	Train-Acc	Val-Acc	Test-Acc	Training Time
0.01	98.66	82.20	81.29	28m 2s
0.001	**99.21**	**81.26**	**81.37**	**28m 31s**
0.0001	99.50	79.04	77.33	28m 17s

Here too the results are not so straightforward. We have opted to choose **0.001** as the best learning rate over the value of 0.01 because of a significantly higher test accuracy. All results plotted are for this value.

4.2.3 Plotting Model Metrics

We plot the loss and accuracy curves for the training and validation sets first:

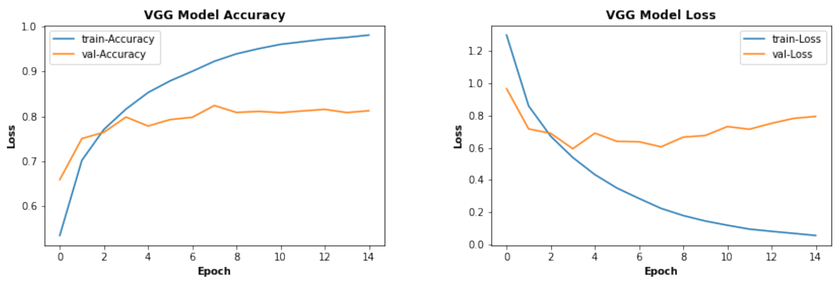


Figure 18: Accuracy and Loss History

Next we plot the confusion matrices for the training, validation and test datasets:

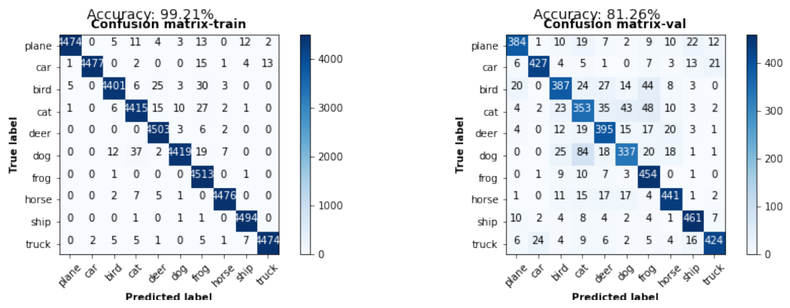


Figure 19: Confusion Matrix - Train Val

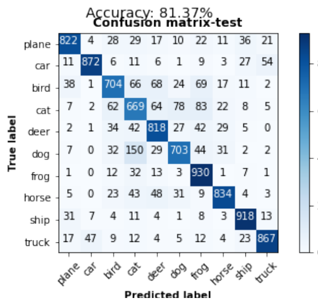


Figure 20: Confusion Matrix - Test

4.2.4 Visualizing the Classification

Here we output the predictions, true labels and the images for a few random samples of the test dataset.

```
True Labels
horse frog truck ship plane cat ship ship
Predictions
horse frog truck ship cat cat ship ship
```

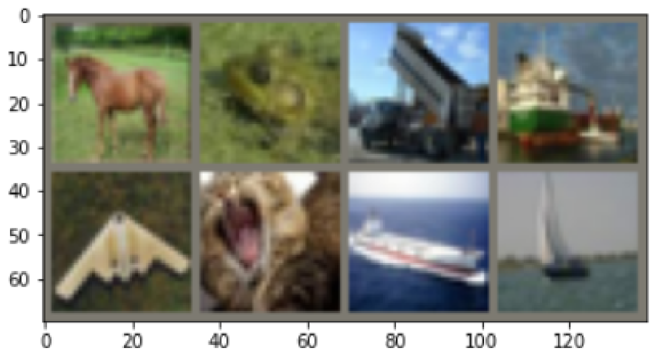


Figure 21: Sample Predictions

4.3 VGG11 Without Final Max Pooling Layer

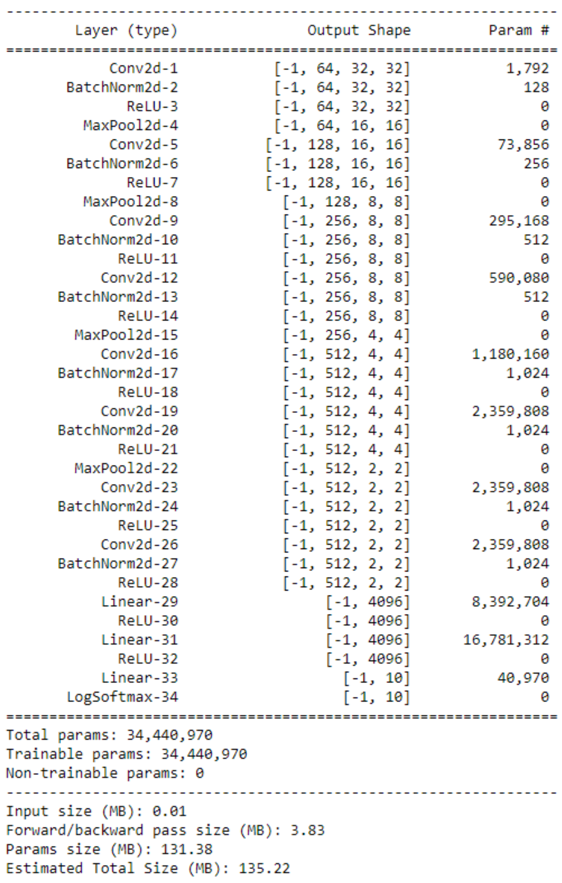
This is the same VGG11 network that was used in the above example except that the last max pooling layer just before the fully connected layers is removed. We retain the batch normalization layers. The number of parameters here is significantly higher than that in previous models since creating a dense connection between the flattened 512*2*2 features output by the convolutional layers and all 4096 neurons in the FC layer is very parameter intensive. The model is shown in the following page:

4.3.1 Model Training

The model training is done with the a batch size of 64 images per batch. The initial learning rate was experimented with. This model was trained for 15 epochs.

We use a SGD optimizer for the training with a standard momentum value of 0.9 which has not been further tuned. We opted against using LR decay for this model in order to offer a better comparison of the performance of each learning rate.

The training here takes approximately 28 to 29 minutes across the various learning rate values.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
         MaxPool2d-4           [-1, 64, 16, 16]               0
            Conv2d-5          [-1, 128, 16, 16]          73,856
       BatchNorm2d-6          [-1, 128, 16, 16]             256
              ReLU-7          [-1, 128, 16, 16]               0
         MaxPool2d-8            [-1, 128, 8, 8]               0
            Conv2d-9            [-1, 256, 8, 8]         295,168
      BatchNorm2d-10            [-1, 256, 8, 8]             512
             ReLU-11            [-1, 256, 8, 8]               0
           Conv2d-12            [-1, 256, 8, 8]         590,080
      BatchNorm2d-13            [-1, 256, 8, 8]             512
             ReLU-14            [-1, 256, 8, 8]               0
        MaxPool2d-15            [-1, 256, 4, 4]               0
           Conv2d-16            [-1, 512, 4, 4]       1,180,160
      BatchNorm2d-17            [-1, 512, 4, 4]           1,024
             ReLU-18            [-1, 512, 4, 4]               0
           Conv2d-19            [-1, 512, 4, 4]       2,359,808
      BatchNorm2d-20            [-1, 512, 4, 4]           1,024
             ReLU-21            [-1, 512, 4, 4]               0
        MaxPool2d-22            [-1, 512, 2, 2]               0
           Conv2d-23            [-1, 512, 2, 2]       2,359,808
      BatchNorm2d-24            [-1, 512, 2, 2]           1,024
             ReLU-25            [-1, 512, 2, 2]               0
           Conv2d-26            [-1, 512, 2, 2]       2,359,808
      BatchNorm2d-27            [-1, 512, 2, 2]           1,024
             ReLU-28            [-1, 512, 2, 2]               0
           Linear-29                 [-1, 4096]       8,392,704
             ReLU-30                 [-1, 4096]               0
           Linear-31                 [-1, 4096]      16,781,312
             ReLU-32                 [-1, 4096]               0
           Linear-33                   [-1, 10]          40,970
       LogSoftmax-34                   [-1, 10]               0
================================================================
Total params: 34,440,970
Trainable params: 34,440,970
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.83
Params size (MB): 131.38
Estimated Total Size (MB): 135.22
```

Figure 22: VGG11 Without Final MaxPooling Layer

4.3.2 Hyperparameter Tuning

We carried out hyperparameter tuning on the learning rate and the results comprising of the training validation and test accuracies is given in the following table:

Learning Rate	Train-Acc	Val-Acc	Test-Acc	Training Time
0.01	97.43	80.88	80.03	29m 14s
0.001	**98.94**	**81.52**	**81.24**	**28m 50s**
0.0001	99.40	78.04	77.95	28m 31s

Here the results are fairly straightforward. We have opted to choose **0.001** as the best learning rate over the other values because of a significantly higher test accuracy. All results plotted are for this value.

4.3.3 Plotting Model Metrics

We plot the loss and accuracy curves for the training and validation sets first:

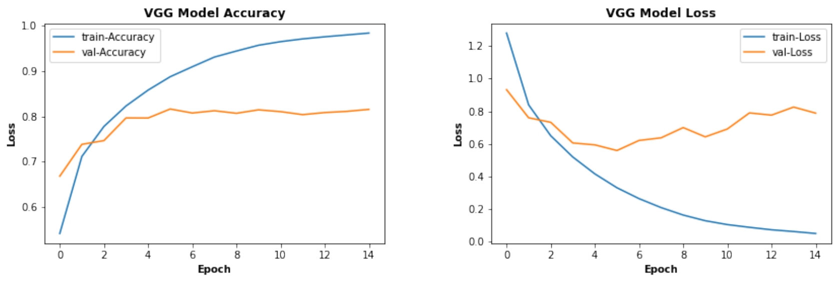


Figure 23: Accuracy and Loss History

Next we plot the confusion matrices for the training, validation and test datasets:

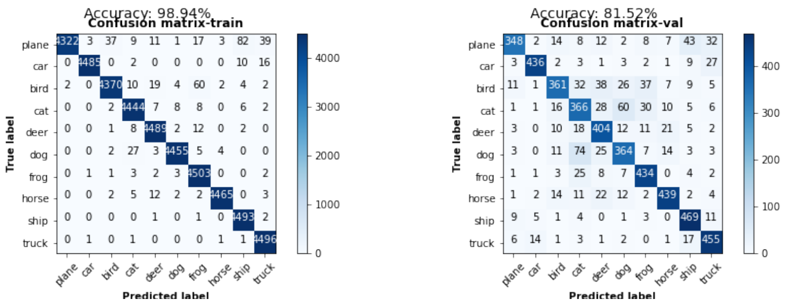


Figure 24: Confusion Matrix - Train Val

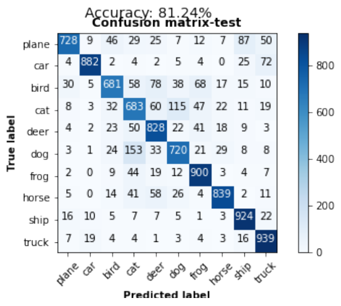


Figure 25: Confusion Matrix - Test

4.3.4 Visualizing the Classification

Here we output the predictions, true labels and the images for a few random samples of the test dataset.

```
True Labels
frog bird car bird cat horse bird frog
Predictions
frog bird car bird dog horse bird frog
```

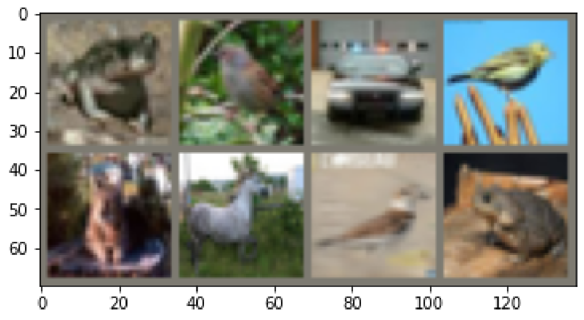


Figure 26: Sample Predictions

4.4 VGG11 Without Fully Connected Layers

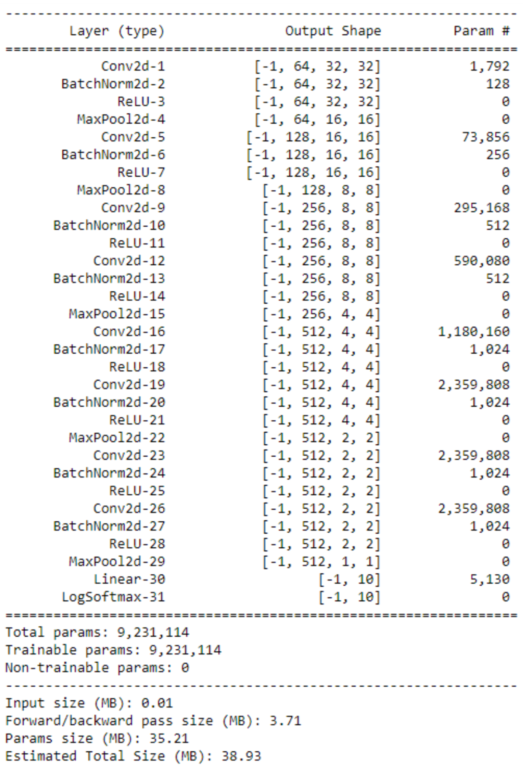
This is the same VGG11 network that was used in the above example except that the fully connected layers are removed all removed except for the output layer. We retain the batch normalization layers as well as the final max pooling layer. The number of parameters here is significantly lower than that in previous models since the fully connected layers contain a major chunk of the parameters in the original VGG11 architecture. The model is shown in the following page:

4.4.1 Model Training

The model training is done with the a batch size of 64 images per batch. The initial learning rate was experimented with. This model was trained for 15 epochs.

We use a SGD optimizer for the training with a standard momentum value of 0.9 which has not been further tuned. We opted against using LR decay for this model in order to offer a better comparison of the performance of each learning rate.

The training here takes approximately 18 minutes across the various learning rate values. This is a significant improvement over the 30 minute time taken for training the other versions of the VGG11 model we have seen so far.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
         MaxPool2d-4           [-1, 64, 16, 16]               0
            Conv2d-5          [-1, 128, 16, 16]          73,856
       BatchNorm2d-6          [-1, 128, 16, 16]             256
              ReLU-7          [-1, 128, 16, 16]               0
         MaxPool2d-8            [-1, 128, 8, 8]               0
            Conv2d-9            [-1, 256, 8, 8]         295,168
      BatchNorm2d-10            [-1, 256, 8, 8]             512
             ReLU-11            [-1, 256, 8, 8]               0
           Conv2d-12            [-1, 256, 8, 8]         590,080
      BatchNorm2d-13            [-1, 256, 8, 8]             512
             ReLU-14            [-1, 256, 8, 8]               0
        MaxPool2d-15            [-1, 256, 4, 4]               0
           Conv2d-16            [-1, 512, 4, 4]       1,180,160
      BatchNorm2d-17            [-1, 512, 4, 4]           1,024
             ReLU-18            [-1, 512, 4, 4]               0
           Conv2d-19            [-1, 512, 4, 4]       2,359,808
      BatchNorm2d-20            [-1, 512, 4, 4]           1,024
             ReLU-21            [-1, 512, 4, 4]               0
        MaxPool2d-22            [-1, 512, 2, 2]               0
           Conv2d-23            [-1, 512, 2, 2]       2,359,808
      BatchNorm2d-24            [-1, 512, 2, 2]           1,024
             ReLU-25            [-1, 512, 2, 2]               0
           Conv2d-26            [-1, 512, 2, 2]       2,359,808
      BatchNorm2d-27            [-1, 512, 2, 2]           1,024
             ReLU-28            [-1, 512, 2, 2]               0
        MaxPool2d-29            [-1, 512, 1, 1]               0
           Linear-30                   [-1, 10]           5,130
       LogSoftmax-31                   [-1, 10]               0
================================================================
Total params: 9,231,114
Trainable params: 9,231,114
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.71
Params size (MB): 35.21
Estimated Total Size (MB): 38.93
```

Figure 27: VGG11 Without Fully Connected Layers

4.4.2 Hyperparameter Tuning

We carried out hyperparameter tuning on the learning rate and the results comprising of the training validation and test accuracies is given in the following table:

Learning Rate	Train-Acc	Val-Acc	Test-Acc	Training Time
0.01	**99.02**	**82.44**	**82.05**	**18m 8s**
0.001	99.30	82.22	81.24	18m 6s
0.0001	99.65	78.04	77.72	19m 52s

Here the results are fairly straightforward. We have opted to choose **0.01** as the best learning rate over the other values because of a significantly higher test accuracy.

As we can see this model is the only one of the large number of configurations tested that has breaks the test accuracy barrier of 82%. Furthermore the removal of the fully connected layers speeds up the average training time by over **35%**. This is a significant improvement in performance and efficiency of the model.

4.4.3 Plotting Model Metrics

We plot the loss and accuracy curves for the training and validation sets first:

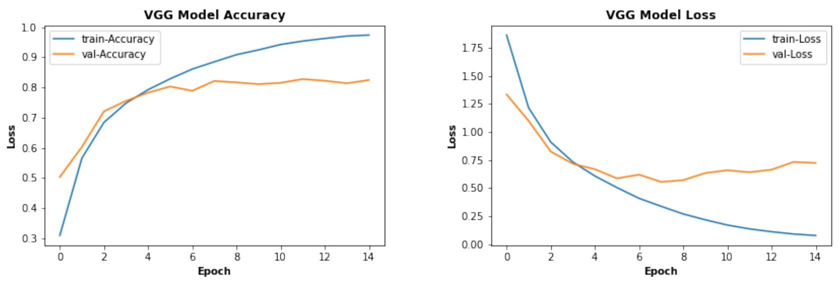


Figure 28: Accuracy and Loss History

Next we plot the confusion matrices for the training, validation and test datasets:

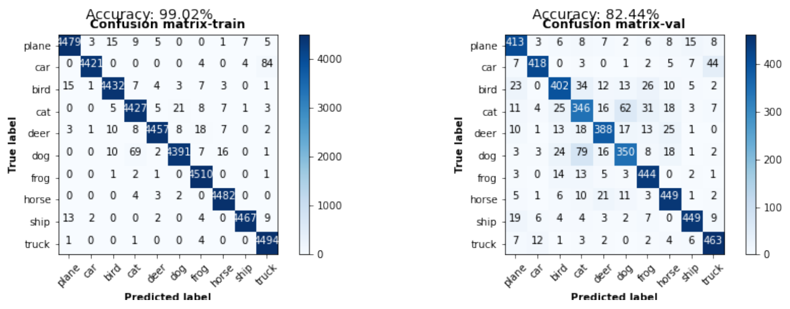


Figure 29: Confusion Matrix - Train Val

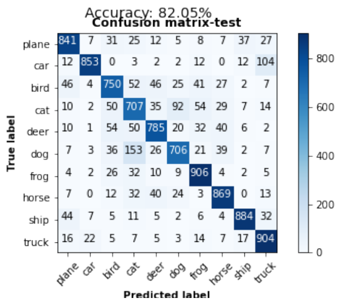
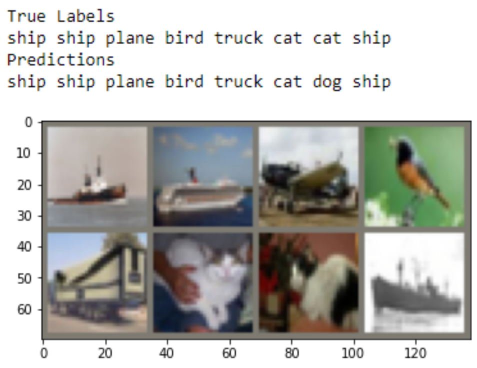


Figure 30: Confusion Matrix - Test

4.4.4 Visualizing the Classification

Here we output the predictions, true labels and the images for a few random samples of the test dataset.



```
True Labels
ship ship plane bird truck cat cat ship
Predictions
ship ship plane bird truck cat dog ship
```

Figure 31: Sample Predictions

4.5 Comparison of Models

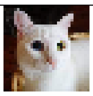
As we discussed earlier the bare VGG11 model with no batch normalization is inadequate as it suffers from a vanishing gradient problem in most layers during the first few epochs. The VGG11 model greatly improves its performance when all the convolutional layers are supplemented with batch normalization. In this case the test accuracy greatly improves over all values of learning rates. When we experiment by removing the final max pooling layer the difference in the test accuracy is not very significant.

The greatest improvement is seen when the fully connected layers at the end of the VGG11 model are removed (except for the final layer). When the features output by the convolutional layers are directly used by the model to classify the image the accuracy in the test set is significantly improved.

Across the models which were tested it was observed that the ideal learning rate was 0.01 to 0.001. When the learning rate is set to 0.01 the model reaches a saturation after around 10 epochs after which the validation accuracy doesn't improve significantly. It is therefore proposed (but not tested) that the model be run with a learning rate of 0.01 for 10 epochs before decaying it to 0.001 for the next 5 epochs. Overall 15 epochs seems to be enough to train the model.

4.6 Custom Image Testing

We now test our trained model on 32x32 images from the internet. In order to resize higher resolution images into 32x32 pixels we use the online tool here. We have tested the model on some images.. The images have been prepared such that the names of the images are the corresponding indexes of the class as per the CIFAR dataset. Thus there is no need to have a further annotated file with label information given. The results are given below:

Custom Test Image	Predicted Label	True Label
	Airplane	Airplane
	Truck	Car
	Bird	Bird
	Cat	Cat
	Deer	Deer
	Dog	Dog
	Frog	Frog
	Dog	Horse
	Ship	Ship
	Truck	Truck

5 Device Information

All the training and testing tasks were run on my personal machine with a **AMD Ryzen 9 - 5900HS** processor and a **NVIDIA RTX 3050** Laptop GPU with CUDA enabled.