HW6 – Part 2: Neural Network on Image Classification

Nafis Neehal

661990881

Code Link in Colab:

https://colab.research.google.com/drive/1-S75WAemnDE-knabrNqouwwWgVz6qjcL

Data Description

Table 1: Data Description

Total training data provided	4000
Total test data provided	1000
Validation to Training Ratio	12.5%
Final number of Validation Data	500
Final number of Training Data	3500
Mini Batch Size (for train, validation, test)	64
Data shape	64*3*60*90 (aspect ratio 1.5 = height/width)

- ✓ I would have gone for the standard 60-20-20 (3k-1k-1k for this dataset) split for train-validation-test. But the number of training examples are inadequate to gain a good accuracy without the network overfitting with only 3k training examples. So, I tried to maximize the number of training examples as much as I could and didn't want to manipulate the test set at all. It might have hampered the validation legitimacy a bit, but at least the network is not that overfitted.
- ✓ I explored data-augmentation for increasing the number of training examples to reduce the effect of overfitting and also introducing the 60-20-20 split, but as I don't have any GPU in my local machine, it was a bit tough to do that using Google Colab. I tried doing it in runtime using "torchvision.transforms" and tried just introducing Horizontal and Vertical Flips, but it ended up consuming a huge amount of time.
- ✓ I applied the following transformations on each data –
    - Resize each image to (3*60*90) maintain aspect ratio of 1.5
    - Converted each image array to tensors
    - Normalize the images
- ✓ I downsampled the data 2 times (one-fourth the size) and got the best performance that I could I achieve from both FCNN and CNN.
- ✓ I normalized the data by setting the mean to [0.5, 0.5, 0.5] and std to [0.5, 0.5, 0.5]. I found this value works best in several articles and literature.
- ✓ I used "torch.randperm" to shuffle all the data before assigning it to training/validation set with seed value = 1. Maybe trying some different seed values would have been another parameter search task, but I just don't have the computation power to do so, and so couldn't tweak and analyze different seed values.
- ✓ I used torchvision.datasets.ImageFolder to load my custom image dataset from Google Drive.

- ✓ I used torchvision.utils.data.DataLoader with num_workers = 4 for parallel data transfer with the setting pin_memory=True which also helped with faster data transfer between CPU and GPU (suggestion from NVidea developer blog and PyTorch forum)
- ✓ I loaded all my data from Google drive after I mounted the drive in Colab.

Network Common Parameters

Table 2: Common Network Parameters

Number of classes	5
Model Name	NN / CNN
Number of Epochs	50
Learning Rate	1e-3
Device	GPU (if available) / CPU
Optimizer	Adam
L2 Regularization with Weight Decay	1e-5
Loss Function	Cross Entropy Loss

Overall issues and general brief process of Train, Validation and Test:

- ✓ In each epoch, I trained the data with all the mini-batches (size = 64), then validate the performance on the validation data. Validation could have been done periodically (i.e., after each 5 epoch), but I did in each epoch to track my model's performance more accurately, specially, it helped tackling the overfitting.
- ✓ I initially cleared all the gradients after the loss was propagated backwards which literally skyrocketed the gradients. Later I fixed it by calling the optimizer.zero_grad() before loss was propagated.
- ✓ No gradient update or loss propagate in validation / test; while model.eval() is on.
- ✓ Pushed all data tensors and my network model to GPU. Used Tesla K-80 (Google Colab's default).
- ✓ In each epoch, kept the track of the best model. If my models validation accuracy increased and validation loss decreased than the current best saved values(highest validation accuracy and lowest loss), I saved it as best model using torch.save() with all parameter states of the model and the optimizer. Then while running final test on the test dataset, I loaded that best model and use that to evaluate the final performance of my network on test data. The epoch value of the final best model state can be used for the threshold for early stopping in order to overfit the network.

General Measures to reduce overfitting

- ✓ L2 regularization with weight decay 1e-5
- ✓ Dropout (p = 0.5)
- ✓ Tried Early stopping as well, but this did not fit well as the model tend to overfit very early (only after 5-6 epochs)
- ✓ Tried to reduce overfitting also by reducing the model complexity (simpler architecture)

HW6 – Part 2(a): Fully Connected Neural Network on Image Classification

Network Architecture:

This image is a rough representation of the network. Exact number of nodes were not possible to show here.

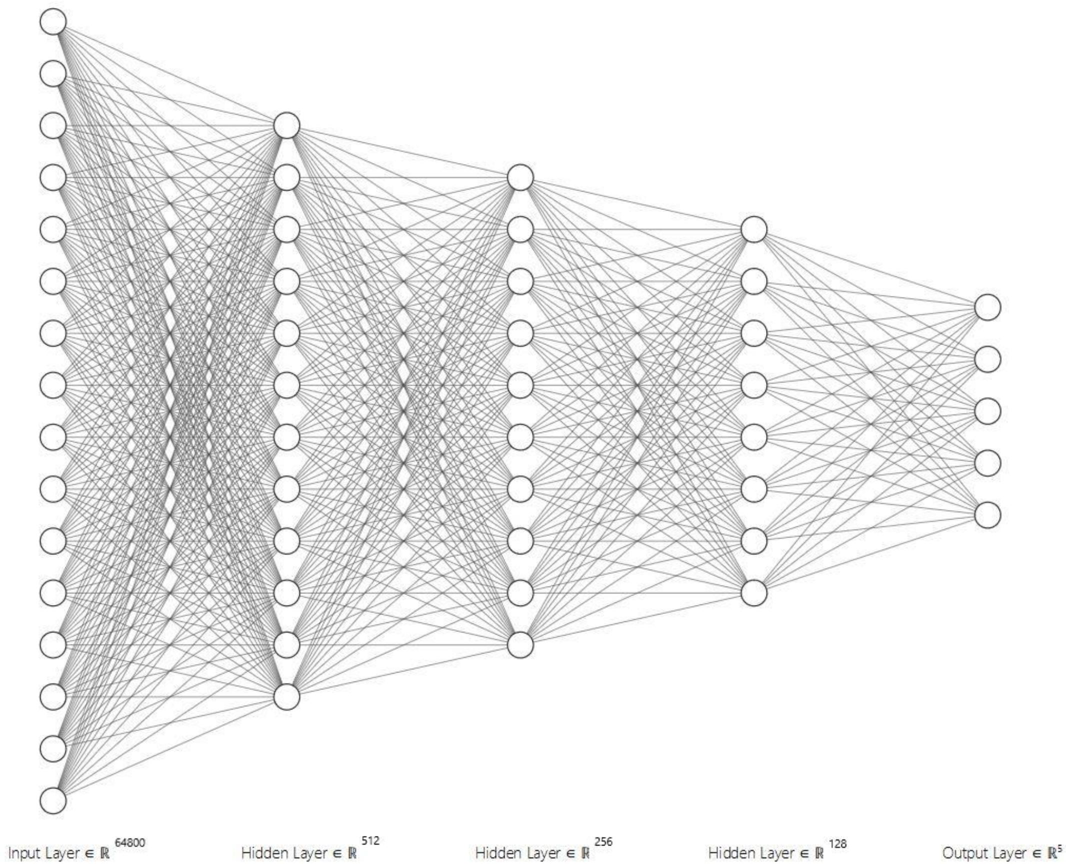

Fig 1: FCNN Representation

Network Description:

Table 3: FCNN Network Architecture

Layer	Activation	Number of Weights	Number of Biases	Dimension
Input	None	0	0	1x16200 (3*60*90)
Hidden 1	ReLU	16200*512	512	1x512
Hidden 2	ReLU	512*256	256	1x256
Hidden 3	ReLU	256*128	128	1x128
Output	LogSoftMax	128*5	5	1x5

- ✓ I used dropout with P=0.5 between all the layers (H1-H2, H2-H3, H3-Output). Helped me with the overfitting. I tried reducing one hidden layer to make the architecture simpler, but then the validation and test accuracy gets severely affected.
- ✓ I explored LeakyReLU activation too for hidden layers, but ReLU performed better.
- ✓ I used CrossEntropyLoss for loss function, and torch.nn.CrossEntropyLoss already contains nn.LogSoftMax which was by default applied on the output of the output layer, that is why I explicitly didn't use any other SoftMax.

<u>Performance Analysis:</u>

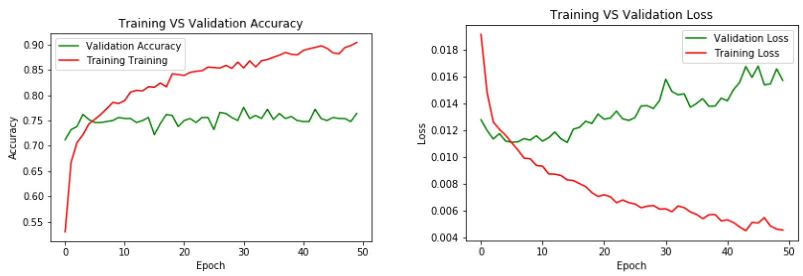

Fig 2: Training and Validation Data in Graph for FCNN

```
Epoch[1/50] Training Accuracy: 0.5300 Training Loss 0.0192 Validation Accuracy: 0.7120 Validation Loss: 0.0128
Epoch[2/50] Training Accuracy: 0.6677 Training Loss 0.0148 Validation Accuracy: 0.7320 Validation Loss: 0.0120
Epoch[3/50] Training Accuracy: 0.7066 Training Loss 0.0126 Validation Accuracy: 0.7380 Validation Loss: 0.0114
Epoch[4/50] Training Accuracy: 0.7220 Training Loss 0.0121 Validation Accuracy: 0.7620 Validation Loss: 0.0118
Epoch[5/50] Training Accuracy: 0.7434 Training Loss 0.0116 Validation Accuracy: 0.7520 Validation Loss: 0.0112
Epoch[6/50] Training Accuracy: 0.7526 Training Loss 0.0111 Validation Accuracy: 0.7460 Validation Loss: 0.0111
Epoch[7/50] Training Accuracy: 0.7623 Training Loss 0.0105 Validation Accuracy: 0.7460 Validation Loss: 0.0111
Epoch[8/50] Training Accuracy: 0.7737 Training Loss 0.0099 Validation Accuracy: 0.7480 Validation Loss: 0.0114
Epoch[9/50] Training Accuracy: 0.7857 Training Loss 0.0099 Validation Accuracy: 0.7500 Validation Loss: 0.0113
Epoch[10/50] Training Accuracy: 0.7837 Training Loss 0.0094 Validation Accuracy: 0.7560 Validation Loss: 0.0116
Epoch[11/50] Training Accuracy: 0.7894 Training Loss 0.0093 Validation Accuracy: 0.7540 Validation Loss: 0.0112
Epoch[12/50] Training Accuracy: 0.8063 Training Loss 0.0088 Validation Accuracy: 0.7540 Validation Loss: 0.0114
Epoch[13/50] Training Accuracy: 0.8097 Training Loss 0.0087 Validation Accuracy: 0.7460 Validation Loss: 0.0119
Epoch[14/50] Training Accuracy: 0.8086 Training Loss 0.0086 Validation Accuracy: 0.7500 Validation Loss: 0.0114
Epoch[15/50] Training Accuracy: 0.8169 Training Loss 0.0083 Validation Accuracy: 0.7560 Validation Loss: 0.0111
Epoch[16/50] Training Accuracy: 0.8157 Training Loss 0.0083 Validation Accuracy: 0.7220 Validation Loss: 0.0121
Epoch[17/50] Training Accuracy: 0.8243 Training Loss 0.0080 Validation Accuracy: 0.7440 Validation Loss: 0.0122
Epoch[18/50] Training Accuracy: 0.8166 Training Loss 0.0078 Validation Accuracy: 0.7620 Validation Loss: 0.0127
Epoch[19/50] Training Accuracy: 0.8423 Training Loss 0.0074 Validation Accuracy: 0.7600 Validation Loss: 0.0125
Epoch[20/50] Training Accuracy: 0.8409 Training Loss 0.0071 Validation Accuracy: 0.7380 Validation Loss: 0.0132
Epoch[21/50] Training Accuracy: 0.8391 Training Loss 0.0072 Validation Accuracy: 0.7500 Validation Loss: 0.0128
Epoch[22/50] Training Accuracy: 0.8451 Training Loss 0.0071 Validation Accuracy: 0.7540 Validation Loss: 0.0129
Epoch[23/50] Training Accuracy: 0.8474 Training Loss 0.0066 Validation Accuracy: 0.7460 Validation Loss: 0.0134
Epoch[24/50] Training Accuracy: 0.8486 Training Loss 0.0068 Validation Accuracy: 0.7560 Validation Loss: 0.0129
Epoch[25/50] Training Accuracy: 0.8557 Training Loss 0.0066 Validation Accuracy: 0.7560 Validation Loss: 0.0127
Epoch[26/50] Training Accuracy: 0.8549 Training Loss 0.0065 Validation Accuracy: 0.7320 Validation Loss: 0.0129
Epoch[27/50] Training Accuracy: 0.8537 Training Loss 0.0062 Validation Accuracy: 0.7660 Validation Loss: 0.0138
Epoch[28/50] Training Accuracy: 0.8591 Training Loss 0.0064 Validation Accuracy: 0.7640 Validation Loss: 0.0138
Epoch[29/50] Training Accuracy: 0.8531 Training Loss 0.0064 Validation Accuracy: 0.7560 Validation Loss: 0.0136
Epoch[30/50] Training Accuracy: 0.8654 Training Loss 0.0061 Validation Accuracy: 0.7500 Validation Loss: 0.0142
Epoch[31/50] Training Accuracy: 0.8540 Training Loss 0.0062 Validation Accuracy: 0.7760 Validation Loss: 0.0158
Epoch[32/50] Training Accuracy: 0.8680 Training Loss 0.0059 Validation Accuracy: 0.7540 Validation Loss: 0.0149
Epoch[33/50] Training Accuracy: 0.8560 Training Loss 0.0064 Validation Accuracy: 0.7600 Validation Loss: 0.0147
Epoch[34/50] Training Accuracy: 0.8683 Training Loss 0.0062 Validation Accuracy: 0.7540 Validation Loss: 0.0147
Epoch[35/50] Training Accuracy: 0.8706 Training Loss 0.0059 Validation Accuracy: 0.7720 Validation Loss: 0.0137
Epoch[36/50] Training Accuracy: 0.8751 Training Loss 0.0057 Validation Accuracy: 0.7520 Validation Loss: 0.0140
Epoch[37/50] Training Accuracy: 0.8791 Training Loss 0.0054 Validation Accuracy: 0.7640 Validation Loss: 0.0144
Epoch[38/50] Training Accuracy: 0.8846 Training Loss 0.0057 Validation Accuracy: 0.7540 Validation Loss: 0.0138
Epoch[39/50] Training Accuracy: 0.8809 Training Loss 0.0057 Validation Accuracy: 0.7580 Validation Loss: 0.0138
Epoch[40/50] Training Accuracy: 0.8797 Training Loss 0.0053 Validation Accuracy: 0.7500 Validation Loss: 0.0144
Epoch[41/50] Training Accuracy: 0.8886 Training Loss 0.0053 Validation Accuracy: 0.7480 Validation Loss: 0.0142
Epoch[42/50] Training Accuracy: 0.8920 Training Loss 0.0051 Validation Accuracy: 0.7480 Validation Loss: 0.0150
Epoch[43/50] Training Accuracy: 0.8946 Training Loss 0.0048 Validation Accuracy: 0.7720 Validation Loss: 0.0156
Epoch[44/50] Training Accuracy: 0.8977 Training Loss 0.0045 Validation Accuracy: 0.7540 Validation Loss: 0.0168
Epoch[45/50] Training Accuracy: 0.8929 Training Loss 0.0051 Validation Accuracy: 0.7500 Validation Loss: 0.0159
Epoch[46/50] Training Accuracy: 0.8837 Training Loss 0.0051 Validation Accuracy: 0.7560 Validation Loss: 0.0168
Epoch[47/50] Training Accuracy: 0.8820 Training Loss 0.0055 Validation Accuracy: 0.7540 Validation Loss: 0.0154
Epoch[48/50] Training Accuracy: 0.8943 Training Loss 0.0049 Validation Accuracy: 0.7540 Validation Loss: 0.0155
Epoch[49/50] Training Accuracy: 0.8983 Training Loss 0.0046 Validation Accuracy: 0.7480 Validation Loss: 0.0166
Epoch[50/50] Training Accuracy: 0.9046 Training Loss 0.0046 Validation Accuracy: 0.7640 Validation Loss: 0.0157
```

Fig 3: Training and Validation Log for FCNN

From the very first sight, it seems obvious that the network is overfitted, at least that's what the graph says. Validation accuracy almost flattens after a certain number of epochs and the trend of the graph of validation loss is going upwards. But if we closely observe the rate of increase in validation loss, it is very negligible.

Best result –

Table 4: Final FCNN Performance

Epoch No	15
Training Accuracy	81.685%
Train Loss	0.0083
Validation Accuracy	75.60%
Validation Loss	0.0111
Test Accuracy	77.70%

Table 5: Classwise Accuracy

Class Name	Accuracy on Test Data
Grass	76%
Ocean	58%
Red carpet	92%
Road	75%
Wheatfield	70%

The reason that the train accuracy is lower than typical rate (above 95% in overfitted networks) is because of the dropout in between the FC layers. Initially, when I trained and validated the network for the first time, the train accuracy was around 98% and validation was around 65% and the validation loss was skyrocketing while train loss was going downwards! So, by tuning the parameters in order to trying and balance the network overfitting, I was able to increase the result up to around 12%.

Although, from the result using the model I chose as best is not that impressive, but the result is at least balanced and states the fact that the model is not overfitted. Epoch No 15 represents here that this is the threshold for "Early Stopping" after which network starts to perform badly. After this point, the validation loss starts to go upwards and validation accuracy flattens with spikes (accuracy fluctuation because of using dropout).

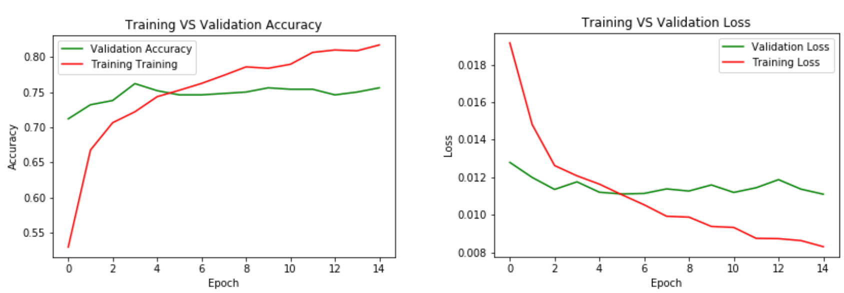

Fig 4: Training and Validation Log for FCNN up to early stopping threshold (Epoch 15)

HW6 – Part 2(b): Convolutional Neural Network on Image Classification

Network Architecture:

This image is a rough representation of the network. Exact number of nodes were not possible to show here.

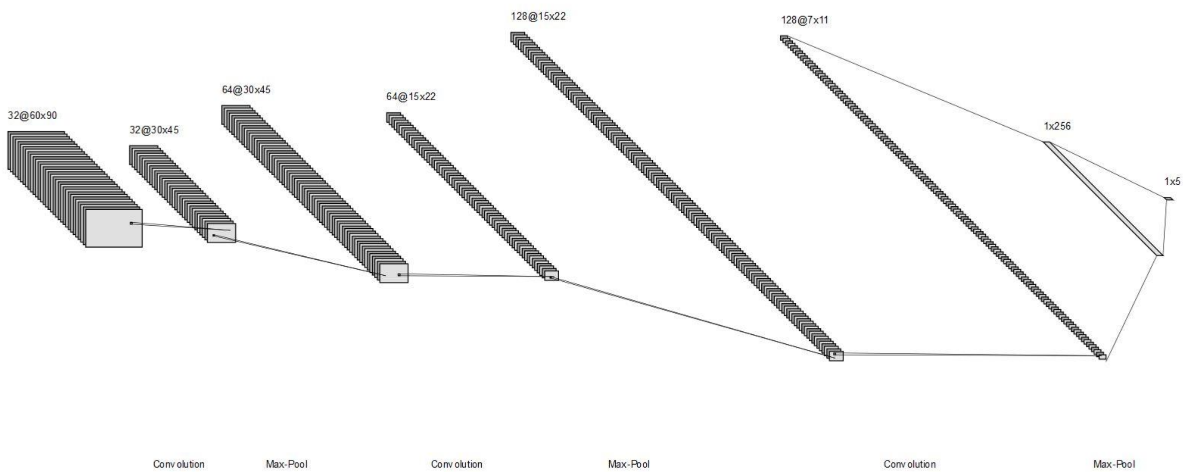

Fig 5: Training and Validation Log for FCNN up to early stopping threshold (Epoch 15)

Network Description:

Table 6: CNN Network Description

Layer	Unit	Input Shape (batch_size*depth*height*width)	Output Shape (batch_size*depth*height*width)	Filter size
Convolution	1	32x3x60x90	32x32x60x90	3x3
Max-Pool		32x32x60x90	32x32x30x45	2x2
Convolution	2	32x32x30x45	32x64x30x45	3x3
Max-Pool		32x64x30x45	32x64x15x22	2x2
Convolution	3	32x64x15x22	32x128x15x22	3x3
Max-Pool		32x128x15x22	32x128x7x11	2x2
Dense-1	4	32x128x7x11	32x1x256x1	N/A
Dense-2	5	32x1x256x1	32x1x5x1	N/A

I have also used –

- Dropout(p=0.5) before each of the Dense layers. I explored with p=0.2 and p=0.8, but p=0.5 gave the optimum result
- There is batch-normalization after each Convolution layer in each of the first 3 units (ConvUnits).
- I used 3x3 filters for Convolution and 2x2 filters for Max-Pool. I would have explored more with different filter size but had limitation in computational resource.
- ReLU was applied after each BatchNorm was applied.

Performance Analysis:

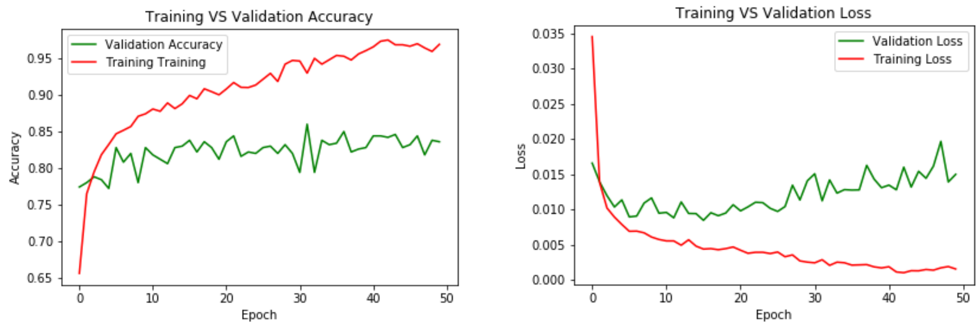

Fig 6: Training and Validation Data in Graph for CNN

```
Epoch[1/50] Training Accuracy: 0.6557 Training Loss 0.0345 Validation Accuracy: 0.7740 Validation Loss: 0.0166
Epoch[2/50] Training Accuracy: 0.7649 Training Loss 0.0139 Validation Accuracy: 0.7800 Validation Loss: 0.0139
Epoch[3/50] Training Accuracy: 0.7946 Training Loss 0.0102 Validation Accuracy: 0.7880 Validation Loss: 0.0120
Epoch[4/50] Training Accuracy: 0.8183 Training Loss 0.0089 Validation Accuracy: 0.7840 Validation Loss: 0.0104
Epoch[5/50] Training Accuracy: 0.8323 Training Loss 0.0079 Validation Accuracy: 0.7720 Validation Loss: 0.0114
Epoch[6/50] Training Accuracy: 0.8469 Training Loss 0.0069 Validation Accuracy: 0.8280 Validation Loss: 0.0090
Epoch[7/50] Training Accuracy: 0.8517 Training Loss 0.0069 Validation Accuracy: 0.8080 Validation Loss: 0.0091
Epoch[8/50] Training Accuracy: 0.8569 Training Loss 0.0067 Validation Accuracy: 0.8200 Validation Loss: 0.0109
Epoch[9/50] Training Accuracy: 0.8709 Training Loss 0.0061 Validation Accuracy: 0.7800 Validation Loss: 0.0117
Epoch[10/50] Training Accuracy: 0.8743 Training Loss 0.0058 Validation Accuracy: 0.8280 Validation Loss: 0.0095
Epoch[11/50] Training Accuracy: 0.8809 Training Loss 0.0056 Validation Accuracy: 0.8180 Validation Loss: 0.0096
Epoch[12/50] Training Accuracy: 0.8777 Training Loss 0.0056 Validation Accuracy: 0.8120 Validation Loss: 0.0088
Epoch[13/50] Training Accuracy: 0.8891 Training Loss 0.0049 Validation Accuracy: 0.8060 Validation Loss: 0.0111
Epoch[14/50] Training Accuracy: 0.8814 Training Loss 0.0057 Validation Accuracy: 0.8280 Validation Loss: 0.0095
Epoch[15/50] Training Accuracy: 0.8880 Training Loss 0.0048 Validation Accuracy: 0.8300 Validation Loss: 0.0094
Epoch[16/50] Training Accuracy: 0.8994 Training Loss 0.0044 Validation Accuracy: 0.8380 Validation Loss: 0.0085
Epoch[17/50] Training Accuracy: 0.8949 Training Loss 0.0045 Validation Accuracy: 0.8220 Validation Loss: 0.0096
Epoch[18/50] Training Accuracy: 0.9086 Training Loss 0.0043 Validation Accuracy: 0.8360 Validation Loss: 0.0091
Epoch[19/50] Training Accuracy: 0.9046 Training Loss 0.0045 Validation Accuracy: 0.8280 Validation Loss: 0.0095
Epoch[20/50] Training Accuracy: 0.9003 Training Loss 0.0047 Validation Accuracy: 0.8120 Validation Loss: 0.0107
Epoch[21/50] Training Accuracy: 0.9083 Training Loss 0.0042 Validation Accuracy: 0.8360 Validation Loss: 0.0098
Epoch[22/50] Training Accuracy: 0.9171 Training Loss 0.0038 Validation Accuracy: 0.8440 Validation Loss: 0.0104
Epoch[23/50] Training Accuracy: 0.9106 Training Loss 0.0039 Validation Accuracy: 0.8160 Validation Loss: 0.0110
Epoch[24/50] Training Accuracy: 0.9103 Training Loss 0.0039 Validation Accuracy: 0.8220 Validation Loss: 0.0110
Epoch[25/50] Training Accuracy: 0.9137 Training Loss 0.0038 Validation Accuracy: 0.8200 Validation Loss: 0.0102
Epoch[26/50] Training Accuracy: 0.9217 Training Loss 0.0040 Validation Accuracy: 0.8280 Validation Loss: 0.0097
Epoch[27/50] Training Accuracy: 0.9297 Training Loss 0.0033 Validation Accuracy: 0.8300 Validation Loss: 0.0104
Epoch[28/50] Training Accuracy: 0.9186 Training Loss 0.0036 Validation Accuracy: 0.8200 Validation Loss: 0.0135
Epoch[29/50] Training Accuracy: 0.9423 Training Loss 0.0027 Validation Accuracy: 0.8320 Validation Loss: 0.0114
Epoch[30/50] Training Accuracy: 0.9474 Training Loss 0.0026 Validation Accuracy: 0.8200 Validation Loss: 0.0141
Epoch[31/50] Training Accuracy: 0.9466 Training Loss 0.0025 Validation Accuracy: 0.7940 Validation Loss: 0.0151
Epoch[32/50] Training Accuracy: 0.9300 Training Loss 0.0029 Validation Accuracy: 0.8600 Validation Loss: 0.0112
Epoch[33/50] Training Accuracy: 0.9503 Training Loss 0.0021 Validation Accuracy: 0.7940 Validation Loss: 0.0142
Epoch[34/50] Training Accuracy: 0.9423 Training Loss 0.0025 Validation Accuracy: 0.8380 Validation Loss: 0.0123
Epoch[35/50] Training Accuracy: 0.9483 Training Loss 0.0025 Validation Accuracy: 0.8320 Validation Loss: 0.0128
Epoch[36/50] Training Accuracy: 0.9543 Training Loss 0.0021 Validation Accuracy: 0.8340 Validation Loss: 0.0128
Epoch[37/50] Training Accuracy: 0.9531 Training Loss 0.0022 Validation Accuracy: 0.8500 Validation Loss: 0.0128
Epoch[38/50] Training Accuracy: 0.9480 Training Loss 0.0022 Validation Accuracy: 0.8220 Validation Loss: 0.0163
Epoch[39/50] Training Accuracy: 0.9563 Training Loss 0.0019 Validation Accuracy: 0.8260 Validation Loss: 0.0143
Epoch[40/50] Training Accuracy: 0.9609 Training Loss 0.0017 Validation Accuracy: 0.8280 Validation Loss: 0.0131
Epoch[41/50] Training Accuracy: 0.9660 Training Loss 0.0019 Validation Accuracy: 0.8440 Validation Loss: 0.0135
Epoch[42/50] Training Accuracy: 0.9737 Training Loss 0.0012 Validation Accuracy: 0.8440 Validation Loss: 0.0128
Epoch[43/50] Training Accuracy: 0.9754 Training Loss 0.0011 Validation Accuracy: 0.8420 Validation Loss: 0.0160
Epoch[44/50] Training Accuracy: 0.9689 Training Loss 0.0013 Validation Accuracy: 0.8460 Validation Loss: 0.0132
Epoch[45/50] Training Accuracy: 0.9689 Training Loss 0.0013 Validation Accuracy: 0.8280 Validation Loss: 0.0154
Epoch[46/50] Training Accuracy: 0.9669 Training Loss 0.0015 Validation Accuracy: 0.8320 Validation Loss: 0.0144
Epoch[47/50] Training Accuracy: 0.9703 Training Loss 0.0014 Validation Accuracy: 0.8440 Validation Loss: 0.0162
Epoch[48/50] Training Accuracy: 0.9646 Training Loss 0.0017 Validation Accuracy: 0.8180 Validation Loss: 0.0197
Epoch[49/50] Training Accuracy: 0.9597 Training Loss 0.0019 Validation Accuracy: 0.8380 Validation Loss: 0.0139
Epoch[50/50] Training Accuracy: 0.9694 Training Loss 0.0016 Validation Accuracy: 0.8360 Validation Loss: 0.0150
```

Fig 7: Training and Validation Log for CNN

For CNN, although the training accuracy goes all the way up to 97% after full training(50 epochs), I chose to early stop at epoch 16, after which the validation loss starts the generic trend of going up in each epoch while the training loss goes downwards which is the prime sign of overfitting. The reason for choosing a version of the network that is comparatively performing less accurately than the final model is to avoid overfitting; which means the version I chose as best would have better performance in generalizing any unknown data than the final one.

Table 7: Final CNN Performance

Epoch	16
Training Accuracy	89.943%
Training Loss	0.004
Validation Accuracy	83.800%
Validation Loss	0.008
Test Accuracy	85.900%

Table 8: Classwise Accuracy:

Class Name	Accuracy on Test Data
Grass	64%
Ocean	100%
Red carpet	91%
Road	100%
Wheatfield	82%

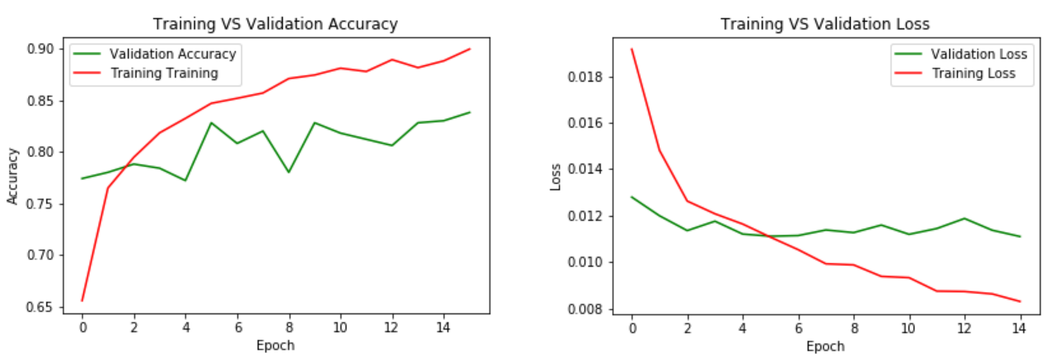

Fig 8: Training and Validation Log for CNN up to early stopping threshold (Epoch 16)

<u>Concluding Remarks:</u>

I could have tweaked the network better if I had the computational resources. Google Colab doesn't allow unlimited usage of GPU and I got banned 2-3 times for 12 hour period as I have been continuously performing arduous computations for the last 1 week.