Part 1: Descriptor and SVM

Code run on Google Collab Notebook

For the descriptor formation, I decided to use number of blocks width=4, number of blocks height=4, and number of histogram bins=4. To my understanding, since the classification this task is aiming to accomplish is quite broad and more based on using large constant regions of images rather than capturing extremely fine details (since we are trying to capture a scene rather than identify specific objects in an image), I went with the minimum dimensions for the overlapping histograms descriptors generation since I believe larger image blocks allows the generation of more robust and broad data points since they overlap and aggregate larger areas of the images which then create less data points that are more broad for scene capture. Also this allows for the creation of an SVM that is more generalized.

After generating the grass, ocean, redcarpet, road, and wheatfield descriptors for each set of data (test, validation, test), I stored the descriptors in .NPZ files for continual use and training of the 5 classifiers. I aggregated the data for each set by reading combining the descriptors from the NPZ files and one hot encoding their true class labels into 5 parameter length vectors (representing the 5 classifiers) by using -1 as incorrect classification and +1 as correct classification.

Next I shuffled the training descriptor and true class label data together, applied StandardScaler().fit_transform on the 1024 descriptor columns, ran grid search as below with max iterations of 1000 and tolerance of 0.001, and stored the grid search sklearn objects into pickle files for persistence:

```
parameters = {"dual":[False], "max_iter":[1000], "tol":[0.001]
              ,"C":[0.1, 0.5, 1, 5, 10]}
clf_grass.fit(X_train_shuffle, Y_train_shuffle[:,0])
clf_ocean.fit(X_train_shuffle, Y_train_shuffle[:,1])
clf_redcarpet.fit(X_train_shuffle, Y_train_shuffle[:,2])
clf_road.fit(X_train_shuffle, Y_train_shuffle[:,3])
clf_wheatfield.fit(X_train_shuffle, Y_train_shuffle[:,4])
```

In order to figure out the best C parameter for the 5 SVM models, I performed grid search using a range [0.1, 0.5, 1, 5, 10] for the C hyperparameters. The results of the grid search showed me that the best classifiers for each classifier were the ones that used 0.1 as the C value. Below are the grid search results for each classifier when trained on the shuffled training data, the individual accuracies of each classifier when applied on the validation set, and the validation data set confusion matrix when using each classifier's best C parameter (0.1). During the grid search process, the best hyperparameter is based on performing cross validation using the inputted training data. In my case, I used the default 5 folds for the cross validation.

```
Individual Accuracies (Validation):
Grass:
[0.84114286 0.83057143 0.82771429 0.82828571 0.82771429]
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  85.86666666666667
Ocean:
[0.87657143 0.86542857 0.86571429 0.86371429 0.864     ]
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  89.06666666666668
RedCarpet:
[0.95885714 0.95257143 0.95171429 0.95285714 0.95314286]
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  97.2
Road:
[0.82628571 0.81542857 0.81       0.80828571 0.80628571]
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  81.86666666666666
Wheatfield:
[0.832      0.81142857 0.80942857 0.808      0.808     ]
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  82.26666666666667

          Validation:
          array([[ 98,   7,   1,  15,  29],
                 [ 10, 118,   2,  14,   6],
                 [  4,   2, 137,   4,   3],
                 [ 10,  18,   3,  99,  20],
                 [ 13,  16,   3,  23,  95]])
```

As you can see the selection of descriptor parameters and selection of 0.1 for C in the SVM model produced fairly accurate results of > 80% for the validation set. Below are the results for the best SVM classifiers being applied on the 750 images testing data:

```
Individual Accuracies (Test):
Grass:
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  85.46666666666667
Ocean:
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  87.06666666666666
RedCarpet:
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  95.06666666666666
Road:
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  82.53333333333333
Wheatfield:
LinearSVC(C=0.1, dual=False, tol=0.001)
Accuracy:  84.53333333333333
```

```
Test:
array([[ 95,  12,   5,   9,  29],
       [  5, 100,   3,  24,  18],
       [  5,   3, 135,   3,   4],
       [ 11,  16,   4,  96,  23],
       [ 17,   8,   2,  24,  99]])
```

One thing that I saw is that the red carpet and ocean classifiers seemed to have the highest accuracies across the grid search training process, validation set, and even the testing set. Upon looking at the datasets for each of those classifiers for which they were trained on, red carpet images had larger instances of vibrant red and ocean had larger instances of vibrant blue and green. In my opinion, this makes them a lot more resistant to noise since more vibrance in those colors can overpower the presence of other colors. I believe this creates for more linearly separable descriptors since colors representative of the classification are more prevalent. As compared to the road classifier, its images contain more noise since almost all images of road are taken outside where mountains, people, grass, cars, and even the shading produced by time of day can distort the images to the point where other features stand more than the actual road itself. Another issue with this classifier is that roads have a sort of grayish color that that makes it hard to distinguish via histogram when there are many cars, trains, and buses that also share similar color profile which can actually invite more misclassification. In addition, I found the definition of a road to be too broad in the images provided. I believe the SVM can

easily distinguish road images if there is large presence of road with very little noise and has the road as the main focus of the image allowing for more linearity in classification than nonlinearity.

Hard to classify as road since train overpowers the focus

Easier to classify as road since more road elements than forest

Part 2: Neural Network and Convolutional Neural Network

Neural Network: Code for Neural network was run on Google Collab Notebook

For the data loading section of this assignment, I created a prepare_dataset function that enables me to ability to load the entirety of the training, validation, and testing datasets into DataLoaders, divide the data into batches, shuffle the data during before each epoch, and resize the images when see necessary.

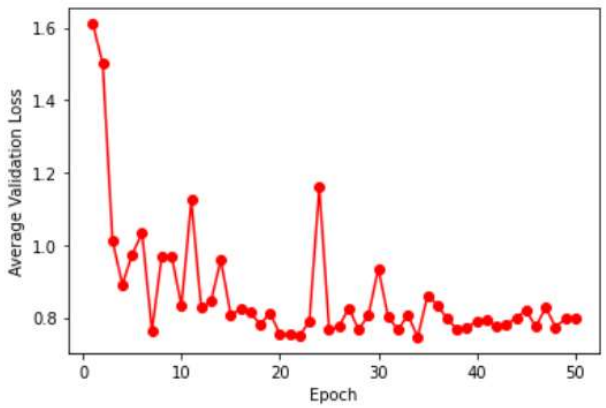
Below I trained multiple NNs and evaluated each to determine the one that I found best suited.

NN1:

Batch size=32, resize images to 100x200, 50 epochs of training, learning rate of 1e-5, layer structure:

```
nn.Linear(3*100*200, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 5)
```

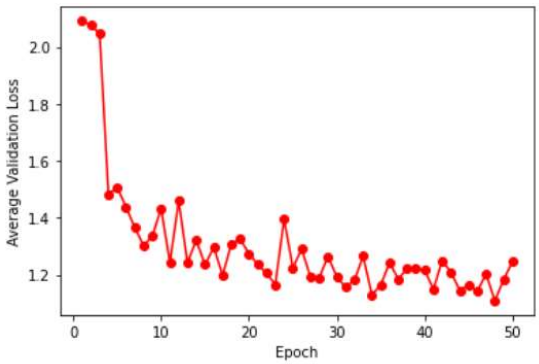
```
Validation:
Test Error:
 Accuracy: 75.6%, Avg loss: 0.797676

Done!
```



NN2:

Batch size=32, resize images to 100x200, 50 epochs of training, learning rate of 1e-6, layer structure:

```
nn.Linear(3*100*200, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 5)
```

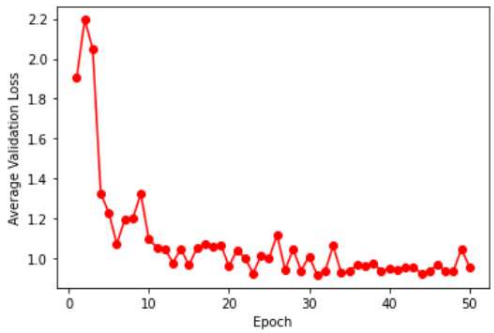


NN3:

Batch size=32, resize images to 50x100, 50 epochs of training, learning rate of 1e-5, layer structure:

```
nn.Linear(3*50*100, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 5)
```

```
Validation:
Test Error:
 Accuracy: 73.2%, Avg loss: 0.957843

Done!
```



For model 1, downsizing the 260x340 images to 100x200 and training for 50 epochs with 1e-5 learning rate produced a curve that dipped down very sharply in the beginning and then slowly tapers down in error as epochs go by. Problem that I see is that there seems to be more variance in this model since even after a 10s of epochs, we see some sharp increases in error especially from epochs 20 to 35 where there a good number of spikes. Despite this, the 50th epoch produced 75.6 percent accuracy and 0.79 error.

For model 2, I kept the model params the same as model 1 but decreased the learning rate to 1e-6 to see if a smaller step size would make the error curve much smoother and potentially give a better error values. It did in fact result in a smoother curve, but after 50 iteration the error did not reach the low that model 1 reached.

For model 3, I resized the images further by ½ and used the same learning rate as model 1 with 50 epochs. The result is a smooth curve that has lower error values for the epochs of model 2 and almost reaches the average validation loss of model 1. Since model 2's requires more training to reach the same level of model 1, I chose model 1 first. Besides the final validation accuracy for model 1 being higher than model 3's accuracy, I saw that for that the last 10 epochs has smaller error rates for model 1 than model 3 (Comparing 0.7-0.8 vs 0.8-0.9). I decided to choose model 1 as my final model.

The final confusion matrix after using model 1 on the test data:

```
Test Error:
 Accuracy: 70.9%, Avg loss: 1.009309

[[ 92  12   2  14  30]
 [ 12  96   1  22  19]
 [  0   2 133  11   4]
 [ 12  14  12  98  14]
 [ 19   4   4  10 113]]
```

Convolutional Neural Network: Code for Neural network was run on personal GPU

For the data loading section of this assignment, I used the same prepare_dataset function that enables me to ability to load the entirety of the training, validation, and testing datasets into DataLoaders from the NN section.

Below I trained multiple CNNs and evaluated each to determine the one that I found best suited. One thing to note is that I didn't resize the images for CNN models since the images were already very small and I figured that CNNs are much better at feature extracting and resizing the images at the max pooling layers. Further resizing might eliminate useful details.

CNN 1:

Batch size=32, 50 epochs of training, learning rate of 1e-5, layer structure:

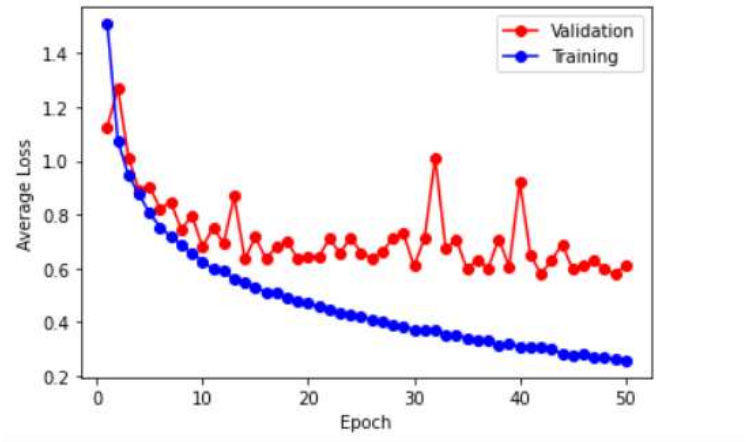
CNN portion:

```
nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),
        nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),
        nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride=1, padding=1),
        nn.ReLU()
```

NN portion:

```
        nn.Flatten(),
        #Padding preserves shape, but 2 max pools divides dims by 4
        nn.Linear(60*90*32, 128),
        nn.ReLU(),
        nn.Linear(128, 5),
```

```
Accuracy: 78.1%, Avg loss: 0.612529

Done!
```



CNN 2:

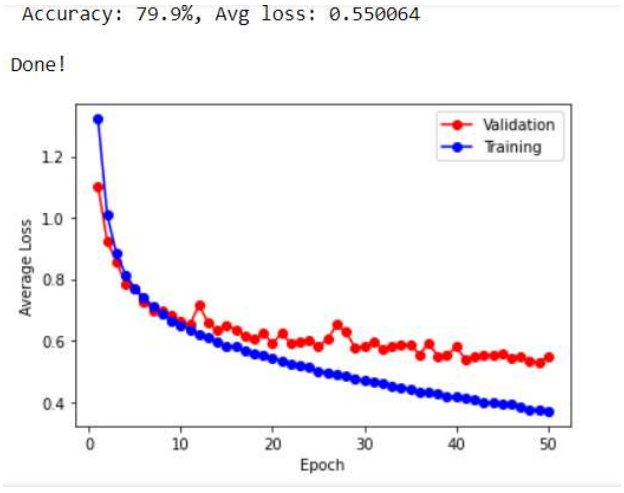
Batch size=32, 50 epochs of training, learning rate of 1e-5, layer structure:

CNN portion:

```
nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),
        nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),
        nn.Conv2d(in_channels=64, out_channels=32, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
    )
```

NN portion:

```
nn.Flatten(),
        #Padding preserves shape, but 2 max pools divides dims by 4
        nn.Linear(60*90*32, 128),
        nn.ReLU(),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.Linear(64, 5)
```



CNN 3:

Batch size=32, 50 epochs of training, learning rate of 1e-5, layer structure:

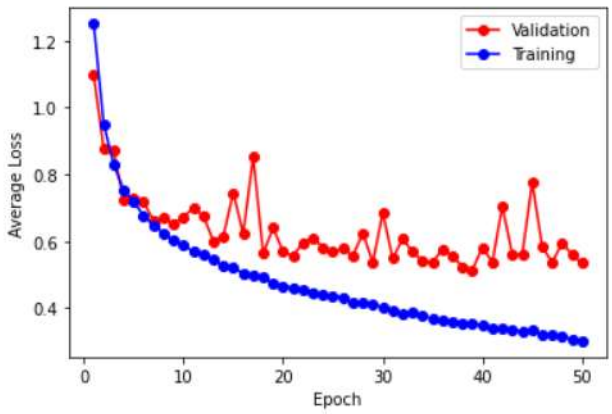
CNN portion:

```
      nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, p
adding=1),
      nn.ReLU(),
      nn.MaxPool2d(2,2),
      nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1,
 padding=1),
      nn.ReLU(),
      nn.MaxPool2d(2,2),
      nn.Conv2d(in_channels=128, out_channels=64, kernel_size=3, stride=1,
 padding=1),
      nn.ReLU()
```

NN portion:

```
      nn.Flatten(),
      nn.Linear(60*90*64, 128),
      nn.ReLU(),
      nn.Linear(128, 64),
      nn.ReLU(),
      nn.Linear(64, 5)
```

```
Validation:
Test Error:
 Accuracy: 81.3%, Avg loss: 0.534679

Done!
```



For all of the CNNs I used a learning rate of 1e-5 since this learning rate was helpful for the NN portion of the assignment for achieving the low error rate quickly. CNN 1 configuration was taken from professor's starter CNN code as a starting point for training. Each of the CNNs created uses padding=1 and stride=1 since I wanted to maximize how much feature extraction the filters can perform with missing the edges. CNN 1 yielded pretty good results on the validation data but I wanted to see if I could stretch it further since we have the added bonus of feature extraction. The curves for training and validation are pretty good as well and the variance isn't enough to cause serious overfitting.

To eliminate the spikes and smoothen the variance of model 1, I created CNN 2 to have an extra hidden layer to the NN to see if I can capture the non-linearities in classification. The resulting set of curves has much less variance and a better accuracy and error rate than CNN 1.

I pushed it a step further and created CNN 3 to model after CNN 2 but have an increases number of input and output channels (64 -> 128) with the same number of convolution layers, because I though more filters might capture more features in the images to bypass noise. The resulting set of curves brought back the variance and spikes, but the final epoch's accuracy and error rate were slightly better.

I decided to stop my CNN modifications here, but I think a good experiment in the future would be to modify the strides since our classification needs a good way to avoiding the capturing of noisy elements. The best classification might be achieved by scanning the borders of an image rather than including the foreground. Since our images were quite small, I found that adding stride reduces the image sizes drastically if we follow the `[(W-K+2P)/S]+1` formula.

Final confusion matrix when using CNN 3:

```
Test Error:
 Accuracy: 76.7%, Avg loss: 0.642606

[[120   4   1  11  14]
 [ 13  98   2  30   7]
 [  1   0 138  10   1]
 [ 14   6   2 118  10]
 [ 30   3   4  12 101]]
```

From the NN to CNN, all classifications did slightly better in the sense that the misclassifications aren't as distributed as they were for the NN.