

ECE 66100 Homework #7

by

Adrien Dubois (dubois6@purdue.edu)

October 31, 2024

Contents

1 Theory Questions

1.1 Question 1:

Conceiver of a new texture detector software. You can use the pyramid representation of an image to capture information in some or all of the octaves.

- On one side, I would build a scale-pyramid representation of the image through mean-pooling layers where each layer reduces the spatial dimensions by a factor of 2 without changing the channel dimension.
- Using this scale pyramid, I could calculate individual Gram Matrices per layer.
- On the other side, I would apply a deep learning architecture for image classification using CNNs similar to the VGG implementation.
- By downsampling the Gram Matrices along the channel dimension by the appropriate factor of 2, I could concatenate the gram matrix with the CNN's dense representation of the image to provide greater texture information that would help guide the learning process. This would be done across each layer of scale pyramid and CNN network.

I don't have any particular examples where I believe that my architecture would work well; however, I believe that it would outperform the Gram Matrix implementation that was performed during this assignment since the deep-learning model would be informed of the Gram Matrix based textural information during the training process and could therefore decide whether or not to use such information for predicting the class labels. Finally, extracting the feature map for the final layer of my CNN encoder would provide a dense-matrix-representation of the image, with textural information fed in through the multi-scale gram matrix pipeline.

2 RGB to HSV:

For this project, we first convert the BGR representation of the image to HSV. This can be visualized as a rotation of the RGB cube along the vertical axis as seen in the graph below from Avi Kak's lecture on texture and color.

Images/RGB_HSV.png

In the RGB space, you would find black pixels close to the origin, while white pixels would be at the corner furthest away from the origin. Therefore, you can think of HSV turning this cube onto the vertical axis where the pixel with the highest intensity (white) is the highest point along the w axis. The hue space then becomes a rotation around that axis, and the saturation is a scalar value of the distance of a color to that vertical axis. In this way, we use the following equations to determine the HSV representation of an image

$$M = \max(R, G, B)$$

$$m = \min(R, G, B)$$

$$c = M - m$$

$$V = M$$

$$H = \begin{cases} 60 \left(\frac{G-B}{c} \bmod 6 \right) & M == R, c \neq 0 \\ 60 \left(\frac{B-R}{c} + 2 \right) & M == G, c \neq 0 \\ 60 \left(\frac{R-G}{c} + 4 \right) & M == B, c \neq 0 \\ 0 & c == 0 \end{cases}$$

$$S = \begin{cases} \frac{c}{V} * 255 & V \neq 0 \\ 0 & V == 0 \end{cases}$$

Lastly, to match the outputs generated through OpenCV, I rescale the huespace to 180deg instead of a full 360. I also apply a ceiling function on the floating point values generated above before converting them to numpy integers.

```

1 def img_BGR_to_HSV(img):
2     img = img.astype(np.float32)
3     img_hsv = np.zeros_like(img)
4
5     # Calculate key parameters through the channel axis
6     M = np.max(img, axis=2)
7     m = np.min(img, axis=2)
8     c = M - m
9     V = M
10
11     # For the rows, if the max is in the first column, etc
12     h0_mask = (M == img[:, :, 2]) & (c != 0) # M == R, c!=0
13     h1_mask = (M == img[:, :, 1]) & (c != 0) # M == G, c!=0
14     h2_mask = (M == img[:, :, 0]) & (c != 0) # M == B, c!=0
15     c_mask = (c == 0) # c == 0
16
17     # Calculate H Values for each row

```

```

18 # We don't just want to use the mask since c can be zero for greyscale. So we want
19 # to only compute on the masks, by checking for where to input values in first.
20 with np.errstate(divide='ignore', invalid='ignore'):
21     img_hsv[:, :, 0] = np.where(h0_mask, (60 * ((img[:, :, 1] - img[:, :, 0]) / c)
22     % 60), img_hsv[:, :, 0])
23     img_hsv[:, :, 0] = np.where(h1_mask, (60 * ((img[:, :, 0] - img[:, :, 2]) / c +
24     2)), img_hsv[:, :, 0])
25     img_hsv[:, :, 0] = np.where(h2_mask, (60 * ((img[:, :, 2] - img[:, :, 1]) / c +
26     4)), img_hsv[:, :, 0])
27     img_hsv[:, :, 0][c_mask] = 0 # No divide by 0 errors are possible here
28 # To follow opencv formatting, I will rescale the hue angles to 180deg instead of
29 360
30 img_hsv[:, :, 0] /= 2
31
32 # Fill in with correct values for the S column: (c/V)
33 img_hsv[:, :, 1][V != 0] = c[V != 0] / V[V != 0] * 255
34
35 # Fill in V col
36 img_hsv[:, :, 2] = V
37
38 return np.ceil(img_hsv).astype(np.uint8)

```

3 Extracting LBP Histograms:

3.1 Algorithm Description:

The LBP histogram method for texture extraction works by looking at every pixel in the image, counting that as a center pixel and creating a binary pattern for the surrounding pixels in a circle around the center. Formally, this binary pattern can be calculate as follows:

- First, it is important to note that this only works for 1 dimensional images. In our assignment we used the Hue channel of HSV images, but greyscaled images would work just as well.
- Consider a coordinate on the image as the center point x
- Evaluate the pixel value at points around the circle. The number of points (P), and the radius of that circle (R) are user-defined hyper-parameters.
 - These points can be evaluated as follows:

$$(x, y) = R \times \cos\left(\frac{2\pi}{P}\right), R \times \sin\left(\frac{2\pi}{P}\right)$$

- It is important to note that since we are using discrete indices (images), we compute the pixel interpolation as follows for pixels on the top-right diagonal (a similar formula is used for other diagonals):

$$\begin{aligned}
 p[1] = & \text{center_value} \cdot (1 - 0.707) \cdot (1 - 0.707) + \\
 & \text{img_h_pad}[y][x + 1] \cdot (1 - 0.707) \cdot 0.707 + \\
 & \text{img_h_pad}[y + 1][x] \cdot 0.707 \cdot (1 - 0.707) + \\
 & \text{img_h_pad}[y + 1][x + 1] \cdot 0.707 \cdot 0.707
 \end{aligned}$$

- Once we have calculated the pixel value for all points, we threshold them using the center pixel. Starting from the top and moving clockwise, we assign a value of 1 if the pixel on the circle is bigger than the center, and 0 if it is less than or equal to the center pixel.
- Next, since we need a rotational-invariant version of the binary pattern, we circularly shift the pattern until we find its minimal representation.
- Lastly, the authors of the LBP paper noticed that only binary patterns with a run of 0s followed by a run of only 1s provided useful information. Therefore, we can encode the binary patterns as follows for the histogram.
 - **If** the `minIntVal` representation involves more than two runs, we encode it by the integer $P + 1$.

- **Else**, if the `minIntVal` representation consists of all 0's, we encode it as 0.
- **Else**, if the `minIntVal` representation consists of all 1's, we encode it as P .
- **Else**: the `minIntVal` representation of a binary pattern has exactly two runs (i.e., a run of 0's followed by a run of 1's). We represent the pattern by the number of 1's in the second run.

3.2 Code Implementation:

```

1 class LBP():
2     def __init__(self, R, P):
3         self.R = R
4         self.P = P
5     def run_lbp(self, img_path):
6         # Read image and convert it to HSV, then use the H channel for all downstream
7         # tasks.
8         img_bgr = cv2.imread(img_path)
9         img_hsv = img_BGR_to_HSV(img_bgr)
10        img_h = img_hsv[:, :, 0]
11
12        # Create padded image of size (64,64) for more feasible computation
13        img_h_sized = cv2.resize(img_h, (62,62), interpolation=cv2.INTER_AREA)
14        img_h_pad = np.pad(img_h_sized, pad_width=1, mode="constant", constant_values=0)
15
16        # Initialize the histogram vector for the image: (We allow a max index of P + 1
17        # 0->9 in this case)
18        lbp_histogram = np.zeros(self.P + 2)
19
20        # Loop through all possible LBP centers:
21        for y in range(self.R, img_h_pad.shape[0]-self.R):
22            for x in range(self.R, img_h_pad.shape[1]-self.R):
23                center_value = img_h_pad[y, x] # Scalar due to greyscale
24                p = np.zeros(8)
25
26                # Check the cardinal direction points (up,down,left,right)
27                if img_h_pad[y+1][x] > center_value:
28                    p[0] = 1
29                if img_h_pad[y][x+1] > center_value:
30                    p[2] = 1
31                if img_h_pad[y-1][x] > center_value:
32                    p[4] = 1
33                if img_h_pad[y][x-1] > center_value:
34                    p[6] = 1
35
36                # We also have to check the diagonals.
37                # To calculate the pixel values at these diagonal points, we need to do
38                # pixel-interpolation
39                # We also apply thresholding on the interpolated points compared to the
40                # center to determine 0/1.
41                # Top right point
42                p[1] = center_value * (1 - 0.707) * (1 - 0.707) + \
43                    img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
44                    img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
45                    img_h_pad[y+1][x+1] * 0.707 * 0.707
46                p[1] = 1 if p[1] > center_value else 0
47
48                # Bottom right point
49                p[3] = center_value * (1 - 0.707) * (1 - 0.707) + \
50                    img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
51                    img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
52                    img_h_pad[y-1][x+1] * 0.707 * 0.707
53                p[3] = 1 if p[3] > center_value else 0
54
55                # Bottom left point
56                p[5] = center_value * (1 - 0.707) * (1 - 0.707) + \
57                    img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
58                    img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
59                    img_h_pad[y-1][x-1] * 0.707 * 0.707
60                p[5] = 1 if p[5] > center_value else 0
61
62                # Top left point
63                p[7] = center_value * (1 - 0.707) * (1 - 0.707) + \
64                    img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \

```

```

61         img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
62         img_h_pad[y+1][x-1] * 0.707 * 0.707
63     p[7] = 1 if p[7] > center_value else 0
64
65     # Now that we have out bitvector representation for the circle of points
66     # We want to find the unique min bitvector to represent the value at
67     # We do this through circular bit-shifts to find the minimal
68     # This method is from Avi Kak's implementation in lecture 16
69     bv = BitVector(bitlist=p)
70     min_val = min([int(bv<<1) for _ in p])
71     min_bv = BitVector(intVal=min_val, size=len(p))
72
73     # Lastly, we use this min-bv value to get the final encoding for that
74     # So we create a min-int-val based integer representation of the binary
75     # From Avi's Notes:
76     # - If the minIntVal representation involves more than two runs, encode
77     # - Else, if the minIntVal representation consists of all 0's, represent
78     # - Else, if the minIntVal representation consists of all 1's, represent
79     # - Else: the minIntVal representation of a binary pattern has exactly
80     # a run of 0s followed by a run of 1s, represent the pattern by
81     # the number of 1's in the second run
82     num_runs = len(min_bv.runs())
83
84     encoding = None
85     # Mix of 1s and 0s
86     if num_runs > 2:
87         encoding = self.P + 1
88     # All 0s (8 of them)
89     elif min_bv.int_val() == 0 and num_runs == 1:
90         encoding = self.P
91     # 8 1s
92     elif min_bv.int_val() == 255 and num_runs == 1:
93         encoding = self.P
94     # Number of 1s in the second pattern if it is a run of all 0s then 1s
95     else:
96         encoding = len(min_bv.runs()[1])
97     lbp_histogram[encoding] += 1
98     return lbp_histogram

```

4 Gram Matrix based texture extraction:

4.1 Gram Matrix

For the Gram Matrix portion of this assignment, I first had to convert the images read using OpenCV from BGR to RGB due to the requirements of Resnet and VGG. Next, I rescaled the images to a shape of (256,256) for faster computation speed of the feature maps. Once I have a feature map, I can compute the gram matrix as follows:

$$G = F \times F^T$$

To do so, I first flattened my input image from a shape of (N, C, H, W) to (N, C, H×W). I can then compute the Gram Matrix by transposing along the channel and height width dimensions. Lastly, to most easily display the gram matrices using a heatmap, it is important to note that I use bilinear interpolation to rescale the matrix from a shape of (N, C, C) to (N, 32, 32). This speeds up the training time for my SVM classifier since it would only use 1024 features instead of 262,144 features per image.

4.2 Code Implementation:

```

1 def get_gram_matrix(feature_mat_list):
2     f_mats = np.array(feature_mat_list)
3     N, C, H, W = f_mats.shape
4     fmats_flat = f_mats.reshape(N, C, H*W)
5
6     # A Gram matrix is the feature_map * feature_map.T
7     gram_matrix = fmats_flat @ fmats_flat.transpose(0, 2, 1)
8
9     # Conver the numpy array to a pytorch tensor for bilinear interpolation in
    downsampling
10    # I also unsqueeze in the first dimension so that pytorch treats the final two
    dimensions as H,W and downsamples on those
11    # Otherwise, would read the it as Batch, Channel, Height and a missing width
12    gram_mat_tensor = torch.from_numpy(gram_matrix).unsqueeze(0)
13
14    # Lastly, we want to resize the gram matrix from 512x512 to (32,32) for easier
    computation
15    # We do this using bilinear interpolation
16    downsampled_matrix = F.interpolate(gram_mat_tensor, size=(32, 32), mode='bilinear',
    align_corners=False)
17
18    return downsampled_matrix.squeeze().numpy()

```

5 Extra Credit: Channel Normalization Parameter based Texture Extraction:

For the channel normalization parameters the process is even more simple and efficient. In this method, we will find the mean and variance of the pixel values across each channel. We can then interleave these values together to create the texture matrix. For displaying the results, I take the flattened result and reshape it into a square matrix that I display using Seaborn's heatmap method.

5.1 Implementation:

```

1 def get_normalization_params(feature_mat_list):
2     f_mats = np.array(feature_mat_list)
3
4     means = f_mats.mean(axis=(2, 3))
5     variances = f_mats.std(axis=(2, 3))
6
7     # I first stack the arrays together, and then reshape the final matrix to interleave
    the means and variances
8     mu_sigma_stacked = np.stack((means, variances), axis=-1)
9     channel_norm_params = mu_sigma_stacked.reshape(f_mats.shape[0], 2*f_mats.shape[1])
10
11    return channel_norm_params

```

6 Results:

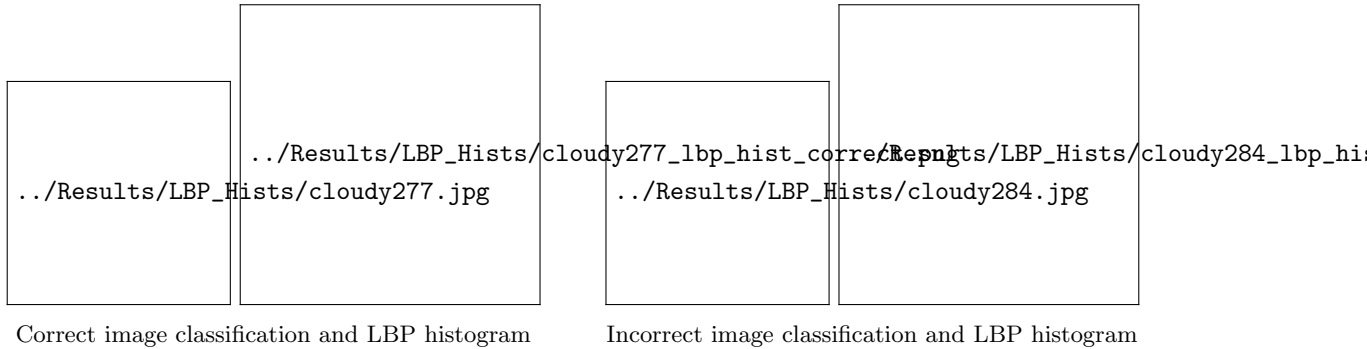
6.1 Dataset Description:

The dataset used for the results section of this assignment includes 1125 photos split into training and test splits (925 training images and 200 test images). These images belong to four different categories: cloudy, rain, sunshine and sunrise, and the dataset is evenly distributed among all of these categories to avoid overfitting. The goal of this assignment is to classify these images based on their textures. We will report a 4×4 confusion matrix for the classification accurac for all texture dectors. It is important to note that the following encoding will be used to represent the class names for the confusion matrices:

- cloudy: 0
- rain: 1
- shine: 2
- sunrise: 3

6.2 LBP Results:

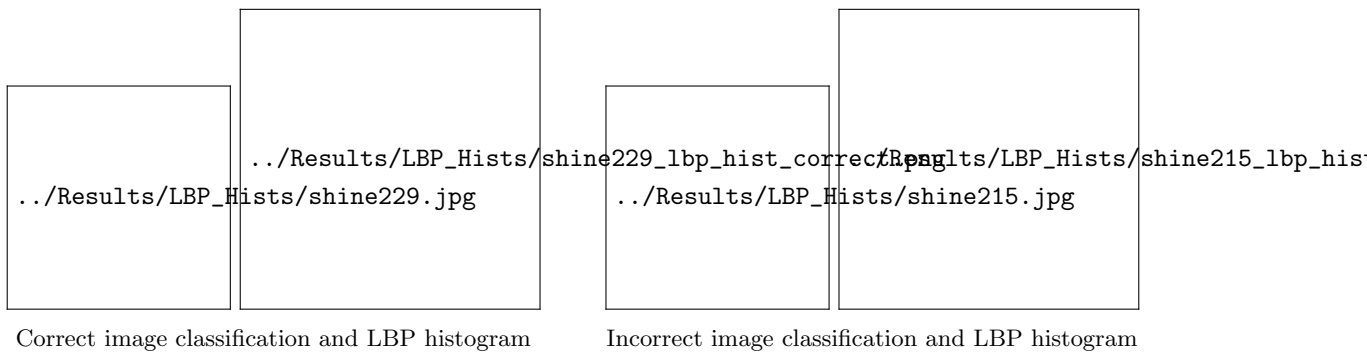
The following bar charts are the histograms for each class. I have included the image followed by its LBP histogram in each example. Additionally, the first image was one that resulted in a correct classification prediction, while the second image was one that resulted in an incorrect prediction.



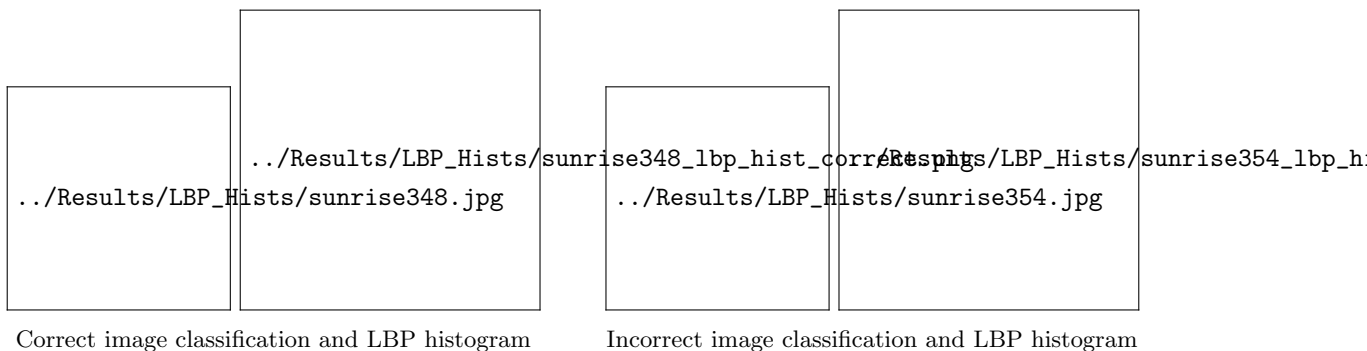
Cloudy: Image classification and histogram pairs



Rain: Image classification and histogram pairs



Sunshine: Image classification and histogram pairs



Sunrise: Image classification and histogram pairs

After training an SVM on the training set, the following results were found by running the trained SVM model on the testing dataset:

Class	Precision	Recall	F1-Score	Support
0	0.71	0.80	0.75	50
1	0.79	0.30	0.43	50
2	0.74	0.40	0.52	50
3	0.44	0.86	0.58	50
Accuracy	0.59 (200 samples)			
Macro Avg	0.67	0.59	0.57	200
Weighted Avg	0.67	0.59	0.57	200

Table 1: Classification Report for SVM Model based on LBP histograms

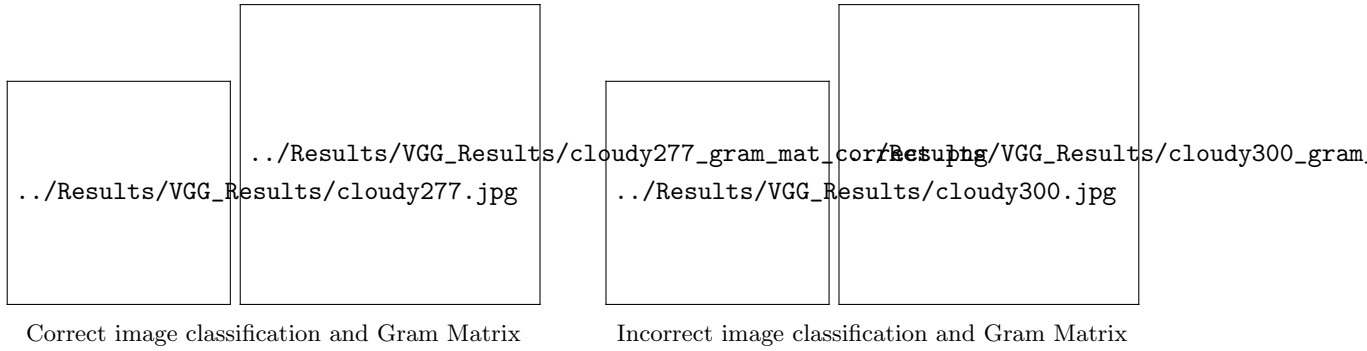
Additionally, I have generated the following confusion matrix to visualize the results in a different way:



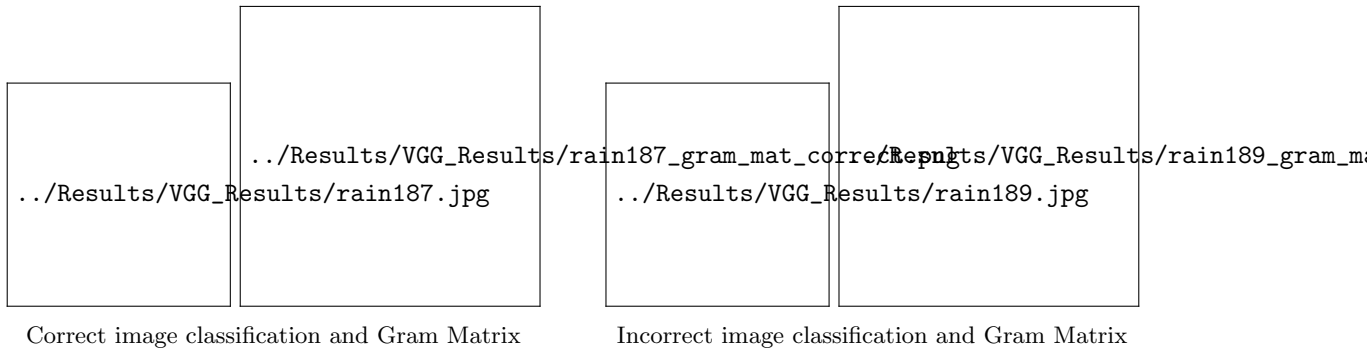
6.3 Gram Matrix Results:

6.3.1 VGG-19 Results:

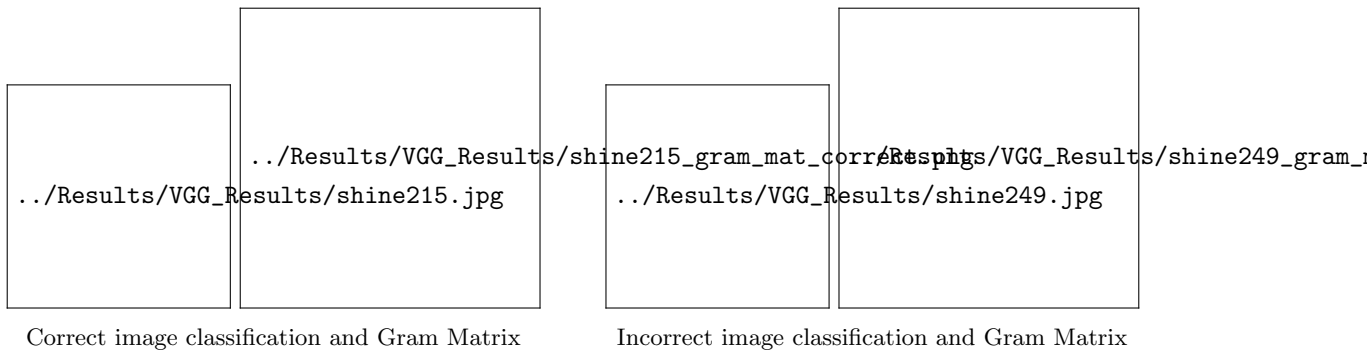
Included below are examples of a correctly classified image, and an incorrectly classified image for each class. The gram matrix associated with that image is also displayed using Seaborn's heatmap method.



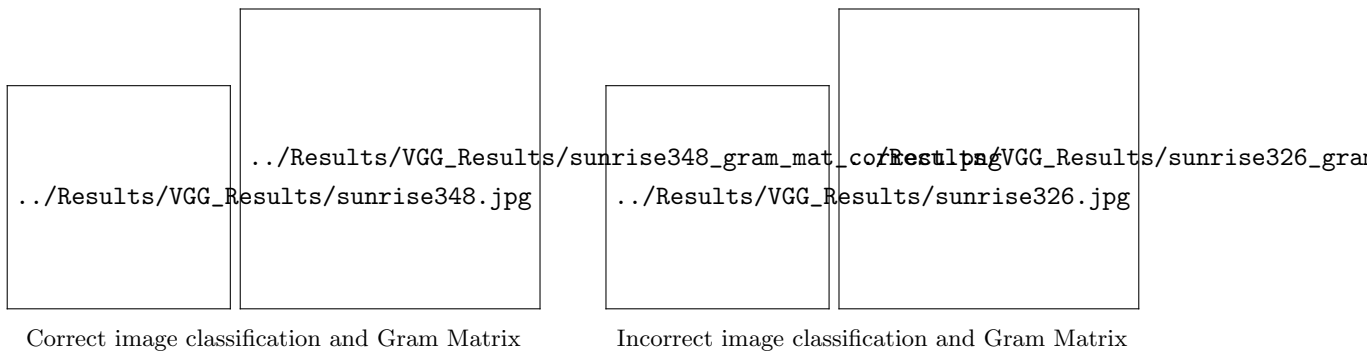
Cloudy: Image classification and Gram Matrix pairs



Rain: Image classification and Gram Matrix pairs



Sunshine: Image classification and Gram Matrix pairs



Sunrise: Image classification and Gram Matrix pairs

After training an SVM on the training set, the following results were found by running the trained SVM model on the testing dataset for VGG:

Additionally, I have generated the following confusion matrix to visualize the results in a different way:

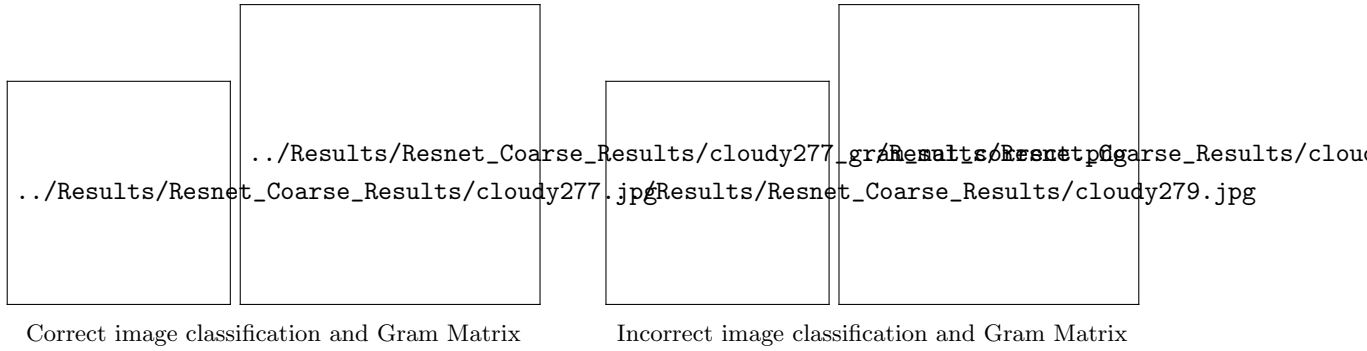
Class	Precision	Recall	F1-Score	Support
0	0.87	0.94	0.90	50
1	0.92	0.88	0.90	50
2	0.93	0.84	0.88	50
3	0.91	0.96	0.93	50
Accuracy	0.905 (200 samples)			
Macro Avg	0.91	0.90	0.90	200
Weighted Avg	0.91	0.91	0.90	200

Table 2: Classification Report for SVM Model based on VGG Gram Matrices

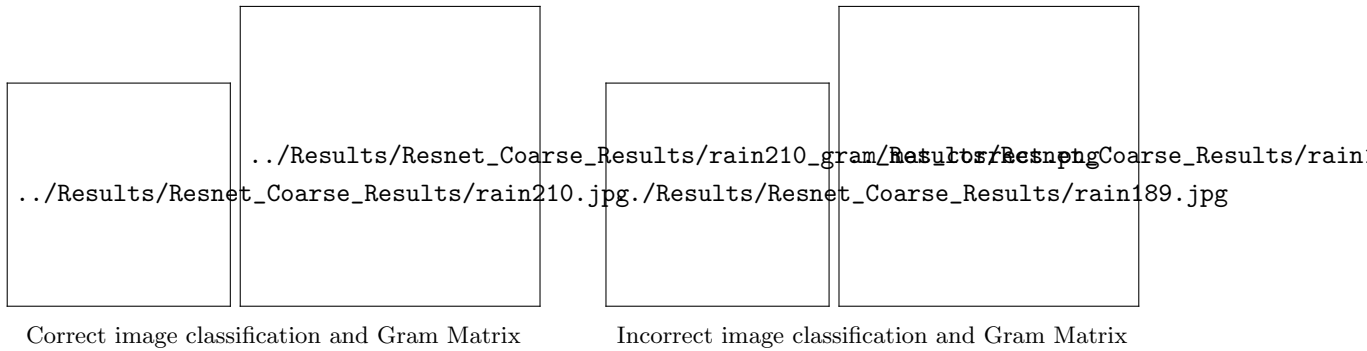


6.3.2 Resnet50-Coarse Results:

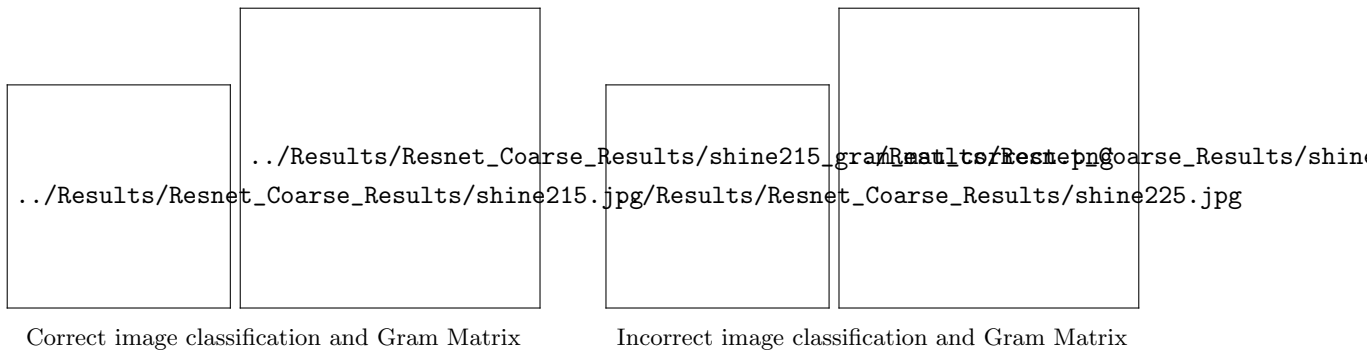
The same results are included below for the Resnet50-Coarse feature maps:



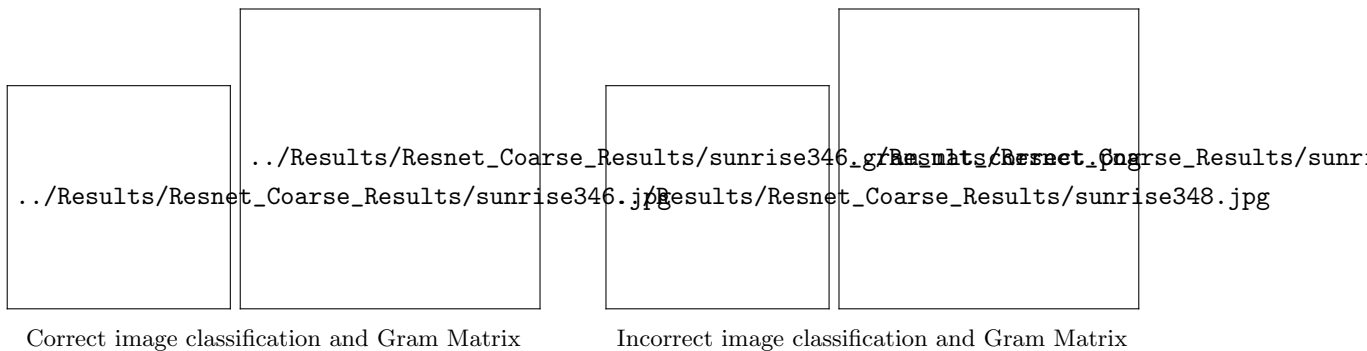
Cloudy: Image classification and Gram Matrix pairs



Rain: Image classification and Gram Matrix pairs



Sunshine: Image classification and Gram Matrix pairs



Sunrise: Image classification and Gram Matrix pairs

After training an SVM on the training set, the following results were found by running the trained SVM model on the testing dataset for Resnet Coarse:

Additionally, I have generated the following confusion matrix to visualize the results in a different way:

Class	Precision	Recall	F1-Score	Support
0	0.57	0.88	0.69	50
1	1.00	0.68	0.81	50
2	0.88	0.60	0.71	50
3	0.80	0.88	0.84	50
Accuracy	0.76 (200 samples)			
Macro Avg	0.81	0.76	0.76	200
Weighted Avg	0.81	0.76	0.76	200

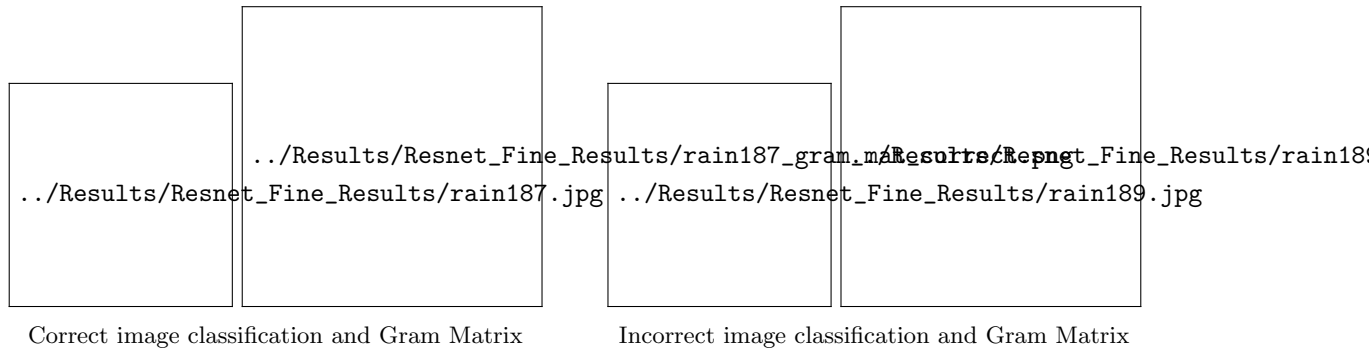
Table 3: Classification Report for SVM Model based on Resnet50-Coarse Gram Matrices



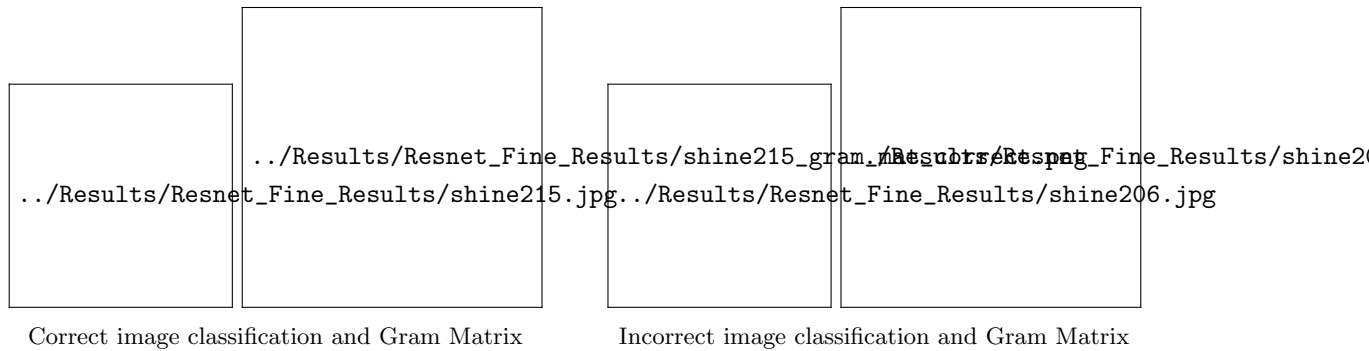
6.3.3 Resnet50-Fine Results:



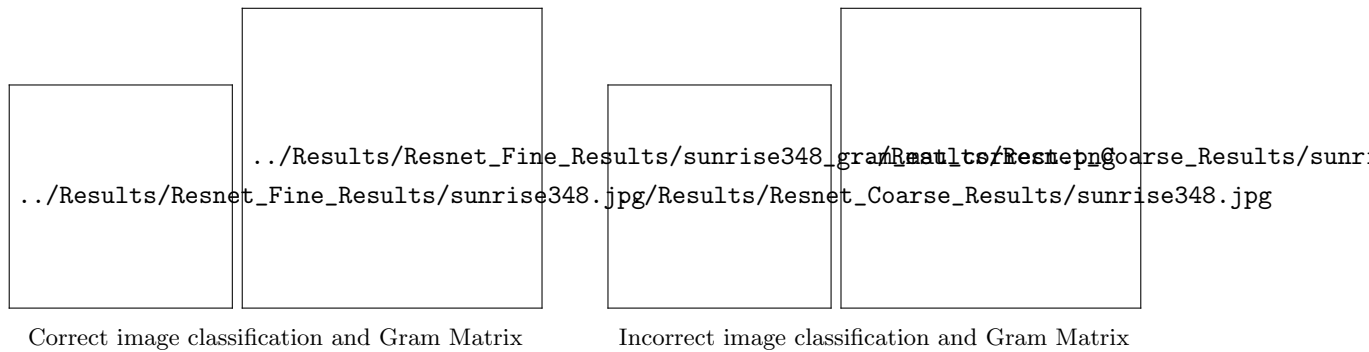
Cloudy: Image classification and Gram Matrix pairs



Rain: Image classification and Gram Matrix pairs



Sunshine: Image classification and Gram Matrix pairs



Sunrise: Image classification and Gram Matrix pairs

After training an SVM on the training set, the following results were found by running the trained SVM model on the testing dataset for Resnet Fine:

Class	Precision	Recall	F1-Score	Support
0	0.82	0.84	0.83	50
1	1.00	0.94	0.97	50
2	0.93	0.82	0.87	50
3	0.84	0.98	0.91	50
Accuracy	0.895 (200 samples)			
Macro Avg	0.90	0.89	0.90	200
Weighted Avg	0.90	0.90	0.90	200

Table 4: Classification Report for SVM Model based on Resnet50-Fine Gram Matrices

Additionally, I have generated the following confusion matrix to visualize the results in a different way:



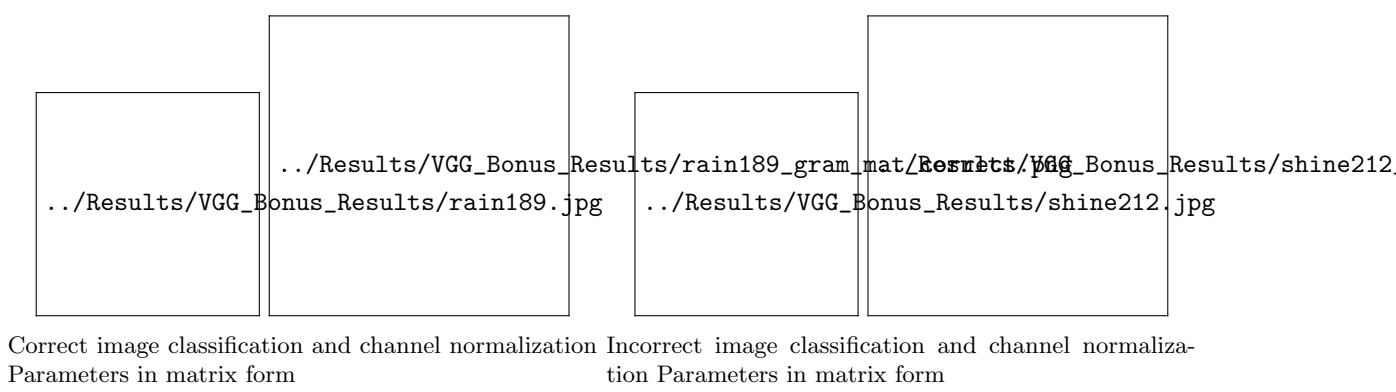
6.4 Discussion of results:

For the required portion of this assignment, the best performing model was the VGG based Gram Matrix extraction. It is logical that the approach that relies on deep learning outperforms the baseline LBP approach that relied only on one channel of the image. This is due to the fact that deep learning models will encode a large amount of information into the feature maps on the inter-pixel correlations, while the LBP baed method only looks at a circle. In this way, deep-convolutional-models "jam" an immense amount of spatial pixel information into the channel dimension which we used calculate the Gram Matrix. Something that was not clear to me however, was that the VGG based method outperformed Resnet50 based approaches even though that model has a lower accuracy on standard datasets such as ImageNet etc. This may be due to architectural differences in VGG that lend itself more to textural information encoded in the feature map.

6.5 Channel Normalization Parameter Results:

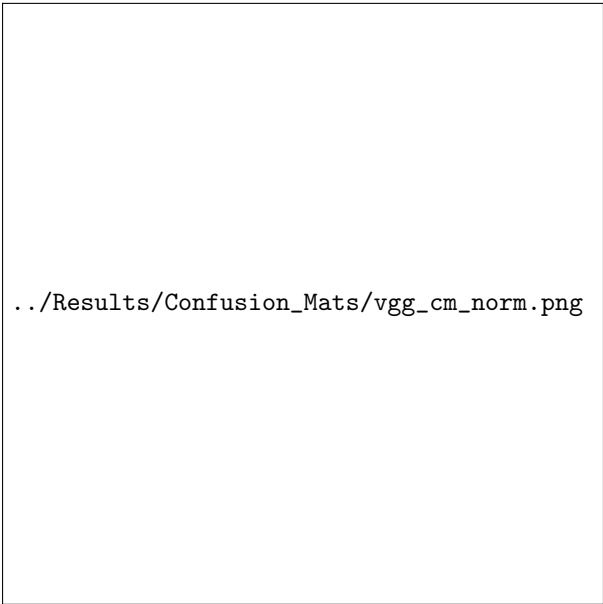
In the following results section, I include one example correct classification and one example incorrect classification for each feature map type. I do not report over all classes since some classes were fully predicted correctly. Additionally, I report accuracy metrics for the SVM training, and a confusion matrix for the prediction errors as has been reported for all other results section of this report.

6.5.1 VGG Bonus Results:

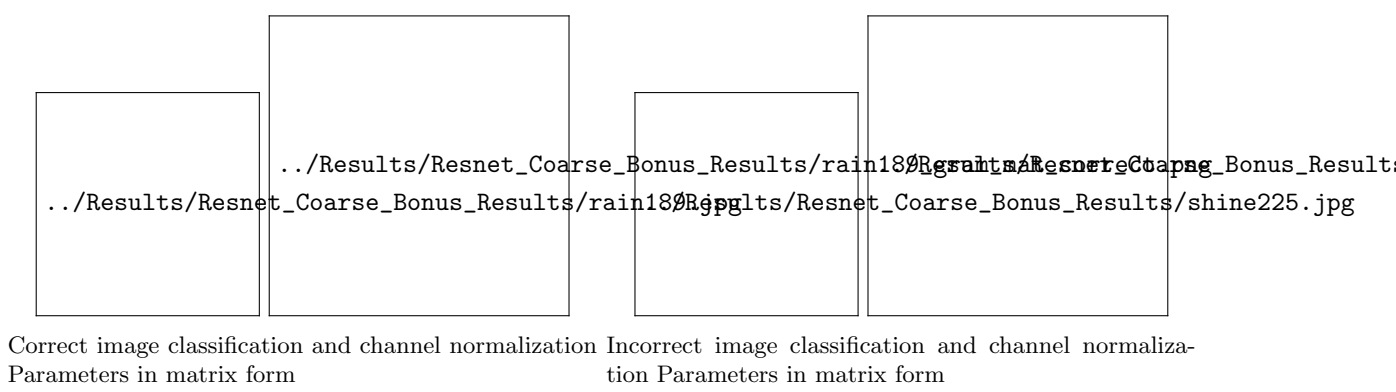


Class	Precision	Recall	F1-Score	Support
0	0.96	0.98	0.97	50
1	1.00	1.00	1.00	50
2	0.98	0.94	0.96	50
3	0.98	1.00	0.99	50
Accuracy	0.98 (200 samples)			
Macro Avg	0.98	0.98	0.98	200
Weighted Avg	0.98	0.98	0.98	200

Table 5: Classification Report for SVM Model based on LBP histograms



6.5.2 Resnet Coarse Bonus Results:

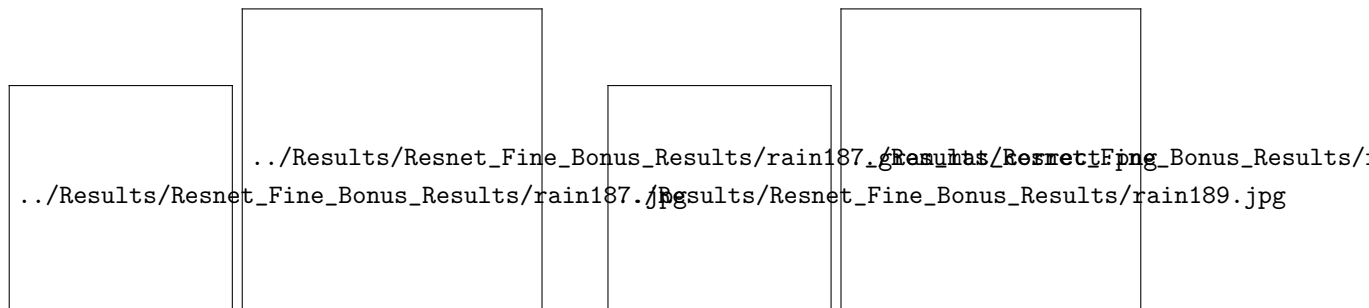


Class	Precision	Recall	F1-Score	Support
0	0.83	0.96	0.89	50
1	1.00	0.94	0.97	50
2	1.00	0.84	0.91	50
3	0.92	0.98	0.95	50
Accuracy	0.93 (200 samples)			
Macro Avg	0.94	0.93	0.93	200
Weighted Avg	0.94	0.93	0.93	200

Table 6: Classification Report for SVM Model based on Channel Normalization parameters



6.5.3 Resnet Fine Bonus Results:



Correct image classification and channel normalization Parameters in matrix form Incorrect image classification and channel normalization Parameters in matrix form

Class	Precision	Recall	F1-Score	Support
0	0.84	0.92	0.88	50
1	1.00	0.94	0.97	50
2	0.98	0.82	0.89	50
3	0.88	0.98	0.92	50
Accuracy	0.915 (200 samples)			
Macro Avg	0.92	0.91	0.92	200
Weighted Avg	0.92	0.92	0.92	200

Table 7: Classification Report for SVM Model based on Channel Normalization parameters



7 Full Code Printout:

Included below is the printout for my entire code for this assignment. It is important to note that since this is a conversion from a python notebook to python code, there could be artifacts in the code that would not be present otherwise.

```

1 # %%
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns

```

```

6 from tqdm import tqdm
7 import pandas as pd
8 from BitVector import BitVector
9 import os
10 from sklearn.svm import SVC
11 from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
12 import re
13 import pickle
14 from vgg_and_resnet import *
15 import torch.nn.functional as F
16
17 # %%
18 def img_BGR_to_HSV(img):
19     img = img.astype(np.float32)
20     img_hsv = np.zeros_like(img)
21
22     # Calculate key parameters through the channel axis
23     M = np.max(img, axis=2)
24     m = np.min(img, axis=2)
25     c = M - m
26     V = M
27
28     # For the rows, if the max is in the first column, etc
29     h0_mask = (M == img[:, :, 2]) & (c != 0) # M == R, c!=0
30     h1_mask = (M == img[:, :, 1]) & (c != 0) # M == G, c!=0
31     h2_mask = (M == img[:, :, 0]) & (c != 0) # M == B, c!=0
32     c_mask = (c == 0) # c == 0
33
34     # Calculate H Values for each row
35     # We don't just want to use the mask since c can be zero for greyscale. So we want
36     # to only compute on the masks, by checking for where to input values in first.
37     with np.errstate(divide='ignore', invalid='ignore'):
38         img_hsv[:, :, 0] = np.where(h0_mask, (60 * ((img[:, :, 1] - img[:, :, 0]) / c)
39         % 6)), img_hsv[:, :, 0])
40         img_hsv[:, :, 0] = np.where(h1_mask, (60 * ((img[:, :, 0] - img[:, :, 2]) / c +
41         2)), img_hsv[:, :, 0])
42         img_hsv[:, :, 0] = np.where(h2_mask, (60 * ((img[:, :, 2] - img[:, :, 1]) / c +
43         4)), img_hsv[:, :, 0])
44         img_hsv[:, :, 0][c_mask] = 0 # No divide by 0 errors are possible here
45     # To follow opencv formatting, I will rescale the hue angles to 180deg instead of
46     # 360
47     img_hsv[:, :, 0] /= 2
48
49     # Fill in with correct values for the S column: (c/V)
50     img_hsv[:, :, 1][V != 0] = c[V != 0]/V[V != 0] * 255
51
52     # Fill in V col
53     img_hsv[:, :, 2] = V
54
55     return np.ceil(img_hsv).astype(np.uint8)
56
57 # %%
58 class LBP():
59     def __init__(self, R, P):
60         self.R = R
61         self.P = P
62
63     def run_lbp(self, img_path):
64         # Read image and convert it to HSV, then use the H channel for all downstream
65         # tasks.
66         img_bgr = cv2.imread(img_path)
67         img_hsv = img_BGR_to_HSV(img_bgr)
68         img_h = img_hsv[:, :, 0]
69
70         # Create padded image of size (64,64) for more feasible computation
71         img_h_sized = cv2.resize(img_h, (62,62), interpolation=cv2.INTER_AREA)
72         img_h_pad = np.pad(img_h_sized, pad_width=1, mode="constant", constant_values=0)
73
74         # Initialize the histogram vector for the image: (We allow a max index of P + 1
75         # 0->9 in this case)
76         lbp_histogram = np.zeros(self.P + 2)
77
78         # Loop through all possible LBP centers:
79         for y in range(self.R, img_h_pad.shape[0]-self.R):

```

```

72     for x in range(self.R, img_h_pad.shape[1]-self.R):
73         center_value = img_h_pad[y, x] # Scalar due to greyscale
74         p = np.zeros(8)
75
76         # Check the cardinal direction points (up,down,left,right)
77         if img_h_pad[y+1][x] > center_value:
78             p[0] = 1
79         if img_h_pad[y][x+1] > center_value:
80             p[2] = 1
81         if img_h_pad[y-1][x] > center_value:
82             p[4] = 1
83         if img_h_pad[y][x-1] > center_value:
84             p[6] = 1
85
86         # We also have to check the diagonals.
87         # To calculate the pixel values at these diagonal points, we need to do
pixel-interpolation
88         # We also apply thresholding on the interpolated points compared to the
center to determine 0/1.
89         # Top right point
90         p[1] = center_value * (1 - 0.707) * (1 - 0.707) + \
91             img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
92             img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
93             img_h_pad[y+1][x+1] * 0.707 * 0.707
94         p[1] = 1 if p[1] > center_value else 0
95
96         # Bottom right point
97         p[3] = center_value * (1 - 0.707) * (1 - 0.707) + \
98             img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
99             img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
100             img_h_pad[y-1][x+1] * 0.707 * 0.707
101         p[3] = 1 if p[3] > center_value else 0
102
103         # Bottom left point
104         p[5] = center_value * (1 - 0.707) * (1 - 0.707) + \
105             img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
106             img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
107             img_h_pad[y-1][x-1] * 0.707 * 0.707
108         p[5] = 1 if p[5] > center_value else 0
109
110         # Top left point
111         p[7] = center_value * (1 - 0.707) * (1 - 0.707) + \
112             img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
113             img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
114             img_h_pad[y+1][x-1] * 0.707 * 0.707
115         p[7] = 1 if p[7] > center_value else 0
116
117         # Now that we have out bitvector representation for the circle of points
around the
118         # We want to find the unique min bitvector to represent the value at
center
119         # We do this through circular bit-shifts to find the minimal
representation:
120         # This method is from Avi Kak's implementation in lecture 16
121         bv = BitVector(bitlist=p)
122         min_val = min([int(bv<<i) for i in range(8)])
123         min_bv = BitVector(intVal=min_val, size=len(p))
124
125         # Lastly, we use this min-bv value to get the final encoding for that
point
126         # So we create a min-int-val based integer representation of the binary
pattern
127         # From Avi's Notes:
128         # - If the minIntVal representation involves more than two runs, encode
it by the integer P + 1
129         # - Else, if the minIntVal representation consists of all 0's, represent
it be the encoding 0.
130         # - Else, if the minIntVal representation consists of all 1's, represent
it by the encoding P.
131         # - Else: the minIntVal representation of a binary pattern has exactly
two runs, that is,
132         #         a run of 0s followed by a run of 1s, represent the pattern by
the number of 1's in the second run

```

```

133         num_runs = len(min_bv.runs())
134
135         encoding = None
136         # Mix of 1s and 0s
137         if num_runs > 2:
138             encoding = self.P + 1
139         # All 0s (8 of them)
140         elif min_bv.int_val() == 0 and num_runs == 1:
141             encoding = self.P
142         # 8 1s
143         elif min_bv.int_val() == 255 and num_runs == 1:
144             encoding = self.P
145         # Number of 1s in the second pattern if it is a run of all 0s then 1s
146         else:
147             encoding = len(min_bv.runs()[1])
148         lbp_histogram[encoding] += 1
149     return lbp_histogram
150
151 # %%
152 class MySVM():
153     def __init__(self):
154         self.classifier = SVC(decision_function_shape="ovr")
155
156     def fit(self, features, labels):
157         # Train the classifier on the train data/labels
158         self.classifier.fit(features, labels)
159
160     def predict(self, features):
161         # Predict the labels for the tes data
162         return self.classifier.predict(features)
163
164     def fit_predict(self, features, labels):
165         # Fit and predict on the same data
166         self.classifier.fit(features, labels)
167         return self.classifier.predict(features)
168
169     def score(self, predicted_labels, true_labels):
170         # Returns the mean accuracy using the test data and labels.
171         return accuracy_score(true_labels, predicted_labels), classification_report(
            true_labels, predicted_labels)
172
173 # %%
174 R = 1
175 P = 8
176 image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    training/")
177 lbp_hist_list = []
178 labels_list = []
179 progress_bar = tqdm(image_list, desc="Training Loop")
180 image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
181
182 for image_name in progress_bar:
183     try:
184         image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    training/" + image_name
185         image_type = re.split(r"([0-9]+)", image_name)[0]
186         label = image_type_to_label[image_type]
187
188         lbp_hist = LBP(R=R, P=P).run_lbp(img_path=image_path)
189         lbp_hist_list.append(lbp_hist)
190
191         # Fill in with image name -> index for training
192         labels_list.append(label)
193     except Exception as e:
194         print("This image did not work: ", image_name)
195
196 # %%
197 svm = MySVM()
198 svm.fit(lbp_hist_list, labels_list)
199
200 # %%
201 result_dict = {"lbp_hist_list": lbp_hist_list, "labels_list": labels_list}

```

```

203 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/lbp_hists.pkl", "wb") as file
204     :
205     pickle.dump(result_dict, file)
206
207 # %%
208 test_image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/
209 data/testing/")
210 test_lbp_hist_list = []
211 test_labels_list = []
212 image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
213 test_progress_bar = tqdm(test_image_list, desc="Testing Loop")
214
215 for image_name in test_progress_bar:
216     try:
217         image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
218 testing/" + image_name
219         image_type = re.split(r"([0-9]+)", image_name)[0]
220         label = image_type_to_label[image_type]
221
222         lbp_hist = LBP(R=R, P=P).run_lbp(img_path=image_path)
223         test_lbp_hist_list.append(lbp_hist)
224
225         # Add in labels based on image name
226         test_labels_list.append(label)
227     except Exception as e:
228         print("This image did not work: ", image_name)
229
230 # %%
231 test_result_dict = {"test_lbp_hist_list": test_lbp_hist_list, "test_labels_list":
232 test_labels_list}
233 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/test_lbp_hists.pkl", "wb") as
234 file:
235     pickle.dump(test_result_dict, file)
236
237 # %%
238 predicted_labels = svm.predict(test_lbp_hist_list)
239
240 # %%
241 accuracy, class_report = svm.score(predicted_labels, test_labels_list)
242
243 # %%
244 confusion_mat = confusion_matrix(test_labels_list, predicted_labels)
245
246 plt.figure(figsize=(8, 6))
247 sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
248 plt.xlabel('Predicted Labels')
249 plt.ylabel('True Labels')
250 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
251 plt.show()
252
253 # %% [markdown]
254 # # Get results for LBP histograms & images success/failure
255
256 # %%
257 lbp_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/LBP_Results/"
258 lbp_hist_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/LBP_Hists/"
259 # I only want to save 1 positive match example and 1 negative match example for each
260 class
261 # The class is therefore the first number, and the second number is for matching labels
262 or not
263 results_gotten = {"01": 0, "00": 0,
264                   "11": 0, "10": 0,
265                   "21": 0, "20": 0,
266                   "31": 0, "30": 0}
267
268 for image_name, test_lbp_hist, test_label, pred_label in zip(test_image_list,
269 test_lbp_hist_list, test_labels_list, predicted_labels):
270     encoding = str(test_label)
271     correct = ""
272     if test_label == pred_label:
273         encoding += "1"
274         correct = "correct"
275     else:

```

```

268         encoding += "0"
269         correct = "false"
270
271     if results_gotten[encoding] == 0:
272         # New type of result to save
273         results_gotten[encoding] += 1
274
275         # Save the resize testing image
276         image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
277         img = cv2.imread(image_path)
278         img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
279         cv2.imwrite(lbp_hist_path+image_name, img_resized)
280
281         # Save the histogram plot
282         plt.figure(figsize=(8,6))
283         plt.bar(range(len(test_lbp_hist)), test_lbp_hist, color='blue') # Customize
color as needed
284         plt.tight_layout()
285         # Save the plot to a file
286         plt.savefig(lbp_hist_path+image_name[:-4] + "_lbp_hist_" + correct + ".png",
format='png', dpi=300)
287         plt.close()
288
289 # %% [markdown]
290 # # Feature Map Extraction
291
292 # %%
293 # We run this once, and save all of the feature maps for all of the images to save
computation time during debugging
294 class FeatureMapper():
295     def __init__(self):
296         pass
297     def get_resized_img_input(self, img_path):
298         img = cv2.imread(img_path)
299         # Convert images to RGB due to how RESNET and VGG expect inputs
300         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
301
302         # Create padded image of size (256,256) for more feasilbe computation
303         img = cv2.resize(img, (256,256), interpolation=cv2.INTER_AREA)
304         return img
305
306     def get_feature_map_vgg(self, img_path):
307         img = self.get_resized_img_input(img_path)
308
309         # The next three lines are from the tutorial included in the instructions
310         vgg = VGG19()
311         vgg.load_weights('vgg_normalized.pth')
312         vgg_feature = vgg(img)
313         return vgg_feature
314
315     def get_feature_map_resnet(self, img_path):
316         img = self.get_resized_img_input(img_path)
317
318         # The next three lines are from the tutorial included in the instructions
319         encoder_name='resnet50'
320         resnet = CustomResNet(encoder=encoder_name)
321         resnet_feat_coarse, resnet_feat_fine = resnet(img)
322         return resnet_feat_coarse, resnet_feat_fine
323
324 # %%
325 image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
training/")
326 vgg_feature_list = []
327 resnet_coarse_feature_list = []
328 resnet_fine_feature_list = []
329 progress_bar = tqdm(image_list, desc="Training Loop")
330 image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
331 img_names = []
332 labels_list = []
333 featureMapper = FeatureMapper()
334
335 for image_name in progress_bar:

```

```

336     try:
337         image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
training/" + image_name
338         image_type = re.split(r"([0-9]+)", image_name)[0]
339         label = image_type_to_label[image_type]
340
341         # Get VGG Feature Map
342         vgg_feature = featureMapper.get_feature_map_vgg(img_path=image_path)
343         vgg_feature_list.append(vgg_feature)
344
345         # Resnet Features
346         resnet_feat_coarse, resnet_feat_fine = featureMapper.get_feature_map_resnet(
img_path=image_path)
347         resnet_coarse_feature_list.append(resnet_feat_coarse)
348         resnet_fine_feature_list.append(resnet_feat_fine)
349
350         # Append the image name:
351         img_names.append(image_name)
352
353         # Fill in with image name -> index for training
354         labels_list.append(label)
355     except Exception as e:
356         print("This image did not work: ", image_name)
357         print(e)
358
359
360 # %%
361 result_dict = {"vgg_feature_list": vgg_feature_list,
362               "resnet_coarse_feature_list": resnet_coarse_feature_list,
363               "resnet_fine_feature_list": resnet_fine_feature_list,
364               "img_names": img_names,
365               "labels_list": labels_list}
366
367 # %%
368 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/training_freature_mats.pkl",
"wb") as file:
369     pickle.dump(result_dict, file)
370
371 # %%
372 test_image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/
data/testing/")
373 test_vgg_feature_list = []
374 test_resnet_coarse_feature_list = []
375 test_resnet_fine_feature_list = []
376 test_img_names = []
377 test_labels_list = []
378 image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
379 featureMapper = FeatureMapper()
380 test_progress_bar = tqdm(test_image_list, desc="Testing Loop")
381
382 for image_name in test_progress_bar:
383     try:
384         image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
385         image_type = re.split(r"([0-9]+)", image_name)[0]
386         label = image_type_to_label[image_type]
387
388         # Get VGG Feature Map
389         test_vgg_feature = featureMapper.get_feature_map_vgg(img_path=image_path)
390         test_vgg_feature_list.append(test_vgg_feature)
391
392         # Resnet Features
393         test_resnet_feat_coarse, test_resnet_feat_fine = featureMapper.
get_feature_map_resnet(img_path=image_path)
394         test_resnet_coarse_feature_list.append(test_resnet_feat_coarse)
395         test_resnet_fine_feature_list.append(test_resnet_feat_fine)
396
397         # Append the image name:
398         test_img_names.append(image_name)
399
400         # Fill in with image name -> index for training
401         test_labels_list.append(label)
402     except Exception as e:

```

```

403         print("This image did not work: ", image_name)
404         print(e)
405
406
407 # %%
408 test_result_dict = {"test_vgg_feature_list": test_vgg_feature_list,
409                    "test_resnet_coarse_feature_list": test_resnet_coarse_feature_list,
410                    "test_resnet_fine_feature_list": test_resnet_fine_feature_list,
411                    "test_img_names": test_img_names,
412                    "test_labels_list": test_labels_list}
413
414 # %%
415 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/testing_freature_mats.pkl", "
416           wb") as file:
417     pickle.dump(result_dict, file)
418
419 # %% [markdown]
420 # # Gram Matrix Calculation:
421
422 # %%
423 def get_gram_matrix(feature_mat_list):
424     f_mats = np.array(feature_mat_list)
425     N, C, H, W = f_mats.shape
426     fmats_flat = f_mats.reshape(N, C, H*W)
427
428     # A Gram matrix is the feature_map * feature_map.T
429     gram_matrix = fmats_flat @ fmats_flat.transpose(0, 2, 1)
430
431     # Conver the numpy array to a pytorch tensor for biliinear interpolation in
432     # downsampling
433     # I also unsqueeze in the first dimension so that pytorch treats the final two
434     # dimensions as H,W and downsamples on those
435     # Otherwise, would read the it as Batch, Channel, Height and a missing width
436     gram_mat_tensor = torch.from_numpy(gram_matrix).unsqueeze(0)
437
438     # Lastly, we want to resize the gram matrix from 512x512 to (32,32) for easier
439     # computation
440     # We do this using bilinear interpolation
441     downsampled_matrix = F.interpolate(gram_mat_tensor, size=(32, 32), mode='bilinear',
442                                       align_corners=False)
443
444     return downsampled_matrix.squeeze().numpy()
445
446 # %%
447 vgg_gram_matrices = get_gram_matrix(vgg_feature_list)
448 resnet_coarse_gram_matrices = get_gram_matrix(resnet_coarse_feature_list)
449 resnet_fine_gram_matrices = get_gram_matrix(resnet_fine_feature_list)
450 test_vgg_gram_matrices = get_gram_matrix(test_vgg_feature_list)
451 test_resnet_coarse_gram_matrices = get_gram_matrix(test_resnet_coarse_feature_list)
452 test_resnet_fine_gram_matrices = get_gram_matrix(test_resnet_fine_feature_list)
453
454 # %%
455 # Flattening the final dimseion is required since SVM can only take in as inputs 2 dims
456 # (Batch, features)
457 vgg_gram_matrices = vgg_gram_matrices.reshape(vgg_gram_matrices.shape[0], -1)
458 resnet_coarse_gram_matrices = resnet_coarse_gram_matrices.reshape(
459     resnet_coarse_gram_matrices.shape[0], -1)
460 resnet_fine_gram_matrices = resnet_fine_gram_matrices.reshape(resnet_fine_gram_matrices.
461     shape[0], -1)
462 test_vgg_gram_matrices = test_vgg_gram_matrices.reshape(test_vgg_gram_matrices.shape[0],
463     -1)
464 test_resnet_coarse_gram_matrices = test_resnet_coarse_gram_matrices.reshape(
465     test_resnet_coarse_gram_matrices.shape[0], -1)
466 test_resnet_fine_gram_matrices = test_resnet_fine_gram_matrices.reshape(
467     test_resnet_fine_gram_matrices.shape[0], -1)
468
469 # %%
470 # Save gram matrices to a file:
471 gram_matrices = {"vgg_gram_matrices": vgg_gram_matrices,
472                 "resnet_coarse_gram_matrices": resnet_coarse_gram_matrices,
473                 "resnet_fine_gram_matrices": resnet_fine_gram_matrices,
474                 "test_vgg_gram_matrices": test_vgg_gram_matrices,
475                 "test_resnet_coarse_gram_matrices": test_resnet_coarse_gram_matrices,

```



```

465 "test_resnet_fine_gram_matrices": test_resnet_fine_gram_matrices}
466 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/all_gram_matrices.pkl", "wb")
    as file:
467     pickle.dump(gram_matrices, file)
468
469 # %% [markdown]
470 # # VGG Final Results
471
472 # %%
473 # VGG SVM:
474 svm = MySVM()
475 svm.fit(vgg_gram_matrices, labels_list)
476 vgg_predicted_labels = svm.predict(test_vgg_gram_matrices)
477 vgg_accuracy, vgg_class_report = svm.score(vgg_predicted_labels, test_labels_list)
478 print("Accuracy: ", vgg_accuracy)
479 print(vgg_class_report)
480
481 # %%
482 vgg_confusion_mat = confusion_matrix(test_labels_list, vgg_predicted_labels)
483
484 plt.figure(figsize=(8, 6))
485 sns.heatmap(vgg_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
486 plt.xlabel('Predicted Labels')
487 plt.ylabel('True Labels')
488 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
489 plt.show()
490
491 # %%
492 vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/VGG_Results/"
493 # I only want to save 1 positive match example and 1 negative match example for each
    class
494 # The class is therefore the first number, and the second number is for matching labels
    or not
495 results_gotten = {"01": 0, "00": 0,
496                  "11": 0, "10": 0,
497                  "21": 0, "20": 0,
498                  "31": 0, "30": 0}
499
500 for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
    test_vgg_gram_matrices, test_labels_list, vgg_predicted_labels):
501     encoding = str(test_label)
502     correct = ""
503     if test_label == pred_label:
504         encoding += "1"
505         correct = "correct"
506     else:
507         encoding += "0"
508         correct = "false"
509
510     if results_gotten[encoding] == 0:
511         # New type of result to save
512         results_gotten[encoding] += 1
513
514     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
515     gram_matrix = gram_matrix.reshape(32, 32)
516
517     # Save the resize testing image
518     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
519     img = cv2.imread(image_path)
520     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
521     cv2.imwrite(vgg_path+image_name, img_resized)
522
523     # Save the gram matrix to display for results section of the report
524     plt.figure(figsize=(8,6))
525
526     # Use seaborn to create a heatmap
527     sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
528     plt.tight_layout()
529     # Save the heatmap to a file
530     plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + correct + ".png", format='
png', dpi=300, bbox_inches="tight")
531     plt.close()

```

```

532
533 # %% [markdown]
534 # # Resnet Coarse Results
535
536 # %%
537 # Resnet Coarse:
538 svm = MySVM()
539 svm.fit(resnet_coarse_gram_matrices, labels_list)
540 resnet_coarse_predicted_labels = svm.predict(test_resnet_coarse_gram_matrices)
541 resnet_coarse_accuracy, resnet_coarse_class_report = svm.score(
542     resnet_coarse_predicted_labels, test_labels_list)
543 print("Accuracy: ", resnet_coarse_accuracy)
544 print(resnet_coarse_class_report)
545
546 # %%
547 resnet_coarse_confusion_mat = confusion_matrix(test_labels_list,
548     resnet_coarse_predicted_labels)
549
550 plt.figure(figsize=(8, 6))
551 sns.heatmap(resnet_coarse_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
552 plt.xlabel('Predicted Labels')
553 plt.ylabel('True Labels')
554 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
555 plt.show()
556
557 # %%
558 resnet_coarse_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/
559     Resnet_Coarse_Results/"
560 # I only want to save 1 positive match example and 1 negative match example for each
561 # class
562 # The class is therefore the first number, and the second number is for matching labels
563 # or not
564 results_gotten = {"01": 0, "00": 0,
565     "11": 0, "10": 0,
566     "21": 0, "20": 0,
567     "31": 0, "30": 0}
568
569 for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
570     test_resnet_coarse_gram_matrices, test_labels_list, resnet_coarse_predicted_labels):
571     encoding = str(test_label)
572     correct = ""
573     if test_label == pred_label:
574         encoding += "1"
575         correct = "correct"
576     else:
577         encoding += "0"
578         correct = "false"
579
580 if results_gotten[encoding] == 0:
581     # New type of result to save
582     results_gotten[encoding] += 1
583
584 # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
585 gram_matrix = gram_matrix.reshape(32, 32)
586
587 # Save the resize testing image
588 image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
589     testing/" + image_name
590 img = cv2.imread(image_path)
591 img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
592 cv2.imwrite(resnet_coarse_path+image_name, img_resized)
593
594 # Save the gram matrix to display for results section of the report
595 plt.figure(figsize=(8,6))
596
597 # Use seaborn to create a heatmap
598 sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
599 plt.tight_layout()
600 # Save the heatmap to a file
601 plt.savefig(resnet_coarse_path+image_name[:-4] + "_gram_mat_" + correct + ".png"
602     , format='png', dpi=300, bbox_inches="tight")
603 plt.close()

```

```

597 # %% [markdown]
598 # # Resnet Fine Results:
599
600 # %%
601 # VGG SVM:
602 svm = MySVM()
603 svm.fit(resnet_fine_gram_matrices, labels_list)
604 resnet_fine_predicted_labels = svm.predict(test_resnet_fine_gram_matrices)
605 resnet_fine_accuracy, resnet_fine_class_report = svm.score(resnet_fine_predicted_labels,
606 test_labels_list)
607 print("Accuracy: ", resnet_fine_accuracy)
608 print(resnet_fine_class_report)
609
610 # %%
611 resnet_fine_confusion_mat = confusion_matrix(test_labels_list,
612 resnet_fine_predicted_labels)
613
614 plt.figure(figsize=(8, 6))
615 sns.heatmap(resnet_fine_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
616 plt.xlabel('Predicted Labels')
617 plt.ylabel('True Labels')
618 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
619 plt.show()
620
621 # %%
622 resnet_fine_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/
623 Resnet_Fine_Results/"
624 # I only want to save 1 positive match example and 1 negative match example for each
625 class
626 # The class is therefore the first number, and the second number is for matching labels
627 or not
628 results_gotten = {"01": 0, "00": 0,
629 "11": 0, "10": 0,
630 "21": 0, "20": 0,
631 "31": 0, "30": 0}
632
633 for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
634 test_resnet_fine_gram_matrices, test_labels_list, resnet_fine_predicted_labels):
635 encoding = str(test_label)
636 correct = ""
637 if test_label == pred_label:
638 encoding += "1"
639 correct = "correct"
640 else:
641 encoding += "0"
642 correct = "false"
643
644 if results_gotten[encoding] == 0:
645 # New type of result to save
646 results_gotten[encoding] += 1
647
648 # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
649 gram_matrix = gram_matrix.reshape(32, 32)
650
651 # Save the resize testing image
652 image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
653 testing/" + image_name
654 img = cv2.imread(image_path)
655 img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
656 cv2.imwrite(resnet_fine_path+image_name, img_resized)
657
658 # Save the gram matrix to display for results section of the report
659 plt.figure(figsize=(8,6))
660
661 # Use seaborn to create a heatmap
662 sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
663 plt.tight_layout()
664 # Save the heatmap to a file
665 plt.savefig(resnet_fine_path+image_name[:-4] + "_gram_mat_" + correct + ".png",
666 format='png', dpi=300, bbox_inches="tight")
667 plt.close()
668
669 # %% [markdown]

```

```

662 # # Bonus: Channel Normalization Parameter Based Texture Descriptor
663
664 # %%
665 def get_normalization_params(feature_mat_list):
666     f_mats = np.array(feature_mat_list)
667
668     means = f_mats.mean(axis=(2, 3))
669     variances = f_mats.std(axis=(2, 3))
670
671     # I first stack the arrays together, and then reshape the final matrix to interleave
672     # the means and variances
673     mu_sigma_stacked = np.stack((means, variances), axis=-1)
674     channel_norm_params = mu_sigma_stacked.reshape(f_mats.shape[0], 2*f_mats.shape[1])
675
676     return channel_norm_params
677
678 # %%
679 vgg_norm_params = get_normalization_params(vgg_feature_list)
680 resnet_coarse_norm_params = get_normalization_params(resnet_coarse_feature_list)
681 resnet_fine_norm_params = get_normalization_params(resnet_fine_feature_list)
682 test_vgg_norm_params = get_normalization_params(test_vgg_feature_list)
683 test_resnet_coarse_norm_params = get_normalization_params(
684     test_resnet_coarse_feature_list)
685 test_resnet_fine_norm_params = get_normalization_params(test_resnet_fine_feature_list)
686
687 # %%
688 vgg_norm_params.shape
689
690 # %% [markdown]
691 # # Channel Norm Params VGG
692
693 # %%
694 # VGG SVM:
695 svm = MySVM()
696 svm.fit(vgg_norm_params, labels_list)
697 vgg_norm_predicted_labels = svm.predict(test_vgg_norm_params)
698 vgg_norm_accuracy, vgg_norm_class_report = svm.score(vgg_norm_predicted_labels,
699     test_labels_list)
700 print("Accuracy: ", vgg_norm_accuracy)
701 print(vgg_norm_class_report)
702
703 # %%
704 vgg_norm_confusion_mat = confusion_matrix(test_labels_list, vgg_norm_predicted_labels)
705
706 plt.figure(figsize=(8, 6))
707 sns.heatmap(vgg_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
708 plt.xlabel('Predicted Labels')
709 plt.ylabel('True Labels')
710 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
711 plt.show()
712
713 # %%
714 vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/VGG_Bonus_Results/"
715 # I only want to save 1 positive match example and 1 negative match example for each
716 # class
717 # The class is therefore the first number, and the second number is for matching labels
718 # or not
719 results_gotten = {"correct": 0, "false": 0}
720
721 for image_name, norm_params, test_label, pred_label in zip(test_progress_bar,
722     test_vgg_norm_params, test_labels_list, vgg_norm_predicted_labels):
723     correct = ""
724     if test_label == pred_label:
725         encoding = "correct"
726     else:
727         encoding = "false"
728
729     if results_gotten[encoding] == 0:
730         # New type of result to save
731         results_gotten[encoding] += 1
732
733     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
734     norm_params = norm_params.reshape(32, 32)

```

```

729
730     # Save the resize testing image
731     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
732     img = cv2.imread(image_path)
733     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
734     cv2.imwrite(vgg_path+image_name, img_resized)
735
736     # Save the gram matrix to display for results section of the report
737     plt.figure(figsize=(8,6))
738
739     # Use seaborn to create a heatmap
740     sns.heatmap(norm_params, cmap="viridis", cbar=True)
741     plt.tight_layout()
742     # Save the heatmap to a file
743     plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format=
'png', dpi=300, bbox_inches="tight")
744     plt.close()
745
746 # %% [markdown]
747 # # Resnet Coarse Results
748
749 # %%
750 # VGG SVM:
751 svm = MySVM()
752 svm.fit(resnet_coarse_norm_params, labels_list)
753 resnet_coarse_norm_predicted_labels = svm.predict(test_resnet_coarse_norm_params)
754 resnet_coarse_norm_accuracy, resnet_coarse_norm_class_report = svm.score(
    resnet_coarse_norm_predicted_labels, test_labels_list)
755 print("Accuracy: ", resnet_coarse_norm_accuracy)
756 print(resnet_coarse_norm_class_report)
757
758 # %%
759 resnet_coarse_norm_confusion_mat = confusion_matrix(test_labels_list,
    resnet_coarse_norm_predicted_labels)
760
761 plt.figure(figsize=(8, 6))
762 sns.heatmap(resnet_coarse_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=
    False)
763 plt.xlabel('Predicted Labels')
764 plt.ylabel('True Labels')
765 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
766 plt.show()
767
768 # %%
769 vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/
Resnet_Coarse_Bonus_Results/"
770 # I only want to save 1 positive match example and 1 negative match example for each
    class
771 # The class is therefore the first number, and the second number is for matching labels
    or not
772 results_gotten = {"correct": 0, "false": 0}
773
774 for image_name, norm_params, test_label, pred_label in zip(test_progress_bar,
    test_resnet_coarse_norm_params, test_labels_list,
    resnet_coarse_norm_predicted_labels):
775     correct = ""
776     if test_label == pred_label:
777         encoding = "correct"
778     else:
779         encoding = "false"
780
781     if results_gotten[encoding] == 0:
782         # New type of result to save
783         results_gotten[encoding] += 1
784
785     # Convert the Norm Params back from (N,2048) -> (N, 32,32) for display
786     # For this calculation, I need first downsample the image from 2048->1024 by
    taking only the even indices and then I can represent the matrix as (32,32)
787     norm_params = norm_params[:,2] # Extract even indices
788     norm_params = norm_params.reshape(32, 32)
789
790     # Save the resize testing image

```

```

791     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
792     img = cv2.imread(image_path)
793     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
794     cv2.imwrite(vgg_path+image_name, img_resized)
795
796     # Save the gram matrix to display for results section of the report
797     plt.figure(figsize=(8,6))
798
799     # Use seaborn to create a heatmap
800     sns.heatmap(norm_params, cmap="viridis", cbar=True)
801     plt.tight_layout()
802     # Save the heatmap to a file
803     plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format=
'png', dpi=300, bbox_inches="tight")
804     plt.close()
805
806 # %% [markdown]
807 # # Resent Fine Results:
808
809 # %%
810 # VGG SVM:
811 svm = MySVM()
812 svm.fit(resnet_fine_norm_params, labels_list)
813 resnet_fine_norm_predicted_labels = svm.predict(test_resnet_fine_norm_params)
814 resnet_fine_norm_accuracy, resnet_fine_norm_class_report = svm.score(
    resnet_fine_norm_predicted_labels, test_labels_list)
815 print("Accuracy: ", resnet_fine_norm_accuracy)
816 print(resnet_fine_norm_class_report)
817
818 # %%
819 resnet_fine_norm_confusion_mat = confusion_matrix(test_labels_list,
    resnet_fine_norm_predicted_labels)
820
821 plt.figure(figsize=(8, 6))
822 sns.heatmap(resnet_fine_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=
    False)
823 plt.xlabel('Predicted Labels')
824 plt.ylabel('True Labels')
825 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
826 plt.show()
827
828 # %%
829 vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/Resnet_Fine_Bonus_Results/"
830
831 # I only want to save 1 positive match example and 1 negative match example for each
class
832 # The class is therefore the first number, and the second number is for matching labels
or not
833 results_gotten = {"correct": 0, "false": 0}
834
835 for image_name, norm_params, test_label, pred_label in zip(test_progress_bar,
    test_resnet_fine_norm_params, test_labels_list, resnet_fine_norm_predicted_labels):
836     correct = ""
837     if test_label == pred_label:
838         encoding = "correct"
839     else:
840         encoding = "false"
841
842     if results_gotten[encoding] == 0:
843         # New type of result to save
844         results_gotten[encoding] += 1
845
846     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
847     norm_params = norm_params.reshape(32, 32)
848
849     # Save the resize testing image
850     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
851     img = cv2.imread(image_path)
852     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
853     cv2.imwrite(vgg_path+image_name, img_resized)

```

```

854     # Save the gram matrix to display for results section of the report
855     plt.figure(figsize=(8,6))
856
857     # Use seaborn to create a heatmap
858     sns.heatmap(norm_params, cmap="viridis", cbar=True)
859     plt.tight_layout()
860     # Save the heatmap to a file
861     plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format=
'png', dpi=300, bbox_inches="tight")
862     plt.close()

```