

ECE 66100 Homework #6

by

Adrien Dubois (dubois6@purdue.edu)

October 31, 2024

Contents

1 Theory Questions	1
1.1 Question 1:	1
2 RGB to HSV:	2
3 Extracting LBP Histograms:	3
3.1 Algorithm Description:	3
3.2 Code Implementation:	4
4 Gram Matrix based texture extraction:	6
4.1 Gram Matrix	6
4.2 Code Implementation:	6
5 Extra Credit: Channel Normalization Parameter based Texture Extraction:	6
5.1 Implementation:	6
6 Results:	7
6.1 Dataset Description:	7
6.2 LBP Results:	7
6.3 Gram Matrix Results:	9
6.3.1 VGG-19 Results:	9
6.3.2 Resnet50-Coarse Results:	11
6.3.3 Resnet50-Fine Results:	14
6.4 Discussion of results:	15
6.5 Channel Normalization Parameter Results:	15
6.5.1 VGG Bonus Results:	16
6.5.2 Resnet Coarse Bonus Results:	16
6.5.3 Resnet Fine Bonus Results:	17
7 Full Code Printout:	18

1 Theory Questions

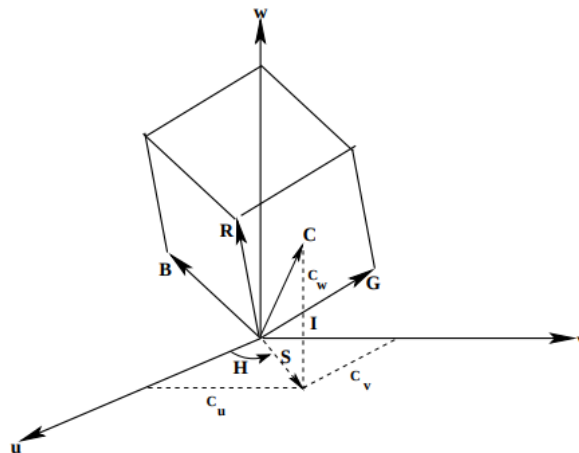
1.1 Question 1:

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.

- 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.

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.


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

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

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         % 6)), 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):

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

- 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.

```

```

35         # To calculate the pixel values at these diagonal points, we need to do
pixel-interpolation
36         # We also apply thresholding on the interpolated points compared to the
center to determine 0/1.
37         # Top right point
38         p[1] = center_value * (1 - 0.707) * (1 - 0.707) + \
39             img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
40             img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
41             img_h_pad[y+1][x+1] * 0.707 * 0.707
42         p[1] = 1 if p[1] > center_value else 0
43
44         # Bottom right point
45         p[3] = center_value * (1 - 0.707) * (1 - 0.707) + \
46             img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
47             img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
48             img_h_pad[y-1][x+1] * 0.707 * 0.707
49         p[3] = 1 if p[3] > center_value else 0
50
51         # Bottom left point
52         p[5] = center_value * (1 - 0.707) * (1 - 0.707) + \
53             img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
54             img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
55             img_h_pad[y-1][x-1] * 0.707 * 0.707
56         p[5] = 1 if p[5] > center_value else 0
57
58         # Top left point
59         p[7] = center_value * (1 - 0.707) * (1 - 0.707) + \
60             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
around the center
66         # We want to find the unique min bitvector to represent the value at
that point
67         # We do this through circular bit-shifts to find the minimal
representation:
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
point
74         # So we create a min-int-val based integer representation of the binary
pattern
75         # From Avi's Notes:
76         # - If the minIntVal representation involves more than two runs, encode
it by the integer P + 1
77         # - Else, if the minIntVal representation consists of all 0's, represent
it be the encoding 0.
78         # - Else, if the minIntVal representation consists of all 1's, represent
it by the encoding P.
79         # - Else: the minIntVal representation of a binary pattern has exactly
two runs, that is,
80         #         a run of 0s followed by a run of 1s, represent the pattern by
the number of 1's in the second run
81         num_runs = len(min_bv.runs())
82
83         encoding = None
84         # Mix of 1s and 0s
85         if num_runs > 2:
86             encoding = self.P + 1
87         # All 0s (8 of them)
88         elif min_bv.int_val() == 0 and num_runs == 1:
89             encoding = self.P
90         # 8 1s
91         elif min_bv.int_val() == 255 and num_runs == 1:
92             encoding = self.P
93         # Number of 1s in the second pattern if it is a run of all 0s then 1s
94         else:
95             encoding = len(min_bv.runs()[1])

```

```

96         lbp_histogram[encoding] += 1
97     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
10    # downsampling
11    # I also unsqueeze in the first dimension so that pytorch treats the final two
12    # dimensions as H,W and downsamples on those
13    # Otherwise, would read the it as Batch, Channel, Height and a missing width
14    gram_mat_tensor = torch.from_numpy(gram_matrix).unsqueeze(0)
15
16    # Lastly, we want to resize the gram matrix from 512x512 to (32,32) for easier
17    # computation
18    # We do this using bilinear interpolation
19    downsampled_matrix = F.interpolate(gram_mat_tensor, size=(32, 32), mode='bilinear',
20    align_corners=False)
21
22    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:

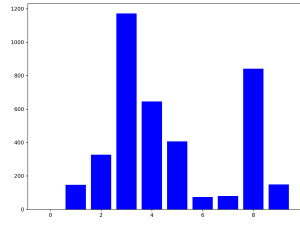
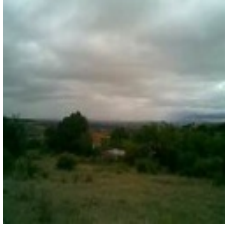
6.1 Dataset Description:

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 accuracies for all texture detectors. It is important to note that the following encoding will be used to represent the class names for the confusion matrices:

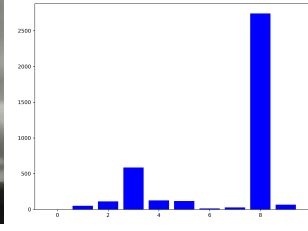
- cloudy: 0
- rain: 1
- shine: 2
- sunrise: 3

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.

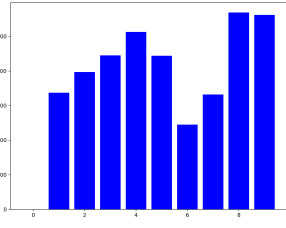


Correct image classification and LBP histogram

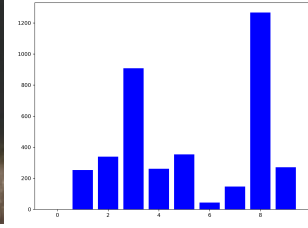
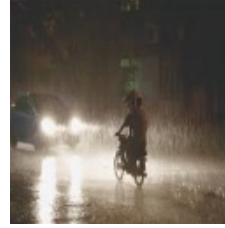


Incorrect image classification and LBP histogram

Cloudy: Image classification and histogram pairs

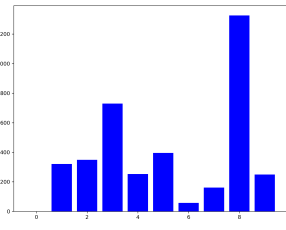
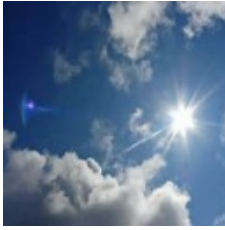


Correct image classification and LBP histogram

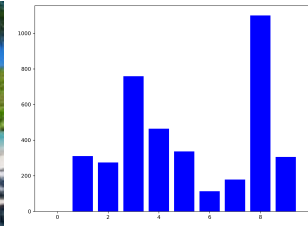


Incorrect image classification and LBP histogram

Rain: Image classification and histogram pairs

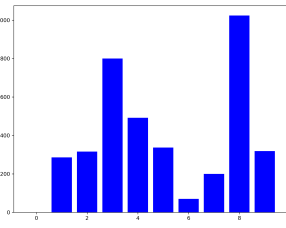


Correct image classification and LBP histogram

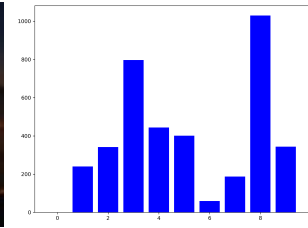
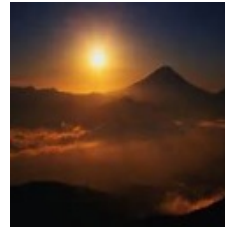


Incorrect image classification and LBP histogram

Sunshine: Image classification and histogram pairs



Correct image classification and LBP histogram



Incorrect image classification and LBP histogram

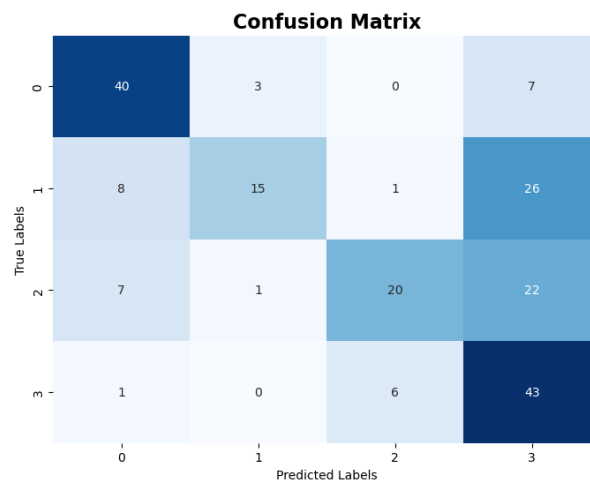
Sunrise: Image classification and histogram pairs

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

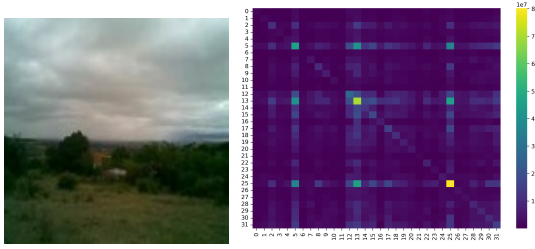
Additionally, I have generated the following confusion matrix to visualize the results in a different way:



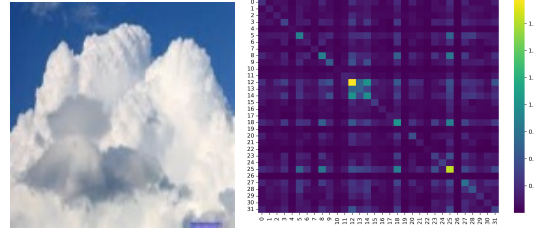
6.3 Gram Matrix Results:

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.

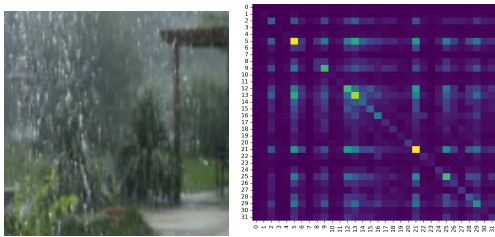


Correct image classification and Gram Matrix

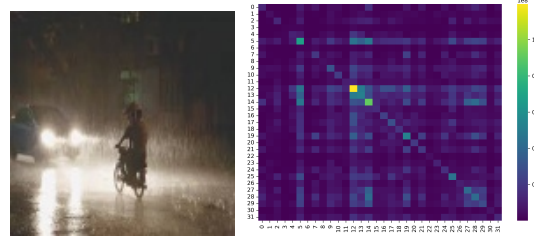


Incorrect image classification and Gram Matrix

Cloudy: Image classification and Gram Matrix pairs

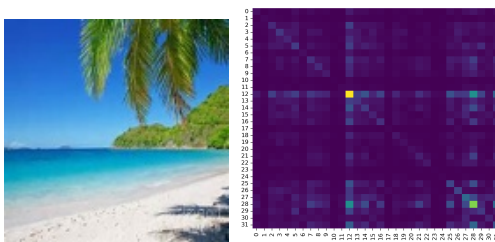


Correct image classification and Gram Matrix

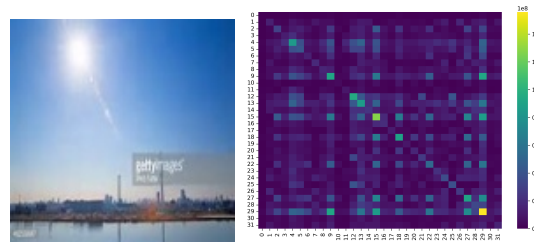


Incorrect image classification and Gram Matrix

Rain: Image classification and Gram Matrix pairs

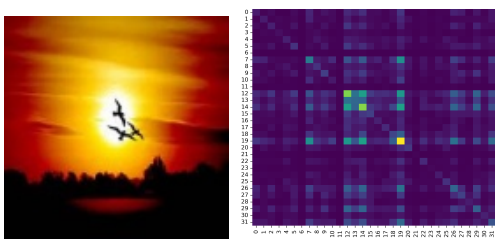


Correct image classification and Gram Matrix

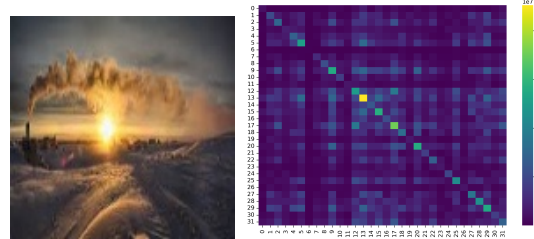


Incorrect image classification and Gram Matrix

Sunshine: Image classification and Gram Matrix pairs



Correct image classification and Gram Matrix



Incorrect image classification and Gram Matrix

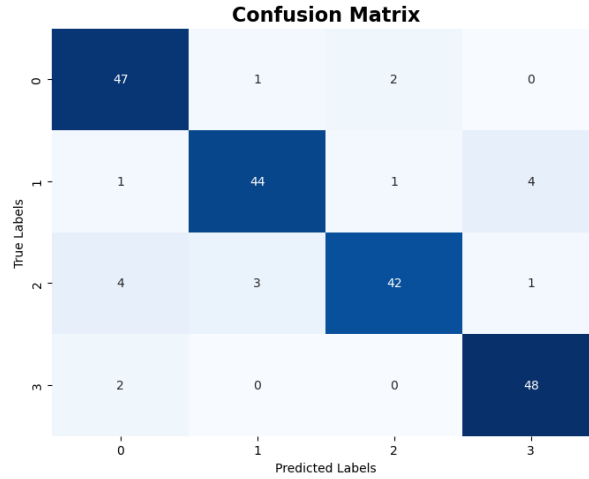
Sunrise: Image classification and Gram Matrix pairs

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:

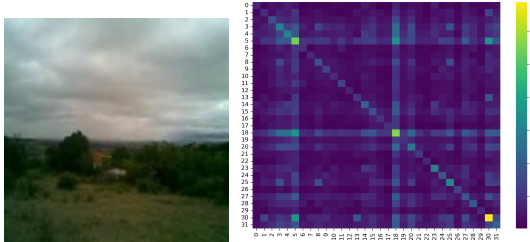
Class	Precision	Recall	F1-Score	Support
0	0.87	0.94	0.90	50
1	0.92	0.88	0.90	50
2	0.93	0.84	0.88	50
3	0.91	0.96	0.93	50
Accuracy	0.905 (200 samples)			
Macro Avg	0.91	0.90	0.90	200
Weighted Avg	0.91	0.91	0.90	200

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

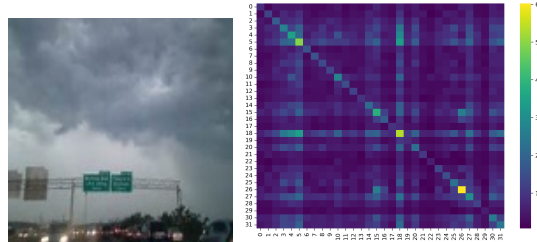


6.3.2 Resnet50-Coarse Results:

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

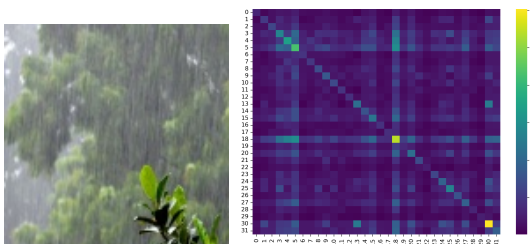


Correct image classification and Gram Matrix

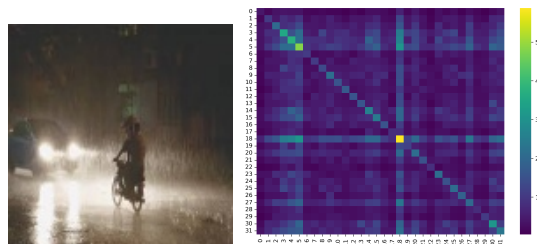


Incorrect image classification and Gram Matrix

Cloudy: Image classification and Gram Matrix pairs

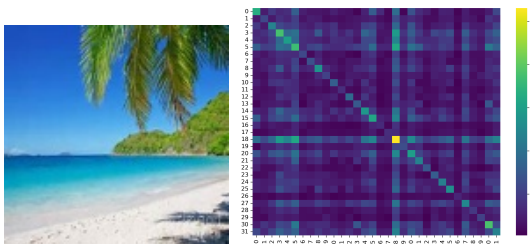


Correct image classification and Gram Matrix

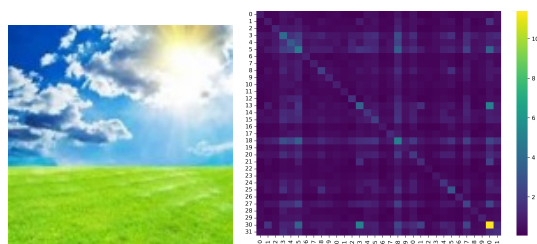


Incorrect image classification and Gram Matrix

Rain: Image classification and Gram Matrix pairs

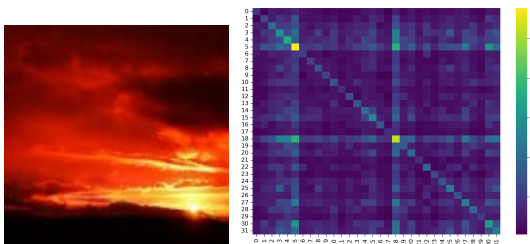


Correct image classification and Gram Matrix

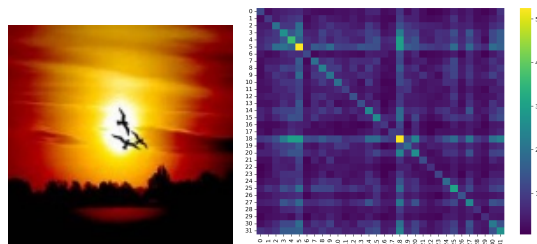


Incorrect image classification and Gram Matrix

Sunshine: Image classification and Gram Matrix pairs



Correct image classification and Gram Matrix



Incorrect image classification and Gram Matrix

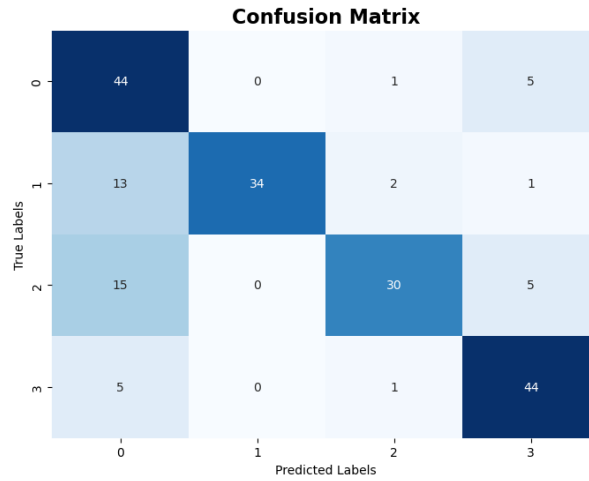
Sunrise: Image classification and Gram Matrix pairs

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:

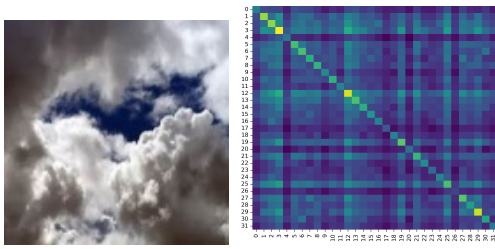
Additionally, I have generated the following confusion matrix to visualize the results in a different way:

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

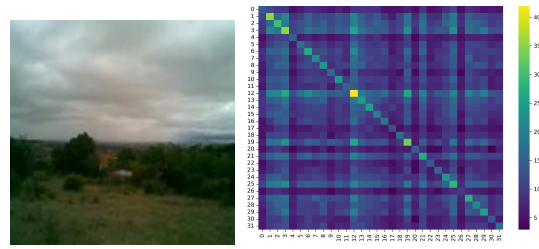
Table 3: Classification Report for SVM Model based on Resnet50-Coarse Gram Matrices



6.3.3 Resnet50-Fine Results:

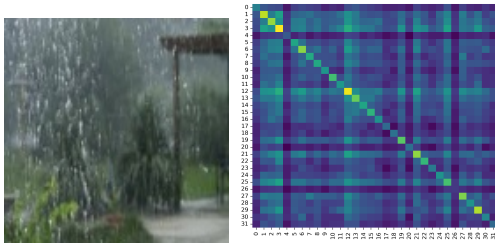


Correct image classification and Gram Matrix

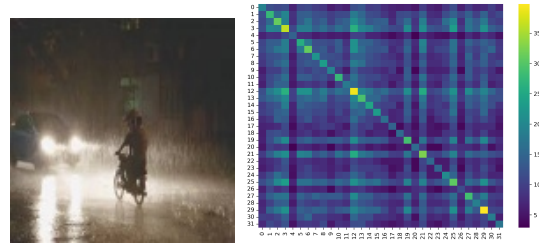


Incorrect image classification and Gram Matrix

Cloudy: Image classification and Gram Matrix pairs

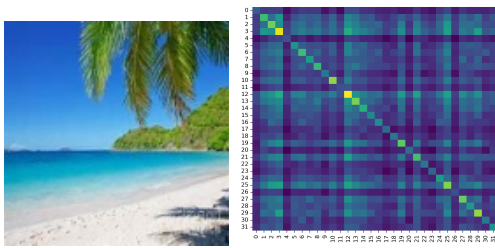


Correct image classification and Gram Matrix

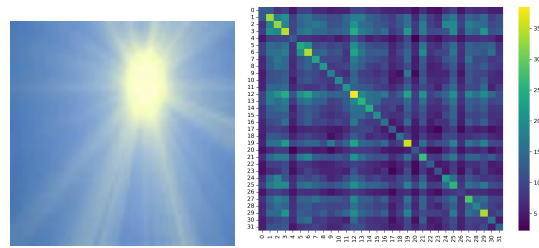


Incorrect image classification and Gram Matrix

Rain: Image classification and Gram Matrix pairs

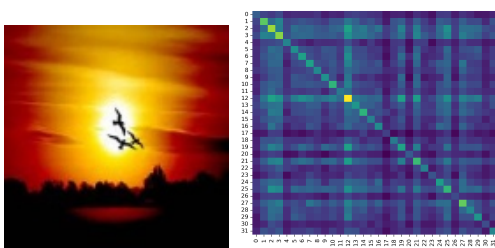


Correct image classification and Gram Matrix

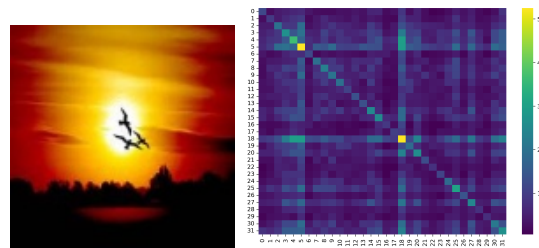


Incorrect image classification and Gram Matrix

Sunshine: Image classification and Gram Matrix pairs



Correct image classification and Gram Matrix



Incorrect image classification and Gram Matrix

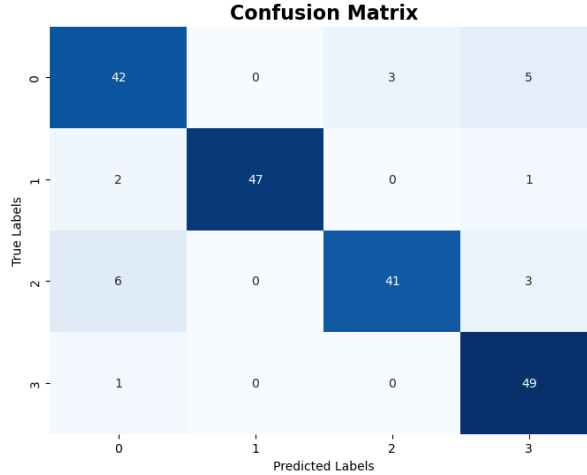
Sunrise: Image classification and Gram Matrix pairs

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:

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

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



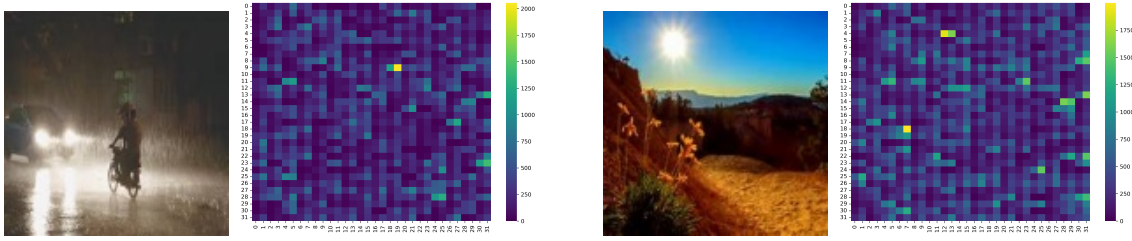
6.4 Discussion of results:

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 based 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 to 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.

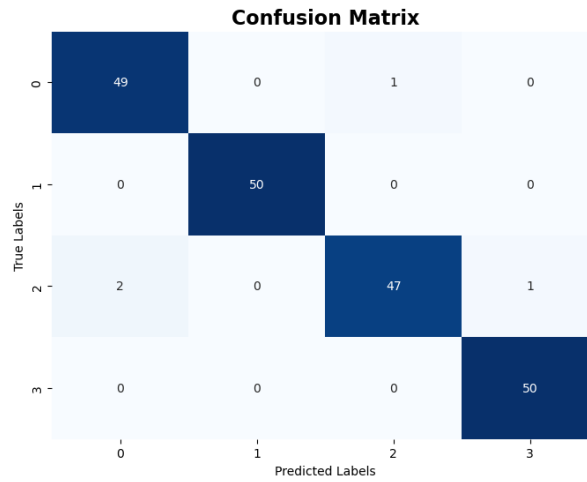
6.5.1 VGG Bonus Results:



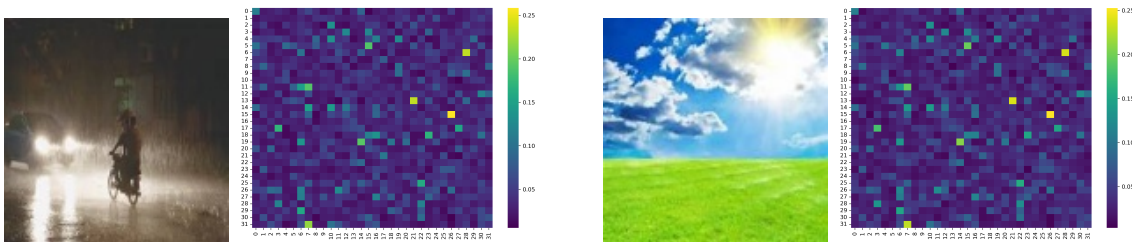
Correct image classification and channel normalization Parameters in matrix form Incorrect image classification and channel normalization Parameters in matrix form

Class	Precision	Recall	F1-Score	Support
0	0.96	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



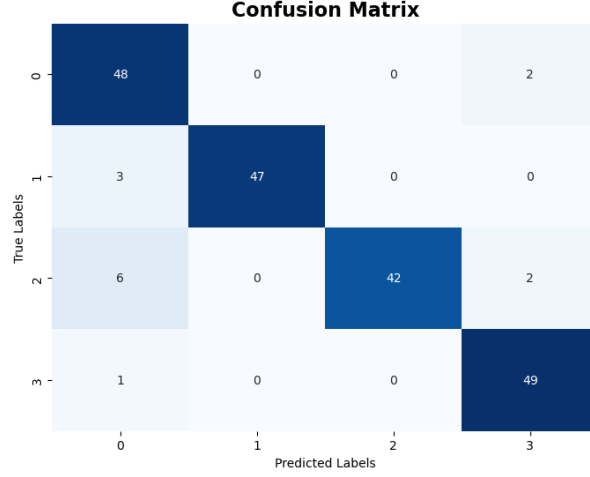
6.5.2 Resnet Coarse Bonus Results:



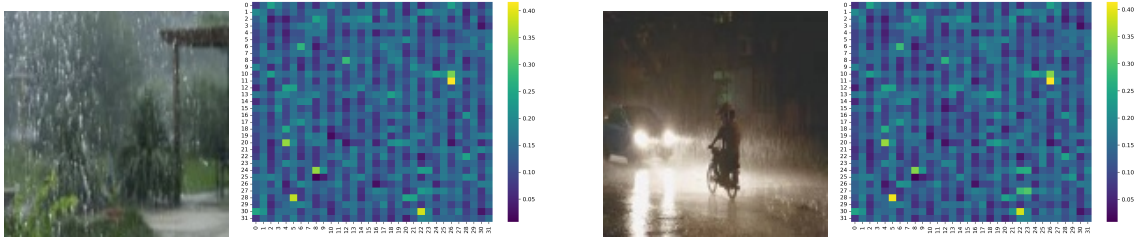
Correct image classification and channel normalization Parameters in matrix form Incorrect image classification and channel normalization Parameters in matrix form

Class	Precision	Recall	F1-Score	Support
0	0.83	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



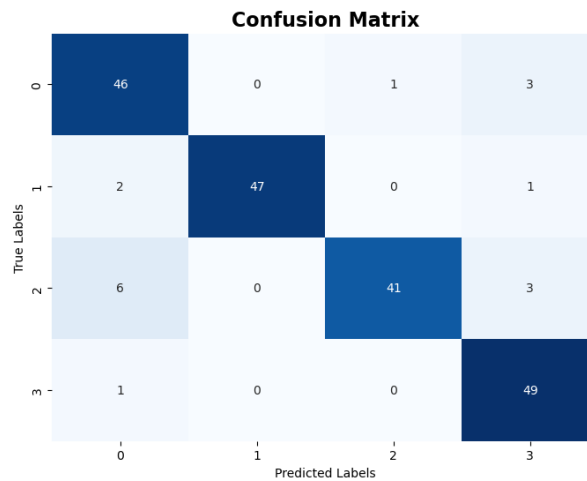
6.5.3 Resnet Fine Bonus Results:



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

```

```

41 # To follow opencv formatting, I will rescale the hue angles to 180deg instead of
42 360
43 img_hsv[:, :, 0] /= 2
44
45 # Fill in with correct values for the S column: (c/V)
46 img_hsv[:, :, 1][V != 0] = c[V != 0]/V[V != 0] * 255
47
48 # Fill in V col
49 img_hsv[:, :, 2] = V
50
51 return np.ceil(img_hsv).astype(np.uint8)
52
53 # %%
54 class LBP():
55     def __init__(self, R, P):
56         self.R = R
57         self.P = P
58     def run_lbp(self, img_path):
59         # Read image and convert it to HSV, then use the H channel for all downstream
60         # tasks.
61         img_bgr = cv2.imread(img_path)
62         img_hsv = img_BGR_to_HSV(img_bgr)
63         img_h = img_hsv[:, :, 0]
64
65         # Create padded image of size (64,64) for more feasible computation
66         img_h_sized = cv2.resize(img_h, (62,62), interpolation=cv2.INTER_AREA)
67         img_h_pad = np.pad(img_h_sized, pad_width=1, mode="constant", constant_values=0)
68
69         # Initialize the histogram vector for the image: (We allow a max index of P + 1
70         # 0->9 in this case)
71         lbp_histogram = np.zeros(self.P + 2)
72
73         # Loop through all possible LBP centers:
74         for y in range(self.R, img_h_pad.shape[0]-self.R):
75             for x in range(self.R, img_h_pad.shape[1]-self.R):
76                 center_value = img_h_pad[y, x] # Scalar due to greyscale
77                 p = np.zeros(8)
78
79                 # Check the cardinal direction points (up,down,left,right)
80                 if img_h_pad[y+1][x] > center_value:
81                     p[0] = 1
82                 if img_h_pad[y][x+1] > center_value:
83                     p[2] = 1
84                 if img_h_pad[y-1][x] > center_value:
85                     p[4] = 1
86                 if img_h_pad[y][x-1] > center_value:
87                     p[6] = 1
88
89                 # We also have to check the diagonals.
90                 # To calculate the pixel values at these diagonal points, we need to do
91                 # pixel-interpolation
92                 # We also apply thresholding on the interpolated points compared to the
93                 # center to determine 0/1.
94                 # Top right point
95                 p[1] = center_value * (1 - 0.707) * (1 - 0.707) + \
96                     img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
97                     img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
98                     img_h_pad[y+1][x+1] * 0.707 * 0.707
99                 p[1] = 1 if p[1] > center_value else 0
100
101                 # Bottom right point
102                 p[3] = center_value * (1 - 0.707) * (1 - 0.707) + \
103                     img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
104                     img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
105                     img_h_pad[y-1][x+1] * 0.707 * 0.707
106                 p[3] = 1 if p[3] > center_value else 0
107
108                 # Bottom left point
109                 p[5] = center_value * (1 - 0.707) * (1 - 0.707) + \
110                     img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
111                     img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
112                     img_h_pad[y-1][x-1] * 0.707 * 0.707
113                 p[5] = 1 if p[5] > center_value else 0

```

```

109         # Top left point
110         p[7] = center_value * (1 - 0.707) * (1 - 0.707) + \
111             img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
112             img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
113             img_h_pad[y+1][x-1] * 0.707 * 0.707
114         p[7] = 1 if p[7] > center_value else 0
115
116
117         # Now that we have out bitvector representation for the circle of points
118         # We want to find the unique min bitvector to represent the value at
119         # We do this through circular bit-shifts to find the minimal
120         # representation:
121         # This method is from Avi Kak's implementation in lecture 16
122         bv = BitVector(bitlist=p)
123         min_val = min([int(bv<<1) for _ in p])
124         min_bv = BitVector(intVal=min_val, size=len(p))
125
126         # Lastly, we use this min-bv value to get the final encoding for that
127         # So we create a min-int-val based integer representation of the binary
128         # From Avi's Notes:
129         # - If the minIntVal representation involves more than two runs, encode
130         # it by the integer P + 1
131         # - Else, if the minIntVal representation consists of all 0's, represent
132         # it be the encoding 0.
133         # - Else, if the minIntVal representation consists of all 1's, represent
134         # it by the encoding P.
135         # - Else: the minIntVal representation of a binary pattern has exactly
136         # two runs, that is,
137         # a run of 0s followed by a run of 1s, represent the pattern by
138         # the number of 1's in the second run
139         num_runs = len(min_bv.runs())
140
141         encoding = None
142         # Mix of 1s and 0s
143         if num_runs > 2:
144             encoding = self.P + 1
145         # All 0s (8 of them)
146         elif min_bv.int_val() == 0 and num_runs == 1:
147             encoding = self.P
148         # 8 1s
149         elif min_bv.int_val() == 255 and num_runs == 1:
150             encoding = self.P
151         # Number of 1s in the second pattern if it is a run of all 0s then 1s
152         else:
153             encoding = len(min_bv.runs()[1])
154         lbp_histogram[encoding] += 1
155         return lbp_histogram
156
157 # %%
158 class MySVM():
159     def __init__(self):
160         self.classifier = SVC(decision_function_shape="ovr")
161
162     def fit(self, features, labels):
163         # Train the classifier on the train data/labels
164         self.classifier.fit(features, labels)
165
166     def predict(self, features):
167         # Predict the labels for the tes data
168         return self.classifier.predict(features)
169
170     def fit_predict(self, features, labels):
171         # Fit and predict on the same data
172         self.classifier.fit(features, labels)
173         return self.classifier.predict(features)
174
175     def score(self, predicted_labels, true_labels):
176         # Returns the mean accuracy using the test data and labels.

```

```

171         return accuracy_score(true_labels, predicted_labels), classification_report(
172             true_labels, predicted_labels)
173     # %%
174     R = 1
175     P = 8
176     image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
177         training/")
178     lbp_hist_list = []
179     labels_list = []
180     progress_bar = tqdm(image_list, desc="Training Loop")
181     image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
182     for image_name in progress_bar:
183         try:
184             image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
185                 training/" + image_name
186             image_type = re.split(r"([0-9]+)", image_name)[0]
187             label = image_type_to_label[image_type]
188
189             lbp_hist = LBP(R=R, P=P).run_lbp(img_path=image_path)
190             lbp_hist_list.append(lbp_hist)
191
192             # Fill in with image name -> index for training
193             labels_list.append(label)
194         except Exception as e:
195             print("This image did not work: ", image_name)
196
197     # %%
198     svm = MySVM()
199     svm.fit(lbp_hist_list, labels_list)
200
201     # %%
202     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     for image_name in test_progress_bar:
215         try:
216             image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
217                 testing/" + image_name
218             image_type = re.split(r"([0-9]+)", image_name)[0]
219             label = image_type_to_label[image_type]
220
221             lbp_hist = LBP(R=R, P=P).run_lbp(img_path=image_path)
222             test_lbp_hist_list.append(lbp_hist)
223
224             # Add in labels based on image name
225             test_labels_list.append(label)
226         except Exception as e:
227             print("This image did not work: ", image_name)
228
229     # %%
230     test_result_dict = {"test_lbp_hist_list": test_lbp_hist_list, "test_labels_list":
231         test_labels_list}
232     with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/test_lbp_hists.pkl", "wb") as
233         file:
234         pickle.dump(test_result_dict, file)
235
236     # %%
237     predicted_labels = svm.predict(test_lbp_hist_list)
238
239     # %%

```

```

236 accuracy, class_report = svm.score(predicted_labels, test_labels_list)
237
238 # %%
239 confusion_mat = confusion_matrix(test_labels_list, predicted_labels)
240
241 plt.figure(figsize=(8, 6))
242 sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
243 plt.xlabel('Predicted Labels')
244 plt.ylabel('True Labels')
245 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
246 plt.show()
247
248 # %% [markdown]
249 # # Get results for LBP histograms & images success/failure
250
251 # %%
252 lbp_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/LBP_Results/"
253 lbp_hist_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/LBP_Hists/"
254 # I only want to save 1 positive match example and 1 negative match example for each
    class
255 # The class is therefore the first number, and the second number is for matching labels
    or not
256 results_gotten = {"01": 0, "00": 0,
257                   "11": 0, "10": 0,
258                   "21": 0, "20": 0,
259                   "31": 0, "30": 0}
260
261 for image_name, test_lbp_hist, test_label, pred_label in zip(test_progress_bar,
262 test_lbp_hist_list, test_labels_list, predicted_labels):
263     encoding = str(test_label)
264     correct = ""
265     if test_label == pred_label:
266         encoding += "1"
267         correct = "correct"
268     else:
269         encoding += "0"
270         correct = "false"
271
272     if results_gotten[encoding] == 0:
273         # New type of result to save
274         results_gotten[encoding] += 1
275
276     # Save the resize testing image
277     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
278 testing/" + image_name
279     img = cv2.imread(image_path)
280     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
281     cv2.imwrite(lbp_hist_path+image_name, img_resized)
282
283     # Save the histogram plot
284     plt.figure(figsize=(8,6))
285     plt.bar(range(len(test_lbp_hist)), test_lbp_hist, color='blue') # Customize
286     color as needed
287     plt.tight_layout()
288     # Save the plot to a file
289     plt.savefig(lbp_hist_path+image_name[:-4] + "_lbp_hist_" + correct + ".png",
290 format='png', dpi=300)
291     plt.close()
292
293 # %% [markdown]
294 # # Feature Map Extraction
295
296 # %%
297 # We run this once, and save all of the feature maps for all of the images to save
298 computation time during debugging
299 class FeatureMapper():
300     def __init__(self):
301         pass
302     def get_resized_img_input(self, img_path):
303         img = cv2.imread(img_path)
304         # Convert images to RGB due to how RESNET and VGG expect inputs
305         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

```

```

302     # Create padded image of size (256,256) for more feasible 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 result_dict = {"vgg_feature_list": vgg_feature_list,
361               "resnet_coarse_feature_list": resnet_coarse_feature_list,
362               "resnet_fine_feature_list": resnet_fine_feature_list,
363               "img_names": img_names,
364               "labels_list": labels_list}
365
366 # %%
367 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/training_freature_mats.pkl",
368         "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 test_result_dict = {"test_vgg_feature_list": test_vgg_feature_list,
408                     "test_resnet_coarse_feature_list": test_resnet_coarse_feature_list,
409                     "test_resnet_fine_feature_list": test_resnet_fine_feature_list,
410                     "test_img_names": test_img_names,
411                     "test_labels_list": test_labels_list}
412
413 # %%
414 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/testing_freature_mats.pkl", "
    wb") as file:
415     pickle.dump(result_dict, file)
416
417 # %% [markdown]
418 # # Gram Matrix Calculation:
419
420 # %%
421 def get_gram_matrix(feature_mat_list):
422     f_mats = np.array(feature_mat_list)
423     N, C, H, W = f_mats.shape
424     fmats_flat = f_mats.reshape(N, C, H*W)
425
426     # A Gram matrix is the feature_map * feature_map.T
427     gram_matrix = fmats_flat @ fmats_flat.transpose(0, 2, 1)
428
429     # Conver the numpy array to a pytorch tensor for biliinear interpolation in
    downsampling
430     # I also unsqueeze in the first dimension so that pytorch treats the final two
    dimensions as H,W and downsamples on those
431     # Otherwise, would read the it as Batch, Channel, Height and a missing width
432     gram_mat_tensor = torch.from_numpy(gram_matrix).unsqueeze(0)
433
434     # Lastly, we want to resize the gram matrix from 512x512 to (32,32) for easier
    computation
435     # We do this using bilinear interpolation
436

```

```

437     downsampled_matrix = F.interpolate(gram_mat_tensor, size=(32, 32), mode='bilinear',
438 align_corners=False)
439
440     return downsampled_matrix.squeeze().numpy()
441
442 # %%
443 vgg_gram_matrices = get_gram_matrix(vgg_feature_list)
444 resnet_coarse_gram_matrices = get_gram_matrix(resnet_coarse_feature_list)
445 resnet_fine_gram_matrices = get_gram_matrix(resnet_fine_feature_list)
446 test_vgg_gram_matrices = get_gram_matrix(test_vgg_feature_list)
447 test_resnet_coarse_gram_matrices = get_gram_matrix(test_resnet_coarse_feature_list)
448 test_resnet_fine_gram_matrices = get_gram_matrix(test_resnet_fine_feature_list)
449
450 # %%
451 # Flattening the final dimseion is required since SVM can only take in as inputs 2 dims
452 # (Batch, features)
453 vgg_gram_matrices = vgg_gram_matrices.reshape(vgg_gram_matrices.shape[0], -1)
454 resnet_coarse_gram_matrices = resnet_coarse_gram_matrices.reshape(
455     resnet_coarse_gram_matrices.shape[0], -1)
456 resnet_fine_gram_matrices = resnet_fine_gram_matrices.reshape(resnet_fine_gram_matrices.
457     shape[0], -1)
458 test_vgg_gram_matrices = test_vgg_gram_matrices.reshape(test_vgg_gram_matrices.shape[0],
459     -1)
460 test_resnet_coarse_gram_matrices = test_resnet_coarse_gram_matrices.reshape(
461     test_resnet_coarse_gram_matrices.shape[0], -1)
462 test_resnet_fine_gram_matrices = test_resnet_fine_gram_matrices.reshape(
463     test_resnet_fine_gram_matrices.shape[0], -1)
464
465 # %%
466 # Save gram matrices to a file:
467 gram_matrices = {"vgg_gram_matrices": vgg_gram_matrices,
468 "resnet_coarse_gram_matrices": resnet_coarse_gram_matrices,
469 "resnet_fine_gram_matrices": resnet_fine_gram_matrices,
470 "test_vgg_gram_matrices": test_vgg_gram_matrices,
471 "test_resnet_coarse_gram_matrices": test_resnet_coarse_gram_matrices,
472 "test_resnet_fine_gram_matrices": test_resnet_fine_gram_matrices}
473 with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/all_gram_matrices.pkl", "wb")
474     as file:
475     pickle.dump(gram_matrices, file)
476
477 # %% [markdown]
478 # # VGG Final Results
479
480 # %%
481 # VGG SVM:
482 svm = MySVM()
483 svm.fit(vgg_gram_matrices, labels_list)
484 vgg_predicted_labels = svm.predict(test_vgg_gram_matrices)
485 vgg_accuracy, vgg_class_report = svm.score(vgg_predicted_labels, test_labels_list)
486 print("Accuracy: ", vgg_accuracy)
487 print(vgg_class_report)
488
489 # %%
490 vgg_confusion_mat = confusion_matrix(test_labels_list, vgg_predicted_labels)
491
492 plt.figure(figsize=(8, 6))
493 sns.heatmap(vgg_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
494 plt.xlabel('Predicted Labels')
495 plt.ylabel('True Labels')
496 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
497 plt.show()
498
499 # %%
500 vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/VGG_Results/"
501 # I only want to save 1 positive match example and 1 negative match example for each
502 class
503 # The class is therefore the first number, and the second number is for matching labels
504 or not
505 results_gotten = {"01": 0, "00": 0,
506 "11": 0, "10": 0,
507 "21": 0, "20": 0,
508 "31": 0, "30": 0}

```

```

500 for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
501 test_vgg_gram_matrices, test_labels_list, vgg_predicted_labels):
502     encoding = str(test_label)
503     correct = ""
504     if test_label == pred_label:
505         encoding += "1"
506         correct = "correct"
507     else:
508         encoding += "0"
509         correct = "false"
510
511     if results_gotten[encoding] == 0:
512         # New type of result to save
513         results_gotten[encoding] += 1
514
515     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
516     gram_matrix = gram_matrix.reshape(32, 32)
517
518     # Save the resize testing image
519     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
520 testing/" + image_name
521     img = cv2.imread(image_path)
522     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
523     cv2.imwrite(vgg_path+image_name, img_resized)
524
525     # Save the gram matrix to display for results section of the report
526     plt.figure(figsize=(8,6))
527
528     # Use seaborn to create a heatmap
529     sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
530     plt.tight_layout()
531     # Save the heatmap to a file
532     plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + correct + ".png", format=
533 'png', dpi=300, bbox_inches="tight")
534     plt.close()
535
536 # %% [markdown]
537 # # Resnet Coarse Results
538
539 # %%
540 # Resnet Coarse:
541 svm = MySVM()
542 svm.fit(resnet_coarse_gram_matrices, labels_list)
543 resnet_coarse_predicted_labels = svm.predict(test_resnet_coarse_gram_matrices)
544 resnet_coarse_accuracy, resnet_coarse_class_report = svm.score(
545     resnet_coarse_predicted_labels, test_labels_list)
546 print("Accuracy: ", resnet_coarse_accuracy)
547 print(resnet_coarse_class_report)
548
549 # %%
550 resnet_coarse_confusion_mat = confusion_matrix(test_labels_list,
551     resnet_coarse_predicted_labels)
552
553 plt.figure(figsize=(8, 6))
554 sns.heatmap(resnet_coarse_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
555 plt.xlabel('Predicted Labels')
556 plt.ylabel('True Labels')
557 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
558 plt.show()
559
560 # %%
561 resnet_coarse_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/
562 Resnet_Coarse_Results/"
563
564 # I only want to save 1 positive match example and 1 negative match example for each
565 class
566 # The class is therefore the first number, and the second number is for matching labels
567 or not
568 results_gotten = {"01": 0, "00": 0,
569                  "11": 0, "10": 0,
570                  "21": 0, "20": 0,
571                  "31": 0, "30": 0}

```

```

564 for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
565 test_resnet_coarse_gram_matrices, test_labels_list, resnet_coarse_predicted_labels):
566     encoding = str(test_label)
567     correct = ""
568     if test_label == pred_label:
569         encoding += "1"
570         correct = "correct"
571     else:
572         encoding += "0"
573         correct = "false"
574
575     if results_gotten[encoding] == 0:
576         # New type of result to save
577         results_gotten[encoding] += 1
578
579     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
580     gram_matrix = gram_matrix.reshape(32, 32)
581
582     # Save the resize testing image
583     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
584 testing/" + image_name
585     img = cv2.imread(image_path)
586     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
587     cv2.imwrite(resnet_coarse_path+image_name, img_resized)
588
589     # Save the gram matrix to display for results section of the report
590     plt.figure(figsize=(8,6))
591
592     # Use seaborn to create a heatmap
593     sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
594     plt.tight_layout()
595
596     # Save the heatmap to a file
597     plt.savefig(resnet_coarse_path+image_name[:-4] + "_gram_mat_" + correct + ".png"
598 , format='png', dpi=300, bbox_inches="tight")
599     plt.close()
600
601 # %% [markdown]
602 # # Resnet Fine Results:
603
604 # %%
605 # VGG SVM:
606 svm = MySVM()
607 svm.fit(resnet_fine_gram_matrices, labels_list)
608 resnet_fine_predicted_labels = svm.predict(test_resnet_fine_gram_matrices)
609 resnet_fine_accuracy, resnet_fine_class_report = svm.score(resnet_fine_predicted_labels,
610 test_labels_list)
611 print("Accuracy: ", resnet_fine_accuracy)
612 print(resnet_fine_class_report)
613
614 # %%
615 resnet_fine_confusion_mat = confusion_matrix(test_labels_list,
616 resnet_fine_predicted_labels)
617
618 plt.figure(figsize=(8, 6))
619 sns.heatmap(resnet_fine_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
620 plt.xlabel('Predicted Labels')
621 plt.ylabel('True Labels')
622 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
623 plt.show()
624
625 # %%
626 resnet_fine_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/
627 Resnet_Fine_Results/"
628
629 # I only want to save 1 positive match example and 1 negative match example for each
630 class
631 # The class is therefore the first number, and the second number is for matching labels
632 or not
633 results_gotten = {"01": 0, "00": 0,
634 "11": 0, "10": 0,
635 "21": 0, "20": 0,
636 "31": 0, "30": 0}

```

```

628 for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
629 test_resnet_fine_gram_matrices, test_labels_list, resnet_fine_predicted_labels):
630     encoding = str(test_label)
631     correct = ""
632     if test_label == pred_label:
633         encoding += "1"
634         correct = "correct"
635     else:
636         encoding += "0"
637         correct = "false"
638
639     if results_gotten[encoding] == 0:
640         # New type of result to save
641         results_gotten[encoding] += 1
642
643     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
644     gram_matrix = gram_matrix.reshape(32, 32)
645
646     # Save the resize testing image
647     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
648 testing/" + image_name
649     img = cv2.imread(image_path)
650     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
651     cv2.imwrite(resnet_fine_path+image_name, img_resized)
652
653     # Save the gram matrix to display for results section of the report
654     plt.figure(figsize=(8,6))
655
656     # Use seaborn to create a heatmap
657     sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
658     plt.tight_layout()
659
660     # Save the heatmap to a file
661     plt.savefig(resnet_fine_path+image_name[:-4] + "_gram_mat_" + correct + ".png",
662 format='png', dpi=300, bbox_inches="tight")
663     plt.close()
664
665 # %% [markdown]
666 # # Bonus: Channel Normalization Parameter Based Texture Descriptor
667
668 # %%
669 def get_normalization_params(feature_mat_list):
670     f_mats = np.array(feature_mat_list)
671
672     means = f_mats.mean(axis=(2, 3))
673     variances = f_mats.std(axis=(2, 3))
674
675     # I first stack the arrays together, and then reshape the final matrix to interleave
676     # the means and variances
677     mu_sigma_stacked = np.stack((means, variances), axis=-1)
678     channel_norm_params = mu_sigma_stacked.reshape(f_mats.shape[0], 2*f_mats.shape[1])
679
680     return channel_norm_params
681
682 # %%
683 vgg_norm_params = get_normalization_params(vgg_feature_list)
684 resnet_coarse_norm_params = get_normalization_params(resnet_coarse_feature_list)
685 resnet_fine_norm_params = get_normalization_params(resnet_fine_feature_list)
686 test_vgg_norm_params = get_normalization_params(test_vgg_feature_list)
687 test_resnet_coarse_norm_params = get_normalization_params(
688     test_resnet_coarse_feature_list)
689 test_resnet_fine_norm_params = get_normalization_params(test_resnet_fine_feature_list)
690
691 # %%
692 vgg_norm_params.shape
693
694 # %% [markdown]
695 # # Channel Norm Params VGG
696
697 # %%
698 # VGG SVM:
699 svm = MySVM()
700 svm.fit(vgg_norm_params, labels_list)
701 vgg_norm_predicted_labels = svm.predict(test_vgg_norm_params)

```

```

696 vgg_norm_accuracy, vgg_norm_class_report = svm.score(vgg_norm_predicted_labels,
697 test_labels_list)
698 print("Accuracy: ", vgg_norm_accuracy)
699 print(vgg_norm_class_report)
700
701 # %%
702 vgg_norm_confusion_mat = confusion_matrix(test_labels_list, vgg_norm_predicted_labels)
703
704 plt.figure(figsize=(8, 6))
705 sns.heatmap(vgg_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
706 plt.xlabel('Predicted Labels')
707 plt.ylabel('True Labels')
708 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
709 plt.show()
710
711 # %%
712 vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/VGG_Bonus_Results/"
713 # I only want to save 1 positive match example and 1 negative match example for each
714 class
715 # The class is therefore the first number, and the second number is for matching labels
716 or not
717 results_gotten = {"correct": 0, "false": 0}
718
719 for image_name, norm_params, test_label, pred_label in zip(test_progress_bar,
720 test_vgg_norm_params, test_labels_list, vgg_norm_predicted_labels):
721     correct = ""
722     if test_label == pred_label:
723         encoding = "correct"
724     else:
725         encoding = "false"
726
727     if results_gotten[encoding] == 0:
728         # New type of result to save
729         results_gotten[encoding] += 1
730
731     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
732     norm_params = norm_params.reshape(32, 32)
733
734     # Save the resize testing image
735     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
736 testing/" + image_name
737 img = cv2.imread(image_path)
738 img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
739 cv2.imwrite(vgg_path+image_name, img_resized)
740
741     # Save the gram matrix to display for results section of the report
742     plt.figure(figsize=(8,6))
743
744     # Use seaborn to create a heatmap
745     sns.heatmap(norm_params, cmap="viridis", cbar=True)
746     plt.tight_layout()
747     # Save the heatmap to a file
748     plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format=
749 'png', dpi=300, bbox_inches="tight")
750     plt.close()
751
752 # %% [markdown]
753 # # Resnet Coarse Results
754
755 # %%
756 # VGG SVM:
757 svm = MySVM()
758 svm.fit(resnet_coarse_norm_params, labels_list)
759 resnet_coarse_norm_predicted_labels = svm.predict(test_resnet_coarse_norm_params)
760 resnet_coarse_norm_accuracy, resnet_coarse_norm_class_report = svm.score(
761     resnet_coarse_norm_predicted_labels, test_labels_list)
762 print("Accuracy: ", resnet_coarse_norm_accuracy)
763 print(resnet_coarse_norm_class_report)
764
765 # %%
766 resnet_coarse_norm_confusion_mat = confusion_matrix(test_labels_list,
767 resnet_coarse_norm_predicted_labels)

```

```

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 # I only want to save 1 positive match example and 1 negative match example for each
    class
831 # The class is therefore the first number, and the second number is for matching labels
    or not
832 results_gotten = {"correct": 0, "false": 0}
833
834 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):
835     correct = ""
836     if test_label == pred_label:
837         encoding = "correct"
838     else:
839         encoding = "false"
840
841     if results_gotten[encoding] == 0:
842         # New type of result to save
843         results_gotten[encoding] += 1
844
845     # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
846     norm_params = norm_params.reshape(32, 32)
847
848     # Save the resize testing image
849     image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
testing/" + image_name
850     img = cv2.imread(image_path)
851     img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
852     cv2.imwrite(vgg_path+image_name, img_resized)
853
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()

```