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 caculate 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 dimesion 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 netowrk.

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

$$M = max(R, G, B)$$
$$m = min(R, G, B)$$
$$c = M - m$$
$$V = M$$
$$H = \begin{cases} 60\left(\frac{G-B}{c} mod 6\right) & M == R, c \neq 0 \\ 60\left(\frac{B-R}{c} + 2\right) & M == G, c \neq 0 \\ 60\left(\frac{R-G}{c} + 4\right) & M == B, c \neq 0 \\ 0 & c == 0 \end{cases}$$
$$S = \begin{cases} \frac{c}{V} * 255 & V \neq 0 \\ 0 & V == 0 \end{cases}$$

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

```
def img_BGR_to_HSV(img):
    img = img.astype(np.float32)
    img_hsv = np.zeros_like(img)

    # Calculate key parameters through the channel axis
    M = np.max(img, axis=2)
    m = np.min(img, axis=2)
    c = M - m
    V = M

    # For the rows, if the max is in the first column, etc
    h0_mask = (M == img[:, :, 2]) & (c != 0) # M == R, c=/=0
    h1_mask = (M == img[:, :, 1]) & (c != 0) # M == G, c=/=0
    h2_mask = (M == img[:, :, 0]) & (c != 0) # M == B, c=/=0
    c_mask = (c == 0)                        # c == 0

    # Calculate H Values for each row
```

```
    # We don't just want to use the mask since c can be zero for greyscale. So we want
    to only compute on the masks, by checking for where to input values in first.
    with np.errstate(divide='ignore', invalid='ignore'):
        img_hsv[:, :, 0] = np.where(h0_mask, (60 * (((img[:, :, 1] - img[:, :, 0]) / c)
    % 6)), img_hsv[:, :, 0])
        img_hsv[:, :, 0] = np.where(h1_mask, (60 * ((img[:, :, 0] - img[:, :, 2]) / c +
    2)), img_hsv[:, :, 0])
        img_hsv[:, :, 0] = np.where(h2_mask, (60 * ((img[:, :, 2] - img[:, :, 1]) / c +
    4)), img_hsv[:, :, 0])
        img_hsv[:, :, 0][c_mask] = 0 # No divide by 0 errors are possible here
    # To follow opencv formatting, I will rescale the hue angles to 180deg instead of
    360
    img_hsv[:, :, 0] /= 2

    # Fill in with correct values for the S column: (c/V)
    img_hsv[:, :, 1][V != 0] = c[V != 0]/V[V != 0] * 255

    # Fill in V col
    img_hsv[:,:,2] = V

    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):

$$\begin{aligned} \mathrm{p[1]} = {} & \mathrm{center\_value} \cdot (1 - 0.707) \cdot (1 - 0.707) + \\ & \mathrm{img\_h\_pad}[y][x+1] \cdot (1 - 0.707) \cdot 0.707 + \\ & \mathrm{img\_h\_pad}[y+1][x] \cdot 0.707 \cdot (1 - 0.707) + \\ & \mathrm{img\_h\_pad}[y+1][x+1] \cdot 0.707 \cdot 0.707 \end{aligned}$$

- 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:

```
class LBP():
    def __init__(self, R, P):
        self.R = R
        self.P = P
    def run_lbp(self, img_path):
        # Read image and convert it to HSV, then use the H channel for all downstream tasks.
        img_bgr = cv2.imread(img_path)
        img_hsv = img_BGR_to_HSV(img_bgr)
        img_h = img_hsv[:, :, 0]

        # Create padded image of size (64,64) for more feasilbe computation
        img_h_sized = cv2.resize(img_h, (62,62), interpolation=cv2.INTER_AREA)
        img_h_pad = np.pad(img_h_sized, pad_width=1, mode="constant", constant_values=0)

        # Initialize the histogram vector for the image: (We allow a max index of P + 1 0->9 in this case)
        lbp_histogram = np.zeros(self.P + 2)

        # Loop through all possible LBP centers:
        for y in range(self.R, img_h_pad.shape[0]-self.R):
            for x in range(self.R, img_h_pad.shape[1]-self.R):
                center_value = img_h_pad[y, x] # Scalar due to greyscale
                p = np.zeros(8)

                # Check the cardinal direction points (up,down,left,right)
                if img_h_pad[y+1][x] > center_value:
                    p[0] = 1
                if img_h_pad[y][x+1] > center_value:
                    p[2] = 1
                if img_h_pad[y-1][x] > center_value:
                    p[4] = 1
                if img_h_pad[y][x-1] > center_value:
                    p[6] = 1

                # We also have to check the diagonals.
                # To calculate the pixel values at these diagonal points, we need to do pixel-interpolation
                # We also apply thresholding on the interpolated points compared to the center to determine 0/1.
                # Top right point
                p[1] = center_value * (1 - 0.707) * (1 - 0.707) + \
                        img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
                        img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
                        img_h_pad[y+1][x+1] * 0.707 * 0.707
                p[1] = 1 if p[1] > center_value else 0

                # Bottom right point
                p[3] = center_value * (1 - 0.707) * (1 - 0.707) + \
                        img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
                        img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
                        img_h_pad[y-1][x+1] * 0.707 * 0.707
                p[3] = 1 if p[3] > center_value else 0

                # Bottom left point
                p[5] = center_value * (1 - 0.707) * (1 - 0.707) + \
                        img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
                        img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
                        img_h_pad[y-1][x-1] * 0.707 * 0.707
                p[5] = 1 if p[5] > center_value else 0

                # Top left point
                p[7] = center_value * (1 - 0.707) * (1 - 0.707) + \
                        img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
```

```
                        img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
                        img_h_pad[y+1][x-1] * 0.707 * 0.707
                p[7] = 1 if p[7] > center_value else 0

                # Now that we have out bitvector representation for the circle of points around the center
                # We want to find the unique min bitvector to represent the value at that point
                # We do this through circular bit-shifts to find the minimal representation:
                # This method is from Avi Kak's implementation in lecture 16
                bv = BitVector(bitlist=p)
                min_val = min([int(bv<<1) for _ in p])
                min_bv = BitVector(intVal=min_val, size=len(p))

                # Lastly, we use this min-bv value to get the final encoding for that point
                # So we create a min-int-val based integer representation of the binary pattern
                # From Avi's Notes:
                # - If the minIntVal representation involves more than two runs, encode it by the integer P + 1
                # - Else, if the minIntVal representation consists of all 0's, represent it be the encoding 0.
                # - Else, if the minIntVal representation consists of all 1's, represent it by the encoding P.
                # - Else: the minIntVal representation of a binary pattern has exactly two runs, that is,
                #          a run of 0s followed by a run of 1s, represent the pattern by the number of 1's in the second run
                num_runs = len(min_bv.runs())

                encoding = None
                # Mix of 1s and 0s
                if num_runs > 2:
                    encoding = self.P + 1
                # All 0s (8 of them)
                elif min_bv.int_val() == 0 and num_runs == 1:
                    encoding = self.P
                # 8 1s
                elif min_bv.int_val() == 255 and num_runs == 1:
                    encoding = self.P
                # Number of 1s in the second pattern if it is a run of all 0s then 1s
                else:
                    encoding = len(min_bv.runs()[1])
                lbp_histogram[encoding] += 1
        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 conver 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 tranposing 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:

```python
def get_gram_matrix(feature_mat_list):
    f_mats = np.array(feature_mat_list)
    N, C, H, W = f_mats.shape
    fmats_flat = f_mats.reshape(N, C, H*W)

    # A Gram matrix is the feature_map * feature_map.T
    gram_matrix = fmats_flat @ fmats_flat.transpose(0, 2, 1)

    # Conver the numpy array to a pytorch tensor for biliinear interpolation in downsampling
    # I also unsqueeze in the first dimension so that pytorch treats the final two dimensions as H,W and downsamples on those
    # Otherwise, would read the it as Batch, Channel, Height and a missing width
    gram_mat_tensor = torch.from_numpy(gram_matrix).unsqueeze(0)

    # Lastly, we want to resize the gram matrix from 512x512 to (32,32) for easier computation
    # We do this using bilinear interpolation
    downsampled_matrix = F.interpolate(gram_mat_tensor, size=(32, 32), mode='bilinear', align_corners=False)

    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:

```python
def get_normalization_params(feature_mat_list):
    f_mats = np.array(feature_mat_list)

    means = f_mats.mean(axis=(2, 3))
    variances = f_mats.std(axis=(2, 3))

    # I first stack the arrays together, and then reshape the final matrix to interleave the means and variances
    mu_sigma_stacked = np.stack((means, variances), axis=-1)
    channel_norm_params = mu_sigma_stacked.reshape(f_mats.shape[0], 2*f_mats.shape[1])

    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.

../Results/LBP_Hists/cloudy277.jpg ../Results/LBP_Hists/cloudy277_lbp_hist_correct.png

Correct image classification and LBP histogram

../Results/LBP_Hists/cloudy284.jpg ../Results/LBP_Hists/cloudy284_lbp_his

Incorrect image classification and LBP histogram

Cloudy: Image classification and histogram pairs

../Results/LBP_Hists/rain210.jpg ../Results/LBP_Hists/rain210_lbp_hist_correct.png

Correct image classification and LBP histogram

../Results/LBP_Hists/rain189.jpg ../Results/LBP_Hists/rain189_lbp_hist_

Incorrect image classification and LBP histogram

Rain: Image classification and histogram pairs

../Results/LBP_Hists/shine229.jpg ../Results/LBP_Hists/shine229_lbp_hist_correct.png

Correct image classification and LBP histogram

../Results/LBP_Hists/shine215.jpg ../Results/LBP_Hists/shine215_lbp_hist

Incorrect image classification and LBP histogram

Sunshine: Image classification and histogram pairs

../Results/LBP_Hists/sunrise348.jpg ../Results/LBP_Hists/sunrise348_lbp_hist_correct.png

Correct image classification and LBP histogram

../Results/LBP_Hists/sunrise354.jpg ../Results/LBP_Hists/sunrise354_lbp_hi

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:

`../Results/Confusion_Mats/lbp_conf_matr.png`

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.

../Results/VGG_Results/cloudy277.jpg ../Results/VGG_Results/cloudy277_gram_mat_correct.png

../Results/VGG_Results/cloudy300.jpg ../Results/VGG_Results/cloudy300_gram_

Correct image classification and Gram Matrix

Incorrect image classification and Gram Matrix

Cloudy: Image classification and Gram Matrix pairs

../Results/VGG_Results/rain187.jpg ../Results/VGG_Results/rain187_gram_mat_correct.png

../Results/VGG_Results/rain189.jpg ../Results/VGG_Results/rain189_gram_m

Correct image classification and Gram Matrix

Incorrect image classification and Gram Matrix

Rain: Image classification and Gram Matrix pairs

../Results/VGG_Results/shine215.jpg ../Results/VGG_Results/shine215_gram_mat_correct.png

../Results/VGG_Results/shine249.jpg ../Results/VGG_Results/shine249_gram_m

Correct image classification and Gram Matrix

Incorrect image classification and Gram Matrix

Sunshine: Image classification and Gram Matrix pairs

../Results/VGG_Results/sunrise348.jpg ../Results/VGG_Results/sunrise348_gram_mat_correct.png

../Results/VGG_Results/sunrise326.jpg ../Results/VGG_Results/sunrise326_gram

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

`../Results/Confusion_Mats/vgg_conf_mat.png`

6.3.2 Resnet50-Coarse Results:

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

../Results/Resnet_Coarse_Results/cloudy277_gram_mat_correct.png

../Results/Resnet_Coarse_Results/cloudy277.jpg

../Results/Resnet_Coarse_Results/clou

../Results/Resnet_Coarse_Results/cloudy279.jpg

Correct image classification and Gram Matrix

Incorrect image classification and Gram Matrix

Cloudy: Image classification and Gram Matrix pairs

../Results/Resnet_Coarse_Results/rain210_gram_mat_correct.png

../Results/Resnet_Coarse_Results/rain210.jpg

../Results/Resnet_Coarse_Results/rain

../Results/Resnet_Coarse_Results/rain189.jpg

Correct image classification and Gram Matrix

Incorrect image classification and Gram Matrix

Rain: Image classification and Gram Matrix pairs

../Results/Resnet_Coarse_Results/shine215_gram_mat_correct.png

../Results/Resnet_Coarse_Results/shine215.jpg

../Results/Resnet_Coarse_Results/shine

../Results/Resnet_Coarse_Results/shine225.jpg

Correct image classification and Gram Matrix

Incorrect image classification and Gram Matrix

Sunshine: Image classification and Gram Matrix pairs

../Results/Resnet_Coarse_Results/sunrise346_gram_mat_correct.png

../Results/Resnet_Coarse_Results/sunrise346.jpg

../Results/Resnet_Coarse_Results/sunri

../Results/Resnet_Coarse_Results/sunrise348.jpg

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

../Results/Confusion_Mats/resnet_coarse_conf_mat.png

6.3.3 Resnet50-Fine Results:

../Results/Resnet_Fine_Results/cloudy272_gram_matrix_correct.png ../Results/Resnet_Fine_Results/cloudy2
../Results/Resnet_Fine_Results/cloudy272.jpg ../Results/Resnet_Fine_Results/cloudy277.jpg

Correct image classification and Gram Matrix Incorrect image classification and Gram Matrix

Cloudy: Image classification and Gram Matrix pairs

../Results/Resnet_Fine_Results/rain187_gram_matrix_correct.png ../Results/Resnet_Fine_Results/rain18
../Results/Resnet_Fine_Results/rain187.jpg ../Results/Resnet_Fine_Results/rain189.jpg

Correct image classification and Gram Matrix Incorrect image classification and Gram Matrix

Rain: Image classification and Gram Matrix pairs

../Results/Resnet_Fine_Results/shine215_gram_matrix_correct.png ../Results/Resnet_Fine_Results/shine2
../Results/Resnet_Fine_Results/shine215.jpg ../Results/Resnet_Fine_Results/shine206.jpg

Correct image classification and Gram Matrix Incorrect image classification and Gram Matrix

Sunshine: Image classification and Gram Matrix pairs

../Results/Resnet_Fine_Results/sunrise348_gram_matrix_correct.png ../Results/Resnet_Coarse_Results/sunri
../Results/Resnet_Fine_Results/sunrise348.jpg ../Results/Resnet_Coarse_Results/sunrise348.jpg

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:

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:

`../Results/Confusion_Mats/resnet_fine_conf_mat.png`

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:

../Results/VGG_Bonus_Results/rain189.jpg
../Results/VGG_Bonus_Results/rain189_gram_mat_correct.png

Correct image classification and channel normalization Parameters in matrix form

../Results/VGG_Bonus_Results/shine212.jpg
../Results/VGG_Bonus_Results/shine212_

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

../Results/Confusion_Mats/vgg_cm_norm.png

6.5.2 Resnet Coarse Bonus Results:

../Results/Resnet_Coarse_Bonus_Results/rain189_gradcam_correct.png

../Results/Resnet_Coarse_Bonus_Results/rain189.jpg

Correct image classification and channel normalization Parameters in matrix form

../Results/Resnet_Coarse_Bonus_Results/shine225.jpg

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

../Results/Confusion_Mats/resnet_coarse_cm_norm.png

6.5.3 Resnet Fine Bonus Results:

../Results/Resnet_Fine_Bonus_Results/rain187.jpg

../Results/Resnet_Fine_Bonus_Results/rain187_gram_mat.png

Correct image classification and channel normalization Parameters in matrix form

../Results/Resnet_Fine_Bonus_Results/rain189.jpg

../Results/Resnet_Fine_Bonus_Results/r

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

../Results/Confusion_Mats/resnet_fine_cm_norm.png

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.

```
# %%
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
from tqdm import tqdm
import pandas as pd
from BitVector import BitVector
import os
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import re
import pickle
from vgg_and_resnet import *
import torch.nn.functional as F

# %%
def img_BGR_to_HSV(img):
    img = img.astype(np.float32)
    img_hsv = np.zeros_like(img)

    # Calculate key parameters through the channel axis
    M = np.max(img, axis=2)
    m = np.min(img, axis=2)
    c = M - m
    V = M

    # For the rows, if the max is in the first column, etc
    h0_mask = (M == img[:, :, 2]) & (c != 0) # M == R, c=/=0
    h1_mask = (M == img[:, :, 1]) & (c != 0) # M == G, c=/=0
    h2_mask = (M == img[:, :, 0]) & (c != 0) # M == B, c=/=0
    c_mask = (c == 0)                         # c == 0

    # Calculate H Values for each row
    # We don't just want to use the mask since c can be zero for greyscale. So we want to only compute on the masks, by checking for where to input values in first.
    with np.errstate(divide='ignore', invalid='ignore'):
        img_hsv[:, :, 0] = np.where(h0_mask, (60 * (((img[:, :, 1] - img[:, :, 0]) / c) % 6)), img_hsv[:, :, 0])
        img_hsv[:, :, 0] = np.where(h1_mask, (60 * ((img[:, :, 0] - img[:, :, 2]) / c + 2)), img_hsv[:, :, 0])
        img_hsv[:, :, 0] = np.where(h2_mask, (60 * ((img[:, :, 2] - img[:, :, 1]) / c + 4)), img_hsv[:, :, 0])
        img_hsv[:, :, 0][c_mask] = 0 # No divide by 0 errors are possible here
    # To follow opencv formatting, I will rescale the hue angles to 180deg instead of 360
    img_hsv[:, :, 0] /= 2

    # Fill in with correct values for the S column: (c/V)
    img_hsv[:, :, 1][V != 0] = c[V != 0]/V[V != 0] * 255

    # Fill in V col
    img_hsv[:,:,2] = V

    return np.ceil(img_hsv).astype(np.uint8)

# %%
class LBP():
    def __init__(self, R, P):
        self.R = R
        self.P = P
    def run_lbp(self, img_path):
        # Read image and convert it to HSV, then use the H channel for all downstream tasks.
        img_bgr = cv2.imread(img_path)
        img_hsv = img_BGR_to_HSV(img_bgr)
        img_h = img_hsv[:, :, 0]

        # Create padded image of size (64,64) for more feasilbe computation
        img_h_sized = cv2.resize(img_h, (62,62), interpolation=cv2.INTER_AREA)
        img_h_pad = np.pad(img_h_sized, pad_width=1, mode="constant", constant_values=0)

        # Initialize the histogram vector for the image: (We allow a max index of P + 1 0->9 in this case)
        lbp_histogram = np.zeros(self.P + 2)

        # Loop through all possible LBP centers:
        for y in range(self.R, img_h_pad.shape[0]-self.R):
```

```
        for x in range(self.R, img_h_pad.shape[1]-self.R):
            center_value = img_h_pad[y, x] # Scalar due to greyscale
            p = np.zeros(8)

            # Check the cardinal direction points (up,down,left,right)
            if img_h_pad[y+1][x] > center_value:
                p[0] = 1
            if img_h_pad[y][x+1] > center_value:
                p[2] = 1
            if img_h_pad[y-1][x] > center_value:
                p[4] = 1
            if img_h_pad[y][x-1] > center_value:
                p[6] = 1

            # We also have to check the diagonals.
            # To calculate the pixel values at these diagonal points, we need to do pixel-interpolation
            # We also apply thresholding on the interpolated points compared to the center to determine 0/1.
            # Top right point
            p[1] = center_value * (1 - 0.707) * (1 - 0.707) + \
                    img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
                    img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
                    img_h_pad[y+1][x+1] * 0.707 * 0.707
            p[1] = 1 if p[1] > center_value else 0

            # Bottom right point
            p[3] = center_value * (1 - 0.707) * (1 - 0.707) + \
                    img_h_pad[y][x+1] * (1 - 0.707) * 0.707 + \
                    img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
                    img_h_pad[y-1][x+1] * 0.707 * 0.707
            p[3] = 1 if p[3] > center_value else 0

            # Bottom left point
            p[5] = center_value * (1 - 0.707) * (1 - 0.707) + \
                    img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
                    img_h_pad[y-1][x] * 0.707 * (1 - 0.707) + \
                    img_h_pad[y-1][x-1] * 0.707 * 0.707
            p[5] = 1 if p[5] > center_value else 0

            # Top left point
            p[7] = center_value * (1 - 0.707) * (1 - 0.707) + \
                    img_h_pad[y][x-1] * (1 - 0.707) * 0.707 + \
                    img_h_pad[y+1][x] * 0.707 * (1 - 0.707) + \
                    img_h_pad[y+1][x-1] * 0.707 * 0.707
            p[7] = 1 if p[7] > center_value else 0

            # Now that we have out bitvector representation for the circle of points around the center
            # We want to find the unique min bitvector to represent the value at that point
            # We do this through circular bit-shifts to find the minimal representation:
            # This method is from Avi Kak's implementation in lecture 16
            bv = BitVector(bitlist=p)
            min_val = min([int(bv<<1) for _ in p])
            min_bv = BitVector(intVal=min_val, size=len(p))

            # Lastly, we use this min-bv value to get the final encoding for that point
            # So we create a min-int-val based integer representation of the binary pattern
            # From Avi's Notes:
            # - If the minIntVal representation involves more than two runs, encode it by the integer P + 1
            # - Else, if the minIntVal representation consists of all 0's, represent it be the encoding 0.
            # - Else, if the minIntVal representation consists of all 1's, represent it by the encoding P.
            # - Else: the minIntVal representation of a binary pattern has exactly two runs, that is,
            #          a run of 0s followed by a run of 1s, represent the pattern by the number of 1's in the second run
```

```python
                num_runs = len(min_bv.runs())

                encoding = None
                # Mix of 1s and 0s
                if num_runs > 2:
                    encoding = self.P + 1
                # All 0s (8 of them)
                elif min_bv.int_val() == 0 and num_runs == 1:
                    encoding = self.P
                # 8 1s
                elif min_bv.int_val() == 255 and num_runs == 1:
                    encoding = self.P
                # Number of 1s in the second pattern if it is a run of all 0s then 1s
                else:
                    encoding = len(min_bv.runs()[1])
                lbp_histogram[encoding] += 1
        return lbp_histogram

# %%
class MySVM():
    def __init__(self):
        self.classifier = SVC(decision_function_shape="ovr")

    def fit(self, features, labels):
        # Train the classifier on the train data/labels
        self.classifier.fit(features, labels)

    def predict(self, features):
        # Predict the labels for the tes data
        return self.classifier.predict(features)

    def fit_predict(self, features, labels):
        # Fit and predict on the same data
        self.classifier.fit(features, labels)
        return self.classifier.predict(features)

    def score(self, predicted_labels, true_labels):
        # Returns the mean accuracy using the test data and labels.
        return accuracy_score(true_labels, predicted_labels), classification_report(
    true_labels, predicted_labels)

# %%
R = 1
P = 8
image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    training/")
lbp_hist_list = []
labels_list = []
progress_bar = tqdm(image_list, desc="Training Loop")
image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}

for image_name in progress_bar:
    try:
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    training/" + image_name
        image_type = re.split(r"([0-9]+)", image_name)[0]
        label = image_type_to_label[image_type]

        lbp_hist = LBP(R=R, P=P).run_lbp(img_path=image_path)
        lbp_hist_list.append(lbp_hist)

        # Fill in with image name -> index for training
        labels_list.append(label)
    except Exception as e:
        print("This image did not work: ", image_name)


# %%
svm = MySVM()
svm.fit(lbp_hist_list, labels_list)

# %%
result_dict = {"lbp_hist_list": lbp_hist_list, "labels_list": labels_list}
```

```python
with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/lbp_hists.pkl", "wb") as file
    :
    pickle.dump(result_dict, file)

# %%
test_image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/
    data/testing/")
test_lbp_hist_list = []
test_labels_list = []
image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
test_progress_bar = tqdm(test_image_list, desc="Testing Loop")

for image_name in test_progress_bar:
    try:
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    testing/" + image_name
        image_type = re.split(r"([0-9]+)", image_name)[0]
        label = image_type_to_label[image_type]

        lbp_hist = LBP(R=R, P=P).run_lbp(img_path=image_path)
        test_lbp_hist_list.append(lbp_hist)

        # Add in labels based on image name
        test_labels_list.append(label)
    except Exception as e:
        print("This image did not work: ", image_name)

# %%
test_result_dict = {"test_lbp_hist_list": test_lbp_hist_list, "test_labels_list":
    test_labels_list}
with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/test_lbp_hists.pkl", "wb") as
     file:
    pickle.dump(test_result_dict, file)

# %%
predicted_labels = svm.predict(test_lbp_hist_list)

# %%
accuracy, class_report = svm.score(predicted_labels, test_labels_list)

# %%
confusion_mat = confusion_matrix(test_labels_list, predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %% [markdown]
# # Get results for LBP histograms & images success/failure

# %%
lbp_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/LBP_Results/"
lbp_hist_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/LBP_Hists/"
# I only want to save 1 positive match example and 1 negative match example for each
    class
# The class is therefore the first number, and the second number is for matching labels
    or not
results_gotten = {"01": 0, "00": 0,
                  "11": 0, "10": 0,
                  "21": 0, "20": 0,
                  "31": 0, "30": 0}

for image_name, test_lbp_hist, test_label, pred_label in zip(test_progress_bar,
    test_lbp_hist_list, test_labels_list, predicted_labels):
    encoding = str(test_label)
    correct = ""
    if test_label == pred_label:
        encoding += "1"
        correct = "correct"
    else:
```

```python
        encoding += "0"
        correct = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Save the resize testing image
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(lbp_hist_path+image_name, img_resized)

        # Save the histogram plot
        plt.figure(figsize=(8,6))
        plt.bar(range(len(test_lbp_hist)), test_lbp_hist, color='blue')  # Customize color as needed
        plt.tight_layout()
        # Save the plot to a file
        plt.savefig(lbp_hist_path+image_name[:-4] + "_lbp_hist_" + correct + ".png", format='png', dpi=300)
        plt.close()

# %% [markdown]
# # Feature Map Extraction

# %%
# We run this once, and save all of the feature maps for all of the images to save computation time during debugging
class FeatureMapper():
    def __init__(self):
        pass
    def get_resized_img_input(self, img_path):
        img = cv2.imread(img_path)
        # Convert images to RGB due to how RESNET and VGG expect inputs
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Create padded image of size (256,256) for more feasilbe computation
        img = cv2.resize(img, (256,256), interpolation=cv2.INTER_AREA)
        return img

    def get_feature_map_vgg(self, img_path):
        img = self.get_resized_img_input(img_path)

        # The next three lines are from the tutorial included in the instructions
        vgg = VGG19()
        vgg.load_weights('vgg_normalized.pth')
        vgg_feature = vgg(img)
        return vgg_feature

    def get_feature_map_resnet(self, img_path):
        img = self.get_resized_img_input(img_path)

        # The next three lines are from the tutorial included in the instructions
        encoder_name='resnet50'
        resnet = CustomResNet(encoder=encoder_name)
        resnet_feat_coarse, resnet_feat_fine = resnet(img)
        return resnet_feat_coarse, resnet_feat_fine

# %%
image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/training/")
vgg_feature_list = []
resnet_coarse_feature_list = []
resnet_fine_feature_list = []
progress_bar = tqdm(image_list, desc="Training Loop")
image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
img_names = []
labels_list = []
featureMapper = FeatureMapper()

for image_name in progress_bar:
```

```
    try:
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    training/" + image_name
        image_type = re.split(r"([0-9]+)", image_name)[0]
        label = image_type_to_label[image_type]

        # Get VGG Feature Map
        vgg_feature = featureMapper.get_feature_map_vgg(img_path=image_path)
        vgg_feature_list.append(vgg_feature)

        # Resnet Features
        resnet_feat_coarse, resnet_feat_fine = featureMapper.get_feature_map_resnet(
    img_path=image_path)
        resnet_coarse_feature_list.append(resnet_feat_coarse)
        resnet_fine_feature_list.append(resnet_feat_fine)

        # Append the image name:
        img_names.append(image_name)

        # Fill in with image name -> index for training
        labels_list.append(label)
    except Exception as e:
        print("This image did not work: ", image_name)
        print(e)


# %%
result_dict = {"vgg_feature_list": vgg_feature_list,
                "resnet_coarse_feature_list": resnet_coarse_feature_list,
                "resnet_fine_feature_list": resnet_fine_feature_list,
                "img_names": img_names,
                "labels_list": labels_list}

# %%
with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/training_freature_mats.pkl",
    "wb") as file:
    pickle.dump(result_dict, file)

# %%
test_image_list = os.listdir("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/
    data/testing/")
test_vgg_feature_list = []
test_resnet_coarse_feature_list = []
test_resnet_fine_feature_list = []
test_img_names = []
test_labels_list = []
image_type_to_label = {"cloudy": 0, "rain": 1, "shine": 2, "sunrise": 3}
featureMapper = FeatureMapper()
test_progress_bar = tqdm(test_image_list, desc="Testing Loop")

for image_name in test_progress_bar:
    try:
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    testing/" + image_name
        image_type = re.split(r"([0-9]+)", image_name)[0]
        label = image_type_to_label[image_type]

        # Get VGG Feature Map
        test_vgg_feature = featureMapper.get_feature_map_vgg(img_path=image_path)
        test_vgg_feature_list.append(test_vgg_feature)

        # Resnet Features
        test_resnet_feat_coarse, test_resnet_feat_fine = featureMapper.
    get_feature_map_resnet(img_path=image_path)
        test_resnet_coarse_feature_list.append(test_resnet_feat_coarse)
        test_resnet_fine_feature_list.append(test_resnet_feat_fine)

        # Append the image name:
        test_img_names.append(image_name)

        # Fill in with image name -> index for training
        test_labels_list.append(label)
    except Exception as e:
```

```python
        print("This image did not work: ", image_name)
        print(e)


# %%
test_result_dict = {"test_vgg_feature_list": test_vgg_feature_list,
                "test_resnet_coarse_feature_list": test_resnet_coarse_feature_list,
                "test_resnet_fine_feature_list": test_resnet_fine_feature_list,
                "test_img_names": test_img_names,
                "test_labels_list": test_labels_list}

# %%
with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/testing_freature_mats.pkl", "wb") as file:
    pickle.dump(result_dict, file)

# %% [markdown]
# # Gram Matrix Calculation:

# %%
def get_gram_matrix(feature_mat_list):
    f_mats = np.array(feature_mat_list)
    N, C, H, W = f_mats.shape
    fmats_flat = f_mats.reshape(N, C, H*W)

    # A Gram matrix is the feature_map * feature_map.T
    gram_matrix = fmats_flat @ fmats_flat.transpose(0, 2, 1)

    # Conver the numpy array to a pytorch tensor for biliinear interpolation in downsampling
    # I also unsqueeze in the first dimension so that pytorch treats the final two dimensions as H,W and downsamples on those
    # Otherwise, would read the it as Batch, Channel, Height and a missing width
    gram_mat_tensor = torch.from_numpy(gram_matrix).unsqueeze(0)

    # Lastly, we want to resize the gram matrix from 512x512 to (32,32) for easier computation
    # We do this using bilinear interpolation
    downsampled_matrix = F.interpolate(gram_mat_tensor, size=(32, 32), mode='bilinear', align_corners=False)

    return downsampled_matrix.squeeze().numpy()

# %%
vgg_gram_matrices = get_gram_matrix(vgg_feature_list)
resnet_coarse_gram_matrices = get_gram_matrix(resnet_coarse_feature_list)
resnet_fine_gram_matrices = get_gram_matrix(resnet_fine_feature_list)
test_vgg_gram_matrices = get_gram_matrix(test_vgg_feature_list)
test_resnet_coarse_gram_matrices = get_gram_matrix(test_resnet_coarse_feature_list)
test_resnet_fine_gram_matrices = get_gram_matrix(test_resnet_fine_feature_list)

# %%
# Flattening the final dimseion is required since SVM can only take in as inputs 2 dims (Batch, features)
vgg_gram_matrices = vgg_gram_matrices.reshape(vgg_gram_matrices.shape[0], -1)
resnet_coarse_gram_matrices = resnet_coarse_gram_matrices.reshape(resnet_coarse_gram_matrices.shape[0], -1)
resnet_fine_gram_matrices = resnet_fine_gram_matrices.reshape(resnet_fine_gram_matrices.shape[0], -1)
test_vgg_gram_matrices = test_vgg_gram_matrices.reshape(test_vgg_gram_matrices.shape[0], -1)
test_resnet_coarse_gram_matrices = test_resnet_coarse_gram_matrices.reshape(test_resnet_coarse_gram_matrices.shape[0], -1)
test_resnet_fine_gram_matrices = test_resnet_fine_gram_matrices.reshape(test_resnet_fine_gram_matrices.shape[0], -1)

# %%
# Save gram matrices to a file:
gram_matrices = {"vgg_gram_matrices": vgg_gram_matrices,
"resnet_coarse_gram_matrices": resnet_coarse_gram_matrices,
"resnet_fine_gram_matrices": resnet_fine_gram_matrices,
"test_vgg_gram_matrices": test_vgg_gram_matrices,
"test_resnet_coarse_gram_matrices": test_resnet_coarse_gram_matrices,
```

```python
"test_resnet_fine_gram_matrices": test_resnet_fine_gram_matrices}
with open("/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Saves/all_gram_matrices.pkl", "wb") as file:
    pickle.dump(gram_matrices, file)

# %% [markdown]
# # VGG Final Results

# %%
# VGG SVM:
svm = MySVM()
svm.fit(vgg_gram_matrices, labels_list)
vgg_predicted_labels = svm.predict(test_vgg_gram_matrices)
vgg_accuracy, vgg_class_report = svm.score(vgg_predicted_labels, test_labels_list)
print("Accuracy: ", vgg_accuracy)
print(vgg_class_report)

# %%
vgg_confusion_mat = confusion_matrix(test_labels_list, vgg_predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(vgg_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %%
vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/VGG_Results/"
# I only want to save 1 positive match example and 1 negative match example for each class
# The class is therefore the first number, and the second number is for matching labels or not
results_gotten = {"01": 0, "00": 0,
                  "11": 0, "10": 0,
                  "21": 0, "20": 0,
                  "31": 0, "30": 0}

for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar, test_vgg_gram_matrices, test_labels_list, vgg_predicted_labels):
    encoding = str(test_label)
    correct = ""
    if test_label == pred_label:
        encoding += "1"
        correct = "correct"
    else:
        encoding += "0"
        correct = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
        gram_matrix = gram_matrix.reshape(32, 32)

        # Save the resize testing image
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(vgg_path+image_name, img_resized)

        # Save the gram matrix to display for results section of the report
        plt.figure(figsize=(8,6))

        # Use seaborn to create a heatmap
        sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
        plt.tight_layout()
        # Save the heatmap to a file
        plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + correct + ".png", format='png', dpi=300, bbox_inches="tight")
        plt.close()
```

```
# %% [markdown]
# # Resnet Coarse Results

# %%
# Resnet Coarse:
svm = MySVM()
svm.fit(resnet_coarse_gram_matrices, labels_list)
resnet_coarse_predicted_labels = svm.predict(test_resnet_coarse_gram_matrices)
resnet_coarse_accuracy, resnet_coarse_class_report = svm.score(
    resnet_coarse_predicted_labels, test_labels_list)
print("Accuracy: ", resnet_coarse_accuracy)
print(resnet_coarse_class_report)

# %%
resnet_coarse_confusion_mat = confusion_matrix(test_labels_list,
    resnet_coarse_predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(resnet_coarse_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %%
resnet_coarse_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/
    Resnet_Coarse_Results/"
# I only want to save 1 positive match example and 1 negative match example for each
    class
# The class is therefore the first number, and the second number is for matching labels
    or not
results_gotten = {"01": 0, "00": 0,
                  "11": 0, "10": 0,
                  "21": 0, "20": 0,
                  "31": 0, "30": 0}

for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar,
    test_resnet_coarse_gram_matrices, test_labels_list, resnet_coarse_predicted_labels):
    encoding = str(test_label)
    correct = ""
    if test_label == pred_label:
        encoding += "1"
        correct = "correct"
    else:
        encoding += "0"
        correct = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
        gram_matrix = gram_matrix.reshape(32, 32)

        # Save the resize testing image
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/
    testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(resnet_coarse_path+image_name, img_resized)

        # Save the gram matrix to display for results section of the report
        plt.figure(figsize=(8,6))

        # Use seaborn to create a heatmap
        sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
        plt.tight_layout()
        # Save the heatmap to a file
        plt.savefig(resnet_coarse_path+image_name[:-4] + "_gram_mat_" + correct + ".png"
    , format='png', dpi=300, bbox_inches="tight")
        plt.close()
```

```
# %% [markdown]
# # Resnet Fine Results:

# %%
# VGG SVM:
svm = MySVM()
svm.fit(resnet_fine_gram_matrices, labels_list)
resnet_fine_predicted_labels = svm.predict(test_resnet_fine_gram_matrices)
resnet_fine_accuracy, resnet_fine_class_report = svm.score(resnet_fine_predicted_labels, test_labels_list)
print("Accuracy: ", resnet_fine_accuracy)
print(resnet_fine_class_report)

# %%
resnet_fine_confusion_mat = confusion_matrix(test_labels_list, resnet_fine_predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(resnet_fine_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %%
resnet_fine_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/Resnet_Fine_Results/"
# I only want to save 1 positive match example and 1 negative match example for each class
# The class is therefore the first number, and the second number is for matching labels or not
results_gotten = {"01": 0, "00": 0,
                  "11": 0, "10": 0,
                  "21": 0, "20": 0,
                  "31": 0, "30": 0}

for image_name, gram_matrix, test_label, pred_label in zip(test_progress_bar, test_resnet_fine_gram_matrices, test_labels_list, resnet_fine_predicted_labels):
    encoding = str(test_label)
    correct = ""
    if test_label == pred_label:
        encoding += "1"
        correct = "correct"
    else:
        encoding += "0"
        correct = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
        gram_matrix = gram_matrix.reshape(32, 32)

        # Save the resize testing image
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(resnet_fine_path+image_name, img_resized)

        # Save the gram matrix to display for results section of the report
        plt.figure(figsize=(8,6))

        # Use seaborn to create a heatmap
        sns.heatmap(gram_matrix, cmap="viridis", cbar=True)
        plt.tight_layout()
        # Save the heatmap to a file
        plt.savefig(resnet_fine_path+image_name[:-4] + "_gram_mat_" + correct + ".png", format='png', dpi=300, bbox_inches="tight")
        plt.close()

# %% [markdown]
```

```python
# # Bonus: Channel Normalization Parameter Based Texture Descriptor

# %%
def get_normalization_params(feature_mat_list):
    f_mats = np.array(feature_mat_list)

    means = f_mats.mean(axis=(2, 3))
    variances = f_mats.std(axis=(2, 3))

    # I first stack the arrays together, and then reshape the final matrix to interleave the means and variances
    mu_sigma_stacked = np.stack((means, variances), axis=-1)
    channel_norm_params = mu_sigma_stacked.reshape(f_mats.shape[0], 2*f_mats.shape[1])

    return channel_norm_params

# %%
vgg_norm_params = get_normalization_params(vgg_feature_list)
resnet_coarse_norm_params = get_normalization_params(resnet_coarse_feature_list)
resnet_fine_norm_params = get_normalization_params(resnet_fine_feature_list)
test_vgg_norm_params = get_normalization_params(test_vgg_feature_list)
test_resnet_coarse_norm_params = get_normalization_params(test_resnet_coarse_feature_list)
test_resnet_fine_norm_params = get_normalization_params(test_resnet_fine_feature_list)

# %%
vgg_norm_params.shape

# %% [markdown]
# # Channel Norm Params VGG

# %%
# VGG SVM:
svm = MySVM()
svm.fit(vgg_norm_params, labels_list)
vgg_norm_predicted_labels = svm.predict(test_vgg_norm_params)
vgg_norm_accuracy, vgg_norm_class_report = svm.score(vgg_norm_predicted_labels, test_labels_list)
print("Accuracy: ", vgg_norm_accuracy)
print(vgg_norm_class_report)

# %%
vgg_norm_confusion_mat = confusion_matrix(test_labels_list, vgg_norm_predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(vgg_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %%
vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/VGG_Bonus_Results/"
# I only want to save 1 positive match example and 1 negative match example for each class
# The class is therefore the first number, and the second number is for matching labels or not
results_gotten = {"correct": 0, "false": 0}

for image_name, norm_params, test_label, pred_label in zip(test_progress_bar, test_vgg_norm_params, test_labels_list, vgg_norm_predicted_labels):
    correct = ""
    if test_label == pred_label:
        encoding = "correct"
    else:
        encoding = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
        norm_params = norm_params.reshape(32, 32)
```

```

        # Save the resize testing image
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(vgg_path+image_name, img_resized)

        # Save the gram matrix to display for results section of the report
        plt.figure(figsize=(8,6))

        # Use seaborn to create a heatmap
        sns.heatmap(norm_params, cmap="viridis", cbar=True)
        plt.tight_layout()
        # Save the heatmap to a file
        plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format='png', dpi=300, bbox_inches="tight")
        plt.close()

# %% [markdown]
# # Resnet Coarse Results

# %%
# VGG SVM:
svm = MySVM()
svm.fit(resnet_coarse_norm_params, labels_list)
resnet_coarse_norm_predicted_labels = svm.predict(test_resnet_coarse_norm_params)
resnet_coarse_norm_accuracy, resnet_coarse_norm_class_report = svm.score(resnet_coarse_norm_predicted_labels, test_labels_list)
print("Accuracy: ", resnet_coarse_norm_accuracy)
print(resnet_coarse_norm_class_report)

# %%
resnet_coarse_norm_confusion_mat = confusion_matrix(test_labels_list, resnet_coarse_norm_predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(resnet_coarse_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %%
vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/Resnet_Coarse_Bonus_Results/"
# I only want to save 1 positive match example and 1 negative match example for each class
# The class is therefore the first number, and the second number is for matching labels or not
results_gotten = {"correct": 0, "false": 0}

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):
    correct = ""
    if test_label == pred_label:
        encoding = "correct"
    else:
        encoding = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Convert the Norm Params back from (N,2048) -> (N, 32,32) for display
        # 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)
        norm_params = norm_params[::2] # Extract even indices
        norm_params = norm_params.reshape(32, 32)

        # Save the resize testing image
```

```
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(vgg_path+image_name, img_resized)

        # Save the gram matrix to display for results section of the report
        plt.figure(figsize=(8,6))

        # Use seaborn to create a heatmap
        sns.heatmap(norm_params, cmap="viridis", cbar=True)
        plt.tight_layout()
        # Save the heatmap to a file
        plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format='png', dpi=300, bbox_inches="tight")
        plt.close()

# %% [markdown]
# # Resent Fine Results:

# %%
# VGG SVM:
svm = MySVM()
svm.fit(resnet_fine_norm_params, labels_list)
resnet_fine_norm_predicted_labels = svm.predict(test_resnet_fine_norm_params)
resnet_fine_norm_accuracy, resnet_fine_norm_class_report = svm.score(resnet_fine_norm_predicted_labels, test_labels_list)
print("Accuracy: ", resnet_fine_norm_accuracy)
print(resnet_fine_norm_class_report)

# %%
resnet_fine_norm_confusion_mat = confusion_matrix(test_labels_list, resnet_fine_norm_predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(resnet_fine_norm_confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.show()

# %%
vgg_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/Results/Resnet_Fine_Bonus_Results/"
# I only want to save 1 positive match example and 1 negative match example for each class
# The class is therefore the first number, and the second number is for matching labels or not
results_gotten = {"correct": 0, "false": 0}

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):
    correct = ""
    if test_label == pred_label:
        encoding = "correct"
    else:
        encoding = "false"

    if results_gotten[encoding] == 0:
        # New type of result to save
        results_gotten[encoding] += 1

        # Convert the vgg_gram_matrix back from (N,1024) -> (N, 32,32) for display
        norm_params = norm_params.reshape(32, 32)

        # Save the resize testing image
        image_path = "/mnt/cloudNAS3/Adubois/Classes/ECE661/HW7/HW7-Auxilliary/data/testing/" + image_name
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (128,128), interpolation=cv2.INTER_AREA)
        cv2.imwrite(vgg_path+image_name, img_resized)

```

```
        # Save the gram matrix to display for results section of the report
        plt.figure(figsize=(8,6))

        # Use seaborn to create a heatmap
        sns.heatmap(norm_params, cmap="viridis", cbar=True)
        plt.tight_layout()
        # Save the heatmap to a file
        plt.savefig(vgg_path+image_name[:-4] + "_gram_mat_" + encoding + ".png", format=
    'png', dpi=300, bbox_inches="tight")
        plt.close()
```