DECEMBER 13, 2019

School of Information Studies
Syracuse University

INTEL Image Classification

Srinivas Reddy Pachika, Nahnsan Guseh, Ayush Kumar, Sucheta Kadavger

IST 718 Big Data Analytics
Prepared for Professor Willard Williamson

Completed on:
December 13th, 2019

INDEX

Abstract

The dataset we will be examining, analyzing and classifying is an image intel dataset. We found the dataset through Kaggle. In this dataset there is over 25,000 images with a listed dimension size of 150 x 150. Each of the images are currently listed in six different categories. Buildings, sea, forest, glaciers, mountains and street. Here is a link to the dataset.
https://www.kaggle.com/puneet6060/intel-image-classification

We plan to use multiple classification models to analyze which classification method will be the most effective and efficient at correctly classifying images. We will also take into account our computing speed in creating these models. The models we plan on using are SVM, CNN models. After examining these models we hope to analyze the two and understand which model works best and why. We want to use techniques that will work the best for us in successfully managing the model, but also creating a practical and equip model. Additionally, We have already began examine the ideal method of feature extraction for our project. We have looked into Scale Invariant Feature Transform(SIFT), Speeded-Up Robust Feature(SURF) and Histogram of Oriented Gradients(HOG). We are currently leaning towards using the HOG analysis for feature extraction as it seems to be the most equip for image classification. Compared to SIFT and SURF; HOG analysis examines the gradients, orientation and edge of an image, while SIFT and SURF methods only analyzes the edges of images.

From here we will compare the models, then examine and inquire upon how well the models classified images as well as what makes the particular model more or less efficient in image classification. We plan on using python, pyspark, keras for this project. However, if we find another successful technology that we feel can help us analyze and interpret this dataset and create an improved model, we will include it in the rest of our report.

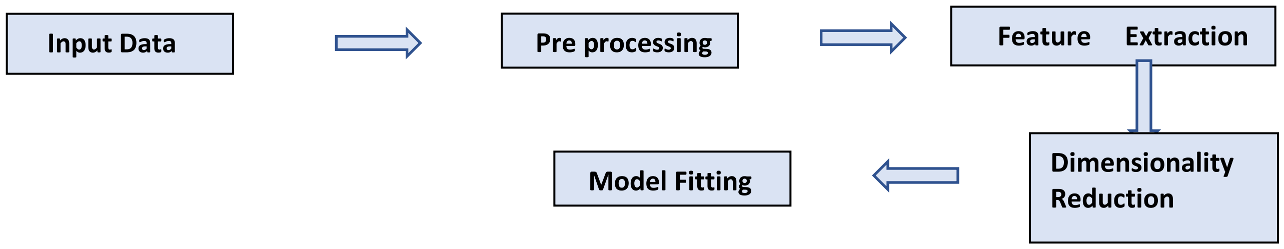
Some questions we will want to answer and inquire upon include:

- Which feature extraction method is best suited for image classification?
- Is there a difference in model accuracy in classifying images using SVM vs CNN?
- reducing dimensions to save computation time and see if it helps in achieving good accuracy?
- What is the ideal model for image classification?

What is Image Classification? Why is it a challenging task?

Image classification is the process of classifying a picture or image into a category or class. The image classification task is challenging for various reasons . One of the primary challenges is to train a model and generalize it to the real-world data. In general, when we perform image classification, we must have a defined dataset and split it into training and testing datasets. Then we build a model and train it on the training set and then test it on the testing dataset which was created from the same original dataset. Since both these datasets are sampled from same pool, the model tend to perform better. However, when it comes to real-world scenarios the test set is a whole new dataset and most of the times the models which provided excellent accuracies also fail to classify new pool of images. This can be a huge challenge in tasks like autonomous driving cars.
For this project we are using Intel Image classification dataset from Kaggle. We believe this would be an intriguing topic as it is extremely relevant in today's society. A successful image classification model has proven to be a great technological advancement that can be useful in an abundance of industries. In this project our aim is to implement this task of image classification with algorithms like SVM and CNN by implanting it on same dataset and see if they perform better than the traditional algorithms.

Work flow of Image classification:



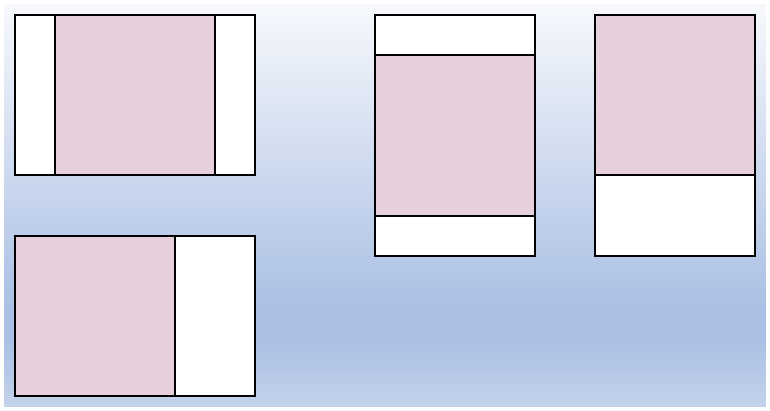
Input Data:

This Data contains around 25,000 images with a of size 150x150 distributed. Each of these images are classified under 6 categories namely 'buildings', 'forest', 'glacier', 'mountain', 'sea', 'street'. We plan on creating a model that can correctly classify each of these images into their perspective categories.

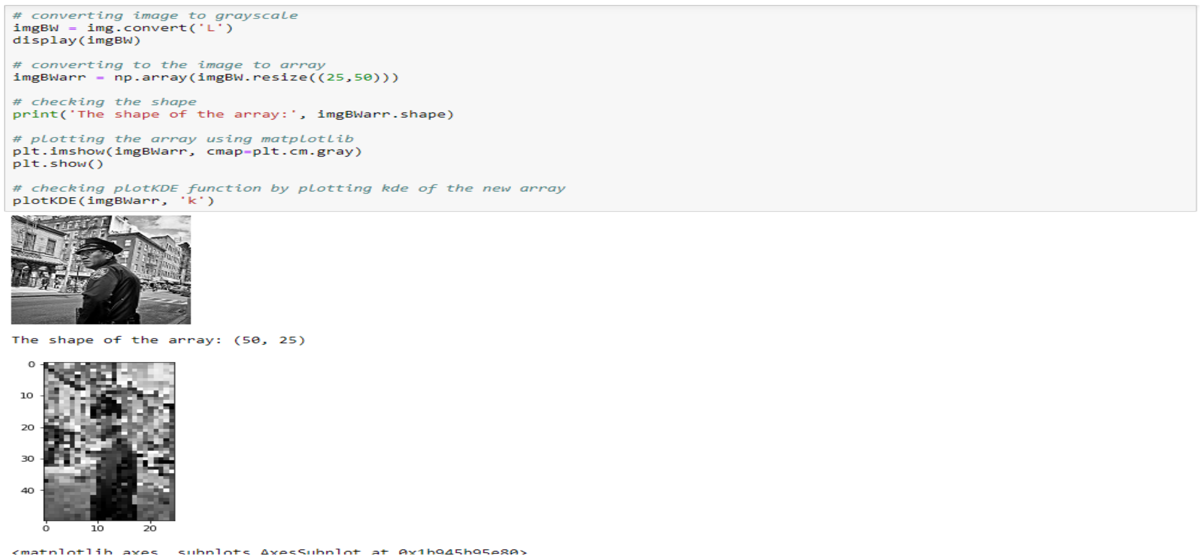
Image Pre-Processing:

1) **Uniform aspect ratio:** Our first step in the image preprocessing is to ensure all our images have same aspect ratio. Most of the algorithms perform better on a square Input images with uniform aspect ratio and hence we must check and crop all the images which are not square. Cropping is done by giving the importance to the center of image. Ensuring that all of our images have the aspect ratio enables us to perform an more feasible analysis.

   Coming to our Data since all our 25,000 images are already uniformly cropped and have same aspect ratio, I skipped this step of pre-processing.



2) **Gray scale conversion:** The most important reason of converting to grey scale is speed of computing. Our images have three channels which are R, G, B and processing these images takes almost 3 times more computational time than processing a greyscale image. In the process we are pursuing of our image classification, color is not needed and does not help us in identifying any edge of an image. Therefore, we can be ignore color and proceed with gray scale images.



```
# converting image to grayscale
imgBW = img.convert('L')
display(imgBW)

# converting to the image to array
imgBWarr = np.array(imgBW.resize((25,50)))

# checking the shape
print('The shape of the array:', imgBWarr.shape)

# plotting the array using matplotlib
plt.imshow(imgBWarr, cmap=plt.cm.gray)
plt.show()

# checking plotKDE function by plotting kde of the new array
plotKDE(imgBWarr, 'k')
```

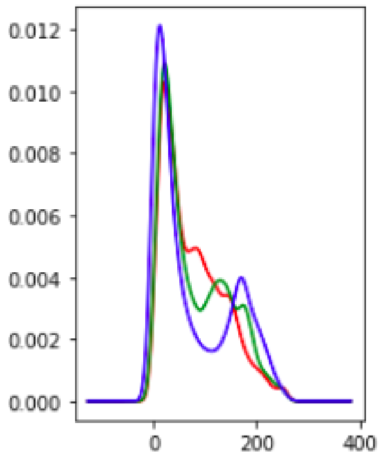
```
The shape of the array: (50, 25)
```

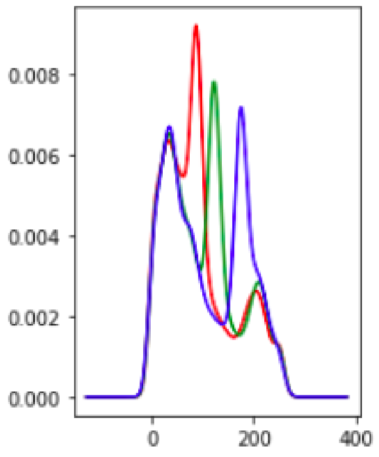
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b945b95e80>
```

3) Additional Visuals:

Prior to analyzing our models in gray scale, we wanted to take a look at the RGB(red, green and blue) values to examine potentially how a model for image classification could work using RGB values. Below is a Kernel Density Estimation graph. This plot displays the frequency of the RGB color values of the images to the left of them. Examining RGB values of an image could be an additional route in imaging classification. Although we did not take this approach, we believe it should be included for further insight.

Kernel Density Estimation





Feature Extraction:

In the process of feature extraction, we must only consider most important aspects of the image which will help in the image identification. In this process we convert the image into an array of feature vector which retains the most important information of an image.
There are many Feature Extraction methods employed for image classifications, some of them are listed as below

1) HOG (Histogram of oriented gradients)
2) SIFT (scale invariant feature transform)
3) SURF (speeded up robust feature)

We have used HOG (Histogram of Gradients) method to extract features of our images.
HOG is different to other feature descriptors for many reason which include:

- HOG Feature Extraction takes both the gradient and orientation of the edge into consideration while creating feature vectors. Where as in the other edge-based feature extraction methods, only give importance to the edge of an image, but not the orientation of gradients.
- In this method we divide the image into small parts called localized portions. Then the gradients and orientations of these localized portions are calculated. Finally, a histogram is generated foe each of these regions separately using gradients and orientation of the pixels.

HOG Calculation:

Step 1

Preprocessing data is done as described above and each of the images are resized in the ratio of 2:1. For my project I have resized all my images to the size (50,25).

Step 2

In the second step we calculate gradients. Gradients are nothing but change in the color intensities in both x and y directions. And these gradients are calculated for every pixel of the image. Let’s take the below for example, This is just an example matrix and the real representation of any image from our data set.

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Now let’s consider the highlighted pixel 85. The gradient value in the direction of X is obtained by subtracting the adjacent pixel intensities in the X direction and the gradient value in the direction of Y is obtained by subtracting the adjacent pixel intensities in the Y direction.

Hence the resultant gradients in the x and y direction for this pixel are:

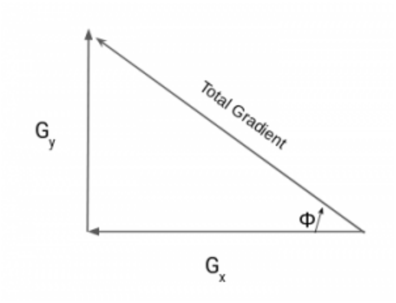
- Change in X direction (G_x) = 89 – 78 = 11
- Change in Y direction (G_y) = 68 – 56 = 8

This process of calculating gradients will give us two resultant matrices – one stores gradients in the x direction and other in the y direction. This process identifies the edges in the image because the magnitude of gradient is higher when a sharp change in the intensity is observed.

Once we calculate the gradients in both X and Y direction separately for all the pixels, we then must calculate the resultant magnitude and orientation for these values.

Step 3:

In the third step we calculate the net gradient magnitude and its direction for each pixel.



From previous steps we got Gx and Gy values as 11 and 8 respectively. Applying Pythagoras theorem, we calculate total gradient magnitude.

$$\text{Total Gradient Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]}$$

$$\text{Total Gradient Magnitude} = \sqrt{[(11)^2+(8)^2]} = 13.6$$

$$\text{Orientation of the pixel} = \tan(\Phi) = Gy / Gx$$

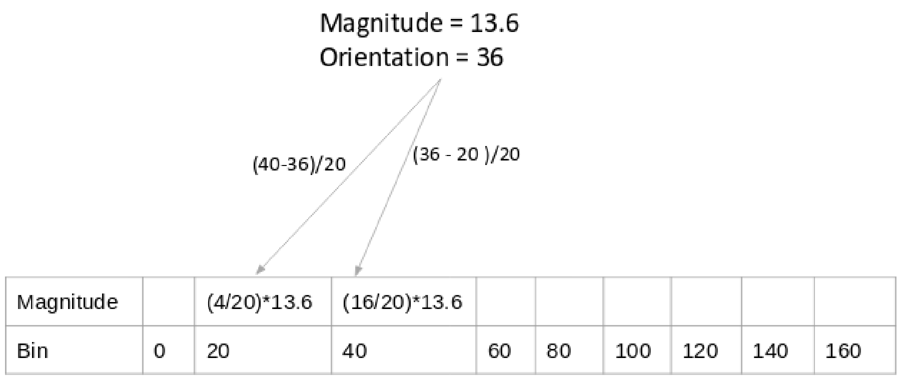
$$\Phi = \text{tan inverse } [(Gy / Gx)]$$

When we substitute the Gx and Gy values in the above equations we get the net magnitude value as 13.6 and orientation as 36

Step 4:

After getting net magnitude and orientation of gradients**.** We have to obtain an histogram by considering angle on x axis and frequency on y axis

- As the first step the orientation values from 0 to 180 are divided into 9 buckets with a bin size of 20.
- Second, for each pixel, we will check the orientation value, and store the corresponding pixel's gradient value to the bins on either side of the pixel gradient in the form of a 9 x 1 matrix as shown in the below fig.



Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

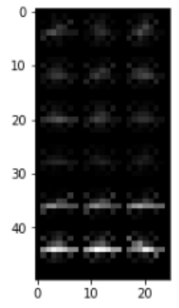
- The histograms created in the HOG feature descriptor are not generated for the whole image. Instead, the image is either divided into 16*16 or 8×8 cells and are calculated for these cells individually. The features of these smaller patches in turn represent the whole image.

```
# using hog ( Histogram Over Gradient ) function to get the hog features of the grey image

hogFeatures, hogImage = hog(graySea, visualize=True, pixels_per_cell=(8,8), cells_per_block= (2,2))

# hog Image
plt.imshow(hogImage, cmap=mpl.cm.gray)
```

```
<matplotlib.image.AxesImage at 0x1b9449f50f0>
```



```
def createFeatures(img):
    # flattening the colors of the image using img.flatten
    colorFeatures = img.flatten()

    # rgb2grey function to for converting the image to the greyscale
    grayImage = rgb2gray(img)

    # using hog function to get the hog features of the grey image
    hogFeatures = hog(grayImage, block_norm='L2-Hys', pixels_per_cell=(8,8))
    #combining the features
    flatFeatures = np.hstack((colorFeatures, hogFeatures))
    return flatFeatures

seaFeatures = createFeatures(sea)

# checking the shape of the matrix
print(seaFeatures.shape)

imageFeatures = seaFeatures
```

```
(4074,)
```

Step 5: Normalize gradients

Data Normalization is performed in order to ensure that every pixel of the image has similar data Distribution which in turn helps in training our data faster. Also, the gradients are sensitive to the overall lighting and in pictures usually some portions are brighter than the other and such lighting variations can be reduced by normalization. For Normalizing we have used Standard Scalar function which standardizes a feature by subtracting the mean and then scaling to unit variance. This Data Distribution Now Resembles a Gaussian Curve Centered At zero.

Step 6: Features for the complete image

As a final step the features of all the 8*8 blocks of our image are combined then these combined feature vectors are used for training our SVM Model.

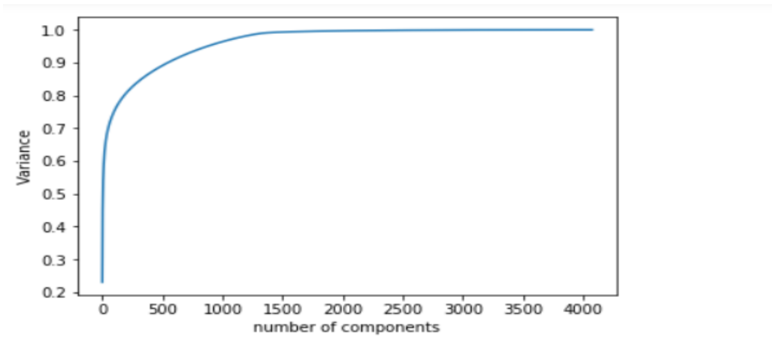
Dimensionality Reduction

For the Dimensionality Reduction I have Used PCA. After Extracting the Features, I Printed the shape of feature matrix to see the number of Features and the result was 4074.

```
# checking shape of feature matrix
print('The shape of the feature matrix is:', featureMatrix.shape)
```

```
The shape of the feature matrix is: (14034, 4074)
```

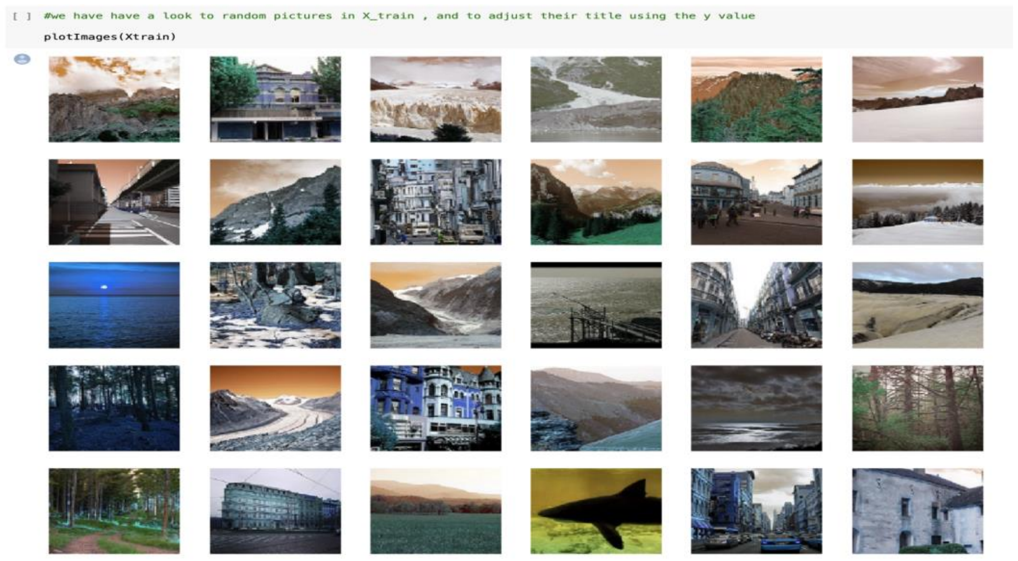
In order to check the appropriate number of components for PCA. I have plotted number of components vs variance scree plot and choose n = 1800 as this was appropriate number of components accounting for variance around 96 percent. Finally, I reduced the dimensions or features to 1800 using PCA.



```
pca = PCA(n_components = 1800)
# using fit_transform to run PCA on our standardized matrix
featureMatrix = pca.fit_transform(featureMatrix)
# look at new shape
print('PCA matrix shape is: ', featureMatrix.shape)
```

```
PCA matrix shape is:  (14034, 1800)
```

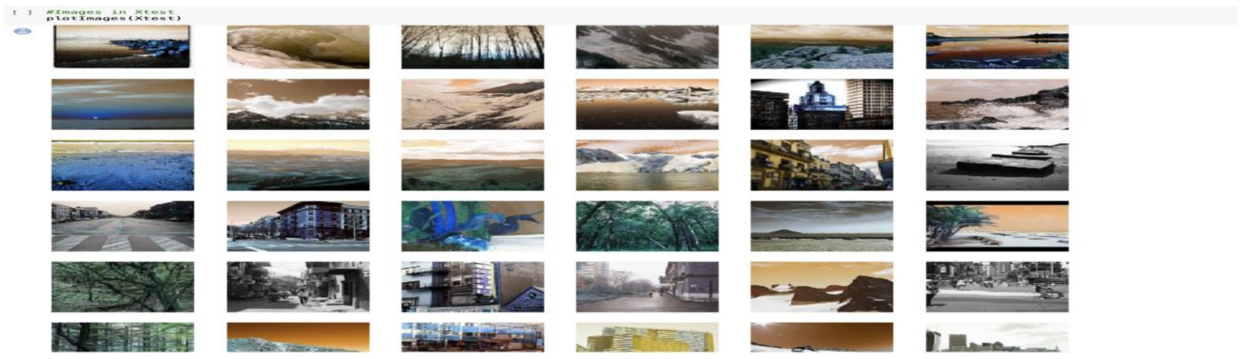
Training Dataset



```
#we have have a look to random pictures in X_train , and to adjust their title using the y value
plotImages(Xtrain)
```

- The training dataset has around 14,034 images which are unlabeled.

Testing Dataset



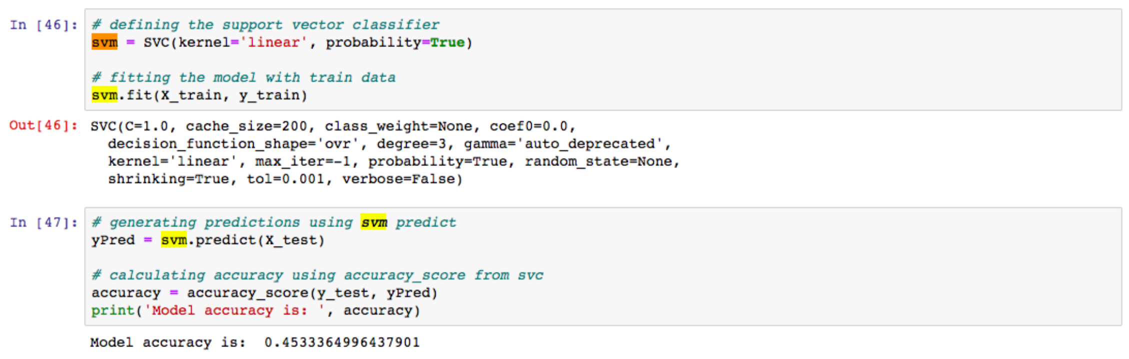
```
#Images in Xtest
plotImages(Xtest)
```

- The testing dataset contains around 3000 classified images:

Model Fitting

1) SVM CLASSIFIER

- The first model we used to train on the data was an SVM Classifier. We split the data to 70:30 of train and test sets and have used SVM's linear kernel to train my data.
- Support Vector Machine is defined by hyperplanes separating the input data based on categories. Support Vector Machine when feed with training data already labeled, generates a hyperplane as the output that basically categorizes the newly added data points optimally.
- With more than 2-dimensional data, Support Vector Machine generated a hyperplane while with 2-dimensional data a single line can be sufficient to classify the data points.
- Support Vector Machine has both the capability of performing regression as well as classification. Support Vector Machine can have either a linear or a non-linear kernel. This aids Support Vector Machine in capturing more details or insights and devoid of tedious calculations.
- Support Vector Machine avoid the struggle of handling high dimensions through linear functions unlike linear regression and rather optimizes the problem by transforming into two different convex quadratic programs. The loss function in Linear Regression penalizes the errors greater than 'ε' and thus being a sparse representative of decision rules and consecutively improving the representative and algorithmic benefits.
- We used linear kernel as the feature are many and its already in high dimension so linear kernel would do a good job of classification. and other parameters like Cost was taken as 1 to avoid over fitting.
- The model accuracy on the test dataset was 41 %


```
In [46]: # defining the support vector classifier
         svm = SVC(kernel='linear', probability=True)

         # fitting the model with train data
         svm.fit(X_train, y_train)

Out[46]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
           kernel='linear', max_iter=-1, probability=True, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

In [47]: # generating predictions using svm predict
         yPred = svm.predict(X_test)

         # calculating accuracy using accuracy_score from svc
         accuracy = accuracy_score(y_test, yPred)
         print('Model accuracy is: ', accuracy)

         Model accuracy is:  0.4533364996437901
```

2) CNN

The Convolutional Neutral Network or the CNN is used in this project as a model to classify images in different classes as mentioned. The model consist of 13 layers and trained to provide the prediction accuracy of 81% approximately.

The model is trained on the training dataset (Xtrain and ytrain) and evaluated on testing dataset (XTest and ytest).

The Model

We have used Keras layers and models to manage variables. Keras models and layers offer the convenient variables and trainable variables properties, which recursively gather up all dependent variables. This makes it easy to manage variables locally to where they are being used.

Sequential Model

In Keras, we assemble layers to build models. A model is (usually) a graph of layers. The most common type of model is a stack of layers: *the tf.keras.Sequential* model.

Layers

There are many tf.keras.layers available, most of them share some common constructor arguments, but we have used the following layers to apply image filters and train the model :

Dense: Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).

Conv2D: This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias = True, a bias vector is created and added to the outputs. Finally, if activation is not none, it is then applied to the outputs as well.

Dropout: Dropout layer (mainly used to prevent overfitting) consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

MaxPool2D: It provides Maximum pooling operation for spatial data.

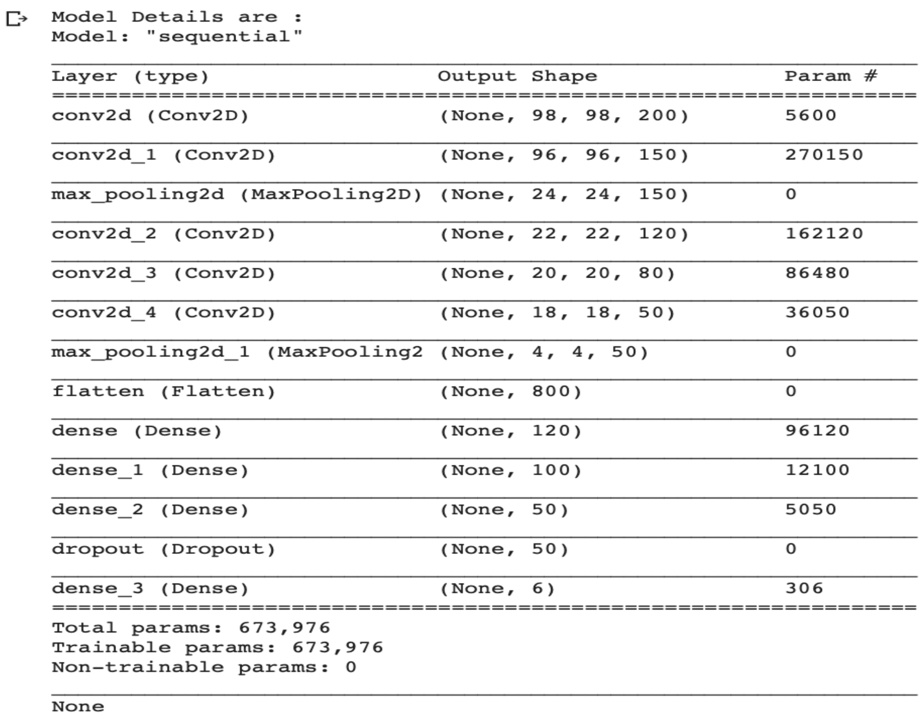
Flatten: It flattens the input. Does not affect the batch size.

Activation

relu : Rectified Linear Unit: It is a piece-wise activation function which will output the input directly if it is positive else will return the zero
softmax : Softmax activation function

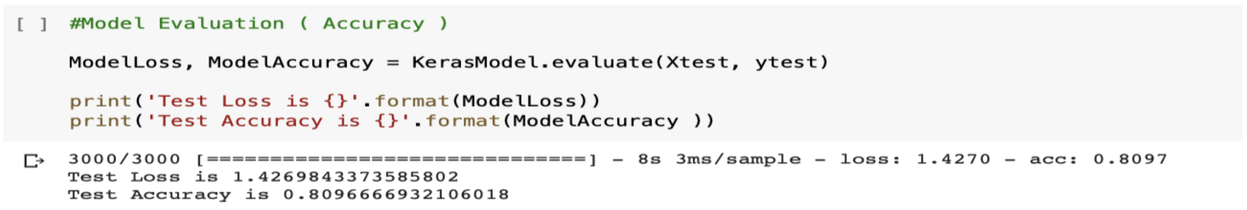
Model Evaluation

We used Keras model evaluator in the package to evaluate the accuracy of the model on the test.



```
Model Details are :
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 200)       5600
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 96, 96, 150)       270150
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 150)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 120)       162120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 20, 20, 80)        86480
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 18, 18, 50)        36050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 50)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 120)               96120
_________________________________________________________________
dense_1 (Dense)              (None, 100)               12100
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout (Dropout)            (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 673,976
Trainable params: 673,976
Non-trainable params: 0
_________________________________________________________________
None
```

Predicted Results: 80.96667%



```
[ ] #Model Evaluation ( Accuracy )

    ModelLoss, ModelAccuracy = KerasModel.evaluate(Xtest, ytest)

    print('Test Loss is {}'.format(ModelLoss))
    print('Test Accuracy is {}'.format(ModelAccuracy ))

    3000/3000 [==============================] - 8s 3ms/sample - loss: 1.4270 - acc: 0.8097
    Test Loss is 1.4269843373585802
    Test Accuracy is 0.8096666932106018
```

Conclusion:

Overall, image classification is an interesting and compelling topic that can provide a plethora of uses to enhance society. We used two models to compare and contrast the ideal method for image classification. Due to the complexity of a CNN model we are able to examine how much more efficient this model is than SVM. Furthermore, using HOG analysis was a way to augment our models to increase its accuracy. Using gradients and orientation of the edge of an image proves that it will enhance the model. Taking a look at other feature extraction methods such as SIFT and SURF and how they would affect our models would be interesting in examine the overall ideal feature extraction to include in a CNN model. Along with that, we were unable to use RGB channels in our analysis, due to computing speed. That too would be interesting to see how the color values affect our model.

References

1) https://medium.com/@dataturks/understanding-svms-for-image-classification-cf4f01232700
2) https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/
3) https://www.kaggle.com/puneet6060/intel-image-classification/kernels
4) https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
5) https://www.sciencedirect.com/topics/computer-science/support-vector-machine
6) https://keras.io/layers/convolutional/