CSE480

Machine Vision

Major Task (1)

Group Members:

Name	*ID*
Mostafa Abdelnasser Hassan	1801145
Mahmoud Gamal Mohamed	1802166
Ahmed Hossam El Din Ramadan	1807260

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A basic challenge is image classification, which aims to understand an image as a whole. The objective is to categorize the image by giving it a certain name. Image Classification often relates to the analysis of photographs with just one item visible. Contrarily, object detection analyses more realistic scenarios where several items may be present in a picture and involves both classification and localization tasks.

The issue at hand is picture categorization. The job of assigning a label to an input image from a predetermined set of categories is known as image classification. It serves as the foundation for various computer vision challenges.

INTRODUCTION

The task of providing a name or class to a complete image is known as image classification. Images are supposed to contain no more than one class. Image classification algorithms accept a picture as input and estimate which class the image belongs to. Image classification, also known as image recognition, is the problem of assigning one (single-label classification) or more (multi-label classification) labels to a given image. Image Classification is a solid problem for benchmarking contemporary computer vision architectures and approaches.

The vast quantity of photos, the high complexity of the data, and the absence of labelled data are the key obstacles in image classification. Images with a significant number of pixels might be quite huge. Each image's data may be multidimensional, with many separate attributes.

Applications for image classification are employed in a variety of fields, including medical imaging, satellite image object identification, traffic control systems, brake light detection, machine vision, and more.

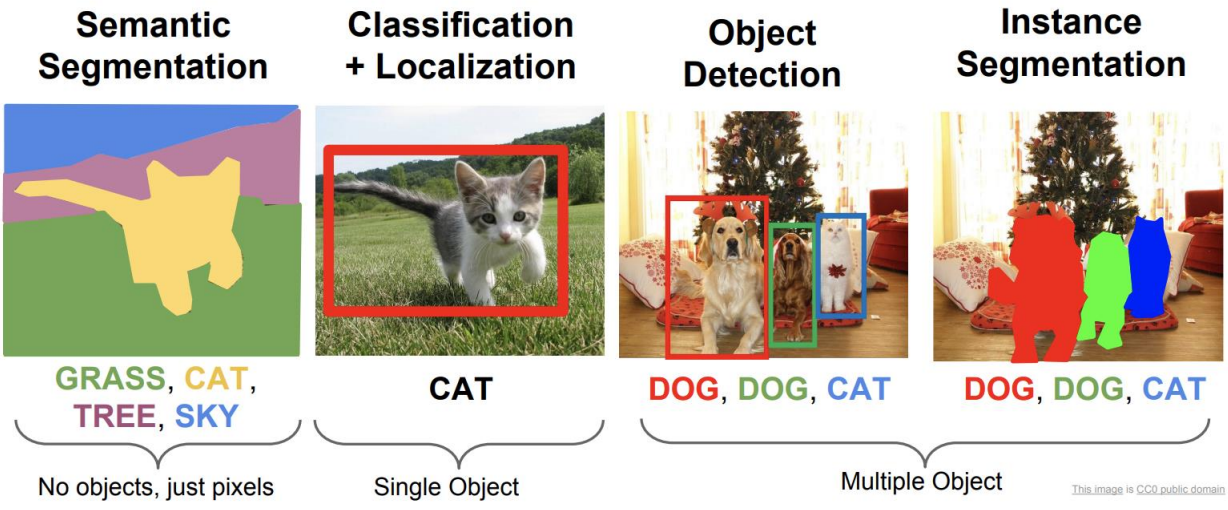

Figure 1: Image Classification

METHODS AND ALGORITHMS

2.1. Histograms of Oriented Gradients (HOG)

When using the HOG approach, first build and then normalise histograms of the distribution of gradient orientations in an image. Because of this normalisation, HOG can identify the edges of objects even when the contrast is very low. These normalised histograms are merged into a feature vector known as the HOG descriptor to train a machine learning system, such as a Support Vector Machine (SVM), to recognise objects in pictures based on their boundaries (edges).

2.1.1. Algorithm steps

1. Using a photo of a specific item
2. Select a region of interest that encompasses the entirety of the object in the image.
3. Determine the gradient's strength and direction for each pixel within the detection window.
4. Create connected, uniform-sized pixels cells of the same size within the detection window.

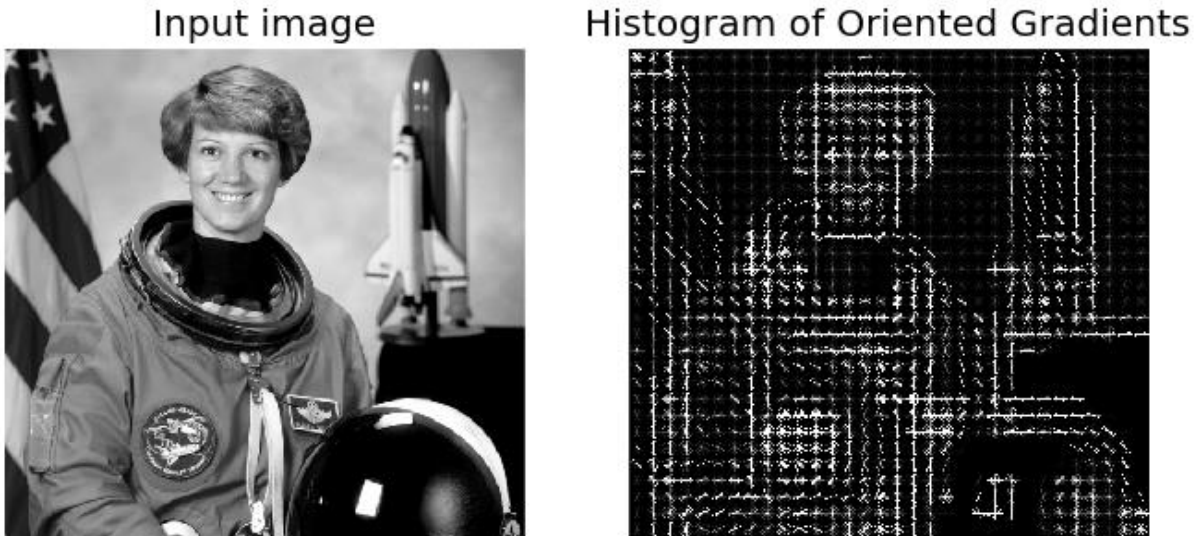

Figure 2: HOG Classifier

5. Group the gradient axes of all the pixels in each cell into a predetermined number of orientation (angular) bins, and then combine the gradient intensities of the gradients in each angular bin to create a histogram for each cell.
6. Group adjacent cells into blocks.
7. To normalise the cell histograms in each block, use the cells that make up that block.
8. Gather each block's normalised histograms into a single feature vector known as the HOG descriptor so that a machine learning system may be trained using it.

2.2. K-Nearest Neighbor (KNN) Classifier

To comprehend KNN, we will design a circumstance in which we have two unique categories. What would happen if a new data point appeared? KNN will assess if it belongs in the red or green categories.

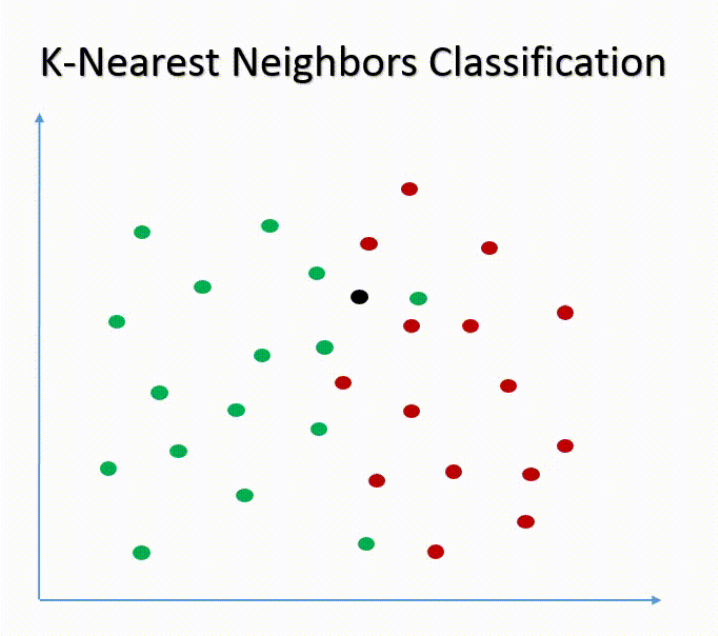

Figure 3: KNN Classifier demonstration

2.2.1. Algorithmic Steps:

1. Select the K-th neighbour.
2. Take the closest neighbor K of the new data point based on Euclidean distance.
3. Count the number of data points in each category among these k neighbours.
4. A new data point should be assigned to the category where you counted the most neighbours.

KNN is a simple and user-friendly algorithm. The issue with KNN is that when the number of instances, predictors, or independent variables increases, it becomes substantially slower.

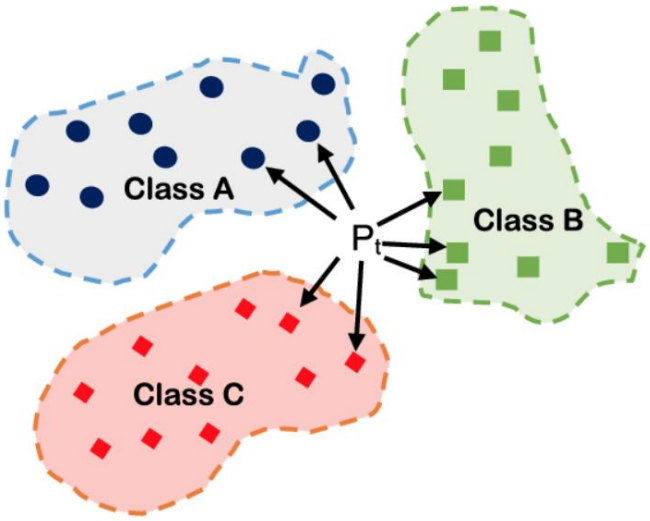

Figure 4: KNN Classification

2.3. K-Mean Classifier

The K-Mean technique is used to cluster your data. The K-means clustering algorithm is used to detect groupings in data that have not been explicitly categorised. This can be used to validate business assumptions about the types of groups that exist or to identify unknown groups in large data sets.

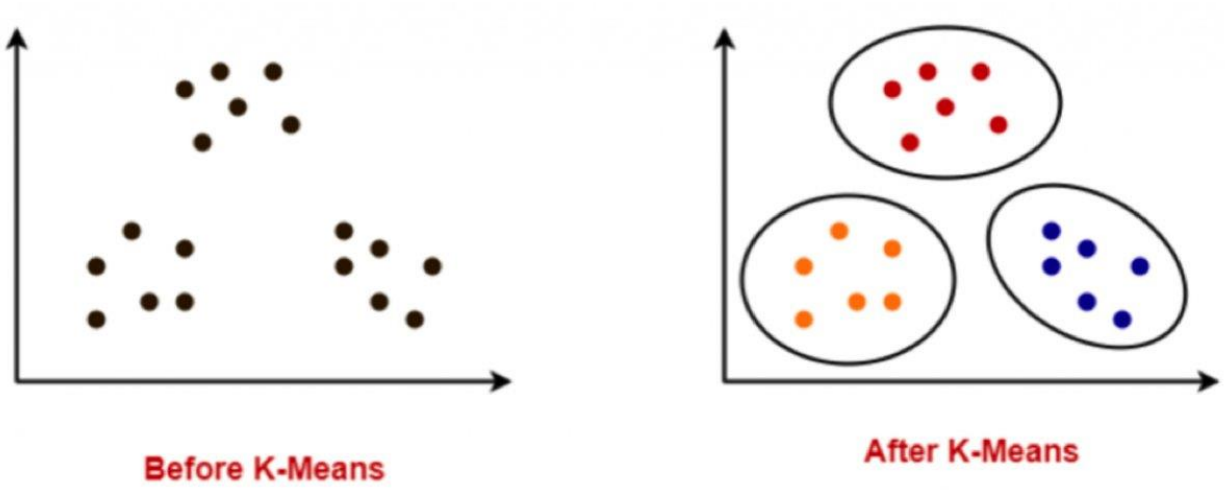

Figure 5: K-means Classifier

2.3.1. Algorithm

Input:

D= {d1, d2, …, dn} // set of n data items.

k // Number of desired cluster

Output:

A set of k clusters.

Steps:

1. Arbitrarily choose k data-items from D as initial centroids.
2. Repeat

   Assign each item di to the cluster which has the closest centroid;

   Calculate new mean for each cluster;

   Until convergence criteria are met.

2.4. Support Vector Machine (SVM) Classifier

Consider the following scenario: we have two separate categories. What would happen if a new data point appeared? SVM determines if it belongs in the red or green category. One technique to determining the border between the two groups is to ask which line to draw. When attempting to categorise or divide two classes, SVM is all about drawing the optimal decision boundary, which is a hyperplane between them.

SVM has the highest margin. SVM can easily handle many continuous and categorical variables. SVM produces a hyperplane in multidimensional space to split multiple classes. SVM iteratively constructs an ideal hyperplane to minimise error. The main goal of SVM is to identify the maximum marginal hyperplane (MMH) that divides the dataset most effectively.

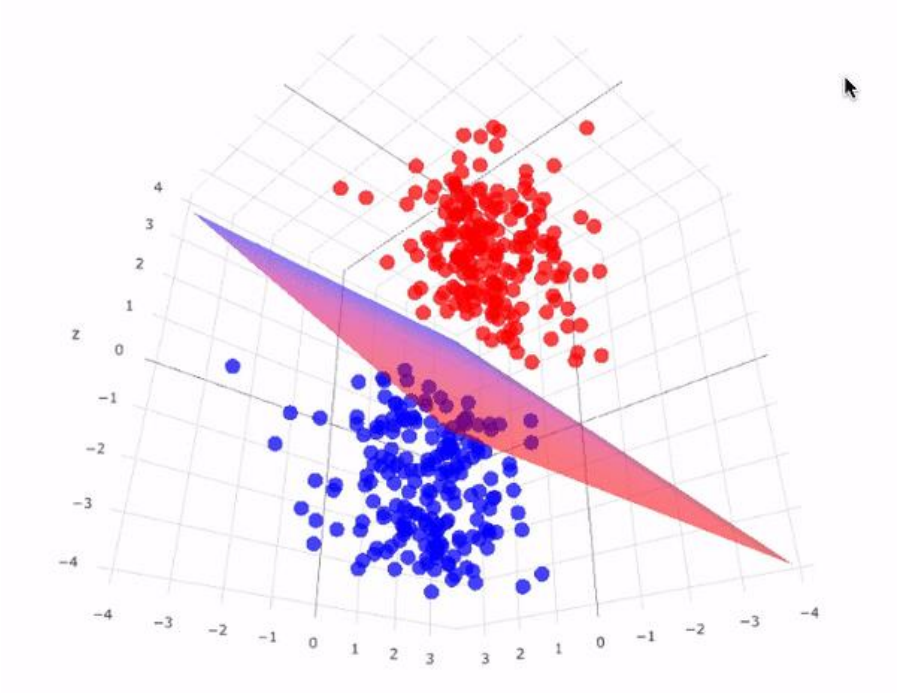

Figure 6: SVM Classifier demonstration

CODES AND OUTPUTS

The Results below are obtained by applying different image classification methods over the CIFAR-100 dataset which is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks.

You can find the code of the following snapshots here: **Machine Vision Project**

3.1. HOG Code

```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

```
def cv2_imshow(img):
    if img.ndim > 2:
        #img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)
    else:
        plt.imshow(img, cmap="gray")
    plt.show()
```

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

```
training_size = 50000
testing_size = 10000
```

```
training_dataset = unpickle('images/cifar-100-python/train')
testing_dataset = unpickle('images/cifar-100-python/test')
```

```
training_images = training_dataset[b'data']
training_labels = training_dataset[b'fine_labels']
testing_images = testing_dataset[b'data']
testing_labels = testing_dataset[b'fine_labels']
```

```
training_images = training_images[:training_size]
training_labels = training_labels[:training_size]
testing_images = testing_images[:testing_size]
testing_labels = testing_labels[:testing_size]
```

```
training_images = training_images.reshape(len(training_images),3,32,32).transpose(0,2,3,1)
training_labels = np.array(training_labels)
training_labels = training_labels.reshape(len(training_images), 1)

testing_images = testing_images.reshape(len(testing_images),3,32,32).transpose(0,2,3,1)
testing_labels = np.array(testing_labels)
testing_labels = testing_labels.reshape(len(testing_images), 1)
```

```
print(training_images.shape)
print(training_labels.shape)
print(testing_images.shape)
print(testing_labels.shape)
```

```
h = []
h_test = []
```

```
hog = cv2.HOGDescriptor((32,32), (16,16), (8,8), (8,8),9)
```

```
for x in training_images:
    h.append(hog.compute(x))

for x in testing_images:
    h_test.append(hog.compute(x))

print("hog computed")
```

3.1.1. HOG Results

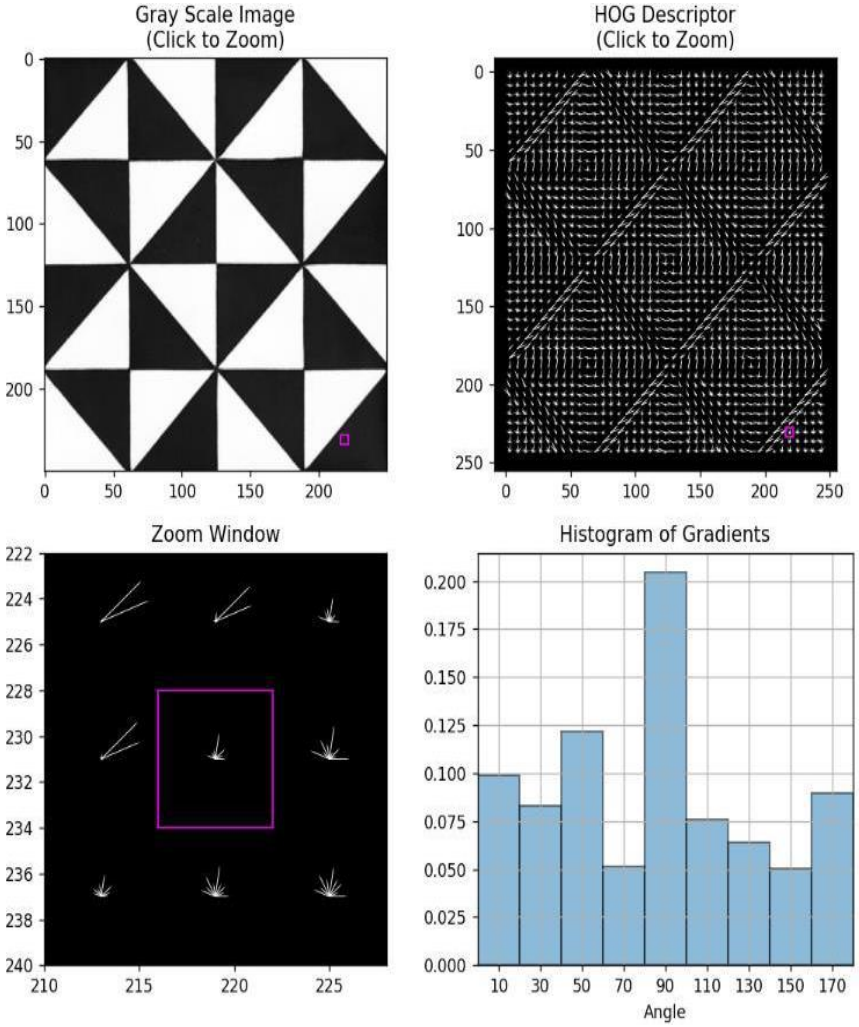

Figure 7: HOG Results

3.2. KNN Code

```
[1]  import os
     import cv2
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
```

```
[2]  def unpickle(file):
         import pickle
         with open(file, 'rb') as fo:
             dict = pickle.load(fo, encoding='bytes')
         return dict
```

```
[3]  training_size = 50000
     testing_size = 10000
```

```
[4]  training_dataset = unpickle('images/cifar-100-python/train')
     testing_dataset = unpickle('images/cifar-100-python/test')
```

```
[ ]  training_images = training_dataset[b'data']
     training_labels = training_dataset[b'fine_labels']
     testing_images = testing_dataset[b'data']
     testing_labels = testing_dataset[b'fine_labels']
```

```
[ ]  training_images = training_images[:training_size]
     training_labels = training_labels[:training_size]
     testing_images = testing_images[:testing_size]
     testing_labels = testing_labels[:testing_size]
```

```
[ ]  training_images = training_images.reshape(len(training_images),3,32,32).transpose(0,2,3,1)
     training_labels = np.array(training_labels)
     training_labels = training_labels.reshape(len(training_images), 1)

     testing_images = testing_images.reshape(len(testing_images),3,32,32).transpose(0,2,3,1)
     testing_labels = np.array(testing_labels)
     testing_labels = testing_labels.reshape(len(testing_images), 1)
```

```
[ ]  print(training_images.shape)
     print(training_labels.shape)
     print(testing_images.shape)
     print(testing_labels.shape)
```

```
[ ]  h = []
     h_test = []
```

```
[ ]  hog = cv2.HOGDescriptor((32,32), (16,16), (8,8), (8,8),9)
```

```
[ ]  for x in training_images:
         h.append(hog.compute(x))

     for x in testing_images:
         h_test.append(hog.compute(x))

     print("hog computed")
```

```
     hog computed
```

```
[ ]  h = np.array(h)
     h_test = np.array(h_test)
     print(h.shape)
     print(h_test.shape)
```

```
     (50000, 324)
     (10000, 324)
```

```
[ ]  x_train = h
     y_train = training_labels

     x_test = h_test
     y_test = testing_labels
```

```
[ ]  y_train=y_train.reshape(-1,)
     y_test=y_test.reshape(-1,)
     print(y_test[0])
```

```
[ ]  def knn_distances(xTrain,xTest,k):
         import numpy as np
         distances = -2 * xTrain@xTest.T + np.sum(xTest**2,axis=1) + np.sum(xTrain**2,axis=1)[:, np.newaxis]
         distances[distances < 0] = 0
         distances = distances**.5
         indices = np.argsort(distances, 0)
         distances = np.sort(distances,0)
         return indices[0:k,:], distances[0:k,:]
```

```
[ ]  def knn_predictions(xTrain,yTrain,xTest,k=8):
         import numpy as np
         indices, distances = knn_distances(xTrain,xTest,k)
         yTrain = yTrain.flatten()
         rows, columns = indices.shape
         predictions = list()
         for j in range(columns):
             temp = list()
             for i in range(rows):
                 cell = indices[i][j]
                 temp.append(yTrain[cell])
             predictions.append(max(temp,key=temp.count))
         predictions=np.array(predictions)
         return predictions
```

```
[ ]  def knn_accuracy(yTest,predictions):
         x=yTest.flatten()==predictions.flatten()
         grade=np.mean(x)
         return np.round(grade*100,2)
```

```
[ ]  predictions = knn_predictions(x_train, y_train, x_test,8)
     print('Size of Predictions Array:\n', predictions.shape)
```

```
Size of Predictions Array:
 (10000,)
```

```
[ ]  print('Accuracy:',knn_accuracy(predictions,y_test),'%')
```

```
Accuracy: 24.74 %
```

```
[ ]  print(predictions[1])
     print(y_test[1])
```

```
45
33
```

```
[ ]  def acc(yTest,predictions):
         correct = 0
         for i in range(10000):
             if (predictions[i]-yTest[i]) == 0:
                 correct = correct + 1
         return (correct / 10000) * 100
```

```
[ ]  print('Accuracy:',acc(predictions,y_test),'%')
```

```
Accuracy: 24.740000000000002 %
```

3.2.1. KNN Results

```
In [18]: predictions = knn_predictions(x_train, y_train, x_test,8)
         print('Size of Predictions Array:\n', predictions.shape)

         Size of Predictions Array:
          (10000,)
```

```
In [19]: print('Accuracy:',knn_accuracy(predictions,y_test),'%')

         Accuracy: 24.74 %
```

Figure 8: KNN accuracy 24.74%

3.3. K-mean

```
import matplotlib.pyplot as plt
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

```
training_size = 50000
testing_size = 10000
```

```
training_dataset = unpickle('images/cifar-100-python/train')
testing_dataset = unpickle('images/cifar-100-python/test')
```

```
training_images = training_dataset[b'data']
training_labels = training_dataset[b'fine_labels']
testing_images = testing_dataset[b'data']
testing_labels = testing_dataset[b'fine_labels']
```

```
training_images = training_images[:training_size]
training_labels = training_labels[:training_size]
testing_images = testing_images[:testing_size]
testing_labels = testing_labels[:testing_size]
```

```
training_images = training_images.reshape(len(training_images),3,32,32).transpose(0,2,3,1)
training_labels = np.array(training_labels)
print(testing_labels.shape)
```

```
[ ]  h = []
     h_test = []
```

```
[ ]  hog = cv2.HOGDescriptor((32,32), (16,16), (8,8), (8,8),9)
```

```
[ ]  for x in training_images:
         h.append(hog.compute(x))

     for x in testing_images:
         h_test.append(hog.compute(x))

     print("hog computed")
```

```
     hog computed
```

```
[ ]  x_train = h
     y_train = training_labels

     x_test = h_test
     y_test = testing_labels
```

```
[ ]  y_train=y_train.reshape(-1,)
     y_test=y_test.reshape(-1,)
```

```
[ ]  def kMeans_init_centroids(X, K):
         randidx = np.random.permutation(X.shape[0])
         centroids = X[randidx[:K]]
         return centroids
```

```
[ ]  def cost_kmeans(X,Y,centroids):
         total_cost=np.zeros_like(centorids)
         clusters=0
         for d in range(centroids.shape[0]):
             cost=0
             m =0
             for i in range(X.shape[0]):
         clusters=0
         for d in range(centroids.shape[0]):
             points=np.zeros_like(X[1])
             m =0
             for i in range(X.shape[0]):
                 if Y[i]==clusters:
                     m+=1
                     points+=X[i]
             centroids[clusters]=(1/m)*(points)
             clusters+=1
         return centroids
```

```
[ ]  k=100
     centorids =kMeans_init_centroids(h,k)
     print(centorids)
     print("\n\n")
     P_cost=np.zeros_like(centorids)
     cost=cost_kmeans(h,y_train,centorids)
     centorids=mean_kmeans(h,y_train,centorids)
     print(centorids)
```

```
[ ]  def compute_centroids(X, idx, K):
         m, n = X.shape
         centroids = np.zeros((K, n))
         for i  in range(K):
             k_pi=X[idx == i]
             centroids[i] = np.mean(k_pi, axis = 0)
         return centroids
```

```
[ ]  def Kmean_acc(idx,y_train):
         correct=0
         for i in range(50000):
             if (idx[i]-y_train[i])==0:
                 correct = correct + 1
         acc = (correct/50000)*100
         print('Accuracy:',acc,'%')
         return idx
```

```
[ ]  def run_kMeans(X, initial_centroids, max_iters=10):
         m, n = X.shape
         K = initial_centroids.shape[0]
         centroids = initial_centroids
         previous_centroids = centroids
         idx = np.zeros(m)
         for i in range(max_iters):
             print("K-Means iteration %d/%d" % (i, max_iters-1))
             idx = find_closest_centroids(X, centroids)
             centroids = compute_centroids(X, idx, K)
         return centroids, idx
```

```
[ ]  centroids, idx = run_kMeans(x_train,centorids,max_iters=1)

     K-Means iteration 0/0
```

```
[ ]  print (idx)
     print (y_train)
```

3.3.1. K-mean Results

```
Kmean_acc(idx,y_train)

Accuracy: 16.342000000000002 %
```

Figure 9: K-mean accuracy 16.342%

3.4. SVM

```
[ ] import os
    import cv2
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    %matplotlib inline
    from sklearn.model_selection import train_test_split
    from sklearn.svm import SVC
    from sklearn.metrics import classification_report, confusion_matrix
```

```
    def cv2_imshow(img):
        if img.ndim > 2:
            #img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            plt.imshow(img)
        else:
            plt.imshow(img, cmap="gray")
        plt.show()
```

```
[ ] def unpickle(file):
        import pickle
        with open(file, 'rb') as fo:
            dict = pickle.load(fo, encoding='bytes')
        return dict
```

```
[ ] training_size = 50000
    testing_size = 10000
```

```
[ ] training_dataset = unpickle('images/cifar-100-python/train')
    testing_dataset = unpickle('images/cifar-100-python/test')
```

```
[ ] training_images = training_dataset[b'data']
    training_labels = training_dataset[b'fine_labels']
    testing_images = testing_dataset[b'data']
    testing_labels = testing_dataset[b'fine_labels']
```

```
[ ] training_images = training_images[:training_size]
    training_labels = training_labels[:training_size]
    testing_images = testing_images[:testing_size]
    testing_labels = testing_labels[:testing_size]
```

```
[ ] training_images = training_images.reshape(len(training_images),3,32,32).transpose(0,2,3,1)
    training_labels = np.array(training_labels)
    training_labels = training_labels.reshape(len(training_images), 1)

    testing_images = testing_images.reshape(len(testing_images),3,32,32).transpose(0,2,3,1)
    testing_labels = np.array(testing_labels)
    testing_labels = testing_labels.reshape(len(testing_images), 1)
```

```
[ ]  print(training_images.shape)
     print(training_labels.shape)
     print(testing_images.shape)
     print(testing_labels.shape)
```

```
[ ]  h = []
     h_test = []
```

```
[ ]  hog = cv2.HOGDescriptor((32,32), (16,16), (8,8), (8,8),9)
```

```
[ ]  for x in training_images:
         h.append(hog.compute(x))

     for x in testing_images:
         h_test.append(hog.compute(x))

     print("hog computed")
```

```
     hog computed
```

```
[ ]  h = np.array(h)
     h_test = np.array(h_test)
     print(h.shape)
     print(h_test.shape)
```

```
     (50000, 324)
     (10000, 324)
```

```
[ ]  x_train = h
     y_train = training_labels

     x_test = h_test
     y_test = testing_labels
```

```
[ ]  y_train=y_train.reshape(-1,)
     y_test=y_test.reshape(-1,)
```

```
[ ]  svclassifier = SVC(kernel='rbf', random_state = 0)
     svclassifier.fit(x_train, y_train)
```

```
[ ]  y_pred = svclassifier.predict(x_test)

[ ]  cm = confusion_matrix(y_test, y_pred)
     print(cm)

     [[60  1  0 ...  1  1  2]
      [ 0 37  0 ...  0  0  1]
      [ 1  1  6 ...  2  3  1]
      ...
      [ 0  1  1 ... 19  2  0]
      [ 0  0  3 ...  0 15  2]
      [ 0  0  0 ...  0  0 37]]

[ ]  cr=classification_report(y_test,y_pred)
     print(cr)
```

3.4.1. SVM Results

```
cr=classification_report(y_test,y_pred)
print(cr)
```

	precision	recall	f1-score	support
0	0.67	0.60	0.63	100
1	0.31	0.37	0.34	100
2	0.15	0.06	0.09	100
3	0.14	0.08	0.10	100
...	...	...	...	...
99	0.37	0.37	0.37	100
accuracy			0.31	10000
macro avg	0.31	0.31	0.30	10000
weighted avg	0.31	0.31	0.30	10000

Figure 10: SVM accuracy = 31%

3.5. Comment

SVM has the highest accuracy among them i.e., it is the most suitable classifier for our data.