Bakhtiyar Abbasov 18011114

# Ödev 2

## 1.Methods:

Utilized libraries are: **cifar10**, **numpy**, **colorsys**, and **matplotlib**. Batch 1 of CIFAR-10 is taken as an Image Dataset. Because of the issues on my personal computer, I never run this code locally. All the outputs and images are displayed through the Google Colab environment.

**Functions:**

***unpickle(file):*** This function takes a batch file as an input and returns a dictionary containing image arrays and labels.

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

***filter(classNo):*** This function takes class name as an argument and returns array value images of that class in the dictionary.

```
def filter(classNo):
  indices = np.where(labels == classNo)[0]
  filtered_data = pixels[indices[:60]]
  return filtered_data
```

***cosine_sim(array1, array2, i, j):*** This function takes two arrays as an input and calculates Cosine Similarity. “IF” condition is used for checking the “Divide by Zero” error since there can be gray images in the dataset, resulting in Hue histograms made of zero.

```
def cosine_sim(array1, array2, i, j):
  A = np.hstack(array1)
  B = np.hstack(array2)
  dot = np.dot(A, B)
  normA = np.linalg.norm(A)
  normB = np.linalg.norm(B)
  if(normA and normB):
    cosine = dot / (normA * normB)
    return cosine
  else:
    return 0
```

***similarImages(train_set, test_set):*** This function takes a training dataset consisting of 250 images and a test set of 10 images belonging to a certain class. After iterating through every test image and calculating similarity between training images, it stores maximum 3 values and indexes of them into the "similarIndex" array.

```
def similarImages(train_set, test_set):
  similar_index = np.zeros((10, 3), int)
  for i, test in enumerate(test_set):
    max = np.zeros(3)
    index = np.ones(3, int)
    for j, train in enumerate(train_set):
      cosine = cosine_sim(test, train, i, j)
      if cosine > max[0]:
        max[2] = max[1]
        max[1] = max[0]
        max[0] = cosine
        index[2] = index[1]
        index[1] = index[0]
        index[0] = j
    #print(index)
    similar_index[i][0] = index[0]
    similar_index[i][1] = index[1]
    similar_index[i][2] = index[2]

  return similar_index
```

***calculateSuccess(array, classNo):*** This function takes a similarity array that contains indexes of similar images. Then it calculates if the indexed images belong to the given class. A boolean array that matches the similarity array is created, filled with "True" values in place of indexes that belong to the given class. Since a minimum of one True value is necessary for success, any method is utilized for counting successful matches. The success rate is then calculated and returned.

```
def calculateSuccess(array, classNo):
  classCheck = np.zeros((10, len(array[0])), bool)
  for i in range(len(array)):
    for j in range(len(array[0])):
      if(int(array[i][j] / 50) - classNo):
        classCheck[i][j] = False
      else:
        classCheck[i][j] = True

  success = np.any(classCheck, axis=1)
  success_rate = (np.sum(success) / 10) * 100

  return success_rate
```

***displayImage(array, classNo):*** This function takes raw image data from the original dataset (1-dimensional 3072 element array) and converts it to a 32x32x3 RGB image array.

```
def displayImage(pixels):
  img_R = pixels[0:1024].reshape((32, 32))
  img_G = pixels[1024:2048].reshape((32, 32))
  img_B = pixels[2048:3072].reshape((32, 32))
  img = np.dstack((img_R, img_G, img_B))
  return img
```

***plotArray and plotImages:*** These functions are used for displaying the output images that are included in the report for every class.

```
def plotArray(pixelArray, className, classTest):
  image = np.empty((10, 9), object)
  for i in range(len(pixelArray)):
    image[i][0] = displayImage(className[i+50])
    image[i][1] = classTest[i]
    image[i][2] = classTest[i+10]
    for j in range(3,9):
      image[i][j] = displayImage(train_data[pixelArray[i][j-3]])

  return image
```

```
def plotImages(display):
    fig, axes = plt.subplots(10, 9, figsize=(20, 25))
    for i in range(10):
      axes[i, 0].set_title("Original Image")
      axes[i, 1].set_title("RGB Hist Image")
      axes[i, 2].set_title("HSV Hist Image")
      for j in range(9):
        axes[i, j].imshow(display[i, j])
        axes[i, j].axis('off')
        if(j>2 and j<6):
            axes[i, j].set_title(f"RGB Similar Image {j-2}")
        elif j>5:
            axes[i, j].set_title(f"HSV Similar Image {j-5}")
    plt.tight_layout()
    plt.show()
```

# 2.Application:

The original image, its Y and H histograms, and most similar RGB and HSV images are displayed below for every class :

# BIRD


Original Image
RGB Hist Image
HSV Hist Image
RGB Similar Image 1
RGB Similar Image 2
RGB Similar Image 3
HSV Similar Image 1
HSV Similar Image 2
HSV Similar Image 3

# CAT


Original Image
RGB Hist Image
HSV Hist Image
RGB Similar Image 1
RGB Similar Image 2
RGB Similar Image 3
HSV Similar Image 1
HSV Similar Image 2
HSV Similar Image 3

# DEER


Original Image
RGB Hist Image
HSV Hist Image
RGB Similar Image 1
RGB Similar Image 2
RGB Similar Image 3
HSV Similar Image 1
HSV Similar Image 2
HSV Similar Image 3

# DOG


Original Image
RGB Hist Image
HSV Hist Image
RGB Similar Image 1
RGB Similar Image 2
RGB Similar Image 3
HSV Similar Image 1
HSV Similar Image 2
HSV Similar Image 3

# FROG


Original Image
RGB Hist Image
HSV Hist Image
RGB Similar Image 1
RGB Similar Image 2
RGB Similar Image 3
HSV Similar Image 1
HSV Similar Image 2
HSV Similar Image 3

# 3.Conclusion:

==========
Success of RGB histogram comparison for BIRD: 90.0
Success of HSV histogram comparison for BIRD: 80.0
Total success of histogram comparison for BIRD: 100.0
==========
Success of RGB histogram comparison for CAT: 30.0
Success of HSV histogram comparison for CAT: 30.0
Total success of histogram comparison for CAT: 60.0
==========
Success of RGB histogram comparison for DEER: 90.0
Success of HSV histogram comparison for DEER: 40.0
Total success of histogram comparison for DEER: 90.0
==========
Success of RGB histogram comparison for DOG: 20.0
Success of HSV histogram comparison for DOG: 10.0
Total success of histogram comparison for DOG: 30.0
==========
Success of RGB histogram comparison for FROG: 0.0
Success of HSV histogram comparison for FROG: 30.0
Total success of histogram comparison for FROG: 30.0
==========
Total success of RGB histogram comparison: 46.0
Total success of HSV histogram comparison: 38.0
Total success of histogram comparison: 62.0
==========

The analysis of RGB and HSV histograms across different classes showcases varying success rates in classification. While RGB tends to excel in certain classes, like Bird and Deer, it struggles in others, like Frog. The HSV histogram comparison showed relatively lower success rates across different classes compared to the RGB. It presented a mixed performance, offering decent results in some classes but being less effective overall. The overall success of the histograms together across all classes was 62.0%. RGB histograms achieved an overall success rate of 46.0%, while HSV histograms scored 38.0%. It's important to note that these results are based on a specific batch from the dataset and a relatively limited number of tests conducted. Therefore, there might be variability in the outcomes when applied to larger or different datasets.

Video Link: https://youtu.be/GQAlUqcSWV0