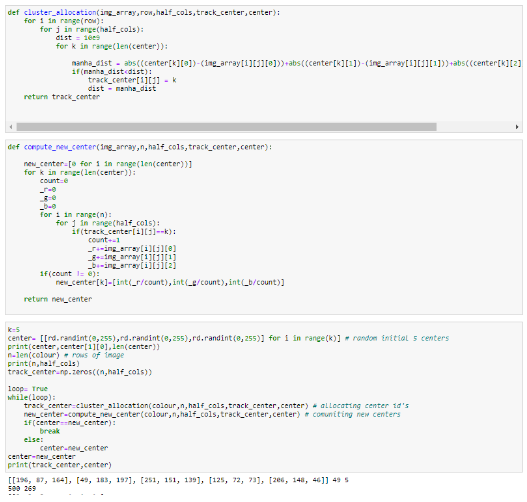
CS 520 Introduction to Artificial Intelligence
Professor Wes Cowan
TA: Aravind Sivaramakrishnan
Assignment 4: Colorization

Akash Patel Netid: adp178

December 2020

1 Basic Agent

The purpose of this project is to understand the concept of neural networks in Artificial Intelligence and explore colourization of the gray-scale image with KNN, Kmeans and neural network using gradient decent. Given a colourful image, we extract top five representative colours using Kmeans clustering and use these top 5 colours to re-color the left half of the image by replacing each pixel's true color with the representative color it was clustered with. We store the the labels for each pixel to map the representative colour in multi-dimensional matrix "track_center". Below is the implementation of Kmeans to get top 5 Representative colours:



```
def cluster_allocation(img_array,row,half_cols,track_center,center):
    for i in range(row):
        for j in range(half_cols):
            dist = 10e9
            for k in range(len(center)):

                manha_dist = abs((center[k][0])-(img_array[i][j][0]))+abs((center[k][1])-(img_array[i][j][1]))+abs((center[k][2]
                if(manha_dist<dist):
                    track_center[i][j] = k
                    dist = manha_dist
    return track_center
```

```
def compute_new_center(img_array,n,half_cols,track_center,center):

    new_center=[0 for i in range(len(center))]
    for k in range(len(center)):
        count=0
        _r=0
        _g=0
        _b=0
        for i in range(n):
            for j in range(half_cols):
                if(track_center[i][j]==k):
                    count+=1
                    _r+=img_array[i][j][0]
                    _g+=img_array[i][j][1]
                    _b+=img_array[i][j][2]
        if(count != 0):
            new_center[k]=[int(_r/count),int(_g/count),int(_b/count)]

    return new_center
```

```
k=5
center= [[rd.randint(0,255),rd.randint(0,255),rd.randint(0,255)] for i in range(k)] # random initial 5 centers
print(center,center[1][0],len(center))
n=len(colour) # rows of image
print(n,half_cols)
track_center=np.zeros((n,half_cols))

loop= True
while(loop):
    track_center=cluster_allocation(colour,n,half_cols,track_center,center) # allocating center id's
    new_center=compute_new_center(colour,n,half_cols,track_center,center) # comuniting new centers
    if(center==new_center):
        break
    else:
        center=new_center
center=new_center
print(track_center,center)
```

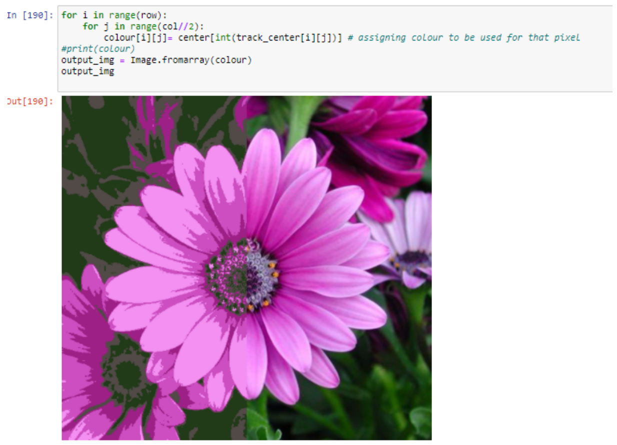
```
[[196, 87, 164], [49, 183, 197], [251, 151, 139], [125, 72, 73], [206, 148, 46]] 49 5
500 269
```

Figure 1: Kmeans

Algorithm:

1. At random select ‘k’ points.
2. Assign each data point to closest cluster.
3. Compute and place the new centroid of each cluster.
4. Reassign the data points to the new closest cluster. If any reassignment took place, repeat step 3 and if no change then the model is ready.

After processing each pixel with the algorithm cluster centroids would be the required dominant colors. We use the these 5 colours(centroids) in our case we got from the image and recolor the left half of the image using these 5 colours. Below is the out after recolouring left side of the image:

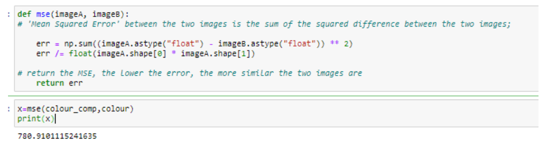


```
In [190]: for i in range(row):
              for j in range(col//2):
                  colour[i][j]= center[int(track_center[i][j])] # assigning colour to be used for that pixel
          #print(colour)
          output_img = Image.fromarray(colour)
          output_img
```

```
Out[190]:
```

Figure 2: Output Kmeans

As we can see that the output using K means is quiet good as we would expect it to be because we are extrating the dominant colours and using these to colour the left half of the image. But we also lose different hues of the colours as our left image is coloured with top 5 dominant colours(top 5 centroids).I found this visually satisfying as the texture and the colours are much closer to the original image. Now, numerically, we find mean squared error of the original image and the image generated using k means(note that right half remains the same, we will be using KNN on right half and compare again at the end of this section). Below is the MSE:



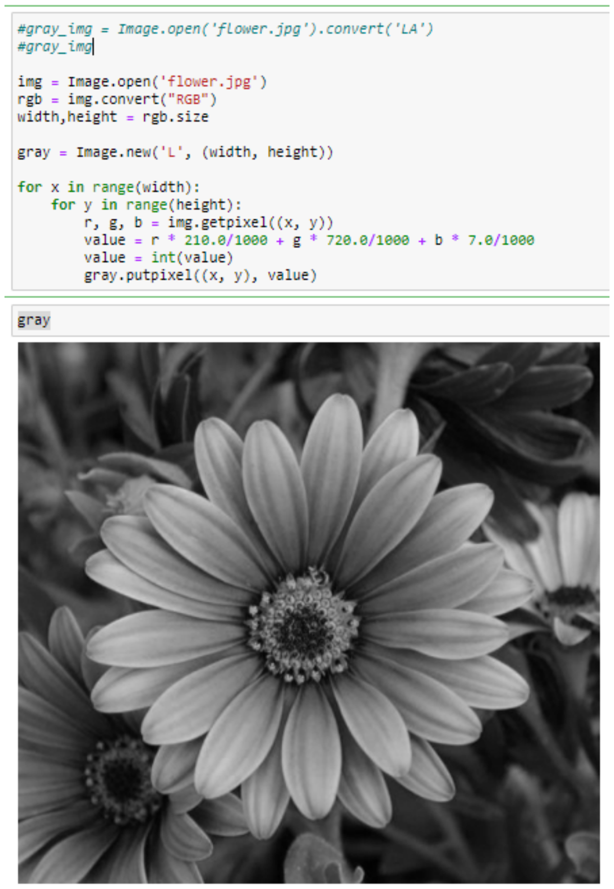
```
def mse(imageA, imageB):
# 'Mean Squared Error' between the two images is the sum of the squared difference between the two images;

    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])

# return the MSE, the lower the error, the more similar the two images are
    return err
```

```
x=mse(colour_comp,colour)
print(x)
```

```
780.9101115241635
```

Figure 3: MSE k-means

MSE 780.9101115241635 of K- means is reasonable while comparing the coloured and original image. Moving on-wards we convert the image into gray-scale using the given equation shown below in the image:



```
#gray_img = Image.open('flower.jpg').convert('LA')
#gray_img

img = Image.open('flower.jpg')
rgb = img.convert("RGB")
width,height = rgb.size

gray = Image.new('L', (width, height))

for x in range(width):
    for y in range(height):
        r, g, b = img.getpixel((x, y))
        value = r * 210.0/1000 + g * 720.0/1000 + b * 7.0/1000
        value = int(value)
        gray.putpixel((x, y), value)
```

```
gray
```

Figure 4: Converted Grayscale of the image

After converting into gray scale image we extract 3x3 gray-scale pixel patches of the right half of the image to be our testing data and find the six most similar 3x3 grayscale pixel patches in the left half gray-scale image(training data). For each of the selected patches in training data (left half of the image), we take the representative colour of the middle pixel (center pixel of 3x3 patch) from the recoloured half (which was done above using K-means).

We create a counter to keep track of the majority representative colour and take that colour to be the colour of the middle pixel in the test data patch. If there is no representative colour or a tie, we take first max value of the counter which is most similar to the test data and use that to colour the middle pixel. This way we generate coloring of the right half of the image.

The final output shown below is the original image, second image with left half colouring done using K-means and right half colouring done with the process describe above. Final output:

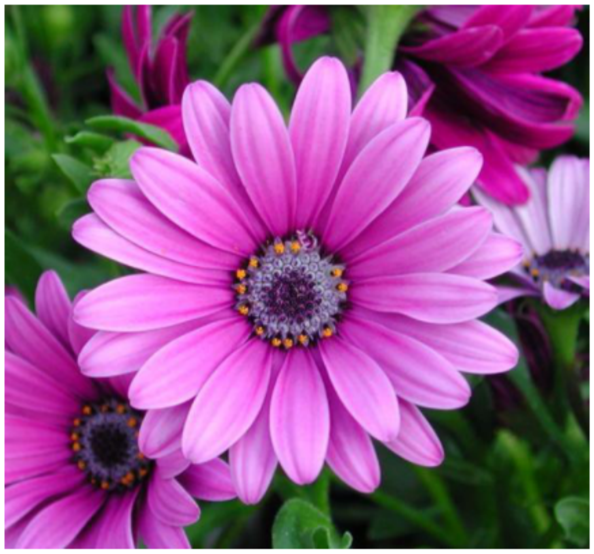


Figure 5: Original Image

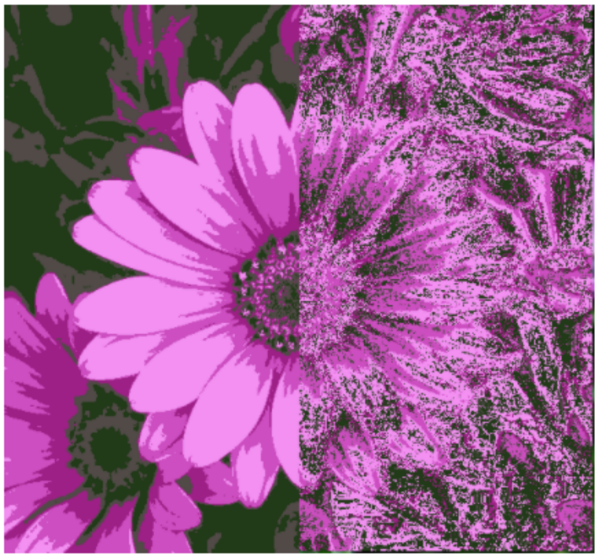
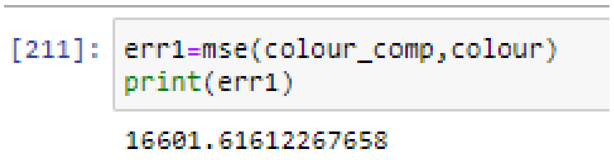


Figure 6: Final Output with left half of the image coloured using top 5 representative colours and right half using the above explained process.

Now we can see that right half of the image is not as good as left half done using K-means. The process we implemented to colour right half (kind of implementing KNN). This primarily because we found six most similar patches in left gray-scale image for the corresponding patch in right half and found the representative colour of those matches in left side of the image coloured using k-means. This itself is not a efficient way to colourise as there may be wide differences in colours in counterparts. That is to say using crude approach to colour the right side of the image.

So Finally we compare the MSE between the original image and the final output image (left half coloured with kmeans and right half using said process (similar to KNN). Below is the Final MSE:



```
[211]: err1=mse(colour_comp,colour)
       print(err1)

       16601.61612267658
```

Figure 7: MSE Final

We can see the MSE of 16601.6161226765 is almost reflective of the output we got compared to KMeans colouring and original Image. The error almost increases 25 times. Though in case of kmeans we kept the right half as same as original image. We also saw how we can measure quality of images using MSE. MSE being zero shows perfect similarity for the images.And a value greater than 1 implies less similarity and will keep increasing as average difference between pixels increase.

1.1 Bonus: Instead of doing a 5-nearest neighbor based color selection, what is the best number of representative colors to pick for your data? How did you arrive at that number?

Instead of using 5- nearest neighbour based colour selection, we can analyse small patches of pixels and keep the count of the colours in small ranges. Then we can plot histogram that collects counts of data organized into a set of predefined bins to keep track of different range of RGB values. This can be imagined as a matrix that contains information of an image (i.e. intensity in the range 0-255) and we can count this data in an organized way since we know that the range of information value for this case is 256 values, we can segment our range in sub parts (bins) and we can keep count of the number of pixels that fall in the range of each bin. Each bin can be interpreted as a single colour and use that to get the best representative k colours.

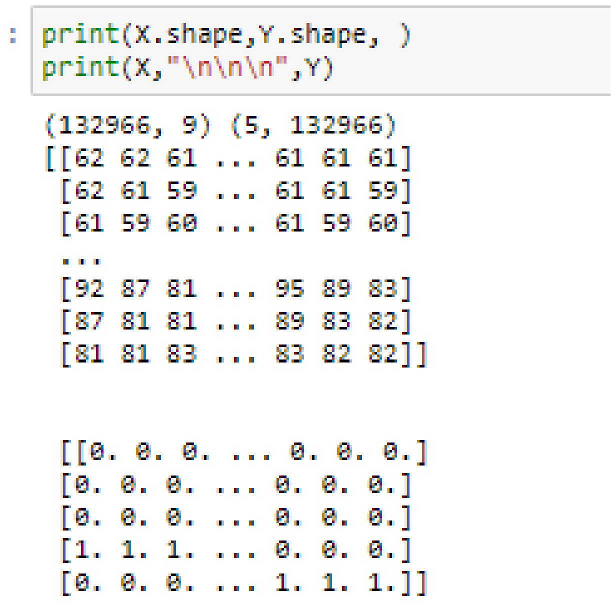
2 Improved Agent

Now we will be implementing a neural network using gradient descent method to come up with a model that can be used to colour the right half of the image(testing data).

2.1 A specification of your solution, including describing your input space, output space, model space, error or loss function, and learning algorithm. For instance, do you want to attempt this as a classification problem (as in the basic agent) or a regression problem, or some kind of soft classification?

2.1.1 Input space

The left side of the gray scaled image will be used as training data for the model. We will be extracting 3 by 3 patches of the left side image and flatten it. So the final input vector will have total elements equal to (number of rows-2) * (half the number of columns-2), -2 because we will be extracting 3 by 3 patches. Now each of these elements will be a vector of nine values (flattened 3x3 patch). This is labelled as X in the implementation. And Y will be true output vector of one flattened image patch, that if center value of 3 by 3 patch has been assigned cluster value, let's 2 by our k means then that means corresponding element in y will be a column vector with value [0,0,1,0,0]. Below is the figure showing X and Y and shapes of X and Y respectively.



```
: print(X.shape,Y.shape, )
  print(X,"\n\n\n",Y)
```

```
(132966, 9) (5, 132966)
[[62 62 61 ... 61 61 61]
 [62 61 59 ... 61 61 59]
 [61 59 60 ... 61 59 60]
 ...
 [92 87 81 ... 95 89 83]
 [87 81 81 ... 89 83 82]
 [81 81 83 ... 83 82 82]]


 [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 1. 1. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 1. 1.]]
```

Figure 8: Input Space for model

2.1.2 Model space

We will be implementing neural network with architecture of [9,20,15,5] where 9 will represent 9 element of our input model space and the last 5 will be out output layer with 5 nodes with corresponding probabilities for the the colour class. And there are two hidden layers with 20 and 15 nodes respectively. We will be using sigmoid function as our activation function and gradient descent method for optimising weights and bias for learning algorithm.

2.1.3 error or loss function

We are using cross entropy to measure the error or cost function and we will be minimizing it using gradient descent method. We will be implementing this as a soft classification problem as we will perform classification based on estimated probabilities.

2.2 How did you choose the parameters (structure, weights, any decisions that needed to be made) for your model?

The weight were randomly initialised and the architecture of my neural network is quiet flexible, we can have as many hidden layers we want. So decision of using 9,20,15,5 as my neural network architecture was after trying multiple architecture with varying nodes and hidden layers (I want up 6 hidden layers but found the model be over fitting).

2.3 Any pre-processing of the input data or output data that you used.

We have used both pre-processing of input data as well as out put data. Pre-processing of input data was as follows:

1. Get the left of half gray-scaled image as testing data, which will be processed further before inputting to the model.This input is referenced as X in implementation.

2. Extract 3x3 patches of the from the left half, flatten each 3x3 patch in to individual 9 size vectors.(shape will be N x 9)

3. Extract second input to model referenced as Y which will be true output vector of each flattened image patch.And transposing it to shape 5 X N as required by Neural network architecture.

Now when we have trained the model we will have the weight and bias list. We use these weights and biases in get_hypotheses function and input the testing data after performing steps 1 and 2 as explained above. From that we get A_list and in the last element of A_list will be N X 5 predicted probabilities for corresponding 5 centroid numbers. Than we will map these to actual values and colour the right side of the image.

2.4 How did you handle training your model? How did you avoid over-fitting?

The main aim of training is to minimize the overall loss of the network. It can be observed that cross entropy error/loss decreases over iterations, but we desire to minimize the perceptual loss as we need the images to be sensible rather than exactly matching the required result which is over-fitting our training data. We tune the number of iterations and learning rate to get a continuous accuracy and make it to a point of convergence. However, we do not want a converged neural network that only works for the training data. We want to generalize the learning to some extent. Given the conversion formula from a color image to gray-scale, a single gray-scale value maps to a raft of (r, g, b) values. As a result, the probability that the model would over-fit to the input data is minimal in a practical number of iterations and learning rate. For our model we tuned it to 500 iterations and learning rate of 0.000001.

2.5 An evaluation of the quality of your model compared to the basic agent. How can you quantify and qualify the differences between their performance? How can you make sure that the comparison is 'fair'?

Output image generated by neural network:

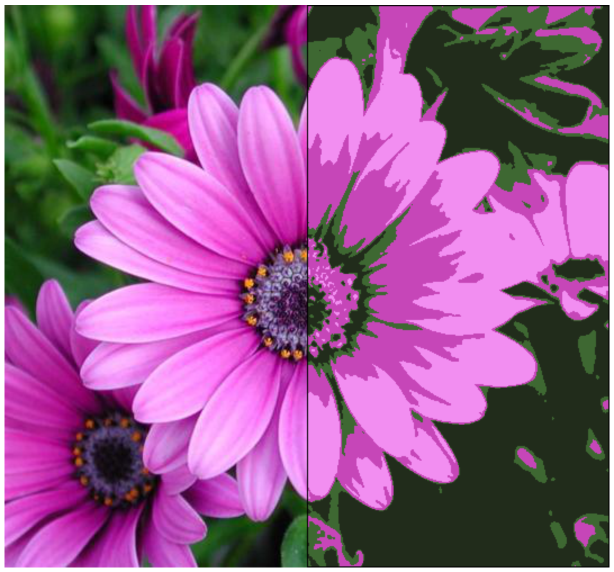
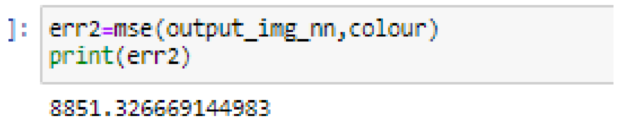


Figure 9: Output image for Neural Network

I compared the output images of the basic agent and neural network by computing mean squared error.

MSE for neural network:



```
]: err2=mse(output_img_nn,colour)
   print(err2)

   8851.326669144983
```

Figure 10: MSE for neural network

MSE for basic agent was 16601.61612267658 and for Neural network after comparing for testing output it is almost 50% less. We can say that comparison is relatively fair because MSE allows us to compare images given that size of the images are same, which is the case for us.For quantifying and qualifying the difference between the performances we analyse the MSE and the recreated visual output.

2.6 How might you improve your model with sufficient time, energy, and resources?

We can improve this model by using convoluted neural networks instead of conventional neural networks.The usage of CNNs are motivated by the fact that they can capture / are able to learn relevant features from an image at different levels. This feature learning is something that conventional neural networks cannot do this.CNN captures the spatial features from an image. Spatial features refer to the arrangement of pixels and the relationship between them in an image. Also, CNN is more efficient in terms of memory and complexity.

Implementing adaptive learning rate would also fasten the training and learning of the model and help to converge the optimization process.Adaptive learning rates can accelerate training and remove some of the pressure of choosing a learning rate and even help us to implement a learning rate schedule. Therefore, I believe implement CNN as well as adaptive learning rate can help us improve the model.