Text Query of Images

CASE STUDY REPORT

Submitted by

REG NO	NAME
CB.EN.U4CSE17021	GOPIKA NARAYANAN
CB.EN.U4CSE17051	SAI SUDHA PANIGRAHI
CB.EN.U4CSE17150	SAYANNAH P.B

in partial fulfilment of the requirements for the COURSE - 15CSE338(Computational Intelligence)

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641112

Table of Contents

1. Abstract

Content-based image retrieval, also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR), is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large databases.In the current era of digital communication, the use of digital images has increased for expressing, sharing and interpreting information. While working with digital images, quite often it is necessary to search for a specific image for a particular situation based on the visual contents of the image. This task looks easy if you are dealing with tens of images but it gets more difficult when the number of images goes from tens to hundreds and thousands, and the same content-based searching task becomes extremely complex when the number of images is in the millions. To deal with the situation, some intelligent way of content-based searching is required to fulfill the searching request with right visual contents in a reasonable amount of time. There are some really smart techniques proposed by researchers for efficient and robust content-based image retrieval. In this research, the aim is to highlight the efforts of researchers who conducted some brilliant work and to provide a proof of concept for intelligent content-based image retrieval techniques.

2. Objective / Problem Definition

To use the technology of computational intelligence to search and display the images based on the text query given by the user. We propose to investigate efficient methods apart from the traditional methods for easy access and convenience of images in a database. Some of day to day applications are Google image search, e-commerce websites that search for products and simple querying of images from large databases.

3. Introduction

Database technologies for pictorial applications were discussed for the first time in that era and the researchers got attraction for this domain since then. Former image retrieval techniques were not that intelligent and sophisticated and they were not able to search for images based on its visual features instead those techniques were based on text-based metadata of images. All images stored in the database were first tagged with the metadata and then images were searched based on the image metadata. Text-based image retrieval methods were used for conventional database applications. They were used with a lot of business applications and purposes but increasing usage and volume of digital images created performance and accuracy issues for text-based image retrieval methods. New methods proposed for image retrieval considered color, texture, and shapes of objects in an image.

4. Literature Survey

a)
G. Raju, Madhu S. Nair,
A fast and efficient color image enhancement method based on fuzzy-logic and histogram,
AEU - International Journal of Electronics and Communications,
Volume 68, Issue 3,
2014

A new fuzzy logic and histogram based algorithm for enhancing low contrast color images has been proposed here. The method is computationally fast compared to conventional and other advanced enhancement techniques. It is based on two important parameters *M* and *K*, where *M* is the average intensity value of the image, calculated from the histogram and *K* is the contrast intensification parameter. The given RGB image is converted into HSV color space to preserve the chromatic information contained in the original image. To enhance the image, only the V component is stretched under the control of the parameters *M* and *K*. The proposed method has been compared with conventional contrast enhancement techniques as well as with advanced algorithms. All the above techniques were based on the principle of transforming the skewed histogram of the original image into a uniform histogram.The performance of the different contrast enhancement algorithms are evaluated based on the visual quality,and the computational time.The inter comparison of different techniques was carried out on different low contrast color images. Based on the performance analysis, we advocate that our proposed Fuzzy Logic method is well suited for contrast enhancement of low contrast color images.

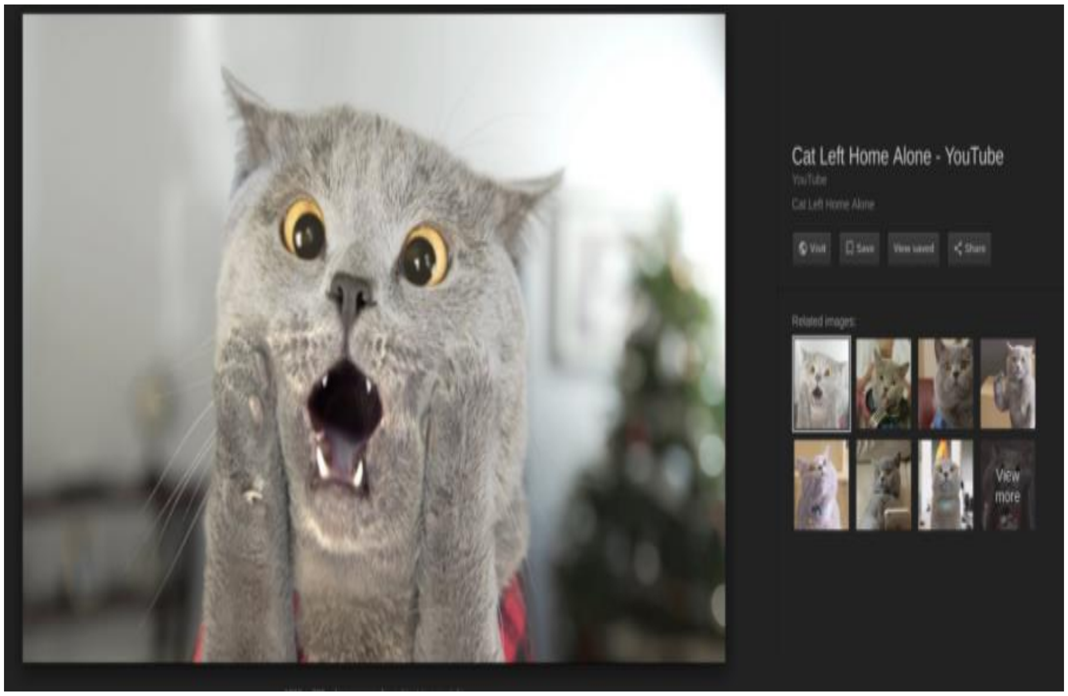
b)
Zhou M., Tanimura Y., Nakada H. (2020) One-Shot Learning Using Triplet Network with kNN Classifier. In: Ohsawa Y. et al. (eds) Advances in Artificial Intelligence. JSAI 2019. Advances in Intelligent Systems and Computing, vol 1128. Springer, Cham

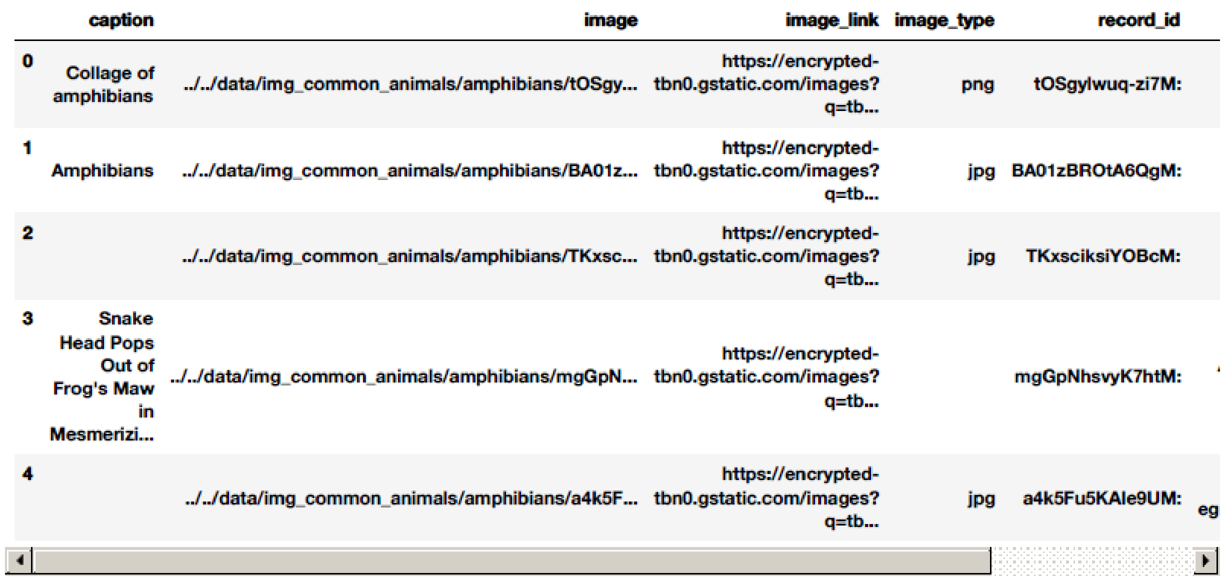
One-shot Learning using Triplet Network with kNN classifier (2019)
In this paperwork, they described a Triplet Network model, inspired by the Siamese Network based on distance metric, which can be used for one-shot learning. They proposed a triplet network with a kNN classifier for the problem of one-shot learning, in which they predict the query images by giving a single example of each class. This triplet network learns a mapping from sample images to the Euclidean space.For this,they used the embeddings of training points trained on the kNN classifier and predicted the label with the embedding of testing points by the classifier. A significant improvement can be obtained by the effectiveness of data augmentation. Of the 3 approaches tested, best results were achieved by augmenting the initial dataset with Triplet Network model. Input triplets for Triplet Network were generated in two ways. One kind of triplets was produced by the augmented dataset, while another one was created by the initial dataset

which was not augmented. For the first type, one sample was selected(used as the anchor instance) from the dataset, then chose another one (used as the positive instance) from the same label. Then the other sample was randomly obtained (used as the negative instance) from any other label. Finally, they concatenated them as a triplet pair. However, for the other type created by the initial dataset, they used the same image as the positive instance to overcome the limitation of lack of samples.

5. <u>Dataset Description</u>

The dataset consists of google images of around 50 animals. Each animal set contains roughly 500 images. Data captured are the image, title, and caption for the image.





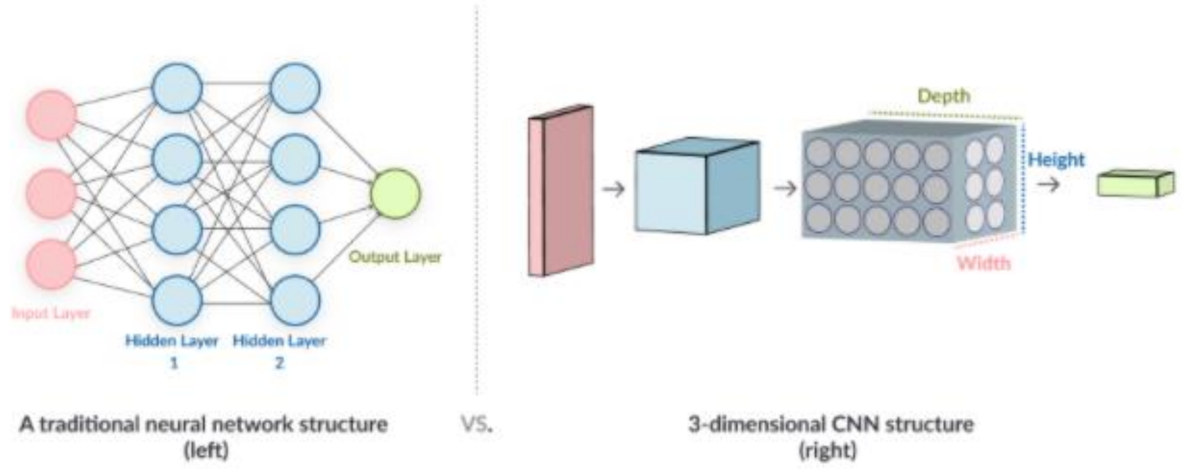
	caption	image	image_link	image_type	record_id	
0	Collage of amphibians	../../data/img_common_animals/amphibians/tOSgy...	https://encrypted-tbn0.gstatic.com/images?q=tb...	png	tOSgylwuq-zi7M:	
1	Amphibians	../../data/img_common_animals/amphibians/BA01z...	https://encrypted-tbn0.gstatic.com/images?q=tb...	jpg	BA01zBROtA6QgM:	
2		../../data/img_common_animals/amphibians/TKxsc...	https://encrypted-tbn0.gstatic.com/images?q=tb...	jpg	TKxsciksiYOBcM:	
3	Snake Head Pops Out of Frog's Maw in Mesmerizi...	../../data/img_common_animals/amphibians/mgGpN...	https://encrypted-tbn0.gstatic.com/images?q=tb...		mgGpNhsvyK7htM:	
4		../../data/img_common_animals/amphibians/a4k5F...	https://encrypted-tbn0.gstatic.com/images?q=tb...	jpg	a4k5Fu5KAIe9UM:	eg

6. Tools Description

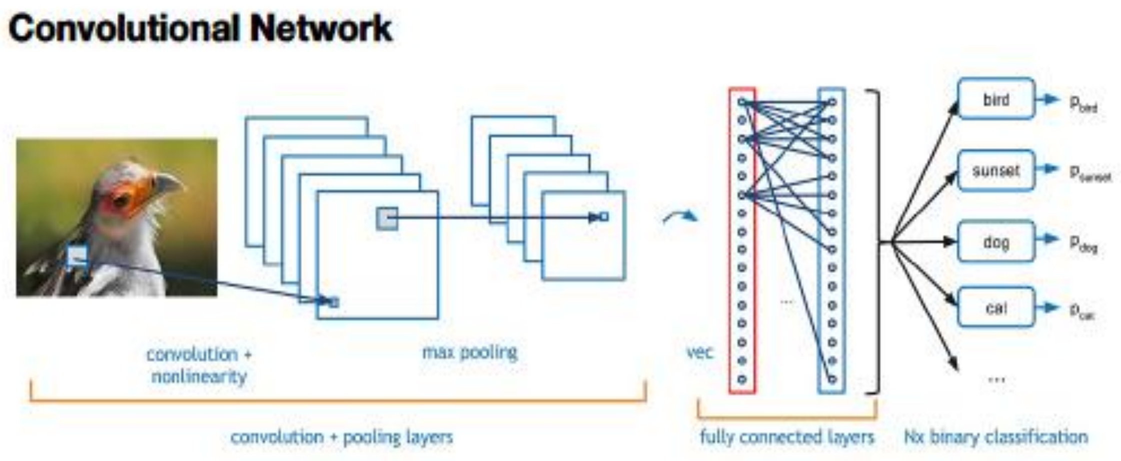
- Packages used: Keras, PIL, Image, Sklearn, Numpy, Pandas, Tensorflow
- Model used:VGG16, Fuzzy algorithms, Genetic algorithms
- Activation function used: LeakyRelu,SoftMax, Relu
- Language: Python 3

7. Architecture Diagram/CI Techniques Model Details

7.a) Neural Networks


Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer
Depth
Height
Width
A traditional neural network structure (left)
VS.
3-dimensional CNN structure (right)

Unlike a fully connected neural network, in a Convolutional Neural Network (CNN) the neurons in one layer don't connect to all the neurons in the next layer. Rather, a convolutional neural network uses a three-dimensional structure, where each set of neurons analyzes a specific region or "feature" of the image. CNNs filter connections by proximity (pixels are only analyzed in relation to pixels nearby), making the training process computationally achievable. In a CNN each group of neurons focuses on one part of the image. For example, in a cat image, one group of neurons might identify the head, another the body, another the tail, etc. There may be several stages of segmentation in which the neural network image recognition algorithm analyzes smaller parts of the images, for example, within the head, the cat's nose, whiskers, ears, etc. The final output is a vector of probabilities, which predicts, for each feature in the image, how likely it is to belong to a class or category.



7.b) Fuzzy Logic

Overall Pipeline

Description:

- Convert input image from RGB to CIELAB, progress on L channel
- Calculate the average pixel intensity - M value
- **Fuzzification:** For each pixel, calculate degree of membership of each class based

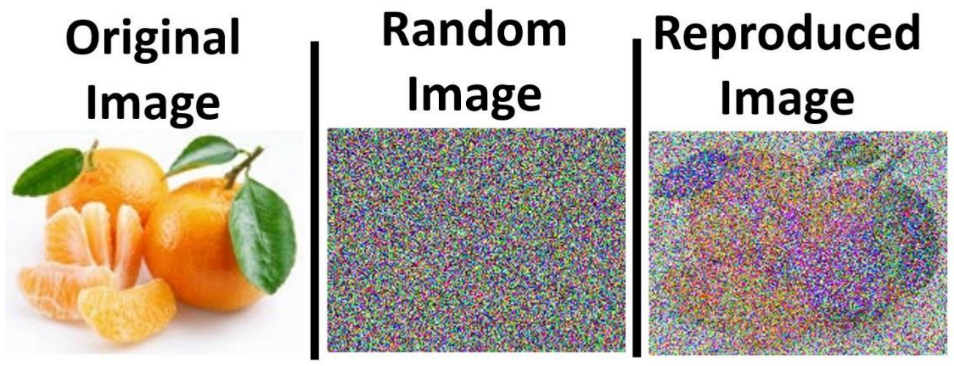
  on pixel intensity and M value.
- **Inference:** Calculate the output fuzzy set from the input pixel intensity based on the proposed rule set
- **Defuzzification:** For each pixel, calculate centroid value of its output fuzzy

  set.
- Normalize output pixel intensity from [-50, 305] to [0, 255]
- Merge modified L channel to the original AB channels, convert output image from CIELAB to RGB.

7.c) Evolutionary Computing

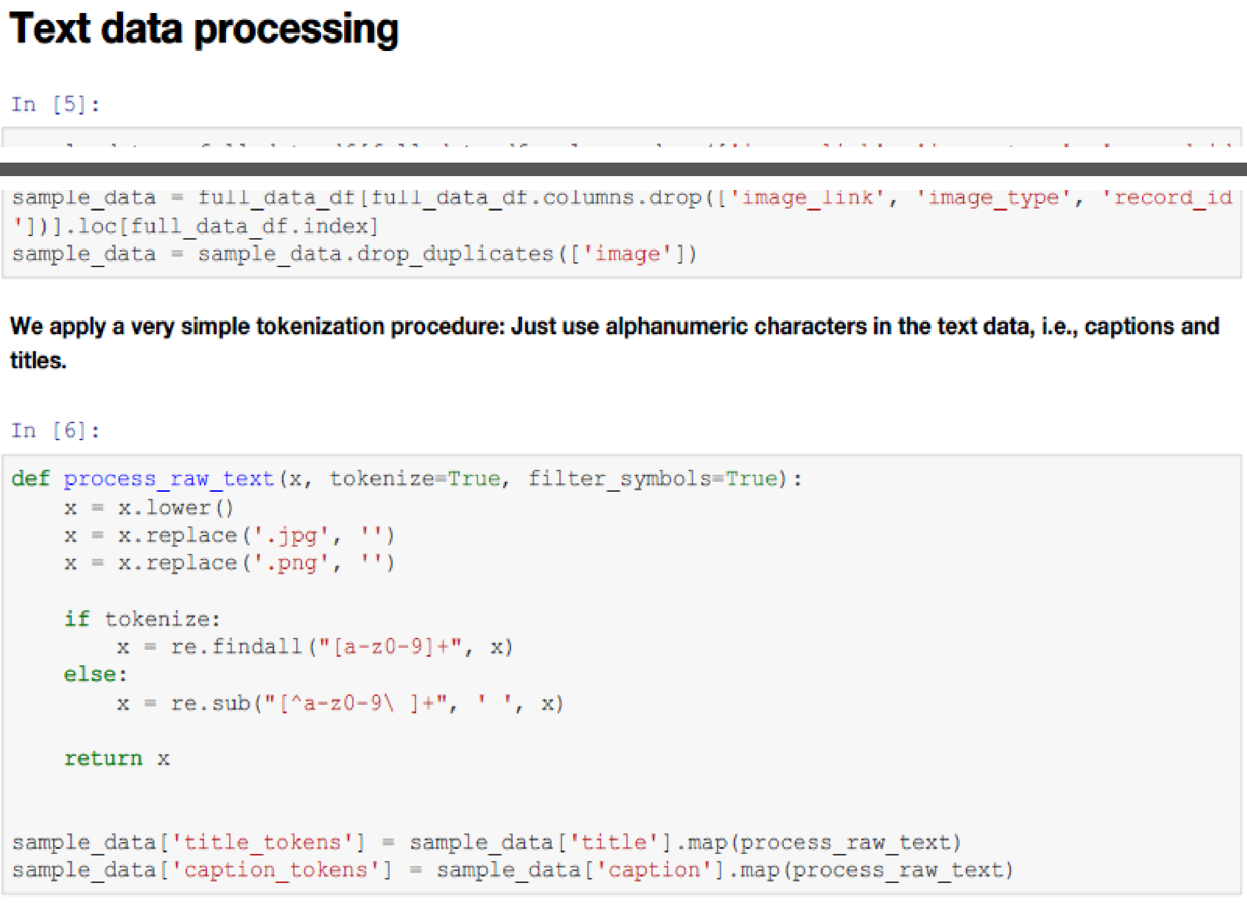
Genetic Algorithm for Reproducing Images

The model project accepts an image as input. This image can have one or more channels (i.e. the image could be binary, gray, or color, such as RGB). RGB is the most popular color model that produces any color as a combination of the 3 color channels Red, Green, and Blue. Hence its abbreviation. The genetic algorithm (GA) starts from a randomly generated image of the same shape as the input image. This randomly generated image is evolved, using crossover and mutation, using GA until it reproduces an image similar to the original one. The exact original image might not be accurately reproduced, but at least a similar image will be generated. The next figure shows that image and how it’s reproduced after 15,000 generations.



8. Implementation and Results

a) Data Preprocessing



Text data processing

In [5]:

```
sample_data = full_data_df[full_data_df.columns.drop(['image_link', 'image_type', 'record_id
'])].loc[full_data_df.index]
sample_data = sample_data.drop_duplicates(['image'])
```

We apply a very simple tokenization procedure: Just use alphanumeric characters in the text data, i.e., captions and titles.

In [6]:

```
def process_raw_text(x, tokenize=True, filter_symbols=True):
    x = x.lower()
    x = x.replace('.jpg', '')
    x = x.replace('.png', '')

    if tokenize:
        x = re.findall("[a-z0-9]+", x)
    else:
        x = re.sub("[^a-z0-9\ ]+", ' ', x)

    return x


sample_data['title_tokens'] = sample_data['title'].map(process_raw_text)
sample_data['caption_tokens'] = sample_data['caption'].map(process_raw_text)
```

Let's apply Jaccard measure to know if the title is almost the same as the caption. We need to do this to avoid redundancy in the text data.

In [7]:

```
def jaccard_similarity(row):
    intersection = len(set(row['title_tokens']).intersection(row['caption_tokens']))
    union = len(set(row['title_tokens']).union(row['caption_tokens']))

    similarity = 1.0 * intersection / union

    return similarity


sample_data['jaccard_sim'] = sample_data.apply(jaccard_similarity, axis=1)
```

By using the similarity score, we select which text data are used. If the title and the caption are dissimilar based on a threshold, then we use both text (concatenated) as input.

In [8]:

```
def get_effective_sample_text(row, sim_thresh=0.5):
    token_set = []
    title_tokens = row['title_tokens']
    caption_tokens = row['caption_tokens']

    if row['jaccard_sim'] > sim_thresh:
        if len(title_tokens) > len(caption_tokens):
            token_set.extend(title_tokens)
        else:
            token_set.extend(caption_tokens)
    else:
        token_set.extend(title_tokens)
        token_set.extend(caption_tokens)

    return token_set


sample_data['tokens'] = sample_data.apply(get_effective_sample_text, axis=1)
sample_data = sample_data.drop(['caption', 'title', 'title_tokens', 'caption_tokens', 'jacca
rd_sim'], axis=1)
```

After taking the relevant text data, we retokenize them with a `Tokenizer` utility class from the keras preprocessing module.

Note that the `tokens` data is already a list of words, so we need to join each token together before fitting the tokenizer.

In [9]:

```
tokenizer = Tokenizer(num_words=10000, filters='', lower=True, split=' ', char_level=False,
oov_token=None)
tokenizer.fit_on_texts(sample_data['tokens'].map(lambda x: ' '.join(x)))
```

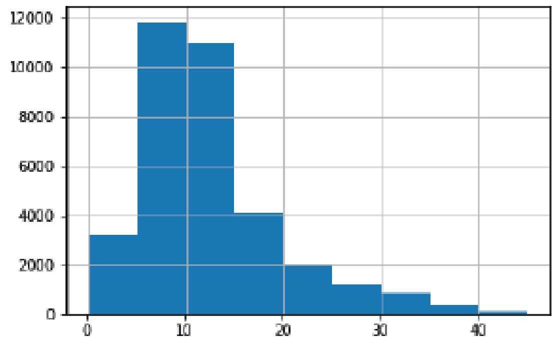
We take the histogram of token length for each item in our dataset. With this, we can have an informed choice for our expected sequence length when standardizing the length of the text input.

In [10]:

```
pd.Series([len(i) for i in tokenizer.texts_to_sequences(
    sample_data['tokens'].map(lambda x: ' '.join(x))
)]).hist(bins=range(0, 50, 5))
```

Out[10]:

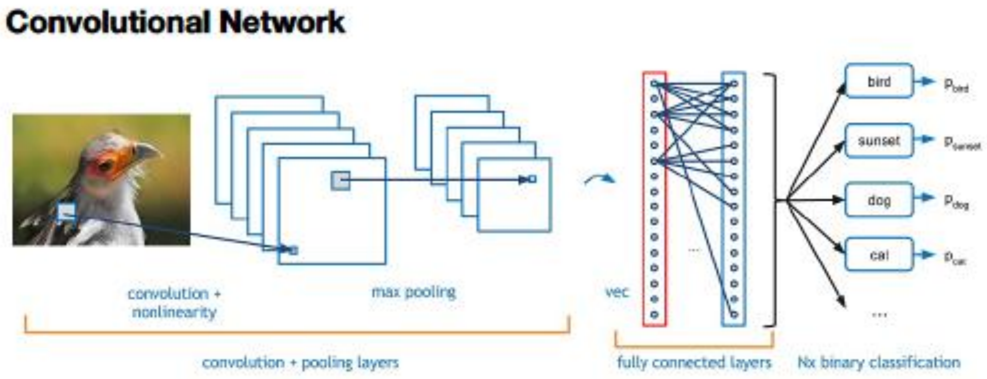
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1e8c085e90>
```



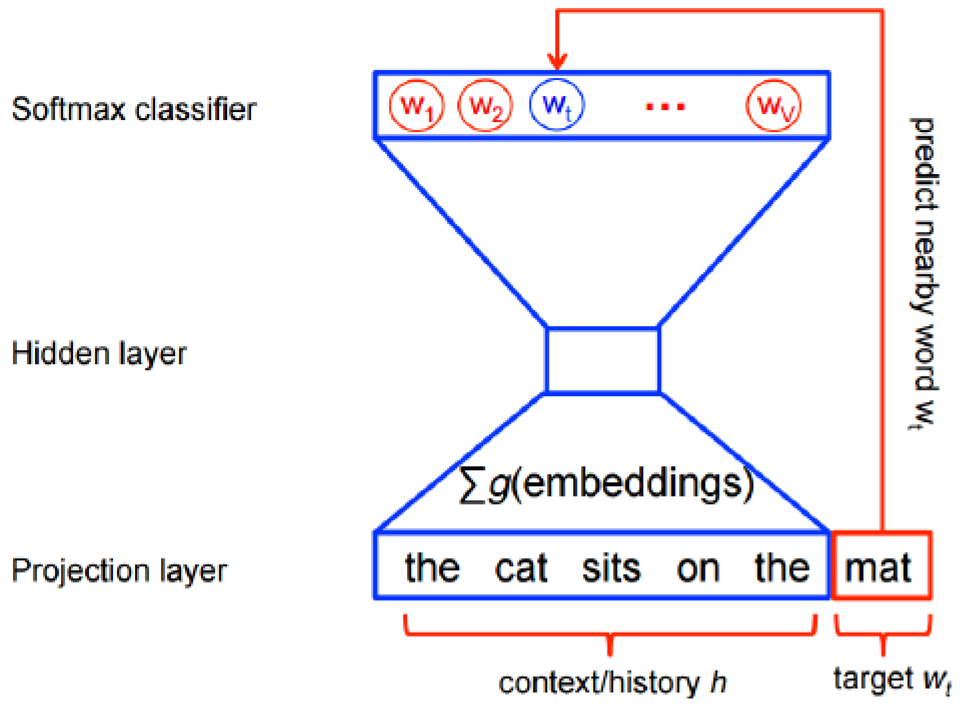
In [11]:

```
MAXLEN = 15
EMBEDDING_DIM = 300
```

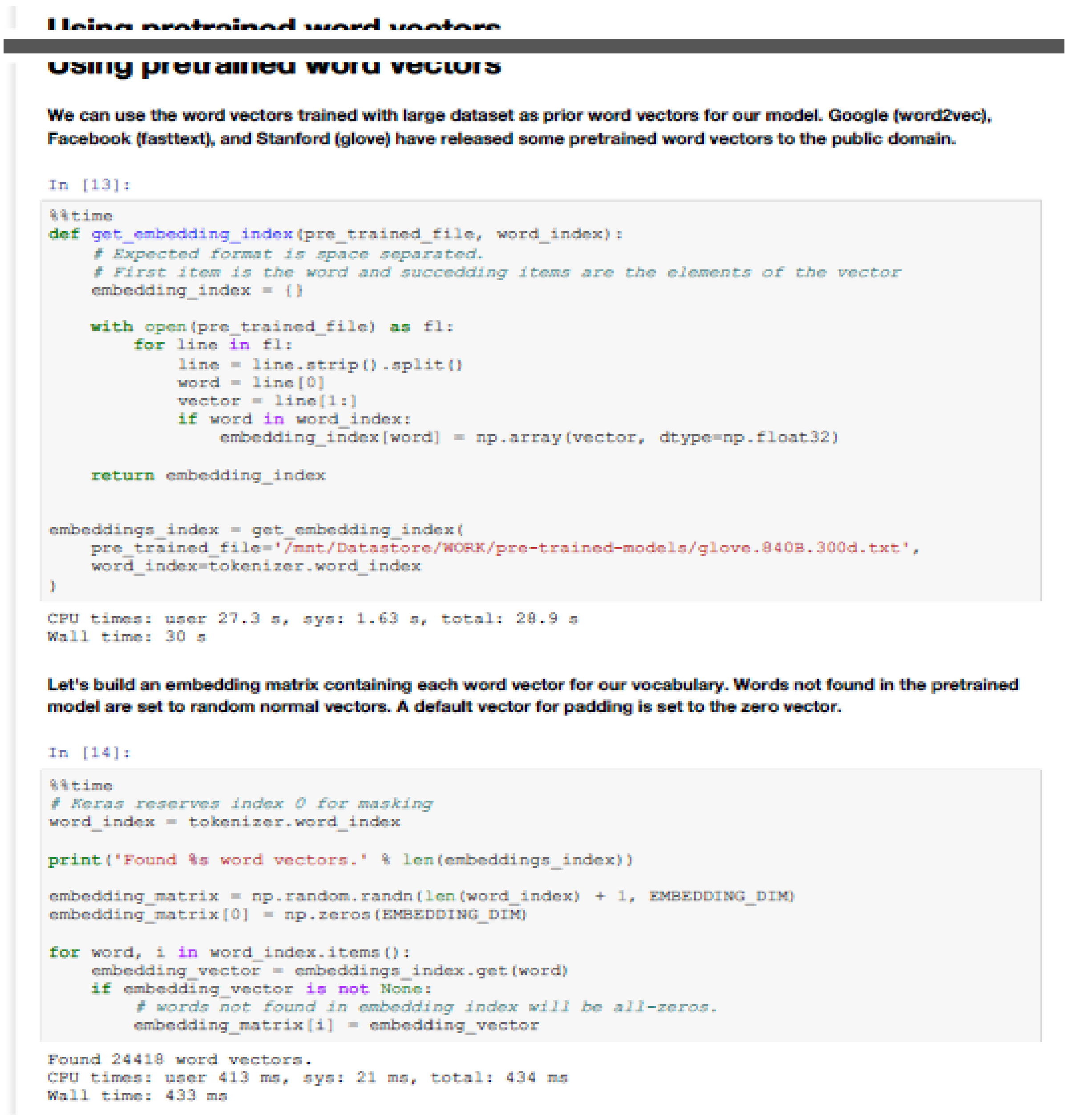
b) Architectural Diagram



Learning a word vector representation



c)EDA



Using pretrained word vectors

We can use the word vectors trained with large dataset as prior word vectors for our model. Google (word2vec), Facebook (fasttext), and Stanford (glove) have released some pretrained word vectors to the public domain.

In [13]:

```
%%time
def get_embedding_index(pre_trained_file, word_index):
    # Expected format is space separated.
    # First item is the word and succedding items are the elements of the vector
    embedding_index = {}

    with open(pre_trained_file) as fl:
        for line in fl:
            line = line.strip().split()
            word = line[0]
            vector = line[1:]
            if word in word_index:
                embedding_index[word] = np.array(vector, dtype=np.float32)

    return embedding_index


embeddings_index = get_embedding_index(
    pre_trained_file='/mnt/Datastore/WORK/pre-trained-models/glove.840B.300d.txt',
    word_index=tokenizer.word_index
)
```

```
CPU times: user 27.3 s, sys: 1.63 s, total: 28.9 s
Wall time: 30 s
```

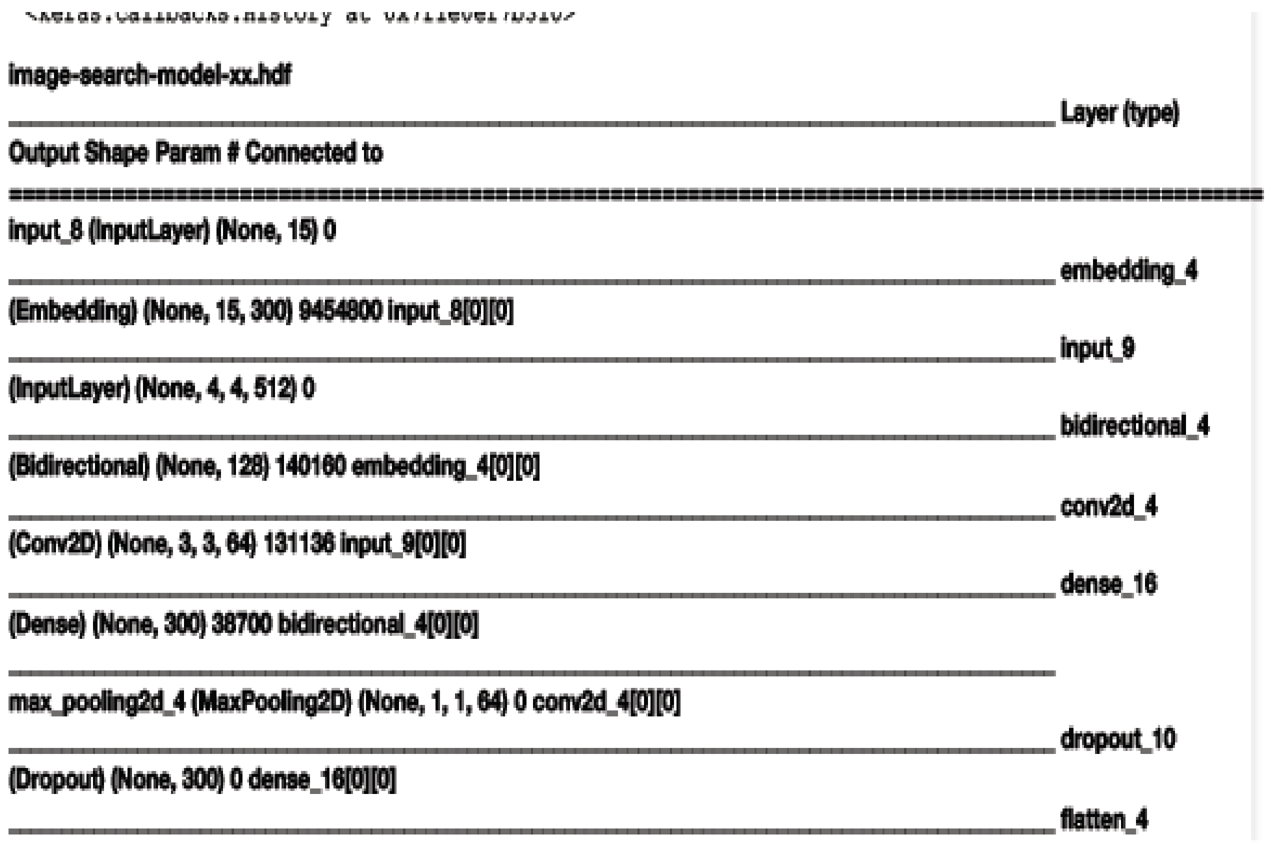
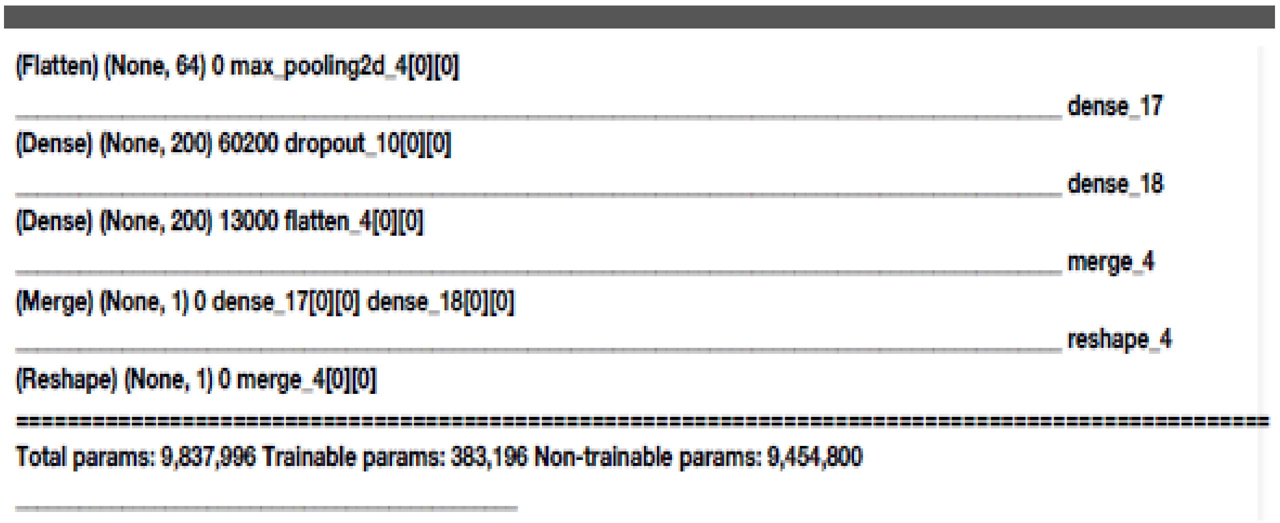
Let's build an embedding matrix containing each word vector for our vocabulary. Words not found in the pretrained model are set to random normal vectors. A default vector for padding is set to the zero vector.

In [14]:

```
%%time
# Keras reserves index 0 for masking
word_index = tokenizer.word_index

print('Found %s word vectors.' % len(embeddings_index))

embedding_matrix = np.random.randn(len(word_index) + 1, EMBEDDING_DIM)
embedding_matrix[0] = np.zeros(EMBEDDING_DIM)

for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

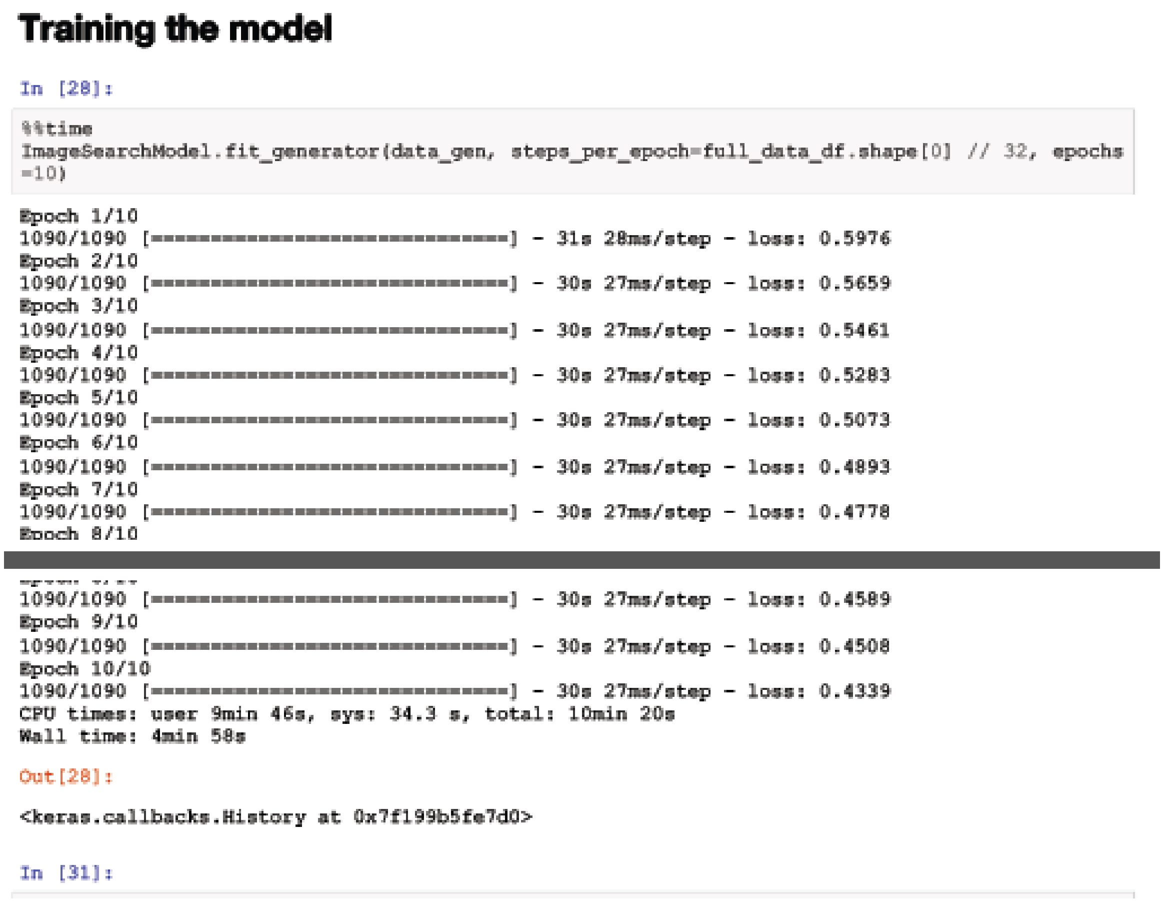
```
Found 24418 word vectors.
CPU times: user 413 ms, sys: 21 ms, total: 434 ms
Wall time: 433 ms
```

d)Model Definition

```
<keras.callbacks.History at 0x7[illegible]>

image-search-model-xx.hdf
________________________________________________________________________________ Layer (type)
Output Shape Param # Connected to
==========================================================================================
input_8 (InputLayer) (None, 15) 0
________________________________________________________________________________ embedding_4
(Embedding) (None, 15, 300) 9454800 input_8[0][0]
________________________________________________________________________________ input_9
(InputLayer) (None, 4, 4, 512) 0
________________________________________________________________________________ bidirectional_4
(Bidirectional) (None, 128) 140160 embedding_4[0][0]
________________________________________________________________________________ conv2d_4
(Conv2D) (None, 3, 3, 64) 131136 input_9[0][0]
________________________________________________________________________________ dense_16
(Dense) (None, 300) 38700 bidirectional_4[0][0]
________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D) (None, 1, 1, 64) 0 conv2d_4[0][0]
________________________________________________________________________________ dropout_10
(Dropout) (None, 300) 0 dense_16[0][0]
________________________________________________________________________________ flatten_4

(Flatten) (None, 64) 0 max_pooling2d_4[0][0]
________________________________________________________________________________ dense_17
(Dense) (None, 200) 60200 dropout_10[0][0]
________________________________________________________________________________ dense_18
(Dense) (None, 200) 13000 flatten_4[0][0]
________________________________________________________________________________ merge_4
(Merge) (None, 1) 0 dense_17[0][0] dense_18[0][0]
________________________________________________________________________________ reshape_4
(Reshape) (None, 1) 0 merge_4[0][0]
==========================================================================================
Total params: 9,837,996 Trainable params: 383,196 Non-trainable params: 9,454,800

________________________________
```

d) Model Performance



Training the model

In [28]:

```
%%time
ImageSearchModel.fit_generator(data_gen, steps_per_epoch=full_data_df.shape[0] // 32, epochs
=10)
```

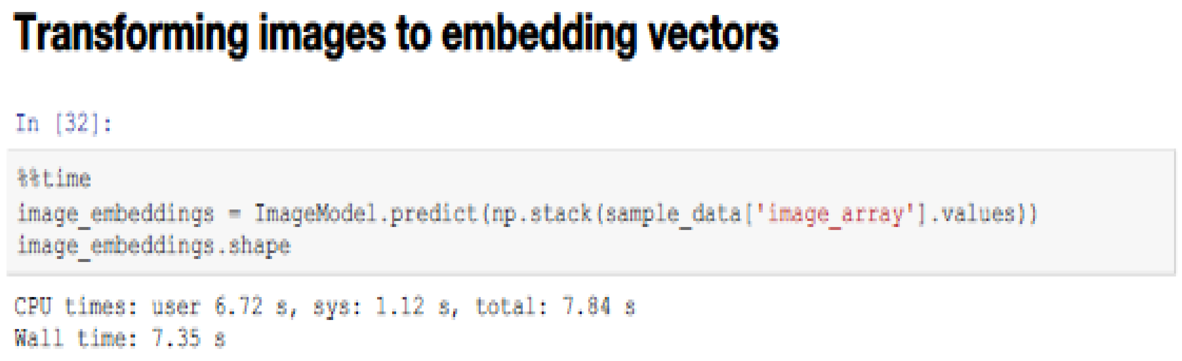
```
Epoch 1/10
1090/1090 [==============================] - 31s 28ms/step - loss: 0.5976
Epoch 2/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.5659
Epoch 3/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.5461
Epoch 4/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.5283
Epoch 5/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.5073
Epoch 6/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.4893
Epoch 7/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.4778
Epoch 8/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.4589
Epoch 9/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.4508
Epoch 10/10
1090/1090 [==============================] - 30s 27ms/step - loss: 0.4339
CPU times: user 9min 46s, sys: 34.3 s, total: 10min 20s
Wall time: 4min 58s
```

Out[28]:

```
<keras.callbacks.History at 0x7f199b5fe7d0>
```

In [31]:

e) Rescaling and Normalization



Transforming images to embedding vectors

In [32]:

```
%%time
image_embeddings = ImageModel.predict(np.stack(sample_data['image_array'].values))
image_embeddings.shape
```

```
CPU times: user 6.72 s, sys: 1.12 s, total: 7.84 s
Wall time: 7.35 s
```

In [55]:

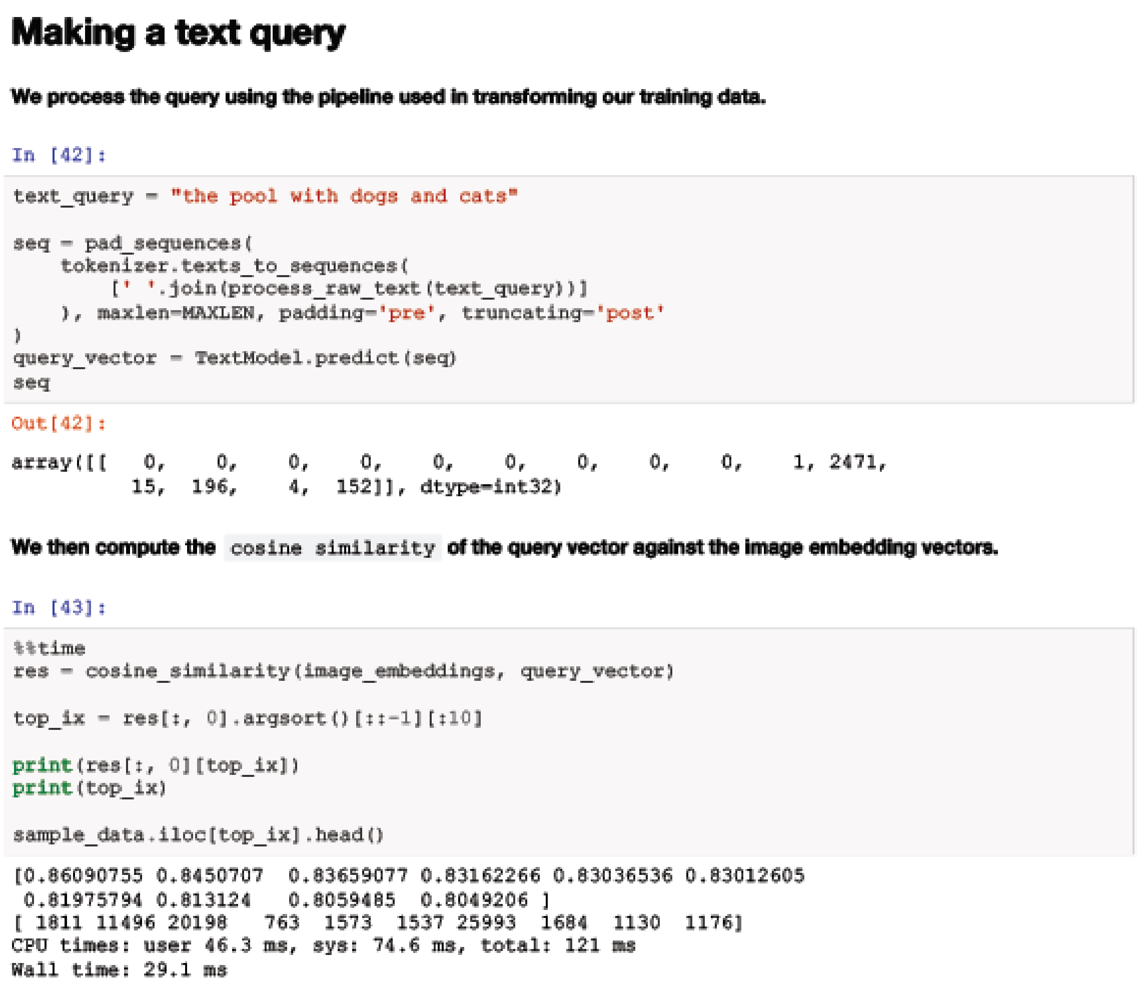
```
image_query = ImageModel.predict(np.stack(sample_data.iloc[[iloc]].image_array.values))
res_im = cosine_similarity(image_embeddings, image_query)

top_im_ix = res_im[:, 0].argsort()[::-1][:10]

print(res_im[:, 0][top_im_ix])
print(top_im_ix)

sample_data.iloc[top_im_ix].head()
```

```
[0.99999964 0.98442847 0.98420405 0.97683805 0.9685759  0.96544635
 0.96136504 0.95940375 0.9588451  0.95852727]
[20194 20335 18018  6810 20234 20266 20187 20211 18570 20505]
```

Out[55]:

	image	tokens	sequence	empty_sequence	image_array
20463	../../data/img_common_animals/red-fox/I4D7[illegible]	[red, fox, wildlife, sno[illegible] photo, on, pi...	[20, 32, 27, 133, 35, [illegible] 20, 32, 27, ...	[illegible]	[[[0.122547864914, [illegible] 0.0539694946...
20604	../../data/img_common_animals/red-fox/zuEcpM5B...	[red, fox, elmwood, park, zoo, elmwood, park, ...	[0, 0, 0, 20, 32, 8676, 98, 34, 8676, 98, 34, ...	False	[[[0.130790114403, 0.0, 0.0, 0.0, 0.1832721084...
18279	../../data/img_common_animals/coyote/Xh_8alpCl...	[coyote, spirit, animal, totem, meaning]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 62, 481, 24, 10...	False	[[[0.228968769312, 0.0, 0.0, 0.0, 0.0927995665...
6905	../../data/img_common_animals/gray-wolf/exc524...	[great, lakes, gray, wolf]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 135, 2989, 7...	False	[[[0.0, 0.0, 0.312162518501, 0.019131347537, 0...
20603	../../data/img_common_animals/red-fox/ValbH3lt...	[red, fox, adoption, kit, canadian, wildlife, ...	[0, 20, 32, 671, 893, 308, 27, 2134, 1470, 543...	False	[[[0.106955885887, 0.0, 0.0, 0.0, 0.2613540887...

g) Prediction



Making a text query

We process the query using the pipeline used in transforming our training data.

In [42]:

```
text_query = "the pool with dogs and cats"

seq = pad_sequences(
    tokenizer.texts_to_sequences(
        [' '.join(process_raw_text(text_query))]
    ), maxlen=MAXLEN, padding='pre', truncating='post'
)
query_vector = TextModel.predict(seq)
seq
```

Out[42]:

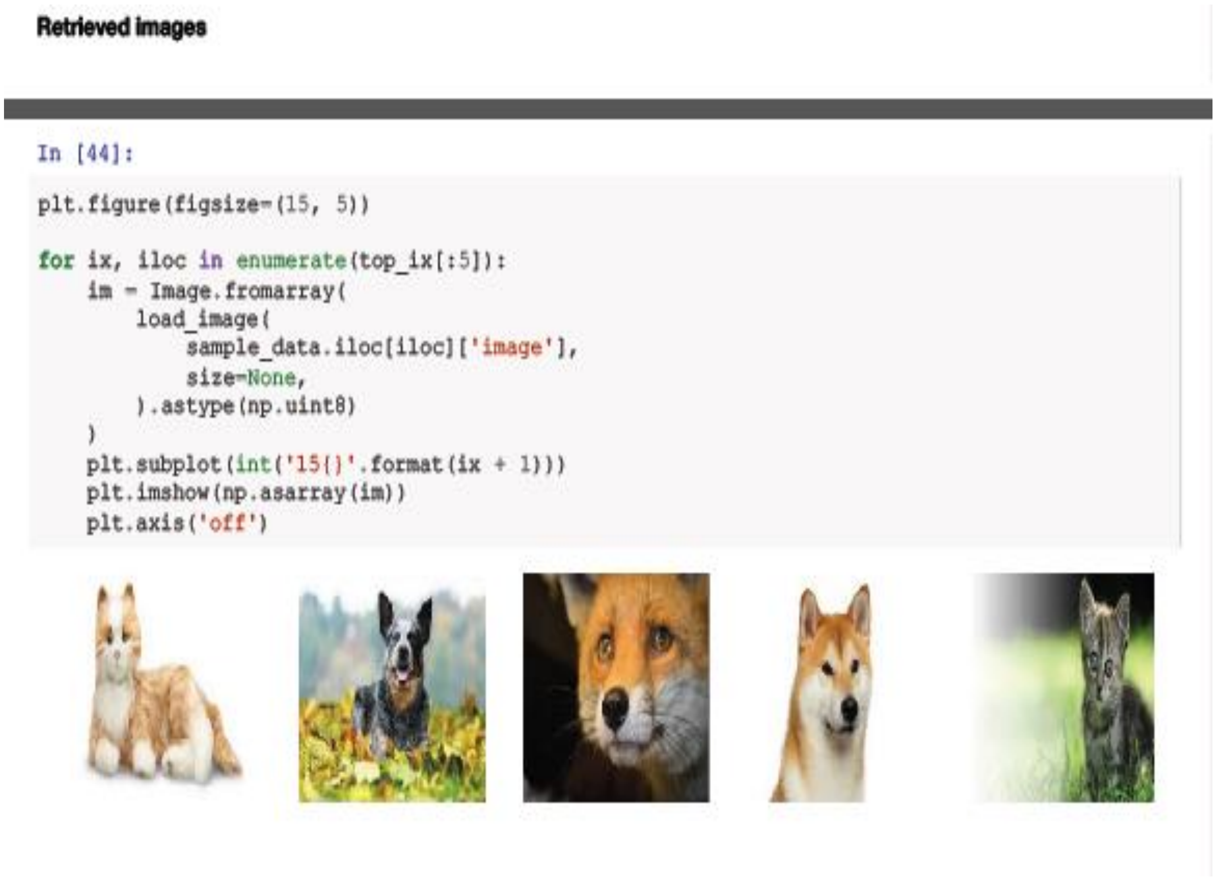
```
array([[   0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 2471,
          15,  196,    4,  152]], dtype=int32)
```

We then compute the `cosine similarity` of the query vector against the image embedding vectors.

In [43]:

```
%%time
res = cosine_similarity(image_embeddings, query_vector)

top_ix = res[:, 0].argsort()[::-1][:10]

print(res[:, 0][top_ix])
print(top_ix)

sample_data.iloc[top_ix].head()
```

```
[0.86090755 0.8450707  0.83659077 0.83162266 0.83036536 0.83012605
 0.81975794 0.813124   0.8059485  0.8049206 ]
[ 1811 11496 20198   763  1573  1537 25993  1684  1130  1176]
CPU times: user 46.3 ms, sys: 74.6 ms, total: 121 ms
Wall time: 29.1 ms
```



Retrieved images

In [44]:

```
plt.figure(figsize=(15, 5))

for ix, iloc in enumerate(top_ix[:5]):
    im = Image.fromarray(
        load_image(
            sample_data.iloc[iloc]['image'],
            size=None,
        ).astype(np.uint8)
    )
    plt.subplot(int('15{}'.format(ix + 1)))
    plt.imshow(np.asarray(im))
    plt.axis('off')
```

9. Method-2: Triple Neural Network

A)Data Preprocessing using Fuzzy algorithm

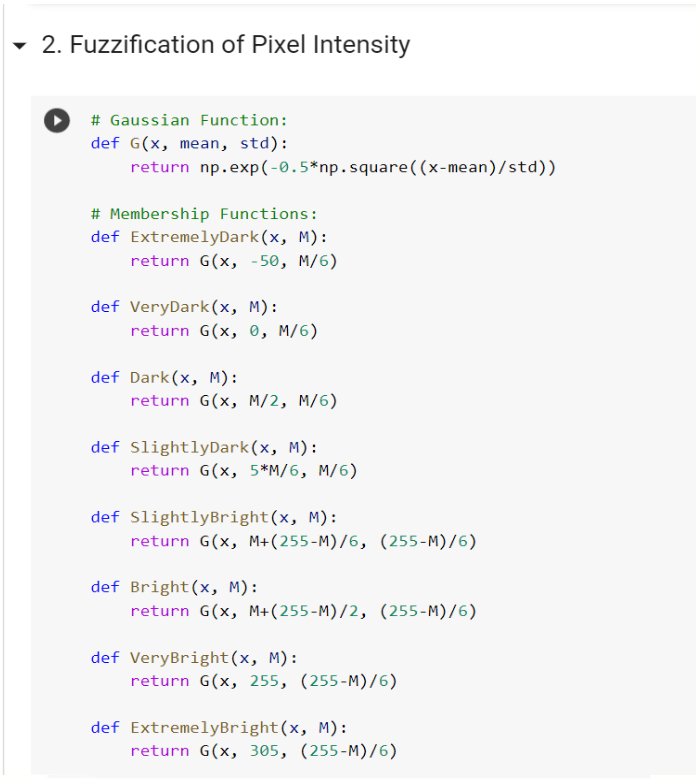
Image enhancement is a method of improving the quality of an image and contrast is a major aspect. Traditional methods of contrast enhancement like histogram equalization result in over/ under enhancement of the image especially of a lower resolution one. This project aims at developing a new fuzzy inference System to enhance the contrast of the images overcoming the shortcomings of the traditional methods**.**

Description:

- Convert input image from RGB to CIELAB, progress on L channel
- Calculate the average pixel intensity - M value
- **Fuzzification:** For each pixel, calculate degree of membership of each class based

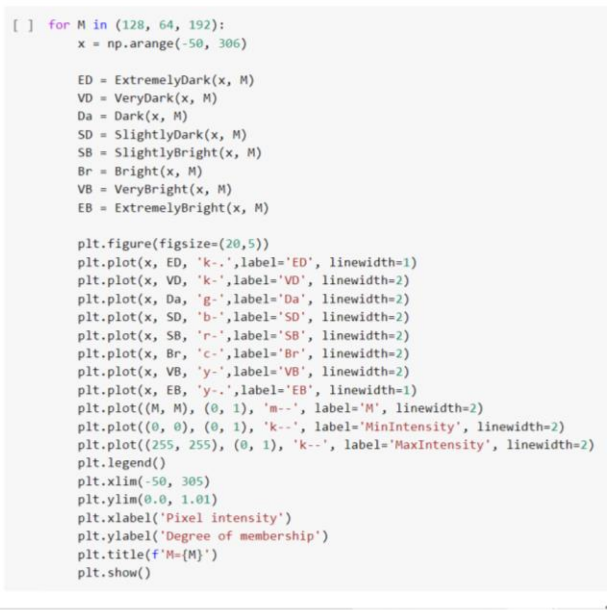
  on pixel intensity and M value.
- **Inference:** Calculate the output fuzzy set from the input pixel intensity based on the proposed rule set

- **Defuzzification:** For each pixel, calculate centroid value of its output fuzzy set.
- Normalize output pixel intensity from [-50, 305] to [0, 255]
- Merge modified L channel to the original AB channels, convert output image from CIELAB to RGB.



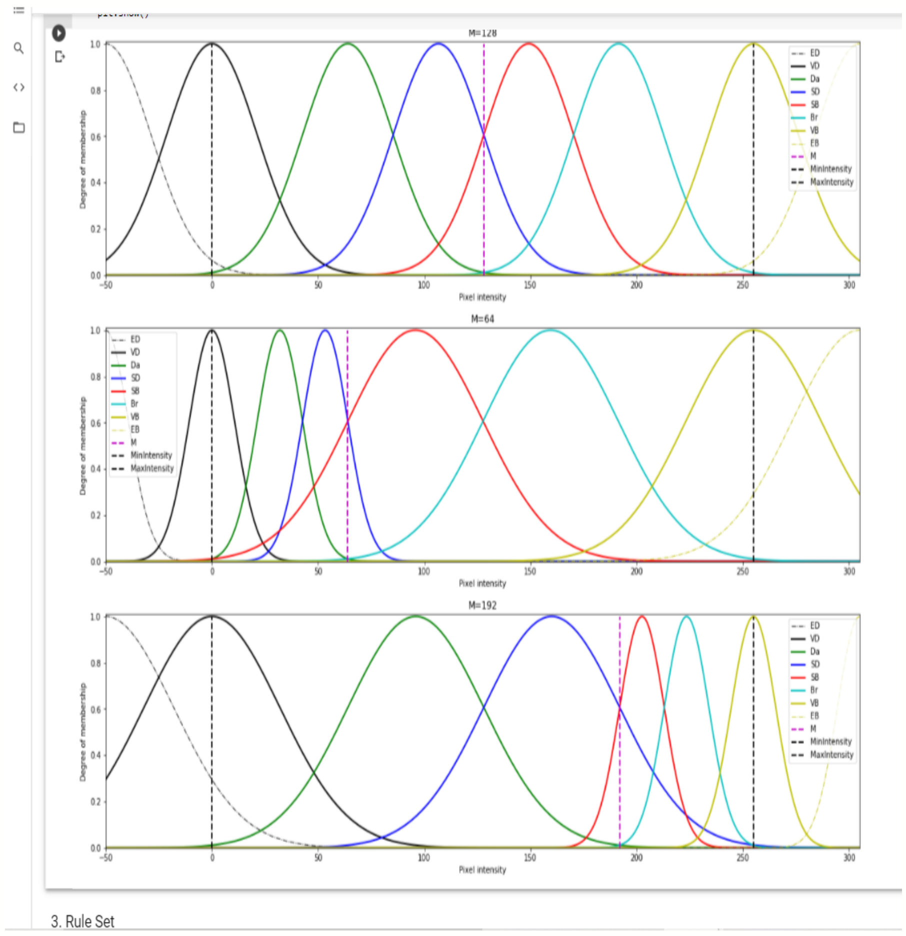
2. Fuzzification of Pixel Intensity

```
# Gaussian Function:
def G(x, mean, std):
    return np.exp(-0.5*np.square((x-mean)/std))

# Membership Functions:
def ExtremelyDark(x, M):
    return G(x, -50, M/6)

def VeryDark(x, M):
    return G(x, 0, M/6)

def Dark(x, M):
    return G(x, M/2, M/6)

def SlightlyDark(x, M):
    return G(x, 5*M/6, M/6)

def SlightlyBright(x, M):
    return G(x, M+(255-M)/6, (255-M)/6)

def Bright(x, M):
    return G(x, M+(255-M)/2, (255-M)/6)

def VeryBright(x, M):
    return G(x, 255, (255-M)/6)

def ExtremelyBright(x, M):
    return G(x, 305, (255-M)/6)
```



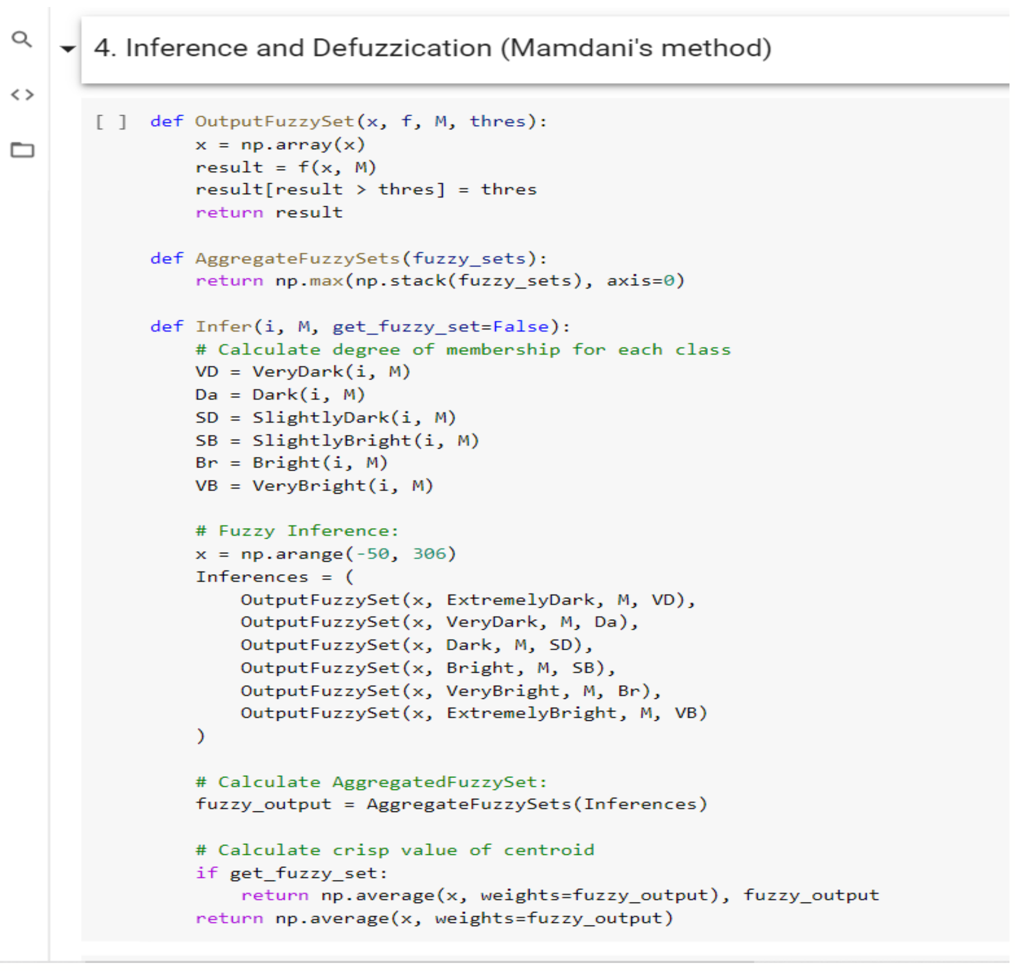
```
[ ] for M in (128, 64, 192):
        x = np.arange(-50, 306)

        ED = ExtremelyDark(x, M)
        VD = VeryDark(x, M)
        Da = Dark(x, M)
        SD = SlightlyDark(x, M)
        SB = SlightlyBright(x, M)
        Br = Bright(x, M)
        VB = VeryBright(x, M)
        EB = ExtremelyBright(x, M)

        plt.figure(figsize=(20,5))
        plt.plot(x, ED, 'k-.',label='ED', linewidth=1)
        plt.plot(x, VD, 'k-',label='VD', linewidth=2)
        plt.plot(x, Da, 'g-',label='Da', linewidth=2)
        plt.plot(x, SD, 'b-',label='SD', linewidth=2)
        plt.plot(x, SB, 'r-',label='SB', linewidth=2)
        plt.plot(x, Br, 'c-',label='Br', linewidth=2)
        plt.plot(x, VB, 'y-',label='VB', linewidth=2)
        plt.plot(x, EB, 'y-.',label='EB', linewidth=1)
        plt.plot((M, M), (0, 1), 'm--', label='M', linewidth=2)
        plt.plot((0, 0), (0, 1), 'k--', label='MinIntensity', linewidth=2)
        plt.plot((255, 255), (0, 1), 'k--', label='MaxIntensity', linewidth=2)
        plt.legend()
        plt.xlim(-50, 305)
        plt.ylim(0.0, 1.01)
        plt.xlabel('Pixel intensity')
        plt.ylabel('Degree of membership')
        plt.title(f'M={M}')
        plt.show()
```

The system uses 7 input membership functions to make the system more accurate, also Gaussian as well as trap membership functions are used which represent sets of pixel intensity values as linguistic variables like dark, gray, bright etc. shown in fig1. These membership functions were chosen after analyzing many images and the range of pixel values for these linguistic variables were set.



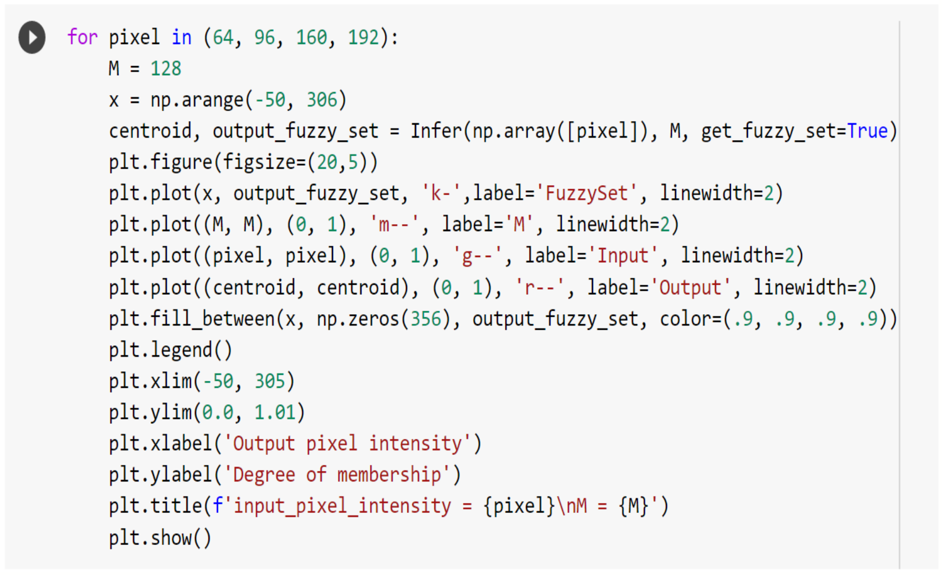
3. Rule Set

- IF input is VeryDark THEN output is ExtremelyDark
- IF input is Dark THEN output is VeryDark
- IF input is SlightlyDark THEN output is Dark
- IF input is SlightlyBright THEN output is Bright
- IF input is Bright THEN output is VeryBright
- IF input is VeryBright THEN output is ExtremelyBright

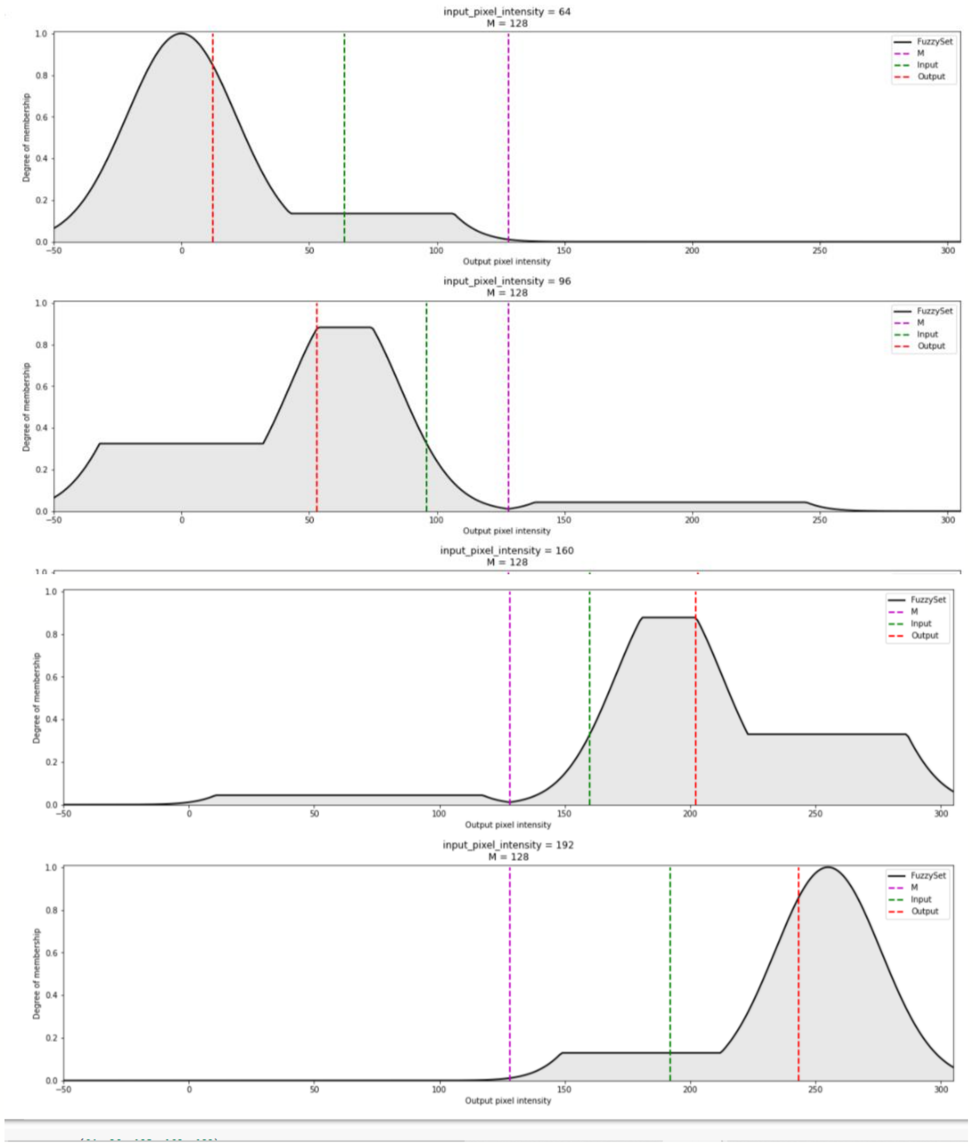


4. Inference and Defuzzication (Mamdani's method)

```
[ ]  def OutputFuzzySet(x, f, M, thres):
         x = np.array(x)
         result = f(x, M)
         result[result > thres] = thres
         return result

     def AggregateFuzzySets(fuzzy_sets):
         return np.max(np.stack(fuzzy_sets), axis=0)

     def Infer(i, M, get_fuzzy_set=False):
         # Calculate degree of membership for each class
         VD = VeryDark(i, M)
         Da = Dark(i, M)
         SD = SlightlyDark(i, M)
         SB = SlightlyBright(i, M)
         Br = Bright(i, M)
         VB = VeryBright(i, M)

         # Fuzzy Inference:
         x = np.arange(-50, 306)
         Inferences = (
             OutputFuzzySet(x, ExtremelyDark, M, VD),
             OutputFuzzySet(x, VeryDark, M, Da),
             OutputFuzzySet(x, Dark, M, SD),
             OutputFuzzySet(x, Bright, M, SB),
             OutputFuzzySet(x, VeryBright, M, Br),
             OutputFuzzySet(x, ExtremelyBright, M, VB)
         )

         # Calculate AggregatedFuzzySet:
         fuzzy_output = AggregateFuzzySets(Inferences)

         # Calculate crisp value of centroid
         if get_fuzzy_set:
             return np.average(x, weights=fuzzy_output), fuzzy_output
         return np.average(x, weights=fuzzy_output)
```

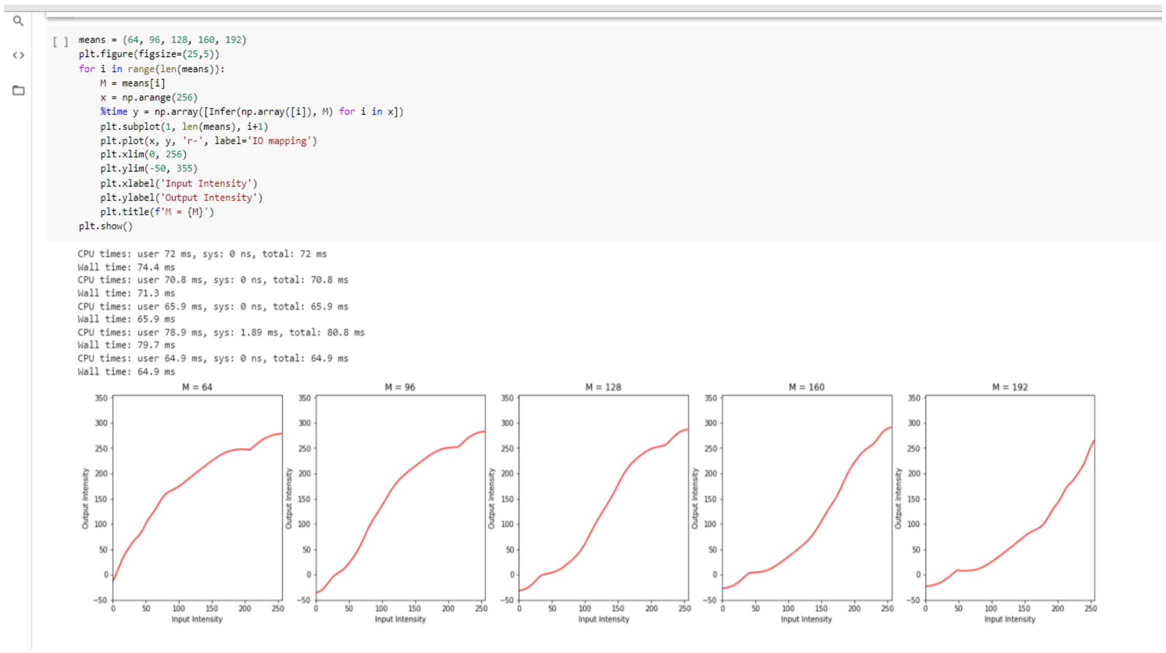


```
for pixel in (64, 96, 160, 192):
    M = 128
    x = np.arange(-50, 306)
    centroid, output_fuzzy_set = Infer(np.array([pixel]), M, get_fuzzy_set=True)
    plt.figure(figsize=(20,5))
    plt.plot(x, output_fuzzy_set, 'k-',label='FuzzySet', linewidth=2)
    plt.plot((M, M), (0, 1), 'm--', label='M', linewidth=2)
    plt.plot((pixel, pixel), (0, 1), 'g--', label='Input', linewidth=2)
    plt.plot((centroid, centroid), (0, 1), 'r--', label='Output', linewidth=2)
    plt.fill_between(x, np.zeros(356), output_fuzzy_set, color=(.9, .9, .9, .9))
    plt.legend()
    plt.xlim(-50, 305)
    plt.ylim(0.0, 1.01)
    plt.xlabel('Output pixel intensity')
    plt.ylabel('Degree of membership')
    plt.title(f'input_pixel_intensity = {pixel}\nM = {M}')
    plt.show()
```



The output membership functions define the amount by which the pixel intensity should be increased or decreased based on the rules defined in the knowledge base. The crisp intensity modification value is obtained from the output membership functions through the process of defuzzification.

The FIS(Fuzzy inference System) was designed using a fuzzy logic toolbox in Matlab. The algorithm uses mamdani FIS. The FIS contains a knowledge base formulated by an expert, this contains IF-THEN rules. These rules map the fuzzy inputs to fuzzy outputs and take place through compositional rule of intuition.

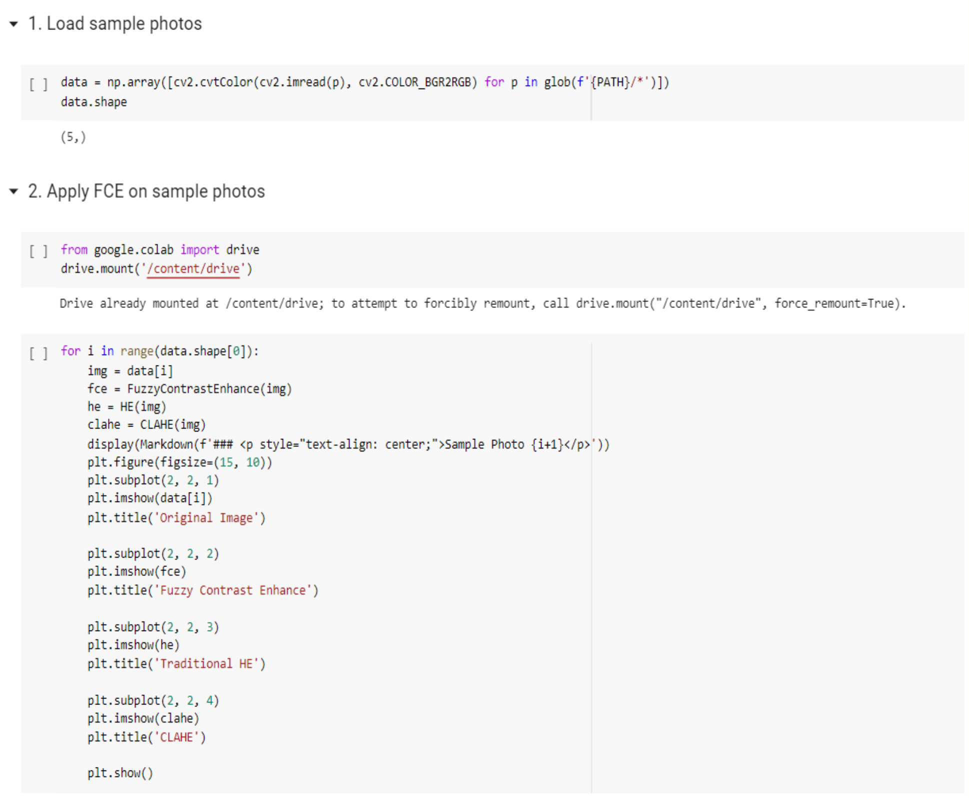


```
means = (64, 96, 128, 160, 192)
plt.figure(figsize=(25,5))
for i in range(len(means)):
    M = means[i]
    x = np.arange(256)
    %time y = np.array([Infer(np.array([i]), M) for i in x])
    plt.subplot(1, len(means), i+1)
    plt.plot(x, y, 'r-', label='IO mapping')
    plt.xlim(0, 256)
    plt.ylim(-50, 355)
    plt.xlabel('Input Intensity')
    plt.ylabel('Output Intensity')
    plt.title(f'M = {M}')
plt.show()
```

```
CPU times: user 72 ms, sys: 0 ns, total: 72 ms
Wall time: 74.4 ms
CPU times: user 70.8 ms, sys: 0 ns, total: 70.8 ms
Wall time: 71.3 ms
CPU times: user 65.9 ms, sys: 0 ns, total: 65.9 ms
Wall time: 65.9 ms
CPU times: user 78.9 ms, sys: 1.89 ms, total: 80.8 ms
Wall time: 79.7 ms
CPU times: user 64.9 ms, sys: 0 ns, total: 64.9 ms
Wall time: 64.9 ms
```

IV. Demonstration

```
# Proposed fuzzy method
def FuzzyContrastEnhance(rgb):
    # Convert RGB to LAB
    lab = cv2.cvtColor(rgb, cv2.COLOR_RGB2LAB)

    # Get L channel
    l = lab[:, :, 0]

    # Calculate M value
    M = np.mean(l)
    if M < 128:
        M = 127 - (127 - M)/2
    else:
        M = 128 + M/2

    # Precompute the fuzzy transform
    x = list(range(-50,306))
    FuzzyTransform = dict(zip(x,[Infer(np.array([i]), M) for i in x]))

    # Apply the transform to l channel
    u, inv = np.unique(l, return_inverse = True)
    l = np.array([FuzzyTransform[i] for i in u])[inv].reshape(l.shape)

    # Min-max scale the output L channel to fit (0, 255):
    Min = np.min(l)
    Max = np.max(l)
    lab[:, :, 0] = (l - Min)/(Max - Min) * 255

    # Convert LAB to RGB
    return cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)

# Traditional method of histogram equalization
def HE(rgb):
    lab = cv2.cvtColor(rgb, cv2.COLOR_RGB2LAB)
    lab[:, :, 0] = cv2.equalizeHist(lab[:, :, 0])
    return cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)

# Contrast Limited Adaptive Histogram Equalization
def CLAHE(rgb):
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    lab = cv2.cvtColor(rgb, cv2.COLOR_RGB2LAB)
    lab[:, :, 0] = clahe.apply(lab[:, :, 0])
    return cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
```



1. Load sample photos

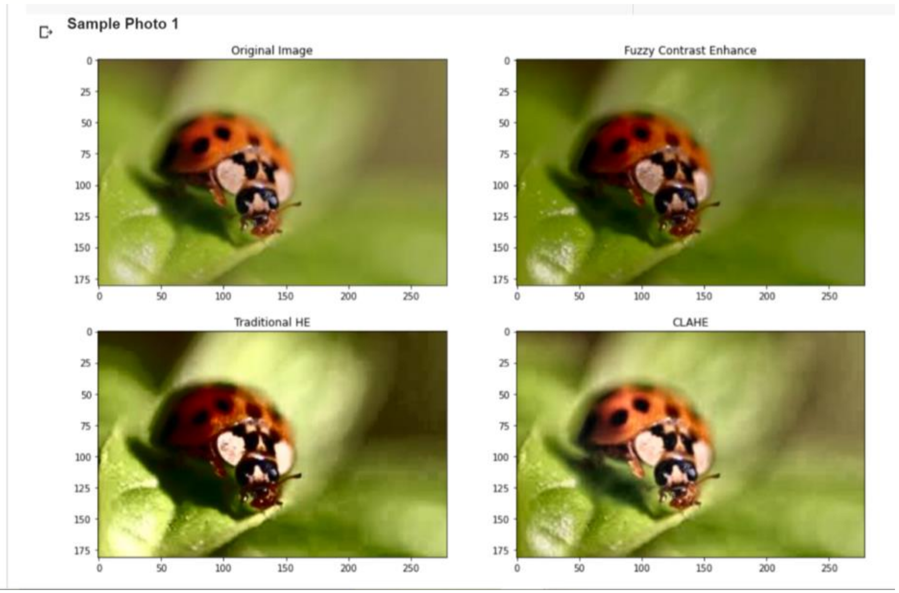
```
[ ] data = np.array([cv2.cvtColor(cv2.imread(p), cv2.COLOR_BGR2RGB) for p in glob(f'{PATH}/*')])
    data.shape

    (5,)
```

2. Apply FCE on sample photos

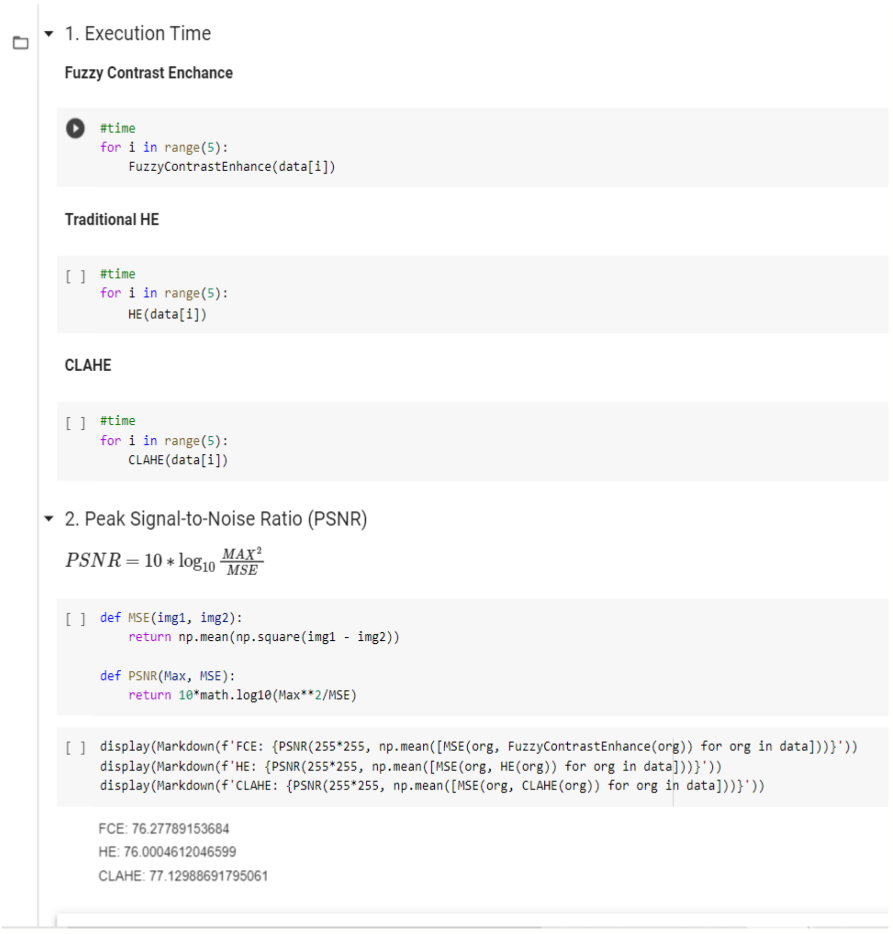
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] for i in range(data.shape[0]):
        img = data[i]
        fce = FuzzyContrastEnhance(img)
        he = HE(img)
        clahe = CLAHE(img)
        display(Markdown(f'### <p style="text-align: center;">Sample Photo {i+1}</p>'))
        plt.figure(figsize=(15, 10))
        plt.subplot(2, 2, 1)
        plt.imshow(data[i])
        plt.title('Original Image')

        plt.subplot(2, 2, 2)
        plt.imshow(fce)
        plt.title('Fuzzy Contrast Enhance')

        plt.subplot(2, 2, 3)
        plt.imshow(he)
        plt.title('Traditional HE')

        plt.subplot(2, 2, 4)
        plt.imshow(clahe)
        plt.title('CLAHE')

        plt.show()
```

In order to determine the efficacy of the system, we have compared the results obtained with histogram equalization. Various low contrast images were taken and fed into the Matlab and the obtained results are as follows.



The low contrast images, histogram equalized image and fuzzy logic processed image and their respective histograms are presented


1. Execution Time

Fuzzy Contrast Enchance

```
#time
for i in range(5):
    FuzzyContrastEnhance(data[i])
```

Traditional HE

```
#time
for i in range(5):
    HE(data[i])
```

CLAHE

```
#time
for i in range(5):
    CLAHE(data[i])
```

2. Peak Signal-to-Noise Ratio (PSNR)

$$PSNR = 10 * \log_{10} \frac{MAX^2}{MSE}$$

```
def MSE(img1, img2):
    return np.mean(np.square(img1 - img2))

def PSNR(Max, MSE):
    return 10*math.log10(Max**2/MSE)
```

```
display(Markdown(f'FCE: {PSNR(255*255, np.mean([MSE(org, FuzzyContrastEnhance(org)) for org in data]))}'))
display(Markdown(f'HE: {PSNR(255*255, np.mean([MSE(org, HE(org)) for org in data]))}'))
display(Markdown(f'CLAHE: {PSNR(255*255, np.mean([MSE(org, CLAHE(org)) for org in data]))}'))
```

```
FCE: 76.27789153684
HE: 76.0004612046599
CLAHE: 77.12988691795061
```

Various evaluation metrics were calculated for images enhanced using histogram equalization and fuzzy logic. The evaluation metrics used are described as follows.

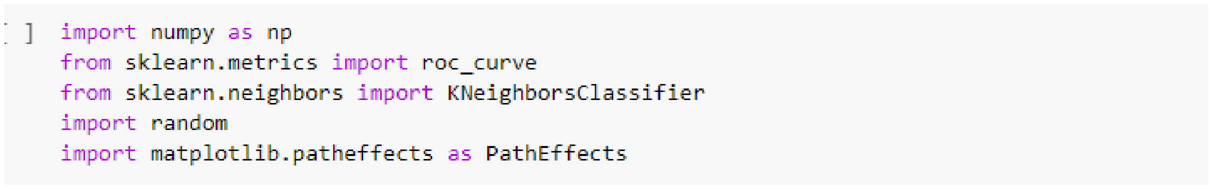
1. Peak Signal to Noise Ratio (PSNR)

PSNR is the ratio of maximum possible power of a signal to power of the corrupting noise. PSNR is an approximation to human perception of reconstruction quality. Higher the PSNR, better is the enhancement, but this may lead to over/under enhancement. Initially, mean square error is calculated and then PSNR is found

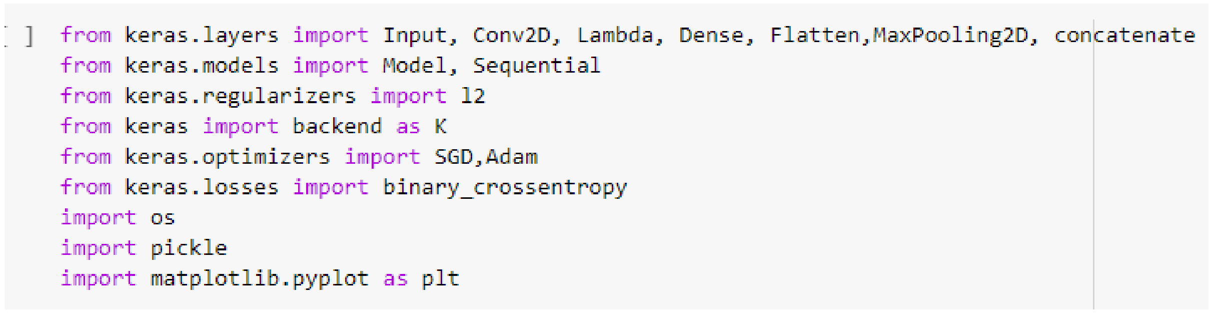
The performance of the different contrast enhancement algorithms are evaluated based on the computational time.The inter comparison of different techniques was carried out on different low contrast color images. Based on the performance analysis, we advocate that our proposed Fuzzy Logic method is well suited for contrast enhancement of low contrast color images.

B) Triplet Neural Network Model

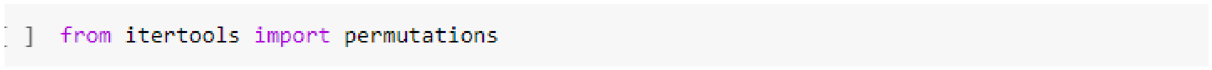
Triplet loss is a loss function for machine learning algorithms where a baseline (anchor) input is compared to a positive (truthy) input and a negative (falsy) input. The distance from the baseline (anchor) input to the positive (truthy) input is minimized, and the distance from the baseline (anchor) input to the negative (falsy) input is maximized. The first formulation equivalent to the triplet loss was introduced (without the idea of using anchors) for metric learning from relative comparisons by M. Schultze and T. Joachims in 2003



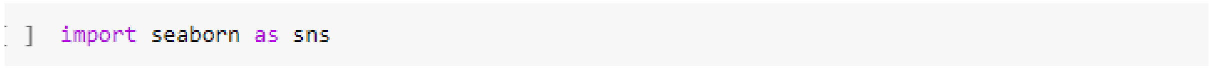
```
import numpy as np
from sklearn.metrics import roc_curve
from sklearn.neighbors import KNeighborsClassifier
import random
import matplotlib.patheffects as PathEffects
```



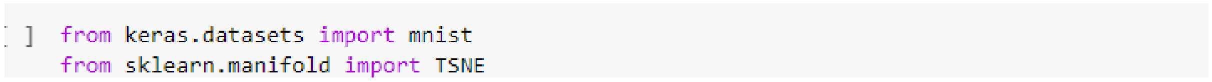
```
from keras.layers import Input, Conv2D, Lambda, Dense, Flatten,MaxPooling2D, concatenate
from keras.models import Model, Sequential
from keras.regularizers import l2
from keras import backend as K
from keras.optimizers import SGD,Adam
from keras.losses import binary_crossentropy
import os
import pickle
import matplotlib.pyplot as plt
```



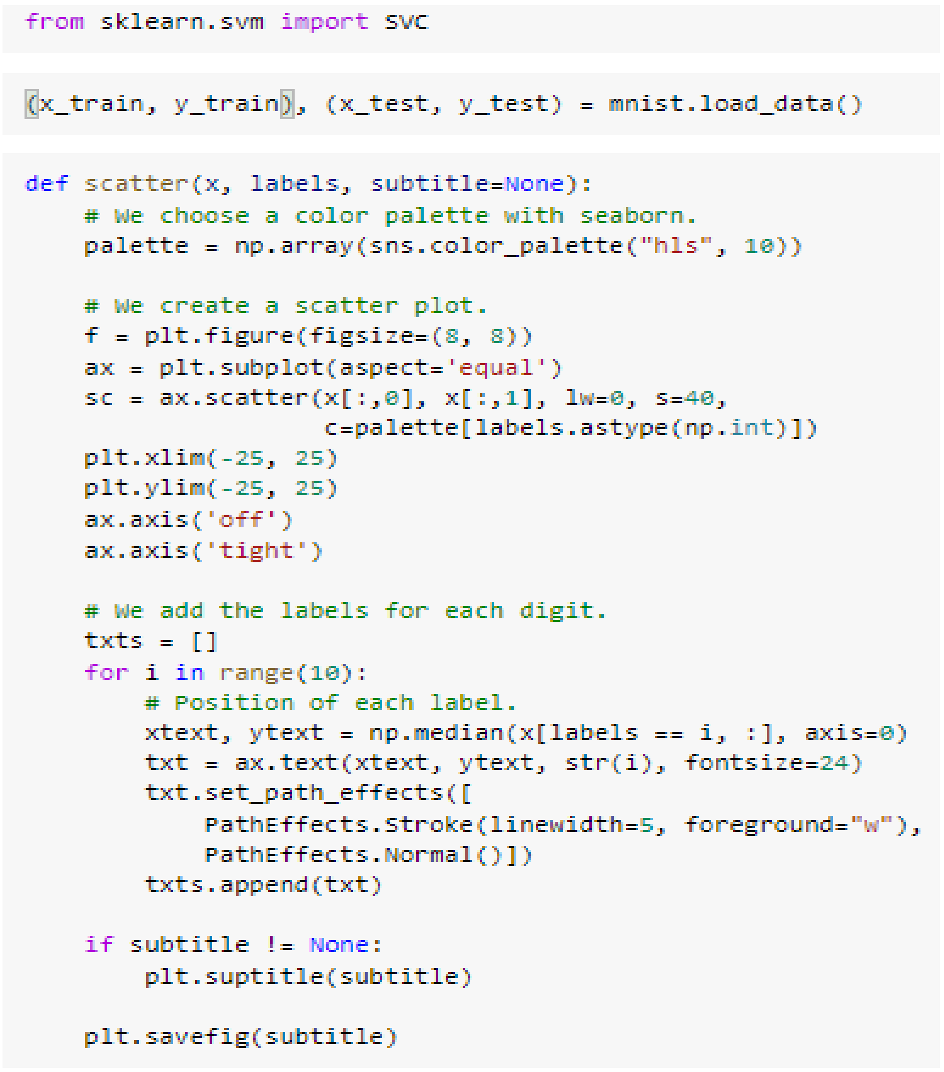
```
from itertools import permutations
```



```
import seaborn as sns
```



```
from keras.datasets import mnist
from sklearn.manifold import TSNE
```

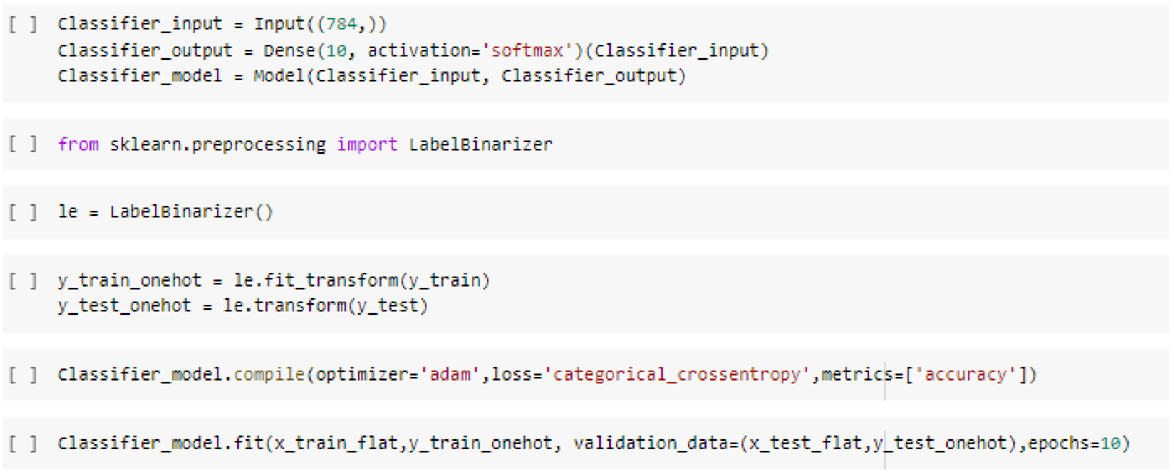


```
from sklearn.svm import SVC
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
def scatter(x, labels, subtitle=None):
    # We choose a color palette with seaborn.
    palette = np.array(sns.color_palette("hls", 10))

    # We create a scatter plot.
    f = plt.figure(figsize=(8, 8))
    ax = plt.subplot(aspect='equal')
    sc = ax.scatter(x[:,0], x[:,1], lw=0, s=40,
                    c=palette[labels.astype(np.int)])
    plt.xlim(-25, 25)
    plt.ylim(-25, 25)
    ax.axis('off')
    ax.axis('tight')

    # We add the labels for each digit.
    txts = []
    for i in range(10):
        # Position of each label.
        xtext, ytext = np.median(x[labels == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
        txts.append(txt)

    if subtitle != None:
        plt.suptitle(subtitle)

    plt.savefig(subtitle)
```

For triplet neural net, we have used mnist dataset and the below figures show samples present in each class for both training and testing data. A basic neural network was designed for classification and this is the first neural network. This was the accuracy obtained for the first neural net:



```
[ ] Classifier_input = Input((784,))
    Classifier_output = Dense(10, activation='softmax')(Classifier_input)
    Classifier_model = Model(Classifier_input, Classifier_output)
```

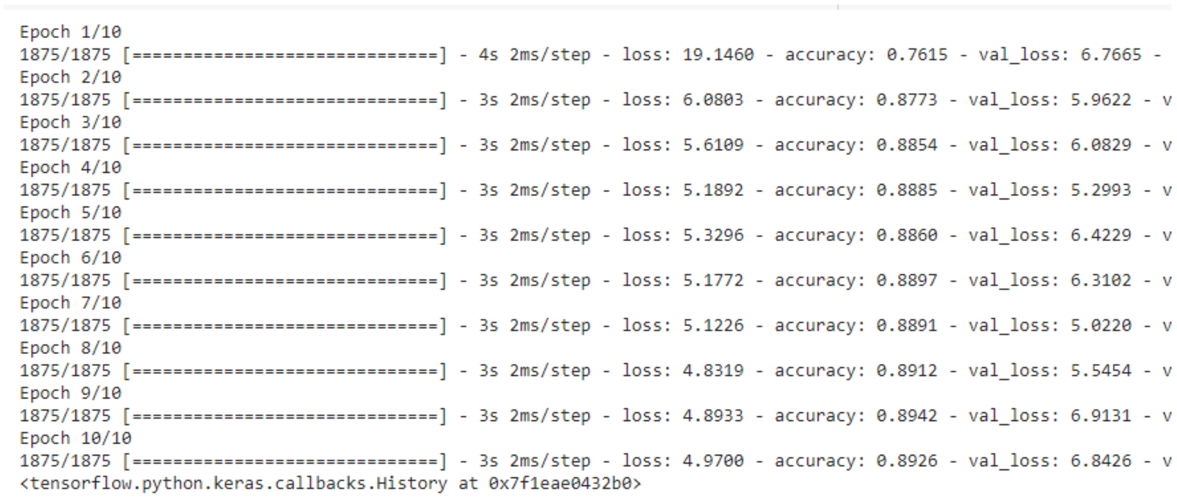
```
[ ] from sklearn.preprocessing import LabelBinarizer
```

```
[ ] le = LabelBinarizer()
```

```
[ ] y_train_onehot = le.fit_transform(y_train)
    y_test_onehot = le.transform(y_test)
```

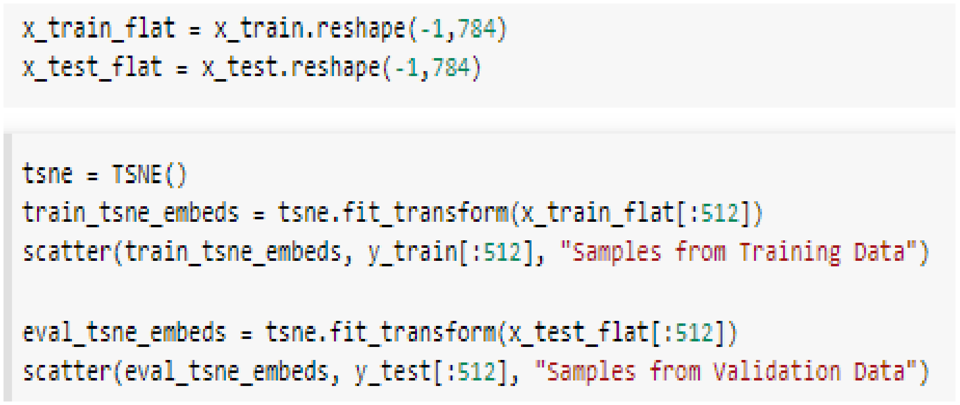
```
[ ] Classifier_model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
[ ] Classifier_model.fit(x_train_flat,y_train_onehot, validation_data=(x_test_flat,y_test_onehot),epochs=10)
```



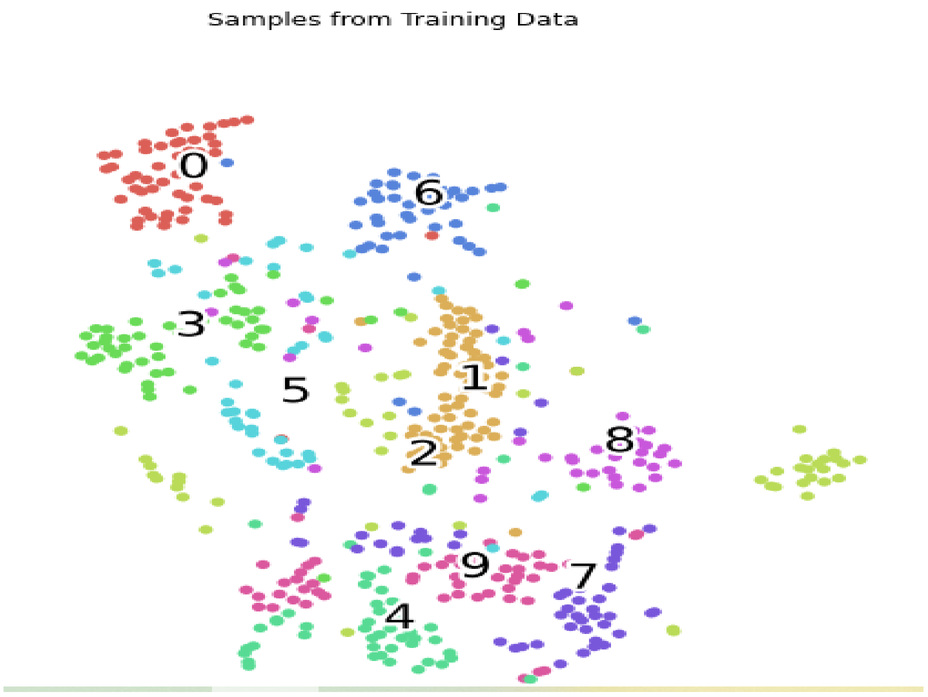
```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 19.1460 - accuracy: 0.7615 - val_loss: 6.7665 -
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 6.0803 - accuracy: 0.8773 - val_loss: 5.9622 - v
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 5.6109 - accuracy: 0.8854 - val_loss: 6.0829 - v
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 5.1892 - accuracy: 0.8885 - val_loss: 5.2993 - v
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 5.3296 - accuracy: 0.8860 - val_loss: 6.4229 - v
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 5.1772 - accuracy: 0.8897 - val_loss: 6.3102 - v
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 5.1226 - accuracy: 0.8891 - val_loss: 5.0220 - v
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 4.8319 - accuracy: 0.8912 - val_loss: 5.5454 - v
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 4.8933 - accuracy: 0.8942 - val_loss: 6.9131 - v
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 4.9700 - accuracy: 0.8926 - val_loss: 6.8426 - v
<tensorflow.python.keras.callbacks.History at 0x7f1eae0432b0>
```

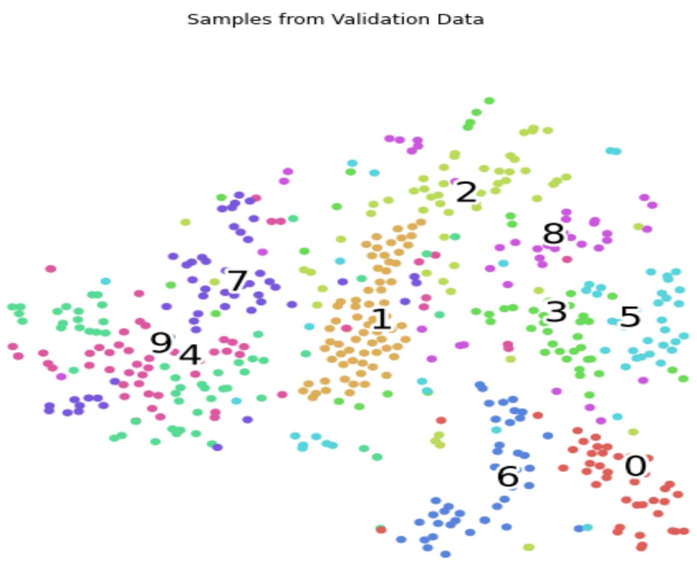
We have used t-SNE for plotting the clustering of the data. This was the classification before sending the data to the model.



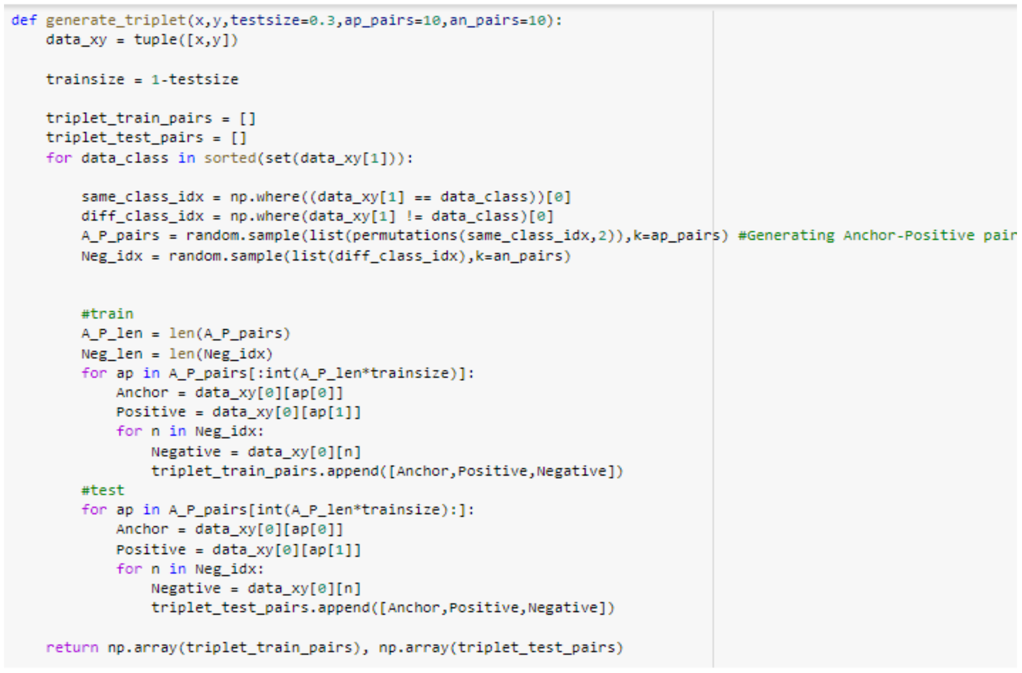
```python
x_train_flat = x_train.reshape(-1,784)
x_test_flat = x_test.reshape(-1,784)
```

```python
tsne = TSNE()
train_tsne_embeds = tsne.fit_transform(x_train_flat[:512])
scatter(train_tsne_embeds, y_train[:512], "Samples from Training Data")

eval_tsne_embeds = tsne.fit_transform(x_test_flat[:512])
scatter(eval_tsne_embeds, y_test[:512], "Samples from Validation Data")
```

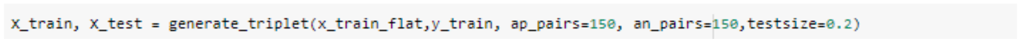




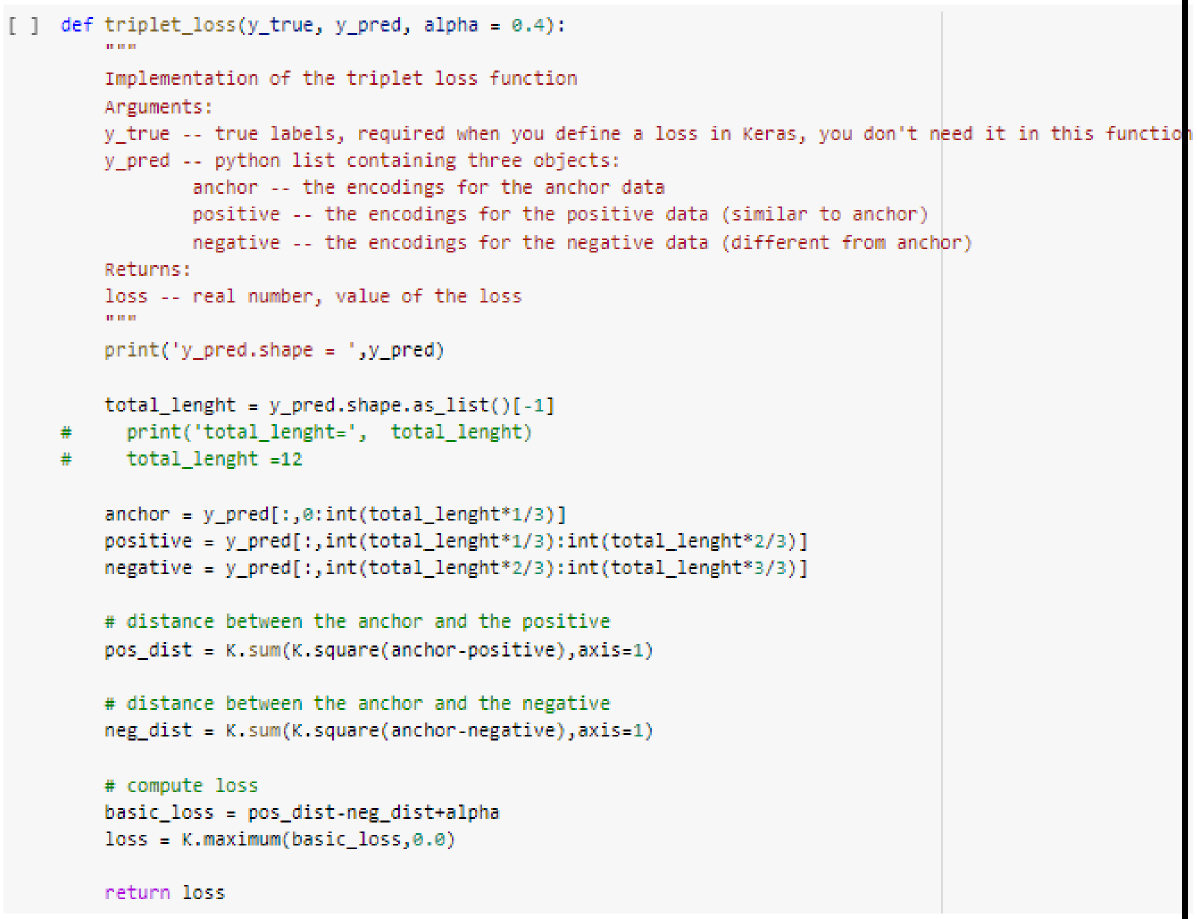
Once this process has been completed, the middle neural network was created for checking the performance of the image search. In order to achieve this an anchor image is kept which is our required image compared against positive and negative images. Positive images are nothing but images that are classified under proper labels and hence very similar to the anchor image. Negative images are wrongly classified images or that doesn’t show similarity to the anchor image. The aim of the model is to increase the creation of positive images rather than negative images in terms of their distance. Concept of K nearest neighbours and Euclidean distance is used here in order to achieve this. The figure below shows the model summary:



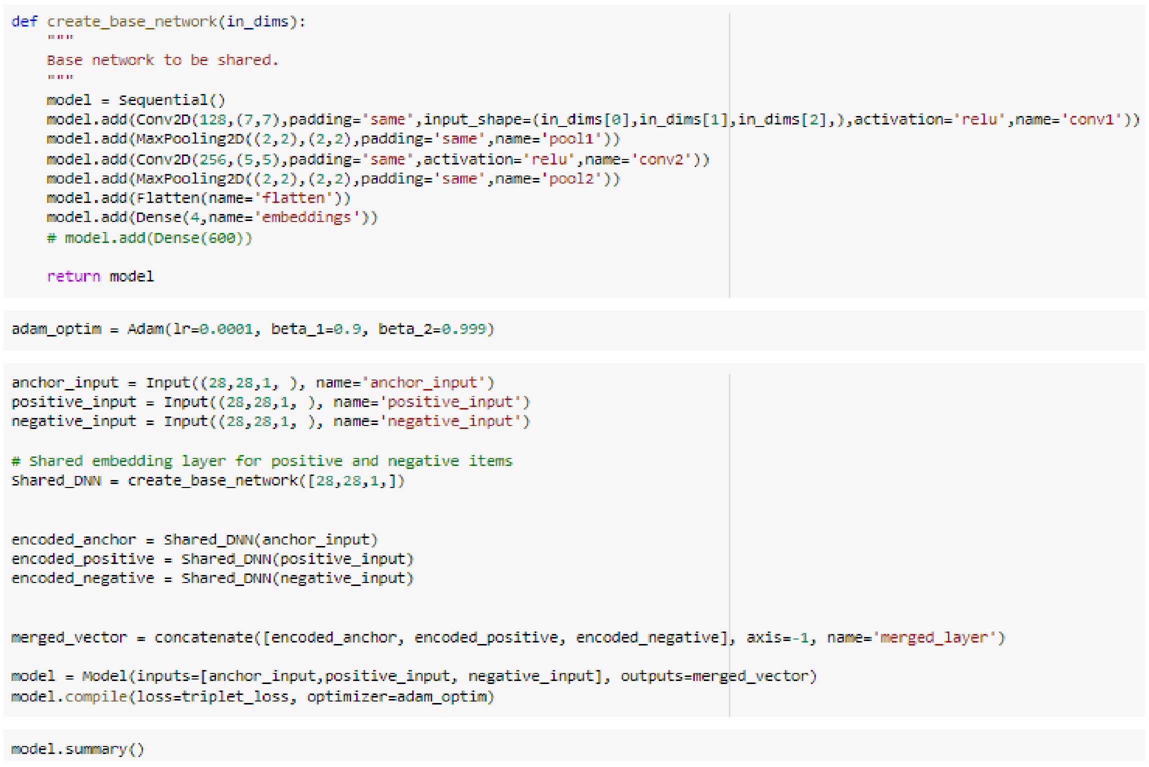
```
def generate_triplet(x,y,testsize=0.3,ap_pairs=10,an_pairs=10):
    data_xy = tuple([x,y])

    trainsize = 1-testsize

    triplet_train_pairs = []
    triplet_test_pairs = []
    for data_class in sorted(set(data_xy[1])):

        same_class_idx = np.where((data_xy[1] == data_class))[0]
        diff_class_idx = np.where(data_xy[1] != data_class)[0]
        A_P_pairs = random.sample(list(permutations(same_class_idx,2)),k=ap_pairs) #Generating Anchor-Positive pair
        Neg_idx = random.sample(list(diff_class_idx),k=an_pairs)


        #train
        A_P_len = len(A_P_pairs)
        Neg_len = len(Neg_idx)
        for ap in A_P_pairs[:int(A_P_len*trainsize)]:
            Anchor = data_xy[0][ap[0]]
            Positive = data_xy[0][ap[1]]
            for n in Neg_idx:
                Negative = data_xy[0][n]
                triplet_train_pairs.append([Anchor,Positive,Negative])
        #test
        for ap in A_P_pairs[int(A_P_len*trainsize):]:
            Anchor = data_xy[0][ap[0]]
            Positive = data_xy[0][ap[1]]
            for n in Neg_idx:
                Negative = data_xy[0][n]
                triplet_test_pairs.append([Anchor,Positive,Negative])

    return np.array(triplet_train_pairs), np.array(triplet_test_pairs)
```



```
X_train, X_test = generate_triplet(x_train_flat,y_train, ap_pairs=150, an_pairs=150,testsize=0.2)
```



```
[ ]  def triplet_loss(y_true, y_pred, alpha = 0.4):
         """
         Implementation of the triplet loss function
         Arguments:
         y_true -- true labels, required when you define a loss in Keras, you don't need it in this functio
         y_pred -- python list containing three objects:
                 anchor -- the encodings for the anchor data
                 positive -- the encodings for the positive data (similar to anchor)
                 negative -- the encodings for the negative data (different from anchor)
         Returns:
         loss -- real number, value of the loss
         """
         print('y_pred.shape = ',y_pred)

         total_lenght = y_pred.shape.as_list()[-1]
     #     print('total_lenght=',  total_lenght)
     #     total_lenght =12

         anchor = y_pred[:,0:int(total_lenght*1/3)]
         positive = y_pred[:,int(total_lenght*1/3):int(total_lenght*2/3)]
         negative = y_pred[:,int(total_lenght*2/3):int(total_lenght*3/3)]

         # distance between the anchor and the positive
         pos_dist = K.sum(K.square(anchor-positive),axis=1)

         # distance between the anchor and the negative
         neg_dist = K.sum(K.square(anchor-negative),axis=1)

         # compute loss
         basic_loss = pos_dist-neg_dist+alpha
         loss = K.maximum(basic_loss,0.0)

         return loss
```

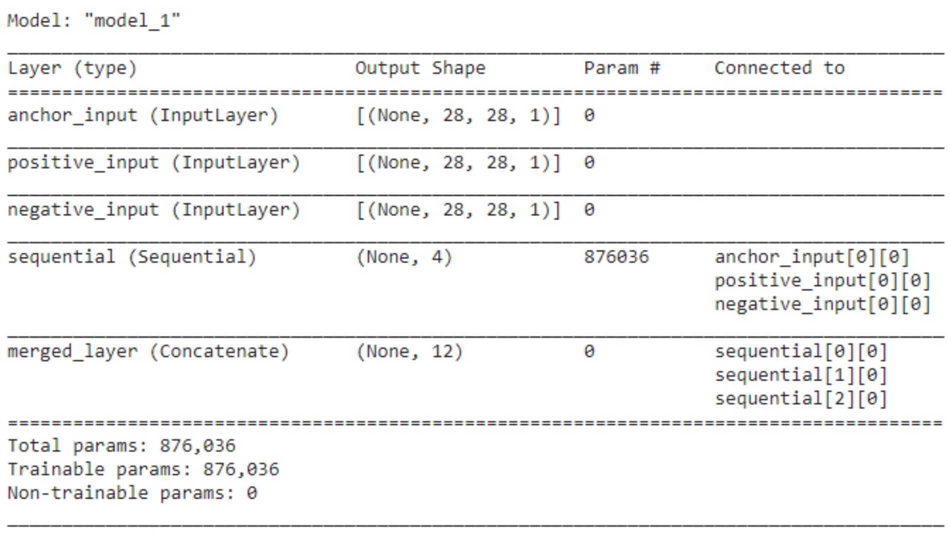


```
def create_base_network(in_dims):
    """
    Base network to be shared.
    """
    model = Sequential()
    model.add(Conv2D(128,(7,7),padding='same',input_shape=(in_dims[0],in_dims[1],in_dims[2],),activation='relu',name='conv1'))
    model.add(MaxPooling2D((2,2),(2,2),padding='same',name='pool1'))
    model.add(Conv2D(256,(5,5),padding='same',activation='relu',name='conv2'))
    model.add(MaxPooling2D((2,2),(2,2),padding='same',name='pool2'))
    model.add(Flatten(name='flatten'))
    model.add(Dense(4,name='embeddings'))
    # model.add(Dense(600))

    return model
```

```
adam_optim = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999)
```

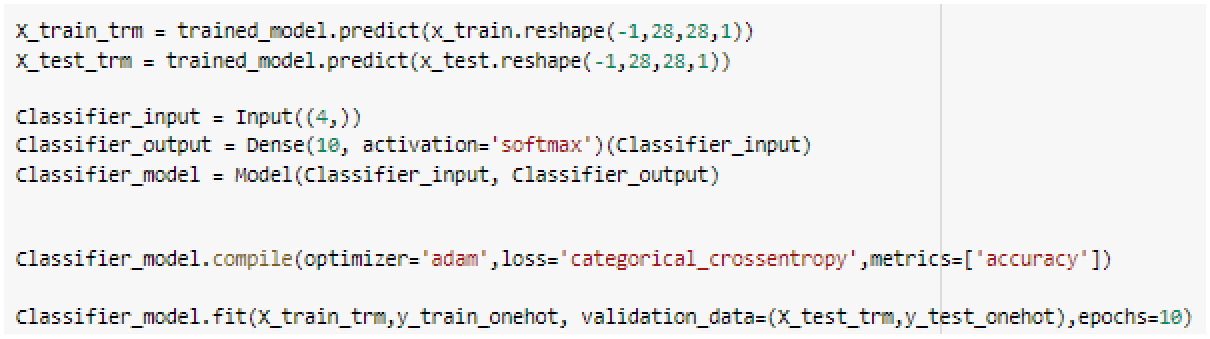
```
anchor_input = Input((28,28,1, ), name='anchor_input')
positive_input = Input((28,28,1, ), name='positive_input')
negative_input = Input((28,28,1, ), name='negative_input')

# Shared embedding layer for positive and negative items
Shared_DNN = create_base_network([28,28,1,])


encoded_anchor = Shared_DNN(anchor_input)
encoded_positive = Shared_DNN(positive_input)
encoded_negative = Shared_DNN(negative_input)


merged_vector = concatenate([encoded_anchor, encoded_positive, encoded_negative], axis=-1, name='merged_layer')

model = Model(inputs=[anchor_input,positive_input, negative_input], outputs=merged_vector)
model.compile(loss=triplet_loss, optimizer=adam_optim)
```

```
model.summary()
```

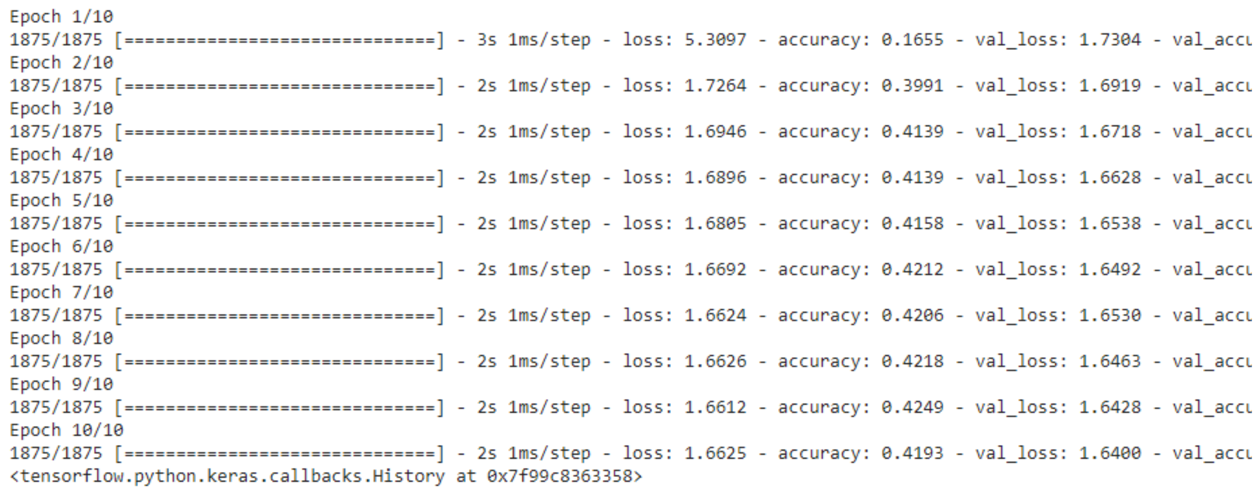


```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
anchor_input (InputLayer)       [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
positive_input (InputLayer)     [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
negative_input (InputLayer)     [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
sequential (Sequential)         (None, 4)            876036      anchor_input[0][0]
                                                                 positive_input[0][0]
                                                                 negative_input[0][0]
__________________________________________________________________________________________________
merged_layer (Concatenate)      (None, 12)           0           sequential[0][0]
                                                                 sequential[1][0]
                                                                 sequential[2][0]
==================================================================================================
Total params: 876,036
Trainable params: 876,036
Non-trainable params: 0
__________________________________________________________________________________________________
```

The final results were sent to another neural net similar to the first neural net in order to show the classification of image classes after training of the middle neural network.The following figure shows the accuracy obtained for the final classification:



```
X_train_trm = trained_model.predict(x_train.reshape(-1,28,28,1))
X_test_trm = trained_model.predict(x_test.reshape(-1,28,28,1))

Classifier_input = Input((4,))
Classifier_output = Dense(10, activation='softmax')(Classifier_input)
Classifier_model = Model(Classifier_input, Classifier_output)


Classifier_model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

Classifier_model.fit(X_train_trm,y_train_onehot, validation_data=(X_test_trm,y_test_onehot),epochs=10)
```



```
Epoch 1/10
1875/1875 [==============================] - 3s 1ms/step - loss: 5.3097 - accuracy: 0.1655 - val_loss: 1.7304 - val_acc
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.7264 - accuracy: 0.3991 - val_loss: 1.6919 - val_acc
Epoch 3/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6946 - accuracy: 0.4139 - val_loss: 1.6718 - val_acc
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6896 - accuracy: 0.4139 - val_loss: 1.6628 - val_acc
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6805 - accuracy: 0.4158 - val_loss: 1.6538 - val_acc
Epoch 6/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6692 - accuracy: 0.4212 - val_loss: 1.6492 - val_acc
Epoch 7/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6624 - accuracy: 0.4206 - val_loss: 1.6530 - val_acc
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6626 - accuracy: 0.4218 - val_loss: 1.6463 - val_acc
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6612 - accuracy: 0.4249 - val_loss: 1.6428 - val_acc
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 1.6625 - accuracy: 0.4193 - val_loss: 1.6400 - val_acc
<tensorflow.python.keras.callbacks.History at 0x7f99c8363358>
```

On visualisation of the results the final classification obtained is as follows:

```
Anchor = X_train[:,0,:].reshape(-1,28,28,1)
Positive = X_train[:,1,:].reshape(-1,28,28,1)
Negative = X_train[:,2,:].reshape(-1,28,28,1)
Anchor_test = X_test[:,0,:].reshape(-1,28,28,1)
Positive_test = X_test[:,1,:].reshape(-1,28,28,1)
Negative_test = X_test[:,2,:].reshape(-1,28,28,1)

Y_dummy = np.empty((Anchor.shape[0],300))
Y_dummy2 = np.empty((Anchor_test.shape[0],1))

model.fit([Anchor,Positive,Negative],y=Y_dummy,validation_data=([Anchor_test,Positive_test,Negative_test],Y_dummy2), batch_size=512, epochs=500)
```

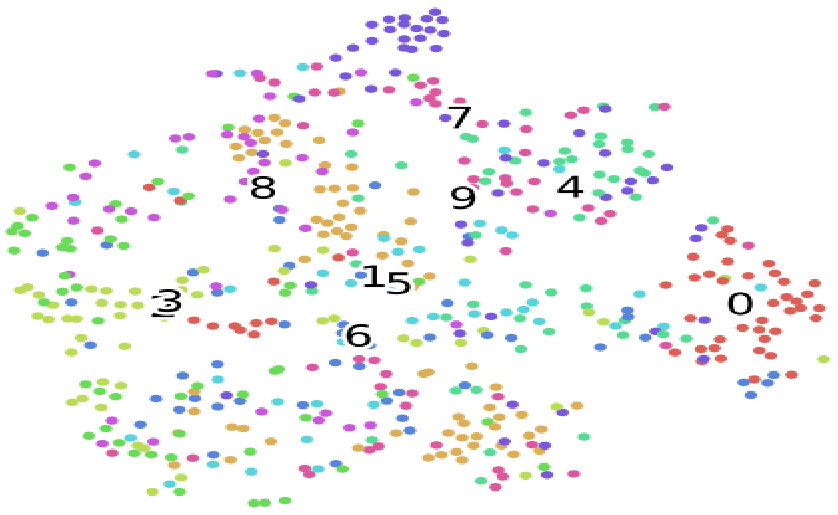
```
trained_model = Model(inputs=anchor_input, outputs=encoded_anchor)
```

```
trained_model.load_weights('triplet_model_MNIST.hdf5')
```

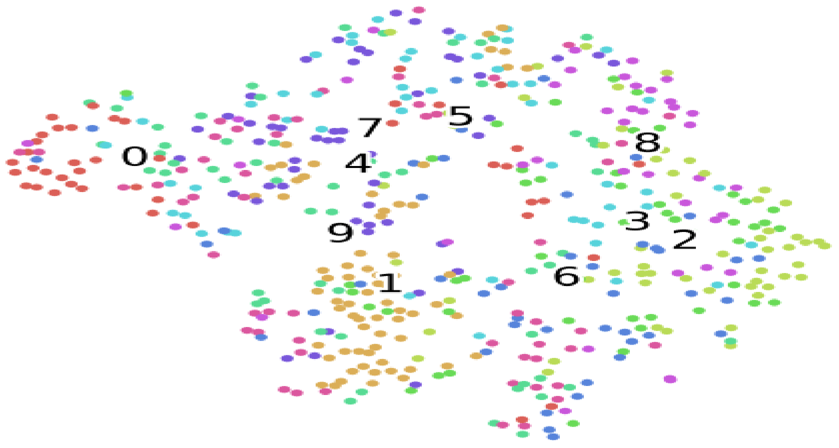
```
tsne = TSNE()
X_train_trm = trained_model.predict(x_train[:512].reshape(-1,28,28,1))
X_test_trm = trained_model.predict(x_test[:512].reshape(-1,28,28,1))
train_tsne_embeds = tsne.fit_transform(X_train_trm)
eval_tsne_embeds = tsne.fit_transform(X_test_trm)
```

```
scatter(train_tsne_embeds, y_train[:512], "Training Data After TNN")
```

Training Data After TNN



Validation Data After TNN



C)<u>Final child images using Genetic Algorithm</u>

Genetic algorithm Implementation

Reproducing Images using a Genetic Algorithm. The project accepts an image as input. This image can have one or more channels (i.e. the image could be binary, gray, or color, such as RGB). The genetic algorithm (GA) starts from a randomly generated image of the same shape as the input image. This randomly generated image is evolved, using crossover and mutation, using GA until it reproduces an image similar to the original one.

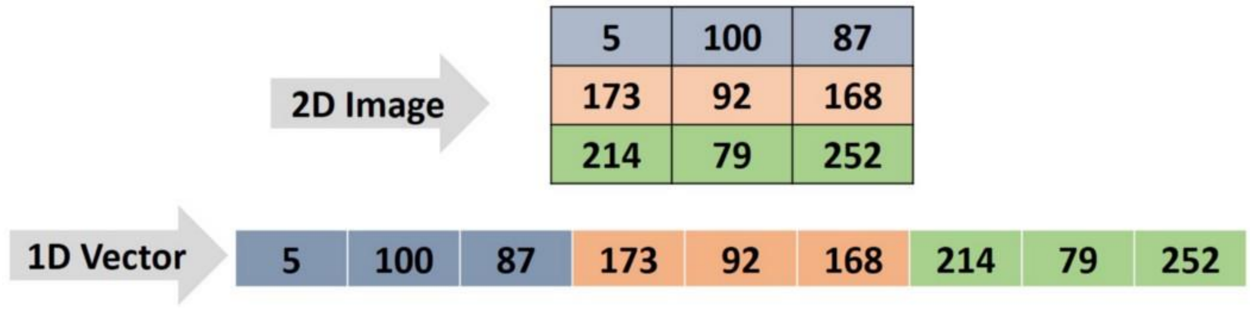
Genetic Algorithm Steps

GA is a random-based optimization technique that has a number of generic steps that are generally followed to solve any optimization problem. These steps are then customized to the problem being solved.

- **Data Representation**
- **Initial Population**
- **Fitness Calculation**
- **Parent Selection**
- **Crossover**
- **Mutation**

Data Representation

The first task for an optimization problem using GA is to think about the best way to represent the data. GA accepts the chromosome (i.e. solution) as a 1D row vector. The input image will not be 1D. The image may be 2D if it’s a binary or a gray image.



code

```
def img2chromosome(img_arr):

    chromosome = numpy.reshape(a=img_arr,
newshape=(functools.reduce(operator.mul, img_arr.shape)))

    return chromosome

def chromosome2img(chromosome, img_shape):

   img_arr = numpy.reshape(a=chromosome, newshape=img_shape)

   return img_arr
```

Initial Population

GA starts an initial population, which is a group of solutions (chromosomes) to the given problem. These solutions are randomly generated. The function not only returns a single chromosome but a group of chromosomes. It’s specified as the second argument n_individuals, which defaults to 8 if not specified:

```
def initial_population(img_shape, n_individuals=8):

   init_population = numpy.empty(shape=(n_individuals,

                                  functools.reduce(operator.mul,
img_shape)),
                                  dtype=numpy.uint8)

    for indv_num in range(n_individuals):

        # Randomly generating initial population chromosomes
genes values.

        init_population[indv_num, :] = numpy.random.random(

                                functools.reduce(operator.mul,
img_shape))*256

    return init_population
```

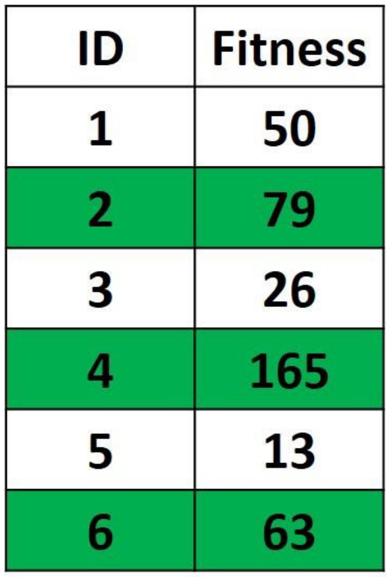
Fitness Calculation

GA starts with a number of bad solutions that are randomly generated. GA is based on the idea that evolving bad solutions might return better solutions. For every generation, the GA selects the best of the solutions in the current population and evolves them, hoping to return better solutions.This is done using a fitness function. Each chromosome is applied to this function and a number is returned. The higher the number, the better the solution. This is known as a maximization function.

```
def fitness_fun(target_chrom, indiv_chrom):

   quality = numpy.mean(numpy.abs(target_chrom-indiv_chrom))

   quality = numpy.sum(target_chrom) - quality

    return quality
```

Parent Selection

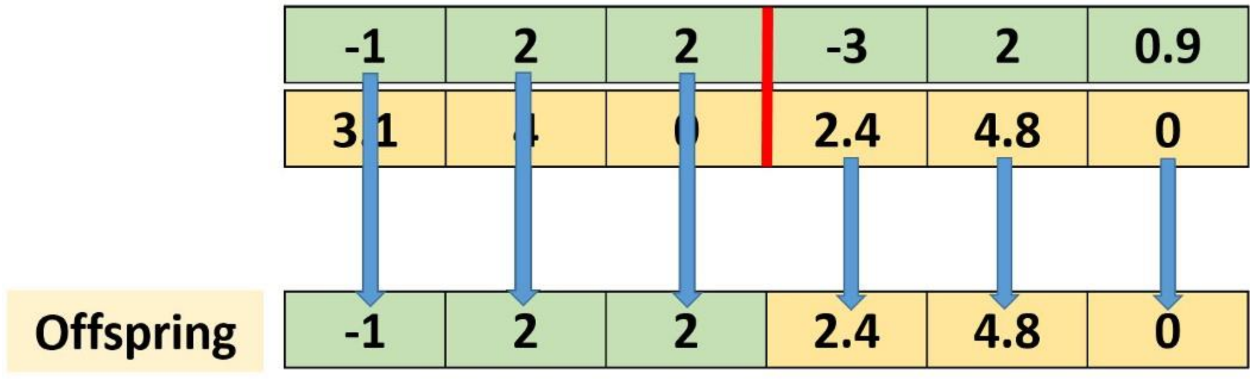
The parents selected from a given population are the best solutions within it. When we say “best solutions”, we’re referring to the solutions with the highest fitness values.



ID	Fitness
1	50
2	79
3	26
4	165
5	13
6	63

Crossover

The next figure shows an example in which single-point crossover is applied between 2 parents in order to create an offspring. Single-point crossover works by selecting a point at the chromosome. Genes before this point are selected from one parent, and genes after it are selected from the other parent. As a result, the offspring shares genes from both parents.



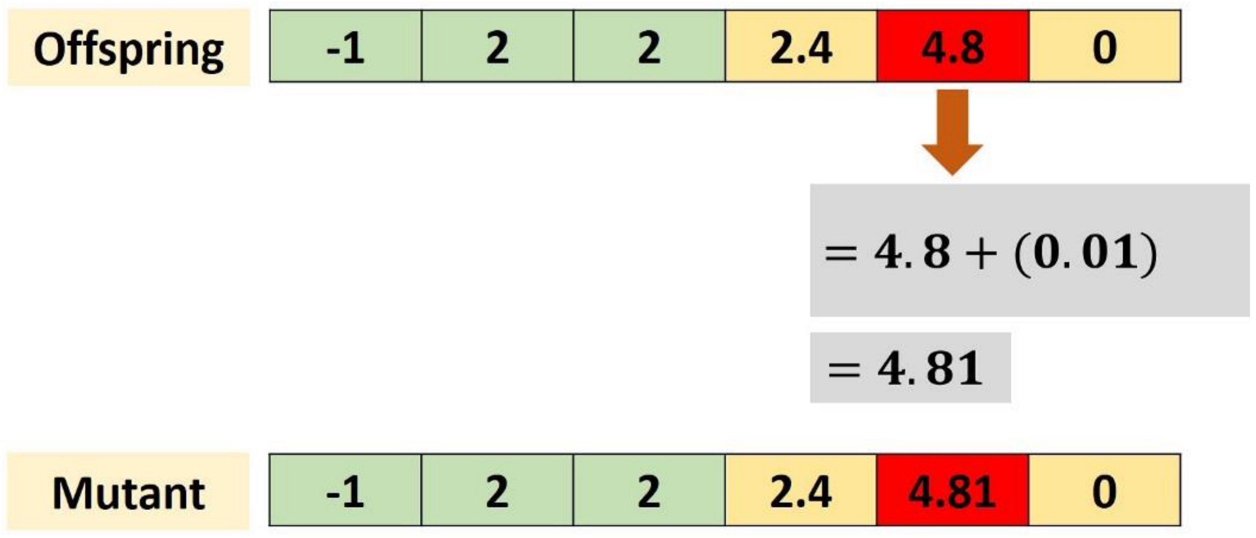
Code

```
def crossover(parents, img_shape, n_individuals=8):

   new_population = numpy.empty(shape=(n_individuals, functools.reduce(operator.mul,

img_shape)), dtype=numpy.uint8)
```

```
    #Previous parents (best elements).
    new_population[0:parents.shape[0], :] = parents

    # Getting how many offspring to be generated. If the population size is 8 and
number of parents mating is 4, then number of offspring to be generated is 4.
    num_newly_generated = n_individuals-parents.shape[0]
    # Getting all possible permutations of the selected parents.
    parents_permutations = list(itertools.permutations(iterable=numpy.arange(0,
parents.shape[0]), r=2))
    # Randomly selecting the parents permutations to generate the offspring.
    selected_permutations = random.sample(range(len(parents_permutations)),
                                          num_newly_generated)
```

Mutation

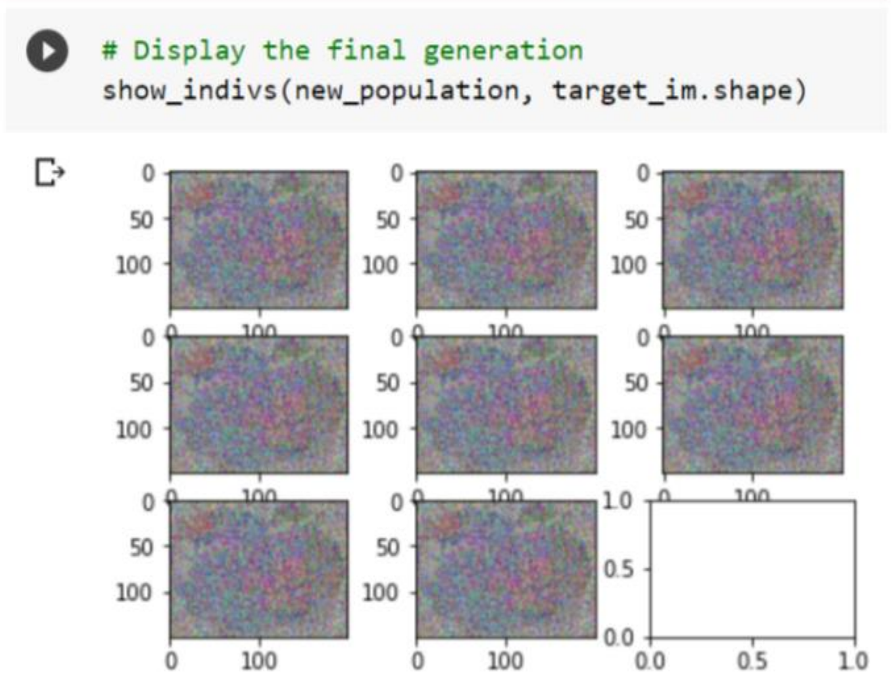
The mutation operation selects some genes within the chromosome and then randomly changes their values. It's implemented according to the mutation() function listed below. It accepts the population returned by the crossover() function, number of parents, and the percent of the genes to be changed.



Code

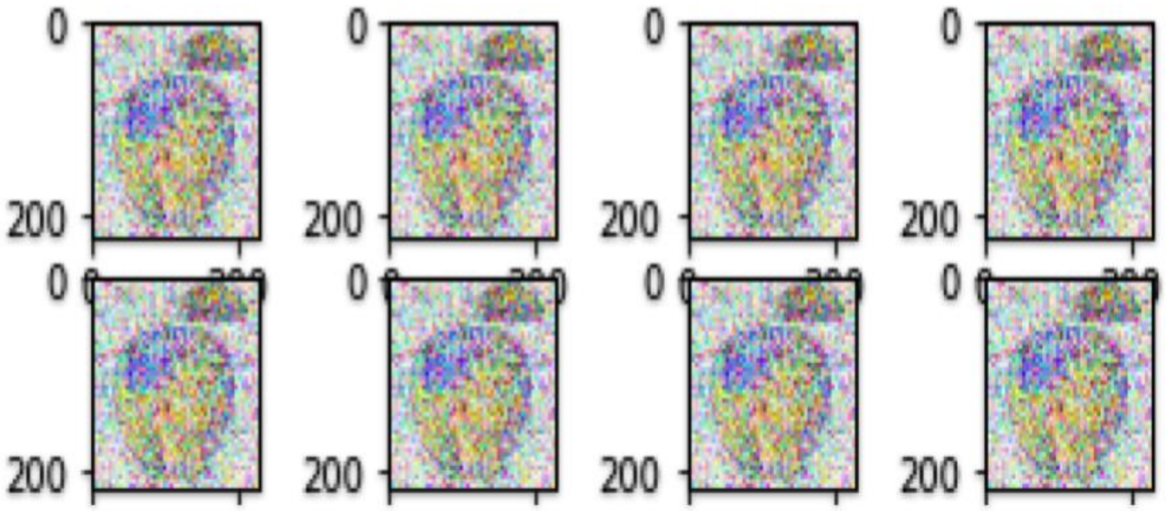
```
def mutation(population, num_parents_mating, mut_percent):


   for idx in range(num_parents_mating, population.shape[0]):

       # A predefined percent of genes are selected randomly.

       rand_idx =
numpy.uint32(numpy.random.random(size=numpy.uint32(mut_percent/1
00*population.shape[1]))*population.shape[1])

       # Changing the values of the selected genes randomly.

       new_values =
numpy.uint8(numpy.random.random(size=rand_idx.shape[0])*256)

       # Updating population after mutation.

       population[idx, rand_idx] = new_values

   return population
```

Results before fine tuning parameters



```
# Display the final generation
show_indivs(new_population, target_im.shape)
```

Increased Quality after fine tuning parameters



10. Comparison/ Performance Analysis

While in the contrast experiment on CNN model, data augmentation resulted in accuracy of 20.8%. However, the experiment on the Triplet model with initial dataset resulted in accuracy of 9.8%, where almost all the data trained with one sample cannot be recognized. This study therefore indicates that the benefits gained from data augmentation may work well on one-shot learning problems. Although the experiment demonstrates a great improvement, the results are subject to certain limitations. Also, the experiments were unable to explore approaches that work on other much larger and complex datasets. For fuzzy logic and genetic algorithm, the parameters were fine tuned to achieve more accuracy as seen in our results.

11. Conclusion

In this case study, we have experimented with models such as Conventional Neural Networks and Triplet Neural Networks for the text query of images. We did this in order to compare the performance of traditional neural networks and an improvised one. Traditional neural net takes in text queries passed after minimal preprocessing and resultant images are generated. In order to improve the image quality and validation of image search we have used fuzzy and triple neural nets. We have used a fuzzy algorithm for the image enhancement process. We have used a Genetic algorithm for producing a set of child images from a target image. In the process we have tried fine tuning the parameters and have come up with enhanced results.

12. References And Related Articles

- Reed, Scott, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. "Generative adversarial text to image synthesis." arXiv preprint arXiv:1605.05396 (2016).

- G. Raju, Madhu S. Nair, A fast and efficient color image enhancement method based on fuzzy-logic and histogram, AEU - International Journal of Electronics and Communications, Volume 68, Issue 3, 2014

- Zhou M., Tanimura Y., Nakada H. (2020) One-Shot Learning Using Triplet Network with kNN Classifier. In: Ohsawa Y. et al. (eds) Advances in Artificial Intelligence. JSAI 2019. Advances in Intelligent Systems and Computing, vol 1128. Springer, Cham. https://doi.org/10.1007/978-3-030-39878-1_21

- https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/

- https://www.sciencedirect.com/science/article/pii/S1665642314716098

- https://www.researchgate.net/publication/303626451_Query-by-Example_Image_Retrieval_in_Microsoft_SQL_Server

- https://link.springer.com/chapter/10.1007/11840930_3

----------------------------THE END----------------------------