THE UNET ARCHITECTURE FOR IMAGE SEGMENTATION

(ANALYSIS OF NEURAL NETWORK)

A Project

Presented to the

Faculty of

University of North Carolina,

Greensboro

In Partial Fulfillment

of the Requirements for the Degree

Master of (Arts/Science)

in

Computer Science

by

(Priyanka Budavi)

(December 2022)

ABSTRACT

Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain type of knowledge. Deep Learning can be supervised, un-supervised or semi-supervised. The architectures such as deep neural networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, medical image analysis, climate science etc. where they have produced results comparable to and in some cases surpassing human expert performance. Deep Learning specifically use Convolutional Neural Network where each level learns to transform its input data into a slightly more abstract and composite representation.

There are various architectures that convolutional neural network use but specifically we will use UNET in this project. The goal of the project is to understand the UNET architecture, implement image segmentation on a batch of images and analyze these images using clustering algorithms. The project is divided into three stages i.e., building the architecture for image segmentation, training the model, and then analyzing the layers to understand the data extracted from the convolutional neural network.

UNET architecture comprises of encoder, decoder, and intermediate layer with skip connections. The components together form a U-shape and hence is the name of the architecture. The intermediate layer obtained after training the model, will be used to analyze the layer and the reason for choosing this layer is that it stores lot of information compared to the other layers of UNET. With this information we check how the data is distributed and create clusters from the data using K-means clustering. Also, since the data is high dimensional, dimension reduction techniques is used to compare the data. The techniques used are PCA, TSNE, UMAP to visualize and compare. To determine the model is performing correctly we need a validation technique that will help us understand that the model trained is predicting correctly. For this, activation maps were implemented to see which part of the image the neural network is being focused at. Finally, when all is analyzed, using the metrics library we calculate the precision and recall quantifying the correctness of the trained model.

ACKNOWLEDGEMENTS

Accomplishment is not a one-day job, it is series of activities that are carried out one after the other for achieving the goal. This is a small gesture and gratitude that we can express for the immense help rendered to us by guide **Prof. Somya Mohanty** for suggesting, guiding, and exhibiting keen interest in this project work. Without his guidance and supervision right from the beginning it would not have been possible.

Table of Contents

CHAPTER ONE

INTRODUCTION

Deep Learning is a field within machine learning and Artificial intelligence that deals with algorithms inspired from human brain so as to help machines with intelligence without any external programming. Machine learning performed better in most of the areas but when it comes to image recognition, speech recognition, it does not perform well. To overcome this obstacle deep learning was introduced to deal with high dimensional data. We see that there are many applications of deep learning and are performing pretty well. One such application is virtual assistants i.e., Alexa and Siri, the suggestion to tag friends in photos on Facebook, product recommendations based on previous purchases in Amazon etc. which is being used in our day-to-day life. Based on the above assumptions, tried to apply this technique to images that were captured along the coastal areas to understand the effect of hurricanes on the buildings and to determine the damages caused to the buildings.

The scope of the project is to train a model to detect buildings and non-buildings from a batch of images and then extracting the information from the bottleneck layer to see what happens in the hidden layers of the neural network. As said, the deep neural network acts as a black box where only the input and output is known but what happens within the layers, how we can visualize the hidden layers and validate the model regarding its performance is what the project scope all about. To validate the model activation maps are implemented so that we get an idea of where the image is being focused at in a neural network.

1.1 Goals, Implication and Results

The project is divided into task, and we will discuss each task with the challenges involved and what was the expected result from the task. The following is the explanation for the same.

Task1: Data Collection

The first task of any analysis is the collection of the data. A good data will always be helpful to get a good accuracy of the model. In this project the data is a batch of images, which were collected along the coastal line to detect the buildings. The challenge here was, the images were of high resolution. So, we had to reduce the image size and then feed to the model for reducing the load on the laptop. This task was accomplished by using the image magic which helped us crop the images and store them in a folder.

Result: A total of five hundred images were cropped and the images were of size (1024, 1024). These images were fed as an input to the model.

Task 2: Creating the architecture of the model

The second task involved building the model. The architecture chosen is UNET and this was built sequentially by creating four blocks i.e., four for encoder and another four for decoder. This forms the U shape of the model as the name suggests. It is a sequential model and is a plain stack of layers where each layer has exactly one input tensor and one output tensor. The model was trained and saved for further use.

Result: A sequential model with stacked layers and a batch of images were fed as an input to the model. The model was then trained with the batch of images so to learn from the pixels and capture the features from the image.

Task 3: Extracting the bottleneck layer of the model and creating a high dimensional data

This task involved extracting the information from the bottleneck layer as this layer has most of the information of the input. Kera's is very well supported with extraction of features as there are multiple methods with which we can extract the information from the layers like weights and outputs of each layer. 85 images were considered for this task and the shape of the bottleneck layer was 128 *128 *512.

Result: A flattened tensor output of the bottleneck layer of size (85, 8388608). The 85 represents the number of images and 8388608 represent the features. This forms the high dimensional data.

Task 4: Reduction high dimensional data to low dimension using reduction techniques

This task involved reducing the data with size (85, 8388608) to low dimension where we can visualize what the layer has information all about. The techniques used are PCA, UMAP, TSNE. The data was reduced from (85, 8388608) to (85, 2) and values obtained were viewed on a scatterplot. This plot was a dynamic plot as when we hover on any data point in the plot, we will be able to view the corresponding images in the neural network. Also, the technique tries to cluster all the similar data to one side forming clusters. The challenge involved here was the getting the dynamic plots. To perform this task, first I stored the images on server but one of the drawbacks involved there was public storage.

The alternative that I found was connecting to the google colab and storing the image object in a data frame using the PIL library made the work easy.

Result: The interactive scatter plot with the images on hover of a datapoint. UMAP performed better as the clusters were far apart giving a clear picture of related items being grouped.

Task 5: Activation Maps and clustering of datapoints

This task involved getting the information from the layers as in where the image is being focused upon. Since every layer extract features, each layer focuses on various parts of the image. The challenge in this task was that getting the last layer information and then performing K-means so as to cluster the similar features together. A silhouette score was obtained to determine the separation of data points.

Result: Using Kmeans a scatterplot was developed to see the separation between the datapoints and then silhouette score was calculated to see the clusters are far apart.

Task 6: Evaluation of the model using metrics

To determine the accuracy of the model precision and recall was calculated. The precision answers the question “What proportion of positive identifications was actually correct? “ Recall attempts to answer the following question “What proportion of actual positives was identified correctly? “ . This helps us determine the quality of the model.

CHAPTER TWO
ARCHITECTURE AND IMPLEMENTATION

2. Architecture

As the name suggest, UNET is a U – shaped architecture which is made of two parts i.e., encoder and decoder. The architecture was first implemented on Biomedical Image Segmentation and from then on it became extremely popular. In layperson terms, human have the ability to see an image and guess what the image is all about. For example, a cat image is placed in front of a man, and he is asked to guess what the picture is all about. The man has been taught from the childhood to classify elements or recognize elements. Having that knowledge, the man was able to guess the image. Now, when it comes to machine it is difficult for it tell in a single glance, that's when UNET comes into picture. The machine learns, understand, and then guesses the input image provided.

As mentioned, it made of two parts encoders where the input image is passed, and this block analyzes each pixel of the image and tries to extract maximum features from it. The decoder then converts these bits and pieces and tries to output what the image output is all about. Between the encoder and decoder there exists a bottleneck layer which stores the maximum of the information. The project that I implemented is trying to understand this information extracted from the bottleneck layer and getting the meaningful insights from this data by using dimensionality reduction techniques to view better on a 2D space, getting the related or similar images by clustering and then analyzing the images as in which part of the image the layer is focusing using activation maps.

In project, UNET is processed block wise sequentially and each block has tasks to do. Below are the tasks carried out when the image is passed from the encoder to the decoder.

Conv2D: This a matrix multiplication where a filter is applied to each block of the images and slides through right to extract the features. Consider a face image, where filters are applied and each layer captures eyes, ears, nose, shape of the chin etc. This is capturing the edges of the Image and the feature extraction gets deeper as we move through the network of the neural layers.

MaxPooling: It is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. It reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer. The use of this technique is to overcome from overfitting problems and also reduces the computational cost. Below is the max pooling operation reducing the matrix from (4,4) to (2,2). Fig 1. Shows the reduced matrix after applying max pooling. In the figure we can see that from each block a max value is chosen and thus the matrix size decreases.

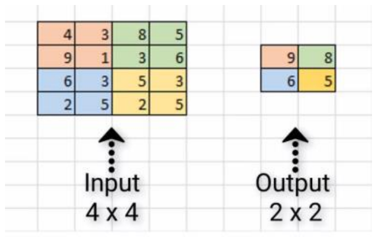
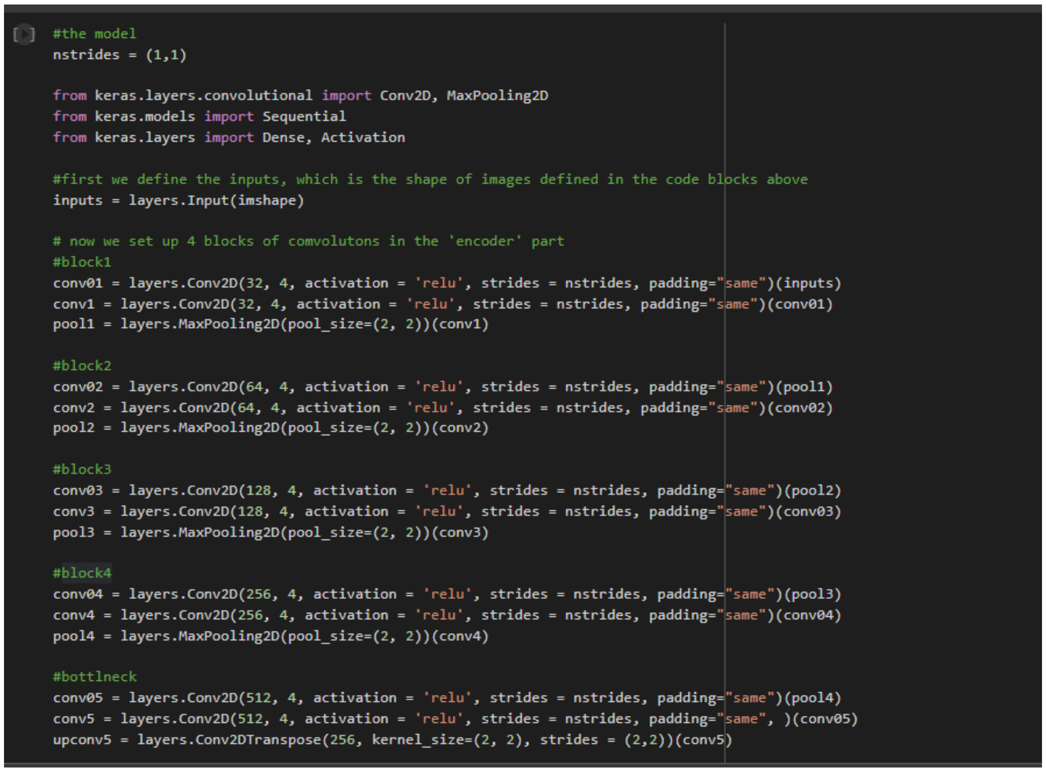


Fig 1. Max Pooling

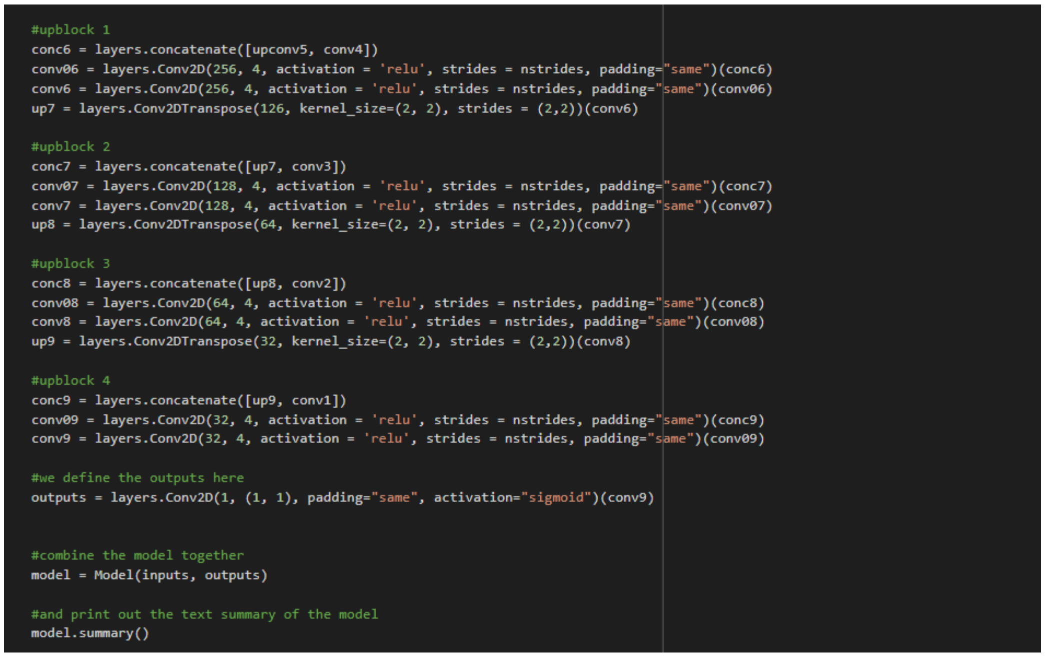
Activation function: The activation function is basically used in neural networks as we update the weights in the network by adding biases, we make sure that the error at the output is minimized and the result is same as the input. In order to achieve this backpropagation, we use activation function that add nonlinearity to the network and makes computation very efficient. Relu is the function used in the project.

Fig 2. And Fig 3. is the screenshot of the architecture with encoding and decoding blocks:



```
#the model
nstrides = (1,1)

from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.layers import Dense, Activation

#first we define the inputs, which is the shape of images defined in the code blocks above
inputs = layers.Input(imshape)

# now we set up 4 blocks of comvolutons in the 'encoder' part
#block1
conv01 = layers.Conv2D(32, 4, activation = 'relu', strides = nstrides, padding="same")(inputs)
conv1 = layers.Conv2D(32, 4, activation = 'relu', strides = nstrides, padding="same")(conv01)
pool1 = layers.MaxPooling2D(pool_size=(2, 2))(conv1)

#block2
conv02 = layers.Conv2D(64, 4, activation = 'relu', strides = nstrides, padding="same")(pool1)
conv2 = layers.Conv2D(64, 4, activation = 'relu', strides = nstrides, padding="same")(conv02)
pool2 = layers.MaxPooling2D(pool_size=(2, 2))(conv2)

#block3
conv03 = layers.Conv2D(128, 4, activation = 'relu', strides = nstrides, padding="same")(pool2)
conv3 = layers.Conv2D(128, 4, activation = 'relu', strides = nstrides, padding="same")(conv03)
pool3 = layers.MaxPooling2D(pool_size=(2, 2))(conv3)

#block4
conv04 = layers.Conv2D(256, 4, activation = 'relu', strides = nstrides, padding="same")(pool3)
conv4 = layers.Conv2D(256, 4, activation = 'relu', strides = nstrides, padding="same")(conv04)
pool4 = layers.MaxPooling2D(pool_size=(2, 2))(conv4)

#bottlneck
conv05 = layers.Conv2D(512, 4, activation = 'relu', strides = nstrides, padding="same")(pool4)
conv5 = layers.Conv2D(512, 4, activation = 'relu', strides = nstrides, padding="same", )(conv05)
upconv5 = layers.Conv2DTranspose(256, kernel_size=(2, 2), strides = (2,2))(conv5)
```

Fig 2 – Architecture of the model (UNET)



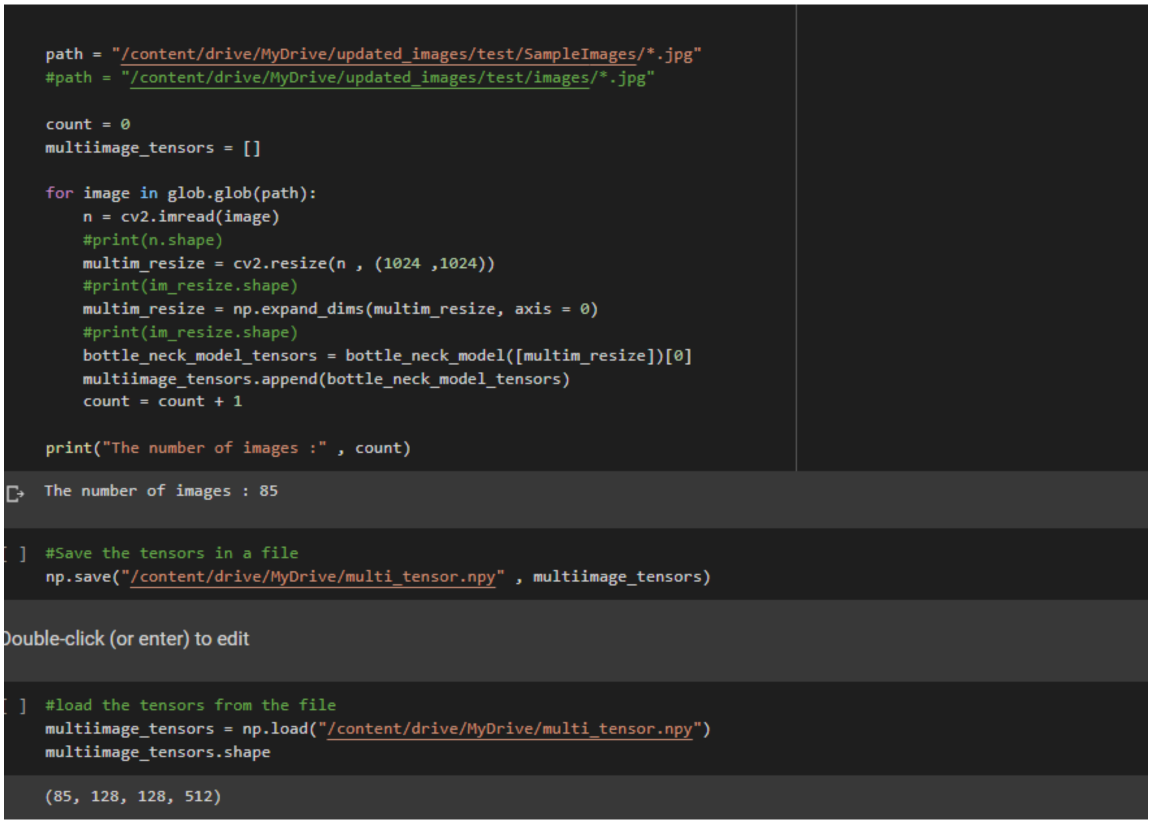
```
#upblock 1
conc6 = layers.concatenate([upconv5, conv4])
conv06 = layers.Conv2D(256, 4, activation = 'relu', strides = nstrides, padding="same")(conc6)
conv6 = layers.Conv2D(256, 4, activation = 'relu', strides = nstrides, padding="same")(conv06)
up7 = layers.Conv2DTranspose(126, kernel_size=(2, 2), strides = (2,2))(conv6)

#upblock 2
conc7 = layers.concatenate([up7, conv3])
conv07 = layers.Conv2D(128, 4, activation = 'relu', strides = nstrides, padding="same")(conc7)
conv7 = layers.Conv2D(128, 4, activation = 'relu', strides = nstrides, padding="same")(conv07)
up8 = layers.Conv2DTranspose(64, kernel_size=(2, 2), strides = (2,2))(conv7)

#upblock 3
conc8 = layers.concatenate([up8, conv2])
conv08 = layers.Conv2D(64, 4, activation = 'relu', strides = nstrides, padding="same")(conc8)
conv8 = layers.Conv2D(64, 4, activation = 'relu', strides = nstrides, padding="same")(conv08)
up9 = layers.Conv2DTranspose(32, kernel_size=(2, 2), strides = (2,2))(conv8)

#upblock 4
conc9 = layers.concatenate([up9, conv1])
conv09 = layers.Conv2D(32, 4, activation = 'relu', strides = nstrides, padding="same")(conc9)
conv9 = layers.Conv2D(32, 4, activation = 'relu', strides = nstrides, padding="same")(conv09)

#we define the outputs here
outputs = layers.Conv2D(1, (1, 1), padding="same", activation="sigmoid")(conv9)


#combine the model together
model = Model(inputs, outputs)

#and print out the text summary of the model
model.summary()
```

Fig 3. Architecture of the model (UNET)

2.1 Flatten the images to create a high dimensional dataset:

From the bottleneck layer we extract the output and create a model of this layer. The shape of the model is (128,128,512). Then a set of images is passed through this model to obtain a tensor of images. The output shape of this model is (85 ,128, 128, 512).

85 is the no. of images and the rest is the model shape. These tensors obtained should be flattened to get a high dimension array for 85 images. Fig 4 shows the code to generate multiple tensors with the bottleneck layer model. We are trying to expand the dimensions in the code because the image is of size (1024, 1024 ,3) but the model has four values and so we are expanding the dimensions.



```
path = "/content/drive/MyDrive/updated_images/test/SampleImages/*.jpg"
#path = "/content/drive/MyDrive/updated_images/test/images/*.jpg"

count = 0
multiimage_tensors = []

for image in glob.glob(path):
    n = cv2.imread(image)
    #print(n.shape)
    multim_resize = cv2.resize(n , (1024 ,1024))
    #print(im_resize.shape)
    multim_resize = np.expand_dims(multim_resize, axis = 0)
    #print(im_resize.shape)
    bottle_neck_model_tensors = bottle_neck_model([multim_resize])[0]
    multiimage_tensors.append(bottle_neck_model_tensors)
    count = count + 1

print("The number of images :" , count)
```

```
The number of images : 85
```

```
#Save the tensors in a file
np.save("/content/drive/MyDrive/multi_tensor.npy" , multiimage_tensors)
```

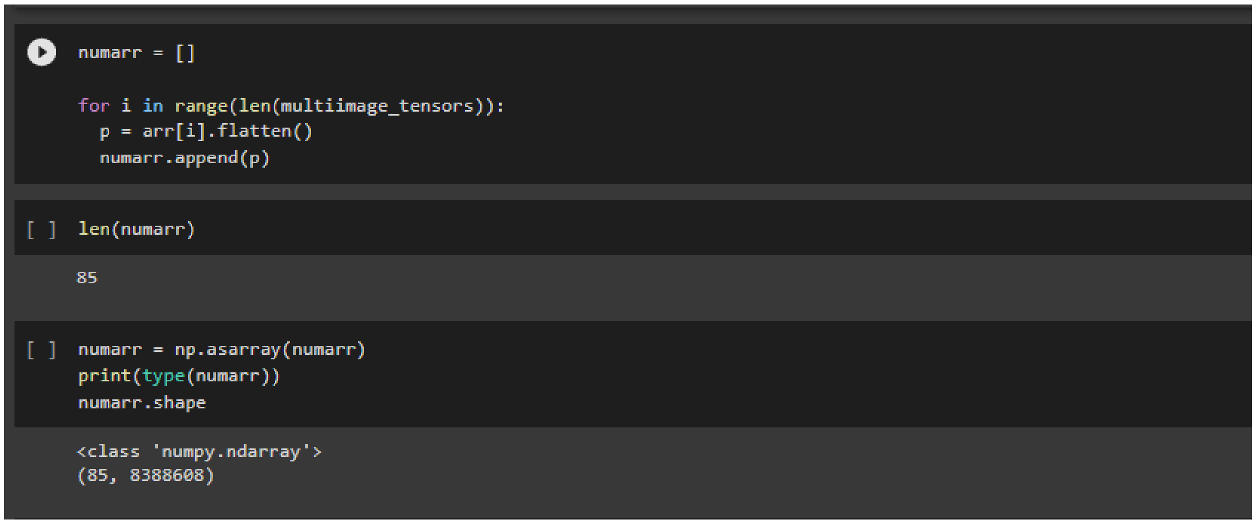
Double-click (or enter) to edit

```
#load the tensors from the file
multiimage_tensors = np.load("/content/drive/MyDrive/multi_tensor.npy")
multiimage_tensors.shape
```

```
(85, 128, 128, 512)
```

Fig 4

Using the flatten () method we flatten the tensors in a single array and Fig 5 is the code for the same. The final array is of shape (85,8388608).



```
numarr = []

for i in range(len(multiimage_tensors)):
  p = arr[i].flatten()
  numarr.append(p)
```

```
len(numarr)
```

```
85
```

```
numarr = np.asarray(numarr)
print(type(numarr))
numarr.shape
```

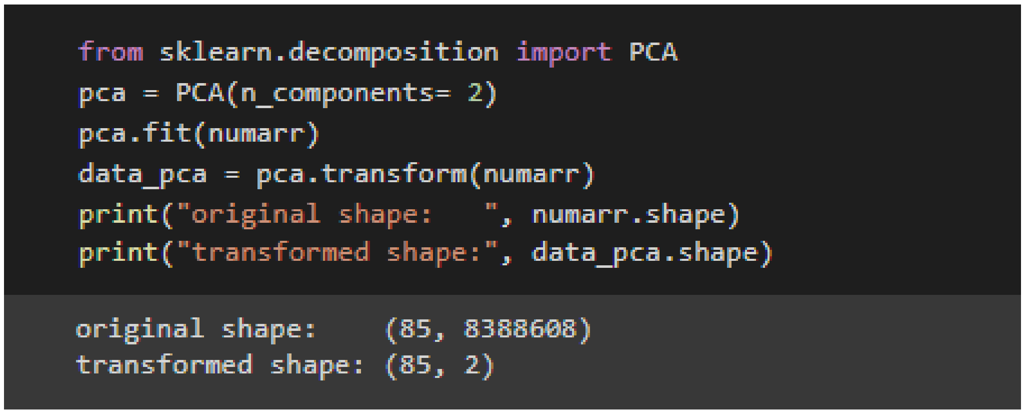
```
<class 'numpy.ndarray'>
(85, 8388608)
```

Fig 5

2.2 Dimension Reduction Technique:

The high dimension data obtained from the bottleneck layer is further reduced to lower dimension by using the dimension reduction Technique. The algorithm used here are the methods from scikit learn to perform PCA, UMAP, TSNE.

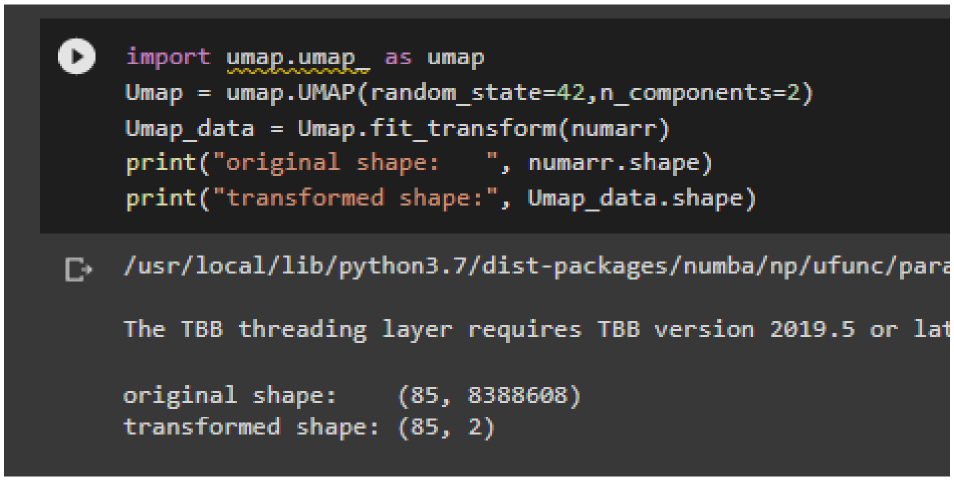
Fig 6 – 8 are the screenshots of method for PCA, UMAP and TSNE technique:



```
from sklearn.decomposition import PCA
pca = PCA(n_components= 2)
pca.fit(numarr)
data_pca = pca.transform(numarr)
print("original shape:    ", numarr.shape)
print("transformed shape:", data_pca.shape)
```

```
original shape:    (85, 8388608)
transformed shape: (85, 2)
```

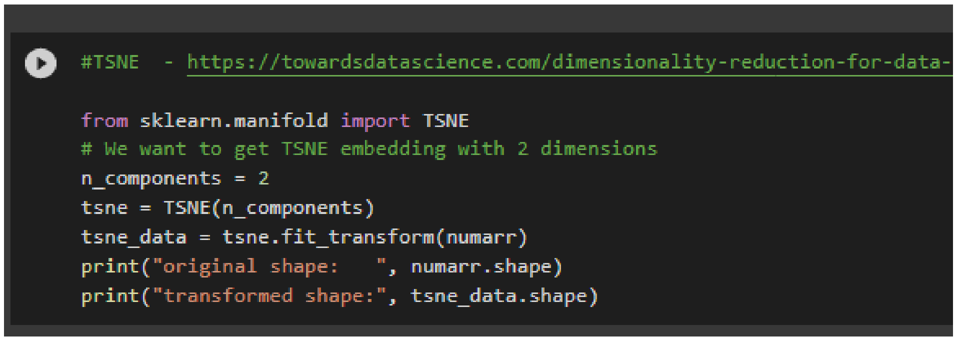
Fig 6



```
import umap.umap_ as umap
Umap = umap.UMAP(random_state=42,n_components=2)
Umap_data = Umap.fit_transform(numarr)
print("original shape:    ", numarr.shape)
print("transformed shape:", Umap_data.shape)
```

```
/usr/local/lib/python3.7/dist-packages/numba/np/ufunc/para

The TBB threading layer requires TBB version 2019.5 or lat

original shape:    (85, 8388608)
transformed shape: (85, 2)
```

Fig 7



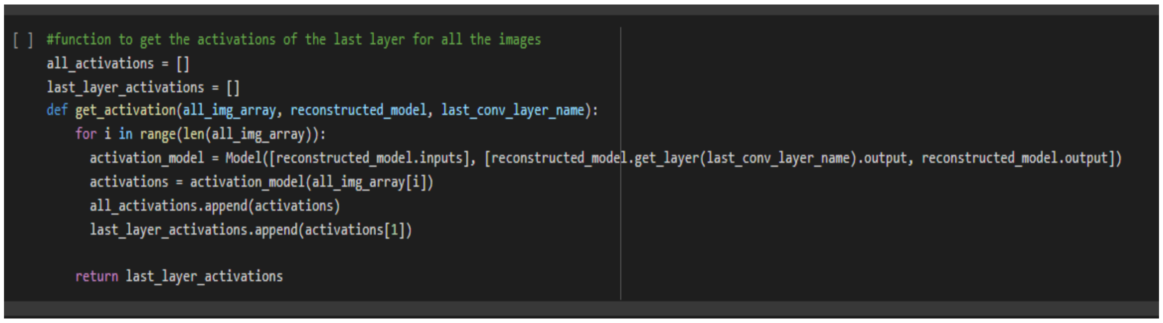
```
#TSNE  - https://towardsdatascience.com/dimensionality-reduction-for-data-

from sklearn.manifold import TSNE
# We want to get TSNE embedding with 2 dimensions
n_components = 2
tsne = TSNE(n_components)
tsne_data = tsne.fit_transform(numarr)
print("original shape:   ", numarr.shape)
print("transformed shape:", tsne_data.shape)
```

Fig 8

With the above methods a 2-dimensional scatterplot was plotted which was dynamic i.e on hover of a data point on the scatterplot, a corresponding image was displayed. The images were stored in the drive and using PIL library for image, the image object was stored in the data frame along with the datapoints. Using this data frame, on hover of a point we will be able to view the corresponding images from the bottleneck layer. When we plot a graph, we see that the images with similar features are grouped together.

2.3 Activation Maps

Every pixel in the input layer is applied with the filter and here filter meaning matrix multiplication of values. This matrix multiplication leads to a feature map, which store every information of the image. The feature map is then used to visualize the image in the convolution layer as in which area of the image is being focused by the layer. To get the class activation maps its required to compute the gradients, average the gradients and then visualize them using GRADCAM algorithm but in this project, I derived the activations of the last layer and formed a model of the same. Then the images that were used to perform the bottleneck analysis were passed to the created model. The result included the image tensors of the model shape. Fig 9 is the screenshot of the code as how the activation maps were generated. The last_layer_activation is an array with all the image activations stored in it which can further be visualized by converting the matrix to image and saving these images on the drive.

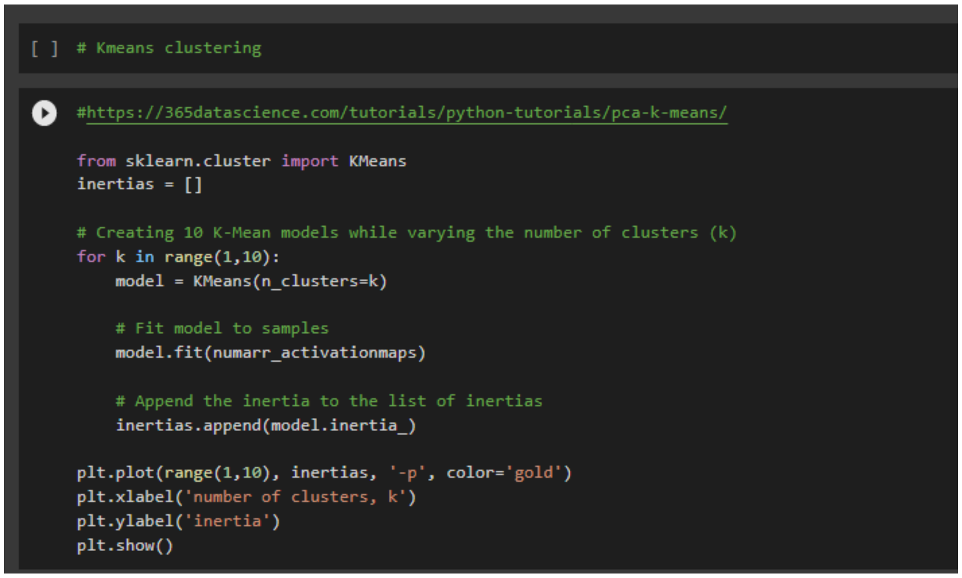


```
[ ] #function to get the activations of the last layer for all the images
    all_activations = []
    last_layer_activations = []
    def get_activation(all_img_array, reconstructed_model, last_conv_layer_name):
        for i in range(len(all_img_array)):
          activation_model = Model([reconstructed_model.inputs], [reconstructed_model.get_layer(last_conv_layer_name).output, reconstructed_model.output])
          activations = activation_model(all_img_array[i])
          all_activations.append(activations)
          last_layer_activations.append(activations[1])

        return last_layer_activations
```

Fig 9

2.4 Kmeans

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. The objective of K-means is simple i.e group similar data points together and discover the underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset. To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative calculations to optimize the positions of the centroids. Further, a silhouette score is calculated to check the quality of the model created. The metrics used in this project are Precision and Recall. Fig 10 is the code to perform Kmeans clustering. In the below code the inertia is the sum of the squares of the distances from the centroid. Now the question arises of choosing the number of clusters. To check for the optimal K value, we need to plot the elbow graph and the point where the inertia starts to decrease. This value where it starts decreasing is considered as an optimal K value. Fig 11 shows the elbow curve with optimal K value to be considered for this model.



```
[ ] # Kmeans clustering

#https://365datascience.com/tutorials/python-tutorials/pca-k-means/

from sklearn.cluster import KMeans
inertias = []

# Creating 10 K-Mean models while varying the number of clusters (k)
for k in range(1,10):
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(numarr_activationmaps)

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(range(1,10), inertias, '-p', color='gold')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.show()
```

Fig 10. Code to determine the optimal K value

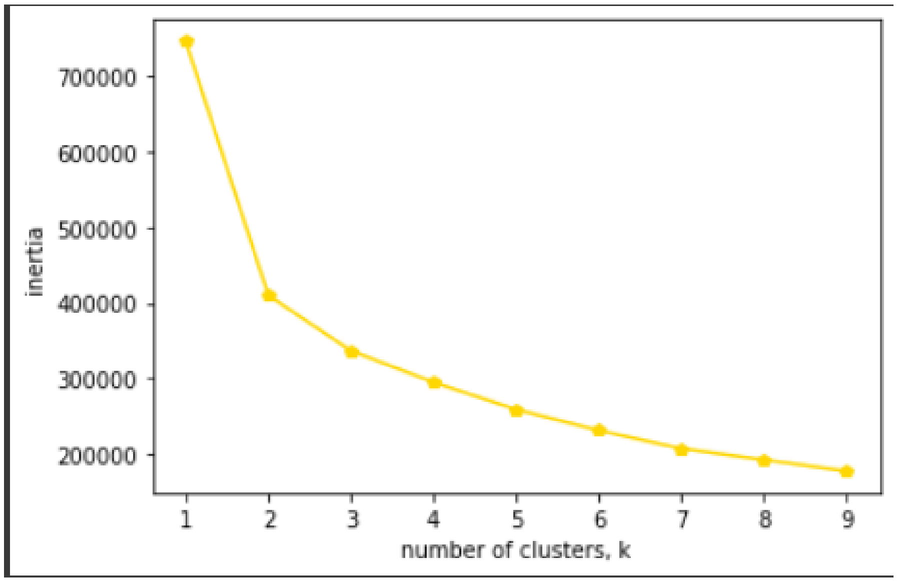


Fig 11. Elbow curve to determine the optimal K value

2.5 Metrics

When a model is trained it is necessary to validate the model regarding its quality. Using metrics, we can calculate the accuracy of the model. In this project, precision and recall were the two metrics that helped us determine the quality of the model.

Precision is a measure of how many of the positive predictions made are correct (true positives). The formula for it is:

Precision Score = TP / (FP + TP)

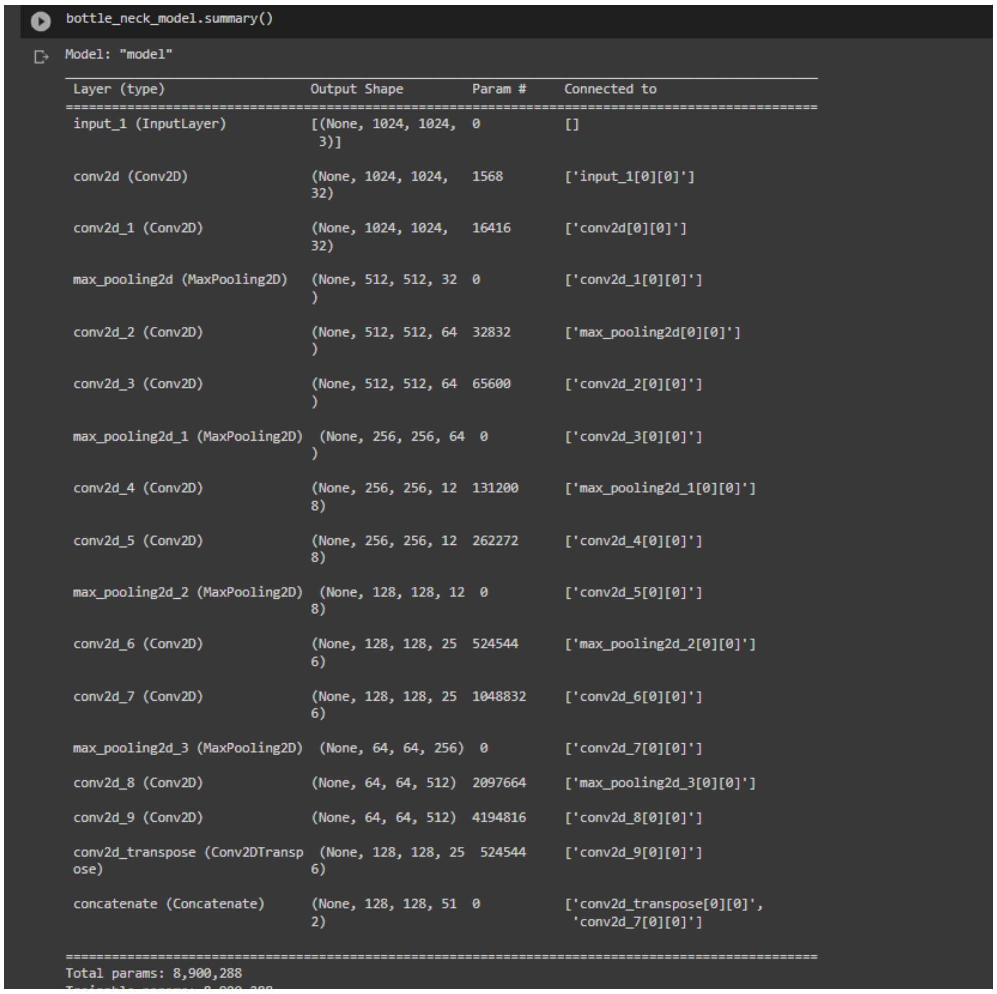
Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data.

Recall Score = TP / (FN + TP)

CHAPTER THREE
RESULTS AND DISCUSSION

3. Architecture of the Intermediate Layer

Fig 11 is the screenshot of the trained model. The input fed is of size (1024,1024, 3) and the summary in the screenshot displays the dimensions which is seen increasing in the encoding block and later it is found to decrease in the decoder block. The final size is 128 * 128 *512 where 512 is the dimensions of the image of size 128 * 128. This is the result of the bottleneck layer after training the entire model which will be further used for analysis.

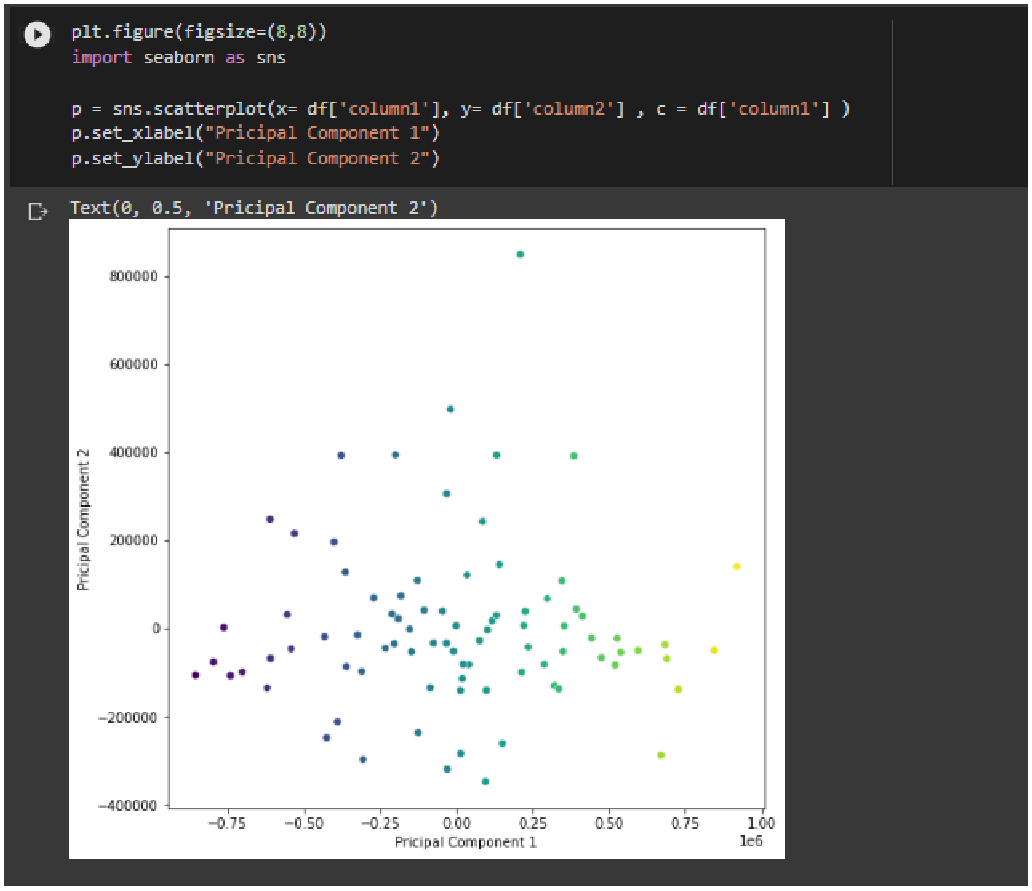


```
bottle_neck_model.summary()

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 1024, 1024,  0           []
                                 3)]

 conv2d (Conv2D)                (None, 1024, 1024,   1568        ['input_1[0][0]']
                                32)

 conv2d_1 (Conv2D)              (None, 1024, 1024,   16416       ['conv2d[0][0]']
                                32)

 max_pooling2d (MaxPooling2D)   (None, 512, 512, 32  0           ['conv2d_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 512, 512, 64  32832       ['max_pooling2d[0][0]']
                                )

 conv2d_3 (Conv2D)              (None, 512, 512, 64  65600       ['conv2d_2[0][0]']
                                )

 max_pooling2d_1 (MaxPooling2D)  (None, 256, 256, 64  0          ['conv2d_3[0][0]']
                                )

 conv2d_4 (Conv2D)              (None, 256, 256, 12  131200      ['max_pooling2d_1[0][0]']
                                8)

 conv2d_5 (Conv2D)              (None, 256, 256, 12  262272      ['conv2d_4[0][0]']
                                8)

 max_pooling2d_2 (MaxPooling2D)  (None, 128, 128, 12  0          ['conv2d_5[0][0]']
                                8)

 conv2d_6 (Conv2D)              (None, 128, 128, 25  524544      ['max_pooling2d_2[0][0]']
                                6)

 conv2d_7 (Conv2D)              (None, 128, 128, 25  1048832     ['conv2d_6[0][0]']
                                6)

 max_pooling2d_3 (MaxPooling2D)  (None, 64, 64, 256)  0          ['conv2d_7[0][0]']

 conv2d_8 (Conv2D)              (None, 64, 64, 512)  2097664     ['max_pooling2d_3[0][0]']

 conv2d_9 (Conv2D)              (None, 64, 64, 512)  4194816     ['conv2d_8[0][0]']

 conv2d_transpose (Conv2DTransp  (None, 128, 128, 25  524544     ['conv2d_9[0][0]']
 ose)                           6)

 concatenate (Concatenate)      (None, 128, 128, 51  0           ['conv2d_transpose[0][0]',
                                2)                                'conv2d_7[0][0]']

==================================================================================================
Total params: 8,900,288
```

Fig 11 – Summary of the bottleneck model

3.1 PCA

PCA was not able to do an excellent job in differentiating the points as the main aim of PCA is to find directions and get a linear projection. Suppose there was a non-linear data it would not be able to handle such data. Also due to the linear relation its prone to outliers and that is the drawback of PCA. So, in the below figure we see that the data does not have high variance to separate the points. Thus, we need to look for other dimension reduction techniques to find the best variance to spread the data points into clusters. Also, a dynamic plot with hover on a point displays the corresponding image on graph. Here the data points average is calculated with the X-axis and Y-axis for the two features that we choose. Then the point of center is calculated using this data. Once the center is obtained, we shift the center to the origin and try to find the best fit line by minimizing the distance from the points to the line and accordingly the similar points are Grouped together by finding the sum of squares distance and once the largest distance is obtained that forms a principal component. Thus, there are two principal components in the below graph.



```
plt.figure(figsize=(8,8))
import seaborn as sns

p = sns.scatterplot(x= df['column1'], y= df['column2'] , c = df['column1'] )
p.set_xlabel("Pricipal Component 1")
p.set_ylabel("Pricipal Component 2")
```

```
Text(0, 0.5, 'Pricipal Component 2')
```

Fig 12. Scatter plot of the PCA

Below is the scatterplot with hover on a data point to display the image

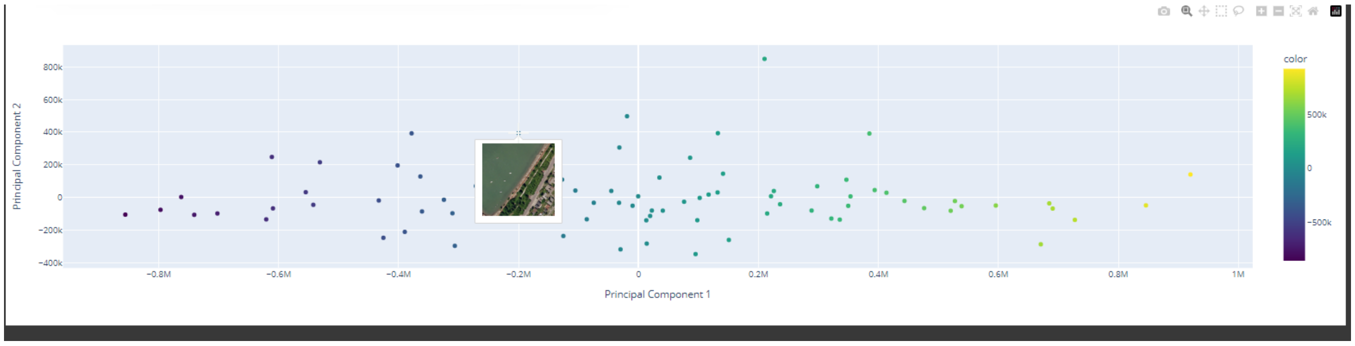


Fig 13. Display of image on hover of a point

3.2 TSNE

T-**SNE** also known as T- distributed stochastic neighbor embedding does a better job as compared to PCA. When it comes to visualizing the different patterns of the clusters all the similar datapoints are clustered together. The main focus of TSNE is retaining the structure of neighboring points as its only concerned with it. Since TSNE deals with random probability, the similar points are assigned to a higher probability and lower for unsimilar ones. The problem with TSNE is that when you project the data points on X or Y axis, the points are cluttered around the same area losing the structure of the points. In order to not lose the structure, all similar points are pulled towards each other and the non-similar once are repelled on a 1D plane till clusters with similar points are not formed. Thus, in the Fig 14 we see clusters of similar points being grouped together. Also, a dynamic plot with image displayed on hover can be seen in Fig 15.

From the below screenshot we see that the variance is high and similar points are grouped together.

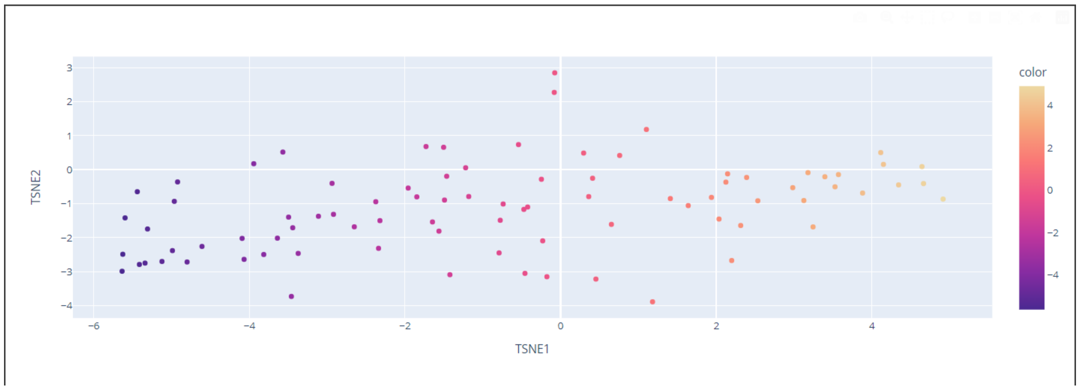


Fig 14. Scatter plot of the TSNE

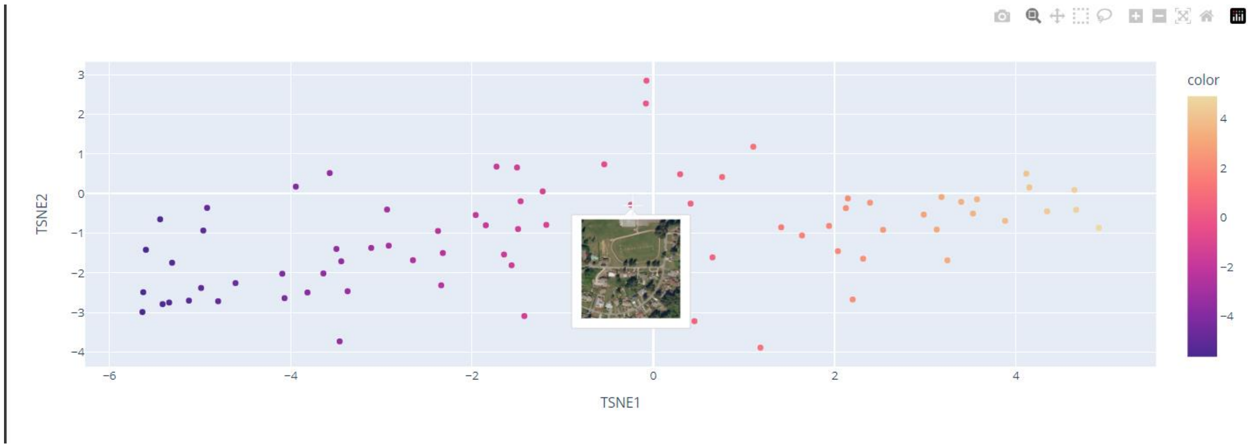


Fig 15. Display of image on hover of a point

3.3 UMAP

UMAP outperformed t-SNE and PCA, if we look and compare all the plots, we can see mini clusters and they are well separated. This algorithm is very effective for visualizing clusters or groups of data points whose dimensionality is high. UMAP is much faster than t-SNE and definitely solves the drawbacks of PCA.

In the below screenshot we see that the points are well separated and has performed better than PCA and TSNE.

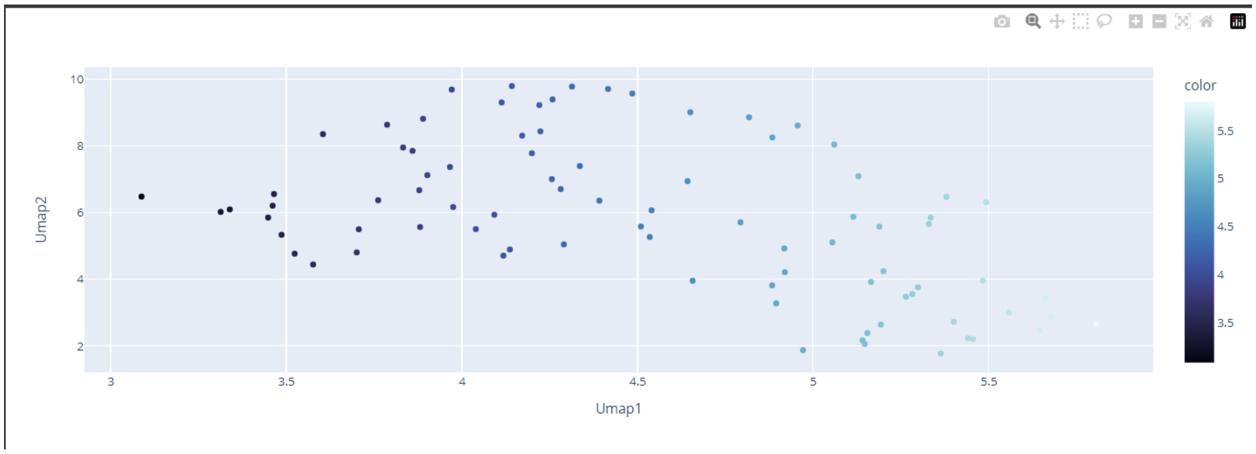


Fig 16. Scatter plot of the UMAP

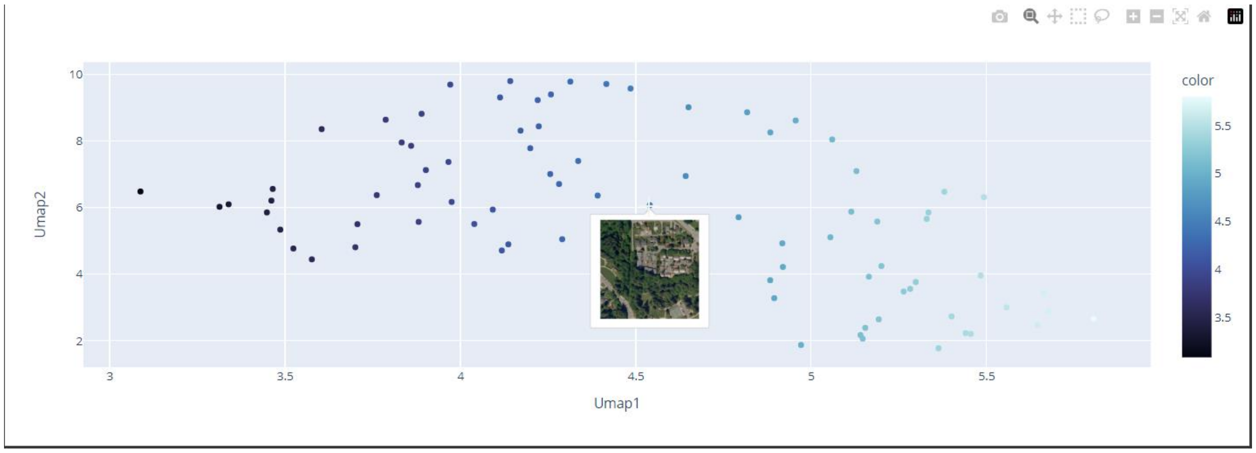


Fig 17. Display of image on hover of a point

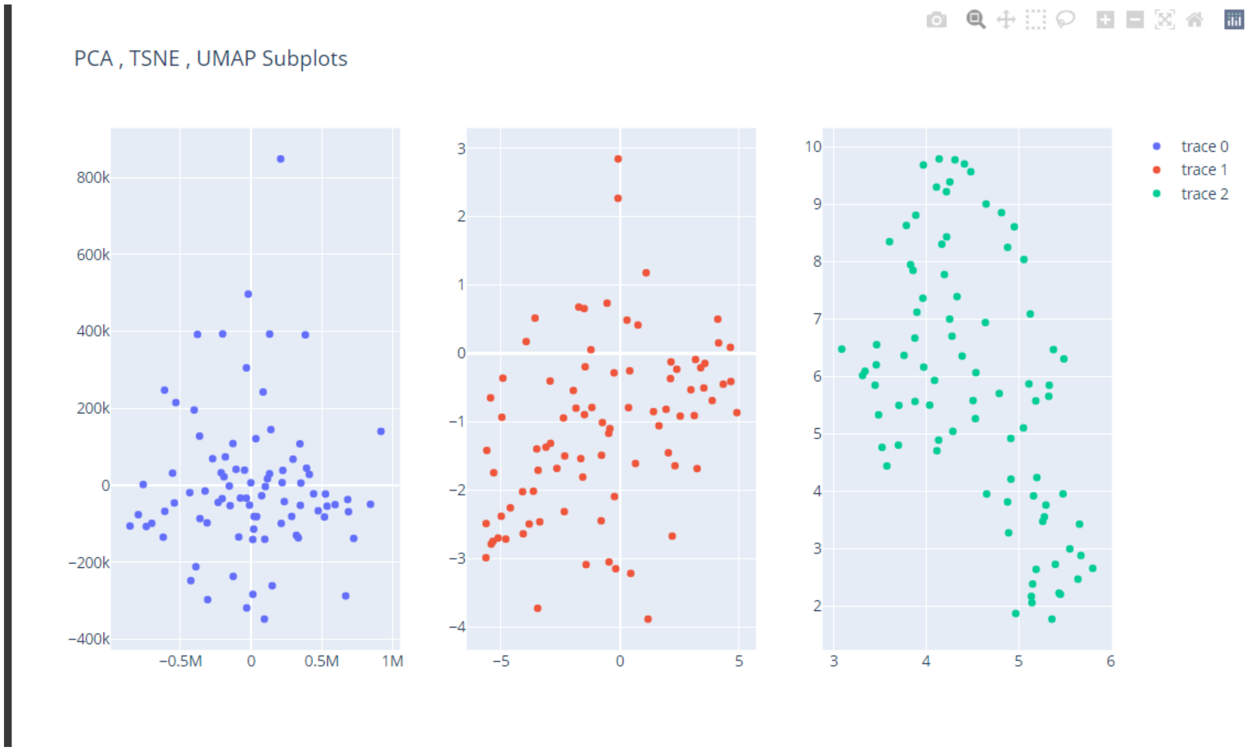


Fig18. Comparing PCA, TSNE and UNET

3.4 Activation Maps

The activation maps is a technique to understand where the model is focusing at in the image. The last layer is considered for visualization and the images obtained from the activation maps were saved in the google drive. Fig 19. shows the images of the activation maps technique where we create a activation model of the last layer and then get the activations of the last layer by passing image array to the model. These activations array is later converted from an array to an image. In the below figure we see that the area of focus is bright yellow which gives us information that the neural network is focusing on that particular area of the image. The activations array of the images is of the size (85, 1, 1024, 1024, 1) where 85 is the no. of images and the rest is the shape of the activation model (last layer). Thus, a machine can visualize what's happening in the hidden layers and get a clear picture of the model if its predicting the images correctly or not.

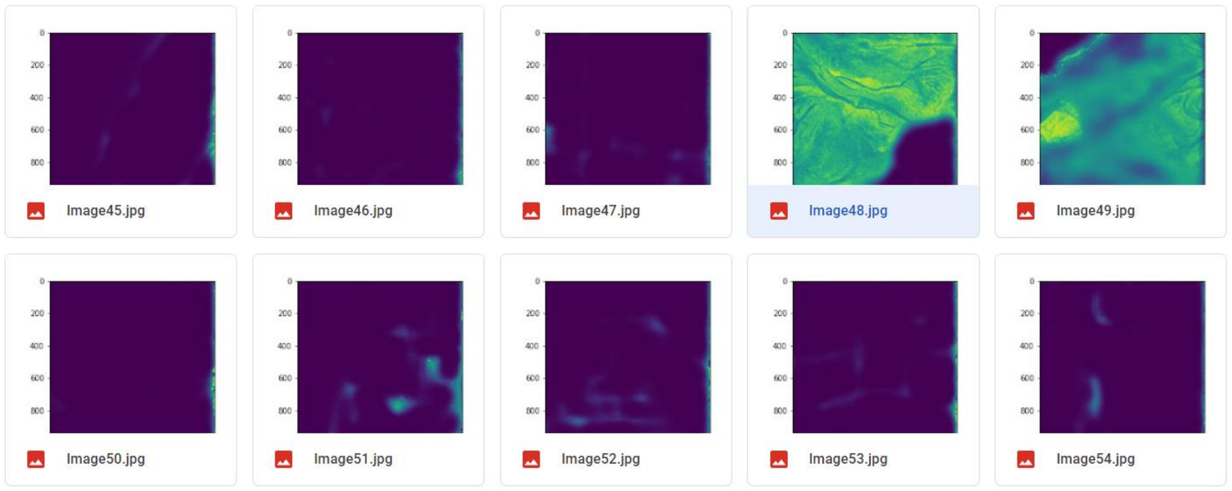
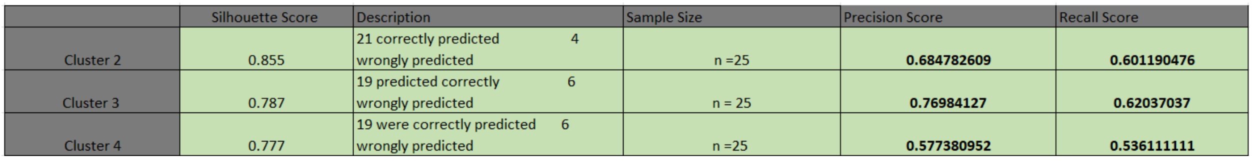


Fig 19. Activation Maps Images

3.5 Metric Comparison and conclusion of the results

To determine the quality of the model its necessary to evaluate the performance of the model and for this precision and recall is used. The model performed well with Kmeans clustering by grouping the similar points together. The clustering with different cluster values is checked starting from 2, 3 and 4. When all the clusters were compared, the optimal cluster size was found to be three and with this a better silhouette score was obtained. Later the same technique was applied for multiple images. According to Fig 20. the silhouette score, precision score and recall score were calculated for twenty-five images with cluster size 2, 3 and 4. The scores are compared, and we can see in the table that cluster three was better performing with silhouette score as approx. 79%, precision as 77% and recall as 62%. The same method was applied for larger size with images equal to 50 and the results seemed the same i.e., three clusters performed well with great scores.



	Silhouette Score	Description	Sample Size	Precision Score	Recall Score
Cluster 2	0.855	21 correctly predicted 4 wrongly predicted	n =25	**0.684782609**	**0.601190476**
Cluster 3	0.787	19 predicted correctly 6 wrongly predicted	n = 25	**0.76984127**	**0.62037037**
Cluster 4	0.777	19 were correctly predicted 6 wrongly predicted	n =25	**0.577380952**	**0.536111111**

Fig 20. Model Performance

Thus, from the above we can conclude that the model trained was performing better with a good precision and recall score. The sample with cluster size equal 3, predicted 19 images correctly and 6 were wrongly predicted. Hence, I can say that we were able to extract the bottleneck layer information then visualize this layer using dynamic scatter plots and by using clustering algorithm we were able to form clusters with similar data points. Lastly, the model performance using metrics talks about the quality of the model.

CHAPTER FOUR
CONCLUSION

The aim of the project is to perform analysis of the bottleneck layer or the intermediate layer as most of the information is available in this part of the neural network. So, the visualization of the bottleneck layer by using a batch of images and building a model of the intermediate layer is something that helped understand what happens inside a neural network when a set of images are passed. With the dynamic plots created we were able to determine the images corresponding to the datapoints and due to the dimension reduction techniques, we were able to group all buildings and non-buildings separately on the scatter plot. Also, when an image is passed in the neural network, which part of the images were being focused on the neural network, we were able to do analysis on that. The best part of the research is we were able to perform tasks which is not visible to the human eye and using the machine learning techniques we were able to understand how the neural network processes an image. This is because neural network acts as a black box and we do not have any idea of what going on inside the hidden layers. Thus, using clustering algorithm, we were able to form clusters of classes that is building, non-building and if it does not belong to this class a new class is formed based on the images. From results, we saw that the model performed well by considering the last layer information and the precision was found to be approx. 79% which is a good metric for evaluating the performance of the model. With three clusters the performance for fifty images was good and we can conclude that with the last layer information we were able to evaluate the performance of the model. The model predicted maximum images as correct and very few as images were labelled incorrect. This was verified by manually checking the images with the data frame generated from Kmeans clustering. Hence, I conclude that the analysis using UNET turned out to be good and received the results as expected.

REFERENCES

[1] https://stackoverflow.com/questions/70050831/plotly-dash-scatter-plot-pointnumber-is-assigned-to-multiple-points-in-hover-da

[2] https://community.plotly.com/t/how-to-embed-images-into-a-dash-app/61839

[3] https://365datascience.com/tutorials/python-tutorials/pca-k-means/

[4] https://www.youtube.com/watch?v=Y6NRFru9WLg

[5] https://stackoverflow.com/questions/63297838/how-can-i-obtain-the-output-of-an-intermediate-layer-feature-extraction

[6] https://stackoverflow.com/questions/49186905/loading-images-in-google-colab

[7] https://www.geeksforgeeks.org/python-pil-image-open-method/

[8] https://dash.plotly.com/dash-enterprise/static-assets

[9] https://stackoverflow.com/questions/70050831/plotly-dash-scatter-plot-pointnumber-is-assigned-to-multiple-points-in-hover-da

[10] https://community.plotly.com/t/how-to-embed-images-into-a-dash-app/61839

[11] https://towardsdatascience.com/dimensionality-reduction-for-data-visualization-pca-vs-tsne-vs-umap-be4aa7b1cb29

[12] https://towardsdatascience.com/dimensionality-reduction-for-data-visualization-pca-vs-tsne-vs-umap-be4aa7b1cb29

[13] https://plotly.com/python/subplots/

[14] https://stackoverflow.com/questions/67102278/expected-ndim-3-found-ndim-4-when-using-k-function-in-keras-backend-to-get-i

[15] https://stackoverflow.com/questions/66079482/why-would-we-expand-a-numpy-array-along-axis-0

[16] https://stackoverflow.com/questions/67856458/how-to-append-multiple-image-into-a-ndarray

[17] https://www.youtube.com/watch?v=uFbDWu0tDrE