1 Problem Statement

Implement a simple encoder-decoder neural network with one hidden layer using the images of George Bush from the LFW Face database. Use the MSE loss and 70:20:10 train:val:test split to train the network. Compare the reconstruction error with a simple (linear)PCA (use the same split. Vary the number of the top few eigen vectors during reconstruction). Include some qualitative results/analysis in the submission report.

2 Theory

The most common type of machine learning models are discriminative. These models recognize the input data and then take appropriate action. For a classification task, a discriminative model learns how to differentiate between various different classes. Based on the model's learning about the properties of each class, it classifies a new input sample to the appropriate label.
If a machine/deep learning model is to recognize the following image, it may understand that it consists of three main elements: a rectangle, a line, and a dot. When another input image has features which resemble these elements, then it should also be recognized as a warning sign.

Discriminative models are not clever enough to draw new images even if they know the structure of these images. Let's take another example to make things clearer.

In contrast with discriminative models, there is another group called generative models which can create new images. For a given input image, the output of a discriminative model is a class label; the output of a generative model is an image of the same size and similar appearance as the input image.

One of the simplest generative models is the autoencoder (AE for short). Autoencoders are a deep neural network model that can take in data, propagate it through a number of layers to condense and understand its structure, and finally generate that data again. In this tutorial we'll consider how this works for image data in particular. To accomplish this task an autoencoder(Figure [3]) uses two different types of networks. The first is called an encoder(Figure [1], and the other is the decoder(Figure [2]). The decoder is just a reflection of the layers inside the encoder. Let's clarify how this works.

The job of the encoder is to accept the original data (e.g. an image) that could have two or more dimensions and generate a single 1-D vector that represents the entire image. The number of elements in the 1-D vector varies based on the task being solved. It could have 1 or more elements. The fewer elements in the vector, the more complexity in reproducing the original image accurately. The 1-D vector generated by the encoder from its last layer is then fed to the decoder. The job of the decoder is to reconstruct the original image with the highest possible quality. The decoder is just a reflection of the encoder. According to the encoder architecture in the previous figure,

the architecture of the decoder is given in the next figure.

The loss is calculated by comparing the original and reconstructed images(on my case, I have used MSE (Mean Square Error loss), i.e. as a metric calculating the extent difference between the pixels in the 2 images. Lesser the value better is the solution

3 Building Autoencoder

Keras is a powerful tool for building machine and deep learning models because it's simple and abstracted, so in little code you can achieve great results. Keras has three ways for building a model:
a. Sequential API
b. Functional API
c. Model Subclassing
The three ways differ in the level of customization allowed. The sequential API allows you to build sequential models, but it is less customizable compared to the other two types. The output of each layer in the model is only connected to a single layer. The functional API is simple, very similar to the sequential API, and also supports additional features such as the ability to connect the output of a single layer to multiple layers. The last option for building a Keras model is model sub-classing, which is fully-customizable but also very complex. Now I will focus on using the functional API for building the autoencoder. I built three models: one for the encoder, another for the decoder, and yet another for the complete autoencoder. Why do we build a model for both the encoder and the decoder? We do this in case you want to explore each model separately. For instance, we can use the model of the encoder to visualize the 1-D vector representing each input image, and this might help you to know whether it's a good representation of the image or not. With the decoder we'll be able to test whether good representations are being created from the 1-D vectors, assuming they are well-encoded (i.e. better for debugging purposes) Finally, by building a model for the entire autoencoder we can easily use it end-to-end by feeding it the original image and receiving the output image directly.
Below I have mentioned the block diagram(Refer Figure [1],[2] and [3]) of the model that I have created for the Image reconstruction.

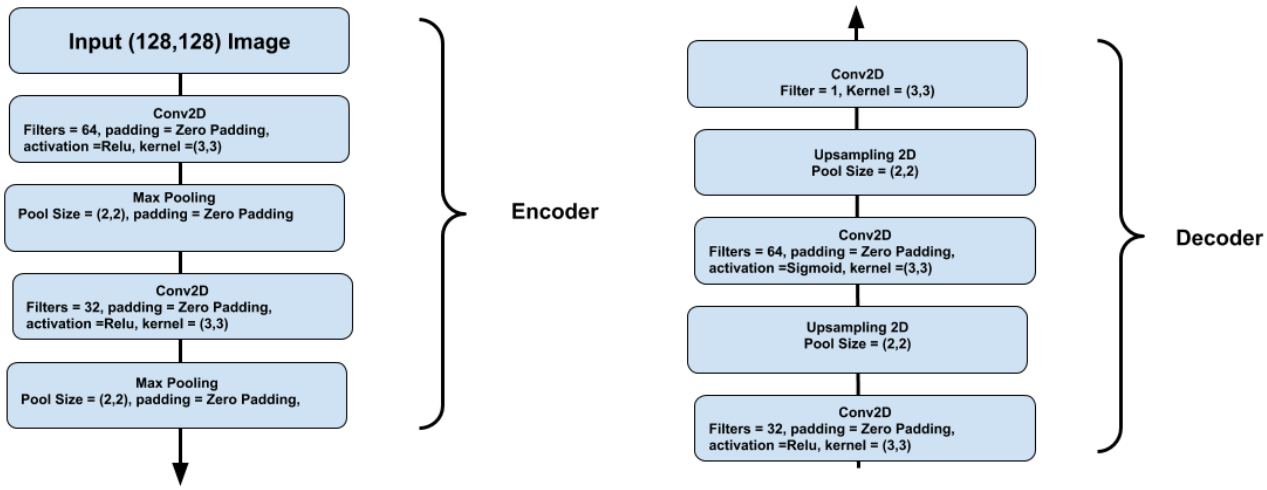

Figure 1: *Encoder-Decoder Block Diagram*

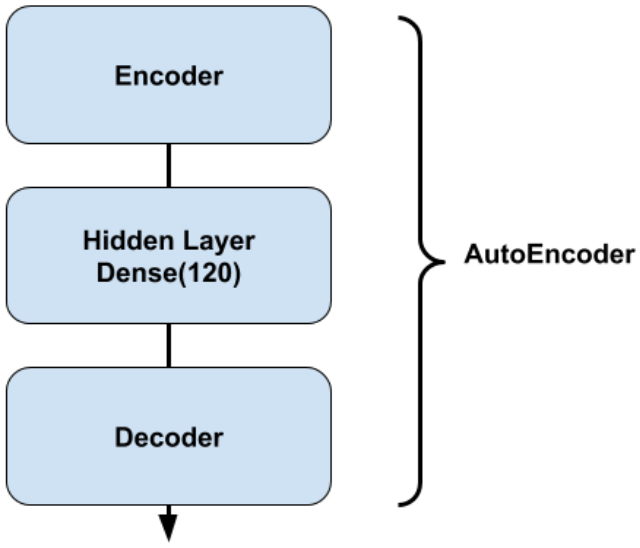

Figure 2: *AutoEncoder*

```
Model: "model_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        [(None, 128, 128, 1)]     0
_________________________________________________________________
conv2d_81 (Conv2D)           (None, 128, 128, 64)      640
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_82 (Conv2D)           (None, 64, 64, 32)        18464
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 32, 32, 32)        0
=================================================================
Total params: 19,104
Trainable params: 19,104
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "model_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        [(None, 128, 128, 1)]     0
_________________________________________________________________
conv2d_81 (Conv2D)           (None, 128, 128, 64)      640
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_82 (Conv2D)           (None, 64, 64, 32)        18464
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 32, 32, 32)        0
_________________________________________________________________
dense_10 (Dense)             (None, 32, 32, 120)       3960
_________________________________________________________________
conv2d_83 (Conv2D)           (None, 32, 32, 32)        34592
_________________________________________________________________
up_sampling2d_30 (UpSampling (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_84 (Conv2D)           (None, 64, 64, 32)        9248
_________________________________________________________________
up_sampling2d_31 (UpSampling (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_85 (Conv2D)           (None, 128, 128, 1)       289
=================================================================
Total params: 67,193
Trainable params: 67,193
Non-trainable params: 0
```

Figure 3: *AutoEncoder*

4 Results

Results of reconstruction using Principal Component Analysis for training set along with the Variance vs. Principal Components graph. More the number of principal components, more eigen values are selected from the covariance matrix. 5 Results of

Figure 4: *PCA comparison - Training Set*
Left -Original, Right- Reconstructed

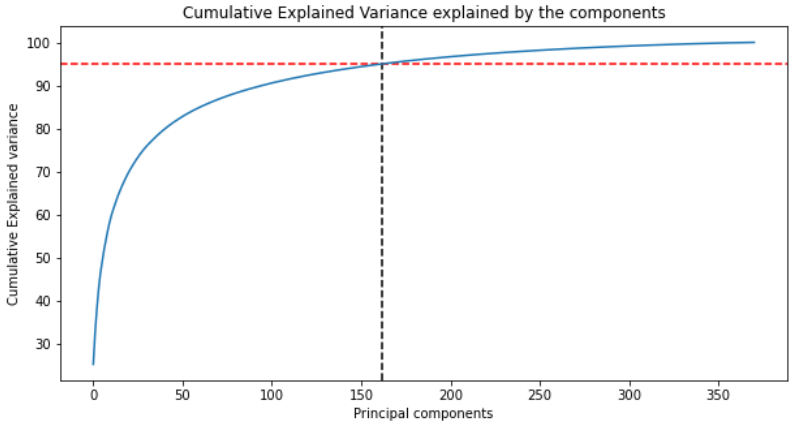

Figure 5: *Variance Vs. Principal Components — Training Set*

reconstruction using Principal Component Analysis for test set along with the Variance vs. Principal Components graph.7 Results of reconstruction using Principal Component Analysis for Validation set along with the Variance vs. Principal Components

Figure 6: *PCA comparison - Test Set*
Left -Original, Right- Reconstructed

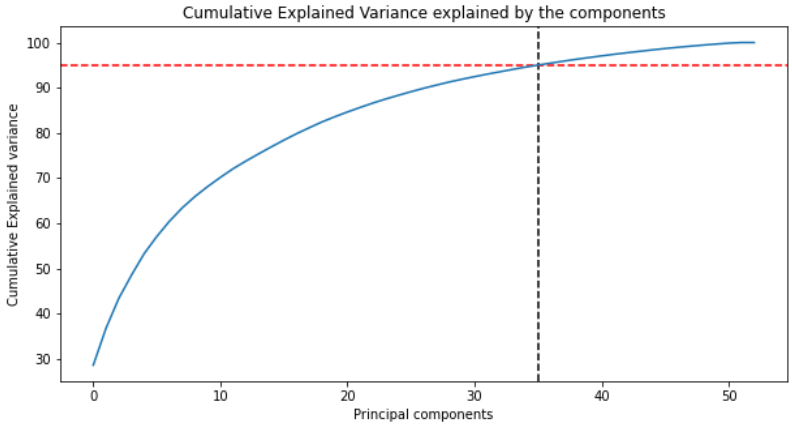

Figure 7: *Variance Vs. Principal Components — Test Set*

graph.9 Comparison of all reconstructed images for different value of k with original image set.10 Comparison of MSE for all k values i.e. k=37,20 and 50.11 Results of reconstruction using Autoencoders for test set along with MSE w.r.t original image. 13

Figure 8: *PCA comparison - Validation Set*
Left -Original, Right- Reconstructed

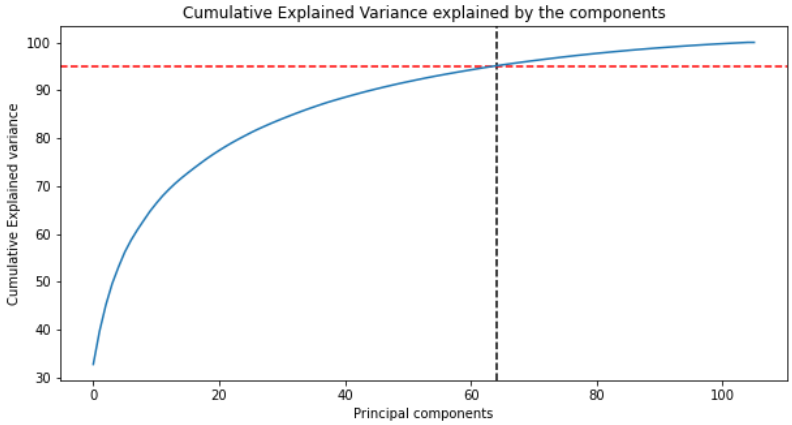

Figure 9: *Variance Vs. Principal Components — Validation Set*

Figure 10: *PCA comparison — Test Set — For different K values Image 1 = Original, Image 2 = Normal k value(k = 36), Image 3 = Value less than normal k value(k=20) and Image 4 = Greater than Normal k value(k=45)*