**INT 404 – ARTIFICIAL INTELLIGENCE**

**PROJECT REPORT**

**TOPIC : IMAGE MORPHING**

**SUBMITTED BY:**

UNNIKRISHNAN M

ROLL:42

11802231

**CONTENTS**

1. Abstract …………… 3

2. Introduction …………… 4

3. Related work ……………5

4. Prerequisite ...………… 8

4.1. Theorem of probability ………………8

4.2. Expectation ……………...8

4.3. KL Divergence ……………….9

5. Theoretical Results …………… 10

5.1. Latent Variable ……………… 12

5.2. Problem of Approximate Inference ……………… 13

5.3. Optimization of the Loss Function ……………… 18

5.4. Reparametrization trick ……………….19

6. Implementation in Tensorflow …………… 21

7. Results ……………29

8. Libraries used …………… 33

9. References ……………33

**1. ABTRACT**

It is a model derived from the auto encoder category with a little modification in the main framework which paves ways to the probabilistic exploitation of the input distribution. In auto encoder, the input datasets are mapped to single value attributes while encoding whereas in variational auto encoder the input samples are mapped to probabilistic distribution. Thus for each attribute of the input we generates a possibilities of values revolving around it. While decoding we sample a value which may lead to a new sample. The objective function make sure that the image generated resembles almost the existing samples and the probabilistic distribution of the attributes are gathered together with no gap between them for the effective exploitation of the features. Thus the model guarantee the user to output the image in a certain directed way with the desired attributed.

**2. INTRODUCTION**

Auto encoder model are capable enough to transform a higher level representation of an object to a lower level representation from which the original image can be reconstructed and have done a great job in congestion control technique. But it cannot make variations in the output to generate new samples. This is the area where the VAE scores.

The VAE model consist of two networks: the recognition model which maps the input sample to a probabilistic distribution of its attributes, thus finding the underlying structure of the input data, and the generative model whose duty is to reconstruct a new sample which resembles the original distribution with the variation of attributes according to user’s preference rather than rendering random sample with no control over the output as generative adversarial network. The VAE is designed in such a way that each attribute of the input sample is mapped to a probabilistic distribution by the encoder from which the generative model randomly sample the data to reconstruct a new image. For the effective conduct and exploitation of the distribution to render the desired result. The objective function consist of two types of losses: data fidelity term and KL divergence loss. The data fidelity term makes sure that the generated samples are not deviated form the prior distribution of samples whereas the KL divergence restrict the distribution from collapsing to zero variance and thus making sure that each mapping sample render a probabilistic distribution of attributes instead of a single value. The objective function is parameterized by the weights and bias of the two models. Optimization of the objective function is carried out in order to learn the parameters and thus achieving the desired objective after training.

**3 . RELATED WORK**

**Create Anime characters**

Game development and animation production are expensive and hire many production artists for relatively routine tasks. GAN can auto-generate and colorize Anime characters.

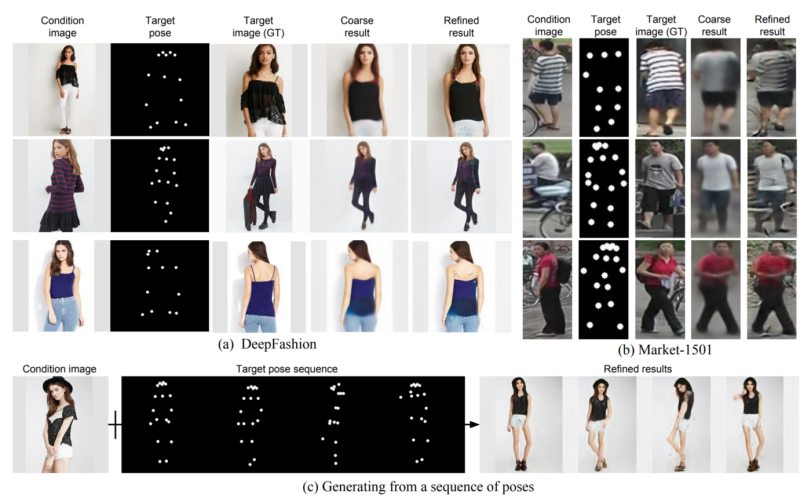
The generator and the discriminator composes of many layers of convolutional layers, batch normalization and ReLU with skip connections.



**Pose Guided Person Image Generation**

With an additional input of the pose, we can transform an image into different poses. For example, the top right image is the ground truth while the bottom right is the generated image.

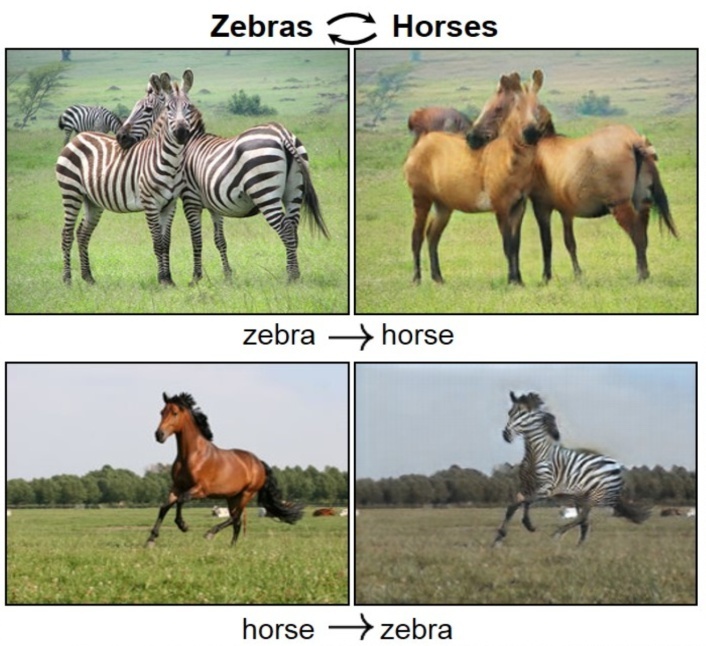
The design composes of a 2-stage image generator and a discriminator. The generator reconstruct an image using the meta-data (pose) and the original image. The discriminator uses the original image as part of the label input to a CGAN design.



**CycleGAN**

Cross-domain transfer GANs will be likely the first batch of commercial applications. These GANs transform images from one domain to another domain .

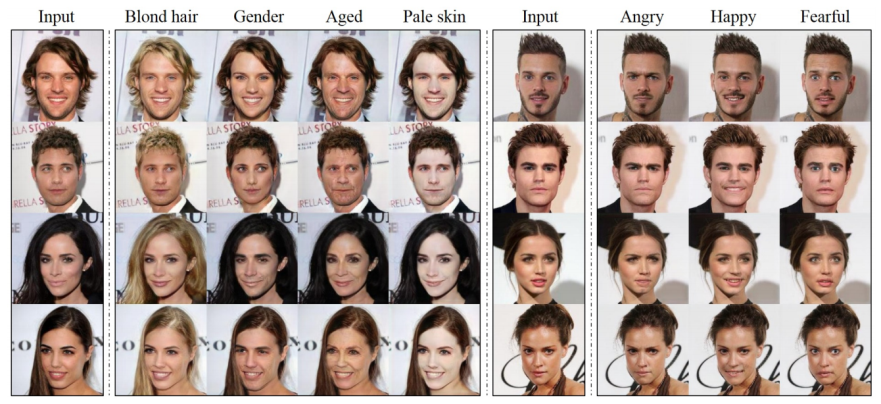
CycleGAN builds 2 networks ***G*** and ***F*** to construct images from one domain to another and in the reverse direction. It uses discriminators *D* to critic how well the generated images are. For example, ***G*** converts real images to Van Gogh style painting and *Dy* is used to distinguish whether the image is real or generated



**StarGAN**

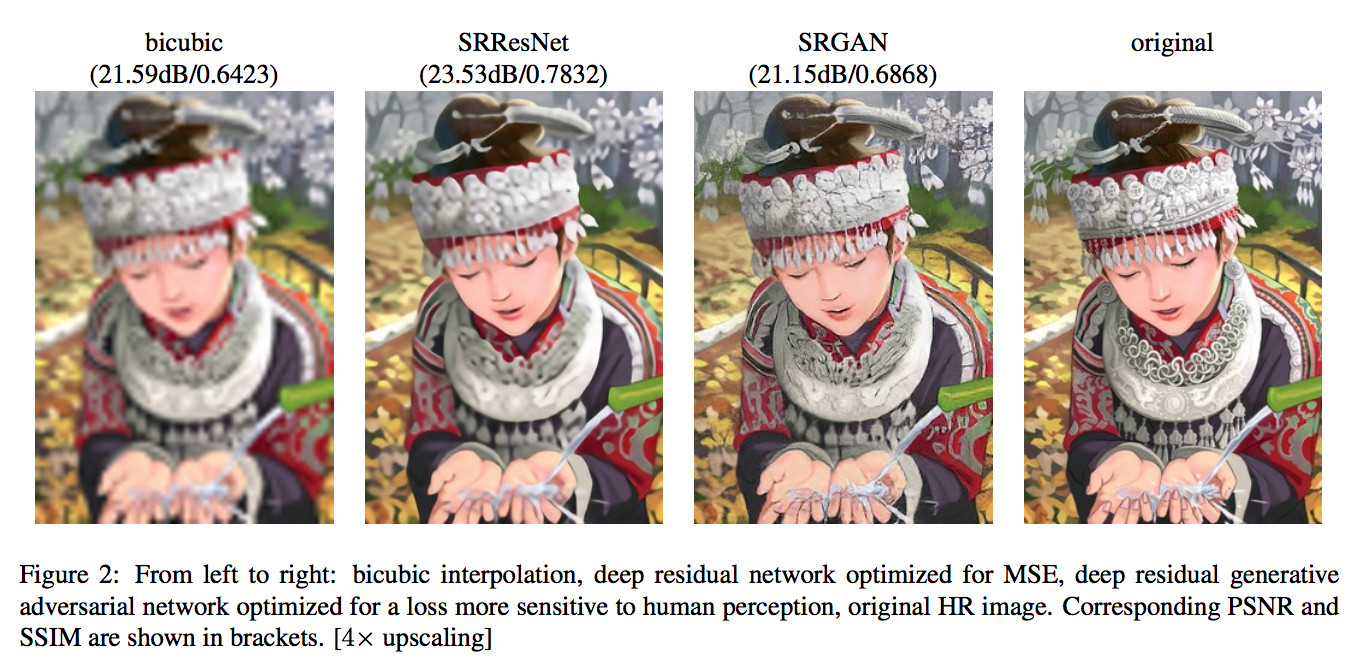
StarGAN is an image-to-image translation for one domain to another. For example, given a happy face, we want to transform it into a fearful face.

In (b), the generator generates a fake image based on an input image and a target domain label (say angry). In (c), given this fake image and the original domain of the image (say happy), it reconstructs the image using the generator. In (d), we feed real and fake images to the discriminator to label it as real or not as well as its domain classification. The cost function will involve reconstruction errors as well as the discriminator cost in identifying the images and their labels.



**Super resolution**

Create super-resolution images from the lower resolution. This is one area where GAN shows very impressive result with immediate commercial possibility



**High-resolution image synthesis**

This is not image segmentation! It is the reverse, generating images from a semantic map. Collecting samples are very expensive. We have trying to supplement training dataset with generated data to lower development cost. It will be handy to generate videos in training autonomous cars rather than see them cruising in your neighborhood.

**4.**  **PRE-REQUISITE**

P(x) -> defines the probability of a random variable x

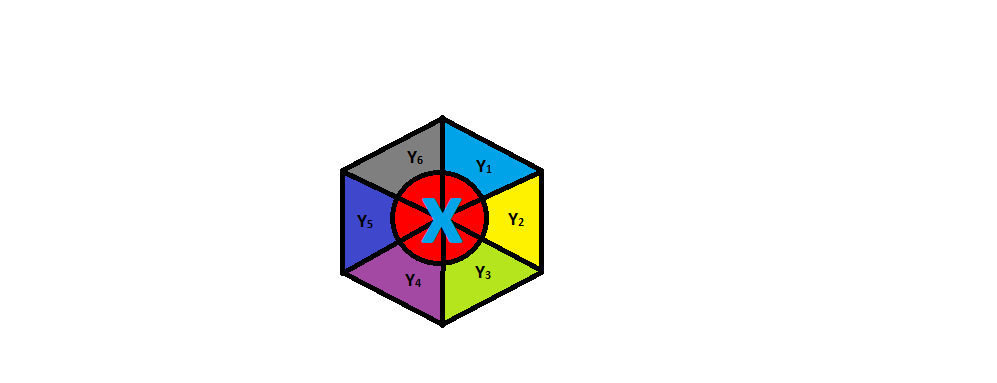
P(x|y) -> defines the probability of a random variable x provided y has already happened

P(x|y) = P(y|x) P(x) / P(x) ………Bayes theorem

**4.1** ***THEOREM OF TOTAL PROBABILITY***

Let y1 ....yn  be a set of mutually exclusive events and events x is the union of N mutually exclusive events, then

P(x) = ∑ P(x|yi)P(y)



**4.2EXPECTATION OF A VARIABLE X ……..E(x)**

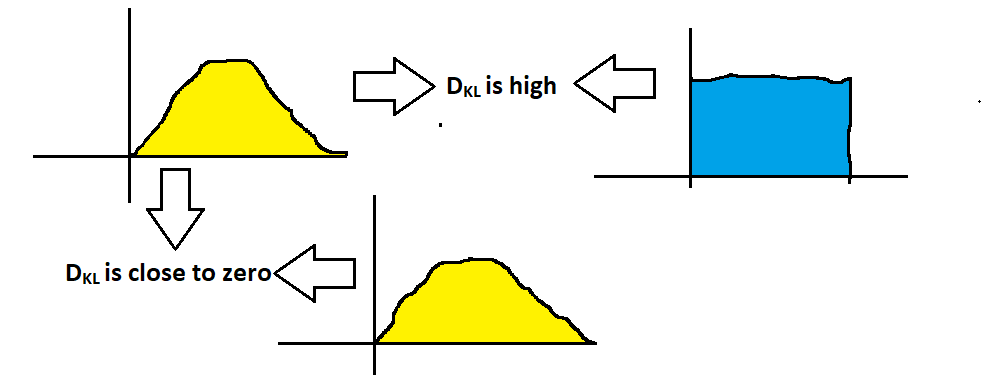
It is a weighted average of the possible values that x can take, each value being weighted according to the probability of that event defined as

E(x) = ∑ Xi P(Xi)

**4.3 KL DIVERGENCE**

Kullback –Leibler divergence is a measure of how one probability distribution P & Q, the KL divergence between them is defined as

DKL(P||Q) = ∑ P(x) log (P(x) /Q(x))

*PROPERTIES:*

* KL(P||Q) or KL(P||Q) >= 0
* KL(P||Q) != KL(Q||P) (not symmetric)

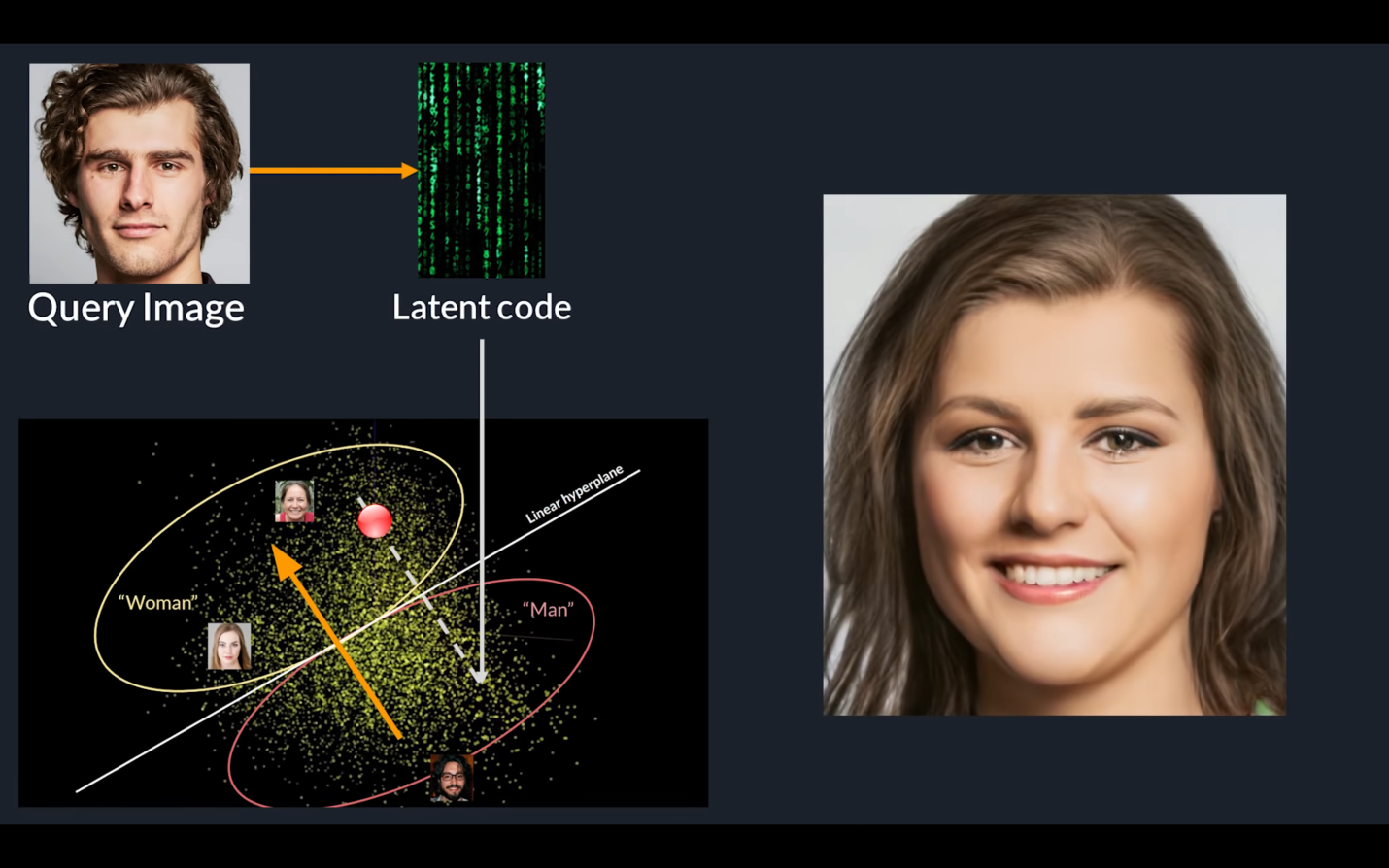
**5. THEORETICAL RESULT**

The goal of VAE is to find a distribution Q(ᵶ|x) of some latent variables ᵶ from which we can sample z ̴ Q(ᵶ|x) to generate new samples x’ ̴ P(x|z)

These are completely new samples but follows the existing input data distribution thus it manages to produce now samples

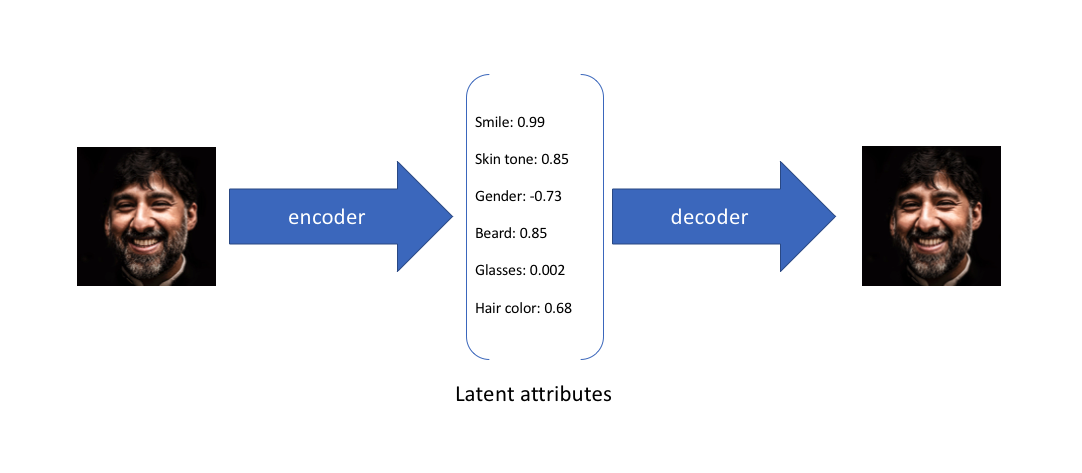
For better understanding why VAE is important we should first know what are problems that the VAE try to address which were existed in the earlier models for example auto encoders



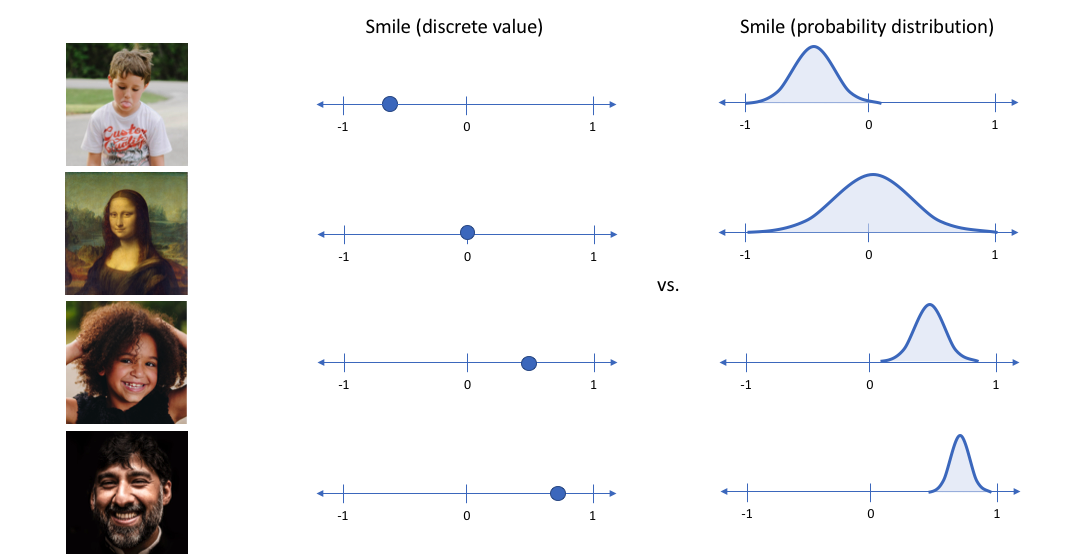
Auto encoders are the models which are used to get low level representation of data of a real data. Higher level representation of the input data is fed into the encoder model Q(ᵶ|x) which convert it into lower level representation Z (latent variable). It is very useful in term sof congestion control by the effective utilization of bandwidth in data transmission. The original image can be recovered by feeding the compressed lower representation Z into a decoder. Auto encoder are capable enough in reconstructing the original data that is already present in the distribution but has failed to render new samples which follows the fame distribution. Here comes the role of VAE. It takes a simple but effective extension of auto encoder, instead of generating a single encoding of the input samples. It generates a probability distribution over each of the encoding using the encoder. The decoder then sample from this probability distribution and generate new samples. ******

**5.1 LATENT VARIABLE**

It corresponds to a real feature of the object that have not been measured

In the above example, we trained auto encoder on a large dataset faces with encoding dimension of 6. An ideal autoencoder will learn the descriptive attributes of fakes like skin, smile etc… in order to describe observation in some compressed form. We have described the input image in terms of latent variable using single value to describe each attribute. Using VAE, we define the latent attributes in probabilistic manner.

With this approach we now represent each latent attribute for a given input as probabilistic distribution. While decoding, we randomly sample from each latent state distribution to render a vector as input for the decoder model.



The derivation of the Loss function can be solved using :

**5.2The Problem Of Approximate Inference**

Let x be a set of observed variable and let Z be the set of latent variable with joint distribution P(ᵶ|x).

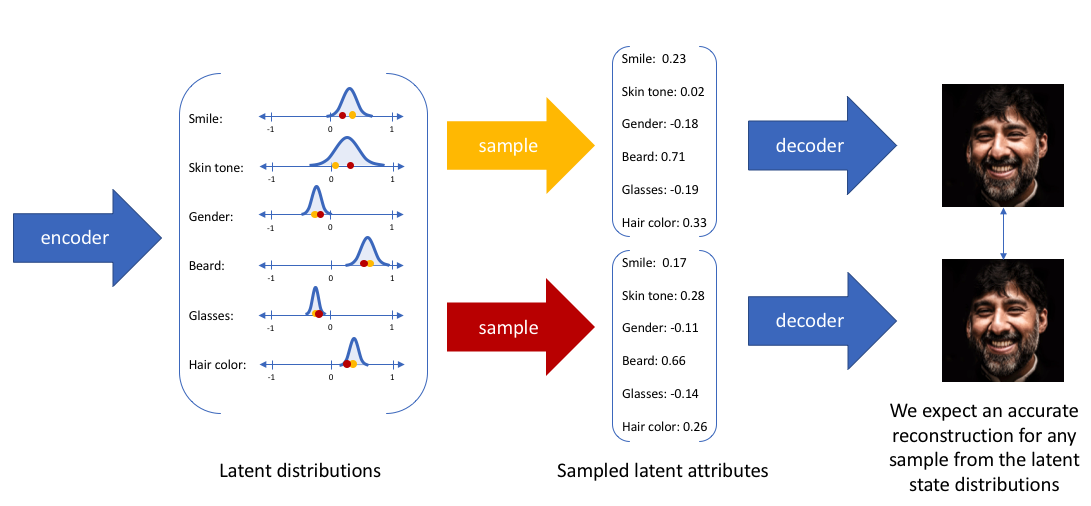
Then the inference problem will complete the conditional distribution of latent variables given the observation

P(ᵶ|x) = P(x|ᵶ) P(ᵶ) / P(x)

Evaluating the above equation is difficult because P(x) cannot be solved

**Reason :** P(x) = ʃ P(x|ᵶ) P(ᵶ) dx , this integral is not available in closed form or is intractable

(i.e., it requires exponentisl time to complete) due to multiple integral involved for latent variable vector Z.



***ALTERNATIVE:***

The alternate approach is to approximate P(ᵶ|x) by another distribution Q(ᵶ|x) what is defined in such a way that it has a tractable solution. This is done using variational inference. The main idea of V.I is to pose the inference problem as an optimization problem. It is done by modeling P(ᵶ|x) using Q(ᵶ|x) where Q(ᵶ|x) has a simple distribution such as Gaussian. This is done by minimizing the KL(Q(ᵶ|x) || P(ᵶ|x)). After rearranging and deriving, we obtain the objective function of VAE as

E z ~ Q(ᵶ|x [ log (P(x|ᵶ)) - DkL (Q(ᵶ|x)||P (ᵶ)) ]

first term represents the reconstruction likelihood and the second term ensures that our learned distribution Q is similar to the prior distribution P.

**LOSS FUNCTION = - objection function**

= - E z ~ Q(ᵶ|x [ log (P(x|ᵶ)) + DkL (Q(ᵶ|x)||P (ᵶ)) ]

So our target is to find the optimal Ø,Ѳ such that

Ѳ\* , Ø\* = argmin Ѳ,Ø  L(Ѳ,Ø)

Intution about Loss Function

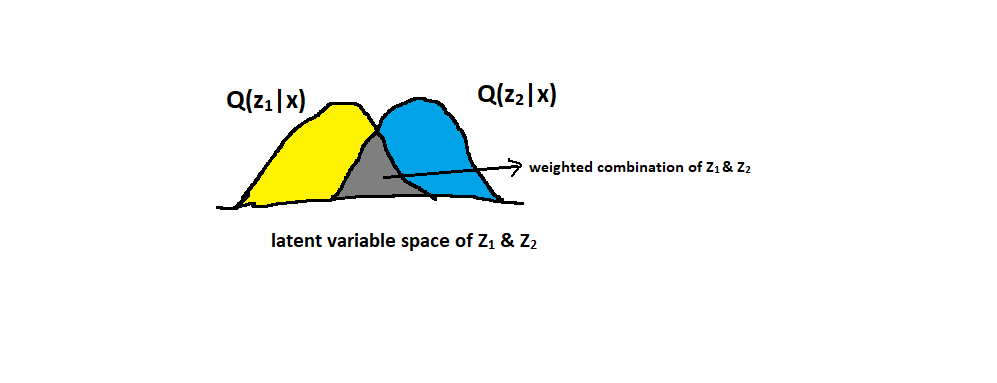
L(Ѳ,Ø) = - E z ~ Q(ᵶ|x [ log (P(x|ᵶ)) + DkL (Q(ᵶ|x)||P (ᵶ)) ]

Ѳ,Ø are new work parameter which are to be learned that’s why loss function is parameterized by them so that after minimizing loss function by adjusting Ø and Ѳ we attain the model that is capable enough to generate the desired output. QØ(ᵶ|x) is neural network which is also called as recognition model which x -> ᵶ. The first term represent the log likelihood.

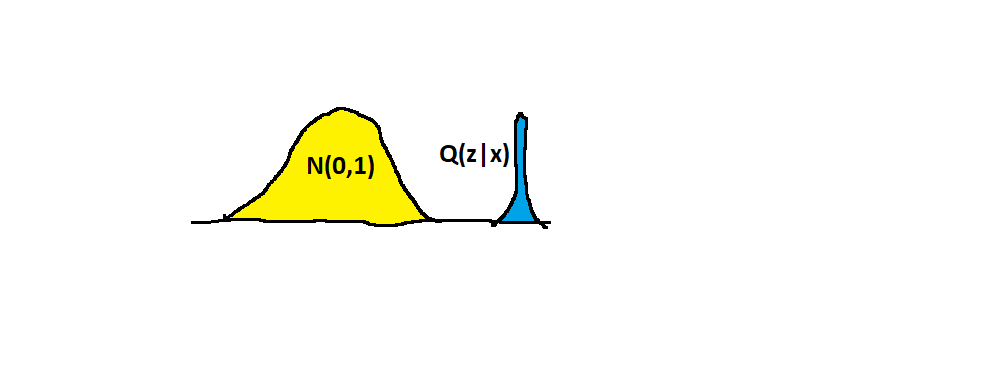
P(x|ᵶ) = N (ψ(ᵶ) , £(ᵶ))

So then we take log of the Gaussian, we get a square error between the data sample x and the mean of the Gaussian distribution P(x|ᵶ) is also called as the generative model us it is taking the input as z to the latent vector and mapping to the X’ the reconstructed sample.

Second term is the regularizer : PѲ(ᵶ) ~ N(0,1)

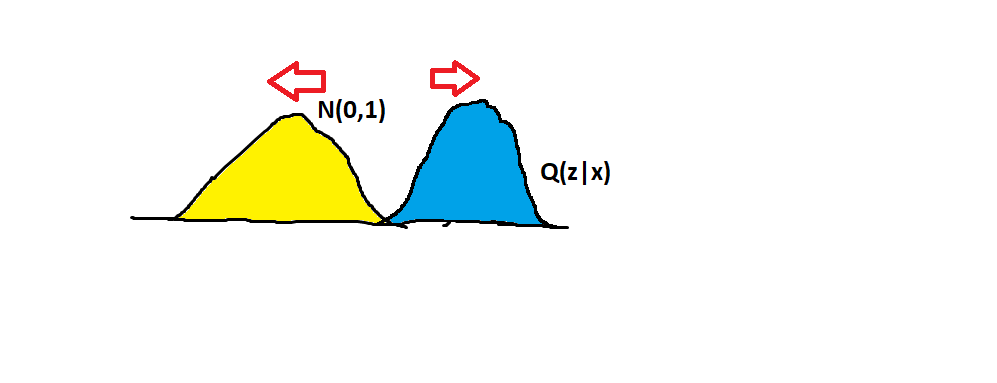
KL divergence not allows the pdf of the latent variable to collapse with zero variance but penalize the deviation from PѲ(ᵶ). Once you feed it to the decoder you get a new sample thich has the mixture of z1 & zz  which is not present in the real distribution but follows the same. 

* Data fidelity 🗹 K.L divergence 🗷

Without regularization, network will cheat by learning narrow distribution. With very narrow variance the distribution effectively representing single value like in auto encoder. 

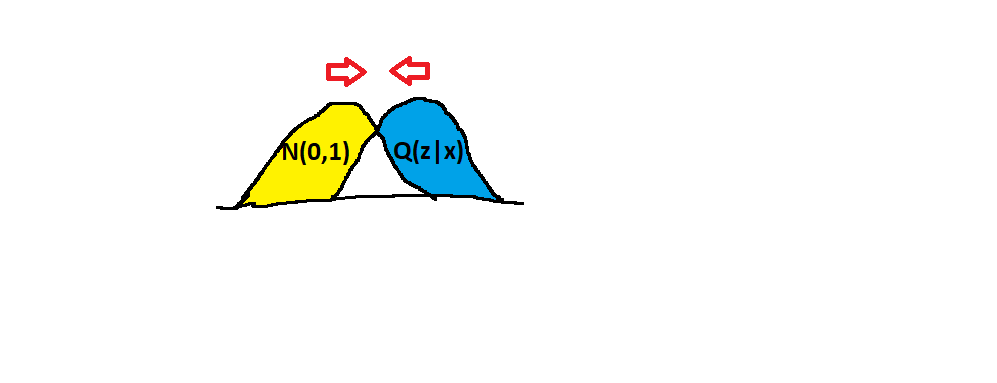
* Data fidelity 🗷 K.L divergence 🗹

Penalizing the reconstruction loss encourages our distribution to deviate from the prior to describe some characteristics of data.



* Data fidelity 🗹 K.L divergence 🗹

1st term results in the attraction between the two distribution. 2nd term ensures the sufficient variance in the distribution so that the model can produce samples of varying attributes.



**5.3 OPTIMIZATION OF THE LOSS FUNCTION**

As discussed, Ѳ\*, Ø\* = argmin Ѳ,Ø  L(Ѳ,Ø)

In variational Baysian method, this loss function is known as the variational lower bound or evidence lower bound (ELBO). This ‘lower bound ‘ part comes form the fact that KL divergence is always non-negative & thus L(Ѳ,Ø) is the lower bound of log PѲ(x).

L(Ѳ,Ø) = - E z ~ Q(ᵶ|x [ log (P(x|ᵶ)) + DkL (Q(ᵶ|x)||P (ᵶ)) ]

And we know DkL (Q(ᵶ|x)||P (ᵶ)) >=0

as a result **, L(Ѳ,Ø) <= log PѲ(x).**

therefore minimizing loss, we are maximixing the lower bound of the probability of generating the real data samples.

**5.4 REPARAMETRIZATION TRICK**

Recall :

L(Ѳ,Ø) = - E z ~ Q(ᵶ|x [ log (P(x|ᵶ)) + DkL (Q(ᵶ|x)||P (ᵶ)) ]

The optimization is carried out with respect to both Ѳ & Ø to learn QØ(ᵶ|x) PѲ(x|ᵶ) at the same time.

minѲ,Ø  L(Ѳ,Ø) = minѲ,Ø  { -E z ~ Q(ᵶ|x [ log (P(x|ᵶ)) + DkL (Q(ᵶ|x)||P (ᵶ)) ] }

we run the algorithm for a fixed number of iterations that is upto N. In that the first step is to find the derivative of L(Ѳ,Ø) with respect to Ѳ. The result will give the optimal value of Ѳi. The next step is to find optimal value of Øi. For that we will find the derivative of L(Ѳ,Ø) with respect to Ø. Here we replace the Ѳ with the optimal value find in last iteration that is Ѳi.

Ѳi = d/dѲ(L(Ѳ,Ø) )

Øi = d/dØ(L(Ѳ,Ø) )

The derivative d/dØ is harder to estimate because Ø appears in the distribution with respect to which expectation is taken.

If we can somehow rewrite the expectation in such a way that the Ø appears inside the expectation then we can push the gradient inside expectation i.e.,

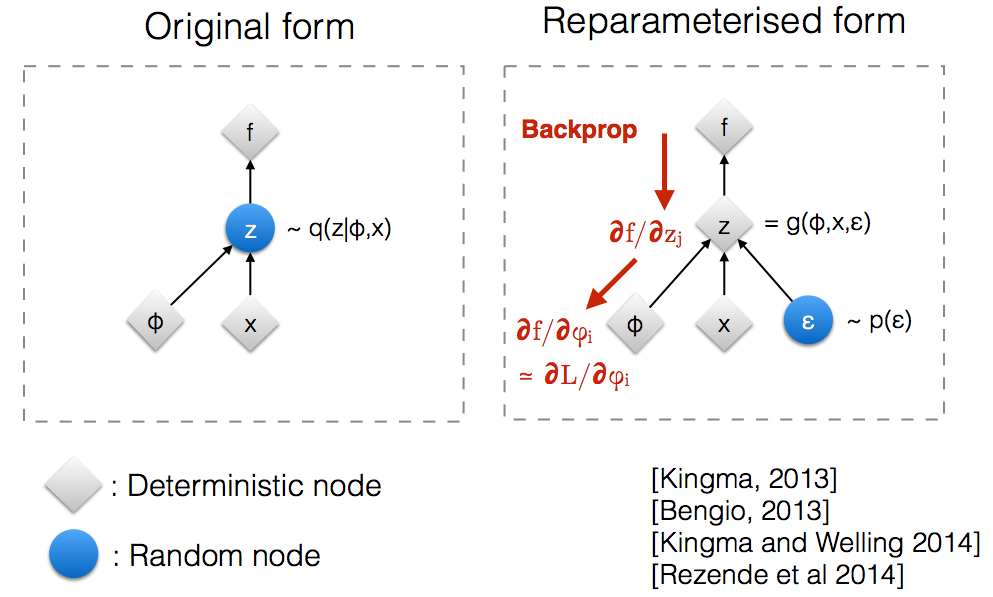
EQ(ᵶ|x)[f(ᵶ)] = EP(£)[f(gØ(£,x))]

Such that Z = gØ(£,x) with £ ~ N(0,1)

In our case, gØ(£,x) = ψØ(x) + £. ∑Ø(x) = Z ~ N(ψØ(x), ∑Ø(x))

Here N(ψØ(x), ∑Ø(x)) is obtained from N(0,1) using the above linear transformation.

Instead of sampling ᵶ ~ QØ(ᵶ|x), we are sampling from N(0,1) i.e, £ ~ N(0,1) and then linear transformation using the above function to realize N(ψØ(x), ∑Ø(x)).



**6.** **IMPLEMENTATION IN TENSORFLOW**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

learning\_para =0.001

epochs=30000

batch\_size =32

from tensorflow.examples.tutorials.mnist import input\_data

database= input\_data.read\_data\_sets('/content/data',one\_hot = True)

**"""network parameters"""**

image\_dimension=784

neural\_network\_dimension=512

latent\_variable\_dimension=2

**"""initialization"""**

def xavier(in\_shape):

val=tf.random\_normal(shape=in\_shape , stddev=1/tf.sqrt(in\_shape[0]/2))

return(val)

**"""weight and bias dictionaries"""**

weight= {"weight\_matrix\_encoder\_hidden":tf.Variable(xavier([image\_dimension,neural\_network\_dimension])),

"weight\_mean\_hidden":tf.Variable(xavier([neural\_network\_dimension,latent\_variable\_dimension])),

"weight\_std\_hidden":tf.Variable(xavier([neural\_network\_dimension,latent\_variable\_dimension])),

"weight\_matrix\_decoder\_hidden":tf.Variable(xavier([latent\_variable\_dimension,neural\_network\_dimension])),

"weight\_decoder":tf.Variable(xavier([neural\_network\_dimension,image\_dimension]))

}

bias = {"bias\_matrix\_encoder\_hidden":tf.Variable(xavier([neural\_network\_dimension])) ,

"bias\_mean\_hidden":tf.Variable(xavier([latent\_variable\_dimension])),

"bias\_std\_hidden":tf.Variable(xavier([latent\_variable\_dimension])),

"bias\_matrix\_decoder\_hidden":tf.Variable(xavier([neural\_network\_dimension])),

"bias\_decoder":tf.Variable(xavier([image\_dimension]))

}

"""encoder\_section"""

image\_x = tf.placeholder(tf.float32,shape=[None,image\_dimension])

encoder\_layer=tf.add(tf.matmul(image\_x,weight["weight\_matrix\_encoder\_hidden"]),bias["bias\_matrix\_encoder\_hidden"])

encoder\_layer=tf.tanh(encoder\_layer)

mean\_layer=tf.add(tf.matmul(encoder\_layer,weight["weight\_mean\_hidden"]),bias["bias\_mean\_hidden"])

std\_layer=tf.add(tf.matmul(encoder\_layer,weight["weight\_std\_hidden"]),bias["bias\_std\_hidden"])

"""reparamatrization\_trick"""

epsilon=tf.random\_normal(tf.shape(std\_layer),dtype=tf.float32,mean=0.0 , stddev=1.0)

latent\_layer=mean\_layer+tf.exp(0.5\*std\_layer)\*epsilon

"""DECODER SECTION"""

decoder\_hidden=tf.add(tf.matmul(latent\_layer,weight["weight\_matrix\_decoder\_hidden"]),bias["bias\_matrix\_decoder\_hidden"])

decoder\_hidden=tf.tanh(decoder\_hidden)

decoder\_output=tf.add(tf.matmul(decoder\_hidden,weight["weight\_decoder"]),bias["bias\_decoder"])

decoder\_output=tf.sigmoid(decoder\_output)

"""VARIATIONAL ENCODER LOSS"""

def loss\_function(original\_image,reconstructed\_image):

data\_fidelity\_loss=original\_image\*tf.log(1e-10 + reconstructed\_image)+(1-original\_image)\*tf.log(1e-10 + 1-reconstructed\_image)

data\_fidelity\_loss=-tf.reduce\_sum(data\_fidelity\_loss,1)

kl\_div\_loss=1+std\_layer-tf.square(mean\_layer)-tf.exp(std\_layer)

kl\_div\_loss=-0.5\*tf.reduce\_sum(kl\_div\_loss,1)

alpha=1

beta=1

network\_loss=tf.reduce\_mean(alpha\*data\_fidelity\_loss+beta\*kl\_div\_loss)

return(network\_loss)

loss\_value= loss\_function(image\_x,decoder\_output)

optimizer=tf.train.RMSPropOptimizer(learning\_para).minimize(loss\_value)

init=tf.global\_variables\_initializer()

sess=tf.Session()

sess.run(init)

for i in range(epochs):

x\_batch,\_ =database.train.next\_batch(batch\_size)

\_,loss= sess.run([optimizer,loss\_value],feed\_dict= {image\_x : x\_batch})

if i%5000 ==0:

print("loss is {0} at iteration{1}".format(loss,i))

"""# TESTING"""

noise\_x = tf.placeholder(tf.float32,shape=[None,latent\_variable\_dimension])

decoder\_hidden=tf.add(tf.matmul(noise\_x,weight["weight\_matrix\_decoder\_hidden"]),bias["bias\_matrix\_decoder\_hidden"])

decoder\_hidden=tf.tanh(decoder\_hidden)

decoder\_output=tf.add(tf.matmul(decoder\_hidden,weight["weight\_decoder"]),bias["bias\_decoder"])

decoder\_output=tf.sigmoid(decoder\_output)

"""OUTPUT VISUALIZATION"""

n=20

x\_limit =np.linspace(-2,2,n)

y\_limit=np.linspace(-2,2,n)

empty\_image=np.empty((28\*n,28\*n))

for i,zi in enumerate(x\_limit):

for j , pi in enumerate(y\_limit):

generated\_latent\_layer=np.array([[zi,pi]]\*batch\_size)

#generated\_latent\_layer=np.random.normal(0,1,size=[batch\_size,latent\_variable\_dimension] )

generated\_image=sess.run(decoder\_output,feed\_dict={noise\_x : generated\_latent\_layer})

empty\_image[(n-i-1)\*28:(n-i)\*28,j\*28:(j+1)\*28]= generated\_image[0].reshape(28,28)

plt.figure(figsize=(8,10))

x,y = np.meshgrid(x\_limit,y\_limit)

plt.imshow(empty\_image,origin="upper",cmap="gray")

plt.grid(False)

plt.show()

x\_sample,y\_sample =database.test.next\_batch(batch\_size+15000)

print(x\_sample.shape)

interim = sess.run(latent\_layer , feed\_dict={image\_x : x\_sample })

print(interim.shape)

colors=np.argmax(y\_sample,1)

plt.figure(figsize=(8,6))

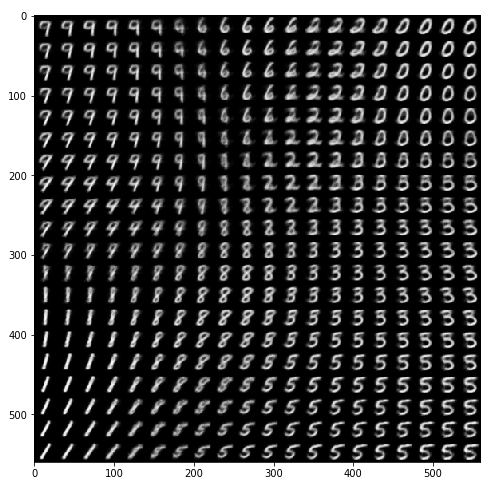
plt.scatter(interim[:,0],interim[:,1],c=colors,cmap='viridis')

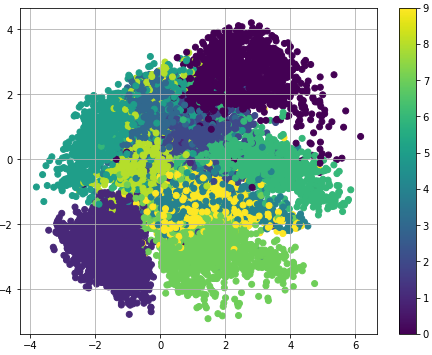
plt.colorbar()

plt.grid()

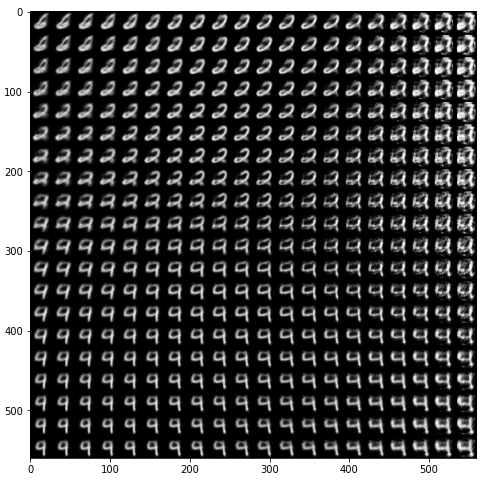
sess.close();

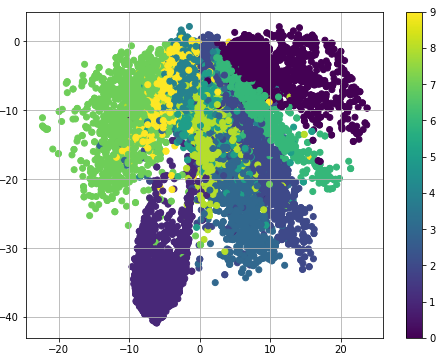
**7.OUTPUT** Data fidelity 🗹 K.L divergence 🗹

****

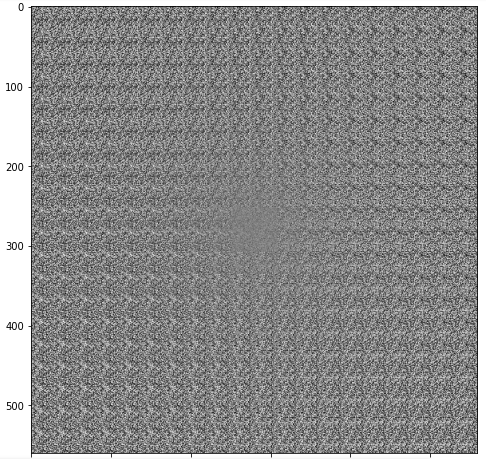


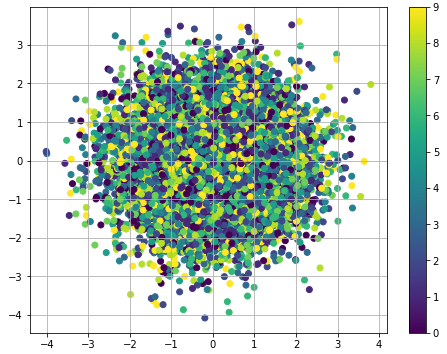
**OUTPUT** Data fidelity 🗹 K.L divergence 🗷

****

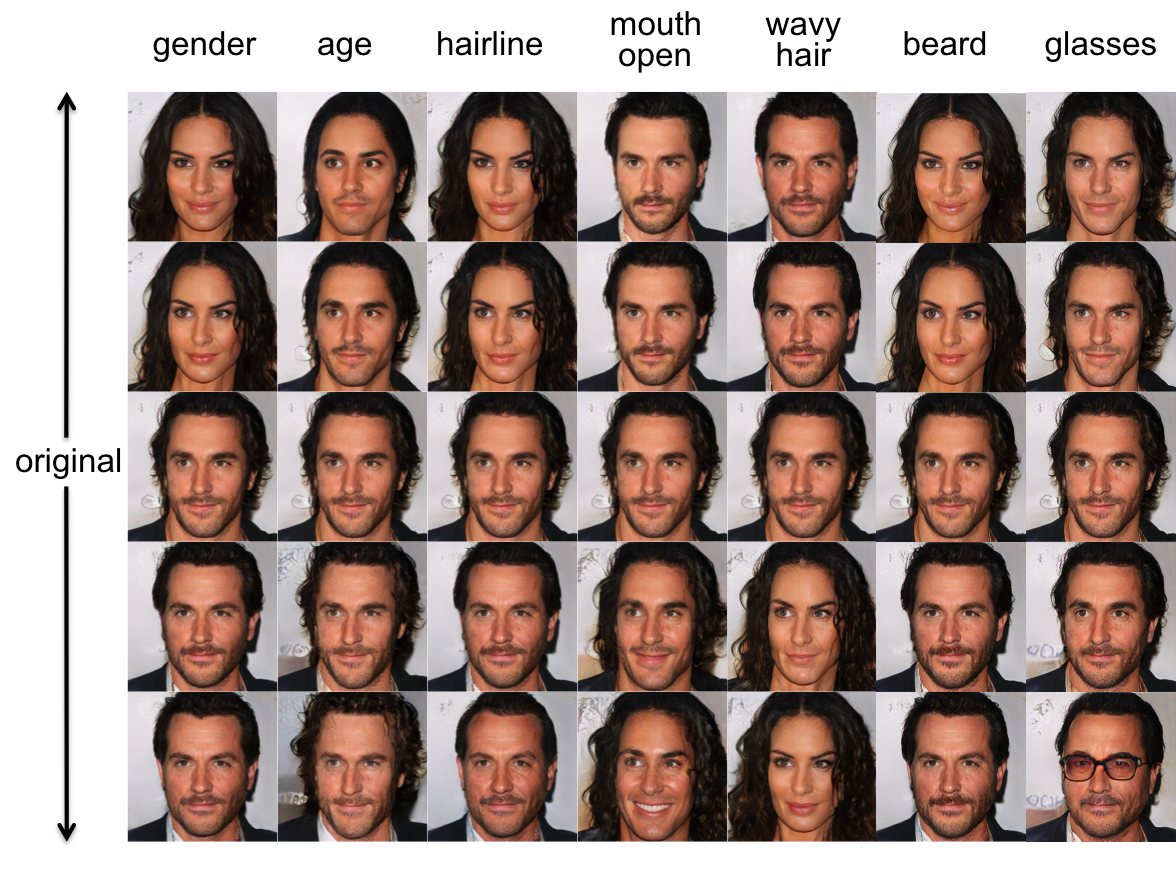
****

**OUTPUT** Data fidelity 🗷 K.L divergence 🗹





**OUTPUT : When FREY FACE dataset is used**

****

**8. LIBRARIES USED**

* Tensorflow
* Keras
* Matplotlib
* Numpy
* Pandas

**9. REFERENCES**

[1] ***Deep learning Cookbook*** by douwe osinga

[2] ***Variational Autoencoder research paper*** by Ian J. Goodfellow

[3] ***Medium blog post***  *(*<https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf>*)*

[4] ***Deep Learning*** by Ian Goodfellow

[5] ***Deep learning with python*** by Francois Chollet

[6] ***Neural Networks and Deep Learning*** by Michael Nielson