# 3D Shape Completion for Autonomous Vehicles

**Mentors**

Abhay Dayal Mathur  Shiven Tripathi

**Team Members**

Akshay K Muraleedharan  Ashish Garg  Gaurav Kumar  Lakshita Mohanty  Prateesh Awasthi  Himanshu PS

# Introduction

The aim of this project is to use ML to successfully reconstruct a 3D model of an object given a small glimpse of the model in the form of LiDAR data and RGB images. In the first few weeks of the project, we were introduced to basic machine learning and deep learning concepts, and implemented semantic segmentation using the UNet architecture. We later moved on to Variational Auto Encoders(VAEs) and finally shape completion.

## 1. (Learning) Basics of Neural Nets

Before diving into the main problem statement, we were required to learn and brush up on our basic Machine Learning concepts. We all studied the course Neural Networks and Deep Learning.

The topics covered in it were:

Linear and Logistic Regression

- Linear Regression : Modelling Relationship between input and output using a linear function.
- Logistic Regression: Linear regression with a nonlinear function as an added step.

- Cost Function and Gradient Descent
  - Cost Function : A function used to give a measure of how different your output is from the expected output.
  - Gradient Descent : The process of calculating the derivatives of the loss function with respect to the various variables in the previous logistic regression steps.
- Vectorization and Derivatives
  - Vectorization helps us to work with large batches of data all at once rather than sequentially using a for loop. It greatly increases the speed of training.
- Neural Networks
  - These are machine learning algorithms that use nodes similar to human neurons to solve various tasks. In this course, we learned a type called “Fully connected neural network”. A neuron in this network usually consists of a logistic regression unit.

- Activation Functions
    - They are nonlinear functions that are necessary to get useful results in a neural network. They help model problems where the relationship between the inputs and outputs is more complex than a simple linear form.
    - In this course, we learnt about 4 Activation Functions:
        - Sigmoid : $\sigma(x) = \frac{1}{1+exp(-x)}$
        - Tanh : Hyperbolic Tangent
        - ReLU : Rectified Linear Unit; (0 if x<0 else x)
        - Leaky ReLU : same as ReLU but has a small positive slope when x<0
- Backpropagation
    - The process of adjusting the various parameters in the previous layers of the neural network using derivatives. It is the actual 'learning' part in the process.
- Hyperparameters
    - Parameters of the whole neural network that are decided by the programmer and not necessarily learnt, for example number of layers, choice of activation function, learning rate, etc.

### Overview of the Assignments:

Python Basics with Numpy:
An introduction to numpy, the Python library for high-level computing. Implemented basic functions such as sigmoid, the loss function, normalization, etc.
Planar Data Classification:
A simple 2 class classification neural network; loading data; creating a simple logistic regression model; understanding the functions from scratch.
Trained the model; got a 90% accuracy
DNN-Step by step:
Implementation of functions for a deep neural network, such as forward activation steps and backward propagation steps.
DNN-Application:
Made a 2-layer network and an L-layer network
Trained both of them on the same test; L-layer network performed better than the 2-layer network

# 2. (Learning) DL in Computer Vision

## 2.1 BASIC IMAGE PROCESSING

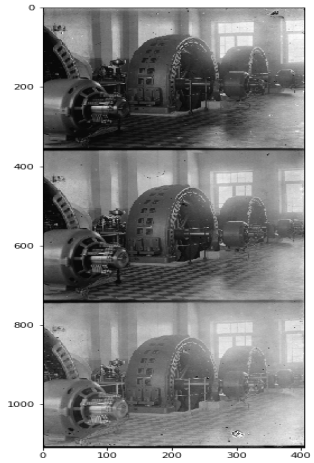
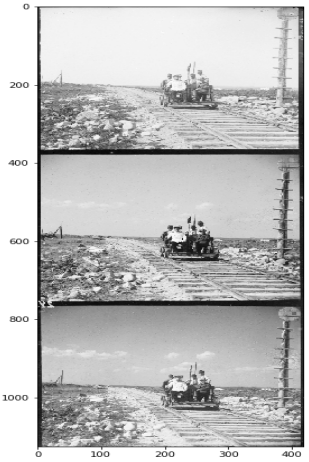
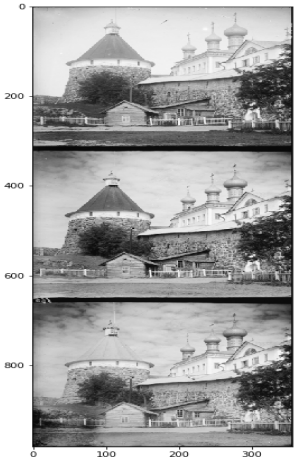
We learned how to perform basic image processing tasks to improve the quality of the images for their subsequent analysis.

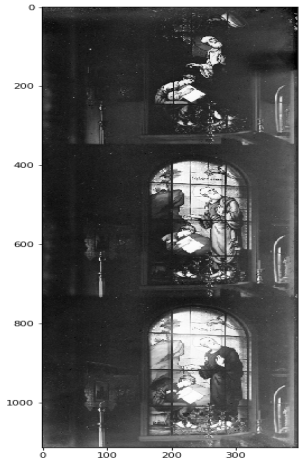
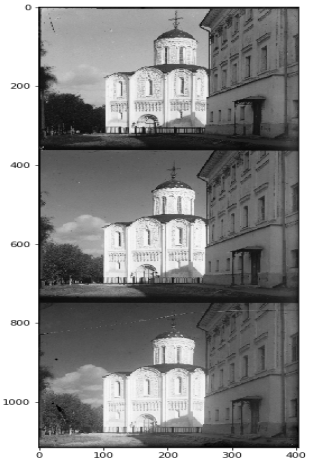
### **Assignment**: Image Alignment

Tasks:

1. Image channels processing and Alignment
2. Face Alignment

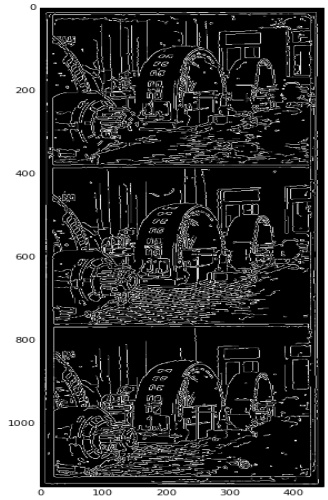
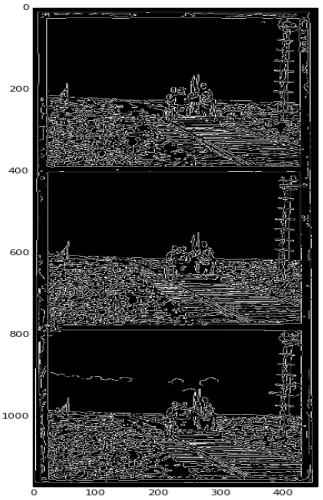
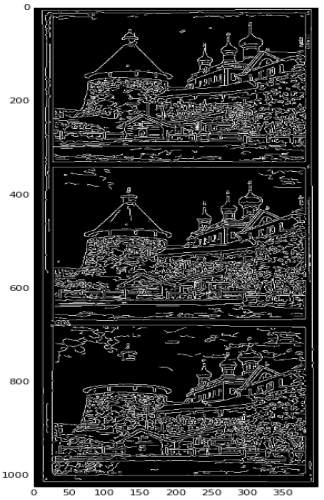
## 1.Image Channels Processing and Alignment:

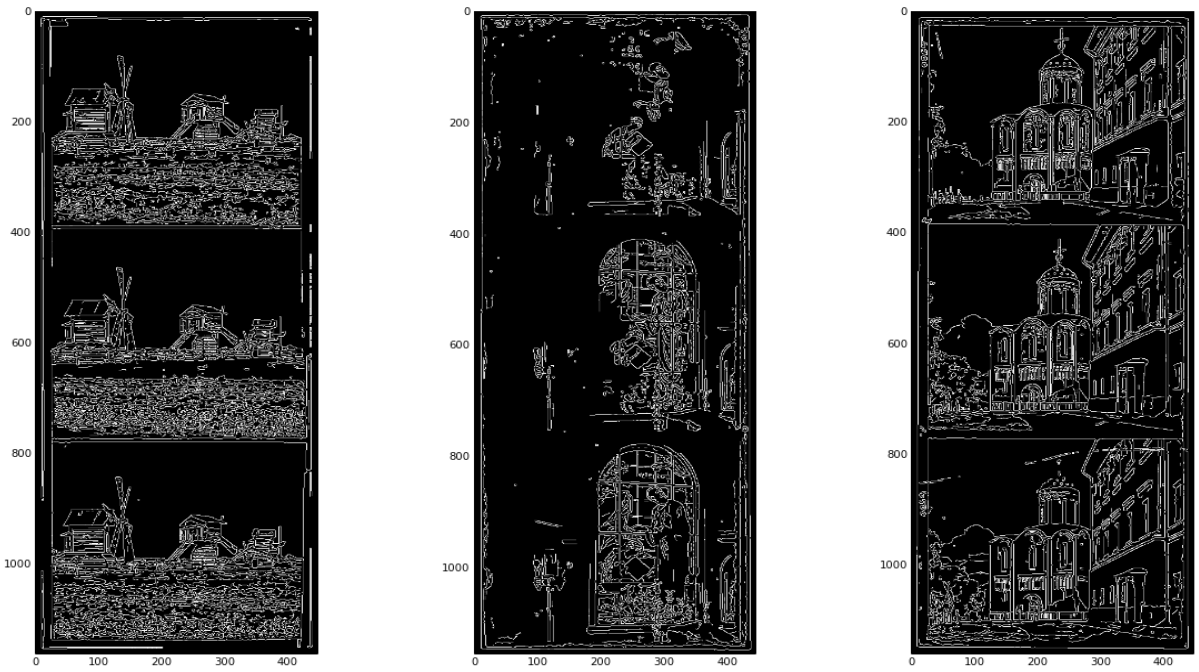




The input images are sets of 3 plates, corresponding to B, G, and R channels (top-down).

Task: To remove the border and align the B,G and R channels.
To do this, implement canny edge detector,



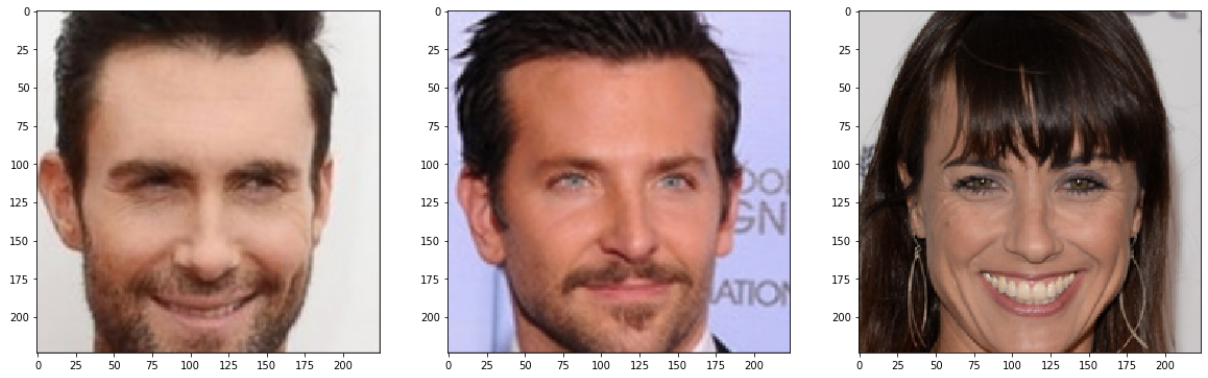
After that, we separate the channels and make an RGB image and search for the best shift for channel Alignment and align the image.

## 2.Face Alignment:

The dataset contains images and corresponding eyes coordinates of each person.

Most of the face images deceptively seem to be aligned, but since many face recognition algorithms are very sensitive to shifts and rotations, we need not only to find a face on the image but also normalize it. Besides, the neural networks usually used for recognition have fixed input size, so, the normalized face images should be resized as well.

- Using labels of position of eyes, find the angle between horizontal line and a line connecting both eyes then, rotate the image to make both eyes on the same level



and locate the face in the image, then crop it and resize the image to 224x224.

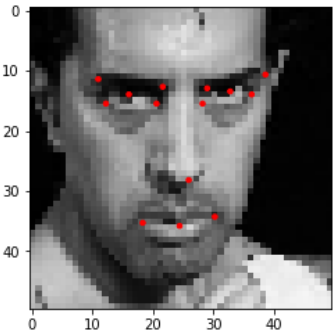
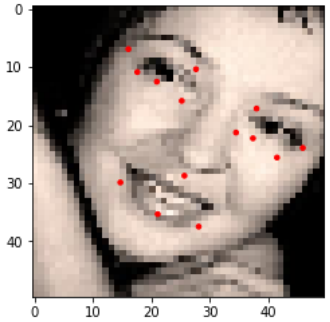
# 2.2 IMAGE CLASSIFICATION and KEYPOINTS REGRESSION

We learned to identify the facial keypoints on random images.

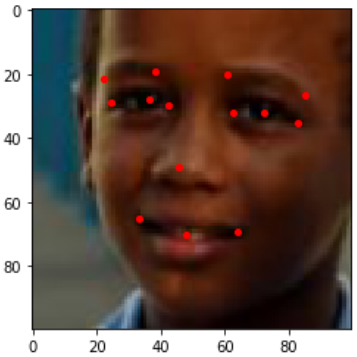
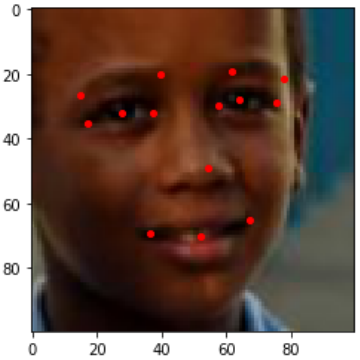
**Assignment:** Facial Keypoints Detection

## Task: creating the facial keypoints for new images.

Data contains images and corresponding facial keypoints.

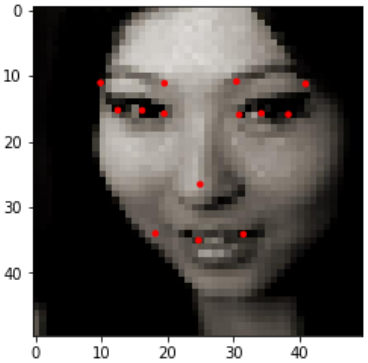
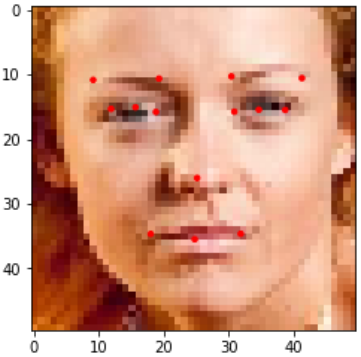



We use horizontal flipping as the mode of data augmentation here, in order to increase the training dataset.

Network Architecture:
3 repetitions of a 2D-Convolutional layer, a Leaky ReLu layer,a Max Pooling layer and then a Dropout layer. This is followed by a flatten layer to bring the multi-dimensional input to a single dimension, for the transition from the convolution layer to the fully connected layer. We then have 2 repetitions of a Dense layer followed by a Leaky ReLu layer, with a Dropout in between. We end it with a Dense layer..

Some results obtained on prediction for the test dataset:

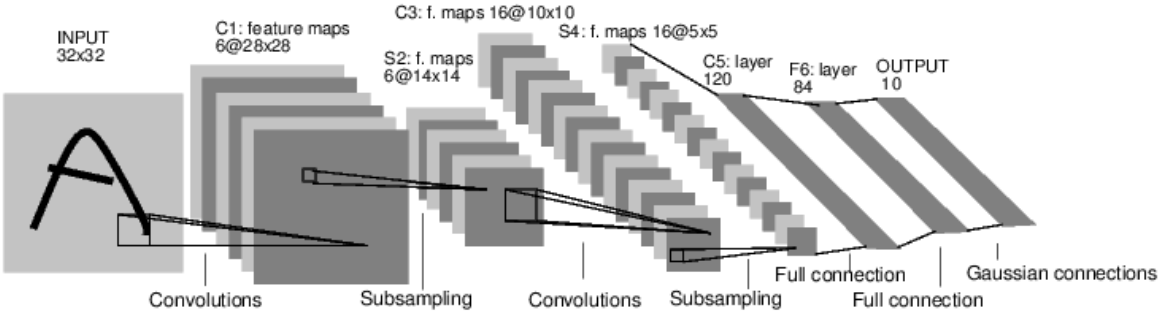
## 2.3 OBJECT DETECTION

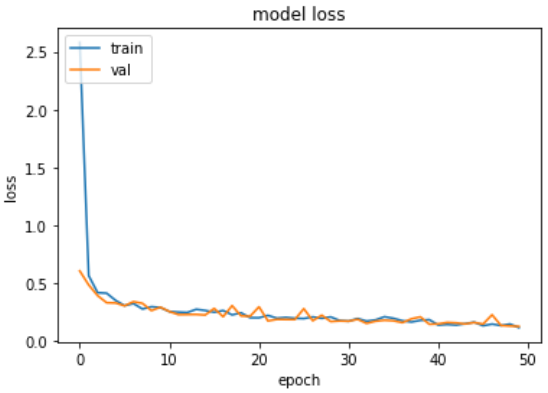
We learned how to detect faces using detectors such as Viola-Jones Face detector and FCNN

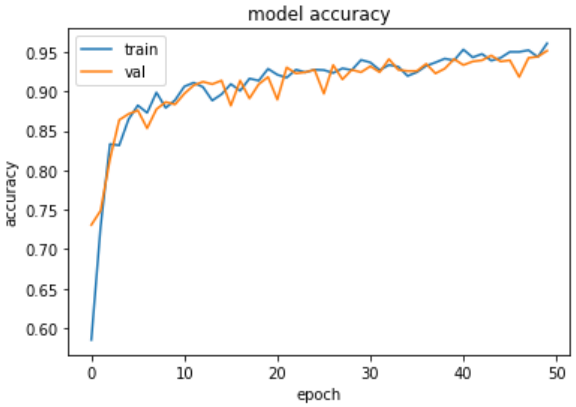
## **Assignment**: Face Detection

Task: Create a box encapsulating faces in a given image.

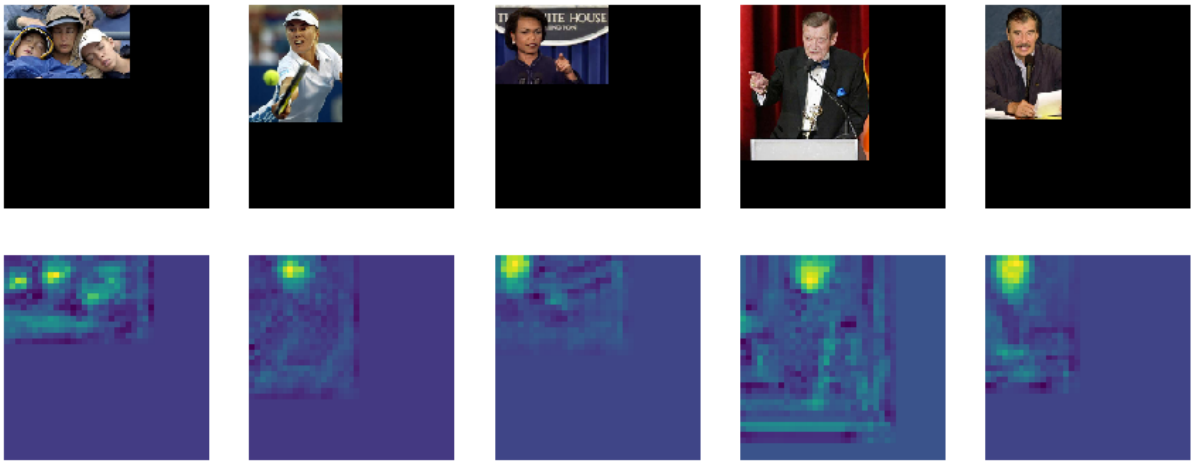
- Create a classifier, which tells if a face is present in the image using LeNet Architecture



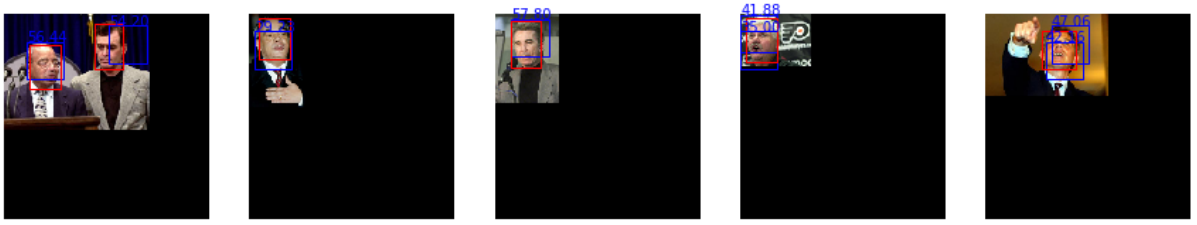




- Then, to detect where the face is,use fully convolutional layers instead of dense layers in the trained classifier to classify each pixel.

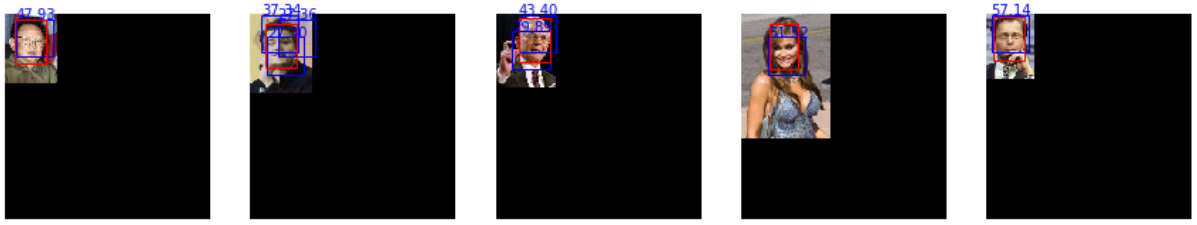


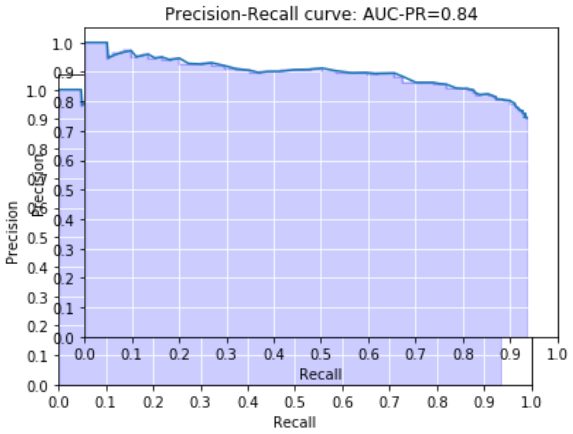
Then using this heatmap, we keep a threshold value above which we classify the pixel as a part of the face, and then we create a bounding box to cover the face.



Scores Used:IOU(Intersection over Union)

Results on Test Set:





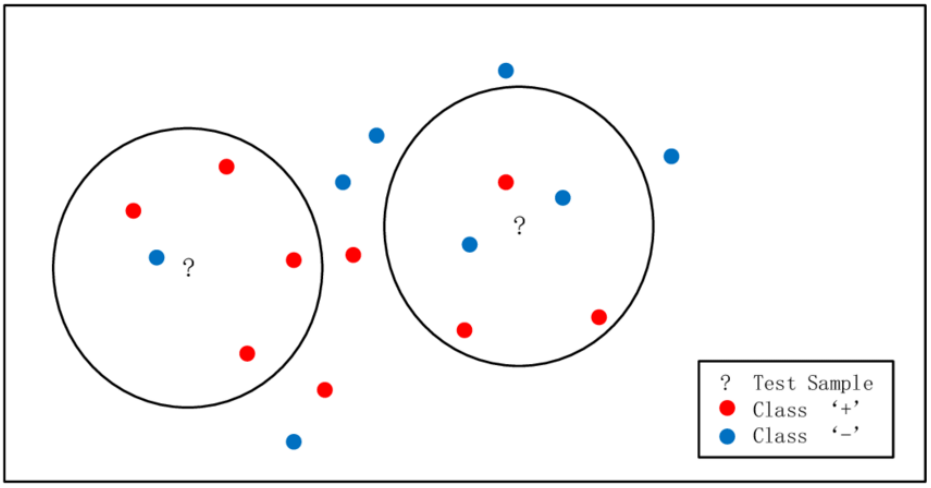

## 2.4 FACE RECOGNITION

We learned how to recognize faces on images and in videos using kNN.

**Assignment:** Face Recognition

- Preprocessing: Using the model trained in the previous assignment to detect faces in the image, we crop the images such that only faces remain.


- Then using kNN, we train a classifier.
  Parameters: n_neighbours=5

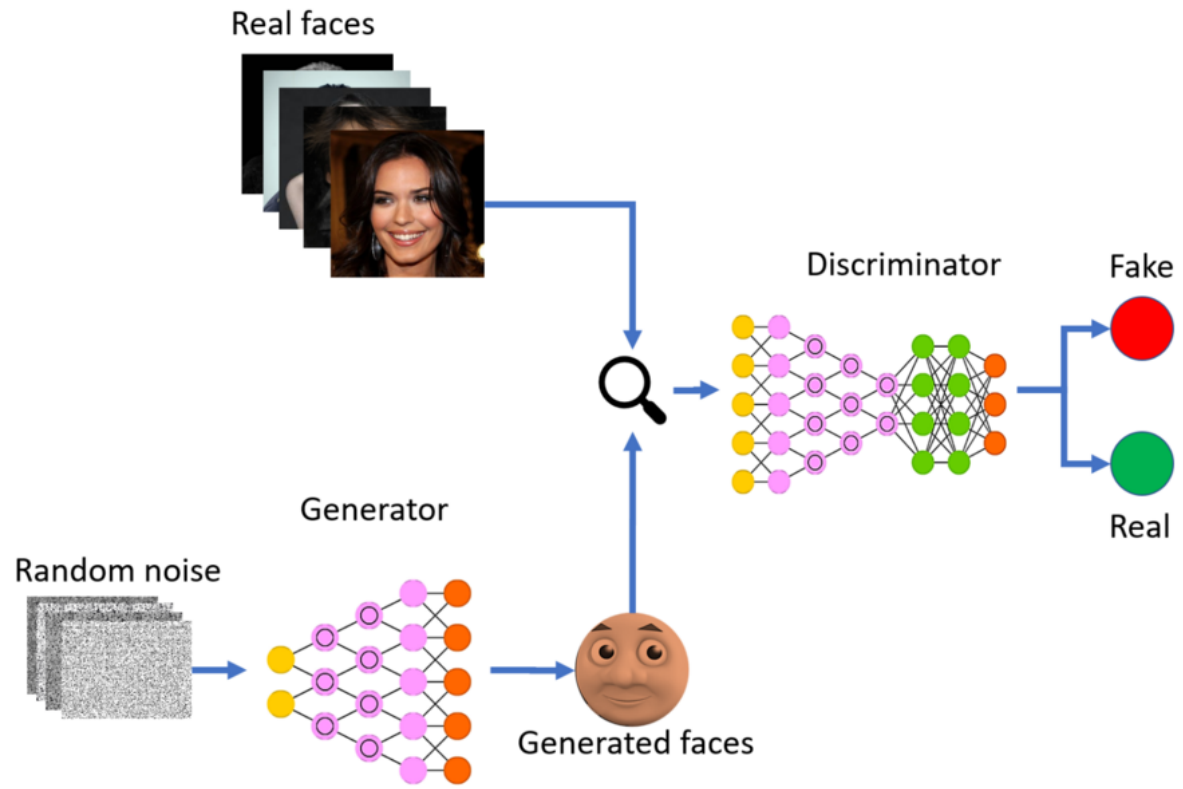



Results on the test set:


## 2.5 IMAGE SEGMENTATION AND SYNTHESIS

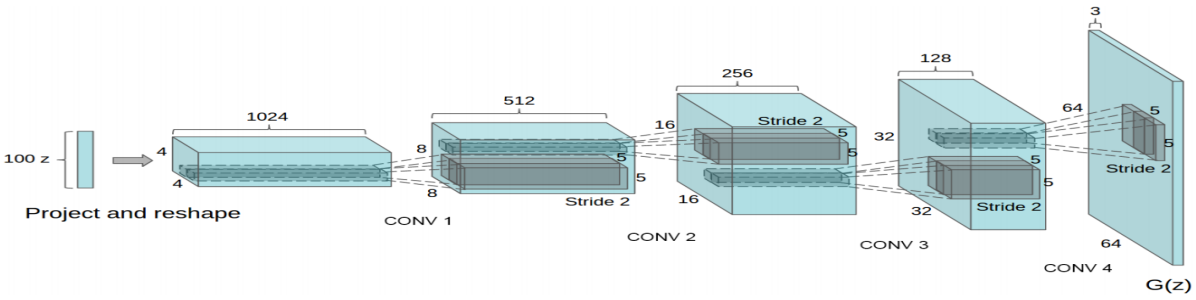
We learned how to perform semantic segmentation on an image and generate new images using Generative Adversarial Networks(GANs).
**Assignment**: Generate faces using GAN.

GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.




- Writing Generator and Discriminator Functions:

  Generator:

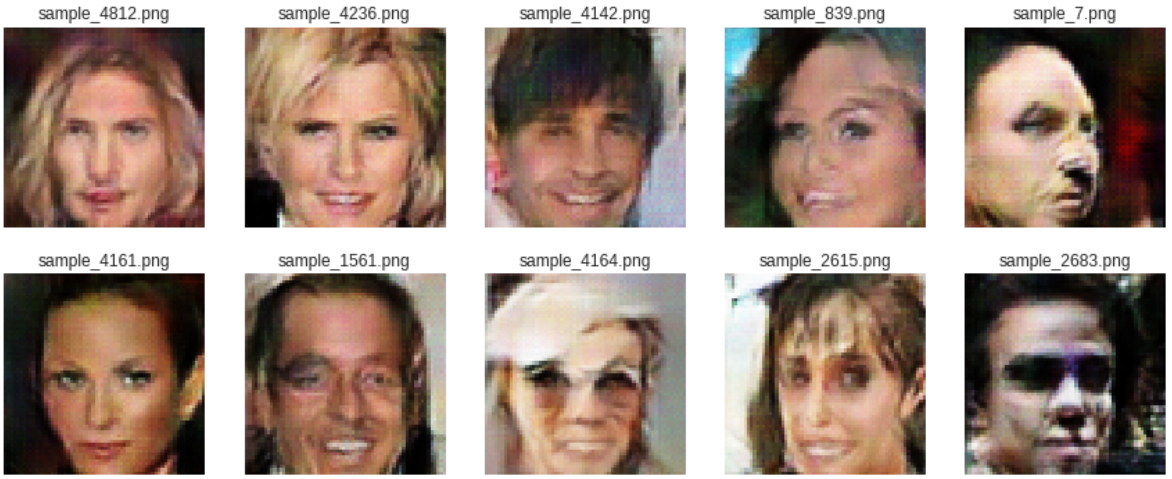



  Generator uses Deconvolution layers to convert a random noise vector into images, while the discriminator function uses convolution layers with dense layers to classify the image into fake or real.

- Writing Loss Function

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

- Train GAN and generate faces
  Results:




# 3. SEMANTIC SEGMENTATION

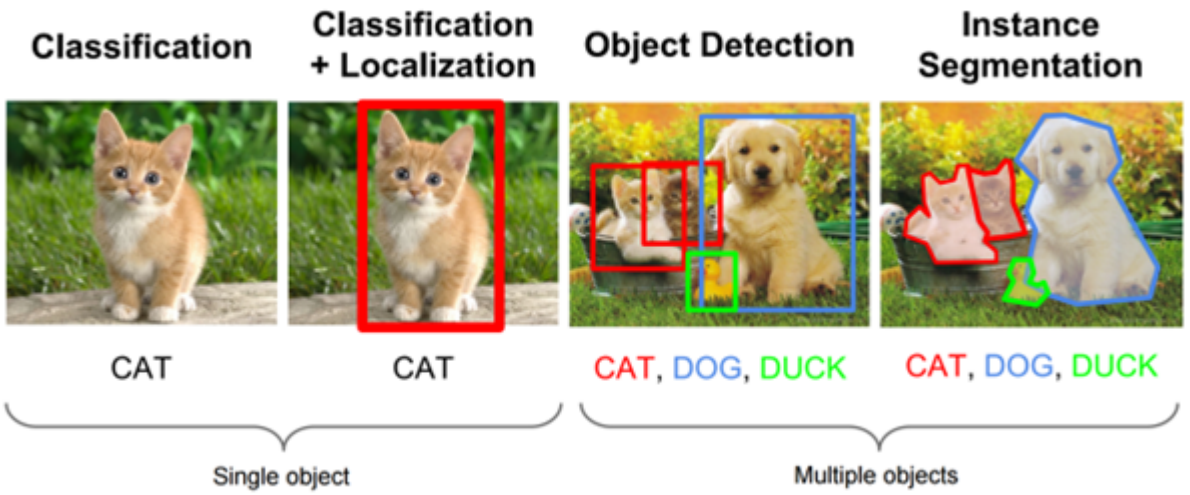
## 3.1. IMAGE SEGMENTATION

It is the process of segmenting an image and labelling it pixel-wise. With the help of this, understanding and analyzing image content has become easier.

### 3.1.1 LEVELS OF IMAGE INFORMATION EXTRACTION

Coarse to fine level of information of image

- Classification: It refers to the labelling of the whole image into predefined classes.
- Object Detection: In this level, classification and identifying position or localization of objects is done.
- Multi-Objects Detection: Identifying location of multiple objects in an image and labelling them into different classes.
- Image-Segmentation: Generating fine-grained image segmentation at pixel level and classifying them into classes.




Source: https://www.kdnuggets.com/2018/09/object-detection-image-classification-yolo.html

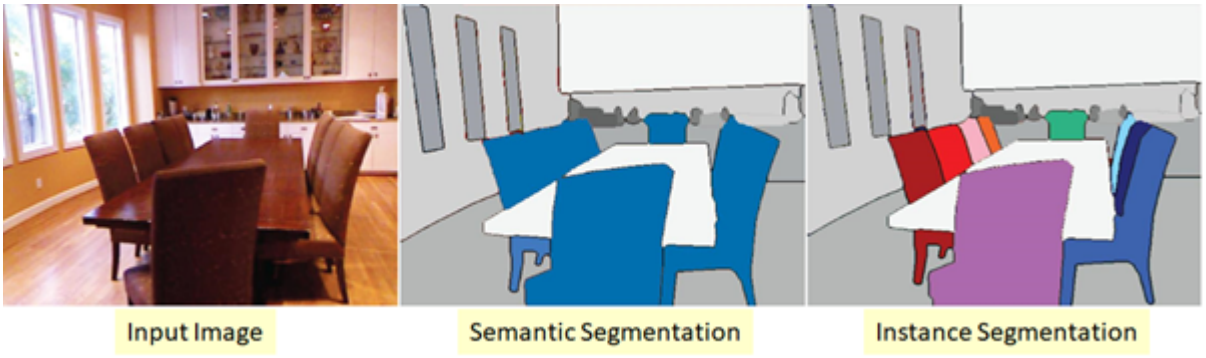
## 3.2. APPLICATION OF IMAGE SEGMENTATION

Now-a-days image segmentation is widely used in various applications because of its more practicality and high accuracy. Some examples are:

- Medical imaging tasks
- Object recognition in remote sensing images via satellite or aerial platforms
- Autonomous vehicle
- Automated traffic control system
- Scene understanding
- Image compression etc.

## 3.3. TYPES OF IMAGE SEGMENTATION

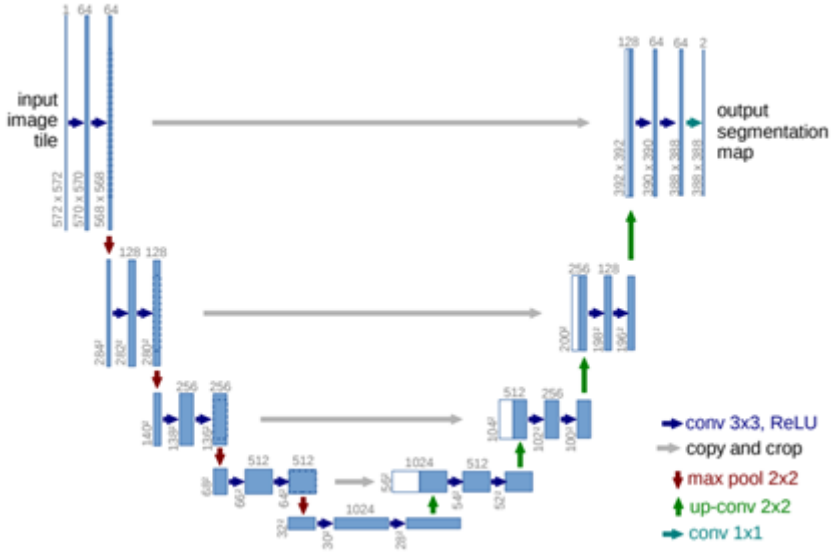
3.3.1 Semantic segmentation: It assigns the same label to all the objects of the same class.
3.3.2 Instant segmentation: It is an object level classification in which each object is classified as a different instance regardless of its class.




Source: https://towardsdatascience.com/review-deepmask-instance-segmentation-30327a072339

## 3.4. UNET

Its name is U-Net because its architecture contains a compressive path and an expansive path which is of "U" shape.



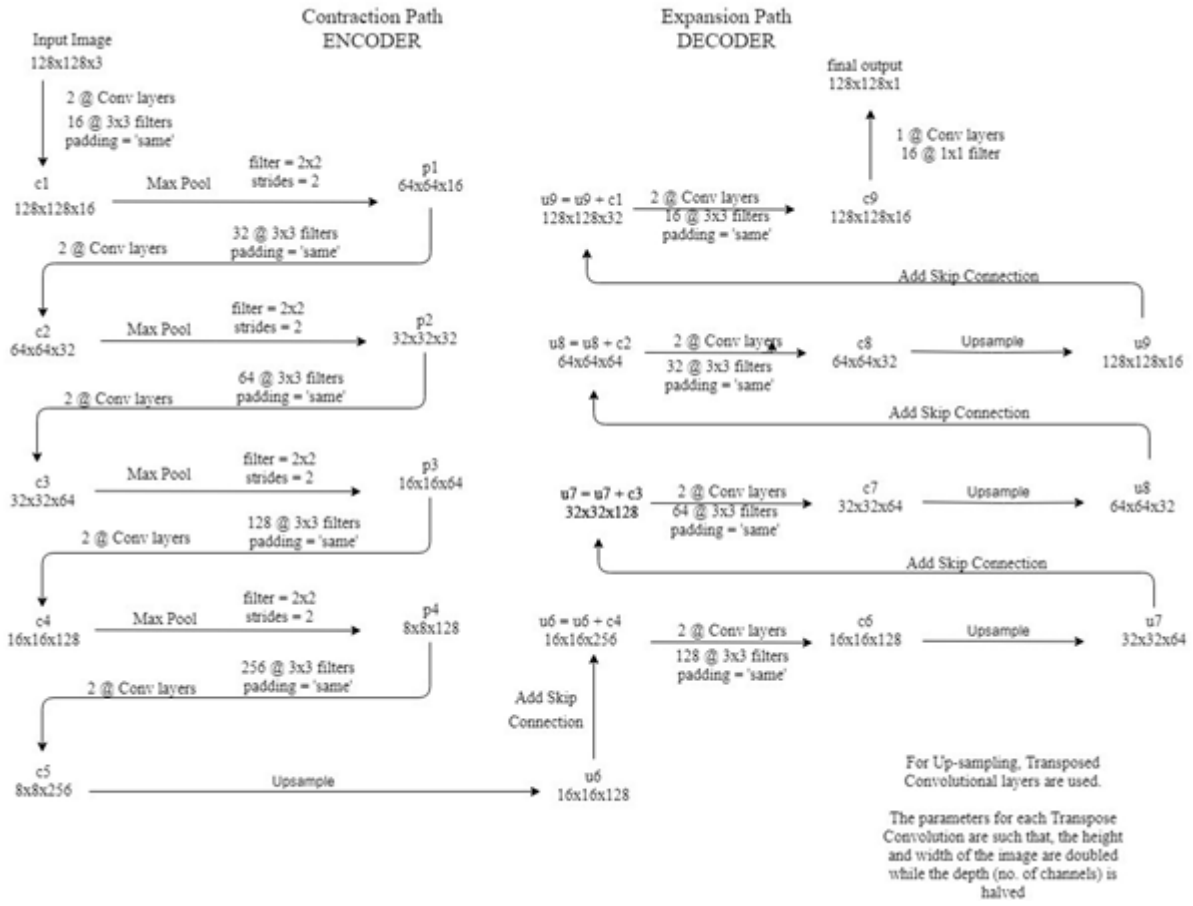

Source: https://medium.com/@abhishekkakiak/semantic-segmentation-unet-9b2444a22853

### 3.4.1 UNET Architecture

UNET is a popular architecture for image segmentation in BioMedical Image Segmentation. Architecture is made up of 2 sections, encoder and decoder.

- Encoder –Stack of convolutional and max pooling layers which capture the context in image.
- Decoder – Symmetrical expanding section which is used to enable precise localization using transposed convolutions.

The given diagram shows the detailed structure of UNET.




### 3.4.2 Implementation of UNET algorithm

*Problem statement*

Implement semantic segmentation for KITTI Dataset by UNET architecture. KITTI Dataset has labels for 3 classes (background, car, pedestrian). In this project work is on shape completion for Cars, so our network output segmentation requires labels only for background (0) and cars (1).

*Dataset*

KITTI semantic segmentation benchmark. It consists of 200 semantically annotated train as well as 200 test images corresponding to the KITTI Stereo and Flow Benchmark 2015. KITTI Dataset published for Autonomous Driving.

*Data augmentation*

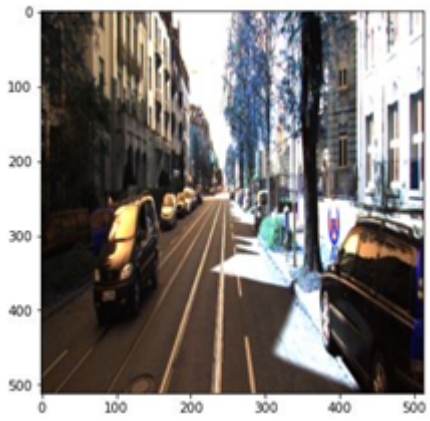
As KITTI dataset has only 200 training images which are not sufficient for data hungry algorithms like image segmentation. So, data augmentation is applied on this data, like

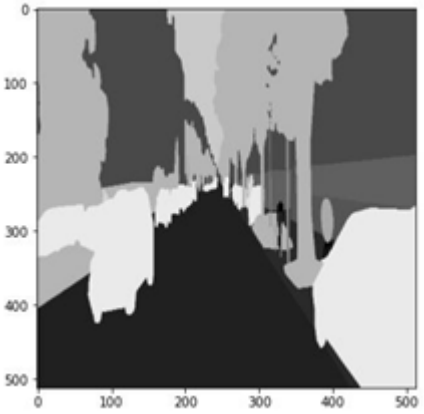
- Horizontal flip
- Vertical flip
- Elastic Transformation
- Grid Distortion
- Optical Distortion

By using data augmentation library “albumentations”, training data of 200 images is increased to 1200 images.

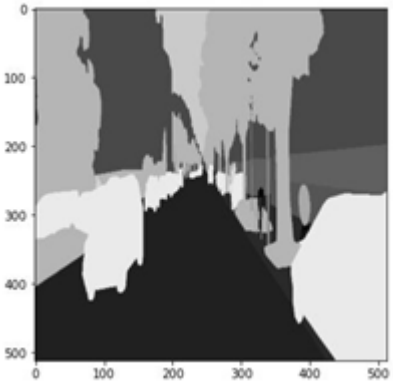
*Data pre-processing*

- Resize image to (512,512) and convert mask image to grayscale image generating sharpness.



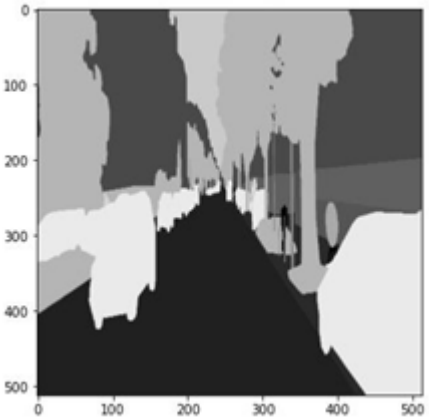


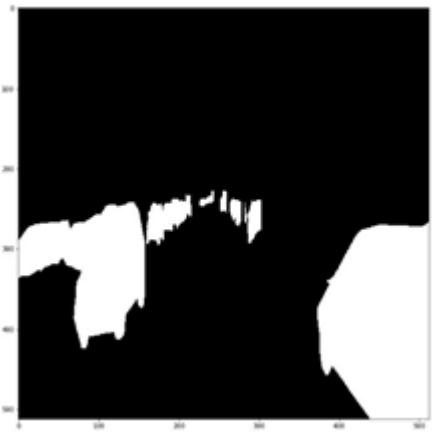
- Resize the image to (16,16)to see the relation between image (512,512) and matrix (16,16). Here we can easily see the car has a grayscale value of 26 and the road has a grey scale value of 7.
  We can do similarly for others also.



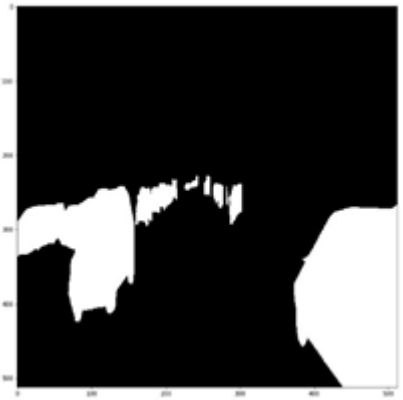
```
[[21 21 11 11 11 11 23 23 21 21 21 21 21 11 11 11]
 [21 21 14 11 11 11 23 23 21 21 21 21 21 11 11 11]
 [21 21 21 11 11 11 23 23 21 21 21 21 11 11 11 11]
 [21 21 21 11 11 11 23 23 21 21 21 21 11 11 11 11]
 [21 21 21 11 11 11 11 23 21 21 21 21 11 11 11 11]
 [21 21 21 11 11 11 21 23 21 21 11 21 11 11 11 11]
 [21 21 21 11 11 11 21 20 16 21 11 21 11 13 13 13]
 [21 21 21 21 21 21 18 26 26 21 13 21 13 13 13 13]
 [25 26 26 26 26 26  7  7  8 22 10 21 13 12 26 22]
 [26 26 26 26 26  7  7  7  7 21 21 21 14 26 26 26]
 [21 21 26 26 26  7  7  7  7  7  8 21 26 26 26 26]
 [21 21 26 26 24  7  7  7  7  7  8 21 26 26 26 26]
 [ 7  7 26 26  7  7  7  7  7  7  7  8 26 26 26 26]
 [ 7  7  7  7  7  7  7  7  7  7  7  7 26 26 26 26]
 [ 7  7  7  7  7  7  7  7  7  7  7  7  8 26 26 26]
 [ 7  7  7  7  7  7  7  7  7  7  7  7  7 26 26 26]]
```

- As the problem is of binary classification, we have to convert grayscale image to binary image (0,1).





- Resize the image to (16,16) to see the relation between image (512,512) and matrix (16,16). Here we can easily visualize 1 is assigned to the car and 0 is assigned to the background.



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

- Split data in train and validation with splitting ratio =0.2

  Train data- 960

  Validation data- 240

  Setting hyperparameters
  - o Batch size = 3
  - o Optimization algorithm = Adam
  - o Learning Rate = 1e-4
  - o No of epochs =25
  - o Metrics – Dice coefficient, IOU, Recall and Precision
  - o Loss function – Binary Cross Entropy + Dice loss

***Results***

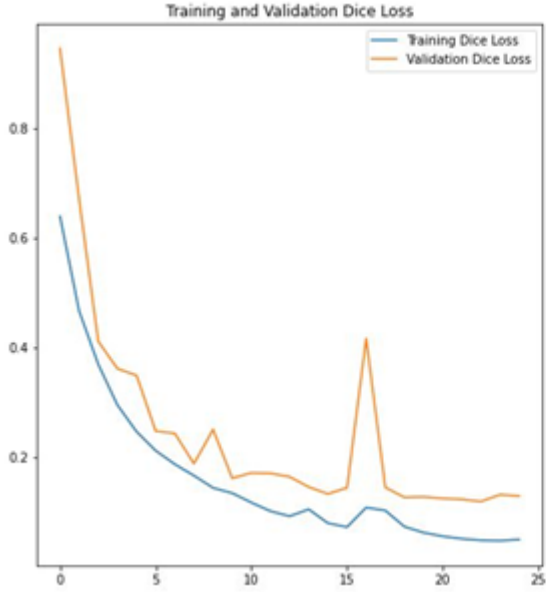
- After 25 epochs – Training, validation respectively

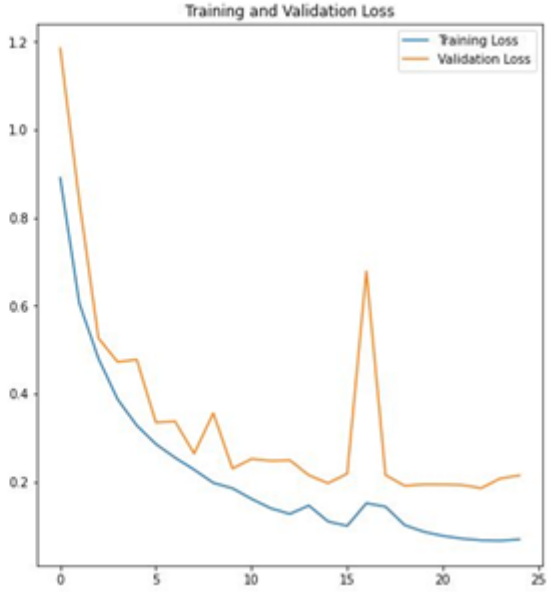
  IOU – 0.8531 , 0.7709

  Dice Coefficient – 0.9199 , 0.8666

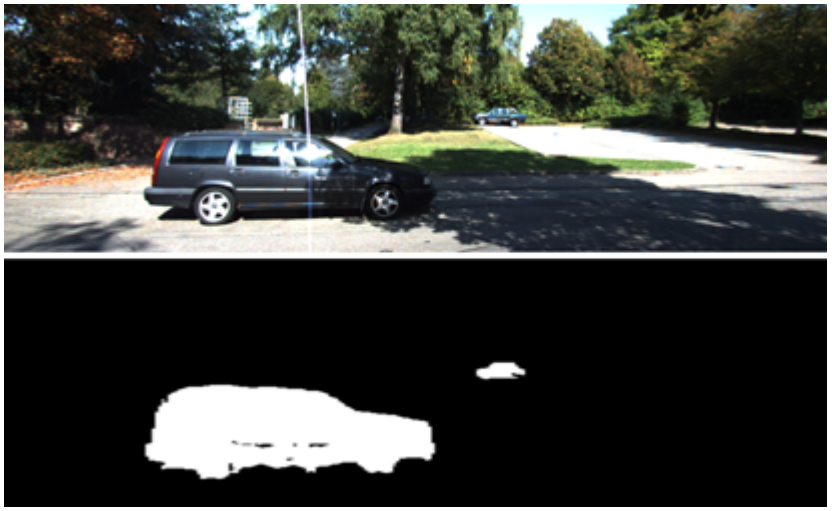
  Recall – 0.9361 , 0.8251

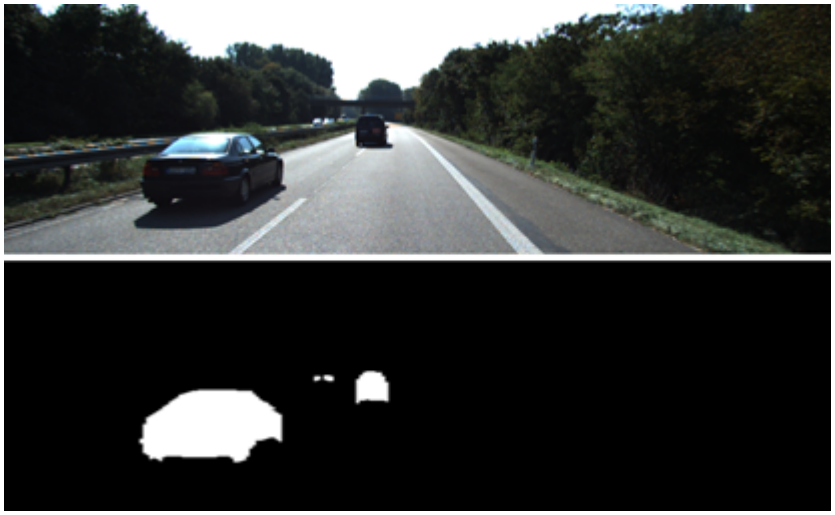
  Precision – 0.9407 , 0.9429



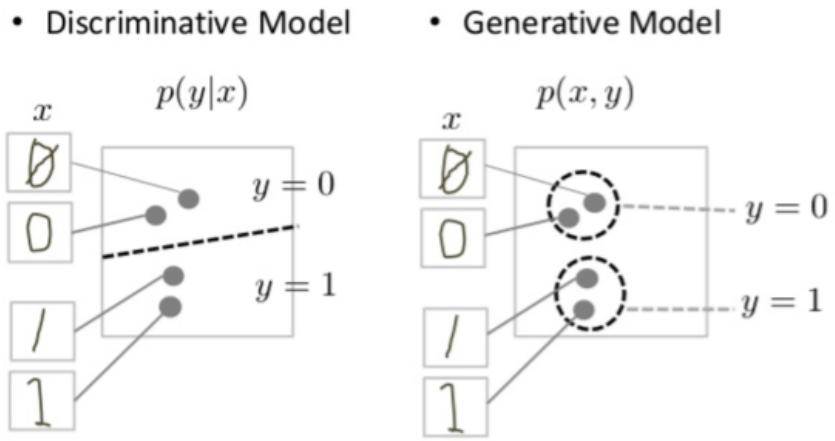
### 3.4.3 RESULTS FROM TEST IMAGES



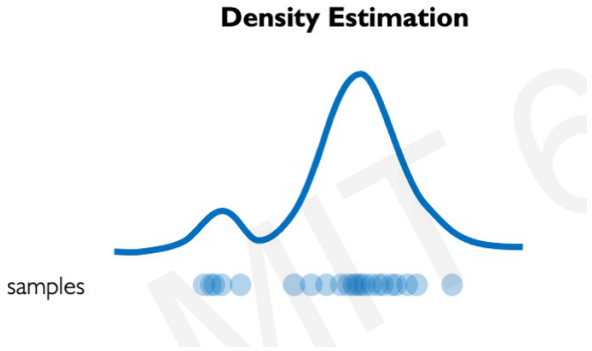


# Generative Modelling:

Generative modeling involves using a model to generate new examples that plausibly come from an existing distribution of samples, such as generating new photographs that are similar but specifically different from a dataset of existing photographs.
In short, Generative models can generate new data instances.



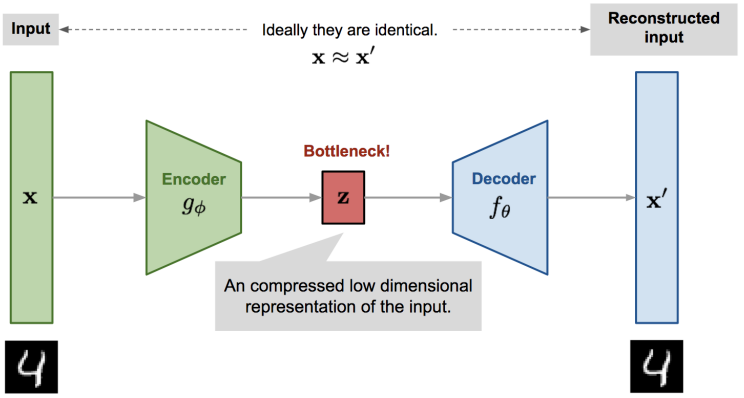

The highlighting difference in between a discriminative model and a generative model, is discriminative model creates
We need to create a model that takes input from a particular distribution and learns from data that fits that distribution.




The peaks of this curve indicate the maximum type of data we have for the input. For example in driving data, about 95% of the data is clean, with some distinctive features, like sunny weather, highway and straight roads. In this example, for a model to be able to properly drive on roads, autonomously, we need it to even work well with outliers, which may include and are not restricted to the following: road blocks either intentional or accidents, pedestrians on the road, harsh weather conditions. Our model has to be trained well even with these outliers, to avoid any unpredictable behaviour.
Auto Encoders
An auto-encoder is an unsupervised deep neural network that aims to encode an image to eliminate noise, reduce dimensions, or perform other processing tasks, and then attempt to reconstruct the original image from the encoded version.

Every autoencoder has the following major characteristics:
Encoder : The layer that reduces the input dimensions and compresses the input data.

- Bottleneck : The layer that contains the encoded data.
- Decoder : The layer that attempts to reconstruct the original data from the encoded one.




The training involves a loss function to check how different the reconstruction is from the original, and backpropagation to modify the parameters of the various layers.

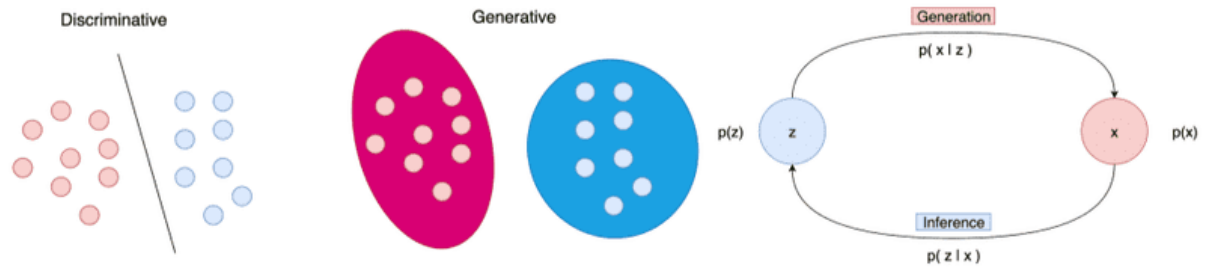
# Latent Variables

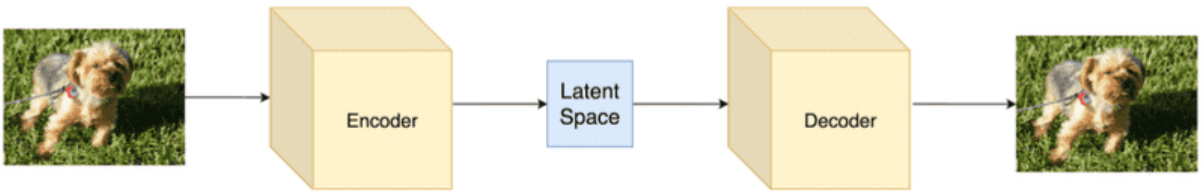
Latent variables are variables in the data that we feed to our models that are not directly observed or measurable (at least, not practically). They represent a kind of hidden structure to the data, and any model that can learn to recognize these latent variables can have a greater accuracy even after being trained on a relatively biased dataset.

For example, in a dataset consisting of  human faces, accounting for latent variables along with training for recognition of the model will reduce the bias that the model might have in recognizing faces of different genders, races, etc.

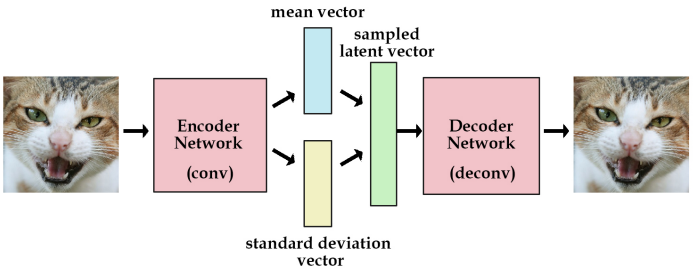
The **latent space**  is the probability distributions of the mentioned latent variables.

# Variational Auto-Encoders (VAEs)



Real world data can have lots of unintended biases, therefore a simple classifier model that directly trains on the data given is likely to have some biases. Our goal is to train a debiased version of this classifier -- one that accounts for potential disparities in feature representation within the training data. Specifically, to build a debiased classifier, we'll train a model that learns a representation of the underlying latent space to the training data. The key design requirement for our model is that it can learn an encoding of the latent features in the data in an entirely unsupervised way. To achieve this, we'll turn to variational autoencoders (VAEs).

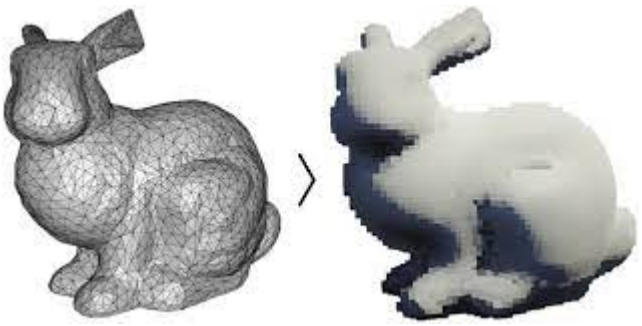



VAEs rely on an encoder-decoder structure to learn a latent representation of the input data. In the context of computer vision, the encoder network takes in input images, encodes them into a series of variables defined by a mean and standard deviation, and then draws from the distributions defined by these parameters to generate a set of sampled latent variables. The decoder network then "decodes" these variables to generate a reconstruction of the original image, which is used during training to help the model identify which latent variables are important to learn.

# Voxelization

Voxelization is the term given to the process of converting data from one source type into a three dimensional volume of data values. The techniques known collectively as volume visualisation can then be applied to the data in order to produce a graphical representation of the object. Generally objects fall into one of two categories for voxelization: Solid Models and Surface Representations. In the case of solid modelling, the notion of interior, surface and exterior is available for each object. Solid objects could be defined as a closed collection of polygons representing the surface, as constructive solid geometry, by sweeping or extruding, as octrees indicating spatial occupancy, implicitly, and using patches.

In the other method of voxelization, a 2D surface is developed into a hollow 3D model, for example a plane becomes a box, and so on.



# References


1. Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation.* 2015
2. Introduction to Keras API
3. MIT 6.S094: Computer Vision, Lecture by Lex Fridman
4. Y. Weng, T. Zhou, Y. Li and X. Qiu, "*NAS-Unet: Neural Architecture Search for Medical Image Segmentation*". 2019
5. Auto Encoders