Capstone Project Report

Self-Driving Car

Name: Arun Govind

Course: AI and ML (Batch 4)

- **Project Statement**

  Implementation of a self-driving car involves the use of many external sensors and a lot of dependence of Neural networks for object detection for understanding the nearby traffic, environment and safety signs. However, one of the most crucial task is to make the car drive in one particular lane. Whenever we drive, these lanes guide us where a road takes us and acts as reference to steer the vehicle. In this project we aim to tackle the problem of lane detection given a set of images from a car dashboard or videos.

  a) Implement an auto-encoder model/ U-Net model that takes the image as input and outputs the images with the lanes marked. You may use python along with OpenCV to implement this.

- **Prerequisites**

- Software:
  - Python 3 (Use anaconda as your python distributor as well)
  - Kaggle (Usage of GPU's and accelerator)

- Tools:
  - Pandas
  - Numpy
  - Matplotlib
  - PIL
  - Random
  - Glob
  - Seaborn
  - TensorFlow
  - Cv2
  - OS

- Dataset: Cityscapes Image Pairs dataset contains labeled videos taken from vehicles driven in Germany.

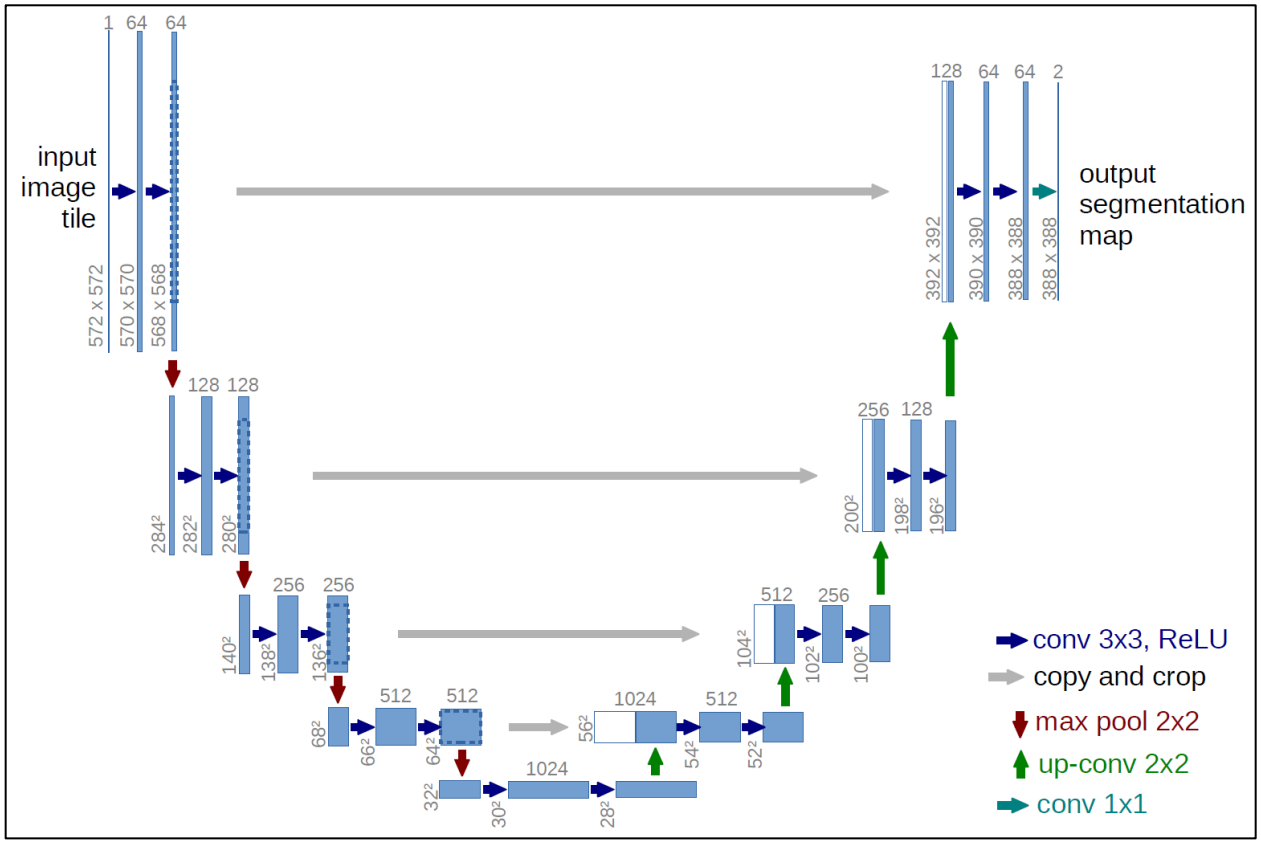
- **Method Used**

Self-driving cars require a deep understanding of their surroundings. To support this, camera frames are used to recognize the road, pedestrians, cars, and sidewalks at a pixel-level accuracy. In this project, we develop a neural network and optimize it to perform semantic segmentation using U-Net and Segnets.

Semantic segmentation is the task of assigning meaning of part of an object. this can be done at the pixel level where we assign each pixel to a target class such as road, car, pedestrian, sign or any number of other classes.

Some applications include autonomous driving, scene understanding, etc. Direct adoption of classification networks for pixel wise segmentation yields poor results mainly because max-pooling and subsampling reduce feature map resolution and hence output resolution is reduced. Even if extrapolated to original resolution, lossy image is generated.

The u-net is convolutional network architecture for fast and precise segmentation of images.



Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

One important modification in U-Net is that there are a large number of feature channels in the upsampling part, which allow the network to propagate context information to higher resolution layers. As a consequence, the expansive path is more or less symmetric to the contracting part, and yields a u-shaped architecture. The network only uses the valid part of each convolution without any fully connected layers.

There are many applications of U-Net in biomedical image segmentation, such as brain image segmentation ("BRATS") and liver image segmentation ("siliver07"). Variations of the U-Net have also been applied for medical image reconstruction.[6] Here are some variants and applications of U-Net as follows:

1. Pixel-wise regression using U-Net and its application on pansharpening.
2. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation.
3. TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation.
4. Image-to-image translation to estimate fluorescent stains.

- **Implementation**

1. Load the required libraries

```
import numpy as np
import pandas as pd
import os
import random
import glob
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from tensorflow.keras.layers import *
from tensorflow.keras import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import backend as K
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
```

2. Creation of Utility Functions

```
def loadImage(path):
    img = Image.open(path)
    img = np.array(img)

    image = img[:,:256]
    image = image / 255.0
    mask = img[:,256:]

    return image, mask

def bin_image(mask):
    bins = np.array([20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240])
    new_mask = np.digitize(mask, bins)
    return new_mask

def getSegmentationArr(image, classes, width=WIDTH, height=HEIGHT):
    seg_labels = np.zeros((height, width, classes))
    img = image[:, : , 0]

    for c in range(classes):
        seg_labels[:, :, c] = (img == c ).astype(int)
    return seg_labels

def give_color_to_seg_img(seg, n_classes=N_CLASSES):

    seg_img = np.zeros( (seg.shape[0],seg.shape[1],3) ).astype('float')
    colors = sns.color_palette("hls", n_classes)
```

3. Assign Train and Test data

```
train_folder = "../input/cityscapes-image-pairs/cityscapes_data/train"
valid_folder = "../input/cityscapes-image-pairs/cityscapes_data/val"
train_filenames = glob.glob(os.path.join(train_folder, "*.jpg"))
valid_filenames = glob.glob(os.path.join(valid_folder, "*.jpg"))

num_of_training_samples = len(train_filenames)
num_of_valid_samples = len(valid_filenames)
```

4. Applying Segmentation Mask and Masked Image

```
image = imgs[0]
mask = give_color_to_seg_img(segs[0])
masked_image = image * 0.5 + mask * 0.5

fig, axs = plt.subplots(1, 3, figsize=(20,20))
axs[0].imshow(image)
axs[0].set_title('Original Image')
axs[1].imshow(mask)
axs[1].set_title('Segmentation Mask')
#predimg = cv2.addWeighted(imgs[i]/255, 0.6, _p, 0.4, 0)
axs[2].imshow(masked_image)
axs[2].set_title('Masked Image')
plt.show()
```

5. Creating Loss Function

```
smooth = 1

def dice_coef(y_true, y_pred):
    And = tf.reduce_sum(y_true * y_pred)
    return (2 * And + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

def dice_coef_loss(y_true, y_pred):
    return 1 - dice_coef(y_true, y_pred)

def iou(y_true, y_pred):
    intersection = tf.reduce_sum(y_true * y_pred)
    sum_ = tf.reduce_sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac

def jac_distance(y_true, y_pred):
    return 1 - iou(y_true, y_pred)
```

6. Designing Model

```
def unet():
    main_input = Input(shape=(HEIGHT, WIDTH, CHANNELS), name = 'img_input')

    ''' ~~~~~~~~~~~~~~~~~~~ ENCODING LAYERS ~~~~~~~~~~~~~~~~~~~ '''

    c1 = Conv2D(32, kernel_size=(3,3), padding = 'same')(main_input)
    c1 = LeakyReLU(0.2)(c1)
    c1 = BatchNormalization()(c1)
    c1 = Conv2D(32, kernel_size=(3,3), padding = 'same')(c1)
    c1 = LeakyReLU(0.2)(c1)
    c1 = BatchNormalization()(c1)

    p1 = MaxPooling2D((2,2))(c1)

    c2 = Conv2D(32*2, kernel_size=(3,3), padding = 'same')(p1)
    c2 = LeakyReLU(0.2)(c2)
    c2 = BatchNormalization()(c2)
    c2 = Conv2D(32*2, kernel_size=(3,3), padding = 'same')(c2)
    c2 = LeakyReLU(0.2)(c2)
    c2 = BatchNormalization()(c2)

    p2 = MaxPooling2D((2,2))(c2)

    c3 = Conv2D(32*4, kernel_size=(3,3), padding = 'same')(p2)
    c3 = LeakyReLU(0.2)(c3)
    c3 = BatchNormalization()(c3)
    c3 = Conv2D(32*2, kernel_size=(1,1), padding = 'same')(c3)
    c3 = LeakyReLU(0.2)(c3)
```

```
def down_block(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(x)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    p = MaxPooling2D((2, 2))(c)
    return c, p

def up_block(x, skip, filters, kernel_size=(3, 3), padding="same", strides=1):
    us = UpSampling2D((2, 2))(x)
    concat = Concatenate()([us, skip])
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(concat)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    return c

def bottleneck(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(x)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    return c
```

7. Training model under 20 epochs

```
TRAIN_STEPS = len(train_gen)
VAL_STEPS = len(val_gen)

model.compile(optimizer="adam", loss='sparse_categorical_crossentropy', metrics=["accuracy"])
history = model.fit(train_gen, validation_data=val_gen, steps_per_epoch=TRAIN_STEPS,
          validation_steps=VAL_STEPS, epochs=EPOCHS, callbacks = [checkpoint])
```

8. Training model under a learning rate

```
learning_rate = 1e-4
decay_rate = (learning_rate - 1e-6) / 100
opt = tf.keras.optimizers.Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=None, decay=decay_rate, amsg

checkpoint = ModelCheckpoint('seg_model_v2.h5', monitor='val_loss', verbose=1, save_best_only=True, save_weight
model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=["accuracy"])
history2 = model.fit(train_gen, validation_data=val_gen, steps_per_epoch=TRAIN_STEPS,
          validation_steps=VAL_STEPS, epochs=10, callbacks = [checkpoint])
```

9. Plot training history

```
loss = history.history["val_loss"]
acc = history.history["val_accuracy"]

plt.figure(figsize=(12, 10))
plt.subplot(211)
plt.title("Val. cce Loss")
plt.plot(loss)
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.subplot(212)
plt.title("Val. Accuracy")
plt.plot(acc)
plt.xlabel("Epoch")
plt.ylabel("Accuracy")

plt.tight_layout()
plt.show()
```

10. Addition of Weights

```
model.load_weights("./seg_model_v2.h5")
```

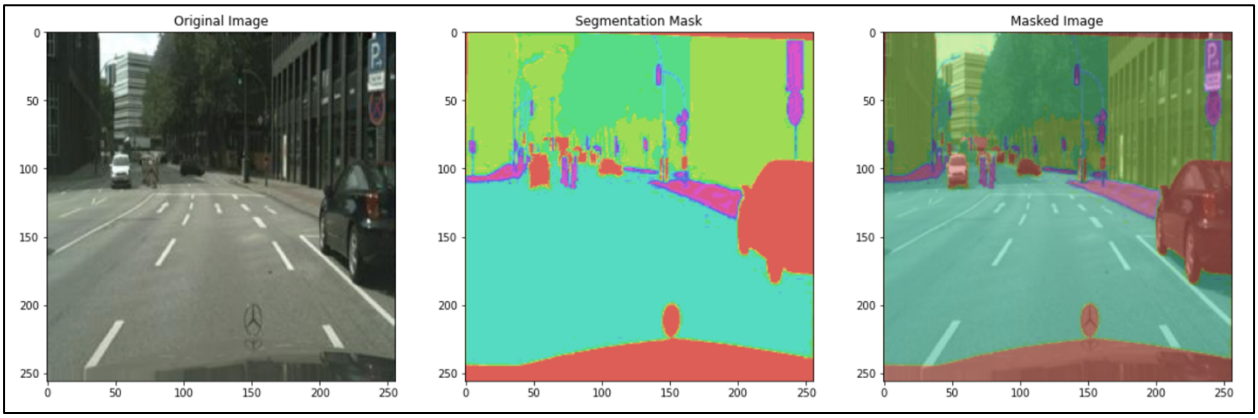
```
test_gen = DataGenerator(valid_filenames, 1)
```

11. Visualize Outputs

```
test_iter = iter(test_gen)
for i in range(12):
    imgs, segs = next(test_iter)
    pred = model.predict(imgs)
    _p = give_color_to_seg_img(np.argmax(pred[0], axis=-1))
    _s = give_color_to_seg_img(segs[0])

    predimg = cv2.addWeighted(imgs[0], 0.5, _p, 0.5, 0)
    trueimg = cv2.addWeighted(imgs[0], 0.5, _s, 0.5, 0)

    plt.figure(figsize=(12, 4))
    plt.subplot(131)
    plt.title("Original")
    plt.imshow(imgs[0])
    plt.axis("off")

    plt.subplot(132)
    plt.title("Prediction")
    plt.imshow(predimg)
    plt.axis("off")

    plt.subplot(133)
    plt.title("Ground truth")
    plt.imshow(trueimg)
    plt.axis("off")
    plt.tight_layout()
    plt.savefig("pred_"+str(i)+".png", dpi=150)
    plt.show()
```

- **Results**

1. Plot of Original, Segmented and Masked Image



2. Model Summary

```
Model: "model_1"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
img_input (InputLayer)       [(None, 256, 256, 3) 0
_________________________________________________________________________________________________
conv2d_18 (Conv2D)           (None, 256, 256, 32) 896         img_input[0][0]
_________________________________________________________________________________________________
leaky_re_lu_18 (LeakyReLU)   (None, 256, 256, 32) 0           conv2d_18[0][0]
_________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 256, 256, 32) 128         leaky_re_lu_18[0][0]
_________________________________________________________________________________________________
conv2d_19 (Conv2D)           (None, 256, 256, 32) 9248        batch_normalization_18[0][0]
_________________________________________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 256, 256, 32) 0           conv2d_19[0][0]
_________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 256, 256, 32) 128         leaky_re_lu_19[0][0]
_________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 128, 128, 32) 0           batch_normalization_19[0][0]
_________________________________________________________________________________________________
conv2d_20 (Conv2D)           (None, 128, 128, 64) 18496       max_pooling2d_4[0][0]
_________________________________________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 128, 128, 64) 0           conv2d_20[0][0]
_________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 128, 128, 64) 256         leaky_re_lu_20[0][0]
_________________________________________________________________________________________________
conv2d_21 (Conv2D)           (None, 128, 128, 64) 36928       batch_normalization_20[0][0]
_________________________________________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 128, 128, 64) 0           conv2d_21[0][0]
_________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 128, 128, 64) 256         leaky_re_lu_21[0][0]
_________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 64, 64, 64)   0           batch_normalization_21[0][0]
```

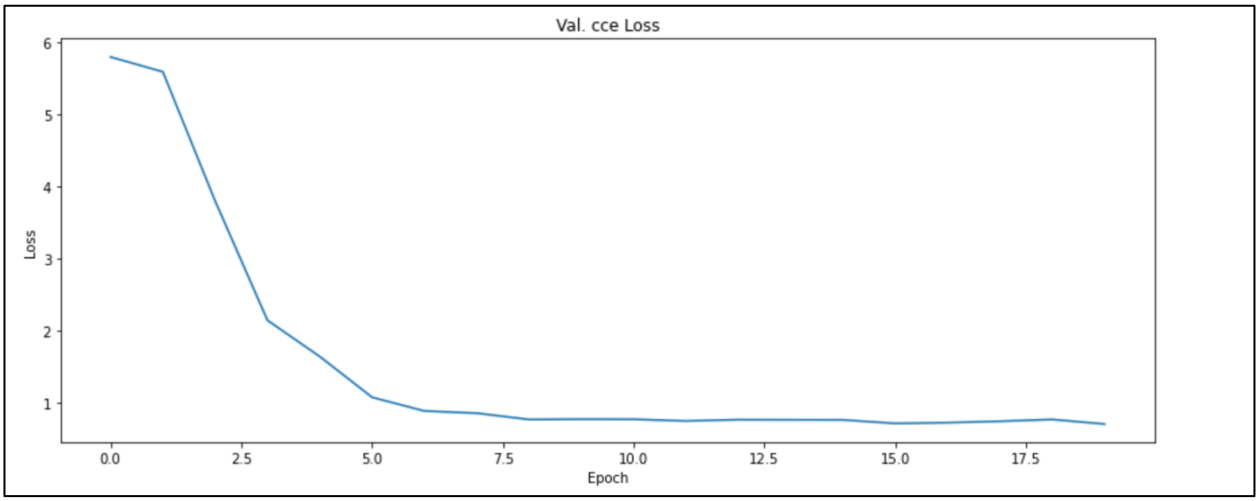
3. Result of first 5 EPOCH Calculation after each Epoch

```
Epoch 1/20
92/92 [==============================] - 50s 517ms/step - loss: 1.6651 - accuracy: 0.5625 - val_loss: 5.7921 - val_accura
cy: 0.3550

Epoch 00001: val_loss improved from inf to 5.79213, saving model to seg_model.h5
Epoch 2/20
92/92 [==============================] - 47s 509ms/step - loss: 0.9929 - accuracy: 0.7367 - val_loss: 5.5872 - val_accura
cy: 0.3550

Epoch 00002: val_loss improved from 5.79213 to 5.58723, saving model to seg_model.h5
Epoch 3/20
92/92 [==============================] - 48s 518ms/step - loss: 0.9106 - accuracy: 0.7521 - val_loss: 3.7981 - val_accura
cy: 0.3712

Epoch 00003: val_loss improved from 5.58723 to 3.79805, saving model to seg_model.h5
Epoch 4/20
92/92 [==============================] - 47s 506ms/step - loss: 0.8453 - accuracy: 0.7656 - val_loss: 2.1480 - val_accura
cy: 0.5050

Epoch 00004: val_loss improved from 3.79805 to 2.14801, saving model to seg_model.h5
Epoch 5/20
92/92 [==============================] - 47s 510ms/step - loss: 0.7967 - accuracy: 0.7776 - val_loss: 1.6469 - val_accura
cy: 0.5953

Epoch 00005: val_loss improved from 2.14801 to 1.64689, saving model to seg_model.h5
Epoch 6/20
92/92 [==============================] - 47s 506ms/step - loss: 0.7507 - accuracy: 0.7874 - val_loss: 1.0829 - val_accura
cy: 0.6968
```

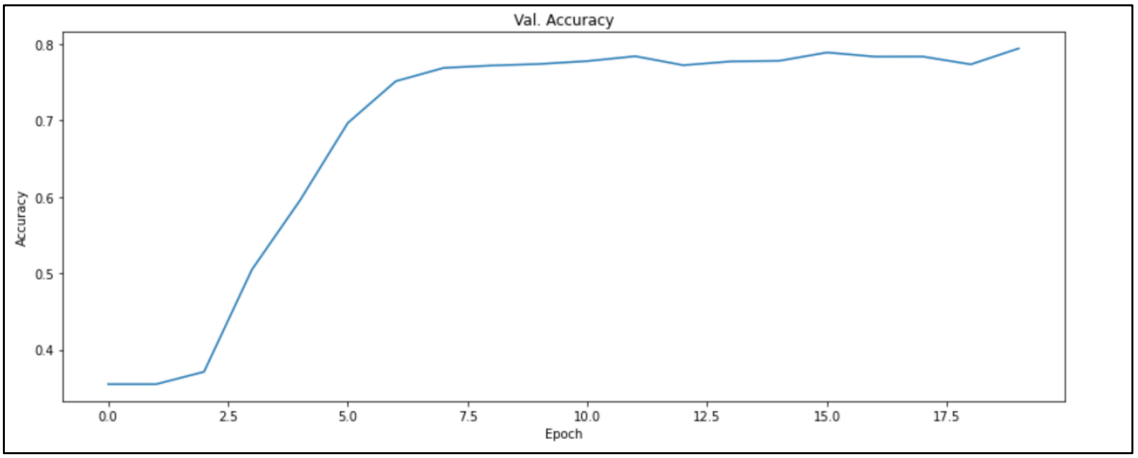
4. Result of first 5 EPOCH after Learning rate is implemented

```
Epoch 1/10
92/92 [==============================] - 50s 519ms/step - loss: 0.5494 - accuracy: 0.8318 - val_loss: 0.6794 - val_accura
cy: 0.7974

Epoch 00001: val_loss improved from inf to 0.67937, saving model to seg_model_v2.h5
Epoch 2/10
92/92 [==============================] - 47s 510ms/step - loss: 0.5321 - accuracy: 0.8363 - val_loss: 0.6875 - val_accura
cy: 0.7993

Epoch 00002: val_loss did not improve from 0.67937
Epoch 3/10
92/92 [==============================] - 48s 515ms/step - loss: 0.5323 - accuracy: 0.8356 - val_loss: 0.6882 - val_accura
cy: 0.7978

Epoch 00003: val_loss did not improve from 0.67937
Epoch 4/10
92/92 [==============================] - 47s 505ms/step - loss: 0.5230 - accuracy: 0.8380 - val_loss: 0.6937 - val_accura
cy: 0.7969

Epoch 00004: val_loss did not improve from 0.67937
Epoch 5/10
92/92 [==============================] - 48s 526ms/step - loss: 0.5172 - accuracy: 0.8398 - val_loss: 0.6952 - val_accura
cy: 0.7962
```

5. Graph of Loss calculated after each EPOCH



6. Graph of Accuracy calculated after each EPOCH



7. Output visualization against True and Predicted Value