Instructions

- Some parts of the code are already done for you
- You need to execute all the cells
- You need to add the code where ever you see `"#### Add your code here ####"`
- Marks are mentioned along with the cells

Face detection

Task is to predict the boundaries(mask) around the face in a given image.

Dataset

Faces in images marked with bounding boxes. Have around 500 images with around 1100 faces man

Mount Google drive if you are using google colab

- We recommend using Google Colab as you can face memory issues and longer runtimes while

```
from google.colab import drive
drive.mount('/content/drive/mydrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

Change current working directory to project folder (1 mark)

```
import os
os.chdir('/content/drive/My Drive/CV_assignment')
```

Load the "images.npy" file (2 marks)

- This file contains images with details of bounding boxes

```
import numpy as np
data = np.load(file='images.npy',allow_pickle=True)
```

Check one sample from the loaded "images.npy" file (2 marks)

Hint - print data[10][1]

```
data[10][1]
```

```
[{'imageHeight': 337,
  'imageWidth': 600,
  'label': ['Face'],
  'notes': '',
  'points': [{'x': 0.48, 'y': 0.10385756676557864},
   {'x': 0.7716666666666666, 'y': 0.6795252225519288}]}]
```

Set image dimensions (1 mark)

- Initialize image height, image width with value: 224

```
IMAGE_WIDTH = 224
IMAGE_HEIGHT = 224
```

Create features and labels

- Here feature is the image
- The label is the mask
- Images will be stored in "X_train" array
- Masks will be stored in "masks" array

```
import cv2
from tensorflow.keras.applications.mobilenet import preprocess_input

masks = np.zeros((int(data.shape[0]), IMAGE_HEIGHT, IMAGE_WIDTH))
X_train = np.zeros((int(data.shape[0]), IMAGE_HEIGHT, IMAGE_WIDTH, 3))
for index in range(data.shape[0]):
    img = data[index][0]
    img = cv2.resize(img, dsize=(IMAGE_HEIGHT, IMAGE_WIDTH), interpolation=cv2.INTER_CUBIC)
    try:
      img = img[:, :, :3]
    except:
      continue
    X_train[index] = preprocess_input(np.array(img, dtype=np.float32))
    for i in data[index][1]:
        x1 = int(i["points"][0]['x'] * IMAGE_WIDTH)
        x2 = int(i["points"][1]['x'] * IMAGE_WIDTH)
        y1 = int(i["points"][0]['y'] * IMAGE_HEIGHT)
        y2 = int(i["points"][1]['y'] * IMAGE_HEIGHT)
        masks[index][y1:y2, x1:x2] = 1
```

Print the shape of X_train and mask array (1 mark)

```
X_train.shape
```

```
masks.shape
```

Print a sample image and image array

```
from matplotlib import pyplot
n = 10
print(X_train[n])
pyplot.imshow(X_train[n])
```

```
pyplot.imshow(masks[n])
```

Create the model (10 marks)

- Add MobileNet as model with below parameter values
    - input_shape: IMAGE_HEIGHT, IMAGE_WIDTH, 3
    - include_top: False

- alpha: 1.0
- weights: "imagenet"

- Add UNET architecture layers
  - This is the trickiest part of the project, you need to research and implement it correctly

```
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.layers import Conv2D, Reshape,MaxPool2D,Conv2DTranspose,concatenate
from tensorflow.keras.models import Model
from tensorflow.keras import Input


def create_model(trainable=True):
    model = MobileNet(include_top=False,input_shape=(IMAGE_HEIGHT,IMAGE_WIDTH,3),alpha=1.0,we
    for layer in model.layers:
        layer.trainable = trainable

    # Add all the UNET layers here
    inputs = Input(shape=(IMAGE_HEIGHT,IMAGE_WIDTH,3))
    x=model(inputs=inputs)
    x=Conv2D(filters=16,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    c1=Conv2D(filters=16,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    p1=MaxPool2D(pool_size=2,strides=2,padding='same')(c1)
    x=Conv2D(filters=32,kernel_size=3,strides=1,padding='same',activation='relu')(p1)
    c2=Conv2D(filters=32,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    p2=MaxPool2D(pool_size=2,strides=2,padding='same')(c2)
    x=Conv2D(filters=64,kernel_size=3,strides=1,padding='same',activation='relu')(p2)
    c3=Conv2D(filters=64,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    p3=MaxPool2D(pool_size=2,strides=2,padding='same')(c3)
    x=Conv2D(filters=128,kernel_size=3,strides=1,padding='same',activation='relu')(p3)
    c4=Conv2D(filters=128,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    u5=Conv2DTranspose(filters=64,kernel_size=(3,3),strides=2,padding='same')(c4)
    u5=concatenate([u5,c3])
    x=Conv2D(filters=64,kernel_size=3,strides=1,padding='same',activation='relu')(u5)
    c5=Conv2D(filters=64,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    u6=Conv2DTranspose(filters=32,kernel_size=(3,3),strides=2,padding='same')(c5)
    u6=concatenate([u6,c2])
    x=Conv2D(filters=32,kernel_size=3,strides=1,padding='same',activation='relu')(u6)
    c6=Conv2D(filters=32,kernel_size=3,strides=1,padding='same',activation='relu')(x)
    u7=Conv2DTranspose(filters=16,kernel_size=(3,3),strides=56,padding='same')(c6)
    output=Conv2D(filters=1,kernel_size=1,strides=1,padding='same',activation='relu')(u7)
    final_model = Model(inputs=inputs,outputs=output)

    return final_model
```

Call the create_model function

```
# Give trainable=False as argument, if you want to freeze lower layers for fast training (but
from tensorflow.keras.backend import clear_session
```

```
from tensorflow.keras.backend import clear_session
clear_session()
model = create_model(trainable=False)

# Print summary
model.summary()
```

Define dice coefficient function (5 marks)

- Create a function to calculate dice coefficient

```
from tensorflow.keras.backend import sum
from tensorflow.keras.layers import dot
def dice_coefficient(y_true,y_pred):
  intersction=sum(y_true*y_pred,axis=[1,2])
  union = sum(y_true,axis=[1,2]) + sum(y_pred,axis=[1,2])
  return ((2*intersction)/union)
```

Define loss

```
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.backend import log, epsilon
def loss(y_true,y_pred):
    return binary_crossentropy(y_true,
                               y_pred) - log(dice_coefficient(y_true, y_pred) + epsilon())
```

Compile the model (2 marks)

- Complie the model using below parameters
    - loss: use the loss function defined above
    - optimizers: use Adam optimizer
    - metrics: use dice_coefficient function defined above

```
model.compile(loss=loss,optimizer='Adam',metrics=[dice_coefficient])
```

Define checkpoint and earlystopping

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
checkpoint = ModelCheckpoint("model-{loss:.2f}.h5", monitor="loss", verbose=1, save_best_only
                             save_weights_only=True, mode="min", period=1)
stop = EarlyStopping(monitor="loss", patience=5, mode="min")
reduce_lr = ReduceLROnPlateau(monitor="loss", factor=0.2, patience=5, min_lr=1e-6, verbose=1,
```

Fit the model (2 marks)

- Fit the model using below parameters
  - epochs: you can decide
  - batch_size: 1
  - callbacks: checkpoint, reduce_lr, stop

```
model.fit(x=X_train,y=masks,batch_size=1,epochs=50,callbacks=[checkpoint,stop,reduce_lr])
```

Get the predicted mask for a sample image (3 marks)

```
n = 10
sample_image = X_train[n]

predicted_mask=model.predict(sample_image.reshape(1,sample_image.shape[0],sample_image.shape[
predicted_mask=predicted_mask.reshape(224,224)
```

Impose the mask on the image (3 marks)

```
sample_image[:,:,0] = sample_image[:,:,0]*predicted_mask
sample_image[:,:,1] = sample_image[:,:,1]*predicted_mask
sample_image[:,:,2] = sample_image[:,:,2]*predicted_mask
```