DEPARTMENT OF ELECTRICAL ENGINEERING

EE267 COMPUTER VISION WITH ARTIFICIAL INTELLIGENCE APPLICATIONS

TERM PROJECT REPORT

Title: **Unmasking Face Masks using Generative Adversarial Networks**

Name: Ramkumar Sivakumar
MSAI

STUDENT ID: 015443727

TERM PROJECT REPORT

TITLE OF PROJECT:

UNMASKING FACE MASKS USING GENERATIVE ADVERSARIAL NETWORKS

ABSTRACT:

The pandemic Covid-19 has been spreading all over the world ever since the beginning of 2020 and has caused people to use masks to cover their noses and mouth. To identify someone with a mask, our brain tries to imagine how they would look without the mask and then guesses who the person could be. Assault and threat have increased significantly with the rise in usage of masks. This project aims to build Generative Adversarial Networks (GANs) to unmask faces with masks similar to the human brain to identify people. The models when deployed could help law enforcement to visualize the faces of people wearing masks.

DETAILED DESCRIPTION OF THE PROJECT:

INTRODUCTION:

The SARS-CoV-2 pandemic is here to stay for a while as the virus is constantly changing through mutations like Alpha, Beta, Gamma, Etc among which the Delta and Omicron variants were reported deadly. Over the past, wearing masks has helped control the spread and it is important to keep our noses and mouth covered until we get past the pandemic. However, wearing masks has also increased the crime rates. It has become easier for perpetrators to commit crimes wearing masks. Traditionally law enforcement uses face composite software to visualize or sketch the faces of the perpetrators. Generative Adversarial Networks, when trained, will efficiently synthesize facial features of masked faces.

OBJECTIVES:

The main objective of this project is to build GANs to synthesize facial features of the masked region on a face. The object will be achieved through the following steps:

- Obtain image dataset of faces without masks: Dataset is obtained from Kaggle for building the model.
- Generate faces with different kinds of masks using [3].
- Pre-process images: Resize images to 128*128. Split dataset into train and test sets. The test dataset will be used to check the model's performance in synthesizing faces.
- Build Generative Adversarial Network with downsampling and upsampling layers using TensorFlow and Keras.
- Training: Train GAN models on Google Collaboratory using different hyper-parameters to get better results.
- Unmask faces using the trained models and evaluate their performance.

TOOLS REQUIRED:

- Google Collaboratory
- Python
- TensorFlow
- OpenCV
- Keras
- Numpy

PRE-PROCESSING DATA:

The dataset was obtained from Kaggle. The dataset consists of 10,000 high-resolution images of faces with and without masks and is of size 25GB.

Figure 1: Without Mask

Figure 2: With Mask

The dataset was preprocessed on the Local machine using Jupyter Notebook and OpenCV 2. The images were reduced to the size of 128*128 considering the GPU resources available. The dataset by default had images of faces with only a single type of mask. To make the model robust to masks of different kinds, [3] was used to generate faces with different masks.

Figure 3: Sample Masks

About 1000 faces with 4 different masks, that is 4000 images were used as the training set. About 400 faces with random masks, that is 400 images were used as the test set.

For the first model, RGB2RGB: The input set had images with masks and the target set had faces without masks.

Figure 4: RGB2RGB Model Train set

For the second model, RGB2SKETCH: The target images were preprocessed differently to obtain a sketch of the faces. The following image processing techniques were applied:

A. Kernel Sharpening
B. Obtaining grayscale image
C. Inversion of the sharpened image
D. Subtracting the grayscale image and applying division
E. Applying Gaussian Blur

Figure 5: RGB2SKETCH Model Train set

The preprocess train and test set were stored as Numpy files and were uploaded to google drive for training the models on Google Collaboratory.

PIX2PIX GENERATIVE ADVERSARIAL NETWORKS:

GANs are a set of layers similar to neural networks that are used for image-to-image translation applications. It is a type of conditional GAN where the generation of the output image is conditional on the input. The cGAN when provided with both the input and target image, learns how to transform the input image into the target image during training. A GAN typically has a generator and a discriminator. The generator tries to generate images close enough to the target images to fool the discriminator. The discriminator compares the generated image to the target image.

Pix2Pix GANs follow a UNET architecture. The UNET architecture has downsampling and upsampling layers that are interconnected with each other. The following diagram shows how the layers are interconnected with each other.

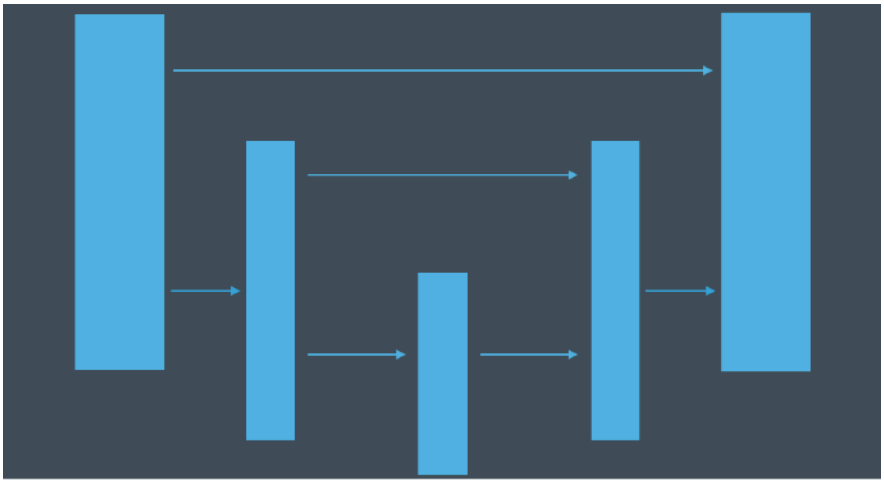


Figure 6: GAN UNET Architecture

The blue blocks are downsampled and upsampled convolution layers. Blocks of the same size have the same dimensions. The weights from the downsampling layers are added to the weights of the corresponding upsampling layer directly. Through this function, the model learns the similarities and dissimilarities between the input and the target image.

MODEL ARCHITECTURE:

The model was built using the TensorFlow and Keras library. The model has 25 layers in total including the input layer. The hidden layers consist of the following:

- Convolution Layers: 15
- Convolution Transpose Layers: 3
- MaxPooling Layers: 3
- Addition Layers: 3
- Input Layer: 1

- The model follows a UNET architecture.

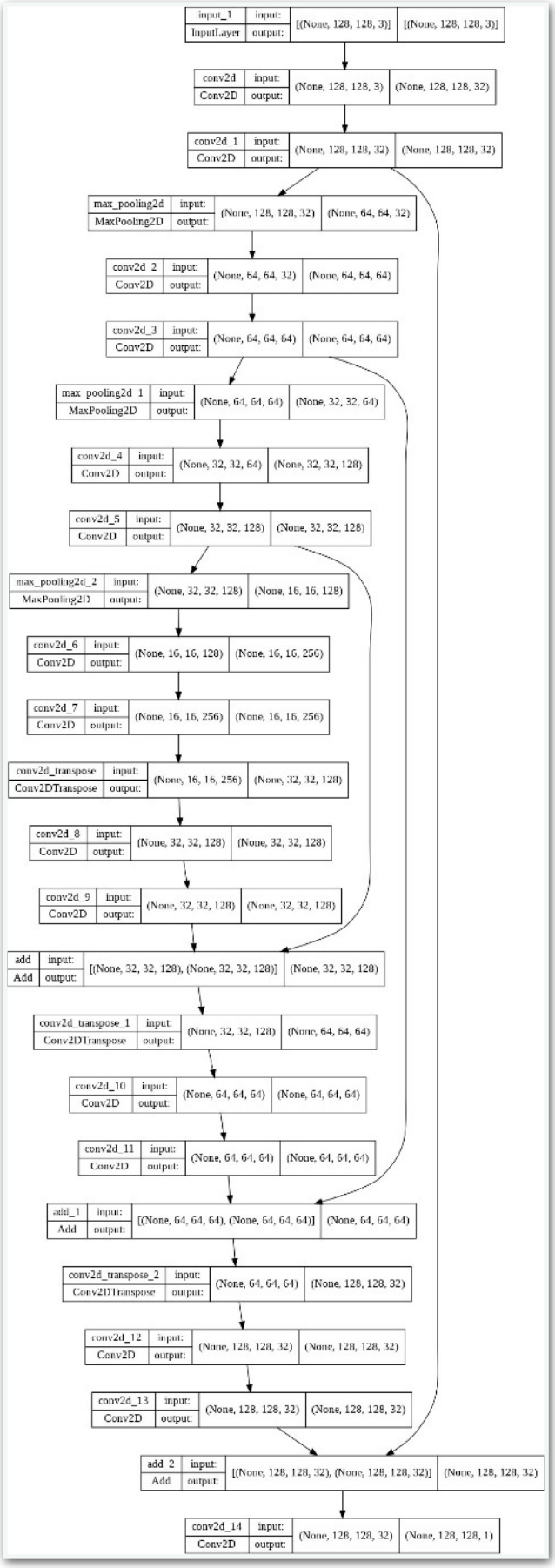


Figure 7: Model Architecture

- The convolution layer convolutes the image over a fixed kernel size and applies the ReLU function over the input with its weights.
- The maxpooling layers applied the maxpool function with a fixed kernel for dimensionality reduction.
- Convolution Transpose layers are used for upsampling the output from the previous convolution layers.
- Addition layers act as a bypassing layer to add the weights of downsampling layers to the corresponding upsampling layers.

The RGB2SKETCH model has an output dimension [128,128,1] since the output is a grayscale image and has only a single channel. On the other hand, the RGB2RGB model has an output dimension [128,128,3] since the color image output has three channels.

DATA AUGMENTATION:

Data augmentation is a technique used to increase the size of the training set. This operation helps in making the model robust to any variations in attributes such as angle, tilt and closeness of the object in the input image during prediction. Functions such as zoom, flip, and tilt are used for augmenting the input set.

A data generator is built using the Keras Sequence class. The function of a data generator is to apply data augmentation in batches and feed the augmented set with the input and target images to the model for training.

MODEL TRAINING:

During the training step, the corresponding input and target images are fed into the model in batches. The model learns the similarities and differences between the input and target images to generate an image similar to the target image. The difference between the target image and the generated image is estimated as training loss. The goal of the training phase is to reduce the training loss and increase the training accuracy.

- RGB2SKETCH model was trained for 100 epochs whereas RGB2RGB model was only trained for 50 epochs due to limitation in GPU resources.
- Adam optimizer with a learning rate of 0.002 was used during the compilation of the model with binary cross-entropy loss.
- ReduceLROnPlateau was initialized with a minimum learning rate of 0.0001 and a factor of 0.2. This function helps in reducing the learning rate when there is no decrease in loss during training.
- EarlyStopping function is used to stop the training midway when there is no improvement in loss during training even after reducing the learning rate on a plateau.
- ModelCheckpoint function is used to save the model with the lowest validation loss during training.

MODEL EVALUATION:

RGB2RGB:

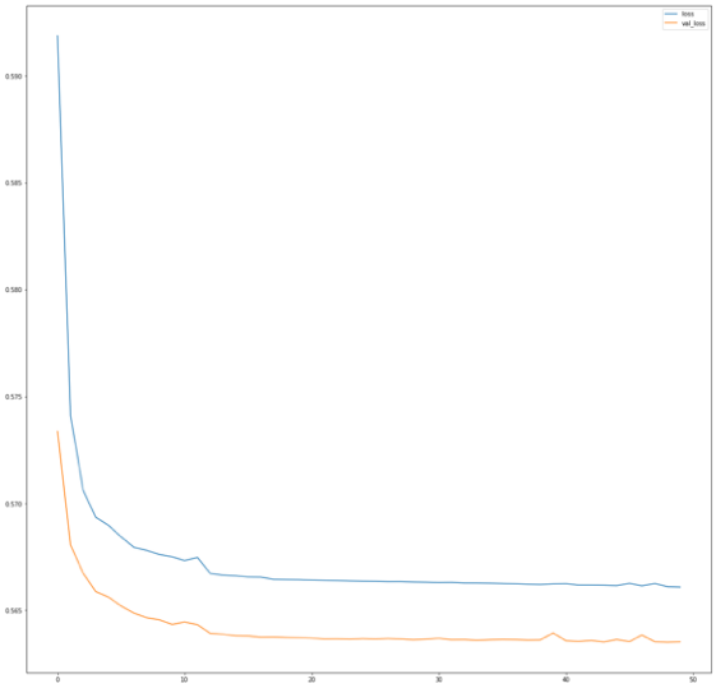


Figure 8: RGB2RGB Loss

After 50 epochs of training, the training loss obtained for the model is 0.566 and the validation loss obtained is 0.5635.

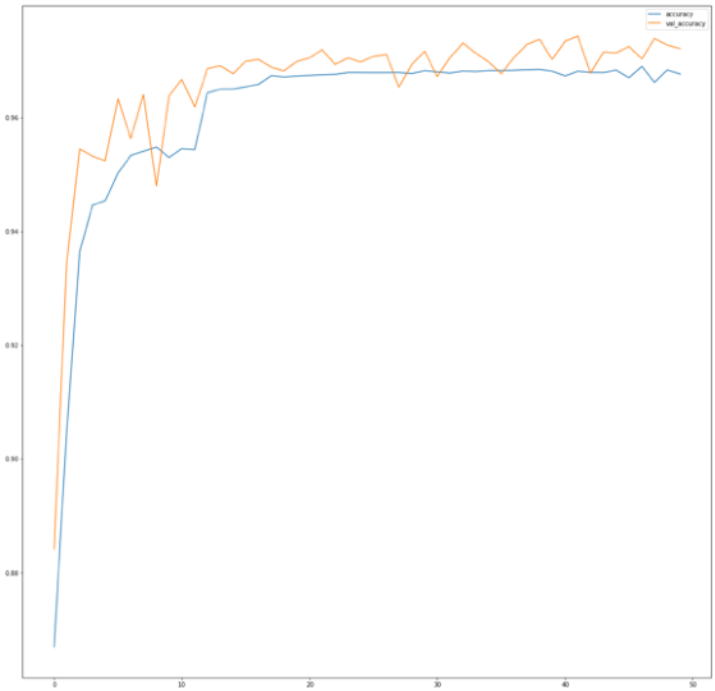


Figure 9: RGB2RGB Accuracy

After 50 epochs of training, the training accuracy obtained for the model is 0.9676 and the validation loss obtained is 0.9721.

RGB2SKETCH:

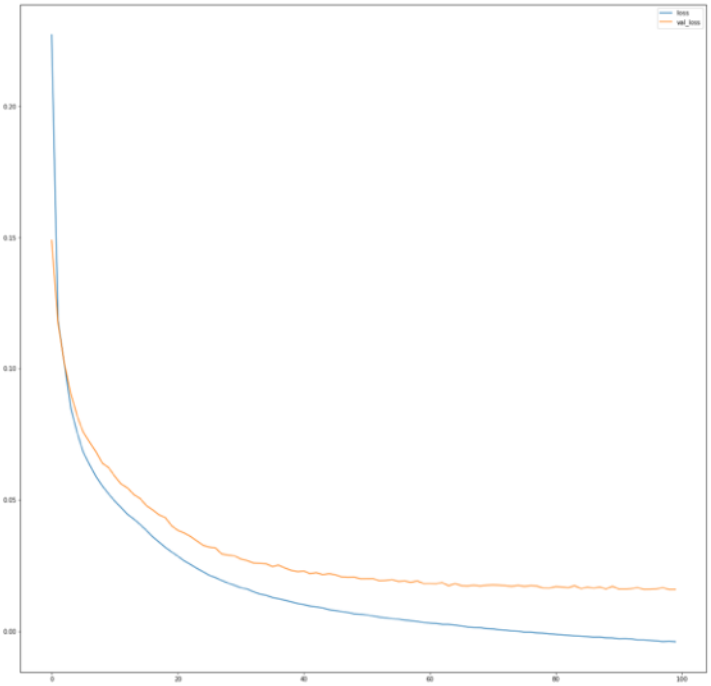


Figure 10: RGB2SKETCH Loss

After 100 epochs of training, the training loss obtained for the model is -0.001 and the validation loss obtained is 0.0025.

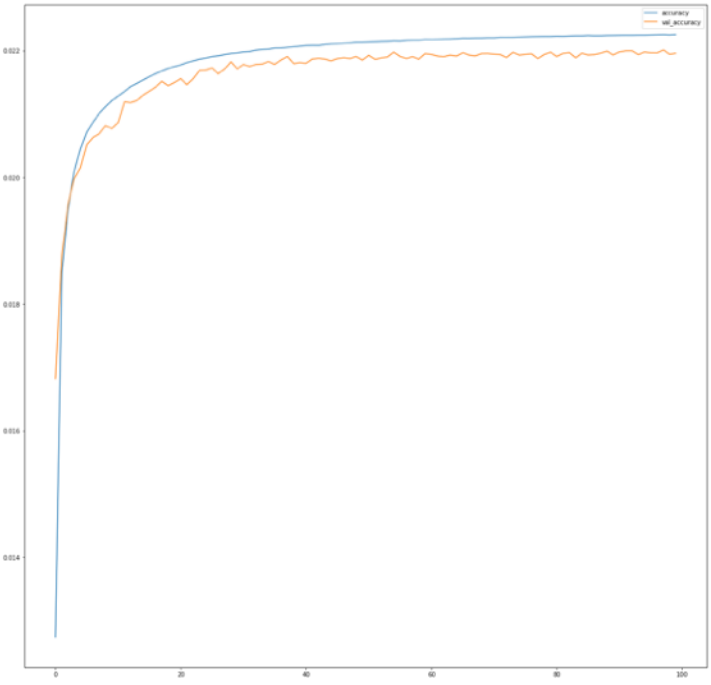


Figure 11: RGB2SKETCH Loss

After 100 epochs of training, the training accuracy obtained for the model is 0.022 and the validation accuracy obtained is 0.021.

The accuracy is low because the model outputs a sketch image. Sketch images tend to have shades of black pixel values. Even if the shades on the images generated by the network are slightly darker or lighter, the accuracy significantly goes down.

SIMULATION/VERIFICATION:

RGB2RGB MODEL:

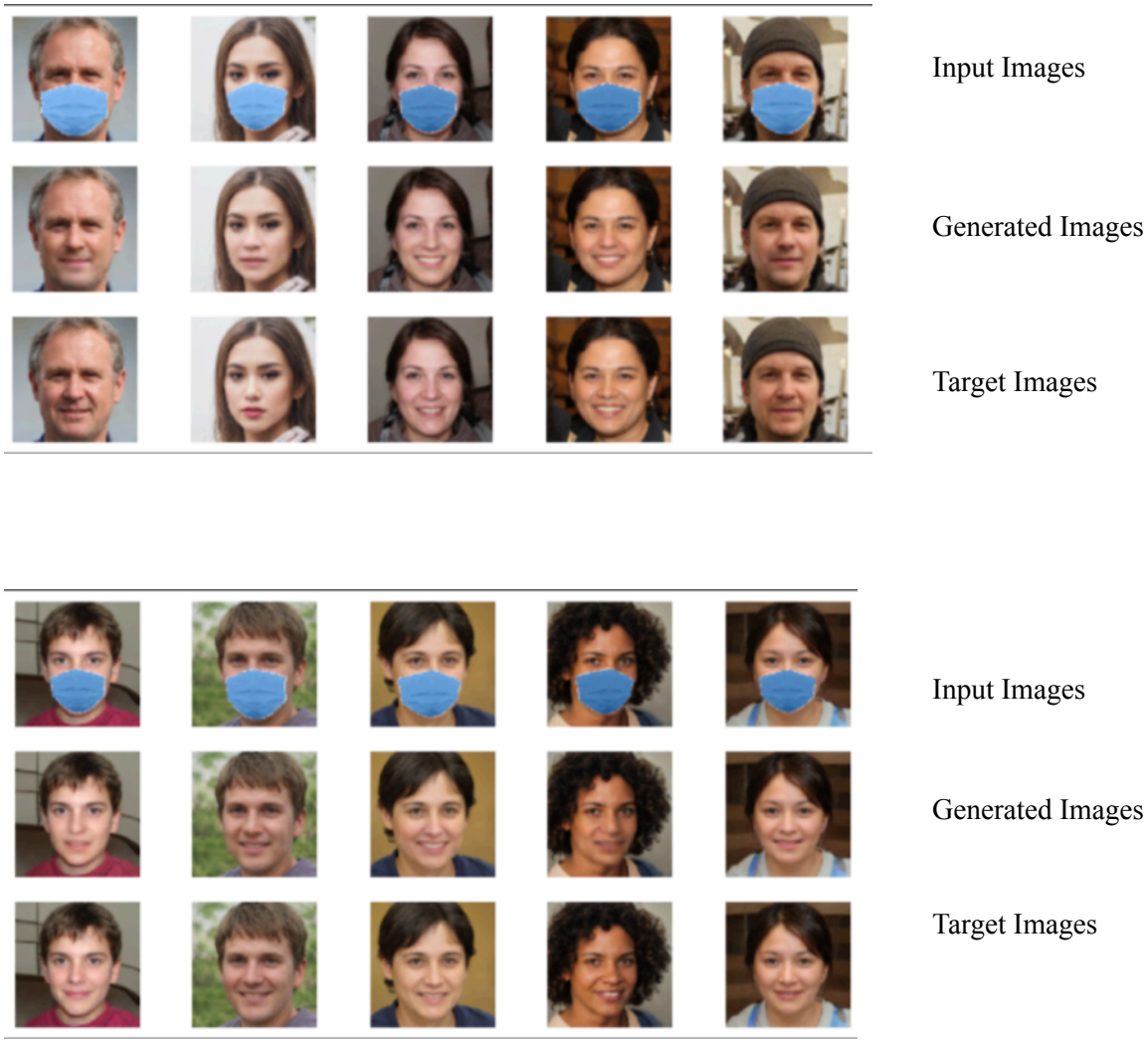


Figure 12: RGB2RGB TEST SET

The faces in the generated images are similar to the faces on the target images. Even though some differences in the facial features such as nose and mouth are observed between the target and generated faces, the target faces can be recognized with the faces generated by the model.

RGB2SKETCH MODEL:

Figure 13: RGB2RGB TEST SET

The faces generated by the RGB2SKETCH model are very similar to the target images. Some of the noticeable differences between the generated and target images are:

- The faces generated by the model are more clean/smooth.
- Beard seems to be missing on the generated faces.
- Dimples on the faces seem to be missing too.

However, the target faces can be recognized with the faces generated by the model.

CONCLUSION:

Thus, RGB2RGB and RGB2SKETCH models can be used to unmask masked faces. The minimum validation loss of the models is 0.5635 and 0.0025. RGB2SKETCH model is more accurate as it eliminates the color component. In the case of obtaining real-like images, the RGB2RGB model is preferred. These models will help in visualizing faces underneath the masked faces. The models when trained with a larger dataset with different facial features will help law enforcement to visualize the faces of the perpetrators. These models can also be used in parallel with face composite software in law enforcement. The model can also be expanded to generate full faces from partial faces.

FUTURE WORKS:

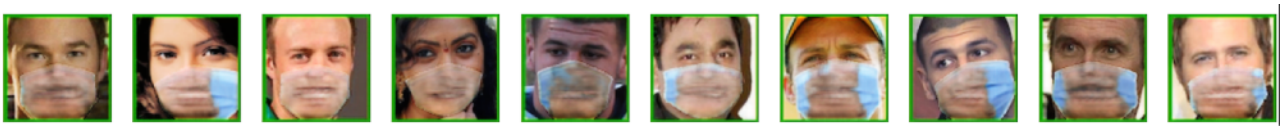
- Collect more training samples with different facial features such as skin color, nose and mouth shapes.
- Change pixel values to 0 in the masked region on the face before training. This will make the model robust to different types of facial masks.
- Increase model complexity for better performance.

REFERENCES:

[1] Prasoon Kottarathil. Face Mask Lite Dataset. https://www.kaggle.com/prasoonkottarathil/facemask-lite-dataset
[2] https://machinelearningmastery.com/how-to-implement-pix2pix-gan-models-from-scratch-with-keras/
[3] https://github.com/aqeelanwar/MaskTheFace
[4] A U-Net Based Discriminator for Generative Adversarial Networks https://doi.org/10.48550/arXiv.2002.12655
[5] https://www.tensorflow.org/

APPENDIX:

OUTPUT ON RANDOM FACES:



CODE CHUNKS: (Code was executed in Python Notebooks)

IMPORT LIBRARIES:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.utils import Sequence
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, Conv2DTranspose, Dropout, Flatten, Reshape, Dense, add
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split
import cv2
import random, math
```

IMAGE PREPROCESSING:

1. Resizing images to 128*128:

```
import glob
import cv2
from tqdm import tqdm
import numpy as np

path = glob.glob("/Volumes/Seagate Backup Plus Drive/CV/archive/without_mask/*")
path.sort()
```

```
j=0
for i in tqdm(path):
    img = cv2.imread(i, cv2.IMREAD_UNCHANGED)
    width = int(200)
    height = int(200)
    dim = (width, height)
    # resize image
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    cv2.imwrite(str(j)+".png",resized)
    j+=1
```

2. Generating Masks using MaskTheFace [3].

```
!git clone https://github com/ramsrk7/MaskTheFace git

!pip install dotmap
import dotmap

!python /content/MaskTheFace/mask_the_face py --path '/content/drive/MyDrive/CV/Train' --mask_type
<mask type> --write_original_image
!python /content/MaskTheFace/mask_the_face py --path '/content/drive/MyDrive/CV/Test' --mask_type
<mask_type> —write_original_image
```

3. Preparing dataset for RGB2RGB Model:

```
X = []
Y = []
test = []
for i in zip(masks_black, nomasks):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = load_img(i[1], target_size=(128,128))
    Y.append(img_to_array(img)/250.)

for i in zip(masks_blue, nomasks):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = load_img(i[1], target_size=(128,128))
    Y.append(img_to_array(img)/250.)

for i in zip(masks_cloth, nomasks):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = load_img(i[1], target_size=(128,128))
    Y.append(img_to_array(img)/250.)

for i in zip(masks_kn95, nomasks):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = load_img(i[1], target_size=(128,128))
    Y.append(img_to_array(img)/250.)

X = np.array(X)
Y = np.array(Y)
```

```
X_test = []
Y_test = []

for i in tqdm(zip(test_X_path_surgical, test_Y_path)):
    img = load_img(i[0], target_size=(128,128))
    X_test.append(img_to_array(img)/250.)
    img = load_img(i[1], target_size=(128,128))
    Y_test.append(img_to_array(img)/250.)

for i in zip(test_X_path_KN95, test_Y_path):
    img = load_img(i[0], target_size=(128,128))
```

```
    X_test.append(img_to_array(img)/250.)
    img = load_img(i[1], target_size=(128,128))
    Y_test.append(img_to_array(img)/250.)

X_test = np.array(X_test)
Y_test = np.array(Y_test)

X.shape, Y.shape, X_test.shape, Y_test.shape
```

4. Preparing dataset for RGB2SKETCH Model:

```
def sketch(img):
    img = cv2.imread(img)
    scale = 0.60
    width = int(128)
    height = int(128)
    dim = (width,height)
    resized = cv2.resize(img,dim,interpolation = cv2.INTER_AREA)
    kernel_sharpening = np.array([[0,-1,0], [-1, 5.2,-1], [0,-1,0]])
    sharpened = cv2.filter2D(resized,-1,kernel_sharpening)
    gray = cv2.cvtColor(sharpened, cv2.COLOR_BGR2GRAY)
    inv = 255-gray
    gauss = cv2.GaussianBlur(inv,ksize=(15,15),sigmaX=0,sigmaY=0)
    def sketch(image,mask):
        return cv2.divide(image,255-mask,scale=256)

    pencil_img = sketch(gray,gauss)

    return pencil_img
```

```
X = []
Y = []
test = []
for i in tqdm(zip(masks_black, nomasks)):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = sketch(i[1])
    Y.append(img_to_array(img)/250.)

for i in tqdm(zip(masks_blue, nomasks)):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = sketch(i[1])
    Y.append(img_to_array(img)/250.)

for i in tqdm(zip(masks_cloth, nomasks)):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = sketch(i[1])
    Y.append(img_to_array(img)/250.)

for i in tqdm(zip(masks_kn95, nomasks)):
    img = load_img(i[0], target_size=(128,128))
    X.append(img_to_array(img)/250.)
    img = sketch(i[1])
    Y.append(img_to_array(img)/250.)

X = np.array(X)
```

```
Y = np.array(Y)
```

```
X_test = []
Y_test = []

for i in tqdm(zip(test_X_path_surgical, test_Y_path)):
    img = load_img(i[0], target_size=(128,128))
    X_test.append(img_to_array(img)/250.)
    img = sketch(i[1])
    Y_test.append(img_to_array(img)/250.)

for i in zip(test_X_path_KN95, test_Y_path):
    img = load_img(i[0], target_size=(128,128))
    X_test.append(img_to_array(img)/250.)
    img = sketch(i[1])
    Y_test.append(img_to_array(img)/250.)

X_test = np.array(X_test)
Y_test = np.array(Y_test)

X.shape, Y.shape, X_test.shape, Y_test.shape
```

```
np.save("X_rgb_128_train.npy",X)
np.save("Y_sketch_128_train.npy",Y)
np.save("X_rgb_128_test.npy",X_test)
np.save("Y_sketch_128_test.npy",Y_test)
```

DATA AUGMENTATION:

```
class Daugmentation:
    def flip(img, t):
        if t[0]==0:
            return img
        else:
            return cv2.flip(img, t[1])

    def zoom(img, t):
        if t[2]==0:
            return img
        else:
            h, w = img.shape[:2]
            nh, nw =  int(t[3]*h), int(t[3]*w)
            dh, dw = h-nh, w-nw
            zimg = img[dh//2:nh+dh//2, dw//2:nw+dw//2]
            zimg = cv2.resize(zimg, (w,h))
            return zimg


    def get_ts(batch_size):
            return [[random.choice([0,1]),random.choice([-1,0,1]), random.randint(0,2),random.uniform(0.4,0.9)]
for i in range(batch_size)]

    def aug(img,t):
        img = Daugmentation.flip(img,t)
        # img = Daugmentation.zoom(img, t)
```

```
        return img


class maSequence(Sequence):
    def __init__(self, x_set, y_set, batch_size, Daugmentation):
        self.x, self.y = x_set, y_set
        self.batch_size = batch_size
        self.augment = Daugmentation

    def __len__(self):
        return int(np.ceil(len(self.x) / float(self.batch_size)))

    def __getitem__(self, idx):
        batch_x = self.x[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]
        ts = self.augment.get_ts(len(batch_x))
          return np.array([self.augment.aug(x,t) for x,t in zip(batch_x,ts)]),np.array([self.augment.aug(y,t) for y,t
in zip(batch_y,ts)])
```

```
batch_size=16

gotrain = maSequence(x_train, y_train, batch_size,Daugmentation)
gotest = maSequence(x_test, y_test, batch_size,Daugmentation)
```

RGB2RGB MODEL BUILDING AND TRAINING:

```
def get_model():
    # encoder
    In = Input(shape=x_train[0].shape)
    c1 = Conv2D(32, 3, activation="relu", padding="same")(In)
    c1 = Conv2D(32, 3, activation="relu", padding="same")(c1)

    m1 = MaxPooling2D(2)(c1)
    c2 = Conv2D(64, 3, activation="relu", padding="same")(m1)
    c2 = Conv2D(64, 3, activation="relu", padding="same")(c2)

    m2 = MaxPooling2D(2)(c2)
    c3 = Conv2D(128, 3, activation="relu", padding="same")(m2)
    c3 = Conv2D(128, 3, activation="relu", padding="same")(c3)

    m3 = MaxPooling2D(2)(c3)
    c4 = Conv2D(256, 3, activation="relu", padding="same")(m3)
    c4 = Conv2D(256, 3, activation="relu", padding="same")(c4)

    u1 = Conv2DTranspose(128, 3, activation="relu", strides=2, padding="same")(c4)
    c5 = Conv2D(128, 3, activation="relu", padding="same")(u1)
    c5 = Conv2D(128, 3, activation="relu", padding="same")(c5)

    a1 = add([c5,c3])
    u2 = Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(a1)
    c6 = Conv2D(64, 3, activation="relu", padding="same")(u2)
    c6 = Conv2D(64, 3, activation="relu", padding="same")(c6)

    a2 = add([c6,c2])
    u3 = Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(a2)
    c7 = Conv2D(32, 3, activation="relu", padding="same")(u3)
```

```
    c7 = Conv2D(32, 3, activation="relu", padding="same")(c7)

    a3 = add([c7,c1])
    Out = Conv2D(3, 3, activation="sigmoid", padding="same")(a3)

    model = Model(In,Out)
    adam = keras.optimizers.Adam(learning_rate=0.002, beta_1=0.9, beta_2=0.999, amsgrad=False)
    model.compile(optimizer=adam,loss="binary_crossentropy",metrics=['accuracy'])
    return model

model = get_model()
model.summary()
```

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                          patience=2, min_lr=0.0001)
checkpointer = ModelCheckpoint(filepath='/content/drive/MyDrive/CV/best_rgb_final_1.h5', verbose=0,
monitor='val_loss',
    mode='min', save_best_only=True)

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

with              '/device:GPU:0'

  history = model.fit_generator generator=gotrain
                          steps_per_epoch=gotrain.__len__()
                          epochs=50

                          callbacks= checkpointer reduce_lr callback
                          validation_data=gotest
                          validation_steps=gotest.__len__())
```

RGB2SKETCH MODEL BUILDING AND TRAINING:

```
def get_model():
    # encoder
    In = Input(shape=x_train[0].shape)
    c1 = Conv2D(32, 3, activation="relu", padding="same")(In)
    c1 = Conv2D(32, 3, activation="relu", padding="same")(c1)

    m1 = MaxPooling2D(2)(c1)
    c2 = Conv2D(64, 3, activation="relu", padding="same")(m1)
    c2 = Conv2D(64, 3, activation="relu", padding="same")(c2)

    m2 = MaxPooling2D(2)(c2)
    c3 = Conv2D(128, 3, activation="relu", padding="same")(m2)
    c3 = Conv2D(128, 3, activation="relu", padding="same")(c3)

    m3 = MaxPooling2D(2)(c3)
    c4 = Conv2D(256, 3, activation="relu", padding="same")(m3)
    c4 = Conv2D(256, 3, activation="relu", padding="same")(c4)

    u1 = Conv2DTranspose(128, 3, activation="relu", strides=2, padding="same")(c4)
    c5 = Conv2D(128, 3, activation="relu", padding="same")(u1)
    c5 = Conv2D(128, 3, activation="relu", padding="same")(c5)

    a1 = add([c5,c3])
    u2 = Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(a1)
```

```
    c6 = Conv2D(64, 3, activation="relu", padding="same")(u2)
    c6 = Conv2D(64, 3, activation="relu", padding="same")(c6)

    a2 = add([c6,c2])
    u3 = Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(a2)
    c7 = Conv2D(32, 3, activation="relu", padding="same")(u3)
    c7 = Conv2D(32, 3, activation="relu", padding="same")(c7)

    a3 = add([c7,c1])
    Out = Conv2D(3, 1, activation="sigmoid", padding="same")(a3)

    model = Model(In,Out)
    adam = keras.optimizers.Adam(learning_rate=0.002, beta_1=0.9, beta_2=0.999, amsgrad=False)
    model.compile(optimizer=adam,loss="binary_crossentropy",metrics=['accuracy'])
    return model

model = get_model()
model.summary()
```

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                          patience=2, min_lr=0.0001)
checkpointer = ModelCheckpoint(filepath='/content/drive/MyDrive/CV/best_rgb_final_1.h5', verbose=0,
monitor='val_loss',
    mode='min', save_best_only=True)

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

with tf.device('/device:GPU:0'):

  history = model.fit_generator(generator=gotrain,
                          steps_per_epoch=gotrain.__len__(),
                          epochs=100,

                          callbacks=[checkpointer,reduce_lr,callback],
                          validation_data=gotest,
                          validation_steps=gotest.__len__())
```