[ ]:
```
import zipfile
from google.colab import drive

drive.mount('/content/drive/')

#poc_DATASET
#zip_ref = zipfile.ZipFile("/content/drive/My Drive/sg_ff_filtered_red.zip", 'r')

#ISGI_20000_200gray_DATASET
# zip_ref = zipfile.ZipFile("/content/drive/My Drive/ISGI_dataset_200g.zip", 'r')

#ISGI_20000_200rgb_DATASET
zip_ref = zipfile.ZipFile("/content/drive/My Drive/ISGI_dataset_200rgb.zip", 'r')

zip_ref.extractall("/tmp/")
zip_ref.close()
```

[ ]:
```
import os

base_dir = '/tmp/ISGI_dataset_200rgb'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'valid') #valid or validations

# Directory with our training FlickerFaces pictures
```

```python
train_ff_dir = os.path.join(train_dir, 'ff')

# Directory with our training StyleGAN pictures
train_sg_dir = os.path.join(train_dir, 'sg')

# Directory with our validation FlickerFaces pictures
validation_ff_dir = os.path.join(validation_dir, 'ff')

# Directory with our validation StyleGAN pictures
validation_sg_dir = os.path.join(validation_dir, 'sg')
```

[ ]:
```python
train_ff_fnames = os.listdir(train_ff_dir)
train_ff_fnames.sort()
print(train_ff_fnames[:10])

train_sg_fnames = os.listdir(train_sg_dir)
train_sg_fnames.sort()
print(train_sg_fnames[:10])
```

[ ]:
```python
print('Training_FlickerFaces images total: \t', len(os.listdir(train_ff_dir)))
print('Training_StyleGAN images total: \t', len(os.listdir(train_sg_dir)))
print('Validation_FlickerFaces images total: \t', len(os.
 ↪listdir(validation_ff_dir)))
print('Validation_StyleGAN images total: \t', len(os.listdir(validation_sg_dir)))
```

[ ]:
```python
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random

#params for graph
nrows = 4
ncols = 4

#index for iteration
pic_index = random.randint(0, 990)
```

[ ]:
```python
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)

pic_index += 8
next_ff_pix =  [os.path.join(train_ff_dir, fname)
                for fname in train_ff_fnames[pic_index-8:pic_index]]
next_sg_pix =  [os.path.join(train_sg_dir, fname)
                for fname in train_sg_fnames[pic_index-8:pic_index]]
```

```python
for i, img_path in enumerate(next_ff_pix+next_sg_pix):
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off')

  img = mpimg.imread(img_path)
  plt.imshow(img)

  plt.show
```

[ ]:
```python
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.layers import BatchNormalization, Dropout
```

[ ]:
```python
input_layer = layers.Input(shape=(200, 200, 3))

x = layers.Conv2D(16, 3, activation='relu')(input_layer)
x = layers.MaxPooling2D(2)(x)
x = Dropout(rate = 0.2)(x)

x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = Dropout(rate = 0.3)(x)

x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = Dropout(rate = 0.4)(x)

x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = Dropout(rate = 0.5)(x)

x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = Dropout(rate = 0.5)(x)

x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = Dropout(rate = 0.5)(x)

x = layers.Flatten()(x)

x = layers.Dense(200, activation='relu')(x)
x = Dropout(rate = 0.5)(x)

output_layer = layers.Dense(1, activation='sigmoid')(x)

model = Model(input_layer, output_layer)
```

```
model.summary()
```

[ ]:
```
input_layer = layers.Input(shape=(200, 200, 3))

x = layers.Conv2D(16, 3, activation='relu')(input_layer)
x = layers.MaxPooling2D(2)(x)

x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

x = layers.Flatten()(x)

x = layers.Dense(512, activation='relu')(x)

output_layer = layers.Dense(1, activation='sigmoid')(x)

model = Model(input_layer, output_layer)

model.summary()
```

[ ]:
```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(learning_rate=0.001),
              metrics=['acc'])
```

[ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   zoom_range = 0.2,
                                   horizontal_flip = True,
                                   vertical_flip = True)

val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(200, 200),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
```

```python
# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(200, 200),
        batch_size=20,
        class_mode='binary')
```

[ ]:
```python
history = model.fit(
      train_generator,
      steps_per_epoch=800,  # 2000 images = batch_size * steps
      epochs=15,
      validation_data=validation_generator,
      validation_steps=100,  # 1000 images = batch_size * steps
      verbose=2)
```

[ ]:
```python
scores = model.evaluate(validation_generator, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

[ ]:
```python
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]
visualization_model = Model(input_layer, successive_outputs)

# prepare a random input image of a FlickerFaces or StyleGAN from the training␣
↪set.
ff_img_files = [os.path.join(train_ff_dir, f) for f in train_ff_fnames]
sg_img_files = [os.path.join(train_sg_dir, f) for f in train_sg_fnames]
img_path = random.choice(ff_img_files + sg_img_files)

img = load_img(img_path, target_size=(200, 200))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255
x /= 255

#  run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers[1:]]
```

```python
# Now  display our representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  if len(feature_map.shape) == 4:
    # Just do this for the conv / maxpool layers, not the fully-connected layers
    n_features = feature_map.shape[-1]  # number of features in feature map
    # The feature map has shape (1, size, size, n_features)
    size = feature_map.shape[1]
    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))
    for i in range(n_features):
      # Postprocess the feature to make it visually palatable
      x = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std()
      x *= 64
      x += 128
      x = np.clip(x, 0, 255).astype('uint8')
      # We'll tile each filter into this big horizontal grid
      display_grid[:, i * size : (i + 1) * size] = x
    # Display the grid
    scale = 20. / n_features
    plt.figure(figsize=(scale * n_features, scale))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

[ ]:

```python
# Retrieve a list of accuracy results on training and validation data
# sets for each training epoch
acc = history.history['acc']
val_acc = history.history['val_acc']

# Retrieve a list of list results on training and validation data
# sets for each training epoch
loss = history.history['loss']
val_loss = history.history['val_loss']

# Get number of epochs
epochs = range(len(acc))

# Plot training and validation accuracy per epoch
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title('Training and validation accuracy')

plt.figure()
```

```
# Plot training and validation loss per epoch
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Training and validation loss')
```

[ ]:
```
model.save('placeholderm2.h5')
```

[ ]:
```
!zip -r /content/model_1 /content/model
```