[ ]:
```
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/sg_ff_filtered_red'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

```
# Directory with our training FlickerFaces pictures
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')
```

[ ]:
```
#imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.backend import clear_session
import tensorflow as tf
tf.test.gpu_device_name()
```

[ ]:
```
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)))
```

[ ]:
```
!pip install optuna
```

[ ]:
```
import optuna
```

[ ]:
```
dropout_rate = [0] * 2

def create_model(trial):

    num_layers = trial.suggest_int("num_layers", 1, 7)
    activation = trial.suggest_categorical("activation", ["relu"])
    dropout_rate[0] = trial.suggest_uniform('dropout_rate'+str(0), 0.0, 0.5)
    dropout_rate[1] = trial.suggest_uniform('dropout_rate'+str(1), 0.0, 0.5)
    mid_units = int(trial.suggest_discrete_uniform("mid_units", 100, 300, 100))
    filters=trial.suggest_categorical("filters", [16, 32, 64, 128])
    kernel_size=trial.suggest_categorical("kernel_size", [3, 3])
    strides=trial.suggest_categorical("strides", [1, 2])

    classifier = Sequential()

    #step 1  - Convolution Layers

    classifier.add(
```

```
        Conv2D(
            filters=filters,
            kernel_size=kernel_size,
            strides=1,
            activation = activation,
            input_shape=(200, 200, 3),
        )
    )

    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    for i in range(1, num_layers):
        classifier.add(
            Conv2D(
                filters=filters,
                kernel_size=kernel_size,
                strides=1,
                activation = activation,
            )
        )
    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    classifier.add(Dropout(dropout_rate[0]))
    classifier.add(Flatten())
    classifier.add(Dense(units = mid_units, activation = activation))
    classifier.add(Dropout(dropout_rate[1]))
    classifier.add(Dense(units = 1, activation ='sigmoid'))

    return classifier
```

[ ]:
```
#image augumentation
from keras.preprocessing.image import ImageDataGenerator

#Data Preparation

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

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory(train_dir,
                                                 target_size = (200, 200),
                                                 batch_size = 10,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory(validation_dir,
                                            target_size = (200, 200),
```

```
                                                batch_size = 10,
                                                class_mode = 'binary')
```

[ ]:
```
training_set
```

[ ]:
```
def objective(trial):

    optimizer = trial.suggest_categorical("optimizer", ["sgd", "adam",␣
 ↪"rmsprop", "adadelta", "adagrad", "adamax"])

    classifier = create_model(trial)

    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy',␣
 ↪metrics = ['accuracy'])

    history = classifier.fit(training_set,
                         steps_per_epoch = 100, # num_samples // batch_size
                         epochs = 5, # entire iteration over dataset
                         validation_data = test_set,
                         validation_steps = 50) #https://keras.io/api/models/
 ↪model_training_apis/

    classifier.save('/drive/MyDrive/Models/trialmodel_' + str(history.
 ↪history['val_accuracy'][-1]) +".h5")


    return history.history["val_accuracy"][-1]
```

[ ]:
```
import pickle

study = optuna.create_study(direction="maximize", )

#studypik = pickle.load(open('study.pickle', 'rb'))
study.optimize(objective, n_trials = 10, timeout = 60 * 60 * 3,␣
 ↪show_progress_bar=True)
print(studypik.best_params)
print(studypik.best_value)
pickle.dump(studypik, open('study.pickle', 'wb'))
```

[ ]:
```
study = optuna.create_study(direction="maximize", )
study.optimize(objective, n_trials = 10, timeout = 60 * 60 * 3,␣
 ↪show_progress_bar=True)
print(study.best_params)
print(study.best_value)
```

[ ]:
```
print(study.best_params)
print(study.best_value)
```

[ ]:
```
fig = optuna.visualization.plot_optimization_history(study)
fig.show()
```

[ ]:
```
fig = optuna.visualization.plot_param_importances(study)
fig.show()
```

[ ]:
```
print(studypik.best_params)
print(studypik.best_value)
pickle.dump(studypik, open('study.pickle', 'wb'))
```

[ ]:
```
print("Number of finished trials: {}".format(len(study.trials)))

print("Best trial:")
trial = study.best_trial

print("  Value: {}".format(trial.value))

print("  Params: ")
for key, value in trial.params.items():
  print("    {}: {}".format(key, value))
```

[ ]:
```
import pickle

studypik = pickle.load(open('study.pickle', 'rb'))
print(studypik.best_params)
print(studypik.best_value)
pickle.dump(studypik, open('study.pickle', 'wb'))
```

[ ]:
```
!pip install pyyaml h5py
```

[ ]:
```
import os

import tensorflow as tf
from tensorflow import keras

print(tf.version.VERSION)
```

[ ]:
```
new_model = tf.keras.models.load_model('/content/drive/MyDrive/optunam.h5')

# Check its architecture
new_model.summary()
```

[ ]:
```
import os


model = tf.keras.models.load_model("/content/trialmodel_0.9764999747276306.h5")
```