```
#Download dataset from Kaggle
! KAGGLE_CONFIG_DIR=/content/ kaggle datasets download daffafauzanazhari/bruised-facememar
```

```
! chmod 600 kaggle.json
```

New Section

```
#import library

import os
import zipfile
import random
import tensorflow as tf
import csv
import numpy as np
import shutil
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from shutil import copyfile
from os import getcwd

```

```
!ls
```

```
#Dataset being extract and placed in directory
path_violence_and_nonviolence = f"{getcwd()}/bruised-facememar.zip"
#shutil.rmtree('/tmp')

local_zip = path_violence_and_nonviolence
zip_ref = zipfile.ZipFile(local_zip, "r")
zip_ref.extractall('/tmp')
zip_ref.close()

```

```
!ls
```

```
# Dataset amount
print(len(os.listdir("/tmp/dataset/dataset/memar")))
print(len(os.listdir("/tmp/dataset/dataset/non-memar")))
```

```
try:
  os.mkdir("/tmp/violence-v-nonviolence/")
  os.mkdir("/tmp/violence-v-nonviolence/training/")
  os.mkdir("/tmp/violence-v-nonviolence/testing/")
  os.mkdir("/tmp/violence-v-nonviolence/training/violence/")
  os.mkdir("/tmp/violence-v-nonviolence/training/nonviolence/")
  os.mkdir("/tmp/violence-v-nonviolence/testing/violence/")
  os.mkdir("/tmp/violence-v-nonviolence/testing/nonviolence/")
except OSError:
  pass
```

```
!ls
```

```
def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
    source_list = random.sample(os.listdir(SOURCE), len(os.listdir(SOURCE)))

    for file number in range(len(source list)):
```

```
            file_source = os.path.join(SOURCE, source_list[file_number-1])
            file_training = os.path.join(TRAINING, source_list[file_number-1])
            file_validation = os.path.join(TESTING, source_list[file_number-1])

            size = os.path.getsize(file_source)

            if (file_number)<(len(source_list)*SPLIT_SIZE):
                if size > 0:
                    copyfile(file_source, file_training)
            elif size > 0:
                copyfile(file_source, file_validation)



VIOLENCE_SOURCE_DIR = "/tmp/dataset/dataset/memar/"
TRAINING_VIOLENCE_DIR = "/tmp/violence-v-nonviolence/training/violence/"
TESTING_VIOLENCE_DIR = "/tmp/violence-v-nonviolence/testing/violence/"
NONVIOLENCE_SOURCE_DIR = "/tmp/dataset/dataset/non-memar/"
TRAINING_NONVIOLENCE_DIR = "/tmp/violence-v-nonviolence/training/nonviolence/"
TESTING_NONVIOLENCE_DIR = "/tmp/violence-v-nonviolence/testing/nonviolence/"

split_size = .7
split_data(VIOLENCE_SOURCE_DIR, TRAINING_VIOLENCE_DIR, TESTING_VIOLENCE_DIR, split_size)
split_data(NONVIOLENCE_SOURCE_DIR, TRAINING_NONVIOLENCE_DIR, TESTING_NONVIOLENCE_DIR, spli
```

```
print(len(os.listdir("/tmp/violence-v-nonviolence/training/violence/")))
print(len(os.listdir("/tmp/violence-v-nonviolence/training/nonviolence/")))
print(len(os.listdir("/tmp/violence-v-nonviolence/testing/violence/")))
print(len(os.listdir("/tmp/violence-v-nonviolence/testing/nonviolence/")))
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation="relu", input_shape=(150,150,3)),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(512, activation="relu"),

    tf.keras.layers.Dense(1, activation="sigmoid"),
])

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

```
TRAINING_DIR ="/tmp/violence-v-nonviolence/training/"
train_datagen = ImageDataGenerator(
    rescale = 1.0/255,
    rotation_range =40,
    width_shift_range =0.2,
    height_shift_range = 0.2,
    shear_range =0.2,
    zoom_range =0.2,
    horizontal_flip =True,
    fill_mode = "nearest")

train_generator = train_datagen.flow_from_directory(TRAINING_DIR, batch_size=10, class_mod

VALIDATION_DIR ="/tmp/violence-v-nonviolence/testing/"
```

```
VALIDATION_DIR = "/tmp/violence-v-nonviolence/testing/"
validation_datagen = ImageDataGenerator(rescale=1.0/255.)

validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR, batch_size=1
```

```
history = model.fit_generator(train_generator,
                              epochs=5,
                              verbose=1,
                              validation_data=validation_generator)
```

Saving Keras Model .h5

```
KERAS_MODEL_NAME = "tf_model_bruisedface-memar.h5"
```

```
model.save(KERAS_MODEL_NAME)
```

```
convert_bytes(get_file_size(KERAS_MODEL_NAME), "MB")
```

```
# PLOT LOSS AND ACCURACY
%matplotlib inline

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

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

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

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r', "Training Accuracy")
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r', "Training Loss")
plt.plot(epochs, val_loss, 'b', "Validation Loss")


plt.title('Training and validation loss')

# Desired output. Charts with training and validation metrics. No crash :)
```

```
import numpy as np

from google.colab import files
from keras.preprocessing import image

uploaded=files.upload()

```

```
for fn in uploaded.keys():

  # predicting images
  path='/content/' + fn
  img=image.load_img(path, target_size=(150, 150))

  x=image.img_to_array(img)
  x=np.expand_dims(x, axis=0)
  images = np.vstack([x])

  classes = model.predict(images, batch_size=10)

  print(classes[0])

  if classes[0]>0:
    print(fn + " is a violence")

  else:
    print(fn + " is a non violence")

```

TF Lite Model (optimization)

```
TF_LITE_MODEL_FILE_NAME = "tf_lite_model_bruisedface-memar_optimized.tflite"
```

```
tf_lite_converter = tf.lite.TFLiteConverter.from_keras_model(model)
tf_lite_converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
# tf_lite_converter.optimizations = [tf.lite.Optimize.DEFAULT]
# tf_lite_converter.target_spec.supported_types = [tf.float16]
tflite_model = tf_lite_converter.convert()
```

```
tflite_model_name = TF_LITE_MODEL_FILE_NAME
open(tflite_model_name, "wb").write(tflite_model)
```

Convert Model Size

```
def get_file_size(file_path):
    size = os.path.getsize(file_path)
    return size
```

```
def convert_bytes(size, unit=None):
    if unit == "KB":
        return print('File size: ' + str(round(size / 1024, 3)) + ' Kilobytes')
    elif unit == "MB":
        return print('File size: ' + str(round(size / (1024 * 1024), 3)) + ' Megabytes')
    else:
        return print('File size: ' + str(size) + ' bytes')
```

```
convert_bytes(get_file_size(TF_LITE_MODEL_FILE_NAME), "KB")
```

Check Input Tensor Shape

```
interpreter = tf.lite.Interpreter(model_path = TF_LITE_MODEL_FILE_NAME)
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

```
output_details = interpreter.get_output_details()
print("Input Shape:", input_details[0]['shape'])
print("Input Type:", input_details[0]['dtype'])
print("Output Shape:", output_details[0]['shape'])
print("Output Type:", output_details[0]['dtype'])
```

Resize Tensor Shape (error)

```
interpreter.resize_tensor_input(input_details[0]['index'], (60, 150, 150, 3))
interpreter.resize_tensor_input(output_details[0]['index'], (60, 1))
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
print("Input Shape:", input_details[0]['shape'])
print("Input Type:", input_details[0]['dtype'])
print("Output Shape:", output_details[0]['shape'])
print("Output Type:", output_details[0]['dtype'])
```

```
validation_generator.dtype
```

```
test_imgs_numpy = np.array(validation_generator, dtype=np.float32)
```

```
interpreter.set_tensor(input_details[0]['index'], test_imgs_numpy)
interpreter.invoke()
tflite_model_predictions = interpreter.get_tensor(output_details[0]['index'])
print("Prediction results shape:", tflite_model_predictions.shape)
prediction_classes = np.argmax(tflite_model_predictions, axis=1)
```

```
acc = accuracy_score(prediction_classes, test_labels)
```

```
print('Test accuracy TFLITE model :', acc)
```