Import frames and train, test, evaluate network on facial images

Importing Dependencies

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
PROJECT_FOLDER = '/content/drive/MyDrive/CNN_Emotion_Classification/'
```

```
# %cd /content/drive/MyDrive/CNN_Emotion_Classification/
```

```
# copy content in main folder
! cp -a {PROJECT_FOLDER}. ./
```

```
!pip install -r Requirements/pireqs_opencv_contrib_env.txt
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_opencv_contrib_env.t
Requirement already satisfied: opencv_contrib_python in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_openc
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_opencv_contrib_env.t
Requirement already satisfied: scikit_learn in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_opencv_contrib
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_opencv_contrib_env.t
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_opencv_contrib_env
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (from -r Requirements/pireqs_opencv_contrib_e
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_c
Collecting jedi>=0.16 (from ipython->-r Requirements/pireqs_opencv_contrib_env.txt (line 2))
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 31.1 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_opencv_c
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_opencv
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_ope
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipyth
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_opencv_co
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_opencv_co
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython->-r Requirements/pireqs_opencv
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras_vggface->-r Requirements/pireqs_opencv_
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from keras_vggface->-r Requirements/pireqs_openc
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (from keras_vggface->-r Requirements/pireqs_opencv
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from keras_vggface->-r Requirements/pireqs_c
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from keras_vggface->-r Requirements/pireqs_openc
Requirement already satisfied: audioread>=2.1.9 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_c
Requirement already satisfied: joblib>=0.14 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_openc
Requirement already satisfied: numba>=0.51.0 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_oper
Requirement already satisfied: soundfile>=0.12.1 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_
Requirement already satisfied: pooch>=1.1 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_opencv_
Requirement already satisfied: soxr>=0.3.2 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_opencv
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/
Requirement already satisfied: lazy-loader>=0.1 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_c
Requirement already satisfied: msgpack>=1.0 in /usr/local/lib/python3.10/dist-packages (from librosa->-r Requirements/pireqs_openc
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/pirec
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/pireqs_op
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/pire
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/pire
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/pireqs
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/pirec
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->-r Requirements/p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->-r Requirements/pireqs_opencv
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->-r Requirements/pireqs_oper
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit_learn->-r Requirements
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pireqs_
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pire
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/p
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pi
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pirec
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pirec
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pire
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pirec
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pi
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirements/pi
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r Requirement
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow->-r
```

Double-click (or enter) to edit

```

```

```
# !pip install keras_applications
```

```
from display import pltDisplay
from pathlib import Path
from matplotlib.colors import ListedColormap
from sklearn.metrics import classification_report, roc_curve, auc, roc_auc_score
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
# from sorting import human_sort
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.layers import BatchNormalization, Dropout, Activation, ReLU, Softmax
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
# import IPython.display as ipd


import constants as const
import csv
import cv2
import json
import logging
import gc
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import seaborn as sns
import tensorflow as tf
import subprocess
import tensorflow.keras.backend as K
import time
import utils

from keras_vggface.vggface import VGGFace
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input as preprocess_imagenet
from keras_vggface.utils import preprocess_input as preprocess_vggface
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  try:
    # Currently, memory growth needs to be the same across GPUs
    for gpu in gpus:
        print("Name:", gpu.name, "  Type:", gpu.device_type)
        print(tf.config.experimental.get_device_details(gpu))
        print(tf.config.experimental.get_memory_info('GPU:0'))
        # tf.config.experimental.set_memory_growth(gpu, True)
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    # Memory growth must be set before GPUs have been initialized
    print(e)
```

```
Name: /physical_device:GPU:0   Type: GPU
{'compute_capability': (7, 5), 'device_name': 'Tesla T4'}
{'current': 0, 'peak': 0}
1 Physical GPUs, 1 Logical GPUs
```

```
# unzip archive in folder
!7z x Generated/Frames_300.zip -oGenerated/
```

```

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Xeon(R) CPU @ 2.00GHz (50653),ASM,AES-NI)

Scanning the drive for archives:
1 file, 4154172627 bytes (3962 MiB)

Extracting archive: Generated/Frames_300.zip
--
Path = Generated/Frames_300.zip
Type = zip
Physical Size = 4154172627
```

```
64-bit = +

Everything is Ok

Folders: 202
Files: 158556
Size:       4143561934
Compressed: 4154172627
```

```
utils.modules_info()
```

```
        OpenCV:
            Version: 4.8.0
        Tensorflow:
            Version: 2.15.0
```

```
log_file = Path(const.logs_path, 'FACER.log')
logging.basicConfig(
    format='%(asctime)s %(message)s',
    filemode='a',
    filename=log_file,
    encoding='utf-8',
    level=logging.INFO,
    force=True
)
```

Importing Dataset

```
data_df = pd.read_csv(Path(const.csv_path, 'dataset.csv'))
```

Frame

Preparing Data

Dataset Creation for ML

```
IMG_WIDTH = IMG_HEIGHT = 224
IMG_CHANNELS = 3
SEED = 42
BATCH_SIZE = 128
VALIDATION_SPLIT = 0.2
EMOTIONS_LABELS = const.EMOTIONS_LABELS  # RAVDESS emotion labels
```

```
EMOTIONS_LABELS
```

```
['neutral', 'calm', 'happy', 'sad', 'angry', 'fearful', 'disgust', 'surprised']
```

```
TOTAL_ELEMENTS = const.DATASET_TOTAL_ELEMENTS
label_names = const.EMOTIONS_LABELS_SORTED.copy()
# label_names.remove('neutral')
# label_names.remove('calm')
# label_names.remove('surprised')

NUM_CLASSES = len(label_names)
```

Dataset Creation - NEW

```
actors_labels = [f'{i:02d}' for i in range(1, 25)]
dist_idxs = {
    '1': [slice(0, 16), slice(16, 20), slice(20, 24)],
    '2': [slice(8, 24), slice(4, 8), slice(0, 4)],
    '3': [slice(4, 20), slice(20, 24), slice(0, 4)]
}
```

```python
# Split actors in train, validation, test
dist_n = 1
train_idxs, val_idxs, test_idxs = [actors_labels[i] for i in dist_idxs[str(dist_n)]]

print(train_idxs, val_idxs, test_idxs)
```

```
['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16'] ['17', '18', '19', '20'] ['21', '22
```

```python
def make_dataset(path, actors_idx, talk_frame=False, acted_frame=False,
                 undersampling=False, preprocess_vgg=True, shuffle=False, sampling=1):

    def parse_image(filename):
        image = tf.io.read_file(filename)
        image = tf.image.decode_jpeg(image, channels=IMG_CHANNELS)
        image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH])
        if (preprocess_vgg):
            image = preprocess_vgg(image)
        else:
            image = image / 255

        return image

    filenames = []
    talk_regex = '*-01.jpg' if talk_frame else '*.jpg'
    acted_regex = '02' if acted_frame else '*'
    gen_regex = f'*-*-*-{acted_regex}-*-*-*-{talk_regex}'

    file_dict = dict()
    for label in sorted(label_names):
        file_dict[label] = []


    for label in label_names:
        for actor in actors_idx:
            for file in Path(path, label, actor).glob(f'{gen_regex}'):
                file_dict[label].append(str(file))


    if shuffle:
        for label, item in file_dict.items():
            logging.info(f'Label: {label}')
            logging.info(f'Array len: {len(item)}')
            random.Random(SEED).shuffle(item)

    arr_len = [len(arr) for arr in file_dict.values()]

    if undersampling:
        filenames = [arr[:min(arr_len)] for arr in file_dict.values()]
    else:
        filenames = [arr for arr in file_dict.values()]

    filenames = sum(filenames, [])

    if shuffle:
        random.Random(SEED).shuffle(filenames)

    labels = [
        label_names.index(EMOTIONS_LABELS[int(utils.get_class(elem)) - 1])
        for elem in filenames
    ]

    if (sampling < 1):
        filenames, _, labels, _ = train_test_split(
            filenames, labels, train_size=sampling, random_state=SEED
        )

    filenames_ds = tf.data.Dataset.from_tensor_slices(filenames)
    labels_ds = tf.data.Dataset.from_tensor_slices(labels)

    images_ds = filenames_ds.map(
        parse_image, num_parallel_calls=tf.data.experimental.AUTOTUNE
    )
    ds = tf.data.Dataset.zip((images_ds, labels_ds))
    # ds = configure_for_performance(ds)

    return [ds, filenames]
```

```
sampling_rate = 1
talk_frame = True
acted_frame = False
preprocess_vgg = 'Imagenet'  # False, Imagenet or VGGFace

if(preprocess_vgg == 'Imagenet'):
    preprocess_vgg = preprocess_imagenet
elif(preprocess_vgg == 'VGGFace'):
    preprocess_vgg = preprocess_vggface

train_ds, train_files = make_dataset(
    const.frames_path, train_idxs, talk_frame=talk_frame, acted_frame=acted_frame,
    preprocess_vgg=preprocess_vgg, shuffle=True, sampling=sampling_rate
)

val_ds, val_files = make_dataset(
    const.frames_path, val_idxs, talk_frame=talk_frame, acted_frame=acted_frame,
    preprocess_vgg=preprocess_vgg, sampling=sampling_rate
)

test_ds, test_files = make_dataset(
    const.frames_path, test_idxs, talk_frame=talk_frame, acted_frame=acted_frame,
    preprocess_vgg=preprocess_vgg, sampling=sampling_rate
)
```

```
# train_ds = train_ds[:31882]
# train_files = train_files[:31882]

# val_ds = val_ds[:8426]
# val_files = val_files[:8426]

# test_ds = test_ds[:8301]
# test_files = test_files[:8301]
```

```
assert len(train_ds) == len(train_files), len(train_files)
assert len(val_ds) == len(val_files), len(val_files)
assert len(test_ds) == len(test_files), len(test_files)
```

```
train_ds_elements = len(train_ds)
test_ds_elements = len(test_ds)
val_ds_elements = len(val_ds)
```

```
print(f'train_ds samples: {train_ds_elements}')
print(f'test_ds samples: {test_ds_elements}')
print(f'val_ds samples: {val_ds_elements}')
```

```
train_ds samples: 67906
test_ds samples: 17994
val_ds samples: 17642
```

Build and train the model

Add operations to reduce read latency while training the model:

`ds.batch` Combines consecutive elements of the dataset into batches. The components of the resulting element will have an additional outer dimension, which will be *batch_size*

`ds.cache` Caches the elements in this dataset.

`ds.prefetch` Allows later elements to be prepared while the current element is being processed. This often improves latency and throughput, at the cost of using additional memory to store prefetched elements.

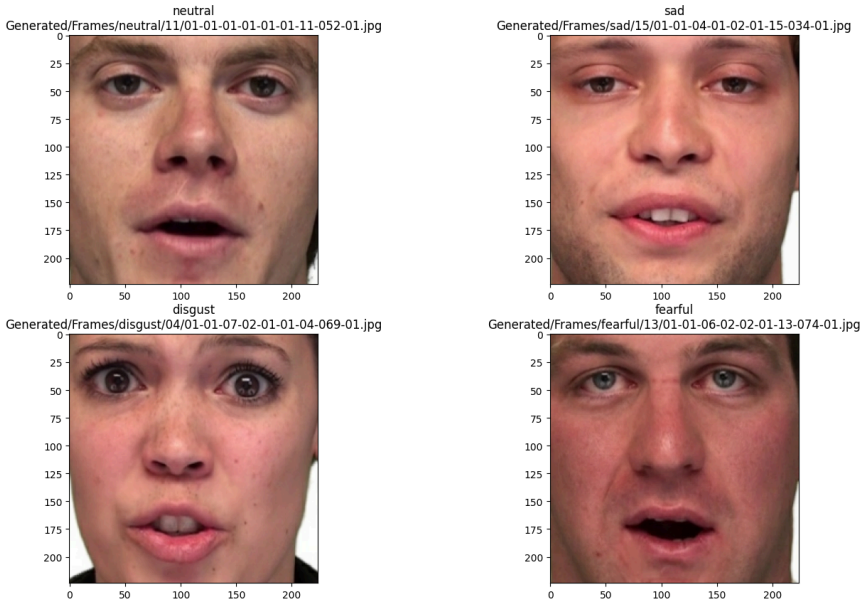
```
def configure_for_performance(ds, batch_size=BATCH_SIZE):
    ds = ds.batch(batch_size)
    # ds = ds.cache()
    # ds = ds.shuffle(buffer_size=1000)
    # ds = ds.repeat()
    ds = ds.prefetch(buffer_size=tf.data.AUTOTUNE)
    return ds
```

```
train_ds = configure_for_performance(train_ds)
val_ds = configure_for_performance(val_ds)
test_ds = configure_for_performance(test_ds)
```

```
for example_images, example_labels in train_ds.take(1):
    print(example_images.shape)
    print(example_labels.shape)
```

```
(128, 224, 224, 3)
(128,)
```

```
plt.figure(figsize=(16, 10))
rows = 2
cols = 2
n = rows * cols
for i in range(n):
    plt.subplot(rows, cols, i + 1)
    image = example_images[i]
    plt.imshow(image)  # 3 channels
    # plt.imshow(image * 255, cmap='gray', vmin=0, vmax=255)  # 1 channel
    plt.title(f'{label_names[example_labels[i]]}\n{train_files[i]}')
```



```
def create_Bilotti_CNN(name='Bilotti_CNN'):

    inputs = Input(shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))

    conv1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inputs)
    conv2 = Conv2D(32, kernel_size=(3, 3), activation='relu')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(64, kernel_size=(3, 3), activation='relu')(pool1)
    conv4 = Conv2D(64, kernel_size=(3, 3), activation='relu')(conv3)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv4 = Conv2D(18, kernel_size=(3, 3), activation='relu')(pool2)
    conv5 = Conv2D(18, kernel_size=(3, 3), activation='relu')(conv4)
    conv6 = Conv2D(18, kernel_size=(3, 3), activation='relu')(conv5)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv6)

    conv7 = Conv2D(56, kernel_size=(3, 3), activation='relu')(pool3)
    conv8 = Conv2D(56, kernel_size=(3, 3), activation='relu')(conv7)
    conv9 = Conv2D(56, kernel_size=(3, 3), activation='relu')(conv8)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv9)

    conv10 = Conv2D(51, kernel_size=(3, 3), activation='relu')(pool4)
    conv11 = Conv2D(51, kernel_size=(3, 3), activation='relu')(conv10)
    conv12 = Conv2D(51, kernel_size=(3, 3), activation='relu')(conv11)
    pool5 = MaxPooling2D(pool_size=(2, 2))(conv12)

    flatten = Flatten()(pool5)

    dense1 = Dense(2048, activation='relu')(flatten)
    drop1 = Dropout(0.25)(dense1)

    dense2 = Dense(1024, activation='relu')(drop1)
    drop2 = Dropout(0.4)(dense2)

    output = Dense(NUM_CLASSES, activation='softmax')(drop2)

    model = Model(inputs, output)

    model._name = name

    return model
```

```
def create_VGG16_Imagenet(name='VGG16_Imagenet'):

    base_model = VGG16(
        weights='imagenet',
        include_top=False,
        input_shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS)
    )
    base_model.trainable = False  # Not trainable weights

    flatten_layer = Flatten()
    dense_layer_1 = Dense(2048, activation='relu')
    drop_1 = Dropout(0.4)
    dense_layer_2 = Dense(1024, activation='relu')
    drop_2 = Dropout(0.4)
    dense_layer_3 = Dense(512, activation='relu')
    drop_3 = Dropout(0.4)
    prediction_layer = Dense(NUM_CLASSES, activation='softmax')

    model = Sequential([
        base_model,
        flatten_layer,
        dense_layer_1,
        drop_1,
        dense_layer_2,
        drop_2,
        dense_layer_3,
        drop_3,
        prediction_layer
    ])

    model._name = name

    return model
```

```python
def create_VGG16_VGGFACE(name='VGG16_VGGFACE'):
    nb_class = NUM_CLASSES

    vgg_model = VGGFace(
        include_top=False, weights='vggface', input_shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS)
    )
    last_layer = vgg_model.get_layer('pool5').output
    x = Flatten(name='flatten')(last_layer)

    x = Dense(512, activation='relu', name='fc6')(x)
    x = Dropout(0.35)(x)
    x = Dense(256, activation='relu', name='fc7')(x)
    x = Dropout(0.35)(x)
    x = Dense(128, activation='relu', name='fc8')(x)
    x = Dropout(0.35)(x)

    out = Dense(nb_class, activation='softmax', name='fc9')(x)

    custom_vgg_model = Model(vgg_model.input, out)
    custom_vgg_model._name = name

    return custom_vgg_model
```

```python
tf.keras.backend.clear_session()  # clear all precedent models and sessions
```

```python
check_path = 'checkpoint.weights.h5'
checkpointer = ModelCheckpoint(
    check_path, monitor='val_accuracy', verbose=1, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch'
)
```

```python
# model = create_cnn_model()
# model = medium_model()
model = create_VGG16_Imagenet()
# model = create_grigorasi_model()
# model = create_Bilotti_CNN()
model.summary()
```

```
Model: "VGG16_Imagenet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_2 (Flatten)         (None, 25088)             0

 dense_7 (Dense)             (None, 2048)              51382272

 dropout_5 (Dropout)         (None, 2048)              0

 dense_8 (Dense)             (None, 1024)              2098176

 dropout_6 (Dropout)         (None, 1024)              0

 dense_9 (Dense)             (None, 512)               524800

 dropout_7 (Dropout)         (None, 512)               0

 dense_10 (Dense)            (None, 8)                 4104

=================================================================
Total params: 68724040 (262.16 MB)
Trainable params: 54009352 (206.03 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
# Define training callbacks

class TimeHistory(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.times = []

    def on_epoch_begin(self, batch, logs={}):
        self.epoch_time_start = time.time()

    def on_epoch_end(self, batch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)


early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    verbose=1,
    patience=5,
    restore_best_weights=True
)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, verbose=1,
                              patience=3, min_lr=0)
```

```
METRICS = ['accuracy']
```

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(
        learning_rate=1e-4
    ),
    # optimizer=tf.keras.optimizers.SGD(),  # for VGG16_VGGFACE
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=METRICS,
)
```

Train Model

```
EPOCHS = 100
time_callback = TimeHistory()
history = model.fit(
    x=train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=[time_callback, early_stopping_callback, reduce_lr]
)
```

```
Epoch 1/100
531/531 [==============================] - 500s 898ms/step - loss: 1.2592 - accuracy: 0.6591 - val_loss: 2.1001 - val_accuracy: 0.47
Epoch 2/100
531/531 [==============================] - 389s 732ms/step - loss: 0.2766 - accuracy: 0.9060 - val_loss: 3.1878 - val_accuracy: 0.47
Epoch 3/100
531/531 [==============================] - 396s 745ms/step - loss: 0.1419 - accuracy: 0.9526 - val_loss: 3.3680 - val_accuracy: 0.47
Epoch 4/100
531/531 [==============================] - ETA: 0s - loss: 0.0990 - accuracy: 0.9667
Epoch 4: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.
531/531 [==============================] - 388s 730ms/step - loss: 0.0990 - accuracy: 0.9667 - val_loss: 3.8887 - val_accuracy: 0.46
Epoch 5/100
531/531 [==============================] - 381s 718ms/step - loss: 0.0402 - accuracy: 0.9870 - val_loss: 4.0004 - val_accuracy: 0.47
Epoch 6/100
531/531 [==============================] - ETA: 0s - loss: 0.0246 - accuracy: 0.9922Restoring model weights from the end of the best
531/531 [==============================] - 381s 717ms/step - loss: 0.0246 - accuracy: 0.9922 - val_loss: 4.1271 - val_accuracy: 0.47
Epoch 6: early stopping
```

```
EPOCHS = len(time_callback.times)
```

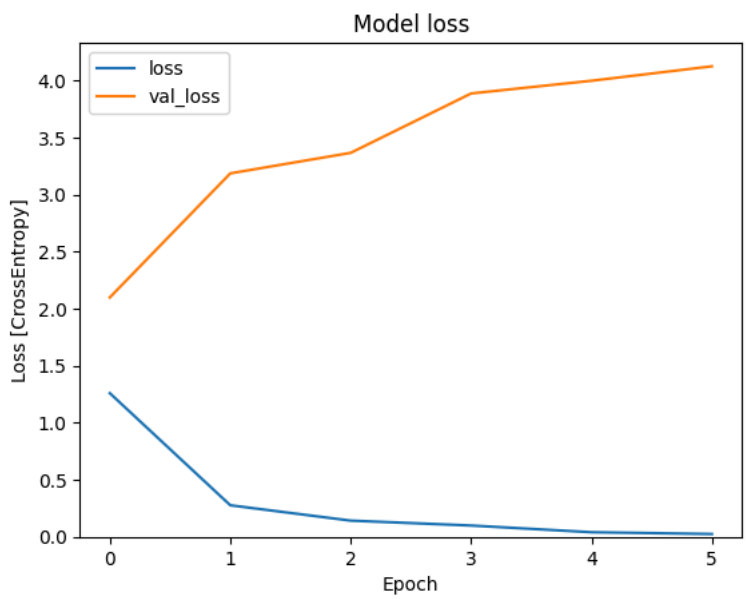
```
# Create model path
model_path = Path(const.models_path, 'Frame', model._name)
run_folders = list(Path(const.models_path, 'Frame', model._name).glob('Run_*'))

if not run_folders:
    model_path = Path(model_path, 'Run_1')
else:
    last_run = run_folders.pop()
    last_run_idx = Path(last_run).name.split('_')[-1]
    model_path = Path(model_path, f'Run_{int(last_run_idx) + 1}')

model_path.mkdir(parents=True, exist_ok=False)
```

```
# Save info on the indexes used for train, val and test
ds_info_path = Path(model_path, f'{model._name}_dataset.txt')
with open(ds_info_path, 'w+', newline='') as res_file:
    res_file.write(f'Train indexes: {train_idxs}\n')
    res_file.write(f'Train files: {train_ds_elements}\n')
    res_file.write(f'Val indexes: {val_idxs}\n')
    res_file.write(f'Val files: {val_ds_elements}\n')
    res_file.write(f'Test indexes: {test_idxs}\n')
    res_file.write(f'Test files: {test_ds_elements}\n')
```

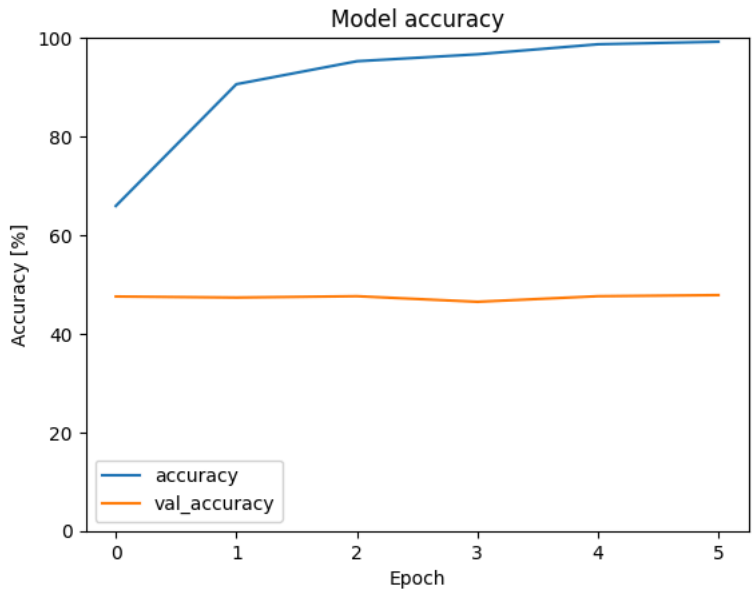
```
metrics = history.history
```

```
mod_loss = metrics['loss']
mod_val_loss = metrics['val_loss']
mod_accuracy = metrics['accuracy']
mod_val_accuracy = metrics['val_accuracy']
# mod_f1 = metrics['fBeta_score']
# mod_val_f1 = metrics['val_fBeta_score']

mod_mean_loss = np.mean(mod_loss)
mod_mean_val_loss = np.mean(mod_val_loss)
mod_mean_accuracy = np.mean(mod_accuracy)
mod_mean_val_accuracy = np.mean(mod_val_accuracy)
# mod_mean_f1 = np.mean(mod_f1)
# mod_mean_val_f1 = np.mean(mod_val_f1)
```

```
# Save Loss
plt.title('Model loss')
plt.plot(history.epoch, mod_loss, mod_val_loss)
plt.legend(['loss', 'val_loss'])
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch')
plt.ylabel('Loss [CrossEntropy]')
plt.savefig(Path(model_path, 'loss.png'))
```




```
# Save Accuracy
plt.title('Model accuracy')
plt.plot(
    history.epoch,
    100 * np.array(mod_accuracy),
    100 * np.array(mod_val_accuracy)
)
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
plt.savefig(Path(model_path, 'accuracy.png'))
```



Unsupported Cell Type. Double-Click to inspect/edit the content.

Unsupported Cell Type. Double-Click to inspect/edit the content.

Evaluate Model

```
model_eval = model.evaluate(test_ds, return_dict=True)
```

```
141/141 [==============================] - 99s 701ms/step - loss: 1.7415 - accuracy: 0.4535
```

```
# Save model
model.save(Path(model_path, f'{model._name}.keras'), overwrite=False)
# Save history
np.save(Path(model_path, f'{model._name}_history.npy'), history)
# Save model image
model_img = tf.keras.utils.plot_model(
    model, Path(model_path, f'{model._name}.png'), show_shapes=True,
    show_layer_names=True, show_layer_activations=True
)
```

```
model_eval
```

```
{'loss': 1.7415306568145752, 'accuracy': 0.45348450541496277}
```

```
test_loss = model_eval['loss']
test_accuracy = model_eval['accuracy']
# test_f1 = model_eval['fBeta_score']
mean_epoch_time = np.mean(time_callback.times)
```

```
# Salvataggio informazioni modello
model_save_path = Path(model_path, f'{model._name}_result.txt')
with open(model_save_path, 'w+', newline='') as res_file:
    res_file.write(f'BATCH: {BATCH_SIZE}\n')
    res_file.write(f'Train loss: {str(mod_loss)}\n')
    res_file.write(f'val_loss: {str(mod_val_loss)}\n')
    res_file.write(f'Train accuracy: {str(mod_accuracy)}\n')
    res_file.write(f'Train val_accuracy: {str(mod_val_accuracy)}\n')
    # res_file.write(f'Train f1_score: {str(mod_f1)}\n')
    # res_file.write(f'Train val_f1_score: {str(mod_val_f1)}\n')
    res_file.write(f'Test loss: {str(test_loss)}\n')
    res_file.write(f'Test accuracy: {str(test_accuracy)}\n')
    # res_file.write(f'Test f1_score: {str(test_f1)}\n')
    res_file.write(f'Mean epoch time: {str(mean_epoch_time)}')
```

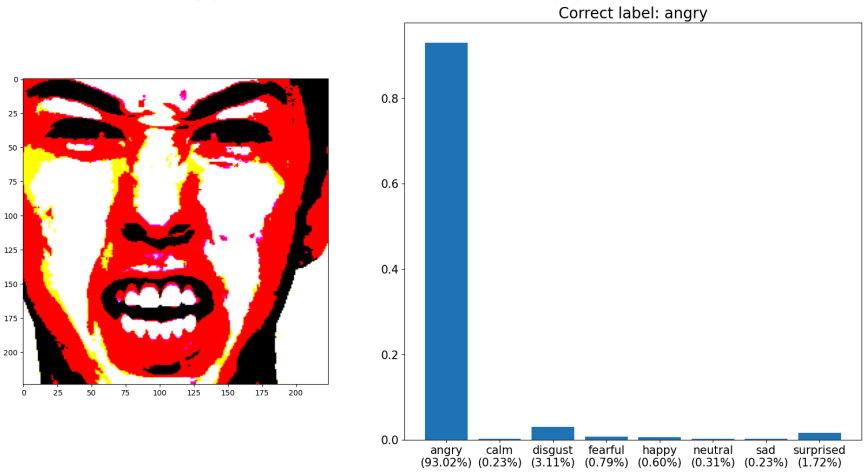
```
# Salvataggio informazioni generali modelli
with open(Path(const.models_path, 'Frame', 'models.csv'), 'a+') as csvfile:
    filewriter = csv.writer(
        csvfile, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL
    )

    # filewriter.writerow(
    #     ["Model Name", "Epochs", "% Validation", "% Test set",
    #      "Train loss", "Train accuracy", "Val loss", "Val accuracy",
    #      "Test loss", "Test accuracy", "Mean epoch time", "Note"]
    # )
    test_ds_perc = utils.trunc((test_ds_elements * 100) / TOTAL_ELEMENTS, 2)
    val_ds_perc = utils.trunc((val_ds_elements * 100) / TOTAL_ELEMENTS, 2)
    full_path = str(Path(model._name, model_path.name))
    filewriter.writerow(
        [full_path, EPOCHS, val_ds_perc, test_ds_perc,
         mod_loss, mod_accuracy, mod_val_loss, mod_val_accuracy,
         test_loss, test_accuracy, mean_epoch_time, '']
    )
```

```
for test_images, test_labels in test_ds.take(1):
    print(test_images.shape)
    print(test_labels.shape)
```

```
(128, 224, 224, 3)
(128,)
```

```
gen = np.random.default_rng(seed=None)
idx = gen.integers(0, len(test_images))
print(test_files[idx])

image = test_images[idx]
label = test_labels[idx]

net_input = utils.extend_tensor(image, 0)
prediction = model(net_input)
prediction = prediction[0].numpy()

valued_arr = []

for idx, name in enumerate(label_names):
    valued_arr.append(f'{name}\n({prediction[idx]:.2%})')

fig, ax = plt.subplots(
    nrows=1, ncols=2, width_ratios=[0.4, 0.6], figsize=(20, 10)
)

pltDisplay(image * 255, ax=ax[0])  # 1 channel
# pltDisplay(image, ax=ax[0])

ax[1].bar(valued_arr, prediction)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title(f'Correct label: {label_names[label]}', fontsize=20)
# plt.xlabel('Predicted class')
# plt.ylabel('Percentage')
plt.show()
```

```
Generated/Frames/angry/21/01-01-05-02-01-01-21-035-01.jpg
```



Display a confusion matrix

Use a confusion matrix to check how well the model did classifying each of the commands in the test set:

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
y_pred = model.predict(test_ds)
```

```
141/141 [==============================] - 91s 643ms/step
```

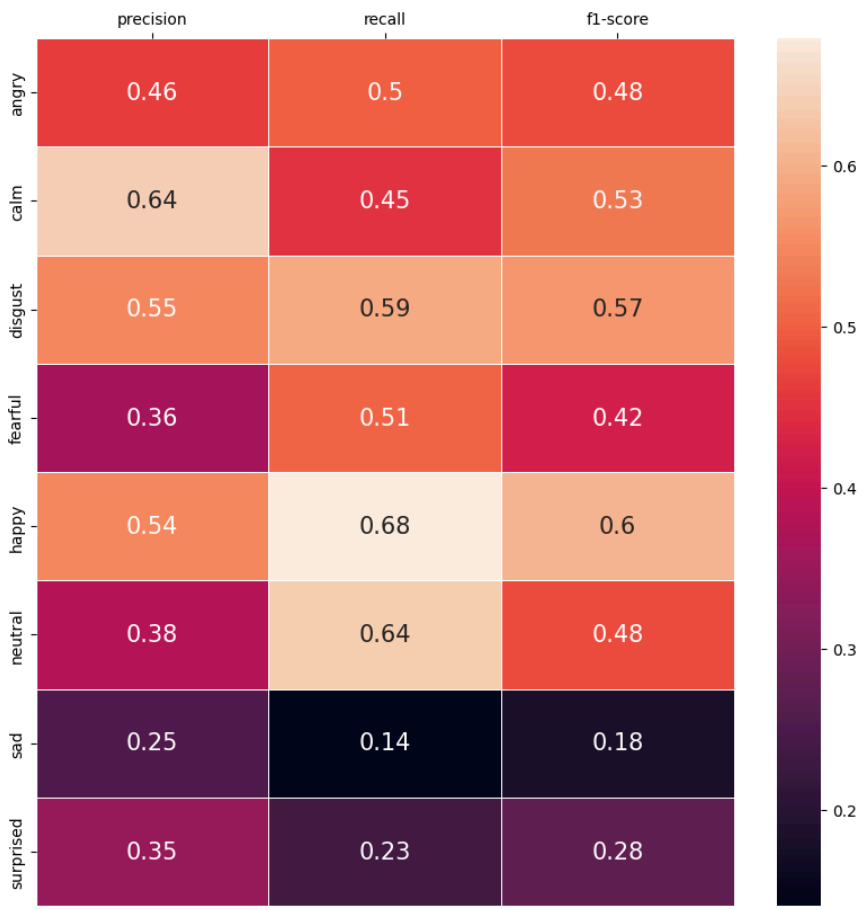
```
y_pred = tf.argmax(y_pred, axis=1, output_type=tf.int32)
```

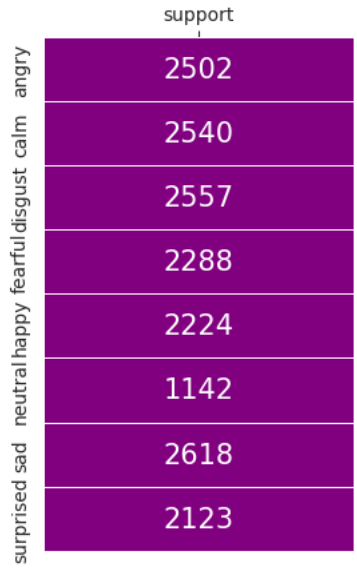
```
y_true = tf.concat(list(test_ds.map(lambda _, lab: lab)), axis=0)
```

```
report = classification_report(
    y_true, y_pred, target_names=label_names,
    output_dict=True, zero_division='warn'
)
```

```
rep_to_csv = pd.DataFrame(data=report).transpose()
```

```
fig, ax = plt.subplots(figsize=(10, 10))
ax.xaxis.tick_top()
sns.heatmap(rep_to_csv.iloc[:NUM_CLASSES, :3],
            cbar=True,
            square=False,
            annot=True,
            annot_kws={'size': 15},
            fmt='.2g',
            linewidths=0.5)
plt.savefig(Path(model_path, f'{model._name}_f1_score.png'))
plt.show()


with sns.axes_style('white'):
    fig, ax = plt.subplots(figsize=(3, 5))
    ax.xaxis.tick_top()
    sns.heatmap(rep_to_csv.iloc[:NUM_CLASSES, 3:],
                cbar=False,
                square=False,
                annot=True,
                annot_kws={'size': 15},
                fmt='.4g',
                cmap=ListedColormap([('purple')]),
                linewidths=0.5)
    plt.savefig(Path(model_path, f'{model._name}_support.png'))
    plt.show()
```





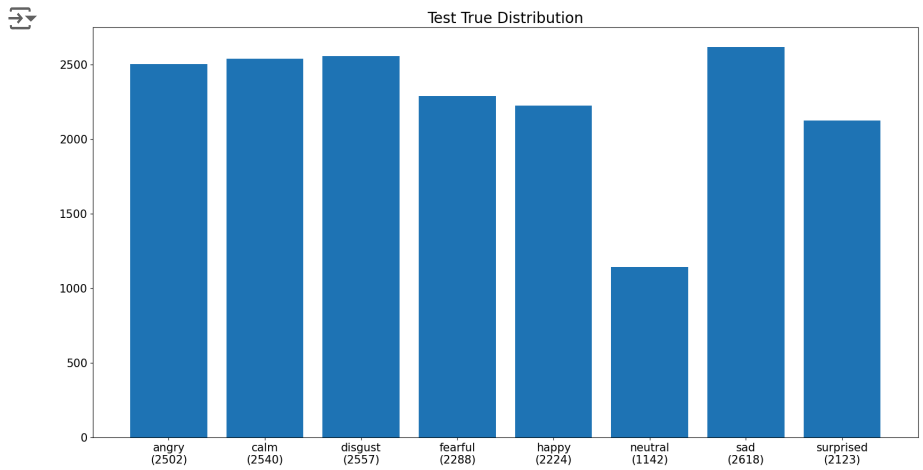

```
report_save_path = Path(model_path, f'{model._name}_report.csv')
rep_to_csv.to_csv(report_save_path)
```

Unsupported Cell Type. Double-Click to inspect/edit the content.

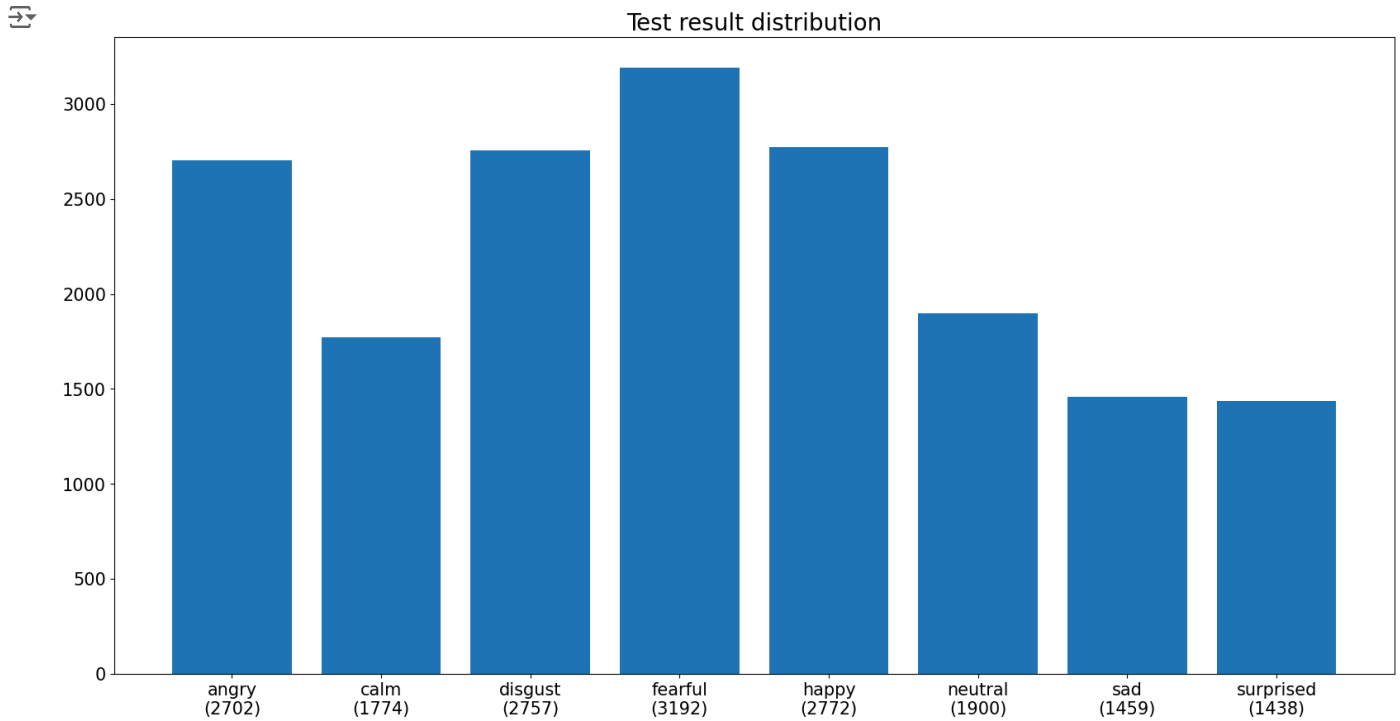
```
np.mean([report[c]['f1-score'] for c in list(report)[:NUM_CLASSES]])
```

```
0.442763100084397
```

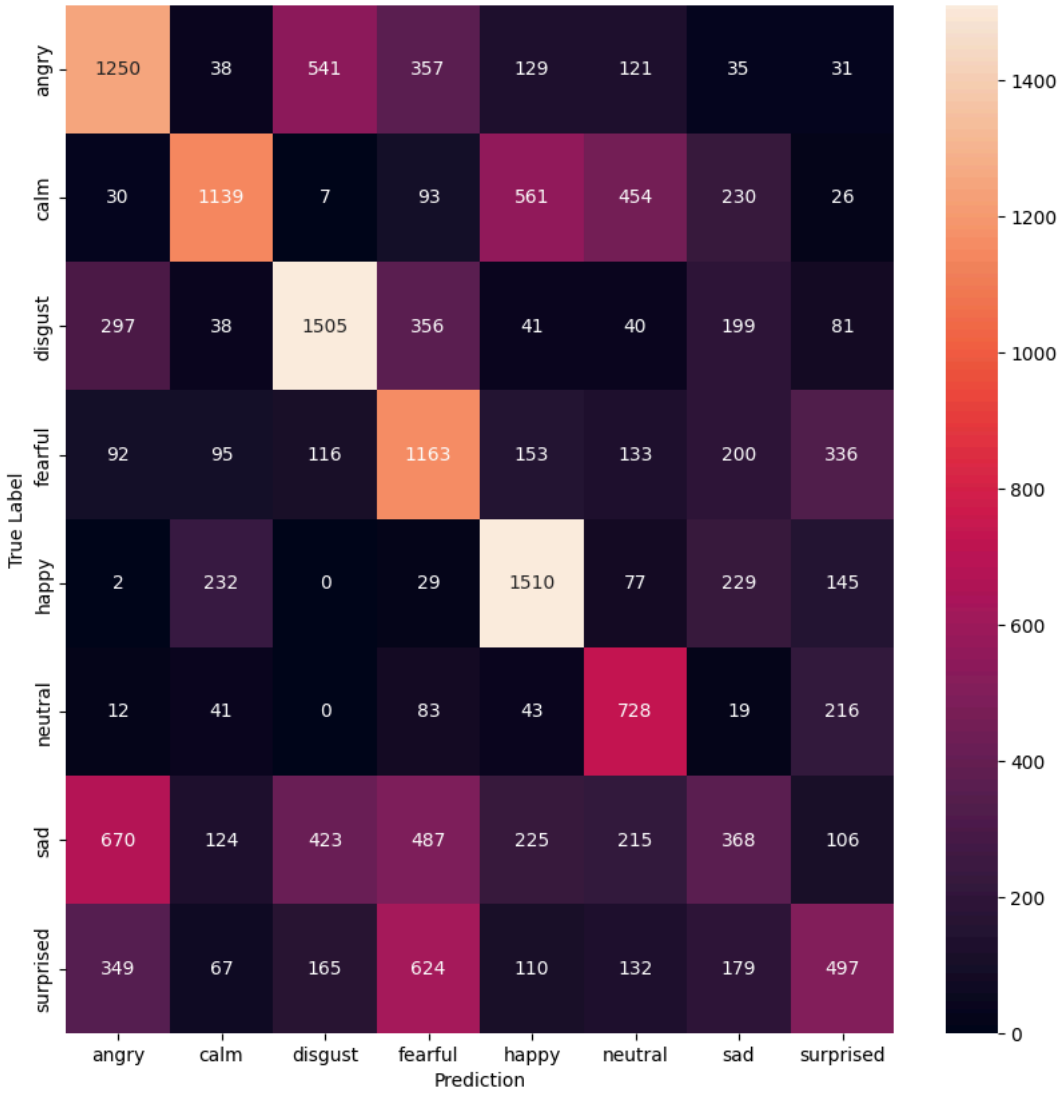
```
# True Test Distribution
# unique, counts = np.unique(y_true, return_counts=True)
# collections.Counter(y_true)
counts = [np.count_nonzero(y_true == idx) for idx in range(len(label_names))]
valued_arr = []
# for i in range(len(label_names)):
for idx, name in enumerate(label_names):
    count = counts[idx]
    valued_arr.append(f'{name}\n({count})')

fig = plt.subplots(figsize=(20, 10))
plt.bar(valued_arr, counts)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Test True Distribution', fontsize=20)
# plt.xlabel('Predicted class')
# plt.ylabel('Percentage')
plt.savefig(Path(model_path, f'{model._name}_trueDist.png'))
plt.show()
```



```
# Predicted Test Distribution
# unique, counts = np.unique(y_pred, return_counts=True)
counts = [np.count_nonzero(y_pred == idx) for idx in range(len(label_names))]
valued_arr = []
unique_idx = 0
for idx, name in enumerate(label_names):
    count = counts[idx]
    valued_arr.append(f'{name}\n({count})')

fig = plt.subplots(figsize=(20, 10))
plt.bar(valued_arr, counts)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Test result distribution', fontsize=20)
# plt.xlabel('Predicted class')
# plt.ylabel('Percentage')
plt.savefig(Path(model_path, f'{model._name}_predDist.png'))
plt.show()
```



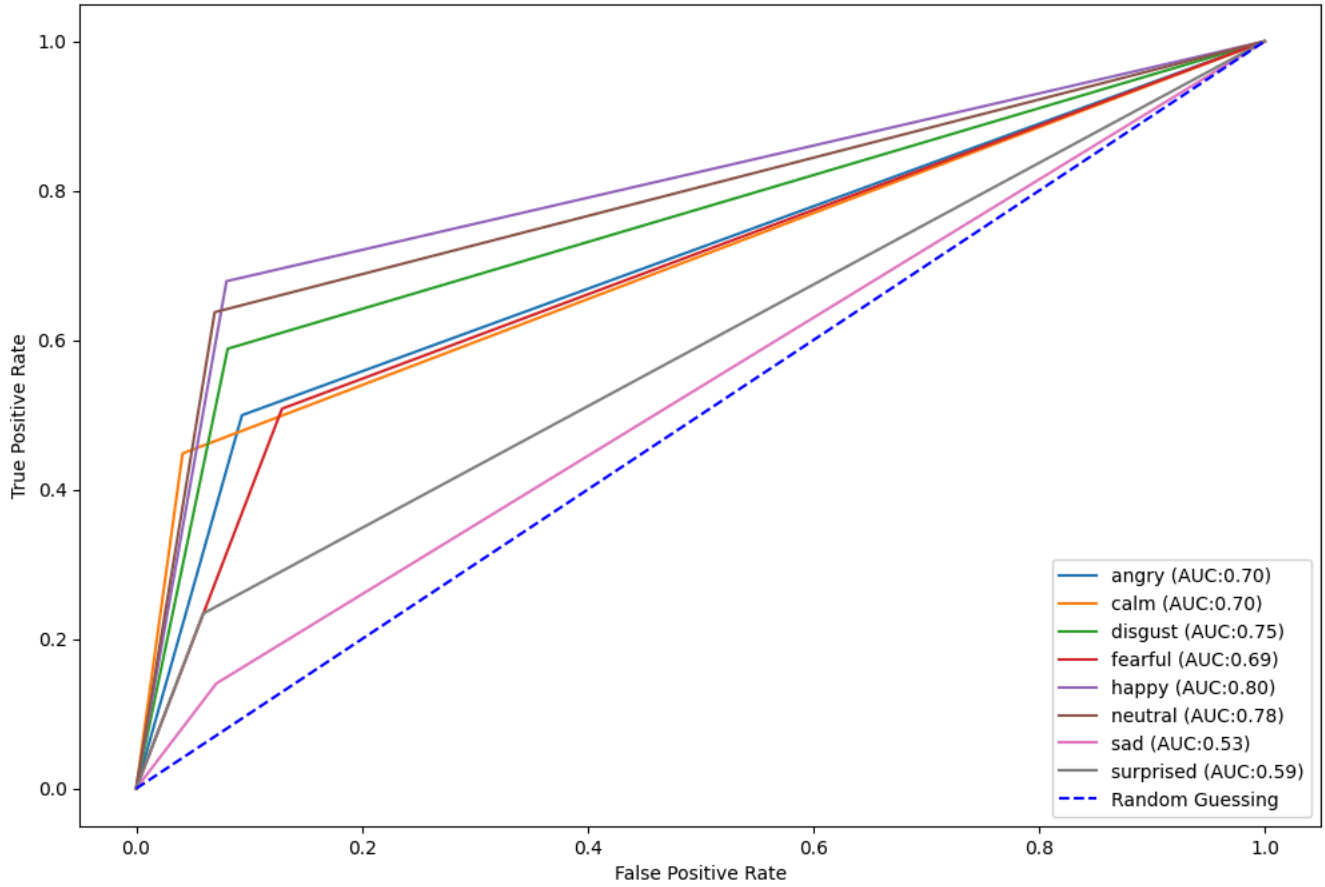

```
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(confusion_mtx,
            xticklabels=label_names,
            yticklabels=label_names,
            annot=True, fmt='.4g')
plt.xlabel('Prediction')
plt.ylabel('True Label')
plt.savefig(Path(model_path, f'{model._name}_heat.png'))
plt.show()
```




```python
def multiclass_roc_auc_score(target, y_test, y_pred, average="macro"):
    # function for scoring roc auc score for multi-class
    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)

    if len(target) > 2:
        for (idx, c_label) in enumerate(target):

            fpr, tpr, thresholds = roc_curve(
                y_test[:, idx].astype(int),
                y_pred[:, idx]
            )
            c_ax.plot(
                fpr, tpr, label='%s (AUC:%0.2f)' % (c_label, auc(fpr, tpr))
            )
    else:
        fpr, tpr, thresholds = roc_curve(
                y_test,
                y_pred
        )
        c_ax.plot(
            fpr, tpr, label='Model (AUC:%0.2f)' % (auc(fpr, tpr)), color='#ff7f0e'
        )
    c_ax.plot(fpr, fpr, color='b', linestyle='--', label='Random Guessing')

    return roc_auc_score(y_test, y_pred, average=average)
```

```
# set plot figure size
fig, c_ax = plt.subplots(1, 1, figsize=(12, 8))

print('ROC AUC score:', multiclass_roc_auc_score(
    label_names,
    tf.reshape(y_true, (y_true.shape[0], 1)),
    tf.reshape(y_pred, (y_pred.shape[0], 1))
))

c_ax.legend()
c_ax.set_xlabel('False Positive Rate')
c_ax.set_ylabel('True Positive Rate')
plt.savefig(Path(model_path, f'{model._name}_ROC.png'))
plt.show()
```

```
ROC AUC score: 0.6944431334629377
```

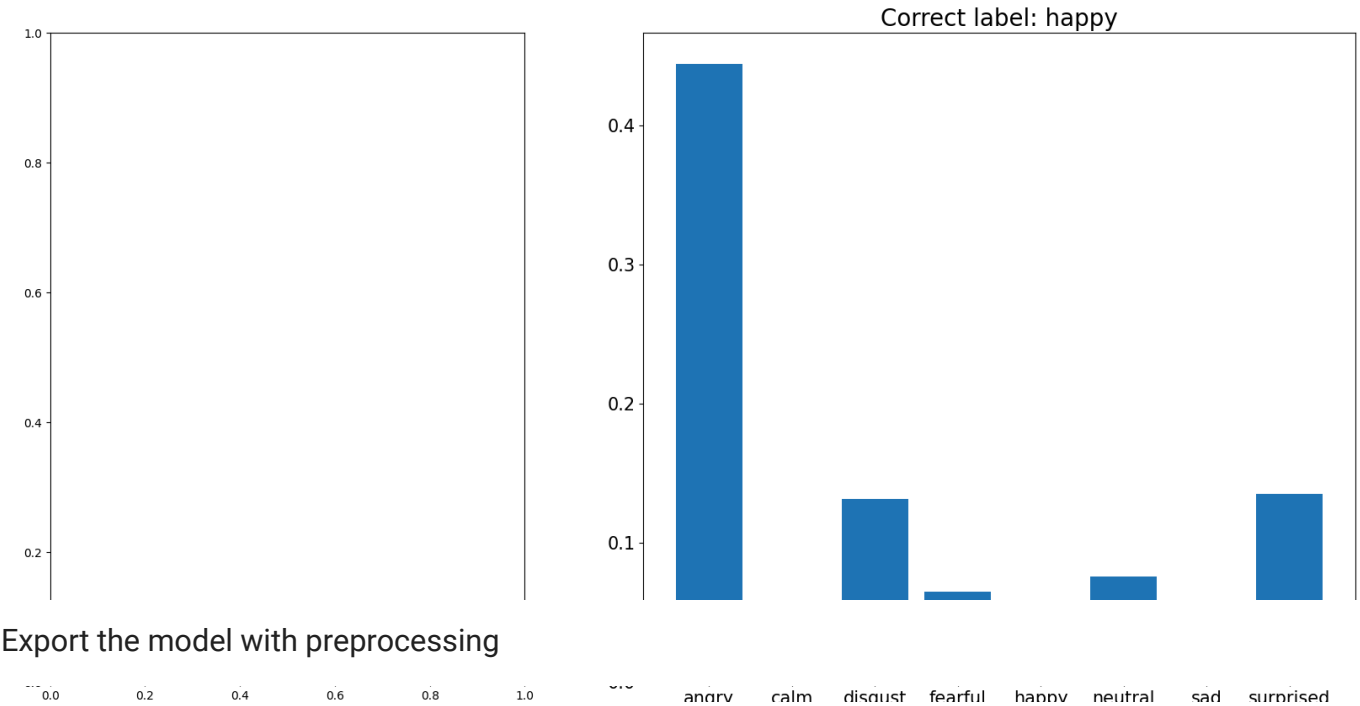


Run inference on an image file

Finally, verify the model's prediction output using an image

```
# Take the first item of each class in test_files
# path = Path(next((subs for subs in test_files if 'angry' in subs), None))
# path = Path(next((subs for subs in test_files if 'calm' in subs), None))
# path = Path(next((subs for subs in test_files if 'disgust' in subs), None))
# path = Path(next((subs for subs in test_files if 'fearful' in subs), None))
path = Path(next((subs for subs in test_files if 'happy' in subs), None))
# path = Path(next((subs for subs in test_files if 'neutral' in subs), None))
# path = Path(next((subs for subs in test_files if 'sad' in subs), None))
# path = Path(next((subs for subs in test_files if 'surprised' in subs), None))
# path = Path('Generated/Frames/disgust/01-01-07-01-01-01-16-049-01.jpg')
print(path)
# print(tf.io.read_file(str(path)))
label = path.parent.parent.name


image = cv2.imread(str(path))
# image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image = cv2.resize(image, (IMG_HEIGHT, IMG_WIDTH))
image = image / 255

net_input = utils.extend_tensor(image, 0)
prediction = model(net_input)
prediction = prediction[0].numpy()

valued_arr = []

for idx, name in enumerate(label_names):
    valued_arr.append(f'{name}\n({prediction[idx]:.2%})')

fig, ax = plt.subplots(
    nrows=1, ncols=2, width_ratios=[0.4, 0.6], figsize=(20, 10)
)

# pltDisplay(image * 255, ax=ax[0])  # 1 channel
# pltDisplay(image, ax=ax[0])  # 3 channels

ax[1].bar(valued_arr, prediction)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title(f'Correct label: {label}', fontsize=20)
# plt.xlabel('Predicted class')
# plt.ylabel('Percentage')
plt.show()
```

```
Generated/Frames/happy/21/01-01-03-02-01-02-21-091-01.jpg
```



Export the model with preprocessing

The model's not very easy to use if you have to apply those preprocessing steps before passing data to the model for inference. So build an end-to-end version:

```
class ExportModel(tf.Module):
    def __init__(self, model):
        self.model = model

```