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
```

Show hidden output

Double-click (or enter) to edit

```
!pip install mlflow databricks-sdk
```

```
Requirement already satisfied: mlflow in /usr/local/lib/python3.10/dist-packages (2.13.2)
Collecting databricks-sdk
  Downloading databricks_sdk-0.28.0-py3-none-any.whl (497 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 497.1/497.1 kB 5.1 MB/s eta 0:00:00
Requirement already satisfied: Flask<4 in /usr/local/lib/python3.10/dist-packages (from mlflow) (2.2.5)
Requirement already satisfied: alembic!=1.10.0,<2 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.13.1)
Requirement already satisfied: cachetools<6,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (5.3.3)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (8.1.7)
Requirement already satisfied: cloudpickle<4 in /usr/local/lib/python3.10/dist-packages (from mlflow) (2.2.1)
Requirement already satisfied: docker<8,>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (7.1.0)
Requirement already satisfied: entrypoints<1 in /usr/local/lib/python3.10/dist-packages (from mlflow) (0.4)
Requirement already satisfied: gitpython<4,>=3.1.9 in /usr/local/lib/python3.10/dist-packages (from mlflow) (3.1.43)
Requirement already satisfied: graphene<4 in /usr/local/lib/python3.10/dist-packages (from mlflow) (3.3)
Requirement already satisfied: importlib-metadata!=4.7.0,<8,>=3.7.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (7.1
Requirement already satisfied: markdown<4,>=3.3 in /usr/local/lib/python3.10/dist-packages (from mlflow) (3.6)
Requirement already satisfied: matplotlib<4 in /usr/local/lib/python3.10/dist-packages (from mlflow) (3.7.1)
Requirement already satisfied: numpy<2 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.25.2)
Requirement already satisfied: opentelemetry-api<3,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.25.0)
Requirement already satisfied: opentelemetry-sdk<3,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.25.0)
Requirement already satisfied: packaging<25 in /usr/local/lib/python3.10/dist-packages (from mlflow) (24.1)
Requirement already satisfied: pandas<3 in /usr/local/lib/python3.10/dist-packages (from mlflow) (2.0.3)
Requirement already satisfied: protobuf<5,>=3.12.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (3.20.3)
Requirement already satisfied: pyarrow<16,>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (14.0.2)
Requirement already satisfied: pytz<2025 in /usr/local/lib/python3.10/dist-packages (from mlflow) (2023.4)
Requirement already satisfied: pyyaml<7,>=5.1 in /usr/local/lib/python3.10/dist-packages (from mlflow) (6.0.1)
Requirement already satisfied: querystring-parser<2 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.2.4)
Requirement already satisfied: requests<3,>=2.17.3 in /usr/local/lib/python3.10/dist-packages (from mlflow) (2.31.0)
Requirement already satisfied: scikit-learn<2 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.2.2)
Requirement already satisfied: scipy<2 in /usr/local/lib/python3.10/dist-packages (from mlflow) (1.11.4)
Requirement already satisfied: sqlalchemy<3,>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (2.0.30)
Requirement already satisfied: sqlparse<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from mlflow) (0.5.0)
Requirement already satisfied: Jinja2<4,>=2.11 in /usr/local/lib/python3.10/dist-packages (from mlflow) (3.1.4)
Requirement already satisfied: gunicorn<23 in /usr/local/lib/python3.10/dist-packages (from mlflow) (22.0.0)
Requirement already satisfied: google-auth~=2.0 in /usr/local/lib/python3.10/dist-packages (from databricks-sdk) (2.27.0)
Requirement already satisfied: Mako in /usr/local/lib/python3.10/dist-packages (from alembic!=1.10.0,<2->mlflow) (1.3.5)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic!=1.10.0,<2->mlflow) (
Requirement already satisfied: urllib3>=1.26.0 in /usr/local/lib/python3.10/dist-packages (from docker<8,>=4.0.0->mlflow) (2.0.7)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from Flask<4->mlflow) (3.0.3)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask<4->mlflow) (2.2.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-packages (from gitpython<4,>=3.1.9->mlflow) (4.0
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth~=2.0->databricks
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth~=2.0->databricks-sdk) (4
Requirement already satisfied: graphql-core<3.3,>=3.1 in /usr/local/lib/python3.10/dist-packages (from graphene<4->mlflow) (3.2.3)
Requirement already satisfied: graphql-relay<3.3,>=3.1 in /usr/local/lib/python3.10/dist-packages (from graphene<4->mlflow) (3.2.0
Requirement already satisfied: aniso8601<10,>=8 in /usr/local/lib/python3.10/dist-packages (from graphene<4->mlflow) (9.0.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata!=4.7.0,<8,>=3.7.0->ml
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2<4,>=2.11->mlflow) (2.1.5)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4->mlflow) (2.8.2)
```

```
Requirement already satisfied: deprecated>=1.2.6 in /usr/local/lib/python3.10/dist-packages (from opentelemetry-api<3,>=1.0.0->mlf
Requirement already satisfied: opentelemetry-semantic-conventions==0.46b0 in /usr/local/lib/python3.10/dist-packages (from opentel
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3->mlflow) (2024.1)
```

```python
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, GlobalAveragePooling2D
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 import EfficientNetB0
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input as preprocess_imagenet
# from keras_vggface.utils import preprocess_input as preprocess_vggface

import mlflow
import gc
```

```python
# Set mlflow parameters
mlflow_params = {
    'local': False,
    'host': '127.0.0.1',
    'port': '8080'
}
if (mlflow_params['local']):
    # start server
    mlflow_server_process = subprocess.Popen(
        f"mlflow server --host {mlflow_params['host']} --port {mlflow_params['port']}",
        shell=False, stdout=subprocess.PIPE, stderr=subprocess.PIPE
    )
    mlflow.set_tracking_uri(uri="http://127.0.0.1:8080")
    mlflow.set_experiment("Frames")
else:
    # https://www.mlflow.org/docs/latest/getting-started/tracking-server-overview/index.html method 2
    # Host: https://community.cloud.databricks.com/
    mlflow.login()
    mlflow.set_tracking_uri(uri="databricks")
    mlflow.set_experiment("/Frames")

mlflow.tensorflow.autolog(
    log_models=False,
    log_datasets=True,
    log_input_examples=True,
    log_model_signatures=True,
    checkpoint=False
)
```

```
2024/06/13 09:41:23 INFO mlflow.utils.credentials: No valid Databricks credentials found, please enter your credentials...
Databricks Host (should begin with https://): https://community.cloud.databricks.com/
Username: micasadmail@gmail.com
Password: ··········
2024/06/13 09:42:50 INFO mlflow.utils.credentials: Successfully connected to MLflow hosted tracking server! Host: https://community
```

```
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 @ 2.20GHz (406F0),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 = 16
VALIDATION_SPLIT = 0.2
EMOTIONS_LABELS = const.EMOTIONS_LABELS  # RAVDESS emotion labels
```

```
TOTAL_ELEMENTS = const.DATASET_TOTAL_ELEMENTS
```

```
label_names = const.EMOTIONS_LABELS_SORTED.copy()
# label_names = ['happy', 'sad']
label_names_gender = []
for em in label_names:
    label_names_gender.append(em + '_female')
    label_names_gender.append(em + '_male')
```

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)]
}
```

```
# 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
```

```
def preprocess_image(image, preprocess_type=None):
    if (preprocess_type == 'Imagenet'):
        image = preprocess_imagenet(image)
    elif (preprocess_type == 'VGGFace'):
        image = preprocess_vggface(image, version=1)
    else:
        image = image / 255

    return image
```

```
def prep_external_image(filename, preprocess_type=None):
    image = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image, channels=IMG_CHANNELS, fancy_upscaling=False)
    image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH])
    return image
    # return preprocess_image(image, preprocess_type)
```

```
def make_dataset(path, actors_idx, talk_frame=False, acted_frame=False,
                 undersampling=False, preprocess_vgg=True, shuffle=False,
                 gender_classes=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 == 'Imagenet'):
            image = preprocess_imagenet(image)
        elif (preprocess_vgg == 'VGGFace'):
            image = preprocess_vggface(image, version=1)

        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()
    file_dict_gender = dict()

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

    for label in sorted(label_names_gender):
        file_dict_gender[label] = []


    for label in label_names:
        for actor in actors_idx:
            for file in Path(path, label, actor).glob(f'{gen_regex}'):
                gender = utils.get_gender(str(file))[1]
                lab = label + f'_{gender}'
                file_dict_gender[lab].append(str(file))
                file_dict[label].append(str(file))

    # if shuffle:
    #     for label, item in file_dict_gender.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 = []

    if (gender_classes):

        for elem in filenames:
            cl = utils.get_class_string(str(elem))
            gender = utils.get_gender(str(elem))[1]
            lab = cl + f'_{gender}'
            labels.append(label_names_gender.index(lab))
    else:
        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))

    return [ds, filenames]
```

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

if (acted_frame):
    print('L\'emozione neutral verrà rimossa')
    if 'neutral' in label_names:
        label_names.remove('neutral')
    if 'neutral_female' in label_names_gender:
        label_names_gender.remove('neutral_female')
    if 'neutral_male' in label_names_gender:
        label_names_gender.remove('neutral_male')

NUM_CLASSES = len(label_names_gender) if gender_classes else len(label_names)

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, gender_classes=gender_classes,
    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, gender_classes=gender_classes,
    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, gender_classes=gender_classes,
    sampling=sampling_rate
)

labels = label_names if not gender_classes else label_names_gender
```

```
# 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.

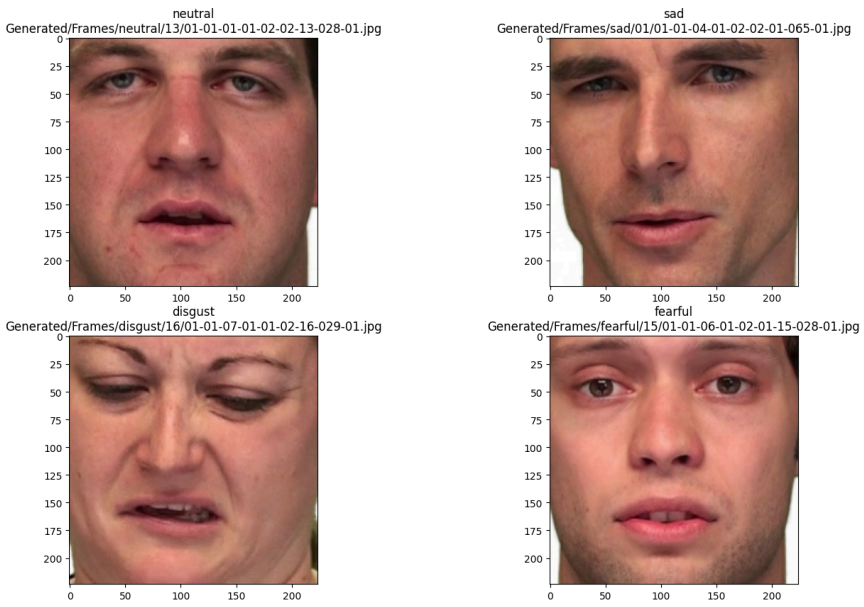
```
def configure_for_performance(ds, batch_size=BATCH_SIZE):
    ds = ds.batch(batch_size)
    # ds = ds.cache()
    # ds = ds.shuffle(buffer_size=1000)
    # ds = ds.repeat()
    ds = ds.prefetch(buffer_size=tf.data.AUTOTUNE)
    return ds
```

```
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)
```

```
(16, 224, 224, 3)
(16,)
```

```
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]}')
```



```python
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
```

```python
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
```

```
def create_EfficientNetB0_Imagenet(name='EfficientNetB0_Imagenet'):

    nb_class = NUM_CLASSES

    inputs = Input(shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS))
    model = EfficientNetB0(
        include_top=False, input_tensor=inputs, weights="imagenet"
    )

    # Freeze the pretrained weights
    model.trainable = False

    # Rebuild top
    x = GlobalAveragePooling2D(name="avg_pool")(model.output)
    x = BatchNormalization()(x)

    top_dropout_rate = 0.2
    x = Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = Dense(nb_class, activation="softmax", name="pred")(x)

    model._name = name
    # Compile
    model = Model(inputs, outputs, name="EfficientNet")

    return model
```

```
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
```

```
NUM_CLASSES
```

```
8
```

```
def create_dog_vs_cat_model(name='dog_cat'):
    vggmodel = VGG16(weights='imagenet', include_top=True)
    for layers in (vggmodel.layers)[:19]:
        layers.trainable = True

    X = vggmodel.layers[-2].output
    # X = Dense(256, activation ='relu')(X)
    # predictions = Dense(1, activation="sigmoid")(X)
    predictions = Dense(NUM_CLASSES, activation="softmax")(X)
    model_final = Model(vggmodel.input, predictions)

    model_final._name = name

    return model_final
```

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

```
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'
)
```

```
# model = create_cnn_model()
# model = medium_model()
# model = create_VGG16_Imagenet()
# model = create_VGG16_VGGFACE()
# model = create_EfficientNetB0_Imagenet()
# model = create_grigorasi_model()
# model = create_Bilotti_CNN()
model = create_dog_vs_cat_model()
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels
553467096/553467096 [==============================] - 6s 0us/step
Model: "dog_cat"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 dense (Dense)               (None, 8)                 32776

=================================================================
Total params: 134293320 (512.29 MB)
Trainable params: 134293320 (512.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# 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(
    monitor='val_loss',
    verbose=1,
    patience=3,
    restore_best_weights=True
)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, verbose=1,
                              patience=7, min_lr=0)
```

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

```
# Model compile dog vs cat
from keras import optimizers
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=optimizers.SGD(learning_rate=0.00001, momentum=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 = 200
time_callback = TimeHistory()
history = model.fit(
    x=train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=[time_callback, early_stopping_callback, checkpointer]
)
```

```
2024/06/13 09:49:15 INFO mlflow.utils.autologging_utils: Created MLflow autologging r
2024/06/13 09:51:38 INFO mlflow.types.utils: MLflow 2.9.0 introduces model signature
2024/06/13 09:52:04 INFO mlflow.types.utils: MLflow 2.9.0 introduces model signature
Epoch 1/200
4245/4245 [==============================] - ETA: 0s - loss: 0.4982 - accuracy: 0.818
Epoch 1: val_accuracy improved from -inf to 0.51581, saving model to checkpoint.weigh
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarnir
  saving_api.save_model(
4245/4245 [==============================] - 1183s 274ms/step - loss: 0.4982 - accura
Epoch 2/200
4245/4245 [==============================] - ETA: 0s - loss: 0.0885 - accuracy: 0.970
Epoch 2: val_accuracy improved from 0.51581 to 0.54535, saving model to checkpoint.we
4245/4245 [==============================] - 1146s 270ms/step - loss: 0.0885 - accura
Epoch 3/200
4245/4245 [==============================] - ETA: 0s - loss: 0.0520 - accuracy: 0.983
Epoch 3: val_accuracy did not improve from 0.54535
4245/4245 [==============================] - 1135s 267ms/step - loss: 0.0520 - accura
Epoch 4/200
4245/4245 [==============================] - ETA: 0s - loss: 0.0399 - accuracy: 0.987

Epoch 4: val_accuracy did not improve from 0.54535
4245/4245 [==============================] - 1134s 267ms/step - loss: 0.0399 - accura
Epoch 4: early stopping
Uploading artifacts: 100%                                    2/2 [00:00<00:00,  2.03it/s]
```

```
# # FINE TUNING
# # unfreeze base model
# vgg_model.trainable = True
# model.summary()

# model.compile(
#     optimizer=tf.keras.optimizers.Adam(
#         learning_rate=1e-5
#     ),
#     # optimizer=tf.keras.optimizers.SGD(),
#     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
#     metrics=METRICS,
# )
# EPOCHS = 2
# time_callback = TimeHistory()
# history = model.fit(
#     x=train_ds,
#     validation_data=val_ds,
#     epochs=EPOCHS,
#     callbacks=[time_callback, early_stopping_callback, reduce_lr]
# )
```

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

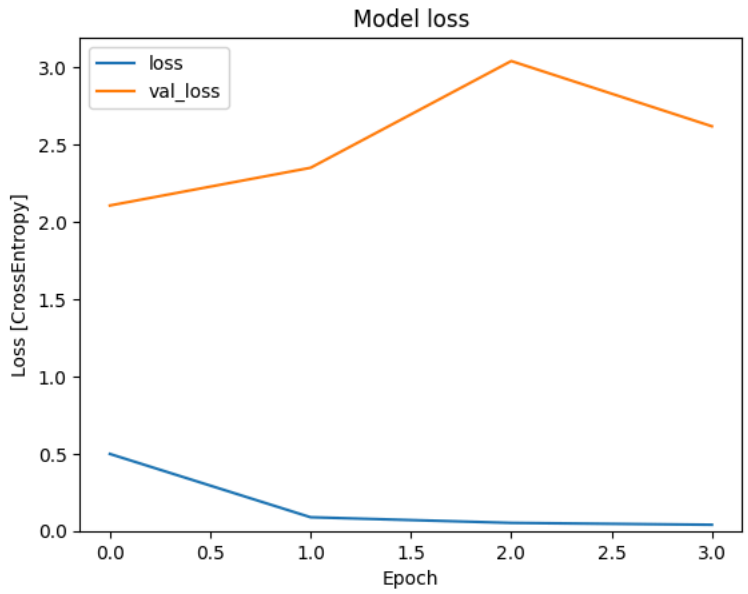
```
# Create model path
model_path = Path(const.models_path, 'Frame', model._name)
run_folders = list(Path(const.models_path, 'Frame', model._name).glob('Run_*'))

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')
```

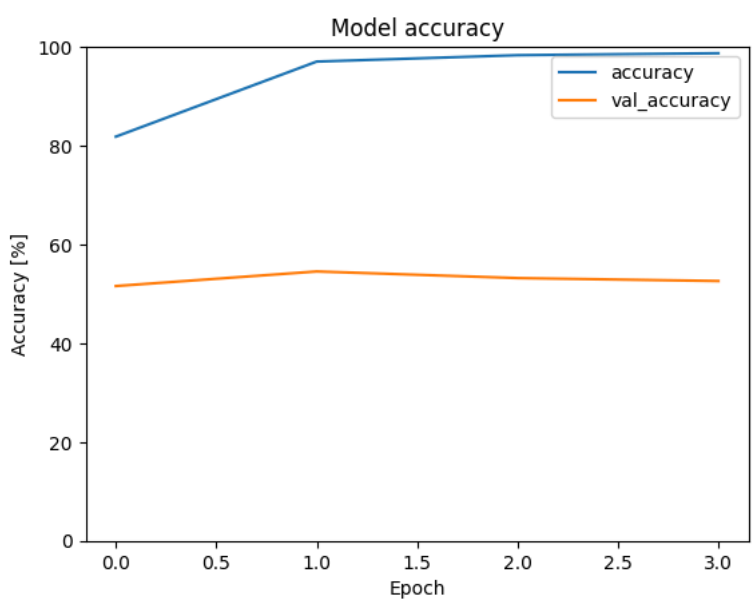
```
metrics = history.history
```

```
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)
```

```
1125/1125 [==============================] - 80s 71ms/step - loss: 2.2596 - accuracy: 0.4440
```

```
hist_df = pd.DataFrame(metrics)
hist_csv_file = Path(model_path, 'history.csv')
with open(hist_csv_file, mode='w') as f:
    hist_df.to_csv(f)
```

```
# 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': 2.259594678878784, 'accuracy': 0.4439813196659088}
```

```
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)}')
```

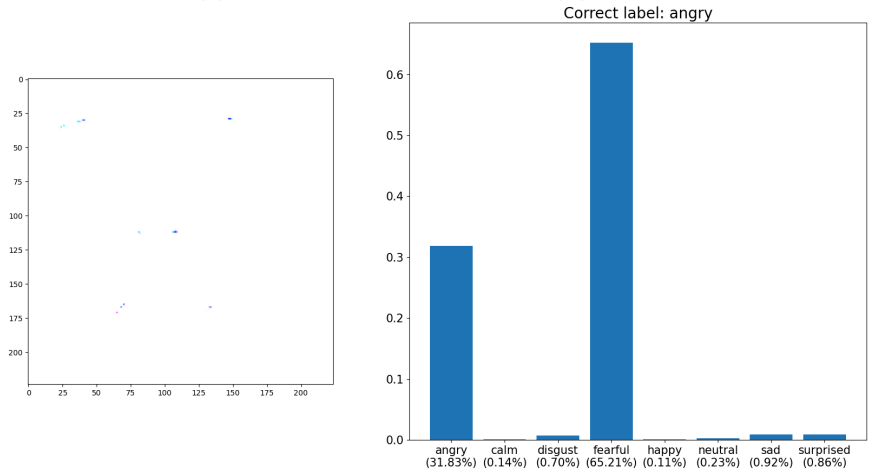
```
# Salvataggio informazioni generali modelli
with open(Path(const.models_path, 'Frame', 'models.csv'), 'a+') as csvfile:
    filewriter = csv.writer(
        csvfile, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL
    )

    # 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)
```

```
(16, 224, 224, 3)
(16,)
```

```
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/21/01-01-05-01-02-01-21-062-01.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)
```

```
1125/1125 [==============================] - 80s 70ms/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)
```

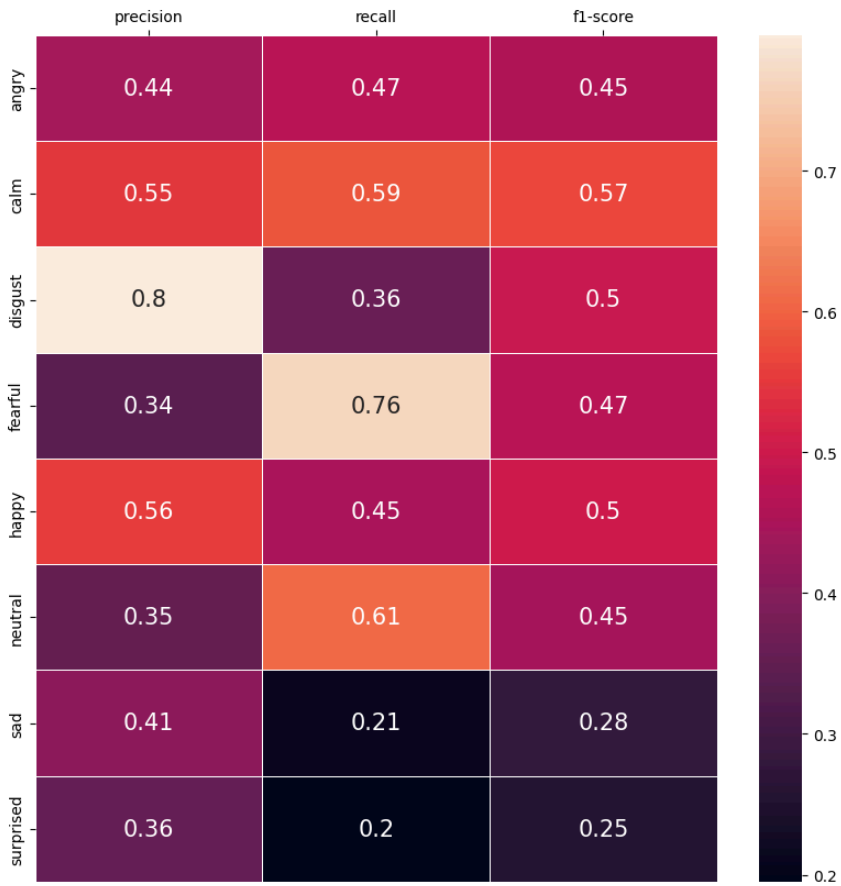
```
label_names
```

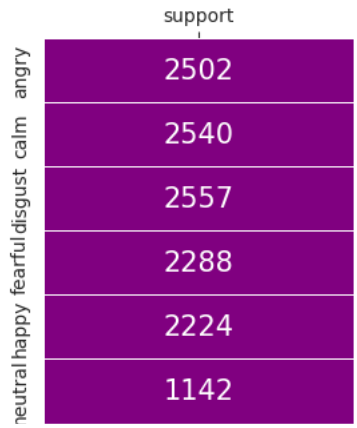
```
['angry', 'calm', 'disgust', 'fearful', 'happy', 'neutral', 'sad', 'surprised']
```

```
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()
```





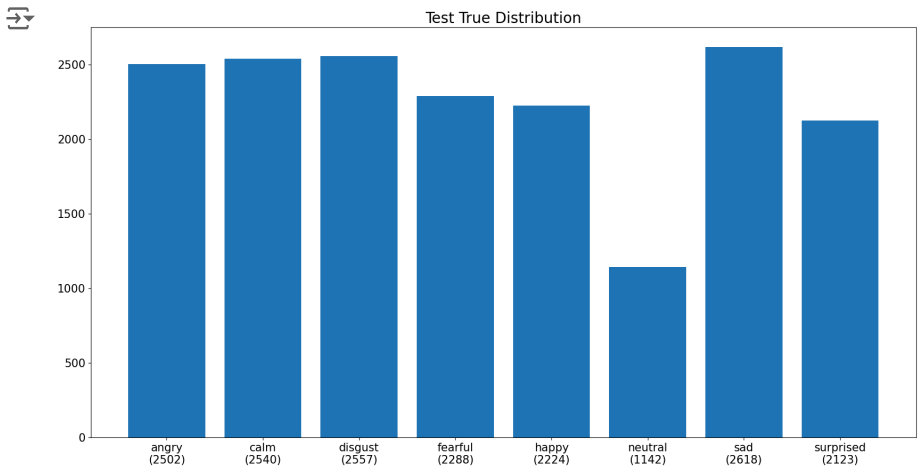

```
report_save_path = Path(model_path, f'{model._name}_report.csv')
rep_to_csv.to_csv(report_save_path)
```

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

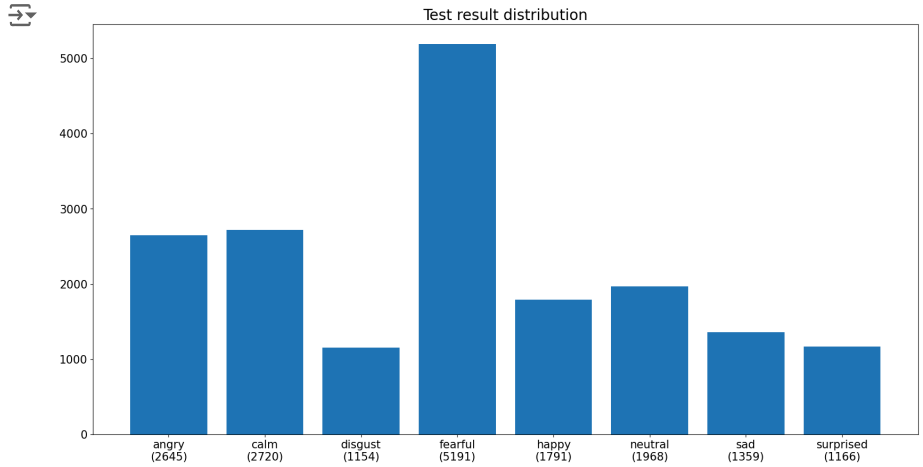
```
np.mean([report[c]['f1-score'] for c in list(report)[:NUM_CLASSES]])
```

```
0.4322732858116839
```

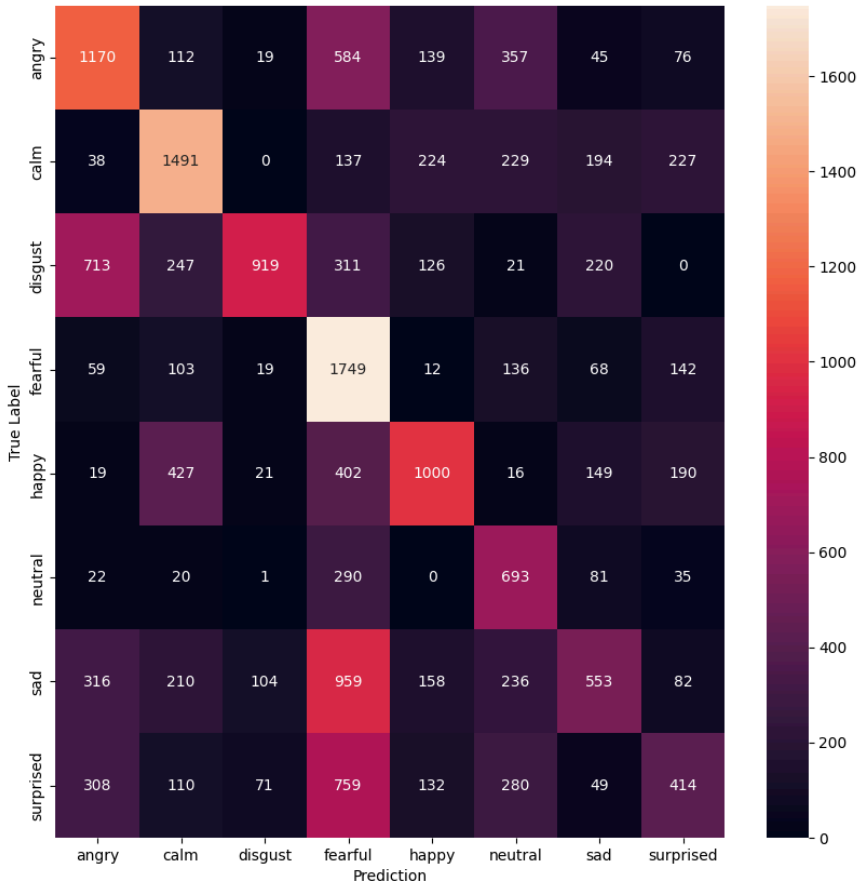
```
# True Test Distribution
# unique, counts = np.unique(y_true, return_counts=True)
# collections.Counter(y_true)
counts = [np.count_nonzero(y_true == idx) for idx in range(len(label_names))]
valued_arr = []
# for i in range(len(label_names)):
for idx, name in enumerate(label_names):
    count = counts[idx]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

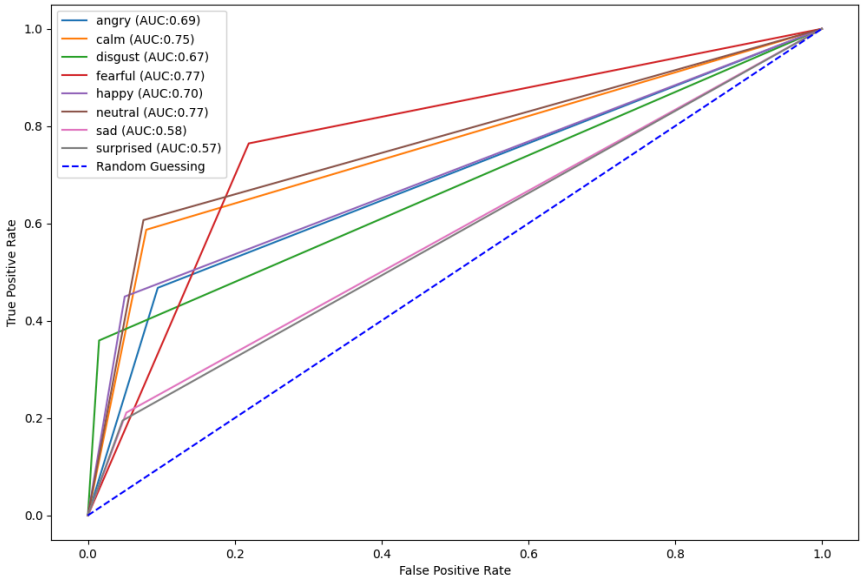



```python
def multiclass_roc_auc_score(target, y_test, y_pred, average="macro"):
    # function for scoring roc auc score for multi-class
    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)

    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)
```

```python
# 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: 0.6879024625140258
```



Run inference on an image file

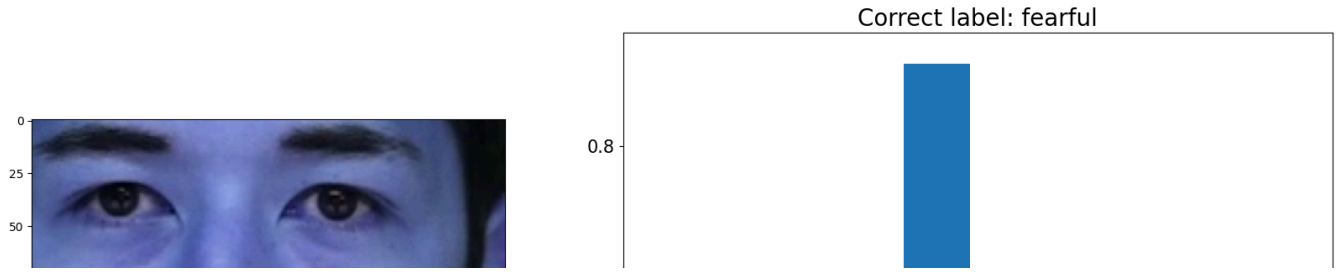
Finally, verify the model's prediction output using an image

```
labels
```

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

```
# 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
# # image = preprocess_input(image)
image = prep_external_image(str(path), preprocess_vgg)
net_input = utils.extend_tensor(image, 0)
prediction = model(net_input)
prediction = prediction[0].numpy()

valued_arr = []

for idx, name in enumerate(labels):
    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 / 255, 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/fearful/21/01-01-06-01-02-01-21-086-01.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: