Multi-label image classifier for images of musicians

Based on: https://www.tensorflow.org/tutorials/images/classification

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

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

Imports

```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras import utils
from keras import preprocessing

import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import pathlib
```

```
# Load the images
data_dir = pathlib.Path('drive/MyDrive/Artists')
data_dir
```

```
PosixPath('drive/MyDrive/Artists')
```

```
michael_jackson = list(data_dir.glob('michael-jackson/*'))
print(len(michael_jackson))
```

```
50
```

```
PIL.Image.open(str(michael_jackson[0]))
```

Data Preprocessing

```
BATCH_SIZE = 10
IMG_HEIGHT = 180
IMG_WIDTH = 180

EPOCHS = 100

train_dataset = preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE)


class_names = train_dataset.class_names
NUM_CLASSES = len(class_names)
```

```
Found 250 files belonging to 4 classes.
Using 200 files for training.
```

```
val_dataset = preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE)
```

```
Found 250 files belonging to 4 classes.
Using 50 files for validation.
```

```
for image_batch, labels_batch in train_dataset:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(10, 180, 180, 3)
(10,)
```

```
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.cache().shuffle(10).prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.cache().shuffle(10).prefetch(buffer_size=AUTOTUNE)


# Normalize
# normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)


# Build the model
print(f'Building a {NUM_CLASSES}-label classifier.')

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(NUM_CLASSES)
])

model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])

model.summary()
```

```
Building a 4-label classifier.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
```

```
conv2d_6 (Conv2D)            (None, 22, 22, 128)       73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 11, 11, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 15488)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1982592
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 516
=================================================================
Total params: 2,080,548
Trainable params: 2,080,548
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(train_dataset, validation_data=val_dataset, epochs=200)
```

```
Epoch 172/200
20/20 [==============================] - 0s 10ms/step - loss: 1.3709e-07 - accuracy:
Epoch 173/200
20/20 [==============================] - 0s 10ms/step - loss: 1.3828e-07 - accuracy:
Epoch 174/200
20/20 [==============================] - 0s 10ms/step - loss: 1.3351e-07 - accuracy:
Epoch 175/200
20/20 [==============================] - 0s 9ms/step - loss: 1.3292e-07 - accuracy:
Epoch 176/200
20/20 [==============================] - 0s 10ms/step - loss: 1.2934e-07 - accuracy:
Epoch 177/200
20/20 [==============================] - 0s 9ms/step - loss: 1.2875e-07 - accuracy:
Epoch 178/200
20/20 [==============================] - 0s 9ms/step - loss: 1.2815e-07 - accuracy:
Epoch 179/200
20/20 [==============================] - 0s 9ms/step - loss: 1.2219e-07 - accuracy:
Epoch 180/200
20/20 [==============================] - 0s 10ms/step - loss: 1.2457e-07 - accuracy:
Epoch 181/200
20/20 [==============================] - 0s 10ms/step - loss: 1.2219e-07 - accuracy:
Epoch 182/200
20/20 [==============================] - 0s 9ms/step - loss: 1.1861e-07 - accuracy:
Epoch 183/200
20/20 [==============================] - 0s 10ms/step - loss: 1.1563e-07 - accuracy:
Epoch 184/200
20/20 [==============================] - 0s 9ms/step - loss: 1.1683e-07 - accuracy:
Epoch 185/200
20/20 [==============================] - 0s 10ms/step - loss: 1.1504e-07 - accuracy:
Epoch 186/200
20/20 [==============================] - 0s 10ms/step - loss: 1.1325e-07 - accuracy:
Epoch 187/200
20/20 [==============================] - 0s 10ms/step - loss: 1.1027e-07 - accuracy:
Epoch 188/200
20/20 [==============================] - 0s 10ms/step - loss: 1.1206e-07 - accuracy:
Epoch 189/200
20/20 [==============================] - 0s 10ms/step - loss: 1.0848e-07 - accuracy:
Epoch 190/200
20/20 [==============================] - 0s 10ms/step - loss: 1.0848e-07 - accuracy:
Epoch 191/200
20/20 [==============================] - 0s 10ms/step - loss: 1.0669e-07 - accuracy:
```

```
Epoch 192/200
20/20 [==============================] - 0s 10ms/step - loss: 1.0431e-07 - accuracy:
Epoch 193/200
20/20 [==============================] - 0s 10ms/step - loss: 1.0490e-07 - accuracy:
Epoch 194/200
20/20 [==============================] - 0s 9ms/step - loss: 1.0312e-07 - accuracy:
Epoch 195/200
20/20 [==============================] - 0s 10ms/step - loss: 1.0133e-07 - accuracy:
Epoch 196/200
20/20 [==============================] - 0s 10ms/step - loss: 9.7752e-08 - accuracy:
Epoch 197/200
20/20 [==============================] - 0s 10ms/step - loss: 9.8944e-08 - accuracy:
Epoch 198/200
20/20 [==============================] - 0s 9ms/step - loss: 9.9540e-08 - accuracy:
Epoch 199/200
20/20 [==============================] - 0s 10ms/step - loss: 9.5963e-08 - accuracy:
Epoch 200/200
20/20 [==============================] - 0s 10ms/step - loss: 9.6560e-08 - accuracy:
```

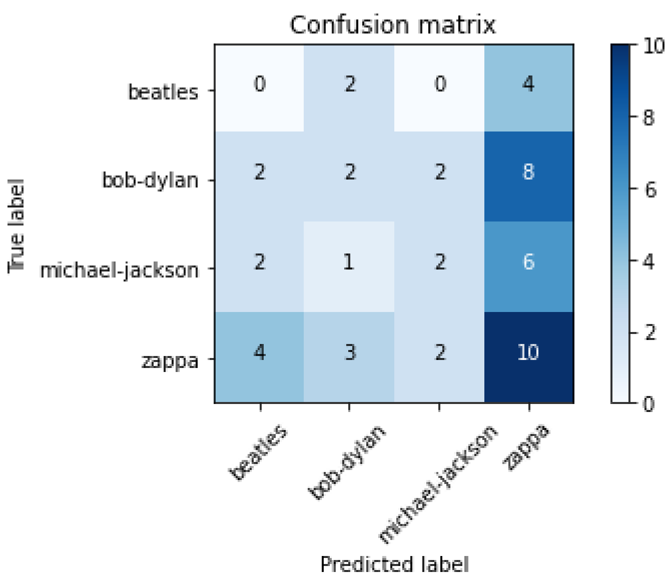
```
# Get confusion matrix on the val dataset
y_pred = model.predict(val_dataset)


# Build confusion matrix
predicted_categories = np.argmax(y_pred, axis=1)
actual_categories = tf.concat([y for x, y in val_dataset], axis=0)

print(f'Predicted: {predicted_categories}')
print(f'Actual: {actual_categories}')
```

```
Predicted: [2 0 3 3 3 1 3 0 3 3 3 3 3 3 3 3 3 3 2 1 3 1 3 3 0 3 3 3 1 0 1 2 3 2 0 3
 1 3 3 1 0 2 2 1 3 0 3 0 3]
Actual: [3 2 1 1 3 0 3 3 3 1 0 1 2 3 2 1 3 2 3 1 2 0 3 3 2 2 2 1 1 1 1 0 2 2 3 3 0
 3 0 3 1 1 2 1 3 1 3 3 3 3]
```

```
cm = tf.math.confusion_matrix(actual_categories, predicted_categories, num_classes=4)


# This code is from:
# https://scikit-learn.org/0.18/auto_examples/model_selection/plot_confusion_matrix.html
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
```

```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    print(cm)

    cm = cm.numpy()
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


plt.figure()
plot_confusion_matrix(cm, classes=class_names, normalize=True, title='Confusion matrix')

plt.show()
```

```
tf.Tensor(
[[ 0  2  0  4]
 [ 2  2  2  8]
 [ 2  1  2  6]
 [ 4  3  2 10]], shape=(4, 4), dtype=int32)
```




```
def predict(model, url):
  predict_url = url
  predict_path = tf.keras.utils.get_file('predict', origin=predict_url)
  print(predict_path)
```

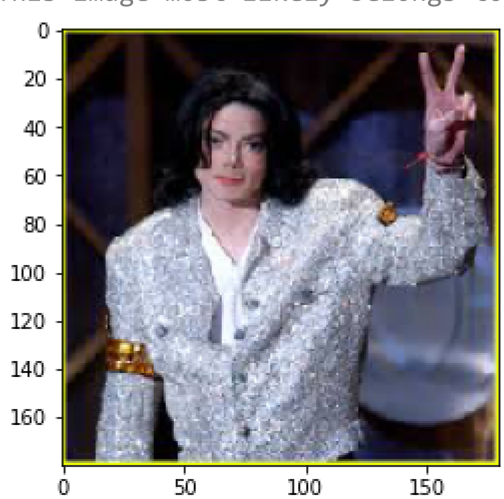
```
  img = keras.preprocessing.image.load_img(
    predict_path, target_size=(IMG_HEIGHT, IMG_WIDTH)
  )

  img_array = []
  img_array = keras.preprocessing.image.img_to_array(img)
  img_array = tf.expand_dims(img_array, 0) # Create a batch

  predictions = model.predict(img_array)
  score = tf.nn.softmax(predictions[0])

  print(
      "This image most likely belongs to {} with a {:.2f} percent confidence."
      .format(class_names[np.argmax(score)], 100 * np.max(score))
  )

  plt.imshow(img)
```

```
sample_urls = [
  "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSjKvc7_8SLeHlsOrLT6-LpnptoOOLDmCo_kg
  "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQCQ6Fk1AbBcRczijclpOKswAm3te5vG7Mfng
  "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTt3tTCLgunkc4BAP4PMsWhUxaGkSxtA6BMFw
  "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTcsXPkjxp4FMKniRZKlewCz3OjYrML5sfDJg
]
```

```
# model.save(f'saved_models/classifier')
```

```
re_model = keras.models.load_model(f'saved_models/classifier')
predict(re_model, sample_urls[1])
```

```
/root/.keras/datasets/predict
This image most likely belongs to zappa with a 100.00 percent confidence.
```



0s completed at 10:17 PM