Tensorflow setup

In [ ]:

```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

# TensorFlow ≥2.0 is required
import tensorflow as tf
assert tf.__version__ >= "2.0"

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)


# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

Every import needed

In [ ]:

```
from tensorflow import keras
from keras import layers
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

import seaborn as sns

import os
import pathlib
import time

# Convert dataset to numpy function
from tensorflow.data import Dataset

from sklearn.metrics import precision_score
```

```
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from tensorflow.keras.utils import to_categorical
from itertools import cycle
from sklearn.metrics import auc

import pandas as pd
```

Load images

In [ ]:
```
PATH = '.\\..\\dataset\\micro_expressions'

train_dir = os.path.join(PATH, 'train')
test_dir = os.path.join(PATH, 'test')

print(train_dir)
print(test_dir)
```

```
.\..\dataset\micro_expressions\train
.\..\dataset\micro_expressions\test
```

In [ ]:
```
import pathlib

train_dir = pathlib.Path(train_dir)
test_dir = pathlib.Path(test_dir)

image_count = len(list(train_dir.glob('*/*.jpg')))
print(image_count)

image_count = len(list(test_dir.glob('*/*.jpg')))
print(image_count)
```

```
6938
1700
```

In [ ]:
```
batch_size = 32
IMG_SIZE = (80, 80)

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  train_dir,
  validation_split=0.15,
  subset="training",
  seed=123,
  shuffle=True,
  image_size=IMG_SIZE,
  batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  train_dir,
  validation_split=0.15,
  subset="validation",
  seed=123,
  shuffle=True,
  image_size=IMG_SIZE,
  batch_size=batch_size)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  test_dir,
  validation_split=None,
  seed=123,
```

```
  shuffle=True,
  image_size=IMG_SIZE,
  batch_size=batch_size)

class_names = train_ds.class_names

train_ds = train_ds.cache().prefetch(1)
val_ds = val_ds.cache().prefetch(1)
test_ds = test_ds.cache().prefetch(1)

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

train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))
```

```
Found 6938 files belonging to 7 classes.
Using 5898 files for training.
Found 6938 files belonging to 7 classes.
Using 1040 files for validation.
Found 1700 files belonging to 7 classes.
```

Util functions used in the code

In [ ]:

```
# Convert dataset to numpy
def batch_dataset_to_numpy(ds):
    in_data = []
    out_data = []
    for inp, out in ds:

        array = inp.numpy()
        for x in array:
            in_data.append(x)

        array = out.numpy()
        for x in array:
            out_data.append(x)

    return in_data, out_data


X_train, y_train = batch_dataset_to_numpy(train_ds)
X_val, y_val = batch_dataset_to_numpy(val_ds)
X_test, y_test = batch_dataset_to_numpy(test_ds)
```

In [ ]:

```
def print_metrics_by_class(array, title=None, is_rate=False):
    i = 0
    print(title)

    for k in class_names:
        if is_rate == True:
            print('\t%s -> %.3f%%' %(k, array[i]))
        else:
            print('\t%s -> %.0f' %(k, array[i]))

        i = i + 1

def print_confusion_matrix_metrics(cnf_matrix):
    FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TP = np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)

    FP = FP.astype(float)
    FN = FN.astype(float)
    TP = TP.astype(float)
```

```
    TN = TN.astype(float)

    print_metrics_by_class(FP, '\nFalse Positives:\n')
    print_metrics_by_class(FN, '\nFalse Negatives:\n')
    print_metrics_by_class(TP, '\nTrue Positives:\n')
    print_metrics_by_class(TN, '\nTrue Negatives:\n')

    # Sensitivity, hit rate, recall, or true positive rate
    TPR = TP/(TP+FN)
    print_metrics_by_class(TPR, '\nTrue Positive Rate:\n', is_rate=True)

    # Specificity or true negative rate
    TNR = TN/(TN+FP)
    print_metrics_by_class(TNR, '\nTrue Negative Rate:\n', is_rate=True)

    # Precision or positive predictive value
    PPV = TP/(TP+FP)
    print_metrics_by_class(PPV, '\nPositive Predictive Value:\n')

    # Negative predictive value
    NPV = TN/(TN+FN)
    print_metrics_by_class(NPV, '\nNegative Predictive Value:\n')

    # Fall out or false positive rate
    FPR = FP/(FP+TN)
    print_metrics_by_class(FPR, '\nFalse Positive Rate:\n', is_rate=True)

    # False negative rate
    FNR = FN/(TP+FN)
    print_metrics_by_class(FNR, '\nFalse Negative Rate:\n', is_rate=True)

    # False discovery rate
    FDR = FP/(TP+FP)
    print_metrics_by_class(FDR, '\nFalse Discovery Rate:\n', is_rate=True)

    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    print_metrics_by_class(ACC, '\nAccuracy:\n', is_rate=True)

def plot_confusion_matrix(y_pred, y_test, labels, normalize_m=None, return_m=False):

    fig, ax = plt.subplots(figsize=(10,10))
    matrix = confusion_matrix(y_test, tf.argmax(y_pred, axis=-1), normalize=normalize_m)

    fmt_m = '2'
    if normalize_m != None:
        fmt_m = '.3f'

    sns.heatmap(
        matrix,
        annot=True,
        fmt=fmt_m,
        cmap='Blues',
        cbar=False,
        xticklabels=class_names,
        yticklabels=class_names,
        ax=ax
    )

    plt.title('Confusion Matrix', fontsize = 20)
    plt.ylabel('Real Label', fontsize = 15)
    plt.xlabel('Predicted Label', fontsize = 15)
    plt.show()

    if return_m == True:
        return matrix

def plot_loss_and_accuracy(history):
```

```
    history_data_frame = pd.DataFrame(history.history)

    print(history_data_frame.loc[:, [ 'accuracy']].max(), history_data_frame.loc[:, [ 'val_

    plt.figure(figsize=(15,10))
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, linestyle='-', marker='.', color='b', label="Training loss")
    plt.plot(epochs, val_loss, linestyle='-', marker='.', color='r', label="Validation loss
    plt.title("Training and validation loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.grid(True)
    plt.show();

    plt.clf()
    plt.figure(figsize=(15,10))
    acc = history.history["accuracy"]
    val_acc = history.history["val_accuracy"]
    plt.plot(epochs, acc, linestyle='-', marker='.', color='b', label="Training accuracy")
    plt.plot(epochs, val_acc, linestyle='-', marker='.', color='r', label="Validation accur
    plt.title("Training and validation accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)
    plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]
    plt.show()

def print_dataset_class_balance(class_names, y, title=None):
    print(title)
    y_np = np.array(y)
    length = len(y_np)

    for i, x  in enumerate(class_names):
        count = len(y_np[y_np == i])
        print(f" {class_names[i]} = {(count/length):.2f} %")

    print('\n')


def batch_dataset_to_numpy(ds):
    in_data = []
    out_data = []
    for inp, out in ds:

        array = inp.numpy()
        for x in array:
            in_data.append(x)

        array = out.numpy()
        for x in array:
            out_data.append(x)

    return in_data, out_data

def prinf_dataset_information():
    # Cardinality (number of batches)
    print('Train Cardinality (number of batches): ', train_ds.cardinality().numpy())
    print('Validation cardinality (number of batches): ', val_ds.cardinality().numpy())
    print('Test cardinality (number of batches): ', test_ds.cardinality().numpy(), end='\n'

    print('Class names: ', class_names, end='\n\n')

    print('Shape of X_train: ', np.array(X_train).shape)
```

```
    print('Shape of y_train: ', np.array(y_train).shape, end='\n\n')
    print('Shape of X_val: ', np.array(X_val).shape)
    print('Shape of y_val: ', np.array(y_val).shape, end='\n\n')
    print('Shape of X_test: ', np.array(X_test).shape)
    print('Shape of y_test: ', np.array(y_test).shape, end='\n\n')

    print_dataset_class_balance(class_names, y_train, 'Class Balance - Train Data:\n')
    print_dataset_class_balance(class_names, y_val, 'Class Balance - Validation Data:\n')
    print_dataset_class_balance(class_names, y_test, 'Class Balance - Test Data:\n')
```

In [ ]:

```
#https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-

graph_colors = ['aqua', 'darkorange', 'cornflowerblue', 'green', 'red', 'blue', 'grey']

def plot_multiclass_roc_curve(y_test, y_pred, class_names, single_class = None, title = Nor

    if title == None:
        title = ''

    # Binarize the output
    y_b = to_categorical(y_test, num_classes = len(class_names))
    yp = y_pred

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    thresh = dict()
    roc_auc = dict()

    for i in range(len(class_names)):
        fpr[str(i)], tpr[str(i)], thresh[str(i)] = roc_curve(y_b[:, i], yp[:, i])
        roc_auc[str(i)] = auc(fpr[str(i)], tpr[str(i)])

    lw = 2 # linewidth

    # Compute micro-average ROC curve and ROC area
    fpr['micro'], tpr['micro'], thresh['micro'] = roc_curve(y_b.ravel(), yp.ravel())
    roc_auc['micro'] = auc(fpr['micro'], tpr['micro'])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[str(i)] for i in range(len(class_names))]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(len(class_names)):
        mean_tpr += np.interp(all_fpr, fpr[str(i)], tpr[str(i)])

    # Finally average it and compute AUC
    mean_tpr /= len(class_names)

    fpr['macro'] = all_fpr
    tpr['macro'] = mean_tpr
    roc_auc['macro'] = auc(fpr['macro'], tpr['macro'])

    # Plot all ROC curves
    plt.figure(figsize=(15,15))
    plt.plot(
        fpr['micro'],
        tpr['micro'],
        label = 'Micro-Avg (area = {0:0.2f})'.format(roc_auc['micro']),
        color = 'deeppink',
        linestyle = '--',
        linewidth = lw
    );

    plt.plot(
```

```
        fpr['macro'],
        tpr['macro'],
        label = 'Macro-Avg (area = {0:0.2f})'.format(roc_auc['macro']),
        color = 'navy',
        linestyle = '--',
        linewidth = lw
    );

    colors = cycle(graph_colors)
    for i, color in zip(range(len(class_names)), colors):
        if single_class == None:
            plt.plot(
                fpr[str(i)],
                tpr[str(i)],
                color = color,
                lw = lw,
                label = 'ROC - "{0}" (area = {1:0.2f})'.format(class_names[i].capitalize(),
            );
        else:
            if single_class == i:
                plt.plot(
                    fpr[str(i)],
                    tpr[str(i)],
                    color = color,
                    lw = lw,
                    label = 'ROC - "{0}" (area = {1:0.2f})'.format(class_names[i].capitali
                );


    plt.plot([0, 1],[0, 1], 'k--', lw = lw, label = 'Random Guessing')
    plt.xlim([-0.025, 1.0])
    plt.ylim([0.0, 1.025])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve (TPR vs FPR)' + title)
    plt.legend(loc = 'lower right')
    plt.show()


    macro_roc_auc_ovo = roc_auc_score(y_b, yp, multi_class="ovo",
                                      average="macro")
    weighted_roc_auc_ovo = roc_auc_score(y_b, yp, multi_class="ovo",
                                         average="weighted")

    macro_roc_auc_ovr = roc_auc_score(y_b, yp, multi_class="ovr",
                                      average="macro")
    weighted_roc_auc_ovr = roc_auc_score(y_b, yp, multi_class="ovr",
                                         average="weighted")

    print("One-vs-One ROC AUC scores:\n\t{:.6f} (macro),\n\t{:.6f} "
              "(weighted by prevalence)"
              .format(macro_roc_auc_ovo, weighted_roc_auc_ovo))
    print("One-vs-Rest ROC AUC scores:\n\t{:.6f} (macro),\n\t{:.6f} "
              "(weighted by prevalence)"
              .format(macro_roc_auc_ovr, weighted_roc_auc_ovr))
```

In [ ]:

```
def print_general_metrics(y_pred, y_test):
    pred = tf.argmax(y_pred, axis = -1)
    precision, recall, fbeta_score, support = precision_recall_fscore_support(y_test, pred,


    print('\nClassification Report:\n')
    print(classification_report(y_test, pred), end='\n\n')

    print('Accuracy: %.3f\n' % accuracy_score(y_test, pred))
    print('Balanced accuracy: %.3f\n' % balanced_accuracy_score(y_test, pred))
    print('F1_Score: %.3f\n' % f1_score(y_test, pred, average='weighted'))
```

```
    print('Precision: %.3f\n' % precision)
    print('Recall: %.3f\n' % recall)
    print('ROC AUC (OVR): %.3f\n' % roc_auc_score(y_test, y_pred, multi_class = 'ovr'))
    print('ROC AUC (OVO): %.3f\n\n' % roc_auc_score(y_test, y_pred, multi_class = 'ovo'))
```

Dataset overview

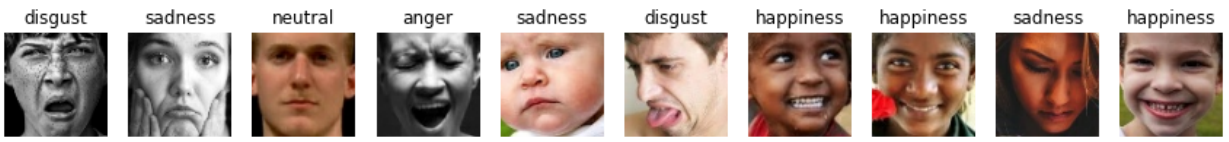
In [ ]:

```
prinf_dataset_information()
```

```
Train Cardinality (number of batches):  185
Validation cardinality (number of batches):  33
Test cardinality (number of batches):  54

Class names:  ['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise']

Shape of X_train:  (5898, 80, 80, 3)
Shape of y_train:  (5898,)

Shape of X_val:  (1040, 80, 80, 3)
Shape of y_val:  (1040,)

Shape of X_test:  (1700, 80, 80, 3)
Shape of y_test:  (1700,)

Class Balance - Train Data:

 anger = 0.19 %
 disgust = 0.08 %
 fear = 0.06 %
 happiness = 0.25 %
 neutral = 0.08 %
 sadness = 0.19 %
 surprise = 0.14 %


Class Balance - Validation Data:

 anger = 0.17 %
 disgust = 0.09 %
 fear = 0.06 %
 happiness = 0.27 %
 neutral = 0.08 %
 sadness = 0.17 %
 surprise = 0.14 %


Class Balance - Test Data:

 anger = 0.19 %
 disgust = 0.08 %
 fear = 0.06 %
 happiness = 0.25 %
 neutral = 0.08 %
 sadness = 0.19 %
 surprise = 0.14 %
```

In [ ]:

```
plt.figure(figsize=(15, 10))
for images, labels in train_ds.take(1):
    for i in range(10):
        ax = plt.subplot(2, 10, i + 1)
        plt.imshow(images[i].numpy().astype("float32"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



Beggining of Tensorflow analysis

Base model without data augmentation

In [ ]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

inputs = keras.Input(shape=(80, 80, 3))

x = keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', padding='same')(input
x = keras.layers.MaxPooling2D()(x)

x = keras.layers.Conv2D(64, 3, activation='relu',padding='VALID')(x)
x = keras.layers.MaxPooling2D()(x)

x = keras.layers.Conv2D(120, 3, activation='relu',padding='VALID')(x)
x = keras.layers.MaxPooling2D()(x)

x = keras.layers.Flatten()(x)

x = keras.layers.Dense(90, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
x = keras.layers.Dense(60, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)

outputs = keras.layers.Dense(7, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 80, 80, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 80, 80, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 38, 38, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 17, 17, 120)       69240
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 120)         0
_________________________________________________________________
flatten (Flatten)            (None, 7680)              0
_________________________________________________________________
dense (Dense)                (None, 90)                691290
_________________________________________________________________
dropout (Dropout)            (None, 90)                0
_________________________________________________________________
dense_1 (Dense)              (None, 60)                5460
_________________________________________________________________
dropout_1 (Dropout)          (None, 60)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 427
=================================================================
Total params: 785,809
```

```
Trainable params: 785,809
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
L = keras.losses.SparseCategoricalCrossentropy()

model.compile(loss=L, optimizer="adam", metrics=["accuracy"])
```

In [ ]:

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=50
)
```

```
Epoch 1/50
185/185 [==============================] - 2s 8ms/step - loss: 1.8441 - accuracy: 0.2567 -
val_loss: 1.6502 - val_accuracy: 0.3683
Epoch 2/50
185/185 [==============================] - 1s 7ms/step - loss: 1.6568 - accuracy: 0.3694 -
val_loss: 1.4549 - val_accuracy: 0.4702
Epoch 3/50
185/185 [==============================] - 1s 7ms/step - loss: 1.5331 - accuracy: 0.4262 -
val_loss: 1.3602 - val_accuracy: 0.5192
Epoch 4/50
185/185 [==============================] - 1s 7ms/step - loss: 1.4303 - accuracy: 0.4742 -
val_loss: 1.2760 - val_accuracy: 0.5356
Epoch 5/50
185/185 [==============================] - 1s 7ms/step - loss: 1.3679 - accuracy: 0.4941 -
val_loss: 1.1834 - val_accuracy: 0.5683
Epoch 6/50
185/185 [==============================] - 1s 7ms/step - loss: 1.2965 - accuracy: 0.5227 -
val_loss: 1.1442 - val_accuracy: 0.5856
Epoch 7/50
185/185 [==============================] - 1s 7ms/step - loss: 1.2172 - accuracy: 0.5605 -
val_loss: 1.0952 - val_accuracy: 0.6029
Epoch 8/50
185/185 [==============================] - 2s 9ms/step - loss: 1.1744 - accuracy: 0.5734 -
val_loss: 1.0851 - val_accuracy: 0.6279
Epoch 9/50
185/185 [==============================] - 2s 9ms/step - loss: 1.1061 - accuracy: 0.6063 -
val_loss: 1.0274 - val_accuracy: 0.6433
Epoch 10/50
185/185 [==============================] - 2s 9ms/step - loss: 1.0558 - accuracy: 0.6233 -
val_loss: 1.0118 - val_accuracy: 0.6356
Epoch 11/50
185/185 [==============================] - 1s 8ms/step - loss: 1.0144 - accuracy: 0.6387 -
val_loss: 1.0045 - val_accuracy: 0.6471
Epoch 12/50
185/185 [==============================] - 2s 8ms/step - loss: 0.9750 - accuracy: 0.6475 -
val_loss: 0.9870 - val_accuracy: 0.6577
Epoch 13/50
185/185 [==============================] - 2s 9ms/step - loss: 0.9228 - accuracy: 0.6621 -
val_loss: 0.9446 - val_accuracy: 0.6731
Epoch 14/50
185/185 [==============================] - 2s 8ms/step - loss: 0.8844 - accuracy: 0.6770 -
val_loss: 0.9657 - val_accuracy: 0.6596
Epoch 15/50
185/185 [==============================] - 1s 8ms/step - loss: 0.8395 - accuracy: 0.6936 -
val_loss: 1.0189 - val_accuracy: 0.6808
Epoch 16/50
185/185 [==============================] - 1s 8ms/step - loss: 0.8224 - accuracy: 0.6989 -
val_loss: 0.9518 - val_accuracy: 0.6923
Epoch 17/50
185/185 [==============================] - 2s 8ms/step - loss: 0.7970 - accuracy: 0.7104 -
val_loss: 0.9277 - val_accuracy: 0.7067
Epoch 18/50
185/185 [==============================] - 2s 8ms/step - loss: 0.7493 - accuracy: 0.7221 -
val_loss: 0.9234 - val_accuracy: 0.6952
Epoch 19/50
185/185 [==============================] - 2s 8ms/step - loss: 0.7205 - accuracy: 0.7353 -
val_loss: 0.9911 - val_accuracy: 0.6779
```

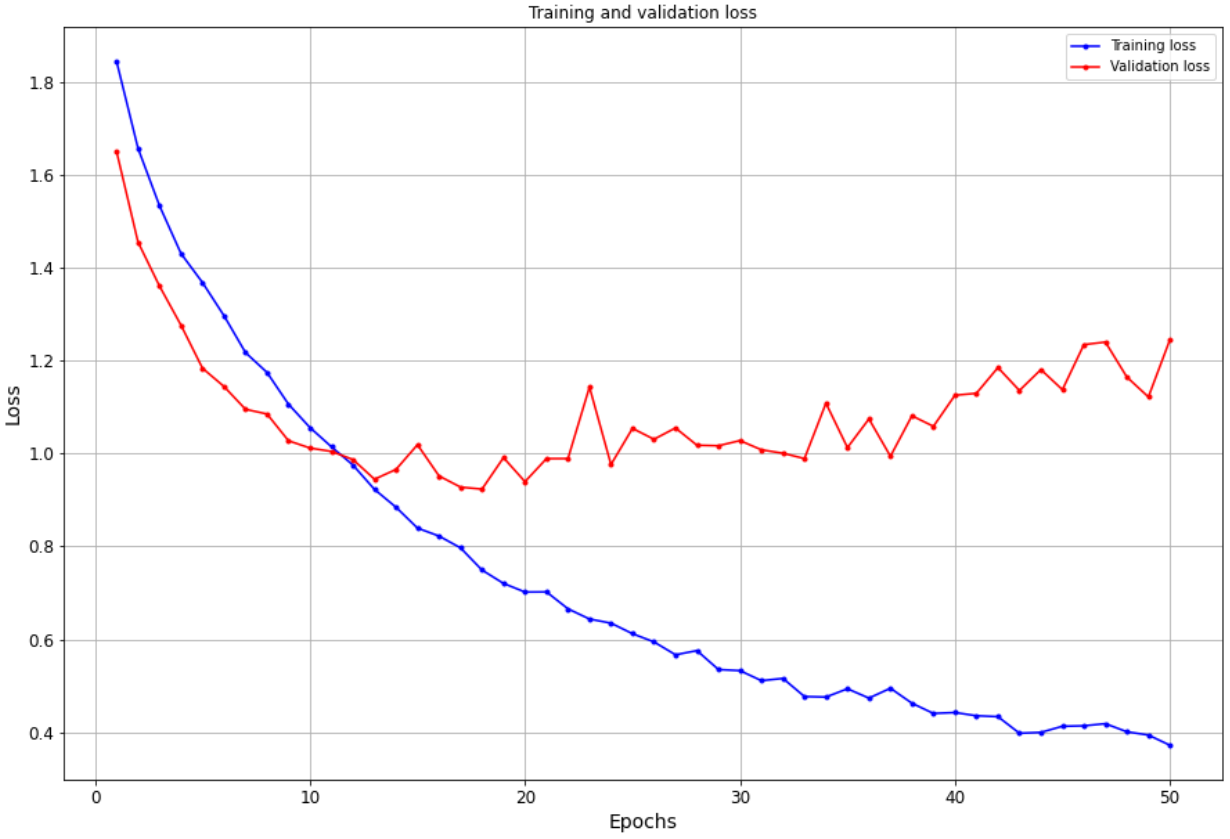
```
Epoch 20/50
185/185 [==============================] - 2s 8ms/step - loss: 0.7019 - accuracy: 0.7435 -
val_loss: 0.9394 - val_accuracy: 0.7029
Epoch 21/50
185/185 [==============================] - 1s 8ms/step - loss: 0.7022 - accuracy: 0.7408 -
val_loss: 0.9890 - val_accuracy: 0.7115
Epoch 22/50
185/185 [==============================] - 1s 7ms/step - loss: 0.6658 - accuracy: 0.7596 -
val_loss: 0.9891 - val_accuracy: 0.7058
Epoch 23/50
185/185 [==============================] - 1s 7ms/step - loss: 0.6438 - accuracy: 0.7691 -
val_loss: 1.1430 - val_accuracy: 0.7058
Epoch 24/50
185/185 [==============================] - 1s 7ms/step - loss: 0.6350 - accuracy: 0.7655 -
val_loss: 0.9763 - val_accuracy: 0.7163
Epoch 25/50
185/185 [==============================] - 1s 7ms/step - loss: 0.6123 - accuracy: 0.7772 -
val_loss: 1.0541 - val_accuracy: 0.7096
Epoch 26/50
185/185 [==============================] - 1s 7ms/step - loss: 0.5946 - accuracy: 0.7874 -
val_loss: 1.0307 - val_accuracy: 0.7212
Epoch 27/50
185/185 [==============================] - 1s 7ms/step - loss: 0.5669 - accuracy: 0.7945 -
val_loss: 1.0550 - val_accuracy: 0.7192
Epoch 28/50
185/185 [==============================] - 1s 7ms/step - loss: 0.5758 - accuracy: 0.7909 -
val_loss: 1.0178 - val_accuracy: 0.7308
Epoch 29/50
185/185 [==============================] - 1s 7ms/step - loss: 0.5351 - accuracy: 0.8081 -
val_loss: 1.0167 - val_accuracy: 0.7260
Epoch 30/50
185/185 [==============================] - 1s 7ms/step - loss: 0.5327 - accuracy: 0.8126 -
val_loss: 1.0278 - val_accuracy: 0.7192
Epoch 31/50
185/185 [==============================] - 1s 7ms/step - loss: 0.5112 - accuracy: 0.8126 -
val_loss: 1.0080 - val_accuracy: 0.7317
Epoch 32/50
185/185 [==============================] - 1s 8ms/step - loss: 0.5160 - accuracy: 0.8188 -
val_loss: 1.0006 - val_accuracy: 0.7337
Epoch 33/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4770 - accuracy: 0.8260 -
val_loss: 0.9891 - val_accuracy: 0.7433
Epoch 34/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4759 - accuracy: 0.8271 -
val_loss: 1.1083 - val_accuracy: 0.7510
Epoch 35/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4937 - accuracy: 0.8227 -
val_loss: 1.0128 - val_accuracy: 0.7346
Epoch 36/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4736 - accuracy: 0.8301 -
val_loss: 1.0744 - val_accuracy: 0.7394
Epoch 37/50
185/185 [==============================] - 2s 10ms/step - loss: 0.4949 - accuracy: 0.8223 -
val_loss: 0.9941 - val_accuracy: 0.7337
Epoch 38/50
185/185 [==============================] - 2s 8ms/step - loss: 0.4626 - accuracy: 0.8332 -
val_loss: 1.0812 - val_accuracy: 0.7375
Epoch 39/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4406 - accuracy: 0.8367 -
val_loss: 1.0585 - val_accuracy: 0.7413
Epoch 40/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4426 - accuracy: 0.8391 -
val_loss: 1.1254 - val_accuracy: 0.7375
Epoch 41/50
185/185 [==============================] - 1s 8ms/step - loss: 0.4356 - accuracy: 0.8437 -
val_loss: 1.1296 - val_accuracy: 0.7490
Epoch 42/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4337 - accuracy: 0.8452 -
val_loss: 1.1849 - val_accuracy: 0.7356
Epoch 43/50
185/185 [==============================] - 1s 7ms/step - loss: 0.3980 - accuracy: 0.8576 -
val_loss: 1.1357 - val_accuracy: 0.7269
Epoch 44/50
185/185 [==============================] - 1s 7ms/step - loss: 0.3998 - accuracy: 0.8547 -
```

```
val_loss: 1.1806 - val_accuracy: 0.7231
Epoch 45/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4130 - accuracy: 0.8554 -
val_loss: 1.1374 - val_accuracy: 0.7183
Epoch 46/50
185/185 [==============================] - 1s 7ms/step - loss: 0.4140 - accuracy: 0.8550 -
val_loss: 1.2345 - val_accuracy: 0.7317
Epoch 47/50
185/185 [==============================] - 1s 8ms/step - loss: 0.4186 - accuracy: 0.8491 -
val_loss: 1.2398 - val_accuracy: 0.7356
Epoch 48/50
185/185 [==============================] - 2s 9ms/step - loss: 0.4011 - accuracy: 0.8574 -
val_loss: 1.1644 - val_accuracy: 0.7317
Epoch 49/50
185/185 [==============================] - 2s 9ms/step - loss: 0.3941 - accuracy: 0.8598 -
val_loss: 1.1217 - val_accuracy: 0.7308
Epoch 50/50
185/185 [==============================] - 1s 7ms/step - loss: 0.3722 - accuracy: 0.8672 -
val_loss: 1.2449 - val_accuracy: 0.7471
```

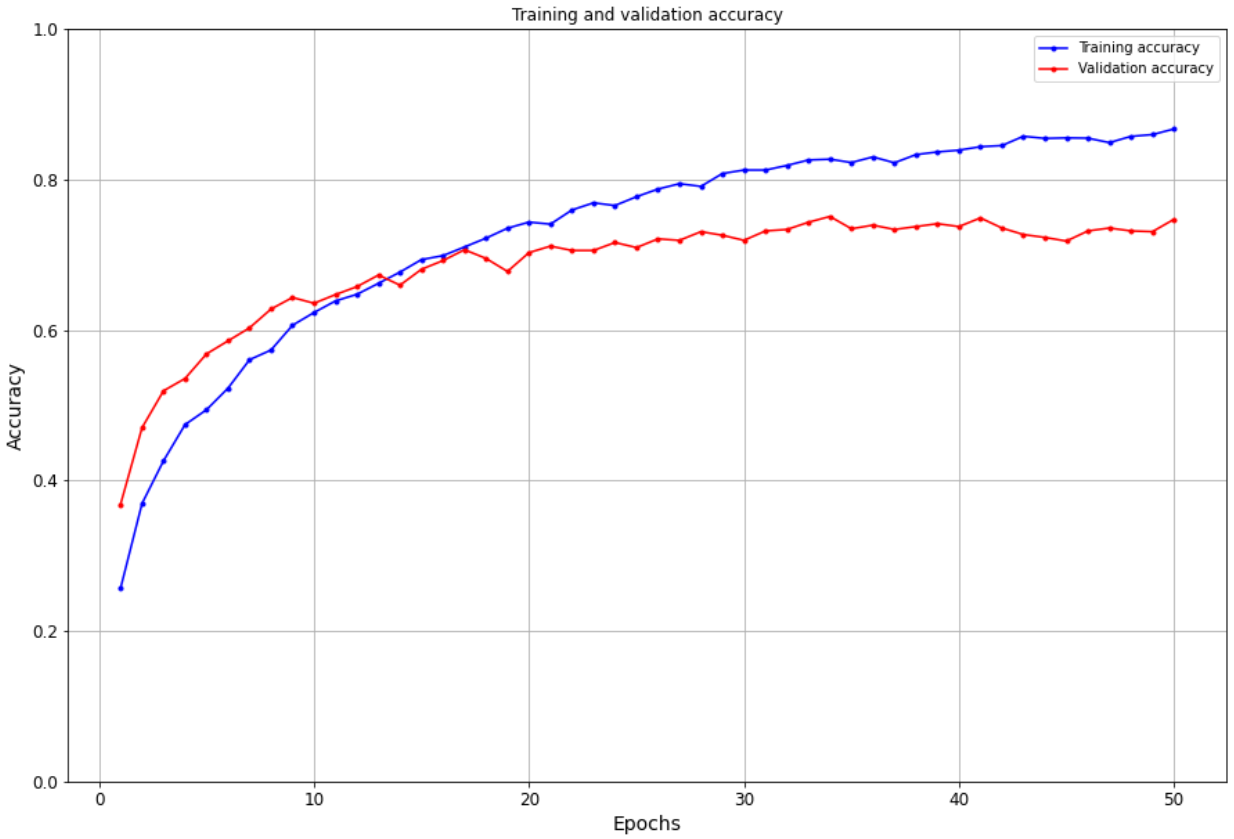
In [ ]:

```
plot_loss_and_accuracy(history)
```

```
accuracy    0.867243
dtype: float64 val_accuracy    0.750962
dtype: float64
```




```
<Figure size 432x288 with 0 Axes>
```



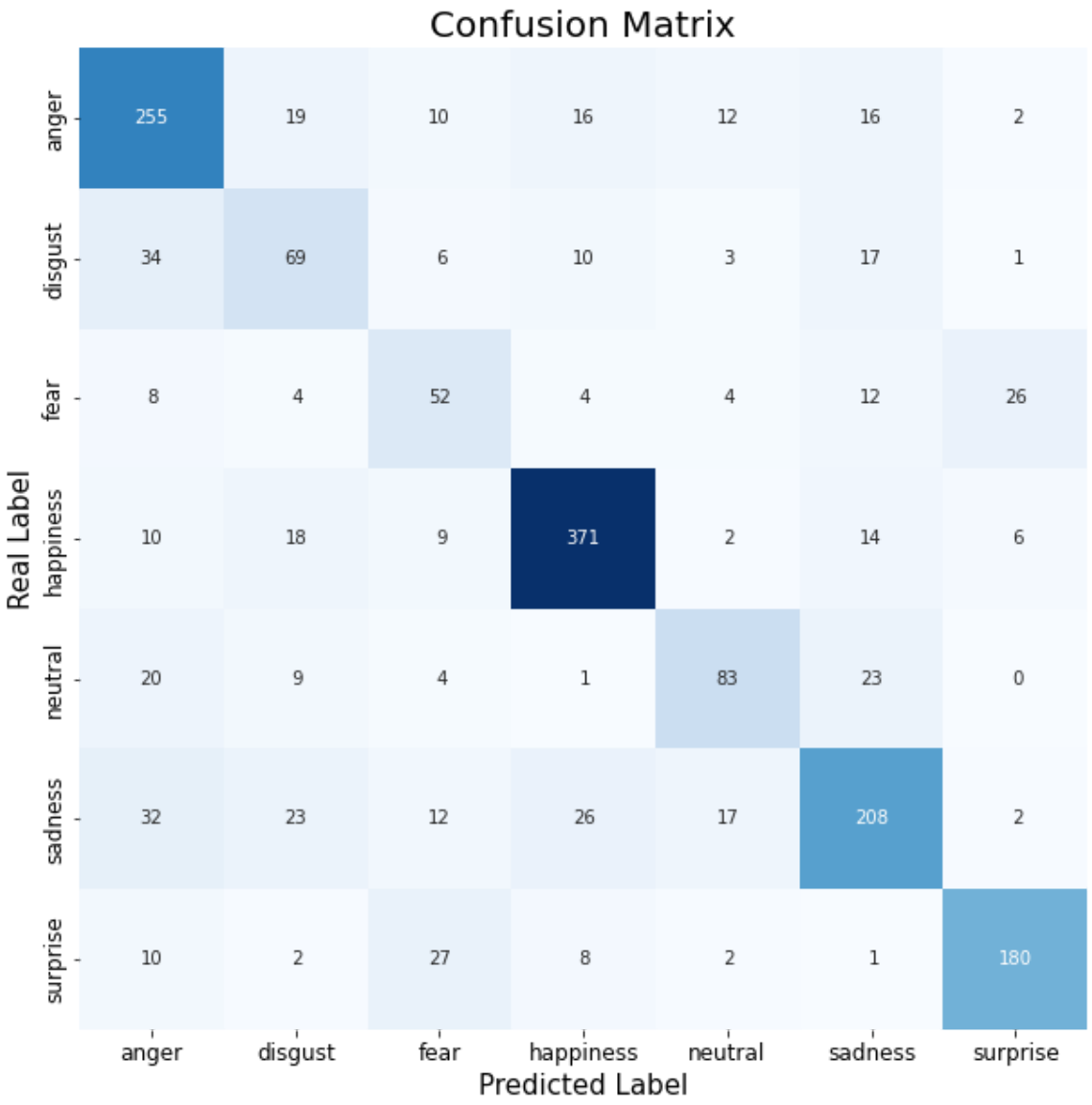
In [ ]:
```
model.evaluate(test_ds)
```

```
54/54 [==============================] - 0s 4ms/step - loss: 1.5071 - accuracy: 0.7165
```

Out[ ]:
```
[1.507069706916809, 0.7164705991744995]
```

In [ ]:
```
# y_pred are probs, we have to convert them to int
y_pred = model.predict(test_ds)
```

In [ ]:
```
cnf_matrix = plot_confusion_matrix(y_pred, y_test, class_names, return_m=True)
```



In [ ]:

```
print_confusion_matrix_metrics(cnf_matrix)
```

```
False Positives:

        anger -> 114
        disgust -> 75
        fear -> 68
        happiness -> 65
        neutral -> 40
        sadness -> 83
        surprise -> 37

False Negatives:

        anger -> 75
        disgust -> 71
        fear -> 58
        happiness -> 59
        neutral -> 57
        sadness -> 112
        surprise -> 50

True Positives:

        anger -> 255
        disgust -> 69
        fear -> 52
        happiness -> 371
        neutral -> 83
        sadness -> 208
```

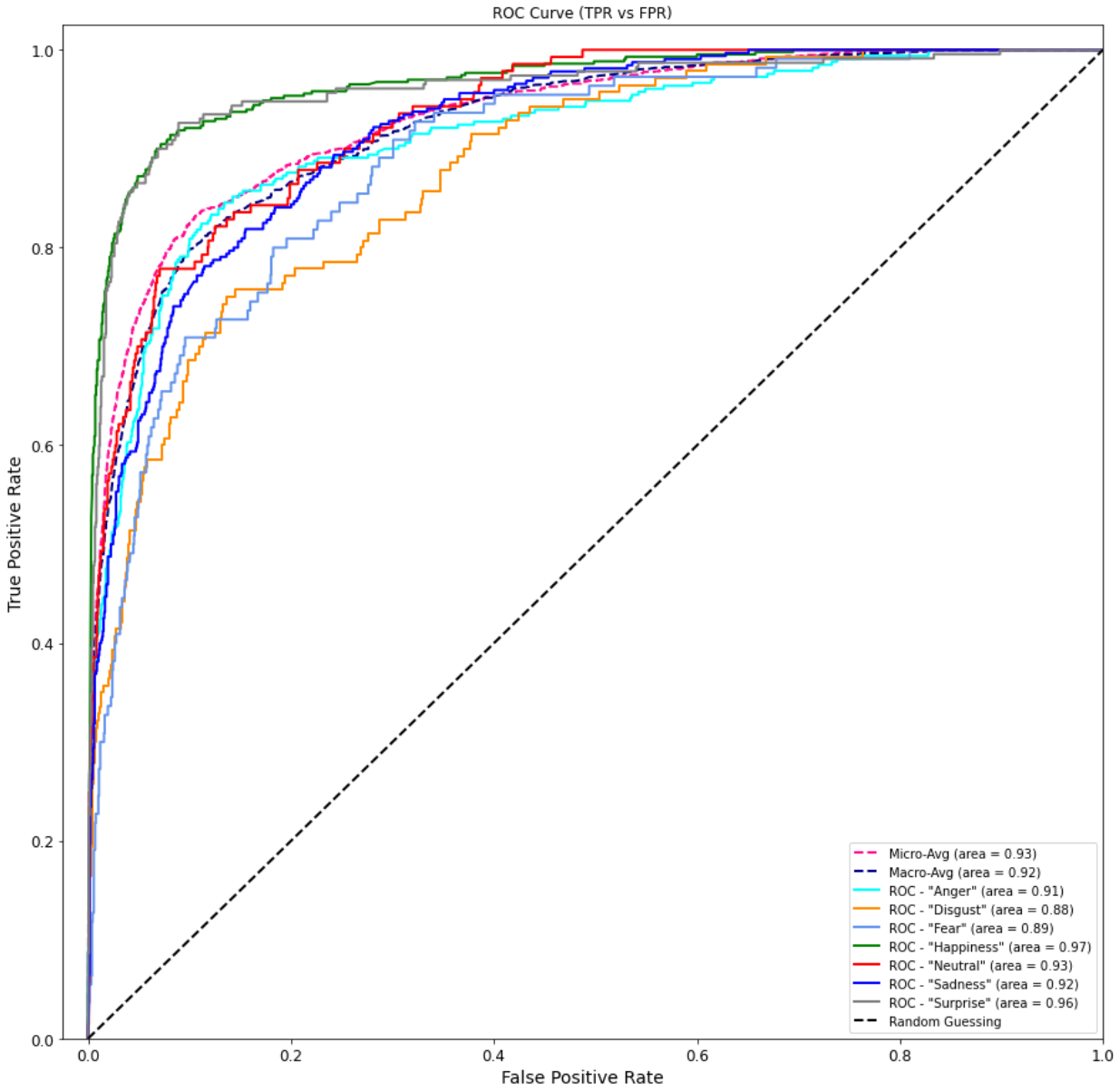
```
        surprise -> 180

True Negatives:

        anger -> 1256
        disgust -> 1485
        fear -> 1522
        happiness -> 1205
        neutral -> 1520
        sadness -> 1297
        surprise -> 1433

True Positive Rate:

        anger -> 0.773%
        disgust -> 0.493%
        fear -> 0.473%
        happiness -> 0.863%
        neutral -> 0.593%
        sadness -> 0.650%
        surprise -> 0.783%

True Negative Rate:

        anger -> 0.917%
        disgust -> 0.952%
        fear -> 0.957%
        happiness -> 0.949%
        neutral -> 0.974%
        sadness -> 0.940%
        surprise -> 0.975%

Positive Predictive Value:

        anger -> 1
        disgust -> 0
        fear -> 0
        happiness -> 1
        neutral -> 1
        sadness -> 1
        surprise -> 1

Negative Predictive Value:

        anger -> 1
        disgust -> 1
        fear -> 1
        happiness -> 1
        neutral -> 1
        sadness -> 1
        surprise -> 1

False Positive Rate:

        anger -> 0.083%
        disgust -> 0.048%
        fear -> 0.043%
        happiness -> 0.051%
        neutral -> 0.026%
        sadness -> 0.060%
        surprise -> 0.025%

False Negative Rate:

        anger -> 0.227%
        disgust -> 0.507%
        fear -> 0.527%
        happiness -> 0.137%
        neutral -> 0.407%
        sadness -> 0.350%
        surprise -> 0.217%

False Discovery Rate:
```

```
        anger -> 0.309%
        disgust -> 0.521%
        fear -> 0.567%
        happiness -> 0.149%
        neutral -> 0.325%
        sadness -> 0.285%
        surprise -> 0.171%

Accuracy:

        anger -> 0.889%
        disgust -> 0.914%
        fear -> 0.926%
        happiness -> 0.927%
        neutral -> 0.943%
        sadness -> 0.885%
        surprise -> 0.949%
```

In [ ]:
```
print_general_metrics(y_pred, y_test)
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.69      0.77      0.73       330
           1       0.48      0.49      0.49       140
           2       0.43      0.47      0.45       110
           3       0.85      0.86      0.86       430
           4       0.67      0.59      0.63       140
           5       0.71      0.65      0.68       320
           6       0.83      0.78      0.81       230

    accuracy                           0.72      1700
   macro avg       0.67      0.66      0.66      1700
weighted avg       0.72      0.72      0.72      1700


Accuracy: 0.716

Balanced accuracy: 0.661

F1_Score: 0.717

Precision: 0.719

Recall: 0.716

ROC AUC (OVR): 0.923

ROC AUC (OVO): 0.910
```

In [ ]:
```
plot_multiclass_roc_curve(y_test, y_pred, class_names)
```




```
One-vs-One ROC AUC scores:
        0.922703 (macro),
        0.931220 (weighted by prevalence)
One-vs-Rest ROC AUC scores:
        0.922703 (macro),
        0.931220 (weighted by prevalence)
```

Create data augmentation

In [ ]:

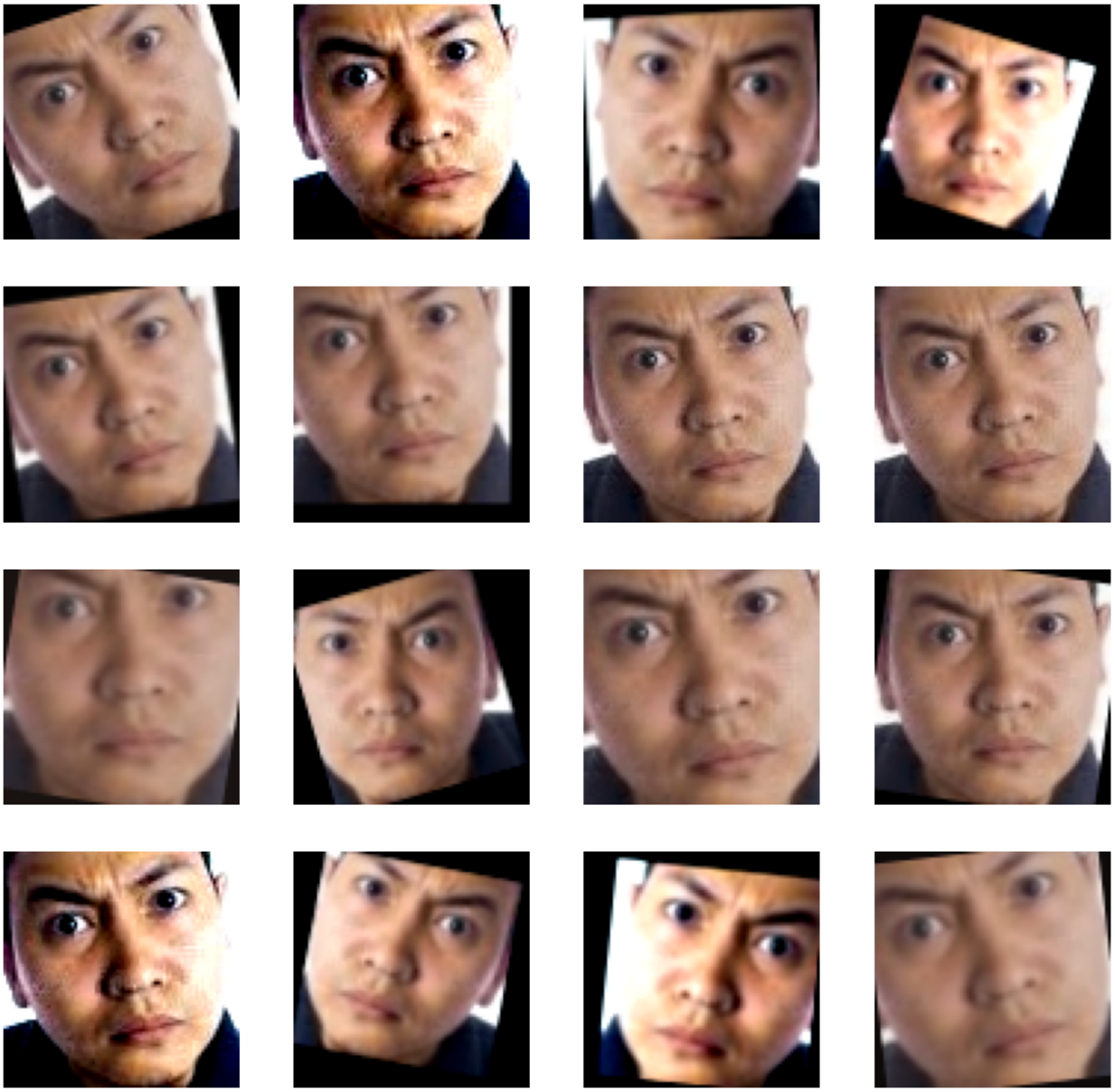
```
import random

transformations = [
    layers.RandomFlip("horizontal",input_shape=(80, 80, 3)),
    layers.RandomTranslation(height_factor=0.1, width_factor=0.1, fill_mode="constant" ,inp
    layers.RandomZoom(0.2,  fill_mode="constant",input_shape=(80, 80, 3)),
    layers.RandomRotation(0.05, fill_mode="constant",input_shape=(80, 80, 3)),
    layers.RandomContrast(factor=[0.5, 1.3],input_shape=(80, 80, 3)),
    layers.GaussianNoise(0.03)
]

def customDataAugmentationLayer(input):
    if random.random() > 0.5 :
        return random.choice(transformations)(input)
    else:
        result = input
        for transformation in transformations:
            result = transformation(result)
```

```
        return result

dataAug = layers.Lambda(lambda input: customDataAugmentationLayer(input))
```

In [ ]:

```
plt.figure(figsize=(20, 20))
for images, _ in train_ds.take(1):
    for i in range(16):
        augmented_images = dataAug(images)
        #augmented_images = CustomDataAugmentationLayer()(images)
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(augmented_images[12].numpy().astype("float32"))
        plt.axis("off")
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
```



Improved CNN

In [ ]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

inputs = keras.Input(shape=(80, 80, 3))

x = keras.Sequential(transformations)(inputs)

x = layers.Conv2D(filters=32, kernel_size=3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=32, kernel_size=3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=32, kernel_size=3,  padding='same',  use_bias=False)(x)
x = layers.Activation("relu")(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D()(x)


x = layers.Conv2D(64, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(64, 3,  padding='same',  use_bias=False)(x)
```

```
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(64, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D()(x)


x = layers.Conv2D(128, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(128, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(128, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D()(x)


x = layers.Conv2D(256, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(256, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(256, 3,  padding='same',  use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Dropout(0.1)(x)


x = layers.Flatten()(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.25)(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(60, activation='relu')(x)
x = layers.Dropout(0.25)(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(30, activation='relu')(x)
x = layers.Dropout(0.25)(x)
x = layers.BatchNormalization()(x)

outputs = keras.layers.Dense(7, activation="softmax")(x)

modelWithAugmentation = keras.Model(inputs=inputs, outputs=outputs)
modelWithAugmentation.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 80, 80, 3)]       0
_________________________________________________________________
sequential (Sequential)      (None, 80, 80, 3)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 80, 80, 32)        864
_________________________________________________________________
batch_normalization (BatchNo (None, 80, 80, 32)        128
_________________________________________________________________
activation (Activation)      (None, 80, 80, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 80, 80, 32)        9216
_________________________________________________________________
batch_normalization_1 (Batch (None, 80, 80, 32)        128
_________________________________________________________________
activation_1 (Activation)    (None, 80, 80, 32)        0
_________________________________________________________________
```

```
conv2d_2 (Conv2D)            (None, 80, 80, 32)        9216
_________________________________________________________________
activation_2 (Activation)    (None, 80, 80, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 80, 80, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 40, 40, 64)        18432
_________________________________________________________________
batch_normalization_3 (Batch (None, 40, 40, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 40, 40, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 40, 40, 64)        36864
_________________________________________________________________
batch_normalization_4 (Batch (None, 40, 40, 64)        256
_________________________________________________________________
activation_4 (Activation)    (None, 40, 40, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 40, 40, 64)        36864
_________________________________________________________________
batch_normalization_5 (Batch (None, 40, 40, 64)        256
_________________________________________________________________
activation_5 (Activation)    (None, 40, 40, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 20, 20, 128)       73728
_________________________________________________________________
batch_normalization_6 (Batch (None, 20, 20, 128)       512
_________________________________________________________________
activation_6 (Activation)    (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 20, 20, 128)       147456
_________________________________________________________________
batch_normalization_7 (Batch (None, 20, 20, 128)       512
_________________________________________________________________
activation_7 (Activation)    (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 20, 20, 128)       147456
_________________________________________________________________
batch_normalization_8 (Batch (None, 20, 20, 128)       512
_________________________________________________________________
activation_8 (Activation)    (None, 20, 20, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 256)       294912
_________________________________________________________________
batch_normalization_9 (Batch (None, 10, 10, 256)       1024
_________________________________________________________________
activation_9 (Activation)    (None, 10, 10, 256)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 10, 10, 256)       589824
_________________________________________________________________
batch_normalization_10 (Batc (None, 10, 10, 256)       1024
_________________________________________________________________
activation_10 (Activation)   (None, 10, 10, 256)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 256)       589824
_________________________________________________________________
batch_normalization_11 (Batc (None, 10, 10, 256)       1024
_________________________________________________________________
activation_11 (Activation)   (None, 10, 10, 256)       0
_________________________________________________________________
dropout (Dropout)            (None, 10, 10, 256)       0
_________________________________________________________________
flatten (Flatten)            (None, 25600)             0
_________________________________________________________________
batch_normalization_12 (Batc (None, 25600)             102400
_________________________________________________________________
dense (Dense)                (None, 128)               3276928
_________________________________________________________________
```

```
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 60)                7740
_________________________________________________________________
dropout_2 (Dropout)          (None, 60)                0
_________________________________________________________________
batch_normalization_14 (Batc (None, 60)                240
_________________________________________________________________
dense_2 (Dense)              (None, 30)                1830
_________________________________________________________________
dropout_3 (Dropout)          (None, 30)                0
_________________________________________________________________
batch_normalization_15 (Batc (None, 30)                120
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 217
=================================================================
Total params: 5,350,403
Trainable params: 5,295,887
Non-trainable params: 54,516
_________________________________________________________________
```

In [ ]:

```
# Tensorboard Logs
root_logdir = os.path.join(os.curdir, 'tensorboard_logs_CNN')

# Sub-diretoria onde ficarão os dados da execução: tem um nome criado a partir da hora de i
def get_run_logdir():
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)

best_model_file_path = "model_CNN.hdf5"

callbacks_list = [
    keras.callbacks.TensorBoard(
        get_run_logdir(),
        histogram_freq = 1
    ),
    tf.keras.callbacks.EarlyStopping(
        monitor='val_accuracy',
        patience = 500,
        min_delta = 0.005,
        restore_best_weights = True,
        mode='auto'
    ),
    keras.callbacks.ModelCheckpoint(
        filepath = best_model_file_path,
        monitor = "val_accuracy",
        verbose=0,
        save_best_only = True,
        save_weights_only = False,
        mode='max'
    )
]

# Compilar e treinar
#from keras.callbacks import ModelCheckpoint
#filepath = 'my_best_model.hdf5'
#checkpoint = ModelCheckpoint(filepath=filepath,
#                             monitor='val_accuracy',
#                             verbose=0,
#                             save_best_only=True,
#                             mode='max')
#callbacks = [checkpoint]
```

In [ ]:

```
L = keras.losses.SparseCategoricalCrossentropy()

modelWithAugmentation.compile(loss=L, optimizer="adam", metrics=["accuracy"])
```

```
history = modelWithAugmentation.fit(
  train_ds,
  validation_data=val_ds,
  epochs=200,
  callbacks=callbacks_list
)
```

```
Epoch 1/200
185/185 [==============================] - 10s 43ms/step - loss: 2.1864 - accuracy: 0.1931
- val_loss: 3.4879 - val_accuracy: 0.1750
Epoch 2/200
185/185 [==============================] - 7s 39ms/step - loss: 1.9369 - accuracy: 0.2599 -
val_loss: 1.9714 - val_accuracy: 0.2500
Epoch 3/200
185/185 [==============================] - 7s 39ms/step - loss: 1.7552 - accuracy: 0.3410 -
val_loss: 1.8187 - val_accuracy: 0.3250
Epoch 4/200
185/185 [==============================] - 7s 38ms/step - loss: 1.5977 - accuracy: 0.4137 -
val_loss: 1.4470 - val_accuracy: 0.4894
Epoch 5/200
185/185 [==============================] - 7s 38ms/step - loss: 1.4905 - accuracy: 0.4556 -
val_loss: 1.4694 - val_accuracy: 0.4606
Epoch 6/200
185/185 [==============================] - 7s 38ms/step - loss: 1.3889 - accuracy: 0.5015 -
val_loss: 1.2969 - val_accuracy: 0.5356
Epoch 7/200
185/185 [==============================] - 7s 38ms/step - loss: 1.3293 - accuracy: 0.5273 -
val_loss: 1.7715 - val_accuracy: 0.3712
Epoch 8/200
185/185 [==============================] - 7s 39ms/step - loss: 1.2618 - accuracy: 0.5544 -
val_loss: 1.0919 - val_accuracy: 0.5990
Epoch 9/200
185/185 [==============================] - 7s 38ms/step - loss: 1.2051 - accuracy: 0.5746 -
val_loss: 1.6163 - val_accuracy: 0.4173
Epoch 10/200
185/185 [==============================] - 7s 40ms/step - loss: 1.1541 - accuracy: 0.5899 -
val_loss: 1.0474 - val_accuracy: 0.6173
Epoch 11/200
185/185 [==============================] - 8s 42ms/step - loss: 1.1155 - accuracy: 0.6060 -
val_loss: 1.0406 - val_accuracy: 0.5942
Epoch 12/200
185/185 [==============================] - 7s 40ms/step - loss: 1.0851 - accuracy: 0.6121 -
val_loss: 1.2565 - val_accuracy: 0.5375
Epoch 13/200
185/185 [==============================] - 7s 39ms/step - loss: 1.0480 - accuracy: 0.6333 -
val_loss: 1.0543 - val_accuracy: 0.5990
Epoch 14/200
185/185 [==============================] - 8s 41ms/step - loss: 1.0353 - accuracy: 0.6392 -
val_loss: 0.8821 - val_accuracy: 0.6933
Epoch 15/200
185/185 [==============================] - 7s 40ms/step - loss: 0.9967 - accuracy: 0.6451 -
val_loss: 0.9958 - val_accuracy: 0.6510
Epoch 16/200
185/185 [==============================] - 7s 39ms/step - loss: 0.9881 - accuracy: 0.6553 -
val_loss: 1.0804 - val_accuracy: 0.6010
Epoch 17/200
185/185 [==============================] - 7s 39ms/step - loss: 0.9524 - accuracy: 0.6640 -
val_loss: 0.9013 - val_accuracy: 0.6779
Epoch 18/200
185/185 [==============================] - 8s 41ms/step - loss: 0.9482 - accuracy: 0.6628 -
val_loss: 0.8775 - val_accuracy: 0.6846
Epoch 19/200
185/185 [==============================] - 7s 40ms/step - loss: 0.9239 - accuracy: 0.6745 -
val_loss: 1.0454 - val_accuracy: 0.6250
Epoch 20/200
185/185 [==============================] - 7s 40ms/step - loss: 0.9055 - accuracy: 0.6765 -
val_loss: 0.8936 - val_accuracy: 0.6740
Epoch 21/200
185/185 [==============================] - 7s 40ms/step - loss: 0.8839 - accuracy: 0.6985 -
val_loss: 0.8053 - val_accuracy: 0.7019
Epoch 22/200
185/185 [==============================] - 7s 40ms/step - loss: 0.8884 - accuracy: 0.6909 -
```

```
val_loss: 0.7536 - val_accuracy: 0.7308
Epoch 23/200
185/185 [==============================] - 7s 40ms/step - loss: 0.8724 - accuracy: 0.6980 -
val_loss: 0.8400 - val_accuracy: 0.6952
Epoch 24/200
185/185 [==============================] - 7s 40ms/step - loss: 0.8669 - accuracy: 0.6962 -
val_loss: 0.7616 - val_accuracy: 0.7375
Epoch 25/200
185/185 [==============================] - 8s 41ms/step - loss: 0.8321 - accuracy: 0.7062 -
val_loss: 0.7725 - val_accuracy: 0.7144
Epoch 26/200
185/185 [==============================] - 7s 39ms/step - loss: 0.8431 - accuracy: 0.7121 -
val_loss: 0.8081 - val_accuracy: 0.7154
Epoch 27/200
185/185 [==============================] - 7s 38ms/step - loss: 0.8129 - accuracy: 0.7163 -
val_loss: 1.0164 - val_accuracy: 0.6317
Epoch 28/200
185/185 [==============================] - 7s 38ms/step - loss: 0.8261 - accuracy: 0.7175 -
val_loss: 0.7511 - val_accuracy: 0.7221
Epoch 29/200
185/185 [==============================] - 7s 39ms/step - loss: 0.8100 - accuracy: 0.7230 -
val_loss: 1.0620 - val_accuracy: 0.6144
Epoch 30/200
185/185 [==============================] - 7s 39ms/step - loss: 0.7800 - accuracy: 0.7335 -
val_loss: 1.0031 - val_accuracy: 0.6500
Epoch 31/200
185/185 [==============================] - 7s 40ms/step - loss: 0.7753 - accuracy: 0.7333 -
val_loss: 0.8805 - val_accuracy: 0.6673
Epoch 32/200
185/185 [==============================] - 8s 41ms/step - loss: 0.7727 - accuracy: 0.7306 -
val_loss: 0.7614 - val_accuracy: 0.7240
Epoch 33/200
185/185 [==============================] - 7s 38ms/step - loss: 0.7388 - accuracy: 0.7460 -
val_loss: 0.8341 - val_accuracy: 0.6875
Epoch 34/200
185/185 [==============================] - 8s 41ms/step - loss: 0.7578 - accuracy: 0.7391 -
val_loss: 0.7842 - val_accuracy: 0.7231
Epoch 35/200
185/185 [==============================] - 7s 39ms/step - loss: 0.7235 - accuracy: 0.7525 -
val_loss: 0.7893 - val_accuracy: 0.7317
Epoch 36/200
185/185 [==============================] - 7s 39ms/step - loss: 0.7247 - accuracy: 0.7528 -
val_loss: 0.6822 - val_accuracy: 0.7567
Epoch 37/200
185/185 [==============================] - 7s 40ms/step - loss: 0.7219 - accuracy: 0.7553 -
val_loss: 0.7777 - val_accuracy: 0.7375
Epoch 38/200
185/185 [==============================] - 7s 40ms/step - loss: 0.7044 - accuracy: 0.7628 -
val_loss: 0.7745 - val_accuracy: 0.7231
Epoch 39/200
185/185 [==============================] - 7s 39ms/step - loss: 0.6888 - accuracy: 0.7603 -
val_loss: 0.7153 - val_accuracy: 0.7548
Epoch 40/200
185/185 [==============================] - 7s 39ms/step - loss: 0.6962 - accuracy: 0.7679 -
val_loss: 0.8409 - val_accuracy: 0.7154
Epoch 41/200
185/185 [==============================] - 7s 38ms/step - loss: 0.7018 - accuracy: 0.7648 -
val_loss: 0.9268 - val_accuracy: 0.6769
Epoch 42/200
185/185 [==============================] - 7s 38ms/step - loss: 0.6745 - accuracy: 0.7791 -
val_loss: 0.6642 - val_accuracy: 0.7596
Epoch 43/200
185/185 [==============================] - 7s 38ms/step - loss: 0.6566 - accuracy: 0.7803 -
val_loss: 0.6709 - val_accuracy: 0.7808
Epoch 44/200
185/185 [==============================] - 7s 38ms/step - loss: 0.6705 - accuracy: 0.7786 -
val_loss: 0.7620 - val_accuracy: 0.7260
Epoch 45/200
185/185 [==============================] - 7s 40ms/step - loss: 0.6438 - accuracy: 0.7818 -
val_loss: 0.6918 - val_accuracy: 0.7606
Epoch 46/200
185/185 [==============================] - 7s 39ms/step - loss: 0.6354 - accuracy: 0.7887 -
val_loss: 0.6468 - val_accuracy: 0.7817
Epoch 47/200
```

```
185/185 [==============================] - 7s 39ms/step - loss: 0.6177 - accuracy: 0.7913 -
val_loss: 0.6483 - val_accuracy: 0.7846
Epoch 48/200
185/185 [==============================] - 7s 38ms/step - loss: 0.6277 - accuracy: 0.7928 -
val_loss: 0.6563 - val_accuracy: 0.7933
Epoch 49/200
185/185 [==============================] - 8s 43ms/step - loss: 0.6106 - accuracy: 0.7967 -
val_loss: 0.7232 - val_accuracy: 0.7779
Epoch 50/200
185/185 [==============================] - 8s 42ms/step - loss: 0.6095 - accuracy: 0.7984 -
val_loss: 0.8676 - val_accuracy: 0.7125
Epoch 51/200
185/185 [==============================] - 8s 42ms/step - loss: 0.5954 - accuracy: 0.8052 -
val_loss: 0.6476 - val_accuracy: 0.7808
Epoch 52/200
185/185 [==============================] - 8s 41ms/step - loss: 0.5910 - accuracy: 0.8071 -
val_loss: 0.6660 - val_accuracy: 0.7837
Epoch 53/200
185/185 [==============================] - 7s 39ms/step - loss: 0.6089 - accuracy: 0.7998 -
val_loss: 0.6201 - val_accuracy: 0.8058
Epoch 54/200
185/185 [==============================] - 7s 40ms/step - loss: 0.5857 - accuracy: 0.8018 -
val_loss: 0.6290 - val_accuracy: 0.7971
Epoch 55/200
185/185 [==============================] - 7s 40ms/step - loss: 0.5642 - accuracy: 0.8142 -
val_loss: 0.7420 - val_accuracy: 0.7683
Epoch 56/200
185/185 [==============================] - 7s 38ms/step - loss: 0.5629 - accuracy: 0.8165 -
val_loss: 0.6990 - val_accuracy: 0.7760
Epoch 57/200
185/185 [==============================] - 7s 40ms/step - loss: 0.5592 - accuracy: 0.8154 -
val_loss: 0.6734 - val_accuracy: 0.7827
Epoch 58/200
185/185 [==============================] - 7s 38ms/step - loss: 0.5444 - accuracy: 0.8216 -
val_loss: 0.5701 - val_accuracy: 0.8106
Epoch 59/200
185/185 [==============================] - 7s 39ms/step - loss: 0.5290 - accuracy: 0.8277 -
val_loss: 0.6060 - val_accuracy: 0.8067
Epoch 60/200
185/185 [==============================] - 8s 42ms/step - loss: 0.5440 - accuracy: 0.8186 -
val_loss: 0.7763 - val_accuracy: 0.7625
Epoch 61/200
185/185 [==============================] - 7s 39ms/step - loss: 0.5280 - accuracy: 0.8301 -
val_loss: 0.6297 - val_accuracy: 0.7971
Epoch 62/200
185/185 [==============================] - 8s 42ms/step - loss: 0.5177 - accuracy: 0.8366 -
val_loss: 0.6561 - val_accuracy: 0.7817
Epoch 63/200
185/185 [==============================] - 7s 40ms/step - loss: 0.5241 - accuracy: 0.8330 -
val_loss: 0.6448 - val_accuracy: 0.7798
Epoch 64/200
185/185 [==============================] - 7s 40ms/step - loss: 0.5192 - accuracy: 0.8316 -
val_loss: 0.6549 - val_accuracy: 0.7865
Epoch 65/200
185/185 [==============================] - 7s 39ms/step - loss: 0.5067 - accuracy: 0.8372 -
val_loss: 0.6761 - val_accuracy: 0.7702
Epoch 66/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4861 - accuracy: 0.8440 -
val_loss: 0.6843 - val_accuracy: 0.7788
Epoch 67/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4819 - accuracy: 0.8438 -
val_loss: 0.6266 - val_accuracy: 0.7971
Epoch 68/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4865 - accuracy: 0.8452 -
val_loss: 0.5607 - val_accuracy: 0.8269
Epoch 69/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4716 - accuracy: 0.8511 -
val_loss: 0.6493 - val_accuracy: 0.7952
Epoch 70/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4612 - accuracy: 0.8499 -
val_loss: 0.6937 - val_accuracy: 0.7971
Epoch 71/200
185/185 [==============================] - 8s 41ms/step - loss: 0.4684 - accuracy: 0.8496 -
val_loss: 0.7955 - val_accuracy: 0.7798
```

```
Epoch 72/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4631 - accuracy: 0.8508 -
val_loss: 0.7537 - val_accuracy: 0.7615
Epoch 73/200
185/185 [==============================] - 7s 40ms/step - loss: 0.4725 - accuracy: 0.8523 -
val_loss: 0.7209 - val_accuracy: 0.7750
Epoch 74/200
185/185 [==============================] - 7s 40ms/step - loss: 0.4476 - accuracy: 0.8594 -
val_loss: 0.6550 - val_accuracy: 0.7923
Epoch 75/200
185/185 [==============================] - 8s 41ms/step - loss: 0.4470 - accuracy: 0.8571 -
val_loss: 0.7885 - val_accuracy: 0.7596
Epoch 76/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4253 - accuracy: 0.8700 -
val_loss: 0.8155 - val_accuracy: 0.7548
Epoch 77/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4431 - accuracy: 0.8618 -
val_loss: 0.6908 - val_accuracy: 0.7808
Epoch 78/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4179 - accuracy: 0.8681 -
val_loss: 0.6445 - val_accuracy: 0.7904
Epoch 79/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4261 - accuracy: 0.8616 -
val_loss: 0.9311 - val_accuracy: 0.7260
Epoch 80/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4317 - accuracy: 0.8605 -
val_loss: 0.6703 - val_accuracy: 0.7933
Epoch 81/200
185/185 [==============================] - 7s 39ms/step - loss: 0.4072 - accuracy: 0.8696 -
val_loss: 0.7347 - val_accuracy: 0.7779
Epoch 82/200
185/185 [==============================] - 7s 38ms/step - loss: 0.4035 - accuracy: 0.8728 -
val_loss: 0.7142 - val_accuracy: 0.7779
Epoch 83/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3868 - accuracy: 0.8835 -
val_loss: 0.7019 - val_accuracy: 0.7846
Epoch 84/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3964 - accuracy: 0.8693 -
val_loss: 0.6793 - val_accuracy: 0.8010
Epoch 85/200
185/185 [==============================] - 7s 40ms/step - loss: 0.3888 - accuracy: 0.8784 -
val_loss: 0.6179 - val_accuracy: 0.8173
Epoch 86/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3965 - accuracy: 0.8713 -
val_loss: 0.6326 - val_accuracy: 0.8029
Epoch 87/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3764 - accuracy: 0.8776 -
val_loss: 0.6829 - val_accuracy: 0.7923
Epoch 88/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3870 - accuracy: 0.8795 -
val_loss: 0.5837 - val_accuracy: 0.8144
Epoch 89/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3902 - accuracy: 0.8801 -
val_loss: 0.5958 - val_accuracy: 0.8154
Epoch 90/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3683 - accuracy: 0.8840 -
val_loss: 0.6259 - val_accuracy: 0.8067
Epoch 91/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3575 - accuracy: 0.8861 -
val_loss: 0.6639 - val_accuracy: 0.7990
Epoch 92/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3702 - accuracy: 0.8756 -
val_loss: 0.6503 - val_accuracy: 0.8067
Epoch 93/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3558 - accuracy: 0.8849 -
val_loss: 0.6849 - val_accuracy: 0.7837
Epoch 94/200
185/185 [==============================] - 8s 41ms/step - loss: 0.3611 - accuracy: 0.8864 -
val_loss: 0.6155 - val_accuracy: 0.8269
Epoch 95/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3638 - accuracy: 0.8889 -
val_loss: 0.5887 - val_accuracy: 0.8106
Epoch 96/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3256 - accuracy: 0.8971 -
```

```
val_loss: 0.6124 - val_accuracy: 0.8260
Epoch 97/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3408 - accuracy: 0.8893 -
val_loss: 0.6234 - val_accuracy: 0.8250
Epoch 98/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3340 - accuracy: 0.8874 -
val_loss: 0.6326 - val_accuracy: 0.8240
Epoch 99/200
185/185 [==============================] - 7s 40ms/step - loss: 0.3506 - accuracy: 0.8900 -
val_loss: 0.5931 - val_accuracy: 0.8202
Epoch 100/200
185/185 [==============================] - 8s 42ms/step - loss: 0.3485 - accuracy: 0.8861 -
val_loss: 0.6724 - val_accuracy: 0.8038
Epoch 101/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3325 - accuracy: 0.8967 -
val_loss: 0.6317 - val_accuracy: 0.8115
Epoch 102/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3296 - accuracy: 0.8940 -
val_loss: 0.6385 - val_accuracy: 0.8058
Epoch 103/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3752 - accuracy: 0.8822 -
val_loss: 0.5690 - val_accuracy: 0.8404
Epoch 104/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3127 - accuracy: 0.9086 -
val_loss: 0.6662 - val_accuracy: 0.8221
Epoch 105/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3188 - accuracy: 0.9001 -
val_loss: 0.6718 - val_accuracy: 0.8154
Epoch 106/200
185/185 [==============================] - 7s 38ms/step - loss: 0.3158 - accuracy: 0.8993 -
val_loss: 0.6637 - val_accuracy: 0.8087
Epoch 107/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2993 - accuracy: 0.9067 -
val_loss: 0.8665 - val_accuracy: 0.7500
Epoch 108/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3087 - accuracy: 0.9040 -
val_loss: 0.6094 - val_accuracy: 0.8202
Epoch 109/200
185/185 [==============================] - 7s 39ms/step - loss: 0.3010 - accuracy: 0.9062 -
val_loss: 0.5994 - val_accuracy: 0.8423
Epoch 110/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2969 - accuracy: 0.9066 -
val_loss: 0.6212 - val_accuracy: 0.8250
Epoch 111/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2981 - accuracy: 0.9057 -
val_loss: 0.5845 - val_accuracy: 0.8308
Epoch 112/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2799 - accuracy: 0.9076 -
val_loss: 0.7031 - val_accuracy: 0.8115
Epoch 113/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2912 - accuracy: 0.9093 -
val_loss: 0.5691 - val_accuracy: 0.8481
Epoch 114/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2789 - accuracy: 0.9157 -
val_loss: 0.6623 - val_accuracy: 0.8260
Epoch 115/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2925 - accuracy: 0.9095 -
val_loss: 0.6653 - val_accuracy: 0.8269
Epoch 116/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2851 - accuracy: 0.9140 -
val_loss: 0.7601 - val_accuracy: 0.7962
Epoch 117/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2653 - accuracy: 0.9169 -
val_loss: 0.5976 - val_accuracy: 0.8404
Epoch 118/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2701 - accuracy: 0.9208 -
val_loss: 0.6571 - val_accuracy: 0.8192
Epoch 119/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2780 - accuracy: 0.9122 -
val_loss: 0.7117 - val_accuracy: 0.8019
Epoch 120/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2715 - accuracy: 0.9129 -
val_loss: 0.6747 - val_accuracy: 0.8106
Epoch 121/200
```

```
185/185 [==============================] - 7s 39ms/step - loss: 0.2720 - accuracy: 0.9157 -
val_loss: 0.5825 - val_accuracy: 0.8375
Epoch 122/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2434 - accuracy: 0.9237 -
val_loss: 0.6496 - val_accuracy: 0.8173
Epoch 123/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2529 - accuracy: 0.9195 -
val_loss: 0.7685 - val_accuracy: 0.8058
Epoch 124/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2800 - accuracy: 0.9130 -
val_loss: 0.7392 - val_accuracy: 0.8125
Epoch 125/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2584 - accuracy: 0.9168 -
val_loss: 0.7034 - val_accuracy: 0.8192
Epoch 126/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2601 - accuracy: 0.9164 -
val_loss: 0.7650 - val_accuracy: 0.7865
Epoch 127/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2626 - accuracy: 0.9156 -
val_loss: 0.6482 - val_accuracy: 0.8279
Epoch 128/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2530 - accuracy: 0.9223 -
val_loss: 0.7802 - val_accuracy: 0.7962
Epoch 129/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2755 - accuracy: 0.9147 -
val_loss: 0.7629 - val_accuracy: 0.8077
Epoch 130/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2462 - accuracy: 0.9249 -
val_loss: 0.7614 - val_accuracy: 0.8038
Epoch 131/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2380 - accuracy: 0.9242 -
val_loss: 0.7683 - val_accuracy: 0.8019
Epoch 132/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2252 - accuracy: 0.9315 -
val_loss: 0.6250 - val_accuracy: 0.8404
Epoch 133/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2450 - accuracy: 0.9210 -
val_loss: 0.7229 - val_accuracy: 0.8115
Epoch 134/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2307 - accuracy: 0.9271 -
val_loss: 0.7654 - val_accuracy: 0.8125
Epoch 135/200
185/185 [==============================] - 7s 41ms/step - loss: 0.2320 - accuracy: 0.9235 -
val_loss: 0.8556 - val_accuracy: 0.7769
Epoch 136/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2405 - accuracy: 0.9213 -
val_loss: 0.6965 - val_accuracy: 0.8087
Epoch 137/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2277 - accuracy: 0.9323 -
val_loss: 0.7123 - val_accuracy: 0.8212
Epoch 138/200
185/185 [==============================] - 7s 40ms/step - loss: 0.2350 - accuracy: 0.9283 -
val_loss: 0.7450 - val_accuracy: 0.8029
Epoch 139/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2154 - accuracy: 0.9352 -
val_loss: 0.7582 - val_accuracy: 0.7904
Epoch 140/200
185/185 [==============================] - 7s 40ms/step - loss: 0.2338 - accuracy: 0.9317 -
val_loss: 0.6779 - val_accuracy: 0.8260
Epoch 141/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2004 - accuracy: 0.9344 -
val_loss: 0.6465 - val_accuracy: 0.8452
Epoch 142/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2043 - accuracy: 0.9349 -
val_loss: 0.6975 - val_accuracy: 0.8125
Epoch 143/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2065 - accuracy: 0.9351 -
val_loss: 0.6595 - val_accuracy: 0.8385
Epoch 144/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2095 - accuracy: 0.9351 -
val_loss: 0.8784 - val_accuracy: 0.7913
Epoch 145/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1927 - accuracy: 0.9417 -
val_loss: 0.8297 - val_accuracy: 0.8029
```

```
Epoch 146/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2291 - accuracy: 0.9291 -
val_loss: 0.7247 - val_accuracy: 0.8279
Epoch 147/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2030 - accuracy: 0.9386 -
val_loss: 0.7440 - val_accuracy: 0.8298
Epoch 148/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2282 - accuracy: 0.9276 -
val_loss: 0.8123 - val_accuracy: 0.8000
Epoch 149/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2054 - accuracy: 0.9359 -
val_loss: 0.7994 - val_accuracy: 0.8163
Epoch 150/200
185/185 [==============================] - 7s 39ms/step - loss: 0.2081 - accuracy: 0.9362 -
val_loss: 0.7385 - val_accuracy: 0.8317
Epoch 151/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1935 - accuracy: 0.9408 -
val_loss: 0.9211 - val_accuracy: 0.8067
Epoch 152/200
185/185 [==============================] - 7s 39ms/step - loss: 0.1990 - accuracy: 0.9410 -
val_loss: 0.7150 - val_accuracy: 0.8288
Epoch 153/200
185/185 [==============================] - 7s 40ms/step - loss: 0.2020 - accuracy: 0.9330 -
val_loss: 0.8248 - val_accuracy: 0.8202
Epoch 154/200
185/185 [==============================] - 8s 41ms/step - loss: 0.1940 - accuracy: 0.9390 -
val_loss: 0.7000 - val_accuracy: 0.8337
Epoch 155/200
185/185 [==============================] - 8s 41ms/step - loss: 0.2008 - accuracy: 0.9405 -
val_loss: 0.6773 - val_accuracy: 0.8317
Epoch 156/200
185/185 [==============================] - 8s 41ms/step - loss: 0.1793 - accuracy: 0.9442 -
val_loss: 0.7745 - val_accuracy: 0.8250
Epoch 157/200
185/185 [==============================] - 7s 39ms/step - loss: 0.1979 - accuracy: 0.9356 -
val_loss: 1.0454 - val_accuracy: 0.7558
Epoch 158/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2061 - accuracy: 0.9371 -
val_loss: 0.7147 - val_accuracy: 0.8375
Epoch 159/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1690 - accuracy: 0.9498 -
val_loss: 0.7038 - val_accuracy: 0.8288
Epoch 160/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1767 - accuracy: 0.9461 -
val_loss: 0.8572 - val_accuracy: 0.8019
Epoch 161/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1905 - accuracy: 0.9422 -
val_loss: 0.6826 - val_accuracy: 0.8250
Epoch 162/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1883 - accuracy: 0.9413 -
val_loss: 0.8330 - val_accuracy: 0.8087
Epoch 163/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1717 - accuracy: 0.9486 -
val_loss: 0.7523 - val_accuracy: 0.8202
Epoch 164/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1935 - accuracy: 0.9422 -
val_loss: 0.7055 - val_accuracy: 0.8462
Epoch 165/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1726 - accuracy: 0.9468 -
val_loss: 0.6793 - val_accuracy: 0.8404
Epoch 166/200
185/185 [==============================] - 7s 39ms/step - loss: 0.1740 - accuracy: 0.9452 -
val_loss: 0.9737 - val_accuracy: 0.7894
Epoch 167/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1819 - accuracy: 0.9422 -
val_loss: 0.7510 - val_accuracy: 0.8317
Epoch 168/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1826 - accuracy: 0.9418 -
val_loss: 0.7585 - val_accuracy: 0.8260
Epoch 169/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1684 - accuracy: 0.9449 -
val_loss: 0.7918 - val_accuracy: 0.8250
Epoch 170/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1857 - accuracy: 0.9408 -
```

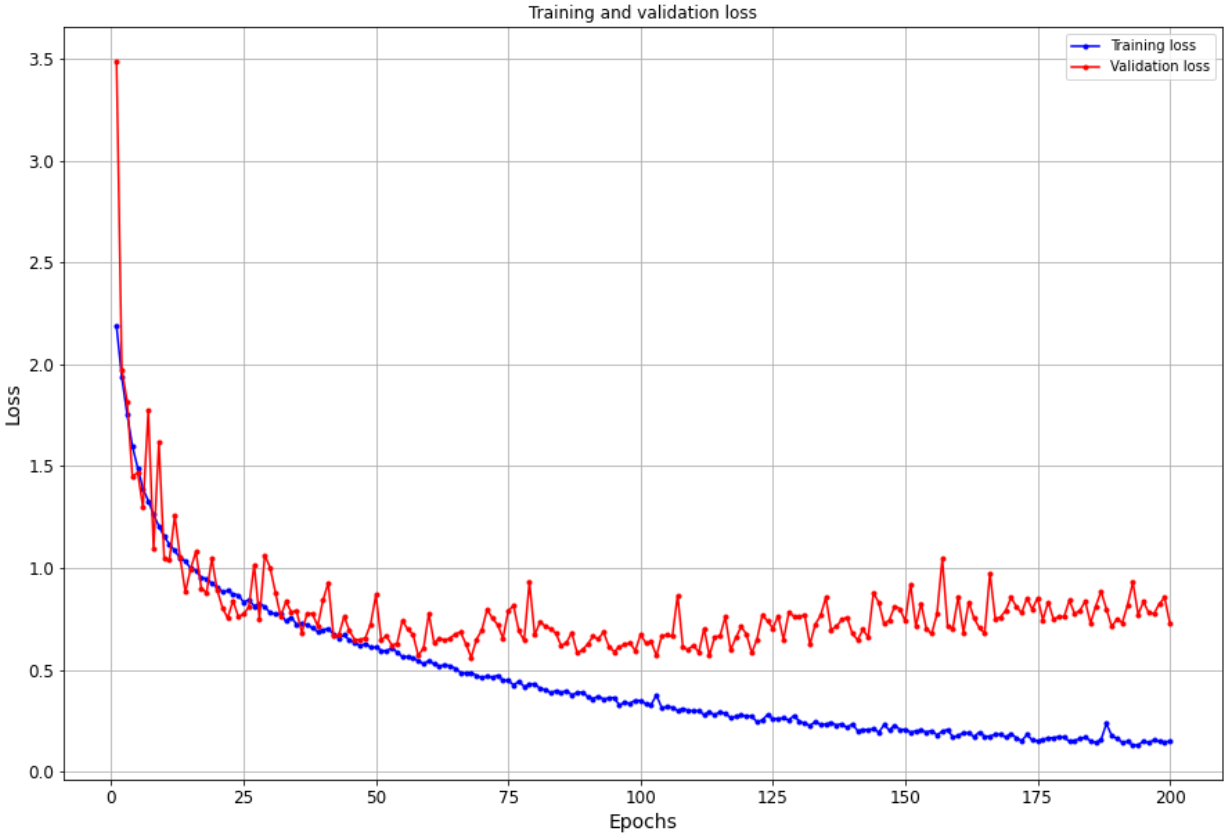
```
val_loss: 0.8549 - val_accuracy: 0.8135
Epoch 171/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1652 - accuracy: 0.9517 -
val_loss: 0.8092 - val_accuracy: 0.8212
Epoch 172/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1516 - accuracy: 0.9544 -
val_loss: 0.7800 - val_accuracy: 0.8404
Epoch 173/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1811 - accuracy: 0.9440 -
val_loss: 0.8512 - val_accuracy: 0.8106
Epoch 174/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1545 - accuracy: 0.9535 -
val_loss: 0.7965 - val_accuracy: 0.7990
Epoch 175/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1512 - accuracy: 0.9515 -
val_loss: 0.8537 - val_accuracy: 0.8279
Epoch 176/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1585 - accuracy: 0.9522 -
val_loss: 0.7448 - val_accuracy: 0.8346
Epoch 177/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1638 - accuracy: 0.9496 -
val_loss: 0.8331 - val_accuracy: 0.8327
Epoch 178/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1664 - accuracy: 0.9486 -
val_loss: 0.7492 - val_accuracy: 0.8337
Epoch 179/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1709 - accuracy: 0.9471 -
val_loss: 0.7598 - val_accuracy: 0.8337
Epoch 180/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1682 - accuracy: 0.9478 -
val_loss: 0.7596 - val_accuracy: 0.8433
Epoch 181/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1488 - accuracy: 0.9534 -
val_loss: 0.8411 - val_accuracy: 0.8260
Epoch 182/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1531 - accuracy: 0.9542 -
val_loss: 0.7745 - val_accuracy: 0.8423
Epoch 183/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1608 - accuracy: 0.9493 -
val_loss: 0.7912 - val_accuracy: 0.8173
Epoch 184/200
185/185 [==============================] - 8s 41ms/step - loss: 0.1677 - accuracy: 0.9507 -
val_loss: 0.8367 - val_accuracy: 0.8154
Epoch 185/200
185/185 [==============================] - 8s 41ms/step - loss: 0.1516 - accuracy: 0.9517 -
val_loss: 0.7287 - val_accuracy: 0.8346
Epoch 186/200
185/185 [==============================] - 8s 41ms/step - loss: 0.1418 - accuracy: 0.9568 -
val_loss: 0.8128 - val_accuracy: 0.8096
Epoch 187/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1566 - accuracy: 0.9490 -
val_loss: 0.8848 - val_accuracy: 0.8106
Epoch 188/200
185/185 [==============================] - 7s 38ms/step - loss: 0.2359 - accuracy: 0.9293 -
val_loss: 0.7993 - val_accuracy: 0.8125
Epoch 189/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1763 - accuracy: 0.9437 -
val_loss: 0.7120 - val_accuracy: 0.8365
Epoch 190/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1622 - accuracy: 0.9496 -
val_loss: 0.7484 - val_accuracy: 0.8327
Epoch 191/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1437 - accuracy: 0.9544 -
val_loss: 0.7308 - val_accuracy: 0.8385
Epoch 192/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1481 - accuracy: 0.9529 -
val_loss: 0.8170 - val_accuracy: 0.8231
Epoch 193/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1295 - accuracy: 0.9585 -
val_loss: 0.9294 - val_accuracy: 0.8067
Epoch 194/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1294 - accuracy: 0.9619 -
val_loss: 0.7718 - val_accuracy: 0.8327
Epoch 195/200
```

```
185/185 [==============================] - 7s 38ms/step - loss: 0.1501 - accuracy: 0.9530 -
val_loss: 0.8341 - val_accuracy: 0.8365
Epoch 196/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1411 - accuracy: 0.9549 -
val_loss: 0.7799 - val_accuracy: 0.8567
Epoch 197/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1561 - accuracy: 0.9527 -
val_loss: 0.7753 - val_accuracy: 0.8365
Epoch 198/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1474 - accuracy: 0.9569 -
val_loss: 0.8242 - val_accuracy: 0.8298
Epoch 199/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1464 - accuracy: 0.9534 -
val_loss: 0.8546 - val_accuracy: 0.8115
Epoch 200/200
185/185 [==============================] - 7s 38ms/step - loss: 0.1477 - accuracy: 0.9527 -
val_loss: 0.7284 - val_accuracy: 0.8356
```

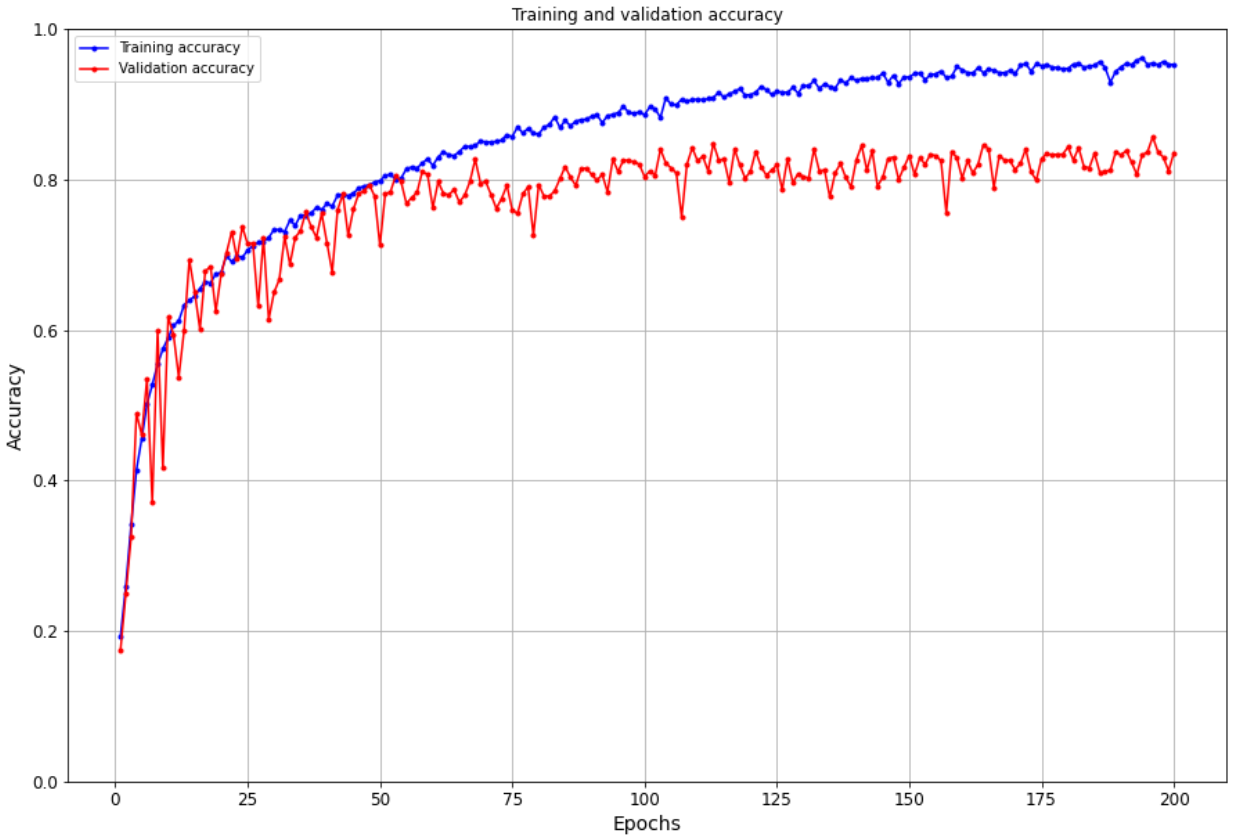
In [ ]:

```
plot_loss_and_accuracy(history)
```

```
accuracy    0.961851
dtype: float64 val_accuracy    0.856731
dtype: float64
```




```
<Figure size 432x288 with 0 Axes>
```

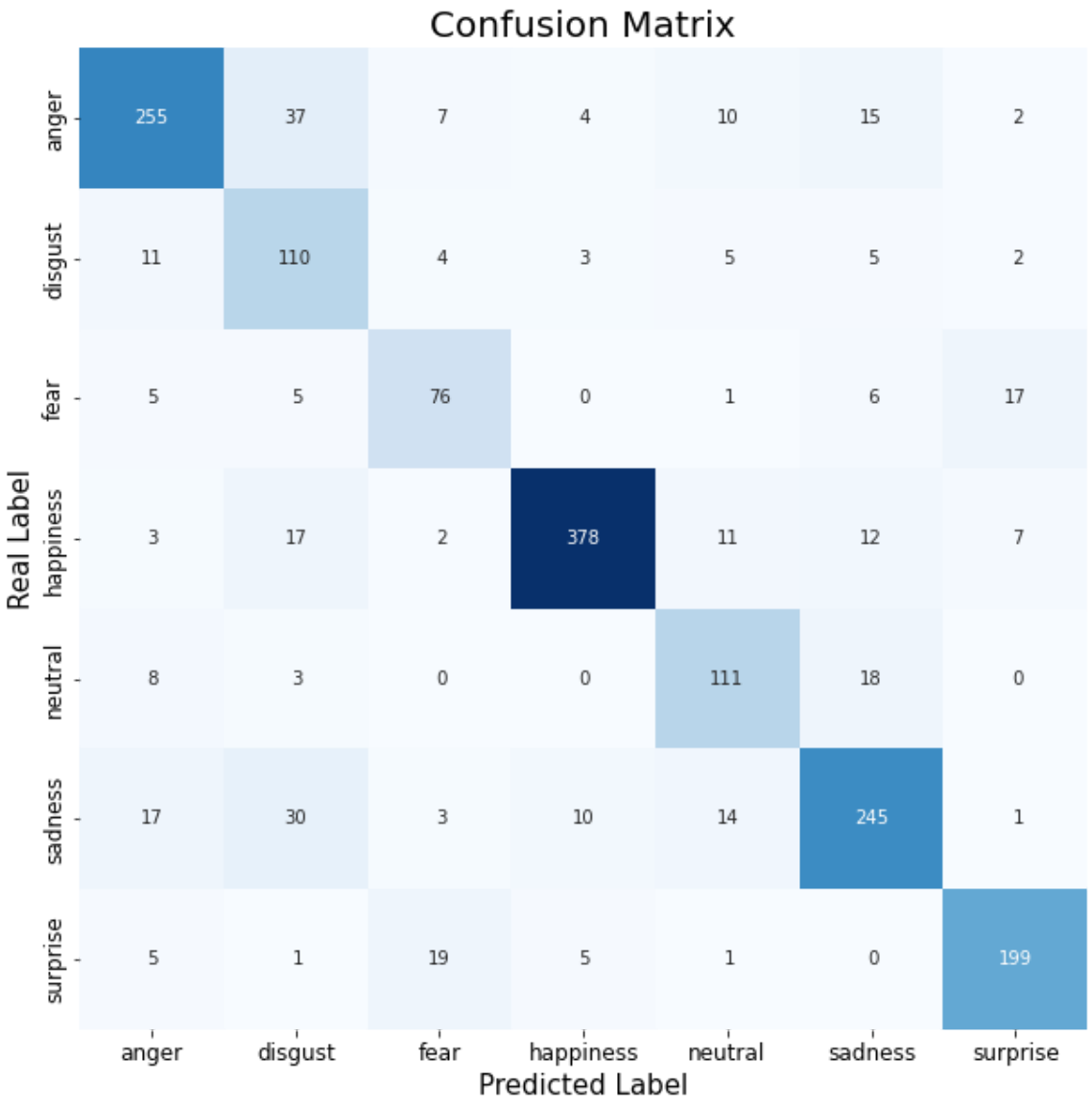


In [ ]:
```
modelWithAugmentation.evaluate(test_ds)
```

```
54/54 [==============================] - 1s 16ms/step - loss: 0.9021 - accuracy: 0.8082
```

Out[ ]:
```
[0.9020760655403137, 0.8082352876663208]
```

In [ ]:
```
y_pred = modelWithAugmentation.predict(test_ds)
cnf_matrix = plot_confusion_matrix(y_pred, y_test, class_names, return_m=True)
```



In [ ]:

```
print_confusion_matrix_metrics(cnf_matrix)
```

```
False Positives:

        anger -> 49
        disgust -> 93
        fear -> 35
        happiness -> 22
        neutral -> 42
        sadness -> 56
        surprise -> 29

False Negatives:

        anger -> 75
        disgust -> 30
        fear -> 34
        happiness -> 52
        neutral -> 29
        sadness -> 75
        surprise -> 31

True Positives:

        anger -> 255
        disgust -> 110
        fear -> 76
        happiness -> 378
        neutral -> 111
        sadness -> 245
```

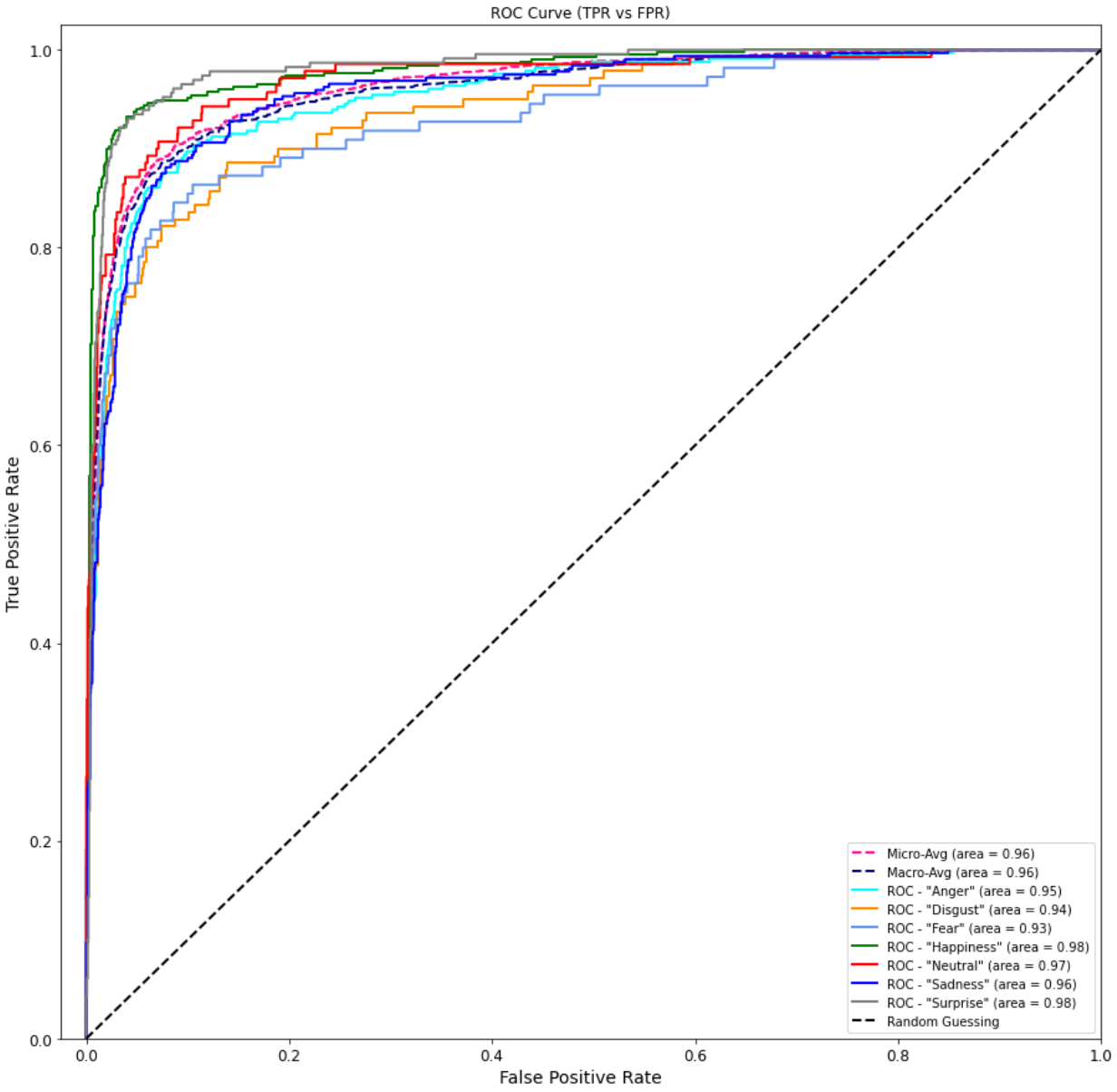
```
        surprise -> 199

True Negatives:

        anger -> 1321
        disgust -> 1467
        fear -> 1555
        happiness -> 1248
        neutral -> 1518
        sadness -> 1324
        surprise -> 1441

True Positive Rate:

        anger -> 0.773%
        disgust -> 0.786%
        fear -> 0.691%
        happiness -> 0.879%
        neutral -> 0.793%
        sadness -> 0.766%
        surprise -> 0.865%

True Negative Rate:

        anger -> 0.964%
        disgust -> 0.940%
        fear -> 0.978%
        happiness -> 0.983%
        neutral -> 0.973%
        sadness -> 0.959%
        surprise -> 0.980%

Positive Predictive Value:

        anger -> 1
        disgust -> 1
        fear -> 1
        happiness -> 1
        neutral -> 1
        sadness -> 1
        surprise -> 1

Negative Predictive Value:

        anger -> 1
        disgust -> 1
        fear -> 1
        happiness -> 1
        neutral -> 1
        sadness -> 1
        surprise -> 1

False Positive Rate:

        anger -> 0.036%
        disgust -> 0.060%
        fear -> 0.022%
        happiness -> 0.017%
        neutral -> 0.027%
        sadness -> 0.041%
        surprise -> 0.020%

False Negative Rate:

        anger -> 0.227%
        disgust -> 0.214%
        fear -> 0.309%
        happiness -> 0.121%
        neutral -> 0.207%
        sadness -> 0.234%
        surprise -> 0.135%

False Discovery Rate:
```

```
        anger -> 0.161%
        disgust -> 0.458%
        fear -> 0.315%
        happiness -> 0.055%
        neutral -> 0.275%
        sadness -> 0.186%
        surprise -> 0.127%

Accuracy:

        anger -> 0.927%
        disgust -> 0.928%
        fear -> 0.959%
        happiness -> 0.956%
        neutral -> 0.958%
        sadness -> 0.923%
        surprise -> 0.965%
```

In [ ]:
```
print_general_metrics(y_pred, y_test)
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.77      0.80       330
           1       0.54      0.79      0.64       140
           2       0.68      0.69      0.69       110
           3       0.94      0.88      0.91       430
           4       0.73      0.79      0.76       140
           5       0.81      0.77      0.79       320
           6       0.87      0.87      0.87       230

    accuracy                           0.81      1700
   macro avg       0.77      0.79      0.78      1700
weighted avg       0.82      0.81      0.81      1700


Accuracy: 0.808

Balanced accuracy: 0.793

F1_Score: 0.812

Precision: 0.822

Recall: 0.808

ROC AUC (OVR): 0.959

ROC AUC (OVO): 0.958
```

In [ ]:
```
plot_multiclass_roc_curve(y_test, y_pred, class_names)
```




```
One-vs-One ROC AUC scores:
        0.958685 (macro),
        0.963208 (weighted by prevalence)
One-vs-Rest ROC AUC scores:
        0.958685 (macro),
        0.963208 (weighted by prevalence)
```

In [ ]:

```
# Start TensorBoard
%load_ext tensorboard
# --port=6006
# --host=127.0.0.1
%tensorboard --logdir=./tensorboard_logs_CNN_V1_Rafael --host=127.0.0.1
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 24208), started 1:03:38 ago. (Use '!kill 24208' to ki
ll it.)
```

Apply trained model to new images

In [ ]:
```
from keras.models import load_model
model = load_model(best_model_file_path)
model.evaluate(test_ds)
```

```
54/54 [==============================] - 1s 10ms/step - loss: 0.8130 - accuracy: 0.8382
```

Out[ ]:
```
[0.8130242824554443, 0.8382353186607361]
```

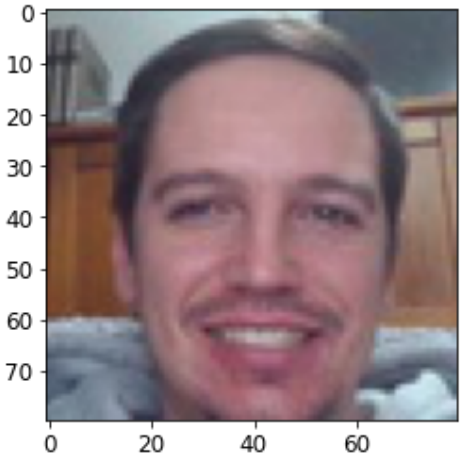
In [ ]:
```
from keras.preprocessing import image

def predict_new_image(image_path, model, class_names):
    img = image.load_img(image_path, target_size=(80,80))
    img_as_array = image.img_to_array(img)
    plt.imshow( img_as_array / 255)
```

```
    x=img_as_array
    x= x/ 255.
    x=np.expand_dims(x, axis=0)
    images = np.vstack([x])

    classes = model.predict(images, batch_size=16)

    print(classes.argmax(axis=-1))
    print(class_names)
    print(np.array(class_names)[classes.argmax(axis=-1)])
```

In [ ]:

```
predict_new_image("im1.jpg", model, class_names)
```

```
[3]
['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise']
['happiness']
```



In [ ]:

```
predict_new_image("im2.jpg", model, class_names)
```

```
[5]
['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise']
['sadness']
```