```
import os
import glob
import shutil
import random
os.chdir('/Users/batu/Desktop/DeepLearningProject/data/happy-vs-
angry')
if  os.path.isdir('test/angry') is False:
    os.makedirs('test/angry')
    os.makedirs('test/happy')

for i in random.sample(glob.glob('train/angry/*'), 1000):
    shutil.move(i, 'test/angry')
for i in random.sample(glob.glob('train/happy/*'), 1000):
    shutil.move(i, 'test/happy')

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_directory = '/Users/batu/Desktop/DeepLearningProject/data/happy-
vs-angry/train'
validation_directory =
'/Users/batu/Desktop/DeepLearningProject/data/happy-vs-angry/validatio
n'
test_directory = '/Users/batu/Desktop/DeepLearningProject/data/happy-
vs-angry/test'

IMAGE_SIZE = (48, 48)
BATCH_SIZE = 10
COLOR_MODE = 'grayscale'

# Seed for reproducibility how can we decide this value?
SEED = None

train_set = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2, horizontal_flip=True, rotation_range=20,
width_shift_range=0.2, height_shift_range=0.2, fill_mode='reflect')
valid_set = ImageDataGenerator(rescale=1./255)
test_set = ImageDataGenerator(rescale=1./255)

train_batch = train_set.flow_from_directory(
    directory = train_directory,
    target_size=IMAGE_SIZE,
    color_mode=COLOR_MODE,
    batch_size=BATCH_SIZE,
    classes=['angry', 'happy'],
)
valid_batch = valid_set.flow_from_directory(
    directory = validation_directory,
    target_size=IMAGE_SIZE,
```

```
    color_mode=COLOR_MODE,
    batch_size=BATCH_SIZE,
    classes=['angry', 'happy'],
)
test_batch = test_set.flow_from_directory(
    directory = test_directory,
    target_size=IMAGE_SIZE,
    color_mode=COLOR_MODE,
    batch_size=BATCH_SIZE,
    classes=['angry', 'happy'],
)
```

```
Found 9157 images belonging to 2 classes.
Found 2785 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

```
assert train_batch.n == 9157
assert valid_batch.n == 2785
assert test_batch.n == 2000
assert train_batch.num_classes == valid_batch.num_classes ==
test_batch.num_classes == 2

import matplotlib.pyplot as plt

imgs, labels = next(train_batch)

def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

plotImages(imgs)
print(labels)
```

```
[[1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
```

```
 [0. 1.]
 [0. 1.]]
```

```python
print(imgs.shape) # (10, 48, 48, 1) 10 images, 48x48 pixels, 1 channel
(grayscale)
```

```
(10, 48, 48, 1)
```

```python
'''
def enhanced_data_augmentation(image, training):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    image = tf.image.random_brightness(image, max_delta=0.5)
    image = tf.image.random_contrast(image, lower=0.2, upper=2.0)
    if training:
        image = tf.image.random_rotation(image,
angles=tf.random.uniform(shape=[], minval=-0.2, maxval=0.2),
fill_mode='reflect')
        image = tf.image.random_zoom(image, zoom_range=(0.8, 1.2),
fill_mode='reflect')
    image = tf.clip_by_value(image, 0, 1)
    return image
'''


from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam

no_of_classes = 2

model = Sequential()

# 1st CNN layer
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


# 2nd CNN layer
model.add(Conv2D(128, (5, 5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
# 3rd CNN layer
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))



# 4th CNN layer
model.add(Conv2D(512, (3, 3) , padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


model.add(Flatten())

# Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))



# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))



model.add(Dense(units=2, activation='softmax'))

opt = Adam(learning_rate=0.0001)  # Note the change from lr to
learning_rate
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the mdoel tomorrow
history = model.fit(x= train_batch, validation_data= valid_batch,
epochs=50, verbose= 2)
```

```
Epoch 1/50
916/916 - 73s - loss: 0.7626 - accuracy: 0.6159 - val_loss: 0.6370 -
val_accuracy: 0.6650 - 73s/epoch - 80ms/step
Epoch 2/50
916/916 - 73s - loss: 0.7180 - accuracy: 0.6242 - val_loss: 0.6287 -
```

```
val_accuracy: 0.6636 - 73s/epoch - 80ms/step
Epoch 3/50
916/916 - 75s - loss: 0.7033 - accuracy: 0.6232 - val_loss: 0.6274 -
val_accuracy: 0.6610 - 75s/epoch - 82ms/step
Epoch 4/50
916/916 - 75s - loss: 0.6801 - accuracy: 0.6422 - val_loss: 0.6231 -
val_accuracy: 0.6603 - 75s/epoch - 82ms/step
Epoch 5/50
916/916 - 75s - loss: 0.6758 - accuracy: 0.6363 - val_loss: 0.6318 -
val_accuracy: 0.6589 - 75s/epoch - 81ms/step
Epoch 6/50
916/916 - 75s - loss: 0.6637 - accuracy: 0.6414 - val_loss: 0.6211 -
val_accuracy: 0.6657 - 75s/epoch - 82ms/step
Epoch 7/50
916/916 - 76s - loss: 0.6518 - accuracy: 0.6505 - val_loss: 0.6301 -
val_accuracy: 0.6707 - 76s/epoch - 82ms/step
Epoch 8/50
916/916 - 78s - loss: 0.6339 - accuracy: 0.6651 - val_loss: 0.6119 -
val_accuracy: 0.6768 - 78s/epoch - 85ms/step
Epoch 9/50
916/916 - 75s - loss: 0.6188 - accuracy: 0.6747 - val_loss: 0.5457 -
val_accuracy: 0.7275 - 75s/epoch - 82ms/step
Epoch 10/50
916/916 - 77s - loss: 0.5928 - accuracy: 0.6919 - val_loss: 0.4868 -
val_accuracy: 0.7555 - 77s/epoch - 84ms/step
Epoch 11/50
916/916 - 80s - loss: 0.5560 - accuracy: 0.7156 - val_loss: 0.4263 -
val_accuracy: 0.7932 - 80s/epoch - 87ms/step
Epoch 12/50
916/916 - 79s - loss: 0.5240 - accuracy: 0.7360 - val_loss: 0.4428 -
val_accuracy: 0.7914 - 79s/epoch - 86ms/step
Epoch 13/50
916/916 - 80s - loss: 0.5058 - accuracy: 0.7477 - val_loss: 0.4122 -
val_accuracy: 0.7993 - 80s/epoch - 87ms/step
Epoch 14/50
916/916 - 80s - loss: 0.4910 - accuracy: 0.7571 - val_loss: 0.3702 -
val_accuracy: 0.8327 - 80s/epoch - 88ms/step
Epoch 15/50
916/916 - 80s - loss: 0.4674 - accuracy: 0.7735 - val_loss: 0.3370 -
val_accuracy: 0.8460 - 80s/epoch - 88ms/step
Epoch 16/50
916/916 - 78s - loss: 0.4618 - accuracy: 0.7769 - val_loss: 0.3265 -
val_accuracy: 0.8578 - 78s/epoch - 85ms/step
Epoch 17/50
916/916 - 81s - loss: 0.4475 - accuracy: 0.7827 - val_loss: 0.3327 -
val_accuracy: 0.8531 - 81s/epoch - 88ms/step
Epoch 18/50
916/916 - 79s - loss: 0.4427 - accuracy: 0.7869 - val_loss: 0.3054 -
val_accuracy: 0.8589 - 79s/epoch - 86ms/step
```

```
Epoch 19/50
916/916 - 79s - loss: 0.4363 - accuracy: 0.7969 - val_loss: 0.3484 - val_accuracy: 0.8427 - 79s/epoch - 86ms/step
Epoch 20/50
916/916 - 81s - loss: 0.4304 - accuracy: 0.7991 - val_loss: 0.4250 - val_accuracy: 0.7939 - 81s/epoch - 88ms/step
Epoch 21/50
916/916 - 80s - loss: 0.4172 - accuracy: 0.8059 - val_loss: 0.2780 - val_accuracy: 0.8772 - 80s/epoch - 87ms/step
Epoch 22/50
916/916 - 80s - loss: 0.4019 - accuracy: 0.8117 - val_loss: 0.2742 - val_accuracy: 0.8786 - 80s/epoch - 87ms/step
Epoch 23/50
916/916 - 75s - loss: 0.4025 - accuracy: 0.8146 - val_loss: 0.3125 - val_accuracy: 0.8654 - 75s/epoch - 82ms/step
Epoch 24/50
916/916 - 77s - loss: 0.4023 - accuracy: 0.8146 - val_loss: 0.3282 - val_accuracy: 0.8585 - 77s/epoch - 85ms/step
Epoch 25/50
916/916 - 76s - loss: 0.4066 - accuracy: 0.8150 - val_loss: 0.2834 - val_accuracy: 0.8783 - 76s/epoch - 83ms/step
Epoch 26/50
916/916 - 75s - loss: 0.3917 - accuracy: 0.8146 - val_loss: 0.2681 - val_accuracy: 0.8851 - 75s/epoch - 82ms/step
Epoch 27/50
916/916 - 78s - loss: 0.3917 - accuracy: 0.8207 - val_loss: 0.2434 - val_accuracy: 0.8984 - 78s/epoch - 86ms/step
Epoch 28/50
916/916 - 76s - loss: 0.3904 - accuracy: 0.8244 - val_loss: 0.2846 - val_accuracy: 0.8768 - 76s/epoch - 83ms/step
Epoch 29/50
916/916 - 74s - loss: 0.3780 - accuracy: 0.8316 - val_loss: 0.2548 - val_accuracy: 0.8955 - 74s/epoch - 81ms/step
Epoch 30/50
916/916 - 75s - loss: 0.3763 - accuracy: 0.8305 - val_loss: 0.2287 - val_accuracy: 0.9092 - 75s/epoch - 82ms/step
Epoch 31/50
916/916 - 79s - loss: 0.3713 - accuracy: 0.8346 - val_loss: 0.2910 - val_accuracy: 0.8783 - 79s/epoch - 86ms/step
Epoch 32/50
916/916 - 76s - loss: 0.3708 - accuracy: 0.8327 - val_loss: 0.2640 - val_accuracy: 0.8890 - 76s/epoch - 83ms/step
Epoch 33/50
916/916 - 76s - loss: 0.3654 - accuracy: 0.8395 - val_loss: 0.2497 - val_accuracy: 0.8991 - 76s/epoch - 83ms/step
Epoch 34/50
916/916 - 78s - loss: 0.3594 - accuracy: 0.8392 - val_loss: 0.3476 - val_accuracy: 0.8431 - 78s/epoch - 86ms/step
Epoch 35/50
```

```
916/916 - 77s - loss: 0.3592 - accuracy: 0.8437 - val_loss: 0.2180 -
val_accuracy: 0.9063 - 77s/epoch - 84ms/step
Epoch 36/50
916/916 - 77s - loss: 0.3506 - accuracy: 0.8461 - val_loss: 0.2363 -
val_accuracy: 0.8984 - 77s/epoch - 84ms/step
Epoch 37/50
916/916 - 78s - loss: 0.3492 - accuracy: 0.8439 - val_loss: 0.2465 -
val_accuracy: 0.8977 - 78s/epoch - 85ms/step
Epoch 38/50
916/916 - 79s - loss: 0.3573 - accuracy: 0.8413 - val_loss: 0.2464 -
val_accuracy: 0.9013 - 79s/epoch - 86ms/step
Epoch 39/50
916/916 - 79s - loss: 0.3446 - accuracy: 0.8504 - val_loss: 0.2354 -
val_accuracy: 0.9095 - 79s/epoch - 87ms/step
Epoch 40/50
916/916 - 78s - loss: 0.3409 - accuracy: 0.8477 - val_loss: 0.2113 -
val_accuracy: 0.9156 - 78s/epoch - 85ms/step
Epoch 41/50
916/916 - 79s - loss: 0.3399 - accuracy: 0.8564 - val_loss: 0.2243 -
val_accuracy: 0.9124 - 79s/epoch - 87ms/step
Epoch 42/50
916/916 - 77s - loss: 0.3337 - accuracy: 0.8519 - val_loss: 0.2389 -
val_accuracy: 0.9020 - 77s/epoch - 85ms/step
Epoch 43/50
916/916 - 78s - loss: 0.3351 - accuracy: 0.8548 - val_loss: 0.2778 -
val_accuracy: 0.8912 - 78s/epoch - 85ms/step
Epoch 44/50
916/916 - 77s - loss: 0.3275 - accuracy: 0.8578 - val_loss: 0.2515 -
val_accuracy: 0.8930 - 77s/epoch - 84ms/step
Epoch 45/50
916/916 - 77s - loss: 0.3267 - accuracy: 0.8556 - val_loss: 0.2238 -
val_accuracy: 0.9059 - 77s/epoch - 84ms/step
Epoch 46/50
916/916 - 74s - loss: 0.3293 - accuracy: 0.8564 - val_loss: 0.2821 -
val_accuracy: 0.8794 - 74s/epoch - 81ms/step
Epoch 47/50
916/916 - 74s - loss: 0.3181 - accuracy: 0.8653 - val_loss: 0.2236 -
val_accuracy: 0.9045 - 74s/epoch - 81ms/step
Epoch 48/50
916/916 - 73s - loss: 0.3267 - accuracy: 0.8585 - val_loss: 0.1990 -
val_accuracy: 0.9167 - 73s/epoch - 80ms/step
Epoch 49/50
916/916 - 75s - loss: 0.3183 - accuracy: 0.8650 - val_loss: 0.2105 -
val_accuracy: 0.9084 - 75s/epoch - 82ms/step
Epoch 50/50
916/916 - 78s - loss: 0.3176 - accuracy: 0.8627 - val_loss: 0.2112 -
val_accuracy: 0.9066 - 78s/epoch - 85ms/step
```

```python
test_directory = '/Users/batu/Desktop/DeepLearningProject/data/happy-vs-angry/test'
```

```
test_batch = test_set.flow_from_directory(
    directory = test_directory,
    target_size=IMAGE_SIZE,
    color_mode=COLOR_MODE,
    batch_size=BATCH_SIZE,
    classes=['angry', 'happy'],
    shuffle=False
)
imgs, labels = next(test_batch)
plotImages(imgs)
print(labels)
```

```
Found 2000 images belonging to 2 classes.
```

```
[[1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

```
print(imgs.shape)
```

```
(10, 48, 48, 1)
```

```
test_batch.classes
```

```
array([0, 0, 0, ..., 1, 1, 1], dtype=int32)
```

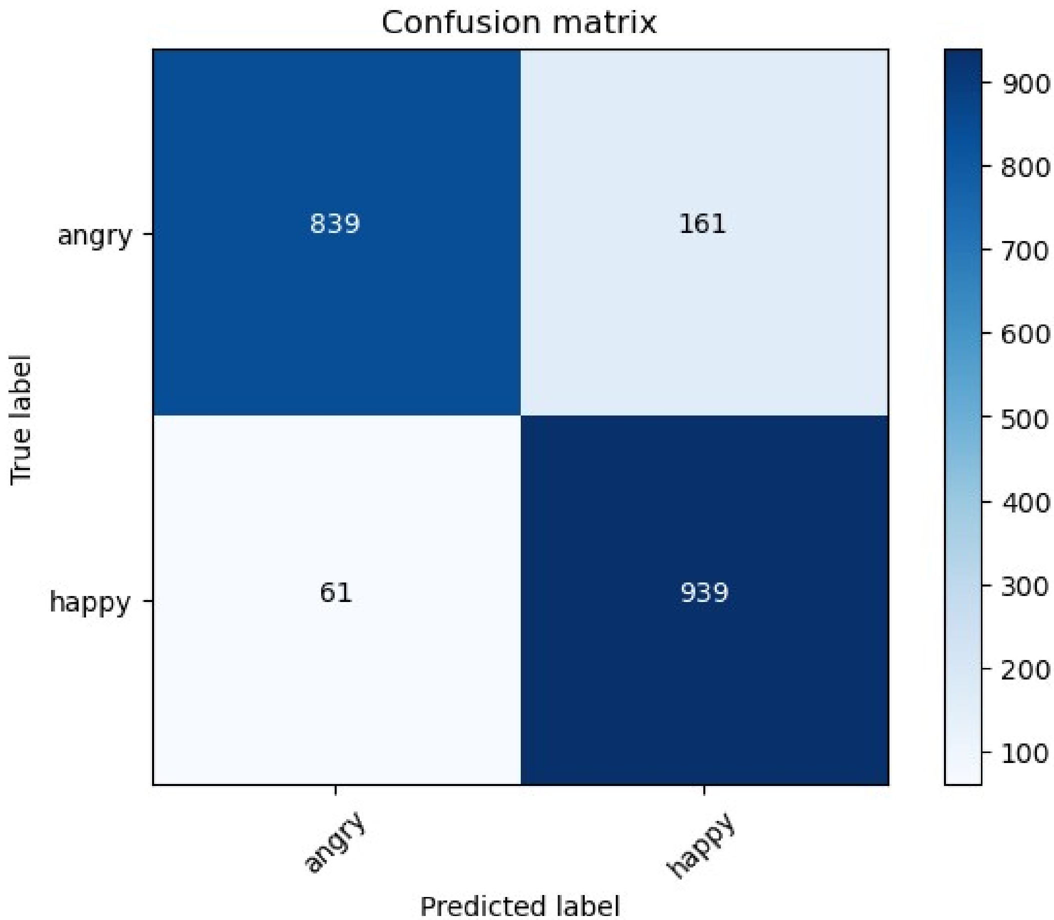
```
prediction = model.predict(x=test_batch, verbose=0)
```

```
import numpy as np
prediction = np.round(prediction)
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [1., 0.]], dtype=float32)
```

```python
# İmport the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true=test_batch.classes,
y_pred=np.argmax(prediction, axis=-1))

from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true=test_batch.classes,
y_pred=prediction.argmax(axis=-1))
# Function to plot confusion matrix
def plot_confusion_matrix(cm, classes, normalize=False,
title='Confusion matrix', cmap=plt.cm.Blues):
    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)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    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')

plot_confusion_matrix(cm, ['angry', 'happy'])
```

```
Confusion matrix, without normalization
[[839 161]
 [ 61 939]]
```

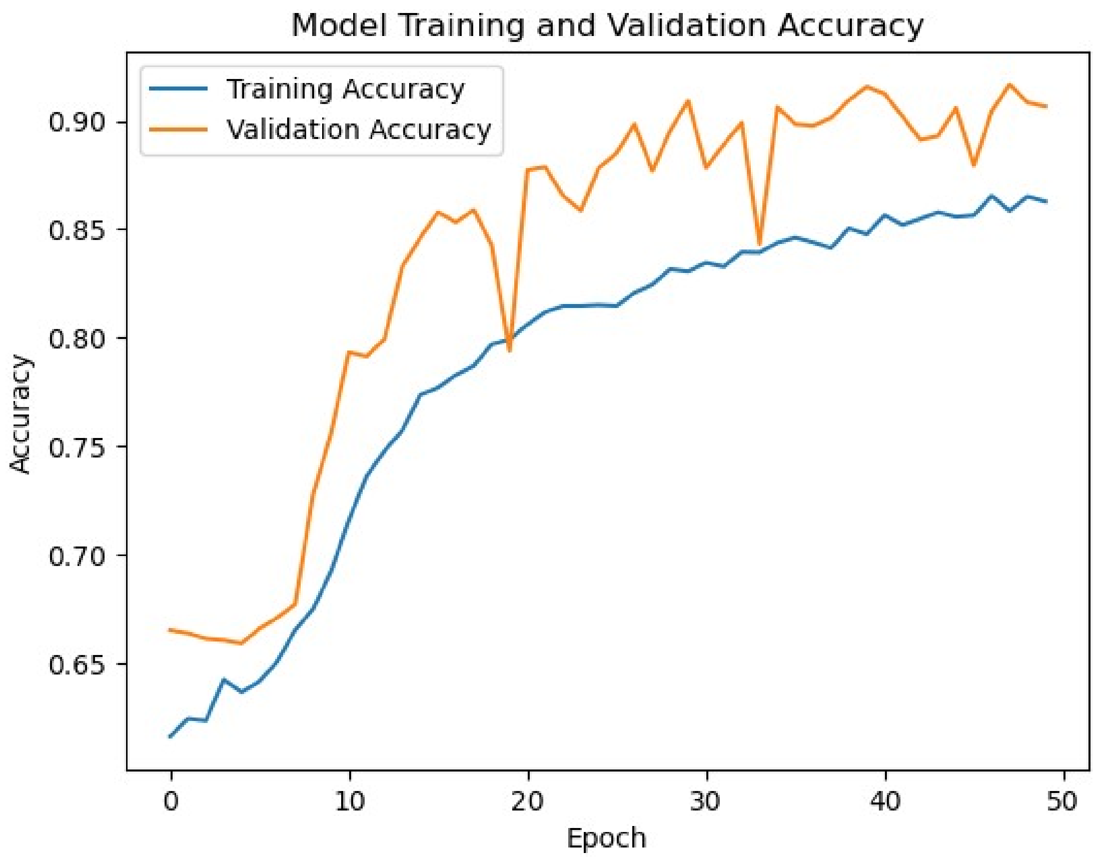



```
from sklearn.metrics import classification_report
print(classification_report(y_true=test_batch.classes,
y_pred=prediction.argmax(axis=-1)))
```

```
              precision    recall  f1-score   support

           0       0.93      0.84      0.88      1000
           1       0.85      0.94      0.89      1000

    accuracy                           0.89      2000
   macro avg       0.89      0.89      0.89      2000
weighted avg       0.89      0.89      0.89      2000
```

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend()
plt.show()
```




```

```