Rachid LAMJOUN
PG AI Program

Perform Facial Recognition with Deep Learning in Keras Using CNN

PG AI – Advanced Deep Learning and Computer Vision Project:

Project2: Perform Facial Recognition with Deep Learning in Keras Using CNN

Writeup

We have the subject of building a CNN model with a validation accuracy above 90%. The model must be able to do facial recognition with deep convolutional neural networks. The dataset in our possession is divided into two parts, one for the training of 240 images and another for the validation of 160 images which present the faces of 20 people.

```
data = np.load('/content/ORL_faces.npz')
data.files
```

```
['testY', 'testX', 'trainX', 'trainY']
```

```
data['trainX'].shape,data['trainY'].shape,
        data['testX'].shape, data['testY'].shape
```

```
((240, 10304), (240,), (160, 10304), (160,))
```

```
# Values of Target Data
print(list(set(data['trainY'])))
print(list(set(data['testY'])))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

We relied on a few main functions:

- **Resize_array():** to have the arrays corresponding to the images of dimension (100,100)
- **get_inputShape():** gives the appropriate shape for the fit()
- **get_generator():** we used the flow() function of the ImageDataGenerator class to build our image generators for the fit() function. This could be interesting in case of need for an augmentation operation
- **get_model():** here we have the definition of model. There was no overfitting to use Dropout layers:

```
def get_model(i_inputShape):
 l_model=Sequential()
 l_model.add(Input((i_inputShape)))
 l_model.add(Conv2D(128, kernel_size=(5,5),activation='relu'))
 l_model.add(MaxPooling2D(pool_size=(2,2)))
 l_model.add(Dense(128,activation='relu'))
 l_model.add(Flatten())
 #l_model.add(Dropout(0.4))
 l_model.add(Dense(20,activation='softmax'))

 return l_model
```

- **fit_model():** We used the callback function to stop the training

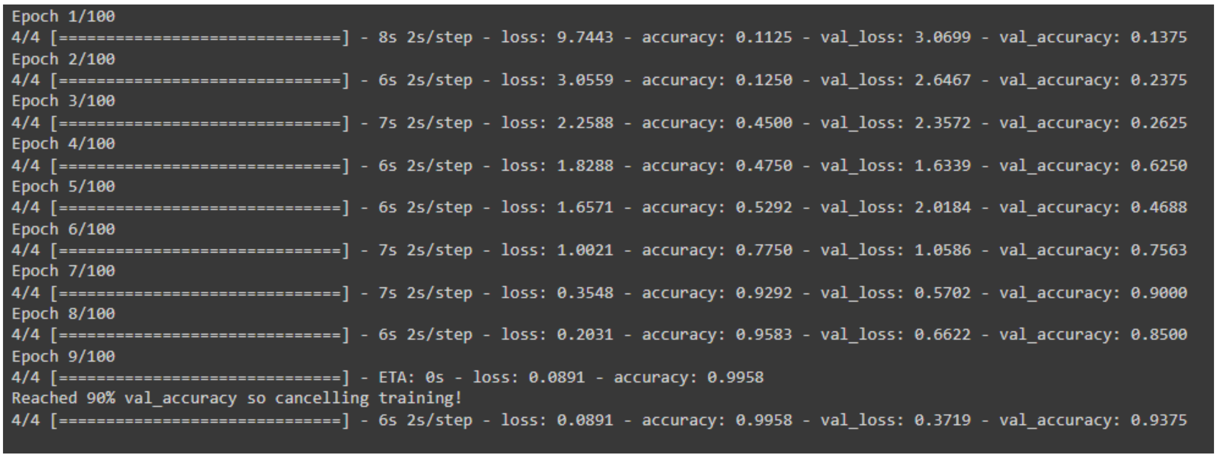
```
def fit_model(i_model, i_train_generator, i_validation_generator,
               i_num_epoch,i_lr,i_verbose=0):
 l_model=i_model
 l_model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=i_lr),
              metrics=['accuracy'])

 class myCallback(tf.keras.callbacks.Callback):
   def on_epoch_end(self, epoch, logs={}):
     if (logs.get('val_accuracy') > 0.90):
       print("\nReached 90% val_accuracy so cancelling training!")
       self.model.stop_training = True
 callbacks = myCallback()
#  validation_steps=8
 l_history= l_model.fit(
   i_train_generator,
   epochs=i_num_epoch,
   validation_data = i_validation_generator,
   callbacks=[callbacks],
   verbose=i_verbose )
 return l_model, l_history
```

- **plot_loss_acc():** The function that plots the evolution of accuracy and loss by epochs for training and validation

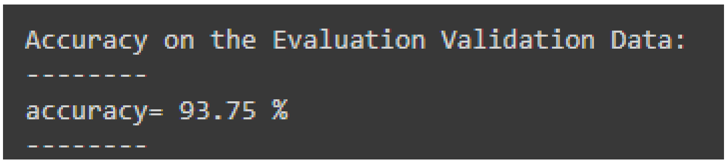
After running the training:

```
#-----------
epochs=100
#-----------
#
X_train_t=np.expand_dims(resize_array(X_train),axis=-1)
X_test_t=np.expand_dims(resize_array(X_test),axis=-1)
#
model_FacRecog,history_FacRecog=fit_model(get_model(get_inputShape(IMAGE_SHAPE)),
                              get_generator(X_train_t,y_train,datagen),
                                   get_generator(X_test_t,y_test,datagen),
                                         epochs,i_lr=0.001, i_verbose=1)

plot_loss_acc(history_FacRecog)


#--- Evaluate on Validation data
#
print('\n\nAccuracy on the Evaluation Validation Data:')
print('--------')
eval(model_FacRecog,get_generator(X_test_t,y_test,datagen))
print('--------')
```

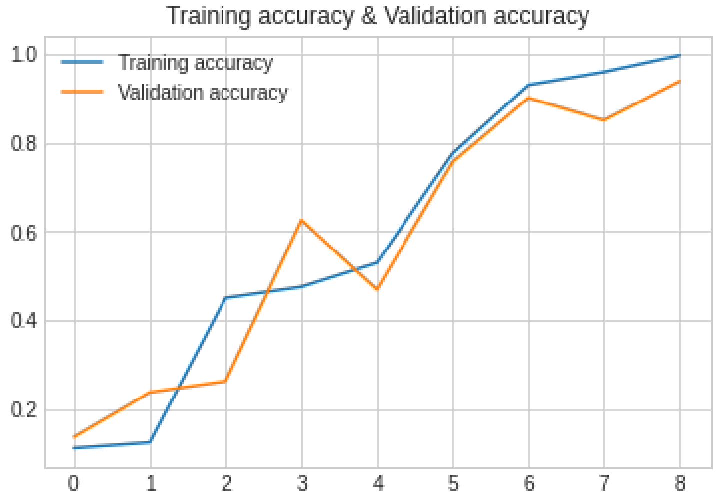
we got the following result:

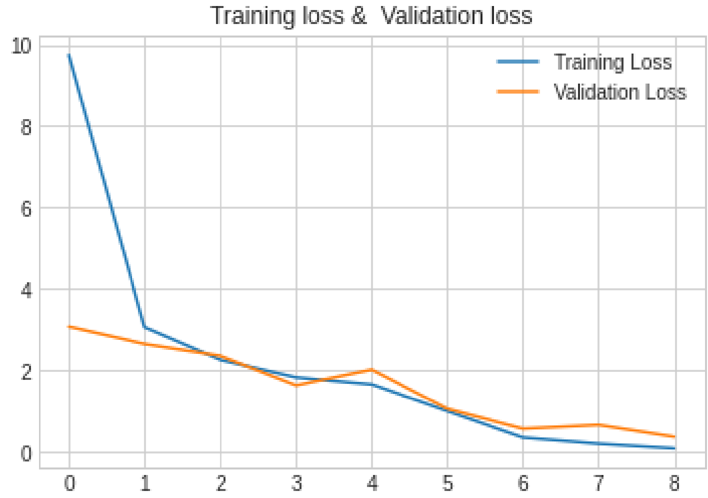


```
Epoch 1/100
4/4 [==============================] - 8s 2s/step - loss: 9.7443 - accuracy: 0.1125 - val_loss: 3.0699 - val_accuracy: 0.1375
Epoch 2/100
4/4 [==============================] - 6s 2s/step - loss: 3.0559 - accuracy: 0.1250 - val_loss: 2.6467 - val_accuracy: 0.2375
Epoch 3/100
4/4 [==============================] - 7s 2s/step - loss: 2.2588 - accuracy: 0.4500 - val_loss: 2.3572 - val_accuracy: 0.2625
Epoch 4/100
4/4 [==============================] - 6s 2s/step - loss: 1.8288 - accuracy: 0.4750 - val_loss: 1.6339 - val_accuracy: 0.6250
Epoch 5/100
4/4 [==============================] - 6s 2s/step - loss: 1.6571 - accuracy: 0.5292 - val_loss: 2.0184 - val_accuracy: 0.4688
Epoch 6/100
4/4 [==============================] - 7s 2s/step - loss: 1.0021 - accuracy: 0.7750 - val_loss: 1.0586 - val_accuracy: 0.7563
Epoch 7/100
4/4 [==============================] - 7s 2s/step - loss: 0.3548 - accuracy: 0.9292 - val_loss: 0.5702 - val_accuracy: 0.9000
Epoch 8/100
4/4 [==============================] - 6s 2s/step - loss: 0.2031 - accuracy: 0.9583 - val_loss: 0.6622 - val_accuracy: 0.8500
Epoch 9/100
4/4 [==============================] - ETA: 0s - loss: 0.0891 - accuracy: 0.9958
Reached 90% val_accuracy so cancelling training!
4/4 [==============================] - 6s 2s/step - loss: 0.0891 - accuracy: 0.9958 - val_loss: 0.3719 - val_accuracy: 0.9375
```



```
Accuracy on the Evaluation Validation Data:
--------
accuracy= 93.75 %
--------
```


Training accuracy & Validation accuracy
Training accuracy
Validation accuracy


Training loss & Validation loss
Training Loss
Validation Loss