

Assessment

Facial Recognition with Deep Learning in Keras Using CNN

DESCRIPTION

Project Objective:

Facial recognition is a biometric alternative that measures unique characteristics of a human face. Applications available today include flight check in, tagging friends and family members in photos, and “tailored” advertising. You are a computer vision engineer who needs to develop a face recognition programme with deep convolutional neural networks. Objective: Use a deep convolutional neural network to perform facial recognition using Keras.

Project Description and Scope:

ORL face database composed of 400 images of size 112 x 92. There are 40 people, 10 images per person. The images were taken at different times, lighting and facial expressions. The faces are in an upright position in frontal view, with a slight left-right rotation.

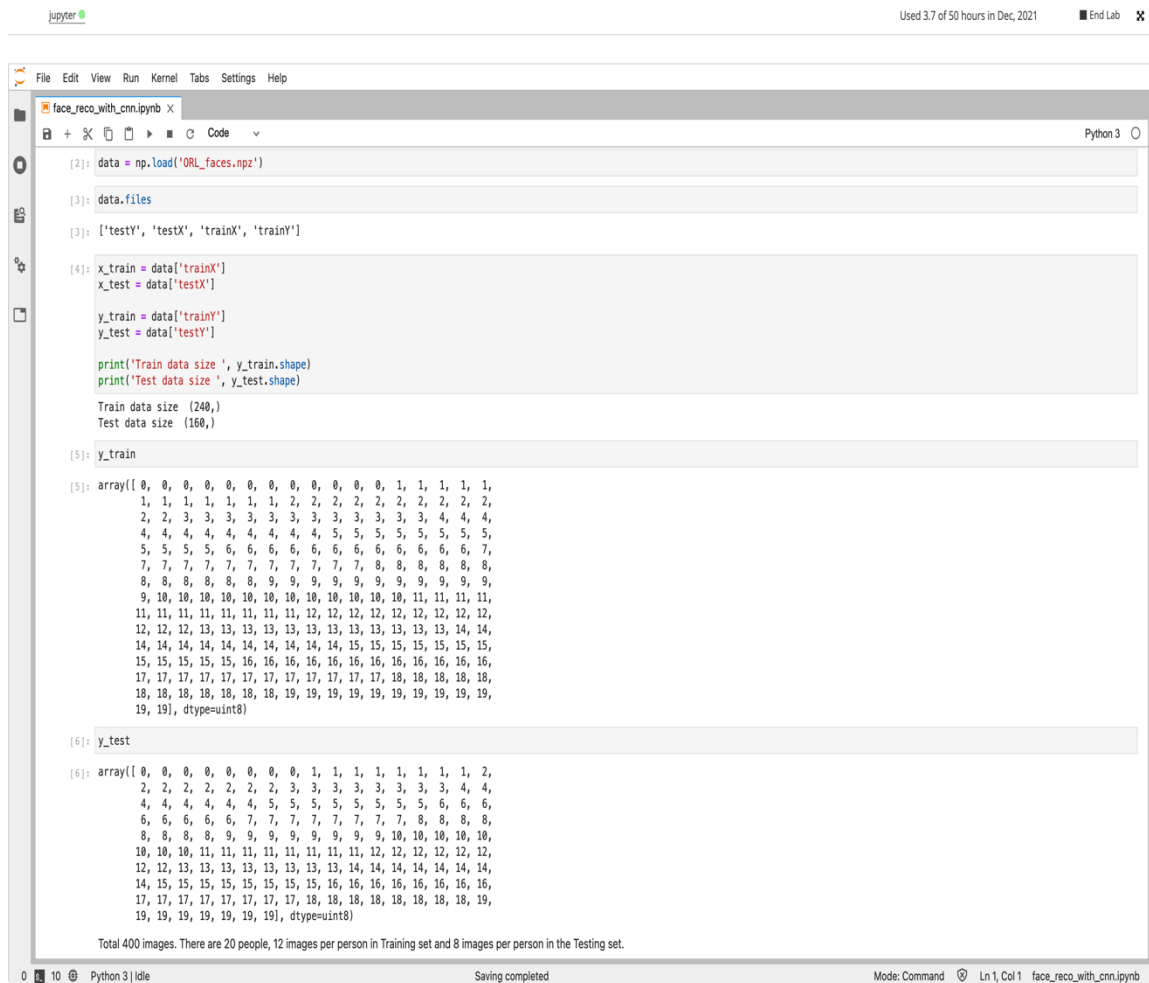
Link to the Dataset: https://www.dropbox.com/s/i7uzp5yxk7wruva/ORL_faces.npz?dl=0

Project Guidelines:

- Load the dataset after loading the dataset, you have to normalize every image.
- Transform the images to equal sizes to feed in CNN
- Build a CNN model that has 3 main layers:
 - i. Convolutional Layer
 - ii. Pooling Layer
 - iii. Fully Connected Layer
- Iterate the model until the accuracy is above 90%

Dataset Info

ORL face database composed of 400 images of size 112 x 92. There are 20 people, 12 images per person in the Train dataset and 8 images per person in Test dataset.

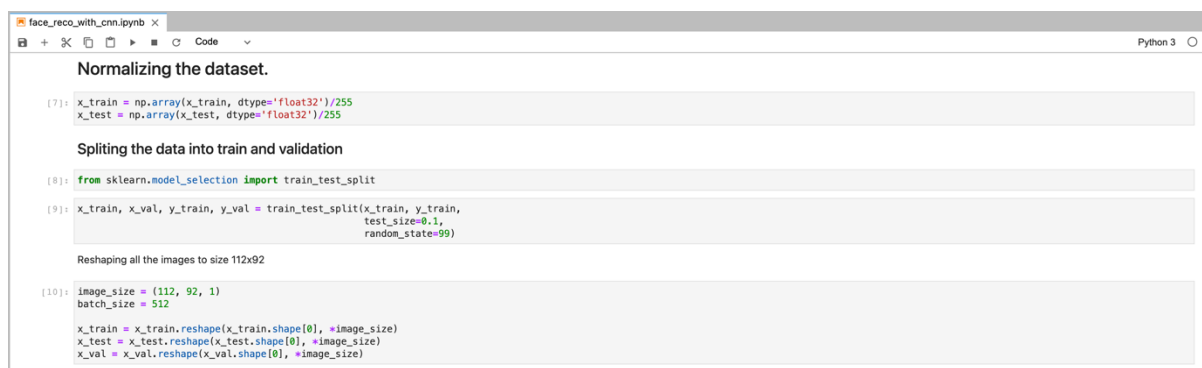


A Jupyter Notebook interface showing the loading and inspection of the ORL face dataset. The notebook is titled 'face_reco_with_cnn.ipynb'. The code cells show the following steps:

- Load the dataset: `data = np.load('ORL_faces.npz')`
- Get the files: `data.files`
- Split the data into train and test sets: `['testY', 'testX', 'trainX', 'trainY']`
- Assign variables: `x_train = data['trainX'], x_test = data['testX'], y_train = data['trainY'], y_test = data['testY']`
- Print the shapes: `print('Train data size ', y_train.shape), print('Test data size ', y_test.shape)`
- Display the first 100 elements of `y_train` and `y_test`.

The output shows the shapes: Train data size (240,) and Test data size (160,). The arrays `y_train` and `y_test` are displayed, showing a sequence of class labels (0-19) repeated 12 and 8 times respectively. A summary note at the bottom states: "Total 400 images. There are 20 people, 12 images per person in Training set and 8 images per person in the Testing set."

All the images are normalized, reshaped and Train dataset is split into Train data and Validation dataset for model training.



A Jupyter Notebook interface showing the normalization and splitting of the dataset. The notebook is titled 'face_reco_with_cnn.ipynb'. The code cells show the following steps:

- Normalizing the dataset.**
`x_train = np.array(x_train, dtype='float32')/255`
`x_test = np.array(x_test, dtype='float32')/255`
- Splitting the data into train and validation**
`from sklearn.model_selection import train_test_split`
`x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=99)`
- Reshaping all the images to size 112x92**
`image_size = (112, 92, 1)`
`batch_size = 512`
`x_train = x_train.reshape(x_train.shape[0], *image_size)`
`x_test = x_test.reshape(x_test.shape[0], *image_size)`
`x_val = x_val.reshape(x_val.shape[0], *image_size)`

Model Building

There are 4 Convolution layers with activation function as relu, 2 Max Pooling layers, 1 Flatten Layer, 2 Fully Connected Layers with activation function as relu, 2 Dropout layers and 1 Output Layer with activation function as softmax.

Sequential Model Summary

```
[13]: face_rec_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 110, 90, 36)	360
conv2d_1 (Conv2D)	(None, 108, 88, 36)	11700
max_pooling2d (MaxPooling2D)	(None, 54, 44, 36)	0
conv2d_2 (Conv2D)	(None, 52, 42, 64)	28800
conv2d_3 (Conv2D)	(None, 50, 40, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 25, 20, 64)	0
flatten (Flatten)	(None, 32000)	0
dense (Dense)	(None, 1024)	32769024
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 20)	10260
Total params: 33,373,872		
Trainable params: 33,373,872		
Non-trainable params: 0		

Model Training and Evaluation

Loss function: Sparse Categorical Cross Entropy

Optimizer: Adam with learning rate=0.0001

```
[14]: face_rec_model.compile(  
      loss='sparse_categorical_crossentropy',  
      optimizer=Adam(learning_rate=0.0001),  
      metrics=['accuracy']  
)
```

```
[15]: callback = EarlyStopping(monitor='loss', patience=3)
```

```
[16]: history = face_rec_model.fit(np.array(x_train), np.array(y_train),  
                                batch_size=batch_size,  
                                epochs=150,  
                                verbose=2,  
                                validation_data=(np.array(x_val), np.array(y_val)),  
                                callbacks=[callback])
```

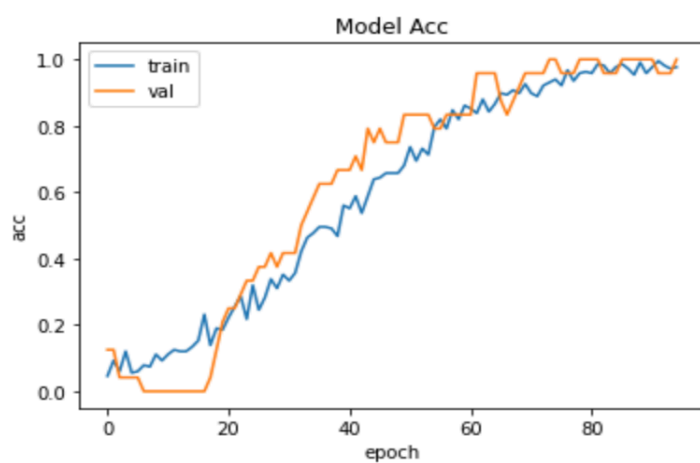
```
Epoch 1/150  
1/1 - 21s - loss: 2.9956 - accuracy: 0.0463 - val_loss: 2.9941 - val_accuracy: 0.1250 - 21s/epoch - 21s/step  
Epoch 2/150  
1/1 - 13s - loss: 2.9941 - accuracy: 0.0926 - val_loss: 3.0013 - val_accuracy: 0.1250 - 13s/epoch - 13s/step  
Epoch 3/150  
1/1 - 13s - loss: 2.9766 - accuracy: 0.0602 - val_loss: 3.0072 - val_accuracy: 0.0417 - 13s/epoch - 13s/step  
Epoch 4/150  
1/1 - 11s - loss: 2.9862 - accuracy: 0.1204 - val_loss: 3.0117 - val_accuracy: 0.0417 - 11s/epoch - 11s/step  
Epoch 5/150  
1/1 - 11s - loss: 2.9758 - accuracy: 0.0556 - val_loss: 3.0155 - val_accuracy: 0.0417 - 11s/epoch - 11s/step  
Epoch 6/150
```

```
[18]: eval = face_rec_model.evaluate(np.array(x_test), np.array(y_test), verbose=0)

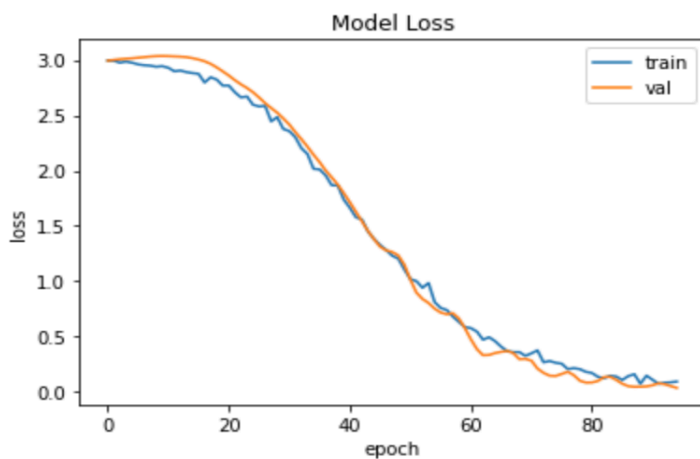
print('Test loss ', eval[0])
print('Test Acc ', eval[1])
```

Test loss 0.21071676909923553
Test Acc 0.949999988079071

```
[19]: plt.title('Model Acc')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epoch')
plt.ylabel('acc')
plt.legend(['train', 'val'])
plt.show()
```



```
[20]: plt.title('Model Loss')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'val'])
plt.show()
```



Conclusion

Model accuracy on the test data is 95%.

```
[23]: acc_score = accuracy_score(y_test, y_pred)
      print('Acc Score ', acc_score)
```

Acc Score 0.95

```
[24]: cmatrix = confusion_matrix(y_test, y_pred)
```

```
ax = sns.heatmap(cmatrix, annot=True)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted Classes')
ax.set_ylabel('Actual Classes')
```

```
plt.show()
```

