MNIST-Keras

January 27, 2021

0.0.1 Convolutional Neural Network

[151]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
plt.style.use('seaborn-white')
import numpy as np
import warnings
warnings.filterwarnings('ignore','DeprecatedWarnings')
warnings.filterwarnings('ignore','UserWarnings')
```

[46]:
```
tf.__version__
```

[46]:
```
'2.4.1'
```

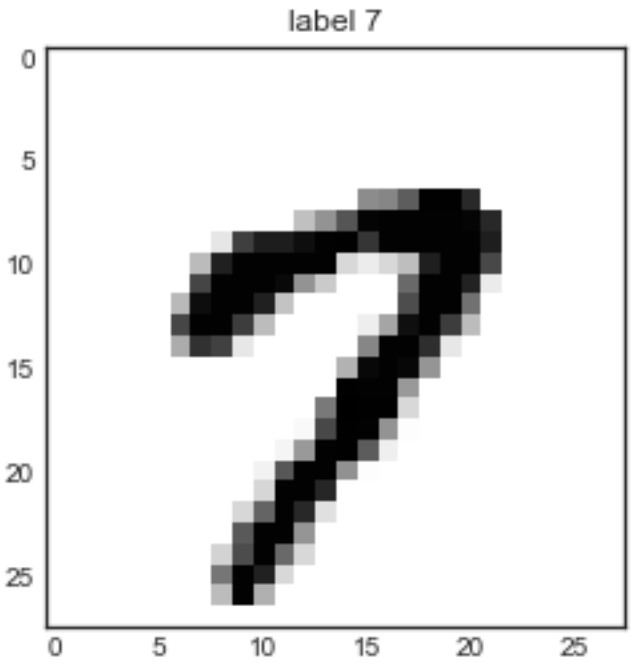
0.0.2 Loading DataSet

[47]:
```
from tensorflow.keras.datasets import mnist
(x_train, y_train), ( x_test, y_test) = mnist.load_data()
```

0.0.3 View

[48]:
```
index = 15
plt.imshow(x_train[index],cmap='binary')
plt.title('label {}'.format(y_train[index]))
```

[48]:
```
Text(0.5, 1.0, 'label 7')
```



0.0.4 Dimensions

[49]:
```
print('X_train Dimension {}'.format(x_train.shape))
print('y_train Dimension {}'.format(y_train.shape)) # (val,1)
print('X_test Dimension {}'.format(x_test.shape))
print('y_test Dimension {}'.format(y_test.shape)) # (val,1)
print('Each Image HxW : {}x{}'.format(x_train[0].shape[0],x_train.shape[1]))
```

```
X_train Dimension (60000, 28, 28)
y_train Dimension (60000,)
X_test Dimension (10000, 28, 28)
y_test Dimension (10000,)
Each Image HxW : 28x28
```

0.0.5 One-Hot Encoding the Target Labels

[50]:
```
from tensorflow.keras.utils import to_categorical
```

[51]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

[52]:
```
y_train.shape # one-hot encoded 10 dimensional vector
```

[52]:
```
(60000, 10)
```

0.0.6 Changing Dimensions

[53]:
```
# current dimensions
print('Current Dimensions of Training Set : {}'.format(x_train.shape))
print('Current Dimensions of Test Set :  {}'.format(x_test.shape))
```

```
Current Dimensions of Training Set : (60000, 28, 28)
Current Dimensions of Test Set :  (10000, 28, 28)
```

[54]:
```
x_train.shape[0]
```

[54]: 60000

[55]:
```
# Changing Dimensions to (60000, 28, 28, 1 ) # where 1 represents the color␣
↪channel
x_train = x_train.reshape(60000,28,28,1)
x_test = x_test.reshape(10000,28,28,1)
print('Changed Dimensions of Training Set {}'.format(x_train.shape))
print('Changed Dimensions of Test Set {}'.format(x_test.shape))
```

```
Changed Dimensions of Training Set (60000, 28, 28, 1)
Changed Dimensions of Test Set (10000, 28, 28, 1)
```

0.0.7 Data Normalisation

[56]:
```
x_train = x_train.astype("float")/255.0
x_test = x_test.astype("float")/255.0
```

0.0.8 Constructing the ConvNet Architecture

[174]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,␣
↪Conv2D, MaxPooling2D
```

[175]:
```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3),␣
↪activation='relu',input_shape=x_train.shape[1:]))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

[176]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[177]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               692352
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
```

[178]:
```
import time
start = time.perf_counter()
model.fit(x_train,y_train, epochs=5, validation_data=(x_test, y_test))

elapsed = time.perf_counter() - start
print('elapsed : {:.2f}'.format(elapsed/60))
```

```
Epoch 1/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2953 -
accuracy: 0.9130 - val_loss: 0.0679 - val_accuracy: 0.9799
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0554 -
accuracy: 0.9825 - val_loss: 0.0578 - val_accuracy: 0.9814
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0305 -
accuracy: 0.9903 - val_loss: 0.0381 - val_accuracy: 0.9864
Epoch 4/5
1875/1875 [==============================] - 10s 6ms/step - loss: 0.0173 -
accuracy: 0.9949 - val_loss: 0.0445 - val_accuracy: 0.9853
Epoch 5/5
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0137 -
accuracy: 0.9956 - val_loss: 0.0447 - val_accuracy: 0.9868
elapsed : 1.10
```

[179]:
```
history = pd.DataFrame(model.history.history)
```

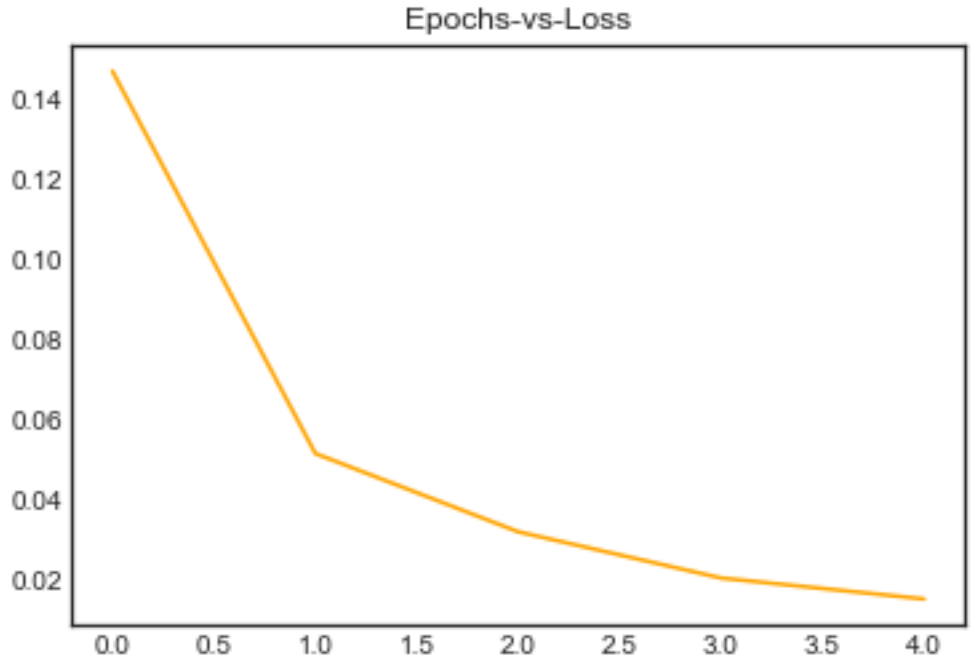
[180]:
```
history
```

[180]:

	loss	accuracy	val_loss	val_accuracy
0	0.147167	0.956133	0.067945	0.9799
1	0.051351	0.984200	0.057815	0.9814
2	0.031731	0.990150	0.038122	0.9864
3	0.020144	0.993933	0.044461	0.9853
4	0.014895	0.995050	0.044664	0.9868

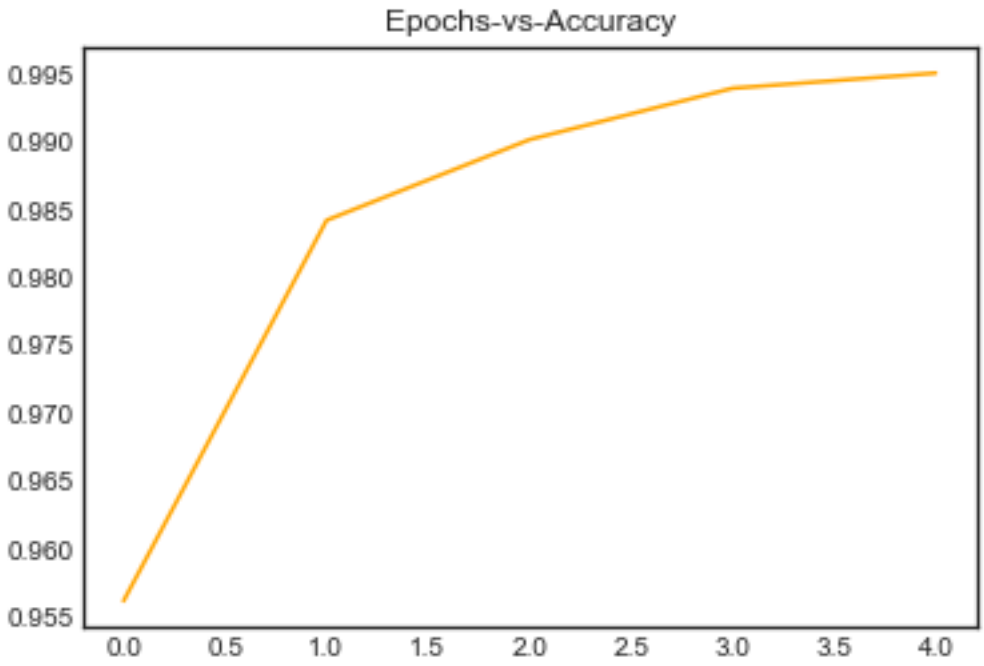
[181]:
```
history['loss'].plot(title='Epochs-vs-Loss',color='orange')
```

[181]: <AxesSubplot:title={'center':'Epochs-vs-Loss'}>



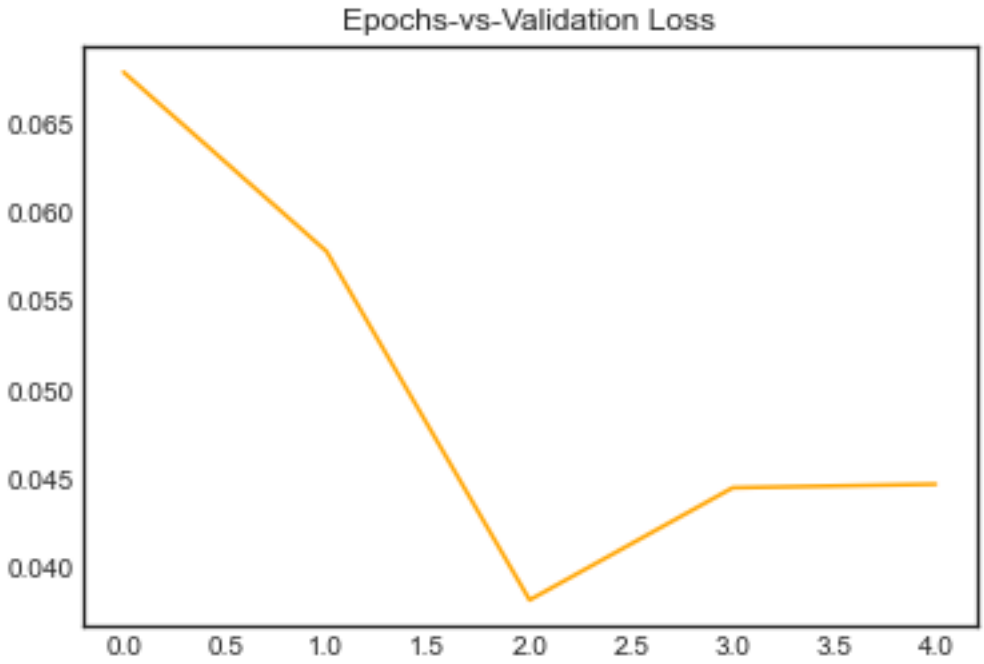
[182]:
```
history['accuracy'].plot(title='Epochs-vs-Accuracy',color='orange')
```

[182]: <AxesSubplot:title={'center':'Epochs-vs-Accuracy'}>



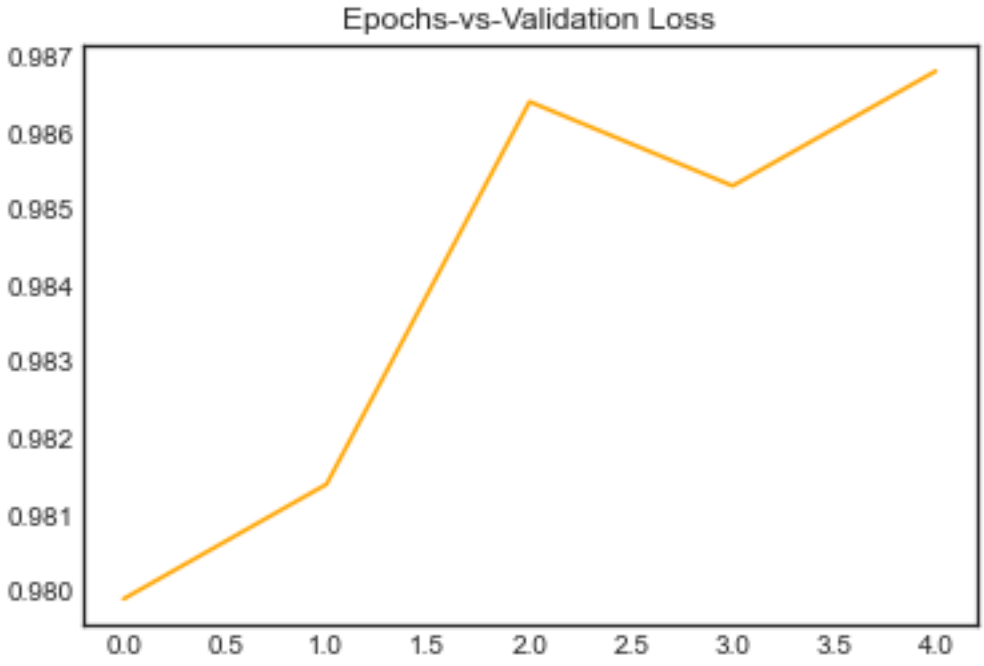
[183]:
```
history['val_loss'].plot(title='Epochs-vs-Validation Loss',color='orange')
```

[183]: <AxesSubplot:title={'center':'Epochs-vs-Validation Loss'}>



[184]:
```
history['val_accuracy'].plot(title='Epochs-vs-Validation Loss',color='orange')
```

[184]:
```
<AxesSubplot:title={'center':'Epochs-vs-Validation Loss'}>
```



[185]:
```
from sklearn.metrics import classification_report
```

[186]:
```
predictions = model.predict_classes(x_test)
metric_report = classification_report(np.argmax(y_test,1),predictions)
```

```
/Users/rohan/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[187]:
```
print(metric_report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.99      0.98      0.99      1032
```

```
           3       0.98      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.97      0.99      0.98       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.97      0.99      0.98       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

[ ]:

[188]:
```
epochs=5
train_acc = history['accuracy'][epochs-1]
test_acc = history['val_accuracy'][epochs-1]
train_acc *= 100
test_acc *= 100
```

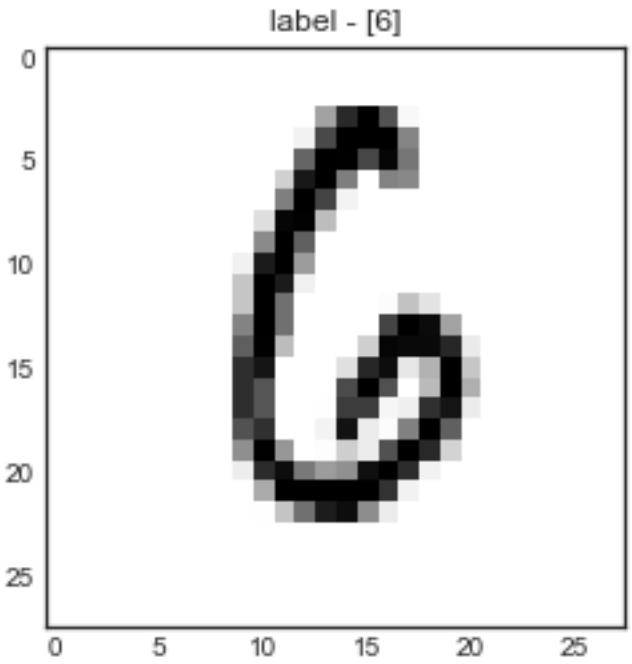
[189]:
```
print('Training Accuracy, last-epoch {:.2f}'.format(train_acc))
print('Testing Accuracy, last-epoch {:.2f}'.format(test_acc))
```

```
Training Accuracy, last-epoch 99.51
Testing Accuracy, last-epoch 98.68
```

0.0.9 Evaluation

[192]:
```
index = np.random.randint(1,10000,1)
random_sample = x_test[index]
label = np.argmax(y_test[index],1)
plt.imshow(random_sample.reshape(28,28),cmap='binary')
plt.title('label - {}'.format(label))
```

[192]:
```
Text(0.5, 1.0, 'label - [6]')
```



[193]:
```
# Evaluating the predicted value
print('Model Thinks this is {}'.format(model.predict_classes(random_sample.
 ↪reshape(1,28,28,1))[0]))
if label[0] == model.predict_classes(random_sample.reshape(1,28,28,1))[0]:
    print('Which is True')
else:
    print('It is incorrect')
```

```
Model Thinks this is 6
Which is True
```

0.0.10 Miscellaneous

[194]:
```
! ls ../../
```

```
MNIST                   pipeline.py
PipeLine Testing .ipynb var-dict.txt
```

[195]:
```
# desc-----string
# project_name-----string
# framework-----string
# prediction_type-----string
# network_type-----string
# architecture-----string
# layers-----int
```

```
# hidden_units-----int
# activations-----string(list)
# epochs-----int
# metrics-----string
# train_accuracy-----float%
# test_accuracy-----float%
# classification_report-----string
# elapsed-----float
# summary-----string
# ipynb-----path
# plots-----path
```

[208]:
```
summary = '''Same as Pytorch's Version of the implementation, Even the Keras␣
 ↪model slightly starts to overfit after the 2nd epoch in this case, but the␣
 ↪difference is not prominent to be concerned with performance, the model is␣
 ↪98% accurate with unseen data.'''
```

[209]:
```
summary
```

[209]:
```
"Same as Pytorch's Version of the implementation, Even the Keras model slightly
starts to overfit after the 2nd epoch in this case, but the difference is not
prominent to be concerned with performance, the model is 98% accurate with
unseen data."
```

[199]:
```
desc = 'The MNIST database of handwritten digits, available from this page, has␣
 ↪a training set of 60,000 examples, and a test set of 10,000 examples. It is␣
 ↪a subset of a larger set available from NIST. The digits have been␣
 ↪size-normalized and centered in a fixed-size image.'
project_name = 'MNIST'
framework = 'Keras'
prediction_type = 'Multi-Class Classification - 10 Classes'
network_type = 'Convolutional Neural Network'
architecture = str(model.summary())
layers = 5
hidden_units = 'None'
activations = "['relu','softmax']"
metrics = str(model.metrics_names)
train_accuracy = '{:.2f}'.format(train_acc)
test_accuracy = '{:.2f}'.format(test_acc)
classification_report = metric_report
elapsed = 'elapsed : {:.2f}'.format(elapsed/60) + ' Mins'
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
```

```
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               692352
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
```

[ ]: