Deep Learning for Image Classification Assessment

SOLUTION

The Data

In [1]:
```
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

Visualizing the Data

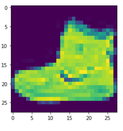
In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
x_train[0]
```

Out[3]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
          0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
          0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
          0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
         10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
         72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,  69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,
         88, 172],
       [  0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   1,   1,   1,
          0, 200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123,
         96, 229],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223,
        245, 173],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220,
        243, 202],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,
          0,  12, 219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212,
        226, 197],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
          0,  99, 244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220,
        245, 119],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,
          0,  55, 236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217,
        217, 209],
       [  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0,   0,
        237, 226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218,
        255,  77],
       [  0,   3,   0,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204,
        228, 207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224,
        244, 159],
       [  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
        226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
        215,   0],
       [  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
        159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
        246,   0],
       [  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
         80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
        225,   0],
       [ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
        241,  65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224,
        229,  29],
       [ 75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206, 198,
        213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220, 221,
        230,  67],
       [ 48, 203, 183, 194, 213, 197, 185, 190, 194, 192, 202, 214, 219,
        221, 220, 236, 225, 216, 199, 206, 186, 181, 177, 172, 181, 205,
        206, 115],
       [  0, 122, 219, 193, 179, 171, 183, 196, 204, 210, 213, 207, 211,
        210, 200, 196, 194, 191, 195, 191, 198, 192, 176, 156, 167, 177,
        210,  92],
       [  0,   0,   0,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188,
        189, 188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192,
        216, 170],
       [  0,   2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243,
        244, 221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,
         58,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,
         35,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

In [4]:
```
plt.imshow(x_train[0])
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x7f9438a5f210>
```



In [5]:
```
y_train[0]
```

Out[5]:
```
9
```

Preprocessing the Data

In [6]:
```
x_train.max()
```

Out[6]:
```
255
```

In [7]:
```
x_train = x_train / 255
```

In [8]:
```
x_test = x_test / 255
```

In [9]:
```
x_train.shape
```

Out[9]:
```
(60000, 28, 28)
```

In [10]:
```
x_train = x_train.reshape(60000, 28, 28, 1)
```

In [11]:
```
x_test = x_test.reshape(10000, 28, 28, 1)
```

In [12]:
```
from tensorflow.keras.utils import to_categorical
```

In [13]:
```
y_train
```

Out[13]:
```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

In [14]:
```
y_cat_train = to_categorical(y_train)
```

In [15]:
```
y_cat_test = to_categorical(y_test)
```

Building the Model

In [16]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

In [17]:
```
model = Sequential()

# CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4, 4),input_shape=(28, 28, 1), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 28 by 28 to 764 BEFORE FINAL LAYER
model.add(Flatten())

# 128 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF NEURONS)
model.add(Dense(128, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model.add(Dense(10, activation='softmax'))


model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 25, 25, 32)        544

 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0
 )

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 128)               589952

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 591,786
Trainable params: 591,786
Non-trainable params: 0
_________________________________________________________________
```

Training the Model

In [19]:
```
model.fit(x_train, y_cat_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 13s 6ms/step - loss: 0.3989 - accuracy: 0.8584
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2761 - accuracy: 0.9009
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2375 - accuracy: 0.9147
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2123 - accuracy: 0.9237
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1957 - accuracy: 0.9298
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1806 - accuracy: 0.9359
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1677 - accuracy: 0.9402
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1545 - accuracy: 0.9445
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1487 - accuracy: 0.9473
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1393 - accuracy: 0.9504
```

Out[19]:
```
<keras.callbacks.History at 0x7f9438a39ad0>
```

Evaluating the Model

In [20]:
```
model.metrics_names
```

Out[20]:
```
['loss', 'accuracy']
```

In [21]:
```
model.evaluate(x_test, y_cat_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.3152 - accuracy: 0.9094
```

Out[21]:
```
[0.3151517750640869, 0.9093999862670898]
```

In [22]:
```
from sklearn.metrics import classification_report
```

In [23]:
```
predictions = model.predict(x_test).argmax(axis=1).astype('uint8')
```

```
313/313 [==============================] - 1s 2ms/step
```

In [24]:
```
y_cat_test.shape
```

Out[24]:
```
(10000, 10)
```

In [25]:
```
y_cat_test[0]
```

Out[25]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

In [26]:
```
predictions[0]
```

Out[26]:
```
9
```

In [27]:
```
y_test
```

Out[27]:
```
array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

In [28]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.89      0.82      0.85      1000
           1       0.99      0.98      0.99      1000
           2       0.86      0.84      0.85      1000
           3       0.89      0.93      0.91      1000
           4       0.85      0.86      0.86      1000
           5       0.98      0.97      0.98      1000
           6       0.73      0.79      0.76      1000
           7       0.95      0.97      0.96      1000
           8       0.99      0.97      0.98      1000
           9       0.97      0.96      0.97      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```