Artificial Intelligence Internship Project

TITLE: DEEP LEARNING PROJECT FASHION MNIST CLASSIFICATION

From: CH Hemanth Nag

Abstract:

Fashion has always been an essential feature in our daily routine. It also plays a significant role in everyone's lives. In this research, convolutional neural networks (CNN) were used to train images of different fashion styles, which were attempted to be predicted with a high success rate. Deep learning has been widely applied in a variety of fields recently. A CNN is a deep neural network that delivers the most accurate answers when tackling real-world situations. Apparel manufacturers have employed CNN to tackle various difficulties on their e-commerce sites, including clothing recognition, search, and suggestion. A set of photos from the Fashion-MNIST dataset is used to train a series of CNN-based deep learning architectures to distinguish between photographs. CNN design, batch normalization, and residual skip connections reduce the time it takes to learn. The CNN model's findings are evaluated using the Fashion-MNIST datasets. In this paper, classification is done with a convolutional layer, filter size, and ultimately connected layers. Experiments are run with different activation functions, optimizers, learning rates, dropout rates, and batch sizes. The results showed that the choice of activation function, optimizer, and dropout rate impacts the correctness of the results.

Objective:

Deep learning is a subfield of machine learning related to artificial neural networks. The word deep means bigger neural networks with a lot of hidden units. Deep learning's CNNs have proved to be the state-of-the-art technique for image recognition tasks. Keras is a deep learning library in Python that provides an interface for creating an artificial neural network. It is an open-sourced program. It is built on top of Tensorflow.

The prime objective of this article is to implement a CNN to perform image classification on the famous fashion MNIST dataset. In this, we will be implementing our own CNN architecture. The process will be divided into three steps: data analysis, model training, and prediction.

Introduction:

The MNIST database of handwritten digits is one of the most widely used data sets used to explore Neural Networks and became a benchmark for model comparison. More recently, Zalando research published a new dataset, with 10 different fashion products. Called fashion MNIST, this dataset is meant to be a replacement for the original MNIST which turned out to be too easy for machine learning folks; even linear classifiers were able to achieve high classification accuracy. The new dataset promises to be more challenging, so that machine learning algorithms have to learn more advanced features to correctly classify the images.

Methodology:

Training the first NN model

Training a Neural Network (NN) requires 4 steps:

Step 1 — Build the architecture
Step 2 — Compile the model
Step 3 — Train the model
Step 4 — Evaluating the model

Step 1 — Build the architecture

First, we'll design the NN architecture by deciding the number of layers and activation functions. We'll start with a simple 3-layer Neural Network. In the first layer we 'flatten' the data, so that a (28x28) shape flattens to 784. The second layer is a dense layer with a *ReLu* activation function and has 128 neurons. The last layer is a dense layer with a softmax activation function that classifies the 10 categories of the data and has 10 neurons.

Step 2 — Compile the model

Next, we compile the model with the following settings:

Loss function — calculates the difference between the output and the target variable. It measures the accuracy of the model during training and we want to minimize this function. In this example, we chose the *sparse_categorical_crossentropy* loss function. Cross-entropy is the default loss function to use for a multi-class classification problem and it's sparse because our targets are not one-hot encodings but are integers.
Optimizer — how the model is updated and is based on the data and the loss function. *Adam* is an extension to the classic stochastic gradient descent and is popular because it's shown to be effective and efficient.
Metrics — monitors the training and testing steps. *Accuracy* is a common metric and it measures the fraction of images that are correctly classified.

Step 3 — Train the model

Next, we train the model by fitting it to the training data, so we give it the input (images) and expected output (labels). Here, an important step to minimize overfitting is validation. There are a few ways to validate, in this case, we use the automatic validation built into the function, where we set the *validation_split* on the training data. Here we use an 80/20 split: 80% for training and 20% for validating.

Step 4 — Evaluate the model

Now that we've set up and trained our model, we need to evaluate its performance. This is done on a test dataset, new data that the model hasn't seen yet. We have to make sure to separate our training and validating dataset from our testing dataset.

Code:

```
         DEEP LEARNING PROJECT FASHION MNIST CLASSIFICATION

In [1]:  import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         import tensorflow as tf
         import keras

In [2]:  (X_train, y_train), (X_test, y_test)=tf.keras.datasets.fashion_mnist.load_data()

In [3]:  # print the shape of data

In [4]:  X_train.shape,y_train.shape, "***************" , X_test.shape,y_test.shape
Out[4]:  ((60000, 28, 28), (60000,), '***************', (10000, 28, 28), (10000,))

In [5]:  X_train[0]
Out[5]:  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,   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,   1,   0,
         69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
        172,  66],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
        200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
        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,   1,   3,   0,  12,
        219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
        209,  52],
       [  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,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
        226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
         77,   0],
       [  0,   3,   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,   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,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188, 189,
        188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192, 216,
```

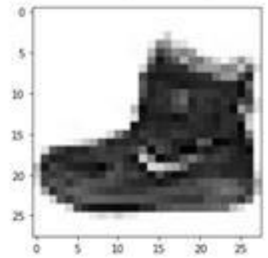
```
       [  2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243, 244,
        221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,  58,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   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,   0]], dtype=uint8)
```

```
In [6]: y_train[0]
Out[6]: 9

In [7]: class_labels = [    "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt",    "Sneaker",  "Bag",  "Ankle boot"]

In [8]: class_labels
Out[8]: ['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
```

```
        'Shirt',
        'Sneaker',
        'Bag',
        'Ankle boot']
```

In [9]:
```
plt.imshow(X_train[0],cmap='Greys')
```

Out[9]: <matplotlib.image.AxesImage at 0x1623dfdd3a0>



In [10]:
```
plt.figure(figsize=(16,16))

j=1
for  i in np.random.randint(0,1000,25):
```

In [10]:
```
plt.figure(figsize=(16,16))

j=1
for  i in np.random.randint(0,1000,25):
  plt.subplot(5,5,j);j+=1
  plt.imshow(X_train[i],cmap='Greys')
  plt.axis('off')
  plt.title('{} / {}'.format(class_labels[y_train[i]],y_train[i]))
```

```
In [11]: X_train.ndim
Out[11]: 3

In [12]: X_train = np.expand_dims(X_train,-1)

In [13]: X_train.ndim
Out[13]: 4

In [14]: X_test=np.expand_dims(X_test,-1)

In [15]: # feature scaling

In [16]: X_train = X_train/255
         X_test= X_test/255

In [17]: # Split dataset

In [18]: from sklearn.model_selection import  train_test_split
         X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_train,test_size=0.2,random_state=2020)

In [19]: X_train.shape,X_Validation.shape,y_train.shape,y_Validation.shape
Out[19]: ((48000, 28, 28, 1), (12000, 28, 28, 1), (48000,), (12000,))
```

```
In [20]: model=keras.models.Sequential([
                                 keras.layers.Conv2D(filters=32,kernel_size=3,strides=(1,1),padding='valid',activation='relu',input_shape
                                 keras.layers.MaxPooling2D(pool_size=(2,2)),
                                 keras.layers.Flatten(),
                                 keras.layers.Dense(units=128,activation='relu'),
                                 keras.layers.Dense(units=10,activation='softmax')
         ])

In [21]: model.summary()

         Model: "sequential"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          conv2d (Conv2D)             (None, 26, 26, 32)        320

          max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
          )

          flatten (Flatten)           (None, 5408)              0

          dense (Dense)               (None, 128)               692352

          dense_1 (Dense)             (None, 10)                1290

         =================================================================
         Total params: 693,962
```

```
         Trainable params: 693,962
         Non-trainable params: 0
         _________________________________________________________________

In [22]: model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])

In [23]: model.fit(X_train,y_train,epochs=10,batch_size=512,verbose=1,validation_data=(X_Validation,y_Validation))

         Epoch 1/10
         94/94 [==============================] - 15s 145ms/step - loss: 0.6369 - accuracy: 0.7810 - val_loss: 0.4438 - val_accuracy: 0.
         8412
         Epoch 2/10
         94/94 [==============================] - 12s 133ms/step - loss: 0.3808 - accuracy: 0.8674 - val_loss: 0.3665 - val_accuracy: 0.
         8706
         Epoch 3/10
         94/94 [==============================] - 12s 133ms/step - loss: 0.3315 - accuracy: 0.8843 - val_loss: 0.3295 - val_accuracy: 0.
         8853
         Epoch 4/10
         94/94 [==============================] - 14s 144ms/step - loss: 0.3080 - accuracy: 0.8908 - val_loss: 0.3133 - val_accuracy: 0.
         8895
         Epoch 5/10
         94/94 [==============================] - 14s 152ms/step - loss: 0.2835 - accuracy: 0.9001 - val_loss: 0.2987 - val_accuracy: 0.
         8945
         Epoch 6/10
         94/94 [==============================] - 13s 139ms/step - loss: 0.2667 - accuracy: 0.9057 - val_loss: 0.2878 - val_accuracy: 0.
         8983
         Epoch 7/10
         94/94 [==============================] - 13s 136ms/step - loss: 0.2476 - accuracy: 0.9124 - val_loss: 0.2781 - val_accuracy: 0.
```

```
Epoch 8/10
94/94 [==============================] - 13s 135ms/step - loss: 0.2343 - accuracy: 0.9159 - val_loss: 0.2738 - val_accuracy: 0.
9016
Epoch 9/10
94/94 [==============================] - 13s 133ms/step - loss: 0.2239 - accuracy: 0.9200 - val_loss: 0.2696 - val_accuracy: 0.
9050
Epoch 10/10
94/94 [==============================] - 12s 132ms/step - loss: 0.2133 - accuracy: 0.9234 - val_loss: 0.2642 - val_accuracy: 0.
9057
```

```
Out[23]: <keras.callbacks.History at 0x1623e97f490>
```

```
In [24]: y_pred = model.predict(X_test)
         y_pred.round(2)

         313/313 [==============================] - 1s 4ms/step

Out[24]: array([[0.  , 0.  , 0.  , ..., 0.01, 0.  , 0.98],
                [0.  , 0.  , 1.  , ..., 0.  , 0.  , 0.  ],
                [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
                ...,
                [0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ],
                [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
                [0.  , 0.  , 0.01, ..., 0.11, 0.09, 0.  ]], dtype=float32)
```

```
In [25]: y_test

Out[25]: array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

```
In [26]: model.evaluate(X_test, y_test)

         313/313 [==============================] - 1s 4ms/step - loss: 0.2708 - accuracy: 0.9014

Out[26]: [0.2708267271518707, 0.9014000296592712]
```

```
In [27]: plt.figure(figsize=(16,16))

         j=1
         for i in np.random.randint(0, 1000,25):
           plt.subplot(5,5, j); j+=1
           plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
           plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])
           plt.axis('off')
```

In [28]:
```python
plt.figure(figsize=(16,30))

j=1
for i in np.random.randint(0, 1000,60):
  plt.subplot(10,6, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])
  plt.axis('off')
```

In [29]:
```python
"""## Confusion Matrix"""
```

Out[29]: '## Confusion Matrix'

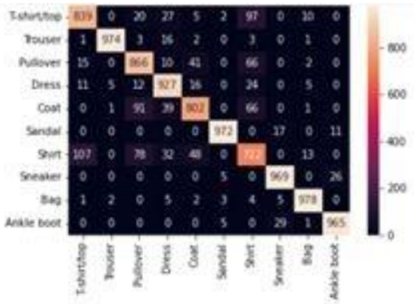
In [30]:
```python
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
```

In [30]:
```python
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(y_test, y_pred_labels)
```

```
<Figure size 1152x648 with 0 Axes>
```

In [31]:
```python
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=class_labels)

from sklearn.metrics import classification_report
cr= classification_report(y_test, y_pred_labels, target_names=class_labels)
print(cr)
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.86      0.84      0.85      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.81      0.87      0.84      1000
       Dress       0.88      0.93      0.90      1000
        Coat       0.88      0.80      0.84      1000
      Sandal       0.98      0.97      0.98      1000
       Shirt       0.74      0.72      0.73      1000
     Sneaker       0.95      0.97      0.96      1000
         Bag       0.97      0.98      0.97      1000
  Ankle boot       0.96      0.96      0.96      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
```



In [32]:
```python
"""# Save Model"""
```

Out[32]: '# Save Model'

In [33]:
```python
model.save('fashion_mnist_cnn_model.h5')
```

In [35]:
```python
#Building CNN model
cnn_model2 = keras.models.Sequential([
                         keras.layers.Conv2D(filters=32, kernel_size=3, strides=(1,1), padding='valid',activation= 'relu', input_
                         keras.layers.Conv2D(filters=128, kernel_size=3, strides=(2,2), padding='same', activation='relu'),
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2,2), padding='same', activation='relu'),
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Flatten(),
                         keras.layers.Dense(units=128, activation='relu'),
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=256, activation='relu'),
                         keras.layers.Dropout(0.5),
                         keras.layers.Dense(units=256, activation='relu'),
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=128, activation='relu'),
                         keras.layers.Dropout(0.10),
                         keras.layers.Dense(units=10, activation='softmax')
                         ])

# complie the model
cnn_model3.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['accuracy'])

#Train the Model
cnn_model3.fit(X_train, y_train, epochs=50, batch_size=512, verbose=1, validation_data=(X_Validation, y_Validation))

cnn_model3.save('fashion_mnist_cnn_model3.h5')

cnn_model3.evaluate(X_test, y_test)
```

```
Epoch 1/20
94/94 [==============================] - 20s 201ms/step - loss: 1.0060 - accuracy: 0.6316 - val_loss: 0.5760 - val_accuracy:
0.7798
Epoch 2/20
94/94 [==============================] - 17s 180ms/step - loss: 0.5459 - accuracy: 0.7929 - val_loss: 0.4506 - val_accuracy:
0.8341
Epoch 3/20
94/94 [==============================] - 15s 165ms/step - loss: 0.4540 - accuracy: 0.8330 - val_loss: 0.3961 - val_accuracy:
0.8561
Epoch 4/20
94/94 [==============================] - 16s 175ms/step - loss: 0.4028 - accuracy: 0.8531 - val_loss: 0.4005 - val_accuracy:
0.8499
Epoch 5/20
94/94 [==============================] - 15s 163ms/step - loss: 0.3664 - accuracy: 0.8682 - val_loss: 0.3401 - val_accuracy:
0.8758
Epoch 6/20
94/94 [==============================] - 16s 172ms/step - loss: 0.3406 - accuracy: 0.8773 - val_loss: 0.3231 - val_accuracy:
0.8814
Epoch 7/20
```

```
0.9074
Epoch 46/50
94/94 [==============================] - 35s 377ms/step - loss: 0.0707 - accuracy: 0.9755 - val_loss: 0.4168 - val_accuracy:
0.9070
Epoch 47/50
94/94 [==============================] - 35s 370ms/step - loss: 0.0701 - accuracy: 0.9754 - val_loss: 0.4635 - val_accuracy:
0.9038
Epoch 48/50
94/94 [==============================] - 34s 365ms/step - loss: 0.0646 - accuracy: 0.9774 - val_loss: 0.4163 - val_accuracy:
0.9082
Epoch 49/50
94/94 [==============================] - 40s 429ms/step - loss: 0.0563 - accuracy: 0.9803 - val_loss: 0.4498 - val_accuracy:
0.9071
Epoch 50/50
94/94 [==============================] - 39s 415ms/step - loss: 0.0644 - accuracy: 0.9778 - val_loss: 0.4397 - val_accuracy:
0.9055
313/313 [==============================] - 3s 9ms/step - loss: 0.4309 - accuracy: 0.9049
Out[35]: [0.4308927357196808, 0.9049000144004822]
```

END

Conclusion:

Hence by using python and deeplearning methon we created a prefarable code for Fashion MNIST where we can classify data accoridingly which can be seen above.