Deep learning for image recognition of Fashion-MNIST dataset and classification of rice type

Main objective

The project is divided in two parts. In the first one, I use two convolutional neural networks for image recognition task on Fashion-MNIST dataset. In this case I will vary some hyperparameters of two neural networks and compare the accuracies of image recognition.

The second part is devoted to classification of supervised machine learning using neural networks. There will be classification of rice type from a Kaggle dataset https://www.kaggle.com/mssmartypants/rice-type-classification. I will leverage two neural network models in this case with different number of hidden layers and compare the accuracy of the neural network model with a common ensemble classifier such as Random Forest.

Data description

Fashion-MNIST dataset

The Fashion-MNIST dataset comprises 70000 images of 10 different classes of objects, which are separated in a training set of 60000 images and a test set of 10000 images. The images are in grey scale with 28 by 28 pixel size. The cloth names are ordered as follows:

```
Out[202]: array(['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',
                 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'], dtype='<U11')
```

The number of the element in this list above represents the corresponding labels for the images.

Rice classification dataset

The rice type classification dataset contains 10 features and the target label. The head of the dataframe cotaining the rice dataset looks as following:

```
df.head()
```

Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	EquivDiameter	Extent	Perimeter	Roundness	AspectRation	Class
4537	92.229317	64.012769	0.719916	4677	76.004525	0.657536	273.085	0.764510	1.440796	1
2872	74.691881	51.400454	0.725553	3015	60.471018	0.713009	208.317	0.831658	1.453137	1
3048	76.293164	52.043491	0.731211	3132	62.296341	0.759153	210.012	0.868434	1.465950	1
3073	77.033628	51.928487	0.738639	3157	62.551300	0.783529	210.657	0.870203	1.483456	1
3693	85.124785	56.374021	0.749282	3802	68.571668	0.769375	230.332	0.874743	1.510000	1

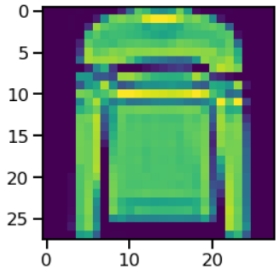
The ‘Class’ columns corresponds to a rice type (0 for ‘Gonen’, and 1 for Jasmine). Ten features representing rice grain parameters are all numeric.

Data exploration and engineering

Fashion-MNIST dataset

The Fashion_MNIST dataset has been loaded to a train set and a test set from keras.dataset library. The pixel values for the images (x_train and x_test) have been scaled by dividing them by 255, the maximal pixel brightness. The y_labels have been converted to categorical type using keras.utils library.

As an example for demonstration, a random image pulled out of a train set looks like this:



Then, in order to feed the images to convolutional NN, I had to add another dimension to the image, as it's a grey scale image, not an RGB image:

```
B [37]: #Adding one more dimension to feed it to Conv2D NN layer
        x_train=x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2],1)
        x_test=x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2],1)

B [38]: x_train.shape[1:]
Out[38]: (28, 28, 1)
```

So, each image in the dataset is now represented as a stack of one single image.

Rice classification dataset

The rice classification dataset has been scanned on presence of not a number elements. They were not found. Then, I have verified whether the classes are balanced as follows:

```
B [122]: data_df.Class.value_counts(normalize=True)

Out[122]: 1    0.549079
          0    0.450921
          Name: Class, dtype: float64
```

It appears they are well balanced, and I used a normal train_test_split fuction with a 0.3 test size to split X and y data for training and testing. The X_train and X_test data have been scaled by StandardScaler from sklearn.preprocessing.

Deep learning models

As discussed before, I will use two CNN models for image recognition of Fashion-MNIST image dataset, and two NN models for classification of rice type in a rice classification dataframe.

Fashion-MNIST dataset

1) The first CNN model comprised one convolutional layer with 16 filters, stride=(1,1), and padding, activation was 'relu', then one convolutional layer with 16 filters, stride=(1,1), no padding, activation was 'relu'. Then a third convolutional layer was added with no padding , other parameters are the same. The max polling was applied after the third layer with pool_size=(2,2), and a dropout regularization of 0.3. The output was flattened and connected to a dense layer of 256 perceptrons with activation of 'relu'. A dropout regularization of value of 0.5 was added here. And finally it was connected to the y_pred dense layer of size of 10 (number of objects expected in the image dataset) with the activation 'softmax', predicting the probability of the image being in these particular classes. The full CNN structure is represented below:

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 28, 28, 16)	160
activation_5 (Activation)	(None, 28, 28, 16)	0
conv2d_7 (Conv2D)	(None, 26, 26, 16)	2320
activation_6 (Activation)	(None, 26, 26, 16)	0
conv2d_8 (Conv2D)	(None, 24, 24, 16)	2320
activation_7 (Activation)	(None, 24, 24, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 16)	0
dropout_2 (Dropout)	(None, 12, 12, 16)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_2 (Dense)	(None, 256)	590080
activation_8 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570
activation_9 (Activation)	(None, 10)	0

Total params: 597,450
Trainable params: 597,450
Non-trainable params: 0

Then, the model was compiled with Adam (learning rate = 0.0005) as optimizer for gradient descend, loss function was categorical cross entropy, metrics – accuracy. The model was fit through 3 epochs with a batch size of 64.

The validation accuracy turns out to be **91 %** and the loss is **0.243**, which are represented here for this model:

```
B [46]: #Accuracy of the model 1
        score_model_1=model_1.evaluate(x_test,y_test)
        print('Validation set: loss is {:.3f}; accuracy is {:.3f}'.format(score_model_1[0],score_model_1[1]))

        313/313 [==============================] - 4s 14ms/step - loss: 0.2433 - accuracy: 0.9097
        Validation set: loss is 0.243; accuracy is 0.910
```

2) So this accuracy is already very good, but in the frame of this project I leverage another CNN model, that has larger strides in the first CNN layer, and has 24 filters instead of 16 as in the previous model. The summary of the model is represented below:

Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 14, 14, 24)	240
activation_28 (Activation)	(None, 14, 14, 24)	0
conv2d_23 (Conv2D)	(None, 12, 12, 24)	5208
activation_29 (Activation)	(None, 12, 12, 24)	0
conv2d_24 (Conv2D)	(None, 10, 10, 24)	5208
activation_30 (Activation)	(None, 10, 10, 24)	0
max_pooling2d_5 (MaxPooling2	(None, 5, 5, 24)	0
dropout_10 (Dropout)	(None, 5, 5, 24)	0
flatten_5 (Flatten)	(None, 600)	0
dense_10 (Dense)	(None, 512)	307712
activation_31 (Activation)	(None, 512)	0
dropout_11 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 10)	5130
activation_32 (Activation)	(None, 10)	0

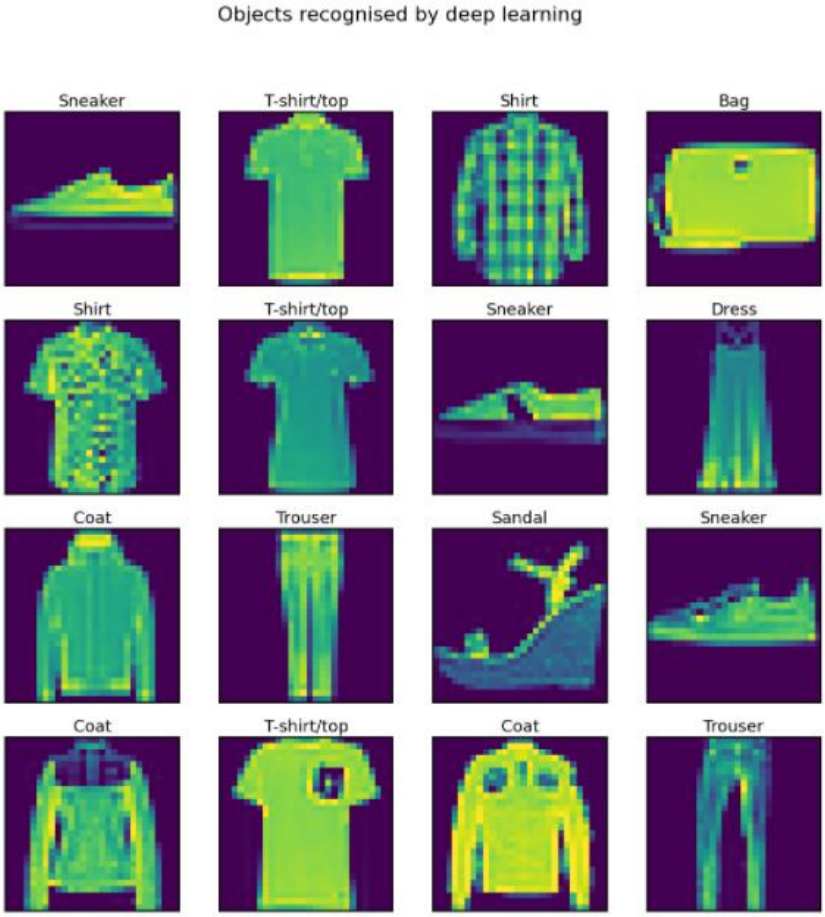
Total params: 323,498
Trainable params: 323,498
Non-trainable params: 0

This model has less training parameters, so I run it through 10 epochs this time, and use RMSprop as an optimizer to minimize the loss. Even after 10 epochs (as compared to 3 epochs for the 1st model), this second model reaches accuracy of **90.2 %**, which is still a bit less than the performance of the first model.

```
B [54]: #Accuracy of the model 2
        score_model_2=model_2.evaluate(x_test,y_test)
        print('Validation set: loss is {:.3f}; accuracy is {:.3f}'.format(score_model_2[0],score_model_2[1]))

        313/313 [==============================] - 2s 7ms/step - loss: 0.2725 - accuracy: 0.9024: 0s - loss: 0.2
        Validation set: loss is 0.272; accuracy is 0.902
```

Now, I choose the first CNN model with the accuracy of 91 % to predict the labels of 16 randomly selected images from the dataset. The labels predicted by the model are written above each of these images. As you can see below the prediction is almost 100 % correct:



Rice classification dataset

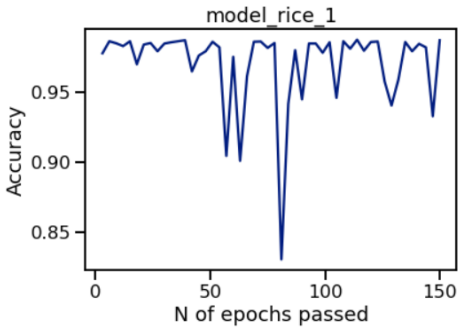
Now I move back to a classification problem of rice type in Kaggle rice type dataframe.

1) The first NN model here is designed to be simple. It comprised only one hidden layer of 20 perceptrons , the activation function was ‘relu’, then the put put layer of only one perceptron as we want to predict the class either 1 or 0, the activation function in the end is sigmoid. The summary of this NN is represented below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 20)                220
_________________________________________________________________
activation_34 (Activation)   (None, 20)                0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 21
=================================================================
Total params: 241
Trainable params: 241
Non-trainable params: 0
_________________________________________________________________
```

I have compiled this model with RMSprop (learning rate= 0.0005) as an optimizer, loss was set to 'binary_crossentropy', metrics included accuracy. Then fitting has been implemented with batch size of 256, through 150 epochs.

The validation accuracy reaches its maximum of 98.7 % quickly after several epochs, which can be seen in this graph:




```
B [203]: valacc_max_model_rice_1=max(val_accuracy)
         print('Maximal validation accuracy for model_rice_1 is {:.3f}'.format(valacc_max_model_rice_1))

         Maximal validation accuracy for model_rice_1 is 0.987
```

The validation accuracy seem to fluctuate and overfit the data after 50 epochs.

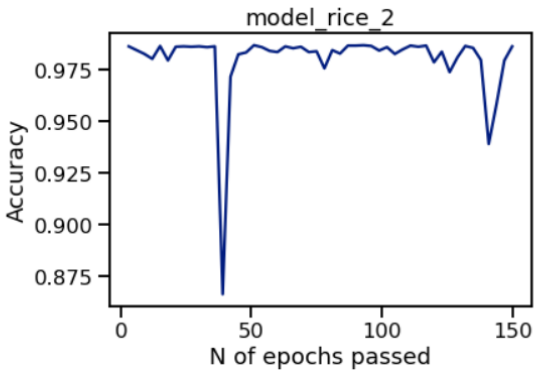
2) The validation accuracy of classification of the rice type is already very good, but I leverage another NN model with 3 hidden layers to see whether the validation accuracy can be decreased.

The new model summary is shown below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 20)                220
_________________________________________________________________
dense_17 (Dense)             (None, 20)                420
_________________________________________________________________
dense_18 (Dense)             (None, 6)                 126
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 7
=================================================================
Total params: 773
Trainable params: 773
Non-trainable params: 0
_________________________________________________________________
```

The hidden layers use 'relu' activation function, while again the final output layer uses sigmoid activation function to predict the class. The compilation has been done with an Adam optimizer (learning rate=0.0005). I have fitted the model through 150 epochs and batch size of 256.

The resulting maximum accuracy is the same as before, around 98.7 %, which is again reached after a few epochs:




```
[190]: valacc_max_model_rice_2=max(val_accuracy)
       print('Maximal validation accuracy for model_rice_2 is {:.3f}'.format(valacc_max_model_rice_2))

       Maximal validation accuracy for model_rice_2 is 0.987
```

3) To have some reference with Supervised machine learning. I have performed the classification with RandomForestClassifier model. In this case, I wanted to verify whether the neural networks can do better in this case. The number of estimators is set to 50.

However, again the model achieved the accuracy of 98.9 %, just like for two previous NN models:

```
B [194]: rf=RandomForestClassifier(n_estimators=50)
         rf=rf.fit(X_rice_train,y_rice_train)

B [196]: y_pred=rf.predict(X_rice_test)
         print('Accuracy with Random Forest is {:.3f}'.format(accuracy_score(y_rice_test,y_pred)))

         Accuracy with Random Forest is 0.989
```

So, the deep learning networks do not performed better than a conventional machine learning ensemble methods for classification of rice type in this dataset.

Key findings

The deep learning model with convolutional neural networks provided more than 91% accuracy to recognize a cloth type from the Fashion-MNIST dataset. The first model having strides of (1,1) led to a bit faster convergence to high accuracy. The fitting has been performed within 4 minutes, and prediction of the object demonstrated on each image is almost always correct.

Based on my results from the rice type prediction with neural networks, I can say that they perform very well. Both the deep neural networks and networks with only one hidden layer lead to over 98 % accuracy of determination of the rice type prediction. Nevertheless, in the current example this doesn't outperform the conventional ML classifiers such as Random Forest.

Possible suggestions and future analysis

For Fashion-MNIST data, I believe that a better choice of activation functions, or a more advanced architecture of neural network would perform a bit better. Although, the current accuracy of 91 % is close to perfection.

As for the rice type classification, the models perform well enough on this dataset with accuracy of nearly 99%. So, I believe the irreducible error will not allow to increase this score more.