Digit recognizer

Matej Šembera, 26.01.2023

Introduction: This project is my solution to kaggle comptetition on Kaggle called Digit recognizer. link to competition

Goal: In this competition we will try to identify numbers from 0 to 9 that were handwritten.

Sections: Before we will dive in the machine learning we will try to explore and prepare the data. To make this notebook transparent, I divided it into several parts.

1. Importing libraries, files and getting know the data
2. K-Nearest neighbors
3. CNN

Sections: We achieved 96,65% accuracy.

1. Importing libraries, files and getting know the data

In [1]:
```
#importing basic libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# K-nearest neighbors
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions

# CNN
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:
```
test = pd.read_csv("C:\\Users\\sembm\\OneDrive\\Dokumenty\\Data_Science\\Kaggle\\Digit recognizer\\test.csv")
train = pd.read_csv("C:\\Users\\sembm\\OneDrive\\Dokumenty\\Data_Science\\Kaggle\\Digit recognizer\\train.csv")
```

In [3]:
```
test.head(5)
```

Out[3]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 784 columns

In [4]:
```
train.head(5)
```

Out[4]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [5]:
```
train.isna().sum().sum()
```

Out[5]:
```
0
```

No missing values in our dataset

In [6]:
```
test.isna().sum().sum()
```

Out[6]:
```
0
```

2. K-nearest neighbors

In [7]:
```
#spliting the data
labels = train["label"]
pixels = train.drop(["label"], axis=1)
```

In [8]:
```
(X_train, X_test, Y_train, Y_test) = train_test_split(pixels, labels, test_size = 0.1, random_state=42)
```

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

Out[9]:
```
(37800, 784)
```

Finding the optimal number of neighbors

In [10]:
```
numbers = np.arange(2, 10)
scores = []
for n in numbers:
    model = KNeighborsClassifier(n_neighbors = n)
    score = cross_val_score(model, X_train, Y_train, cv=5)
    scores.append(score.mean())
scores
```

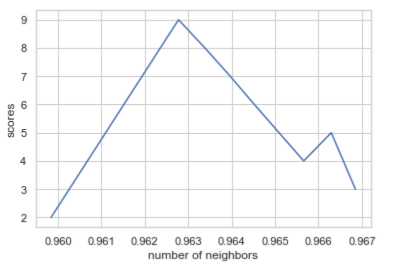
Out[10]:
```
[0.9598412698412698,
 0.9668518518518517,
 0.9656613756613759,
 0.9662962962962963,
 0.9645238095238096,
 0.9639682539682539,
 0.9633862433862432,
 0.9627777777777776]
```

We can see that most optimal score was achieved for n = 3

In [11]:
```
sns.set(style='whitegrid')
graph = sns.lineplot(scores, numbers)
graph.set(xlabel = "number of neighbors", ylabel = "scores")
```

```
C:\Users\sembm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positiona
l argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[11]:
```
[Text(0.5, 0, 'number of neighbors'), Text(0, 0.5, 'scores')]
```



K-Nearest neighbors is not the best algorithm for this project. The CNN is more suitable.

2. CNN

In [12]:
```
# Loading and preprocessing the data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Defining the model architecture
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compilling and training the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=128, epochs=10, validation_data=(X_test, y_test))
```

```
Epoch 1/10
469/469 [==============================] - 14s 27ms/step - loss: 0.2153 - accuracy: 0.9378 - val_loss: 0.0847 - val_accuracy: 0.9734
Epoch 2/10
469/469 [==============================] - 12s 27ms/step - loss: 0.0692 - accuracy: 0.9798 - val_loss: 0.0526 - val_accuracy: 0.9830
Epoch 3/10
469/469 [==============================] - 12s 26ms/step - loss: 0.0463 - accuracy: 0.9860 - val_loss: 0.0480 - val_accuracy: 0.9833
Epoch 4/10
469/469 [==============================] - 12s 26ms/step - loss: 0.0333 - accuracy: 0.9900 - val_loss: 0.0466 - val_accuracy: 0.9860
Epoch 5/10
469/469 [==============================] - 12s 26ms/step - loss: 0.0255 - accuracy: 0.9923 - val_loss: 0.0406 - val_accuracy: 0.9853
Epoch 6/10
469/469 [==============================] - 12s 27ms/step - loss: 0.0182 - accuracy: 0.9943 - val_loss: 0.0420 - val_accuracy: 0.9860
Epoch 7/10
469/469 [==============================] - 13s 27ms/step - loss: 0.0138 - accuracy: 0.9960 - val_loss: 0.0420 - val_accuracy: 0.9865
Epoch 8/10
469/469 [==============================] - 12s 26ms/step - loss: 0.0112 - accuracy: 0.9967 - val_loss: 0.0409 - val_accuracy: 0.9872
Epoch 9/10
469/469 [==============================] - 12s 26ms/step - loss: 0.0083 - accuracy: 0.9976 - val_loss: 0.0441 - val_accuracy: 0.9865
Epoch 10/10
469/469 [==============================] - 12s 27ms/step - loss: 0.0053 - accuracy: 0.9987 - val_loss: 0.0452 - val_accuracy: 0.9876
```

Out[12]:
```
<keras.callbacks.History at 0x2b0916192b0>
```

In [13]:
```
# Getting predictions on the test set
y_pred = model.predict(X_test)

# Converting the predicted probabilities to class labels
y_pred_class = np.argmax(y_pred, axis=1)
y_test_class = np.argmax(y_test, axis=1)

# Calculating the accuracy score
accuracy = accuracy_score(y_test_class, y_pred_class)
print('Accuracy:', accuracy)

# Printing the confusion matrix
print('Confusion matrix:')
print(confusion_matrix(y_test_class, y_pred_class))

# Printing the classification report
print('Classification report:')
print(classification_report(y_test_class, y_pred_class))
```

```
313/313 [==============================] - 1s 3ms/step
Accuracy: 0.9876
Confusion matrix:
[[ 973    0    0    0    0    1    3    1    2    0]
 [   0 1128    2    0    0    0    4    0    1    0]
 [   1    1 1009    2    2    1    3    7    5    1]
 [   0    0    1 1005    0    2    0    1    1    0]
 [   0    0    0    0  972    0    1    0    0    9]
 [   1    0    0    6    0  881    4    0    0    0]
 [   3    1    0    1    1    1  951    0    0    0]
 [   0    2    8    1    0    0    0 1014    1    2]
 [   4    0    3    1    0    2    1    2  955    6]
 [   1    2    0    3    4    4    0    6    1  988]]
Classification 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      0.98      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.98      0.99      0.99       958
           7       0.98      0.99      0.98      1028
           8       0.99      0.98      0.98       974
           9       0.98      0.98      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
```

In [14]:
```
test = test.values.reshape(-1,28,28,1)
result2 = model.predict(test)
result2
```

```
875/875 [==============================] - 3s 3ms/step
```

Out[14]:
```
array([[0., 0., 1., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 1., ..., 0., 0., 0.]], dtype=float32)
```

In [15]:
```
result2 = np.argmax(result2,axis = 1)
result2 = pd.Series(result2,name="Label")
result2
```

Out[15]:
```
0        2
1        0
2        9
3        0
4        3
        ..
27995    9
27996    7
27997    3
27998    9
27999    2
Name: Label, Length: 28000, dtype: int64
```

In [16]:
```
submission = pd.concat([pd.Series(range(1,28001),name = "ImageId"),result2],axis = 1)
submission.to_csv("digit_recognizer.csv",index=False)
submission
```

Out[16]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	0
4	5	3
...	...	...
27995	27996	9
27996	27997	7
27997	27998	3
27998	27999	9
27999	28000	2

28000 rows × 2 columns

Our current model overfits. The model says that he got accuracy of 99%. But the submission had an accuracy only of 94%. We will try to solve by including the dropout.

In [17]:
```
from keras.layers import Dropout
```

In [18]:
```
# Defining the model architecture
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# Compiling and training the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=128, epochs=10, validation_data=(X_test, y_test))
```

```
Epoch 1/10
469/469 [==============================] - 16s 32ms/step - loss: 0.3599 - accuracy: 0.8902 - val_loss: 0.1057 - val_accuracy: 0.9676
Epoch 2/10
469/469 [==============================] - 15s 32ms/step - loss: 0.1413 - accuracy: 0.9588 - val_loss: 0.0654 - val_accuracy: 0.9789
Epoch 3/10
469/469 [==============================] - 15s 32ms/step - loss: 0.1038 - accuracy: 0.9692 - val_loss: 0.0511 - val_accuracy: 0.9823
Epoch 4/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0865 - accuracy: 0.9736 - val_loss: 0.0473 - val_accuracy: 0.9838
Epoch 5/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0760 - accuracy: 0.9773 - val_loss: 0.0428 - val_accuracy: 0.9861
Epoch 6/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0686 - accuracy: 0.9792 - val_loss: 0.0371 - val_accuracy: 0.9871
Epoch 7/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0617 - accuracy: 0.9807 - val_loss: 0.0380 - val_accuracy: 0.9870
Epoch 8/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0558 - accuracy: 0.9823 - val_loss: 0.0387 - val_accuracy: 0.9865
Epoch 9/10
469/469 [==============================] - 15s 31ms/step - loss: 0.0529 - accuracy: 0.9840 - val_loss: 0.0361 - val_accuracy: 0.9870
Epoch 10/10
469/469 [==============================] - 15s 31ms/step - loss: 0.0481 - accuracy: 0.9847 - val_loss: 0.0358 - val_accuracy: 0.9881
```

Out[18]:
```
<keras.callbacks.History at 0x2b0925a38e0>
```

In [19]:
```
result3 = model.predict(test)
result3 = np.argmax(result3,axis = 1)
result3 = pd.Series(result3,name="Label")
submission3 = pd.concat([pd.Series(range(1,28001),name = "ImageId"),result2],axis = 1)
submission3.to_csv("digit_recognizer3.csv",index=False)
submission3
```

```
875/875 [==============================] - 3s 3ms/step
```

Out[19]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	0
4	5	3
...	...	...
27995	27996	9
27996	27997	7
27997	27998	3
27998	27999	9
27999	28000	2

28000 rows × 2 columns