Tieto_internship2

June 14, 2020

0.1 X and O classification

[54]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import cv2
import matplotlib.pyplot as plt
from PIL import Image
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from sklearn.metrics import classification_report, confusion_matrix
```

We will start by loading dataset. I took some pictuers of X and O written on paper by pen. Then I generated some modifications of that pictures resulting in dataset containing 3645 samples in `XOtrain` folder.

[55]:
```
import cv2
data = np.array([])
for i in range(7,3652):
    image = np.array(Image.open("X_and_O/XOtrain/" + str(i) + ".jpg").
 ↪resize((28,28)))
    data = np.append(data, image)
```

[56]:
```
data = data.reshape(3645, 28, 28, 3)
```

[57]:
```
data_X = np.concatenate((data[0:123], data[246:1640], data[2050:2091],␣
 ↪data[2501:2542], data[2952:2993], data[3403:3444]))
data_O = np.concatenate((data[123:246], data[1640:2050], data[2091:2501],␣
 ↪data[2542:2952], data[2993:3403], data[3444:]))
```

I divided pictures to two arrays and added labels to them. Let's devide them to training data and testing data.

[58]:
```
data_X_train = data_X[:1500]
data_X_test = data_X[1500:]
data_O_train = data_O[:1500]
```

```
data_0_test = data_0[1500:]
```

[59]:
```
data_train_y = np.array([1 for i in range(len(data_X_train))] + [0 for i in␣
↪range(len(data_0_train))])
data_test_y = np.array([1 for i in range(len(data_X_test))] + [0 for i in␣
↪range(len(data_0_test))])
data_train_X = np.concatenate((data_X_train, data_0_train))
data_test_X = np.concatenate((data_X_test, data_0_test))
data_train_y = to_categorical(data_train_y)
data_test_y = to_categorical(data_test_y)
```

Now when we have our data ready, let's choose a model. For binary picture classification is the best choice convolutional neural network. We will build few convolutional layers for picture preprocessing and then few NN layers for classification

[60]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('softmax'))

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

model.fit(data_train_X, data_train_y, epochs=8, batch_size=32)
```

```
Epoch 1/8
3000/3000 [==============================] - 2s 744us/step - loss: 2.0254 -
accuracy: 0.5973
Epoch 2/8
3000/3000 [==============================] - 2s 635us/step - loss: 0.4085 -
accuracy: 0.8187
```

```
Epoch 3/8
3000/3000 [==============================] - 2s 642us/step - loss: 0.2247 -
accuracy: 0.91100s - loss:
Epoch 4/8
3000/3000 [==============================] - 2s 637us/step - loss: 0.1273 -
accuracy: 0.9543
Epoch 5/8
3000/3000 [==============================] - 2s 602us/step - loss: 0.1346 -
accuracy: 0.9527
Epoch 6/8
3000/3000 [==============================] - 2s 601us/step - loss: 0.1116 -
accuracy: 0.9657
Epoch 7/8
3000/3000 [==============================] - 2s 601us/step - loss: 0.0805 -
accuracy: 0.9707
Epoch 8/8
3000/3000 [==============================] - 2s 597us/step - loss: 0.0607 -
accuracy: 0.9823
```

[60]:
```
<keras.callbacks.callbacks.History at 0x1a0bcafe490>
```

After we trained our model let's evaluate it. Let's make confusion matrix and compute accuracy on testing data.

[61]:
```
pred_y = model.predict(data_test_X)
pred_y_labels = np.argmax(pred_y, axis=1)
```

[62]:
```
from sklearn.metrics import classification_report, confusion_matrix

true_y_labels = np.argmax(data_test_y, axis=1)
print("\nConfusion Matrix")
print(confusion_matrix(true_y_labels, pred_y_labels))
print("\nClassification Report")
target_names = [str(i) for i in range(2)]
count = 0
for i in range(len(true_y_labels)):
    if true_y_labels[i] == pred_y_labels[i]:
        count += 1
count/len(true_y_labels)
```

```
Confusion Matrix
[[452  12]
 [  0 181]]

Classification Report
```

[62]:
```
0.9813953488372092
```

As we see convolutional NN was very good choice, we reached approximately 98% accuracy.

0.2 Application for prediction

You can find application source code in `X_and_O` folder. It takes one positional argument of possible output (possible extension to save image you took by it). As I didn't added the extension yet don't be confused by useless argument.

You can execute application by terminal. Open the folder `X_and_O` (which contains all the source code you need and training set) and execute exactly this command: `python XOclassifier.py --output output`.

Give it about 2 minutes because application has to train the model and run your webcam. In application you just need to put X or O written on paper (it would be best if you wrote it by blue pen, because training set was build on the letters written by blue pen) in the front of your camera. Then click the button `Snapshot!`. On the right corner should pop off the window with the final classification of your photo. You can take more photos if you want, application is properly running until you turn it off.

0.3 Conclusion

As I mentioned before, the best model for such task is convolutional NN. As we can see it is performing really good on testing dataset. I experienced some imperfections though, during testing the application. They were mainly caused because I used pictures with X or O written by some different pen or captured by different angles. This can be fixed by altering training dataset and adding more different photos in it.

I see potentional improvements mainly in data preprocessing with more time we could add data augmentation before entering the model. CNN `fit_generate` might be better than my `generator` as well so that is another possible way of improvement. Another idea that came to my mind is the in application I could use some existing trained model saved somewhere so it wouldn't load so long in the beginning.

I hope that you like my work on this task and thank you for this opportunity.

Marek Jankola