Computer Vision

Project: Image classifier with sensitive data recognition

Authors:

Adrian Gwoździej and Artur Liszewski

***Abstract*—** The raport reference to the image classifier issue with sensitive data recognition feature. Project consist two main steps, recognize among three different classes. If the classifier will return credit card number, then the algorithm will hide the number, encrypt and store as a key

Keywords—Deep Learning, Classification, Algorithm, Vision, Neural Networks

I. Introduction

Computer vision and deep learning networks (especially convolutional neural networks) are very crucial tools to read content of plenty of images in order to create classification. Using computer vision allows the user to read and manipulate data. During this project computer vision was used to recognize set of credit card numbers using special OCR pattern image, which involve numbers from 0 to 9 in special font known as OCR font. Deep Learning was used in order to create convolutional neural network which involve certain numbers of hidden layers.

II. Programming Environment

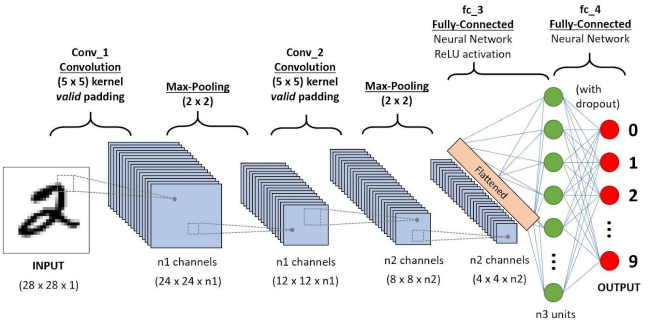
Python programming language in version 3.7 was used to create convolutional neural network as well as to create computer vision sensitive data recognition from credit cards. Tensorflow and Keras was used on the Jupyter notebook environment due to technical problems with right processor architecture. In this case the only way to use Tensorflow and Keras libraries is using Jupyter notebook which include these packages.

Also It was used many other important libraries like cv2, which involves computer vision functions like reading images and treat it as an array of numbers.

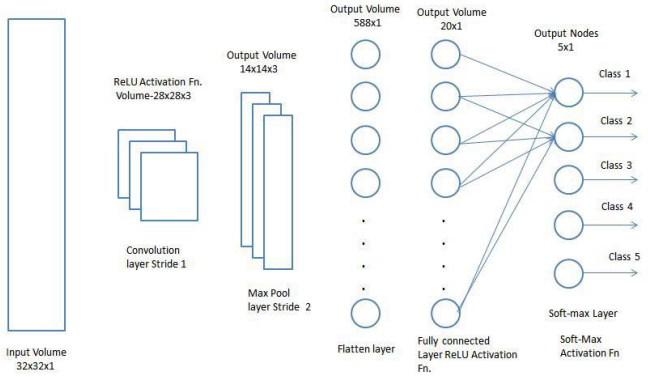
For visualisation and creating chart, the library matlib was implemented

III. Convolutional Neural Networks

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



Picture 1. Scheme of CNN



Picture 2. Fully connected CNN

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. [1]

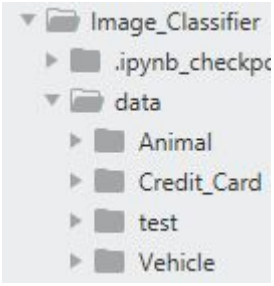
IV. Image Classification

First step is to implement the array of categories where we are deciding how many classes are needed with what names.

In this project there are only three categories: Animals, Credit cards, Vehicles. The more classes are added, the longer the training of the model will last. Unfortunately, a computer which have already trained the model is quite old without any CUDA tensors which are crucial in order to training models in the proper way. For not commercial tasks

and just only for presenting main core of the project it is totally enough

The next important step is creating a folder with set of images for every each category. It is important to care about similar number of pictures, because in other way the whole model may be unbalanced.



Picture 2. Nested Tree with categories within project directory

In programming environment declared the image size. Even when we will analyze certain image with different size, the algorithm will compress the image to the indicated size. The nature of convolutional neural network is about analyzing every single pixel within the image. It means that for bigger images, training process takes longer. During the project the image size was declared at 100x100 pixels. Every single pixel is represented in the array in range between 0 and 255 which include the color + transparency value also between within mentioned threshold.

Next step is building model. We have to decide how many convolutional layers do we need and also how many hidden layers. In this project has been selected 3 convolutional layers and 2 hidden layers with softmax activation. The model was compiled with optimizer “adam”.

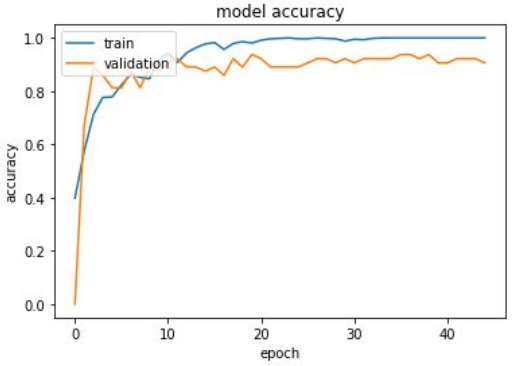
```
062
Epoch 41/45
573/573 [==============================] - 38s 67ms/sample - loss: 2.9772e-04 - acc: 1.0000 - val_loss: 0.9511 - val_acc: 0.9
062
Epoch 42/45
573/573 [==============================] - 37s 65ms/sample - loss: 4.3293e-04 - acc: 1.0000 - val_loss: 0.9503 - val_acc: 0.9
219
Epoch 43/45
573/573 [==============================] - 37s 64ms/sample - loss: 4.3330e-04 - acc: 1.0000 - val_loss: 0.9376 - val_acc: 0.9
219
Epoch 44/45
573/573 [==============================] - 37s 65ms/sample - loss: 8.1518e-04 - acc: 1.0000 - val_loss: 0.9388 - val_acc: 0.9
219
Epoch 45/45
573/573 [==============================] - 35s 62ms/sample - loss: 5.2593e-04 - acc: 1.0000 - val_loss: 1.0153 - val_acc: 0.9
062
Saved model to disk
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

Picture 3. Epochs with loss values and accuracy

The last phase involves model.fit, when we are declaring how many epochs do we want to implement

model.fit(X, y, batch_size=32, epochs=45, validation_split=0.1)

```
Out[116]: <matplotlib.legend.Legend at 0x14500ab5a90>
```



Picture 4. Chart which shows train and validation values along epochs

The best situation is getting accurace close to the 1 and loss value close to the zero.

Last thing is to call the model for our data with resizing method. Algorithm has to compare trained model to the new unknown data and the size and shape must be the same.

```
def prepare(file):
    IMG_SIZE = 100
    img_array = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
model = tensorflow.keras.models.load_model("CNN3.model")
image = prepare("data/test/01.png")
prediction = model.predict([image])
prediction = list(prediction[0])
print(CATEGORIES[prediction.index(max(prediction))])
```

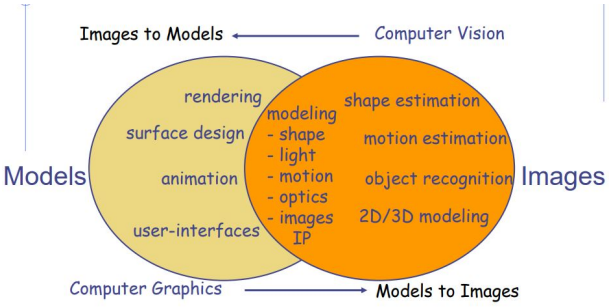
```
Credit_card
```

Now the whole process is completed. For the non commercial tasks this model is good enough to recognize data within 3 classes and mostly it is working fine.

V. Computer Vision

Computer vision is a field of study focused on the problem of helping computers to see. The goal of computer vision is to understand the content of digital images. Typically, this involves developing methods that attempt to reproduce the capability of human vision.

Understanding the content of digital images may involve extracting a description from the image, which may be an object, a text description, a three-dimensional model, and so on. [2] For this project we are focusing on the Optical character recognition (OCR)



Picture 5. Scheme of Vision diagram[3]

VI. Optical Character Recognition

Main tasks along the Credit Card recognition pattern

- Localize the four groupings of four digits on a credit card.
- Extract each of these four groupings followed by segmenting each of the sixteen numbers individually.
- Recognize credit card digits by using template matching and the OCR font.

Picture 6. OCR credit card pattern reference

Algorithm recognize contours from above picture and sorts from left to right. Algorithm also computes bounding box around each contour. Every digit is resized to a fixed size in order to apply template for digit recognition

Next goal is to isolate the 16-digit credit card number Algorithm needs to find and isolate the numbers before can initiate template matching to identify each of the digits. Kernel as a small matrix which algorithm slide across the image to do (convolution) operations such as blurring, sharpening, edge detection, or other image processing operations.

The next step is to convert image to grayscale. Converting the image to grayscale is a requirement prior to applying the rest of the image processing pipeline. Also it is helpful to use “Top-hat” operation which reveals light regions against a dark background.

Picture 7. Reveals light regions within black background

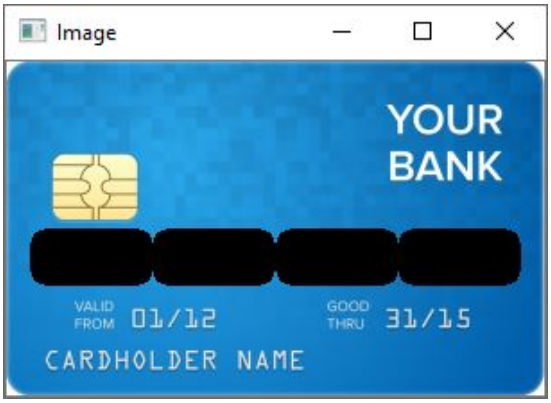
Next step is to isolate the digits using Scharr gradient computation of the top-hat image in the x direction.

Picture 8. Schaar gradient magnitude representation

Picture 9. Reveals candidate regions for the credit card numbers

Lastly the algorithm founds the contours and initialize the list of digit grouping locations. On the top of every each digit will be generate a rectangle, which hide every single number on a credit card.

VII. Result



Picture 10. Every digit is hidden by a rectangle

The last step is to read credit card number and encrypt it and also read hoster's name.

```
(base) C:\Users\Adi\Documents\Coding\Artificial Intelligence\Machine Learning\Python\Sensiti
ve_Data>main2.py --reference images/OCR_Images/ocr_a.png --image images/Credit_Cards/credit0
.jpg
Credit Card #: 4201324753885103
Credit Card Type: Visa
CARDHOLDER NAME
```

Picture 11. Output result from the algorithm

While we are getting this data we are running encrypting function which allows to store whole the data in certain safe form.

```
Path: images/Credit_Cards/credit0.jpg
    Credit card number: gAAAAABeEvG6R7ZrcTGhTbiX7_2vvRRiwd3
    Credit card type: Visa
    Hoster's name: CARDHOLDER NAME
```

Picture 12. Stored data in a safe manner

Hoster's name was recognized using tesseract library, which was created to recognize text on the top of the image but it is working very good only with pure white background which is rarely with credit cards. This is no need to hide hoster's name as long as credit card number has already hidden. Algorithm has to have ideal conditions to do it in properly way. Without good enough quality algorithm fails.

Picture 13. The algorithm fails due to bad quality image or when numbers color is similar to the background

VIII. Conclusion

Unfortunately, we were not able to apply OCR images to real credit card images, so that certainly raises the question if this method would be reliable on actual, real-world images. Given changes in lighting condition, viewpoint angle, and other general noise, it's likely that we might need to take a more machine learning oriented approach.

Regardless, at least for several example images, with good enough quality and without any noises which may impact on digits algorithm was able to successfully apply template matching as a form of OCR. This project shows that machine learning approach can reach better results in this case and computer vision problems are closely correlated with quality of image and reference patterns. Machine Learning has prediction features which can be helpful during work with bad quality images or when some digit might be even imposed by some noise.

IX. Reference

[1]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2]https://vision.unipv.it/CV/1.%20Introduction.pdf

[3]https://machinelearningmastery.com/what-is-computer-vision/