Group 17 - CS 178 Final Report

Group Member:

Yingyu Yang

Jiaqian Wu

Junyang Jin

Problem Statement

The problem we chose for this project is **license plate recognition** (LPR). After dataset training, our program is capable of recognizing characters in preprocessed images of vehicle registration plates with a high accuracy rate.

Task Decomposition

We decomposed the task at hand into the following 6 parts (the member responsible for each part is attached accordingly):

- Brainstorm for the project topic.
    - Jiaqian Wu, Junyang Jin, Yingyu Yang
- Research for related approaches and datasets
    - Jiaqian Wu, Junyang Jin, Yingyu Yang
- Build the CNN model for digit recognition using the MNIST dataset (as an intermediate step that prepares the model for the more complex task).
    - Jiaqian Wu, Junyang Jin, Yingyu Yang
- Build the CNN model for license plate recognition using Russian license plates.
    - Jiaqian Wu, Junyang Jin, Yingyu Yang
- Report write up.
    - Yingyu Yang
- Make the demo poster.
    - Jiaqian Wu, Junyang Jin, Yingyu Yang

Approach

Digit Recognition

For digit recognition, we used the convolutional neural network (CNN) with Keras as the backend supporting library. Our model has three groups of convolutional layers, a flatten layer, and a fully-connected layer. We trained and tested our model on the MNIST dataset, which is a large database of handwritten digits. Our model's accuracy rate for both the training data and the testing data (which is relatively smaller) reached 1.0 at the 20th epoch (with gradient descent). Since the result gave us a pretty confident accuracy rate, we decided to adapt the same model for license plate recognition. Therefore, please refer to "Build the Model" section under "License Plate Recognition" for more details on the model.

License Plate Recognition

For license plate recognition, we divided the problem into four subproblems, which are listed below. The following section will explain how we solved each subproblem in detail.

1. Preprocess raw images of license plates.
2. Build the network model with training data.
3. Validate the network model on test data.
4. Decode the output (from single characters to a complete license plate).

Preprocess

In the preprocessing step, we confined each character on the license plate with a bounding box using findContour and boundingRect function from OpenCV. We limited the size of the box to extract to avoid getting irrelevant contour boxes. The screenshot below (fig 1.1) shows how we implemented it. (see Citation section about this implementation)

```
imgray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(imgray, 127, 255, 0)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (1, 5))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for contour in contours:
    # get rectangle bounding contour
    (x,y,w,h) = cv2.boundingRect(contour)

    # discard areas that are too large
    if h > 60 and w > 60:
        pass
        continue

    # discard areas that are too small
    if h < 30 or w < 10:
        pass
        continue

    # draw rectangle around contour on original image
    # plt.imshow(imgray[x:x+w,y:y+h])

    detected.append((x,y,w,h))
```

Fig 1.1

We then extracted characters from license plates using detected coordinates and size of the bounding boxes combined with numpy index slicing. We resized each character image so that they are of the same size. This is because the input size for convolutional neural network layers should stay the same. Extracted character images look like something below (fig 1.2). The image preprocess step prepares the dataset for model training of each individual character.

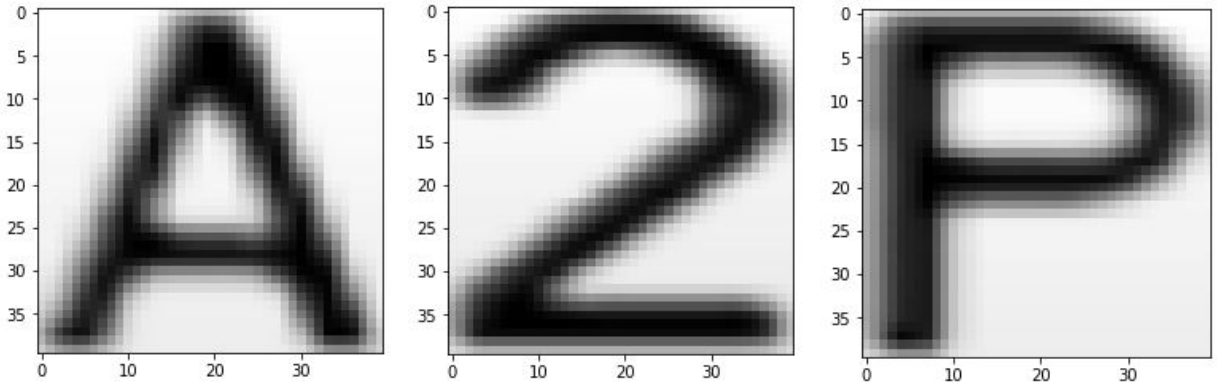


Fig 1.2

Build the Model

In this step, we built our model based on the convolutional neural network we had for digit recognition. We used Keras as the supporting library, which uses the TensorFlow backend. We developed two models, and chose the one that had the highest accuracy rate.

Our final model has three groups of layers. Each layer group consists of a 2D convolutional layer, an activation function, and a pooling layer. We set the dimensionality of the output space (filters) to 32 for the first convolutional layer, 64 for the second convolutional layer, and 128 for the third convolutional layer. Then, we used leaky rectified linear units (ReLU) as the activation function. Here, we chose leaky ReLU over normal ReLU to avoid the “dead ReLU” problem, which happens when the ReLU always have values under 0. After this, we added the max pooling layer with a kernel of dimensions 2 * 2.

After building convolutional layers, we added a flatten layer to flatten the output of the network for the final fully-connected layer. Our fully-connected layer consists of two dense layers. The first one uses leaky ReLU as its activation function. The second one uses softmax function for the output layer.

The screenshot below (fig 1.3) provides a summary of our CNN model.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 40, 40, 32)	320
leaky_re_lu_1 (LeakyReLU)	(None, 40, 40, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 20, 20, 32)	0
conv2d_2 (Conv2D)	(None, 20, 20, 64)	18496
leaky_re_lu_2 (LeakyReLU)	(None, 20, 20, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 10, 10, 64)	0
conv2d_3 (Conv2D)	(None, 10, 10, 128)	73856
leaky_re_lu_3 (LeakyReLU)	(None, 10, 10, 128)	0
max_pooling2d_3 (MaxPooling2	(None, 5, 5, 128)	0
flatten_1 (Flatten)	(None, 3200)	0
dense_1 (Dense)	(None, 128)	409728
leaky_re_lu_4 (LeakyReLU)	(None, 128)	0
dense_2 (Dense)	(None, 36)	4644

Total params: 507,044
Trainable params: 507,044
Non-trainable params: 0

Fig 1.3

Validate the Model

We trained and tested our model using the fit function provided by the Keras library. For training, we set the number of samples per gradient update (batch size) to 64, and the number of epochs to 20. We also provided the fit function with test data for cross validation. At the 20th epoch, our model's accuracy rate on both the training data and the testing data reaches 1.0. For further information on testing results, please refer to the "Result" section.

Decode the Output

Now that we've finished training and testing our model, it's time to apply it to some test cases and observe the actual result. Since our model is only responsible for recognizing single characters, we need to decode the output to return an expected value, that is, to assemble individual predictions back together to form the plate number. The screenshot below (fig 1.4) shows how we implemented it.

```
# Doing things in the background
showcase1 = extract_digit_image(validation_showcase_file)
temporary_storage = []
# temporary_storage = model.predict_classes(showcase1)
for i in range(len(showcase1)):
    temporary_storage.append(model.predict(np.array(showcase1[i]).reshape(-1,40,40,1)))

# print(np.argmax(temporary_storage[1]))
# print("Letters: ", letters)

showcase_label = labels_to_text([np.argmax(temporary_storage[i])
                                 for i in range(len(temporary_storage))])
print("The OCR result: ",showcase_label)
```

Fig 1.4

Citation

For the digit recognition part, we found some useful website with introduction of how to use MNIST dataset, which gave us basic ideas of how to use Keras to implement our model. We used the code cited from a DataCamp Community article, written by Aditya Sharma. ***The part of code we used included training and testing on MNIST digit dataset, and also visualizing the output. The copyright is reserved by Aditya Sharma.*** With the method provided on that website, we were able to visualize the layers of convolutional neural network and see validation accuracy outputs. We tried to design the layers for our convolutional network based on that article as well. We modified and added other convolution layers and fully connected layers to classify our data, and we finally decided to use the one included in this report after facing with problems of overfitting and reduced accuracy. (see experience for details)

We then focused on researching for ideas to recognize characters on license plates. At first we found a license plate recognition related article named “Latest Deep Learning OCR with Keras and Supervisely in 15 minutes”, written by a user named “Supervise.ly“. The approach of this article was to use both convolutional neural network and Recurrent Neural Networks to train and decode the whole plate images. This approach was also intriguing, but later we found another approach that brought us more inspiration. From this article, we get access to the Russian license plate dataset which we used for training and testing.

The approach that we finally decided to implement was inspired by the article “Number Plate Detection with a Multi-Convolutional Neural Network Approach with Optical Character Recognition for Mobile Devices”, which was written by Gerber, Christian and Mokdong Chung. The paper contained a whole process of license plate recognition, which started from detecting a car and then its license plate. Finally the model would use some way to separate each character on the license plate and train or test them separately. For these three tasks that involved detection or recognition, the paper stated that they used different convolutional neural network for each problem. Since our focus is on license plate recognition, we tried to implement the last network and other preprocessing steps needed for our own datasets. Basically, our ways of doing this is similar, which is to extract each characters out and then train them separately, just like what we did for each digit in MNIST dataset. Since we trained the characters separately, we came up with ideas to make our last step be decoding the predicted output back to a string of license plate characters. It worked successfully at last.

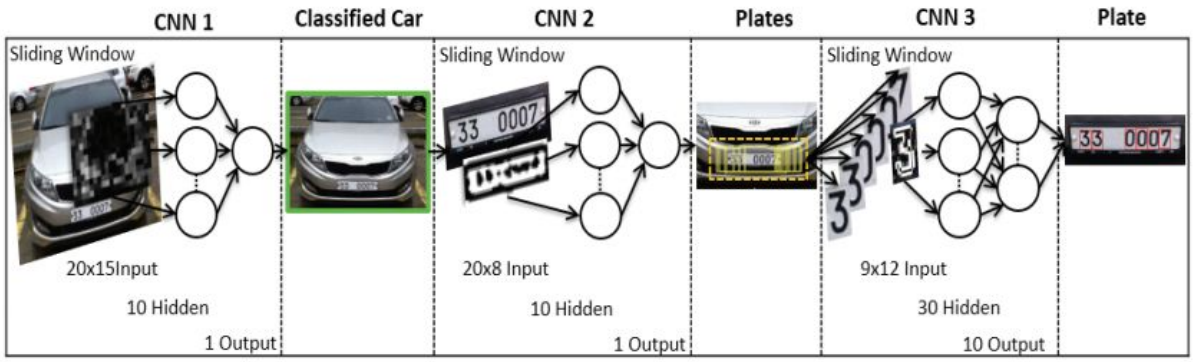


Fig. 3. Number plate classification based on multiple convolutional neural networks (CNNs).

Fig 1.5 Approach from cited article

In order to find bounding boxes of the characters, we read some helpful article about contour using skills. One of them was named “Recognizing digits with OpenCV and Python”, which was written by Adrian Rosebrock. It was about recognizing digits on digital alarm clocks. Though it seemed to be unrelated to our topic, the approach it used to extract the digits was what we were interested. It used opencv to turn an image in to binary image, and then used implemented contour finding function to find bounding box for each character. ***The part of code mentioned in Preprocess section is mainly from this article, we only modified it to fit our purpose. The copyright is reserved by Adrian Rosebrock. *** This was very helpful for us when we were confused about extraction of characters.

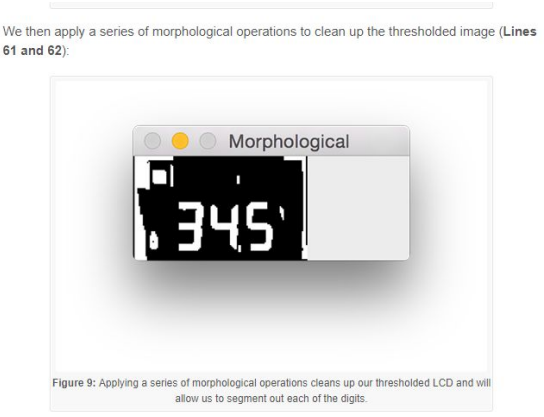


Fig 1.6 Binary image conversion example from the article

Experience

We experienced several milestones when completing this project. At first, we decided to choose digit recognition as our research subject. However, after realizing that this topic was way easier than what we'd expected, we decided to move on to a more complex and challenging problem, which, after hours of group discussion, turned out to be license plate recognition. However, despite the topic change, we sticked to the convolutional neural network all the way.

Initially, we wanted to keep our project doable, which is why we chose digit recognition. After pinning down the topic, we discussed it with the professor during office hours, and accepted his suggestion of using MNIST dataset as training and testing data, and the convolutional neural network as the learning model.

Then, we did some research on digit recognition and convolutional neural network. The details of our research can be found in the Citation section. With inspirations from the materials we read online, we built the convolutional neural network for digit recognition using three groups of convolutional layers, a flatten layer, and a fully-connected layer. The detailed information about our model can be found in the Approach section. We trained and tested our model using the MNIST dataset, and got a very good score. The data showed us that our model worked pretty well for the digit recognition problem.

With this successful experience, we moved on to a more complicated subject — license plate recognition. At first, we weren't sure about how to recognize a group of characters in a single image, until one of our group members (Jiaqian) found a paper that talks about license plate recognition in great detail. We studied their way of approaching the task at hand, and decided to solve one of the problems they described in the paper. More information about this paper can be found at the Citation section.

During the actual coding experience for license plate recognition, we first encountered the problem of resizing. After we get help from Professor Mjolsness, we decided look up how to implement bilinear interpolation in order to resize images. After some more research, we found a method that is taught in other course called warping, which can do the same thing as bilinear interpolation which is to scale an image differently along length and width. There is a homemade function for resizing included in our code, which would work fine for resizing images. However, as shown in the comparison in our code, when compared to CV2 library resizing function, the one we wrote ran far slower than

the library one. Therefore, for convenience, we used CV2 resizing function when training, though we did implement our own resizing function.

After the resizing problem was solved, we picked the dataset of Russian license plates, and applied our model for digit recognition to it. However, after first try, the validation accuracy for this model was only 0.17. Then, we realized that it was the preprocessing step that went wrong. We couldn't catch some of the characters on the license plates, because of the potential problem discussed in the following paragraph.

The problem is that we found our preprocess part, which is to separate each character from the license plate, could not handle skewed license plate. We figured out that it was our implementation using contour boxing that could have limited the ability to capture characters. In our poster, we also showed this problem. Therefore, we chose the current dataset which had better preprocessed images of license plates to avoid problems. In the future we could certainly improve our model to make it fit more circumstances, for example, to make it flexible with images that are skewed or of different background size.

In order to better understand this problem, we did an experiment to train and test our model on another license plate dataset. This dataset included many lower quality images. Most of the license plates in the database are shot at a skew angle. We wanted to test if our model could handle more complex situations. The result was quite disappointing. Our model couldn't recognize characters on these images correctly. We think the reason might be that it is harder to catch characters on lower quality photos. If we added another CNN model that is dedicated to extracting license plates in lower quality images, our model would have a better performance score. However, since this part is out of our research scope, we decided not to delve into it, and leave it as a future topic to study on.

Then, we trained and tested our model on the dataset again with a adjusted limitation for bounding box and the result turns out to be satisfying this time. The training and validation accuracy rates (on 500 validation data) we got for license plate recognition both reached 1.0 at the 20th epoch. This result proves that our model is successful, and can be capable of processing more complex data which is preprocessed.

During the coding process, we tested another CNN model to see which one is more suitable for the problem. The new model was the result of adding another group of convolutional layers (Conv2D, LeakyReLU, MaxPooling2D) to the original model. As a result, it has four groups of layers. Then we trained and tested the model on the same dataset. However, while the training accuracy rate stayed the same, the validation

accuracy rate for this model was a little lower than the original one. Thus, we concluded that adding an additional group of layers would lead to the case of overfitting, and kept our original model.

Result

The result of training and testing our model on the dataset using gradient descent (batch_size = 64, epochs = 20):

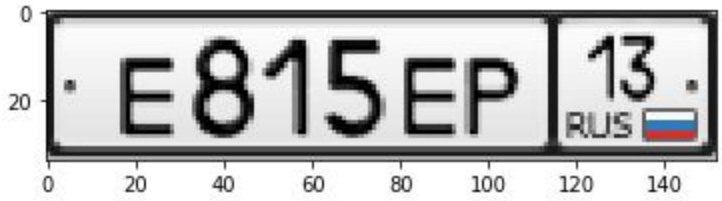
```
Train on 5000 samples, validate on 500 samples
Epoch 1/20
5000/5000 [==============================] - 13s 3ms/step - loss: 0.7861 - acc: 0.8018 - val_loss: 0.0624 - val_acc:
0.9620
Epoch 2/20
5000/5000 [==============================] - 12s 2ms/step - loss: 0.0100 - acc: 0.9966 - val_loss: 8.6398e-04 - val_a
cc: 1.0000
Epoch 3/20
5000/5000 [==============================] - 13s 3ms/step - loss: 3.8197e-04 - acc: 1.0000 - val_loss: 3.7068e-04 - v
al_acc: 1.0000
Epoch 4/20
5000/5000 [==============================] - 12s 2ms/step - loss: 1.9644e-04 - acc: 1.0000 - val_loss: 2.2552e-04 - v
al_acc: 1.0000
Epoch 5/20
5000/5000 [==============================] - 12s 2ms/step - loss: 1.2177e-04 - acc: 1.0000 - val_loss: 1.5363e-04 - v
al_acc: 1.0000
Epoch 6/20
5000/5000 [==============================] - 13s 3ms/step - loss: 8.6835e-05 - acc: 1.0000 - val_loss: 1.1769e-04 - v
al_acc: 1.0000
Epoch 7/20
5000/5000 [==============================] - 13s 3ms/step - loss: 6.4913e-05 - acc: 1.0000 - val_loss: 9.2886e-05 - v
al_acc: 1.0000
Epoch 8/20
5000/5000 [==============================] - 14s 3ms/step - loss: 4.9576e-05 - acc: 1.0000 - val_loss: 7.3068e-05 - v
al_acc: 1.0000
Epoch 9/20
5000/5000 [==============================] - 13s 3ms/step - loss: 3.9915e-05 - acc: 1.0000 - val_loss: 6.2735e-05 - v
al_acc: 1.0000

Epoch 10/20
5000/5000 [==============================] - 13s 3ms/step - loss: 3.1378e-05 - acc: 1.0000 - val_loss: 5.3361e-05 - v
al_acc: 1.0000
Epoch 11/20
5000/5000 [==============================] - 13s 3ms/step - loss: 2.6131e-05 - acc: 1.0000 - val_loss: 4.6448e-05 - v
al_acc: 1.0000
Epoch 12/20
5000/5000 [==============================] - 13s 3ms/step - loss: 2.1606e-05 - acc: 1.0000 - val_loss: 4.0399e-05 - v
al_acc: 1.0000
Epoch 13/20
5000/5000 [==============================] - 13s 3ms/step - loss: 1.8153e-05 - acc: 1.0000 - val_loss: 3.7088e-05 - v
al_acc: 1.0000
Epoch 14/20
5000/5000 [==============================] - 13s 3ms/step - loss: 1.5559e-05 - acc: 1.0000 - val_loss: 3.2482e-05 - v
al_acc: 1.0000
Epoch 15/20
5000/5000 [==============================] - 13s 3ms/step - loss: 1.3247e-05 - acc: 1.0000 - val_loss: 2.8616e-05 - v
al_acc: 1.0000
Epoch 16/20
5000/5000 [==============================] - 13s 3ms/step - loss: 1.1388e-05 - acc: 1.0000 - val_loss: 2.5939e-05 - v
al_acc: 1.0000
Epoch 17/20
5000/5000 [==============================] - 13s 3ms/step - loss: 9.8558e-06 - acc: 1.0000 - val_loss: 2.3890e-05 - v
al_acc: 1.0000
Epoch 18/20
5000/5000 [==============================] - 13s 3ms/step - loss: 8.8015e-06 - acc: 1.0000 - val_loss: 2.1886e-05 - v
al_acc: 1.0000
Epoch 19/20
5000/5000 [==============================] - 13s 3ms/step - loss: 7.8807e-06 - acc: 1.0000 - val_loss: 2.0503e-05 - v
al_acc: 1.0000
Epoch 20/20
5000/5000 [==============================] - 13s 3ms/step - loss: 7.0986e-06 - acc: 1.0000 - val_loss: 1.9242e-05 - v
al_acc: 1.0000
```

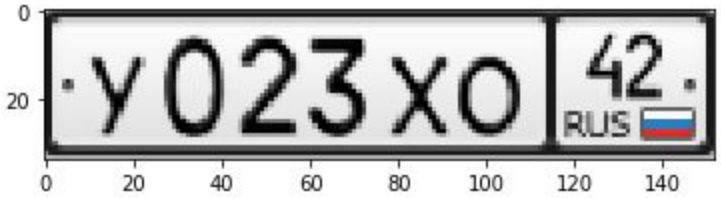
Note: acc refers to training accuracy; val_acc refers to validation accuracy

Sample outputs for our license plate recognition program:

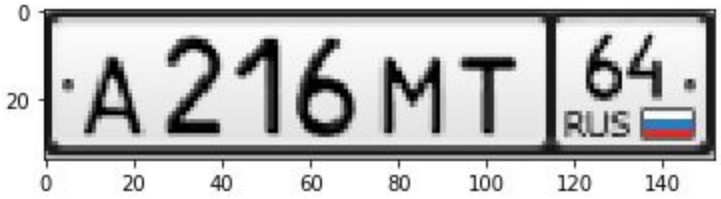
```
-------------- Input License Plate --------------
The OCR result:  E815EP13
```




```
-------------- Input License Plate --------------
The OCR result:  Y023XO42
```




```
-------------- Input License Plate --------------
The OCR result:  A216MT64
```



Conclusion

We started from analyzing MNIST dataset using convolutional neural network and then went over different readings and chose Keras as our library since it has its own implemented MNIST dataset and designing hidden layers is more convenient.

By using convolutional neural network in Keras, we can extract different features easily with combination of Conv2D and Max Pooling layers. As shown in our experiment, our result of training and testing has a reasonably high accuracy when validating on test data.

However, our program is very restrictive about requirements for preprocessing. From our experience, the preprocessed image can not be skewed or have different sizes of plates, due to our implementation of bounding boxes finding. Otherwise our preprocess to extract characters may miss some characters that would affect the matching of training data and labels. In order to make our program better, we should consider to build convolutional neural network for training and testing for other preprocessing like detecting a car and its car plate. We could train the model well that it can help us filter the license plate to be a reasonable processed state that our digit separation part of the program could figure out what to do. Then we could use our digit recognition CNN to classify them.

We faced other difficulties when implementing our approach, like preprocessing and decoding for output of our model. Nevertheless we came up with ideas through thinking and reading others' reference. We had more characters to recognize other than digits, which requires us to have a total of 36 types of characters to classify. With the advantage that our designed neural network could extract features well, the application of convolutional neural network part was successful. However, our model was still not perfect that the learned model sometimes could not distinguish digit 0 and alphabet O, or digits 6 and 8, most of other characters are correctly recognized. We think there are still further progress to be made to improve our model.

Bibliography

Sharma, Aditya. “Convolutional Neural Networks in Python with Keras.” *DataCamp*, 5 Dec. 2017, www.datacamp.com/community/tutorials/convolutional-neural-networks-python#comments.

Moghazy. “Convolutional Neural Networks with Data Augmentation using Keras.” *kaggle*, www.kaggle.com/moghazy/guide-to-cnns-with-data-augmentation-keras. Accessed 19 March 2019.

Karpathy, Andrej. “CS231n Convolutional Neural Networks for Visual Recognition.” *cs231n*, cs231n.github.io/convolutional-networks/. Accessed 19 March 2019.

Deshpande, Adit. “A Beginner's Guide To Understanding Convolutional Neural Networks.” *deshpande3*, 20 July 2016, adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

Supervise.ly. “Latest Deep Learning OCR with Keras and Supervisely in 15 minutes.” *hackernoon*, 2 November 2017, hackernoon.com/latest-deep-learning-ocr-with-keras-and-supervisely-in-15-minutes-34aecd630ed8.

Gerber, Christian, and Mokdong Chung. “Number Plate Detection with a Multi-Convolutional Neural Network Approach with Optical Character Recognition for Mobile Devices.” *Journal of Information Processing Systems*, vol. 12, no. 1, 2016, pp. 100-108.

Rosebrock, Adrian. “Recognizing digits with OpenCV and Python.” *PyImageSearch*, 13 Feb. 2017, www.pyimagesearch.com/2017/02/13/recognizing-digits-with-opencv-and-python/.