VINUNIVERSITY

# Final Report Programming Project

# Theme: *Developing a Web-based License Plate Recognition System using Python*

---

**COMP1010 Introduction to Programming** **Fall 2021**

## Contents

## 1 What is License Plate Recognition?

License Plate Recognition is basically a program that can detect, read numbers and characters of the license plate from a digital picture or video. Nowadays, license plate recognition is a futuristic solution for security and vehicle management operations in buildings such as stolen vehicles detection or ticketless parking, toll payment. Moreover, this technology can process immediately and independently without any installation. In this program, there are three main phases: license plate detection (find and extract the license plate), character segmentation (segment character into small pieces) and character recognition.

## 2 License plate detection

In the first stage, we will determine and extract license plate from an input image/frame. After the license plate is located and extracted from the whole picture, it has to be transformed into a standardised form for further processing: normalised background colours (normalised contrast and brightness values), size and uniform plate orientation.
To be more specific, we have used a pre-trained model Warped Planar Object Detection Network (WPOD-NET) to search for license plates in the original image. Next, this model will apply affine transformation on the extracted license plate and modify it to a frontal view.

### 2.1 Warped Planar Object Detection Network (WPOD-NET)

Warped Planar Object Detection Network [1] is used to detect license plate in different deformation and regresses coefficients of an affine transformation matrix that straighten the distorted license plate into a rectangular shape with a frontal view. That means in this network, our extracted license plate will go through a rectification process before entering the next phase.
The WPOD-NET was developed using insights from YOLO, SSD and Spatial Transformer Networks (STN). While YOLO and SSD can recognize multiple objects with a high speed, they can not apply transformation to "unwarp" our distorted license plate. On the other hand, STN can be used for detecting non-rectangular regions with only one spatial transformation over the entire input. Therefore, WPOD-NET was built as a solution with both fast objects detection and multiple transformation.

### 2.2 Network Architecture

The WPOD-NET architecture was built with 21 3x3 convolutional layers, while 14 of them are inside smaller blocks. Except the detection block, the entire network mainly used ReLU function as an activation. Additionally, there are 4 max pooling layers of size $2 \times 2$ and stride 2 that can keep the important features and reduce the input dimension. At the end, the detection block was constructed by two parallel convolutional layers: one using a softmax function to calculate the probability and another for regressing the affine parameters without activation.

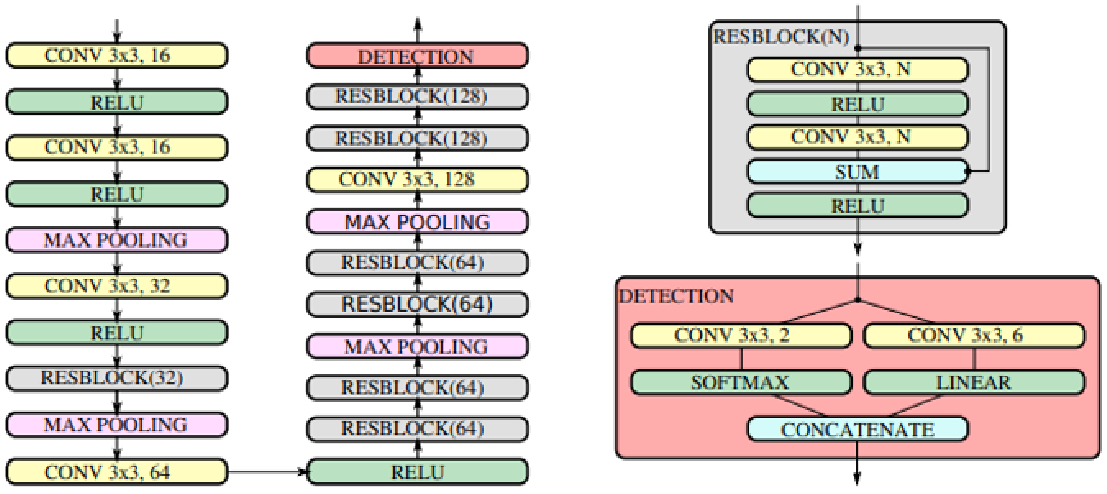


Figure 1: WPOD-NET architecture

### 2.3 Demo

From the full picture (Figure 2), the WPOD-NET can cut and extract the license plate (Figure 3). It is also noticeable that our license plate is straightened from a parallelogram into a rectangular shape with a frontal view.

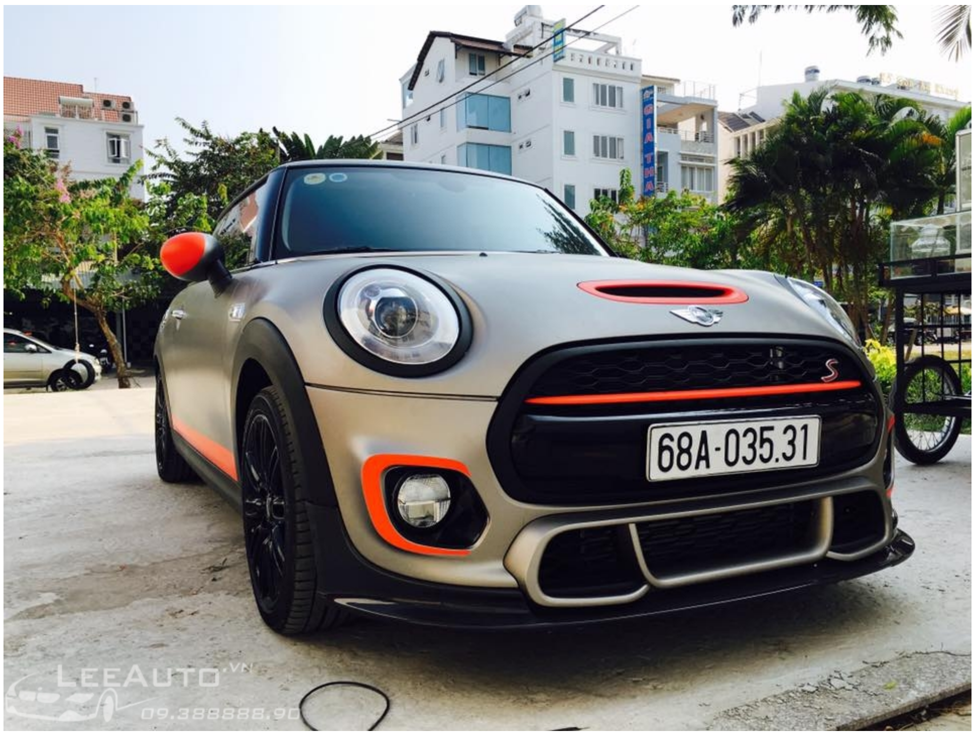


Figure 2: Original picture

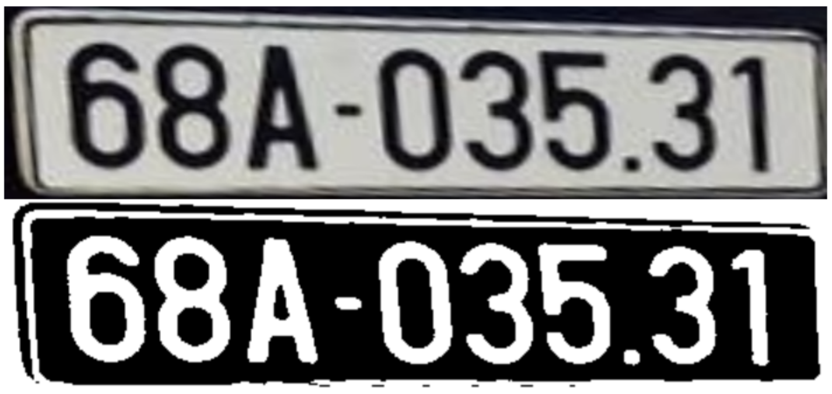


Figure 3: Extracted license plate after applying affine transformation

## 3 Character segmentation

In the next step, the characters will be segmented from the license plate which is expected to be separated from each other and precisely localised. After some transformations, our license plate characters appear as white on a black background with some noise blobs that need to determine whether it is a character or not. To tackle this problem, Connected-component analysis algorithm is introduced to figure separated character and filter the blobs based on their proportion of white pixels over the total.

### 3.1 Connected component labeling

In computer vision, connected components in a 2D binary image are clusters of pixels with the same value (0 or 255), which are connected to each other through either 4-pixel (4 faces), or 8-pixel (4 faces or 4 corners) connectivity. The algorithm connected component labeling (also known as connected component analysis, blob extraction, or region labeling) is an application of graph theory used to detect connected regions in binary image. And this algorithm can be implemented by OpenCV library or Depth-first search (DFS) algorithm.

#### 3.1.1 Using OpenCV library

In OpenCV library, there is a connected components function that will return the total number of unique components and a mask has the same dimensions as our input thresh image. For each pixel in the mask, it will associate with an integer ID value that indicates its appropriate connected component.

#### 3.1.2 Using non-recursive DFS

Let $\{M_{i,j}\}$ be the pixel matrix of the plate image. Define an undirectional graph $G = (V, E)$ such that $V = \{G_{i,j}\}$, $E$ is the set of edges satisfying that for each pair of neighboring pixels, there is an edge in $E$ that connect 2 vertices that correspond to such pixels. We make an observation that, for each of the possible character that can appear on a license plate (which are:

'0','1','A','B','C','D','E','F','G','H','K','L','2','M','N','P','R','S','T','U','V','X','Y','3','Z','4','5','6','7','8','9'), its visual appearance are connected, in other words each character can be drawn using a pen without lifting the pen. Therefore, we made an assumption that the pixelated form of each character must also be connected. As a result, each character in the license plate is represented by a connected component in $G$.

At this point, although it is possible to analyse the pixel pattern for each of the 31 possible characters, there would be too many cases to handle with traditional conditional statements. Therefore, we extracted a rectangular bounding box that contain the character, then we will use a pre-trained convolutional neural network for automatic detection.

Below is the implementation of the `get_bounding_boxes` function, which is based on the idea of connected component. When encounter a white pixel, we perform depth first search (DFS) to visit all neighboring white pixels. We chose to implement non-recursive DFS over traditional DFS to enhance performance since recursion is slow. While spreading to visit all white pixels in a component, we keep track of the boundaries of the component to save as the bounding box.

```python
def get_bounding_boxes(img, value = 255, lower_bound = 1/80, ↩
    upper_bound = 1/10):
    size = img.shape
    visited = np.zeros(size, dtype=np.bool_)
    i = 0
    boxes = []
    lower = int(size[0] * size[1] * lower_bound)
    upper = int(size[0] * size[1] * upper_bound)
    while i < size[0]:
        j = 0
        while j < size[1]:
            if img[i][j] == value and not visited[i][j]:
                qi = [i]
                qj = [j]
                visited[i][j] = True
                ihigh = i
                ilow = i
                jhigh = j
                jlow = j
                while len(qi) > 0:
                    fronti = qi.pop()
                    frontj = qj.pop()
                    ihigh = max(ihigh, fronti)
                    ilow = min(ilow, fronti)
                    jhigh = max(jhigh, frontj)
                    jlow = min(jlow, frontj)
                    for dx, dy in [(-1, 0), (1, 0), (0, 1), (0, -1)]:
                        nexti = fronti + dx
                        nextj = frontj + dy
                        if 0 <= nexti < size[0] and 0 <= nextj < size[1]:
                            if not visited[nexti][nextj] and img[nexti][nextj] == value:
                                visited[nexti][nextj] = True
                                qi.append(nexti)
                                qj.append(nextj)
                width = jhigh - jlow + 1
                height = ihigh - ilow + 1
                area = width * height
                if lower <= area <= upper and 6 <= width and 10 <= height:
                    boxes.append(((ilow, jlow),(ihigh, jhigh)))
            j += 1
        i += 1
    boxes = sorted(boxes, key = functools.cmp_to_key( lambda x, y: x[0][1] - y[0][1] if abs((x[1][0] - x[0][0]) - (y[1][0] - y[0][0])) < 20 else x[0][0] - y[0][0] ))
    return boxes
```

When the list of separated components are ready, we need to filter the real characters and dispose the noise blobs. In order to do this, we need to calculate a lower-bound and upper-bound portion that a character should take in order to be considered as valid characters (by experiments, we think that a valid character should take from 1.25% to 10% area of the plate. Those components that have the number of pixels lie within the allowed range will be stored into a list. The final step of this phase is finding contours and drawing a bounding box around each character, then this components list will be enter the next stage: character recognition.

We try sorting the bounding boxes in accordance with their natural positions - from top to bottom, then from left to right. To be specific, if the 2 boxes are roughly in the same horizontal line (less than 20 pixels apart from each other), we assume that they are in the same line and sort them by their horizontal coordinates. Otherwise, the boxes will lie on different rows, so we will sort them by their vertical coordinates.

## 3.2 Demo

Now, we will process the retrieved license plate from the previous stage. After applying the component connected algorithm, we will receive distinct characters like in the Figure 4.

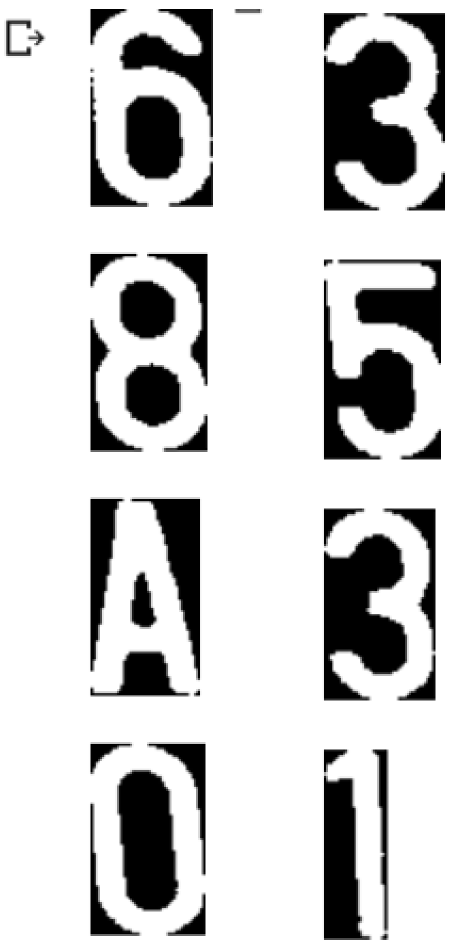


Figure 4: Separate characters after appplying connected component algorithm

## 4 Character recognition

When the characters are properly subdivided, the final step is apply the character recognition algorithm for each segmented character image. The output of character recognition process is the character of the given picture. By combining all these recognized characters, we will receive the raw text of our license plate.

### 4.1 Multi-layered convolutional neural network

To recognize each segmented character, we will use a two-layered convolutional neural network: the first layer looks for primitive patterns, such as horizontal or vertical lines and the second one looks for primitive shapes. To be more specific, there are ten 5x5 filters in the first layer with the relu activation and the pooling layer to summarize the presence of important features in the input and reduce the dimension of the image. While the spatial dimensions decreases, the number of filters in the second layer increases to 20 which is called pyramid architecture. After convolution, we flatten 20x4x4 tensor into one vector of size 320, and then add LogSoftmax as an output activation function to calculate probability for 31 classes of characters and numbers.

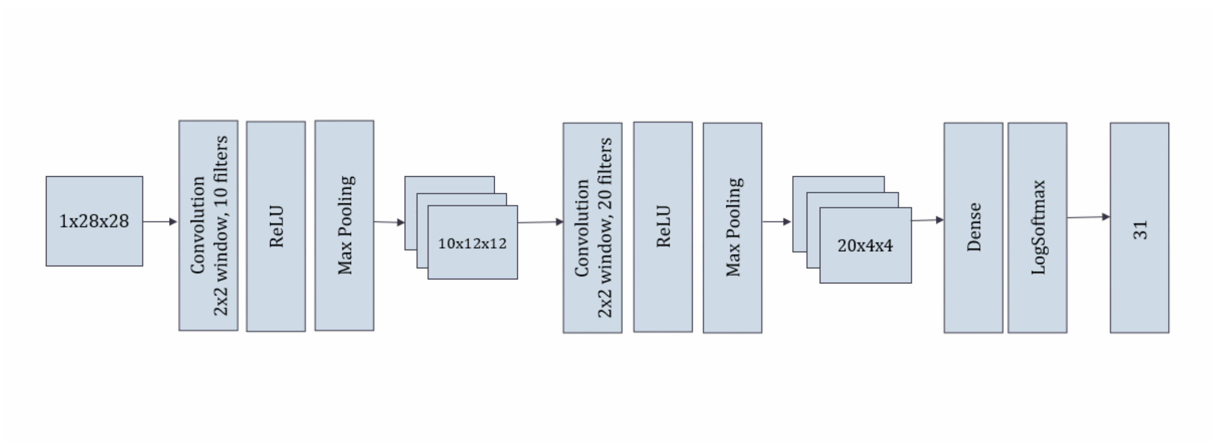
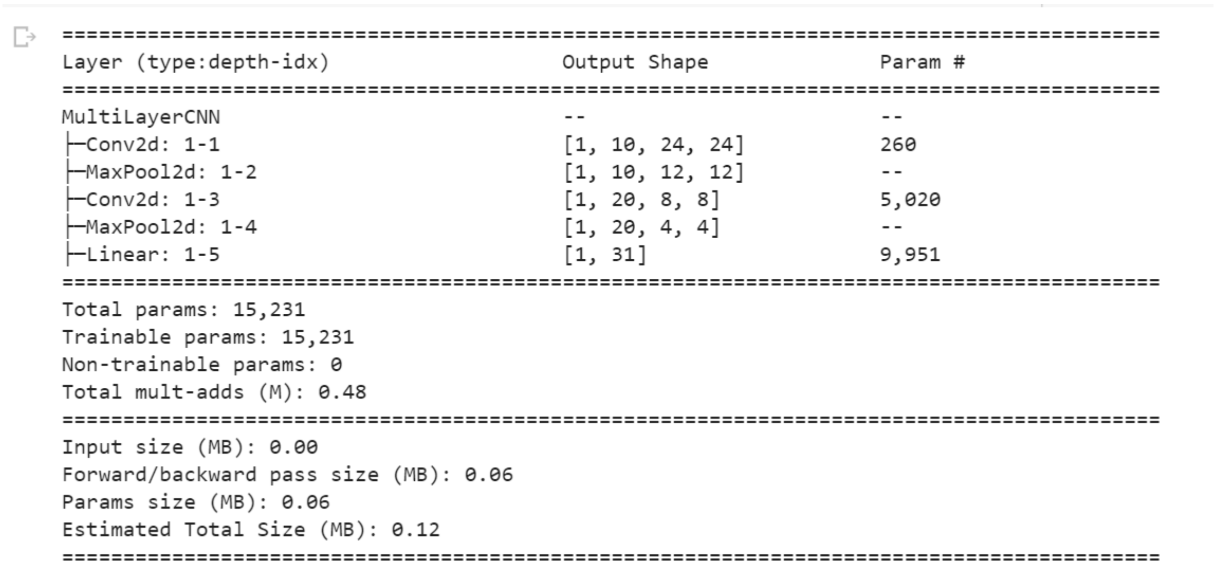

1x28x28
Convolution 2x2 window, 10 filters
ReLU
Max Pooling
10x12x12
Convolution 2x2 window, 20 filters
ReLU
Max Pooling
20x4x4
Dense
LogSoftmax
31

Figure 5: Convolutional neural network architecture



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MultiLayerCNN                            --                        --
├─Conv2d: 1-1                            [1, 10, 24, 24]           260
├─MaxPool2d: 1-2                         [1, 10, 12, 12]           --
├─Conv2d: 1-3                            [1, 20, 8, 8]             5,020
├─MaxPool2d: 1-4                         [1, 20, 4, 4]             --
├─Linear: 1-5                            [1, 31]                   9,951
==========================================================================================
Total params: 15,231
Trainable params: 15,231
Non-trainable params: 0
Total mult-adds (M): 0.48
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.06
Estimated Total Size (MB): 0.12
==========================================================================================
```

Figure 6: A detailed layer-by-layer structure of the network

## 4.2 Implementation

For this part, we have used PyTorch library to implement neural network and train our model. Before the training process, we need to initialize some parameters for the training function:

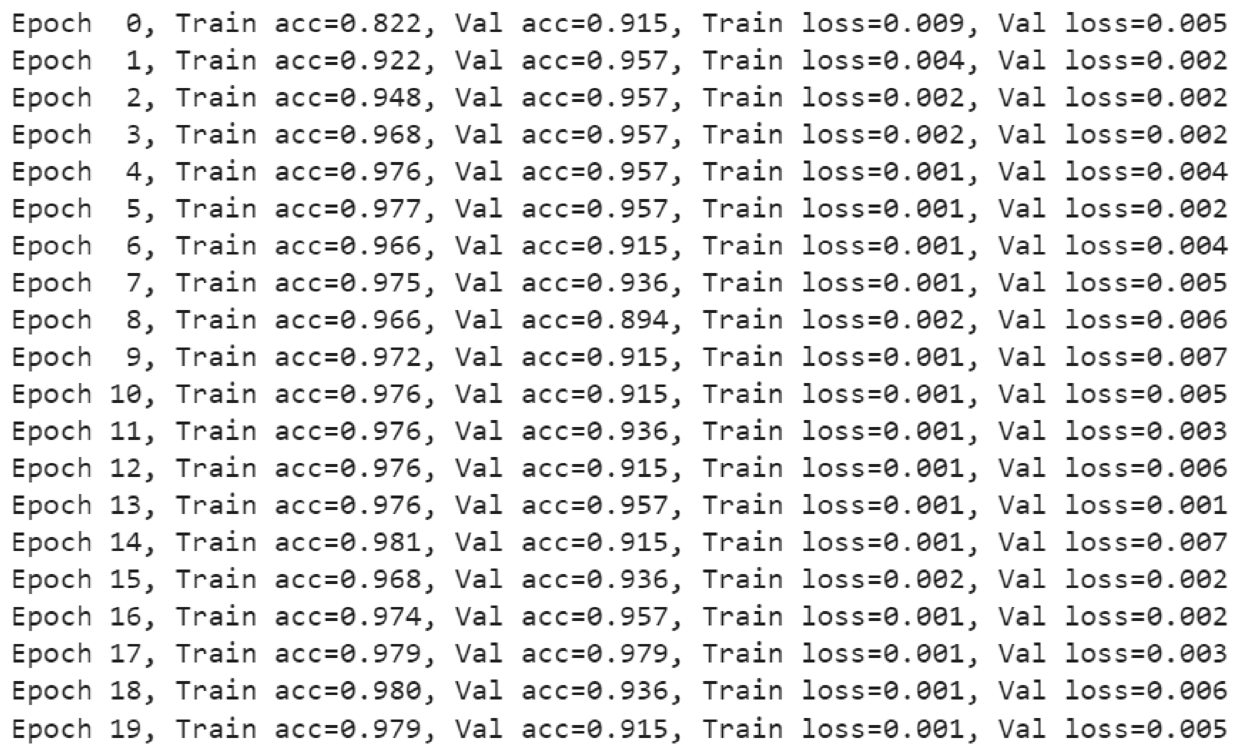
- A neural network
- Dataloader: because we will train this model through different batch, the Dataloader will defines the current training data.
- Learning rate: the learning speed of this network. We need to choose this parameter carefully because if the learning rate is too high, our algorithm will diverge to infinite error; otherwise, it will take our model a very long time to learn.

- Optimizer: Adam algorithm (Adaptive Moment Estimation) is used to reduce the losses and to provide the most accurate results possible.
- Loss Function: NLLLoss (Negative log likelihood loss function) will calculate the loss - discrepancy between predicted and expected result.

During the training process, we will go through each batch in the dataset to do the following tasks:

- Compute the difference between the prediction and the target.
- Minimize the difference by using the Optimizer to fine-tune the weights of the network.
- Calculate the accuracy by dividing the number of correct cases to the total.

After that, the training function will calculate and return average loss and training accuracy which indicates the improvement of our model. Besides, a validate function will be create to check how well the model performs on the dataset so that it can generalize to other data and avoid overfitting. Finally, we train the model for 20 epochs and monitor the training and validation accuracy.



```
Epoch  0, Train acc=0.822, Val acc=0.915, Train loss=0.009, Val loss=0.005
Epoch  1, Train acc=0.922, Val acc=0.957, Train loss=0.004, Val loss=0.002
Epoch  2, Train acc=0.948, Val acc=0.957, Train loss=0.002, Val loss=0.002
Epoch  3, Train acc=0.968, Val acc=0.957, Train loss=0.002, Val loss=0.002
Epoch  4, Train acc=0.976, Val acc=0.957, Train loss=0.001, Val loss=0.004
Epoch  5, Train acc=0.977, Val acc=0.957, Train loss=0.001, Val loss=0.002
Epoch  6, Train acc=0.966, Val acc=0.915, Train loss=0.001, Val loss=0.004
Epoch  7, Train acc=0.975, Val acc=0.936, Train loss=0.001, Val loss=0.005
Epoch  8, Train acc=0.966, Val acc=0.894, Train loss=0.002, Val loss=0.006
Epoch  9, Train acc=0.972, Val acc=0.915, Train loss=0.001, Val loss=0.007
Epoch 10, Train acc=0.976, Val acc=0.915, Train loss=0.001, Val loss=0.005
Epoch 11, Train acc=0.976, Val acc=0.936, Train loss=0.001, Val loss=0.003
Epoch 12, Train acc=0.976, Val acc=0.915, Train loss=0.001, Val loss=0.006
Epoch 13, Train acc=0.976, Val acc=0.957, Train loss=0.001, Val loss=0.001
Epoch 14, Train acc=0.981, Val acc=0.915, Train loss=0.001, Val loss=0.007
Epoch 15, Train acc=0.968, Val acc=0.936, Train loss=0.002, Val loss=0.002
Epoch 16, Train acc=0.974, Val acc=0.957, Train loss=0.001, Val loss=0.002
Epoch 17, Train acc=0.979, Val acc=0.979, Train loss=0.001, Val loss=0.003
Epoch 18, Train acc=0.980, Val acc=0.936, Train loss=0.001, Val loss=0.006
Epoch 19, Train acc=0.979, Val acc=0.915, Train loss=0.001, Val loss=0.005
```

Figure 7: Training and validate accuracy, training and validate loss

After the training process, we will save the model to recognize the segmented characters. For each selected character in the previous phase, we will use this model to calculate the probability of 31 classes and choose the one with the highest probability. Simultaneously, a string is created to store the recognized characters, which are also our result.

### 4.3 Demo

In this phase, we will use the model to recognize distinct characters in our set. After going through the model, we will collect and incorporate those identified characters to get the final result (Figure 9).

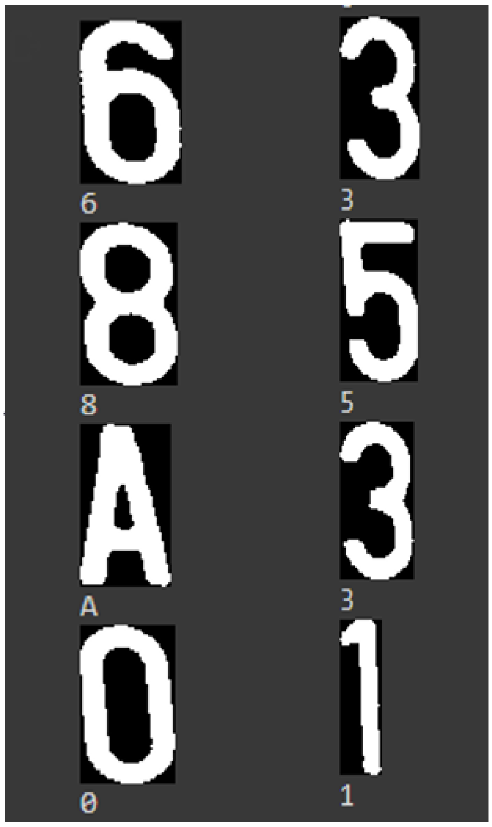
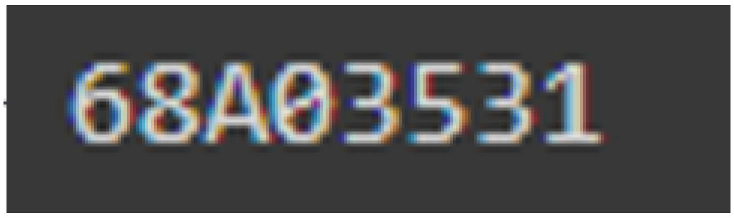


Figure 8: Recognize each character in the component list



```
68A03531
```

Figure 9: Final result

## 5 Web Interface

### 5.1 Framework

Flask is a micro web framework written in Python. It is called "micro" because it aims to keep the core simple but extensible. For instance, it does not have some pre-installed functions that are found commonly on other web frameworks, such as form validation, database abstraction layer, authentication, etc. Flask is popular because of its minimalism, and is used by many well-known websites such as Pinterest and LinkedIn.

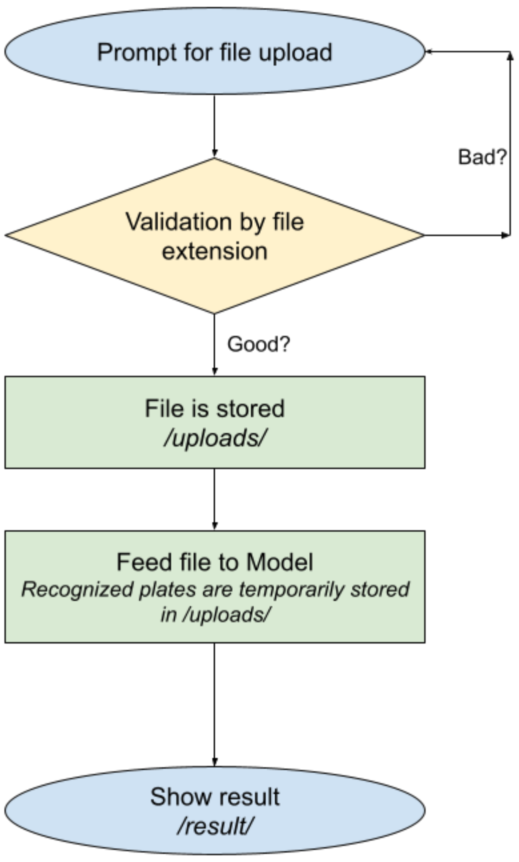
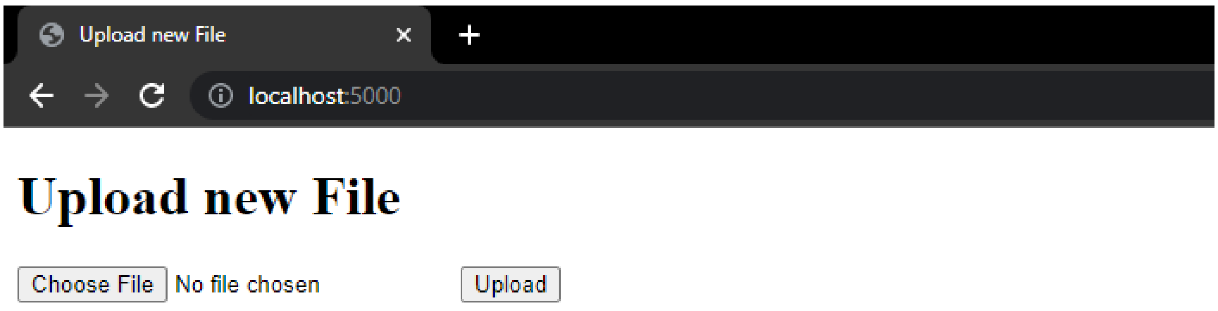


Figure 10: The data flow of the Web application

## 5.2 Data flow

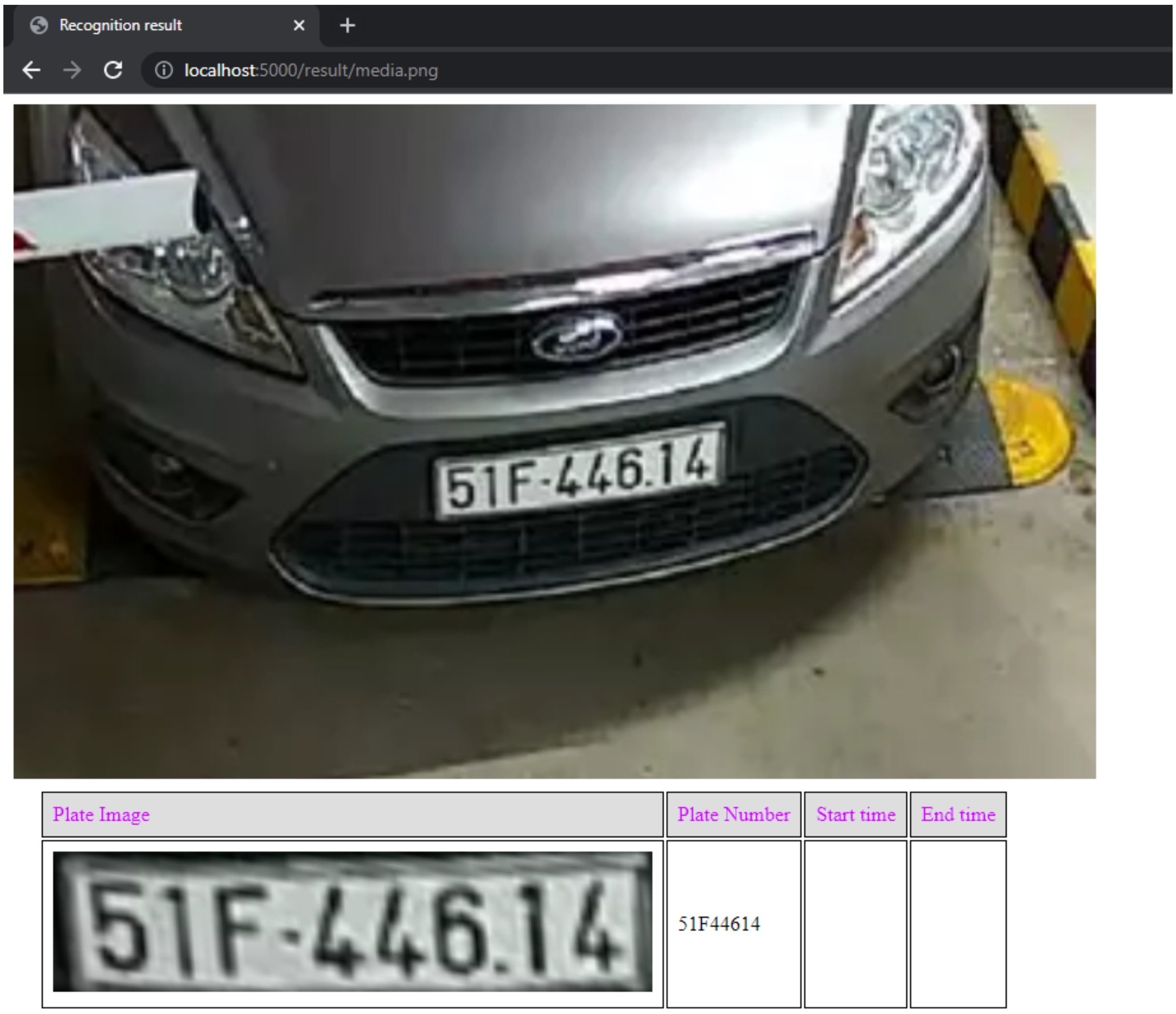
First, users upload their image / video via the upload interface. The media file will then be validated - to check whether it is an actual media file, if it is, it will be stored in the server's `/uploads/` folder and the model will be executed. When the model finishes its calculation, the user will be directed to the `/results/` URL to see the result presented in a tabular form.

### 5.3 Demo



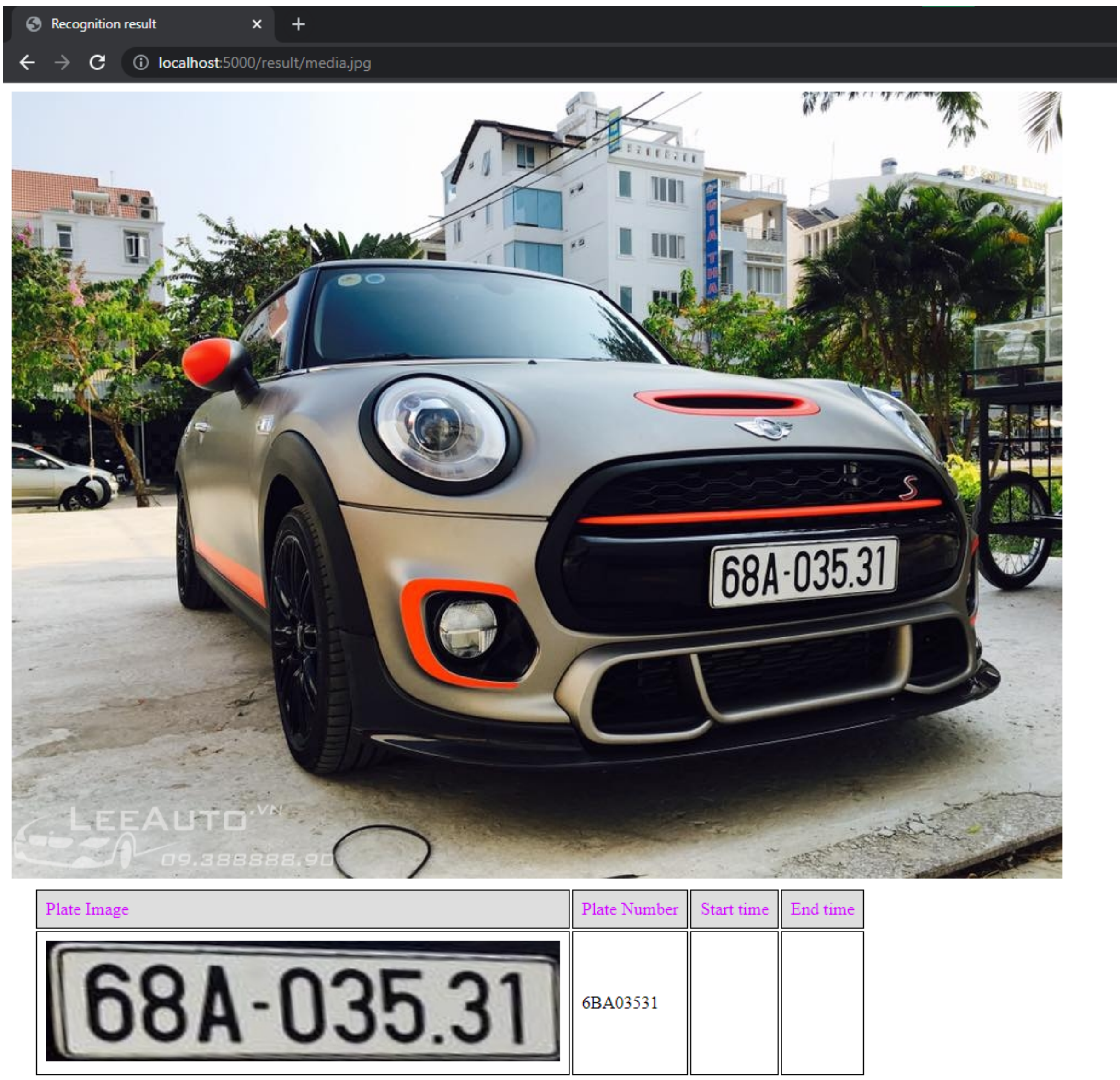
Upload new File

Figure 11: A minimalist Upload interface



| Plate Image | Plate Number | Start time | End time |
|---|---|---|---|
| 51F-446.14 | 51F44614 | | |

Figure 12: Output demo



| Plate Image | Plate Number | Start time | End time |
|---|---|---|---|
| 68A-035.31 | 6BA03531 | | |

Figure 13: Output demo 2

# 6 Conclusion

## 6.1 Evaluation

Overall, this license-plate recognition have some pros and cons in different circumstances.

### 6.1.1 Advantages

- WPOD-NET is a powerful model to detect, transform and straighten cockeyed plate to frontal view.
- The convolutional neural network can automatically and efficiently filter the important features of the character without any supervision.

### 6.1.2 Disadvantages

- In the awful light condition, our extracted license plate can be blur when it is converted to binary image.
- The heuristic parameters need to be adjusted to fix different kinds of license plates.
- There are some misclassification between B-8, Z-2,...

## 6.2 Future development

This model can be improved by adding more layers in the convolutional neural network so that it can recognize those confusing pairs better. Hyper-parameters can further be tuned for better result. Moreover, some filter can be applied to adjust the light condition like minimizing the dazzling bright.

## References

[1] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. In *2018 European Conference on Computer Vision (ECCV)*, pages 580–596, Sep 2018.