Final Project Report
Machine Learning

Bill ON
Department of Information Engineering
and Computer Science
411085099

Clémence MILLET
Department of Information Engineering
and Computer Science
711183498

Table Of Contents :

I. Introduction

In 2022, 200,000 traffic accidents were registered in Taiwan. We would like to find a way to reduce this number. A solution to that is to use automatic license plate recognition. Why would it be effective ? First it will enforce traffic regulation by enhancing surveillance, for instance, red-light violations, parking violations, illegal turns and other traffic infractions. It can also be used to monitor uninsured vehicles, locate wanted cars and overall help with any investigation. It can also be easily implemented since license plate recognition is already used for speed excess control. In these ways, authorities can enhance their ability to enforce traffic regulations, identify high-risk situations, and respond promptly to incidents, ultimately contributing to a reduction in accidents.
The goal of this project is to automatically read the license plate number of a provided car picture.

```
image_path = list(df['filepath'].apply(getFilename))
file_path = image_path[87]
img = cv2.imread(file_path) #read the image
img = io.imread(file_path) #Read the image
fig = px.imshow(img)
fig.update_layout(width=600, height=500, margin=dict(l=10, r=10, b=10, t=10),xaxis_title='Cars333.png with bounding box')
fig.add_shape(type='rect',x0=df['xmin'][87], x1=df['xmax'][87], y0=df['ymin'][87], y1=df['ymax'][87], xref='x', yref='y',line_color='cyan')
```

Example of a Car picture & its License Plate Bounding box

Display of a picture from the Dataset with its associated Bounding Box

II. Methodology

A. Dataset

https://github.com/Asikpalysik/Automatic-License-Plate-Detection/tree/main/images
This Dataset contains over 200 annotated car pictures with their license plates displayed at various angles.

B. Technologies used

Google Colab : an online platform specialized for Artificial Intelligence projects. It allows us to efficiently train and run our models by saving space through Google Drive storage, and to work faster because Google's machines have better performances than most computers.

Kaggle : the most popular platform for datasets and images. There is a special Kaggle plugin to directly import a Dataset to a Google Colab project without downloading it on your machine.

Tensorflow/Keras : famous Python library for Image Processing projects. It is all the more practical since Google Colabs run on Jupyter Notebooks, hence the obligation to use Python and its libraries.

OpenCV & Pandas : these are text processing libraries. The bounding boxes coordinates are stored in XML files. These libraries are used to dynamically retrieve the coordinates and the filepath of the picture they're linked to.

PyTesseract : This Python library grants us the ability to directly read text from a picture.

C. Model used

We build a model from the InceptionResNetV2 pre-trained Convolutional Neural Network (CNN) which is 164 layers deep. It is really rich and effective for image classification which is what we need here for our license plate recognition. This network is a combination of the Inception structure and the Residual connection.

```
inception_resnet = InceptionResNetV2(weights="imagenet",include_top=False, input_tensor=Input(shape=(224,224,3)))
# ---------------------
headmodel = inception_resnet.output
headmodel = Flatten()(headmodel)
headmodel = Dense(500,activation="relu")(headmodel)
headmodel = Dense(250,activation="relu")(headmodel)
headmodel = Dense(4,activation='sigmoid')(headmodel)


# ---------- model
model = Model(inputs=inception_resnet.input,outputs=headmodel)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219055592/219055592 [==============================] - 1s 0us/step
```

construct the model

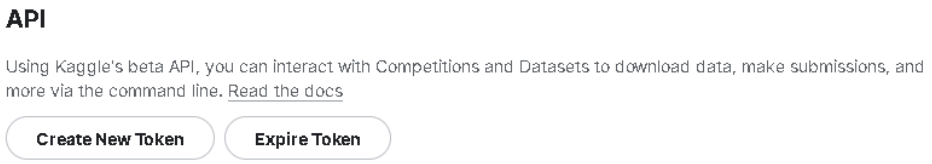
Model Layers

We tried some activation methods such as Sigmoidand Leaky RELU but they didn't offer as much performance as the RELU function in this case. We limited ourselves to these layers in order to avoid overfitting the model.

III. Programming

A. Dataset Importation

Since we're using Goolge Colab, we can take advantage of the dedicated Kaggle plugin to directly import the dataset. To do so, we must first download the kaggle.json file from Kaggle's website : https://www.kaggle.com/settings



Kaggle Profile Page API section

Select the "Create New Token" option to generate the file containing your Kaggle credentials. This file is mandatory to store the Dataset. What we meant earlier by "directly import the dataset" is that the Jupyter Notebook will download the Dataset to your Google Drive as a compressed archive. It will then ask to be granted access to your Google Drive in order to store locally the downloaded files.

B. Preprocessing

The images are resized to 224 x 224 pixels which is the standard compatible size of the pre-trained transfer learning model. After that, both images and labels are normalized and converted into numpy arrays. Then we split our data into a training and testing set using the train_test_split function from the sklearn package and with a training size of 80%.

```
[ ]  #Targeting all our values in array selecting all columns
     labels = df.iloc[:,1:].values
     data = []
     output = []
     for ind in range(len(image_path)):
         image = image_path[ind]
         img_arr = cv2.imread(image)
         h,w,d = img_arr.shape
         # Prepprocesing
         load_image = load_img(image,target_size=(224,224))
         load_image_arr = img_to_array(load_image)
         norm_load_image_arr = load_image_arr/255.0 # Normalization
         # Normalization to labels
         xmin,xmax,ymin,ymax = labels[ind]
         nxmin,nxmax = xmin/w,xmax/w
         nymin,nymax = ymin/h,ymax/h
         label_norm = (nxmin,nxmax,nymin,nymax) # Normalized output
         # Append
         data.append(norm_load_image_arr)
         output.append(label_norm)
```

Iterate through each picture to associate them with their bounding boxes coordinates

```
[ ]  X = np.array(data,dtype=np.float32)
     y = np.array(output,dtype=np.float32)
```

Convert data to array

```
[ ]  x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=0)
     x_train.shape,x_test.shape,y_train.shape,y_test.shape

     ((346, 224, 224, 3), (87, 224, 224, 3), (346, 4), (87, 4))
```

Split the data into training and testing set using sklearn.

Image Preprocessing

C. Training

We built a model from the InceptionResNetV2 pre-trained Convolutional Neural Network (CNN) which is 164 layers deep. It is really rich and effective for image classification which is what we need here for our license plate recognition. This network is a combination of the Inception structure and the Residual connection. Indeed the summary is very long.
After that we initialize a TensorBoard of object detection for the callbacks of our model fitting so we can visualize the training performance of our model. We train this model on 80 epochs and with a batch size of 10. The training took …s on Google collab using the T4 GPU. Don't forget to save the model in order to be able to load it later without having to go through all the training process again.

```
tfb = TensorBoard('object_detection')
history = model.fit(x=x_train,y=y_train,batch_size=10,epochs=80,
                    validation_data=(x_test,y_test),callbacks=[tfb])
```

```
Epoch 1/80
35/35 [==============================] - 117s 517ms/step - loss: 0.0394 - val_loss: 0.0279
Epoch 2/80
35/35 [==============================] - 8s 226ms/step - loss: 0.0109 - val_loss: 0.0210
Epoch 3/80
35/35 [==============================] - 8s 228ms/step - loss: 0.0069 - val_loss: 0.0177
Epoch 4/80
35/35 [==============================] - 8s 228ms/step - loss: 0.0054 - val_loss: 0.0169
Epoch 5/80
35/35 [==============================] - 8s 225ms/step - loss: 0.0045 - val_loss: 0.0209
Epoch 6/80
35/35 [==============================] - 8s 232ms/step - loss: 0.0041 - val_loss: 0.0253
Epoch 7/80
35/35 [==============================] - 8s 227ms/step - loss: 0.0040 - val_loss: 0.0202
Epoch 8/80
35/35 [==============================] - 8s 232ms/step - loss: 0.0032 - val_loss: 0.0155
Epoch 9/80
35/35 [==============================] - 8s 231ms/step - loss: 0.0027 - val_loss: 0.0131
Epoch 10/80
35/35 [==============================] - 8s 236ms/step - loss: 0.0026 - val_loss: 0.0138
```

Model training

D. Predicting

After loading the test images and making sure their shape is (1, 224, 224, 3), we can make our predictions.

Test picture

The output coordinates are normalized so we need to denormalize them first. Finally, we display the image with the bounding box from the predictions and extract the license plate with a textual output.

```
test_arr = image_arr_224.reshape(1,224,224,3)
coords = model.predict(test_arr)
denorm = np.array([w,w,h,h])
coords = coords * denorm
```

```
1/1 [==============================] - 5s 5s/step
```

Prediction

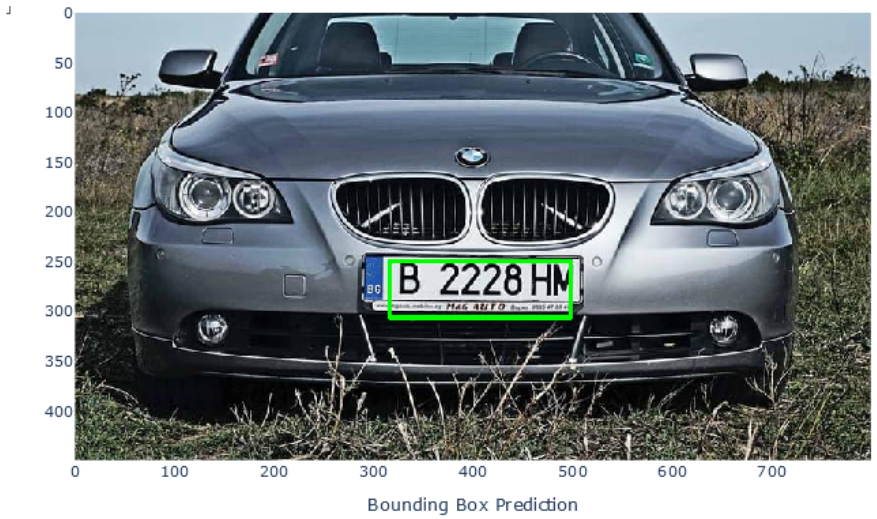
```
coords = coords.astype(np.int32)
xmin, xmax,ymin,ymax = coords[0]
pt1 =(xmin,ymin)
pt2 =(xmax,ymax)
print(pt1, pt2)
```

```
(235, 127) (431, 169)
```

```
cv2.rectangle(image,pt1,pt2,(0,255,0),3)
fig = px.imshow(image)
fig.update_layout(width=700, height=500, margin=dict(l=10, r=10, b=10, t=10),  xaxis_title="test picture")
```

Test picture after prediction

E. Results




```
] img = np.array(load_img(path))
  xmin ,xmax,ymin,ymax = cods[0]
  roi = img[ymin:ymax,xmin:xmax]
  fig = px.imshow(roi)
  fig.update_layout(width=(xmax-xmin), height=(ymax-ymin), margin=dict(l=10, r=10, b=10, t=10),xaxis_title='Cropped image')
```

ɪtraction of the license plates with the coordinates the model found

```
] # extract text from image
  text = pt.image_to_string(roi)
  print(text)
```

```
B 2228 HN
```

License Plate Reading

Upon feeding a random car picture to the model, it automatically highlights its Bounding Box. We then use these newfound coordinates to isolate the License Plate and finally we use PyTesseract to print its content.

F. Output verification

```
from sklearn.metrics import f1_score, precision_score, recall_score

# Make predictions
y_pred = model.predict(x_test)
```

```
3/3 [==============================] - 0s 183ms/step
```

```
# Convert predicted coordinates to actual bounding boxes
denorm = np.array([w, w, h, h])
y_pred_denorm = y_pred * denorm
y_pred_denorm = y_pred_denorm.astype(np.int32)
```

```
# Convert ground truth coordinates to actual bounding boxes
y_true_denorm = y_test * denorm
y_true_denorm = y_true_denorm.astype(np.int32)
```

```
# Flatten the arrays for f1_score calculation
y_true_flat = y_true_denorm.flatten()
y_pred_flat = y_pred_denorm.flatten()
```

```
# Apply a threshold (adjust as needed)
threshold = 0.5
y_pred_flat_binary = (y_pred_flat > threshold).astype(int)
```

```
# Calculate F1 score, precision, and recall
f1 = f1_score(y_true_flat, y_pred_flat_binary, average='micro')
precision = precision_score(y_true_flat, y_pred_flat_binary, average='micro')
recall = recall_score(y_true_flat, y_pred_flat_binary, average='micro')
```

```
print(f'F1 Score: {f1}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

```
F1 Score: 0.0
Precision: 0.0
Recall: 0.0
```

Performance Metrics

Since we already had the Loss function from the model training, we opted to continue measuring our model's abilities through other metrics.
We used the F1 Score to measure the amounts of False Positive compared to the total amount of Positive Results.
The Precision metric measures how well the model performs by comparing the model's prediction with known coordinates.
The Recall is the percentage of correctly identified elements.

IV. Sources

What is the F1 score ? :
https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/F1Score

How to calculate a model's F1 score :
https://stackoverflow.com/questions/70589698/tensorflow-compute-precision-recall-f1-score

How to retrieve an online image for Google Colab :
https://stackoverflow.com/questions/70659797/how-to-read-an-image-from-a-url-in-colab