

Kalyani Nikure

Munni Vidiyala

Ravali Bellamkonda

University of Missouri-Kansas City
Summer 2021, Python Deep Learning

Traffic Signs Prediction

August 1, 2021

Abstract

The purpose of this project is to implement a deep learning-based model for the classification of forty-three different types of traffic signals. We evaluated the classification accuracy and prediction speed of popular CNN-based architectures. An effective but simple Convolutional Neural Network (CNN) architecture is employed, which gives high accuracy and prediction speed. We further extended the model by deploying it on a web-based application to predict the classification based on images uploaded by the user.

Introduction

1. Problem Statement:

Automotive industry is one of the largest industries in the world. A great deal of research is being done in the artificial intelligence field and in vehicles that are autonomous and self-driving, including Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc. A total of 92 million motor vehicles were produced worldwide in 2019, and more than 2.5 million of those cars were made in the USA alone. Over 1400 autonomous vehicles are being tested in the US by 80 companies alone. Self-driving cars were forecast to be on the road in excess of 10 million by 2020. Although this prediction was quite exciting, it clearly did not come true. Several reasons explain this, but the most important reason is safety. For self-driving cars to operate in perfect conditions, they must identify every single detail on the road, including traffic signs, not just for the passengers, but for all road users. There has been talk of self-driving cars in which the passenger is fully dependent on the vehicle. The vehicles must know and obey all traffic rules, however, if they are to achieve automatic

driving at level 5. In order for this technology to be accurate, the vehicles need to be able to interpret traffic signs and take appropriate actions.

2. Project Proposal:

Among all the modeling networks for classifying images, the convolutional neural network (CNN) is one of the most powerful and popular ones. Our Python project example builds a deep neural network model to classify traffic signs into different categories using an image. Using this model, we are capable of reading and understanding traffic signs, which is very important for autonomous vehicles.

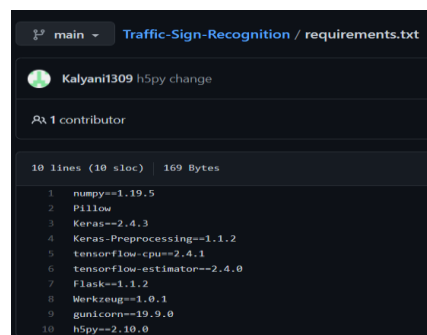
The Dataset

For this project, we are using the public dataset available at Kaggle: [Traffic Signs Dataset](#)

More than 50,000 images of traffic signs are included in the dataset. Furthermore, 43 different classes are divided into it. There are a variety of classes in the dataset, some with many images and others with few. Dataset size is approximately 300 MB. We have two folders in the dataset: a train folder containing images for each class and a test folder for testing our model.

Technical Specifications

This project requires prior knowledge of Tensorflow, Keras, Matplotlib, Scikit-learn, Heroku, Flask, Pandas, PIL and image classification. We have installed below necessary packages used for this Python data science project to deploy code on the Heroku web application which is specified in the requirement.txt file.



```
main Traffic-Sign-Recognition / requirements.txt
Kalyani1309 h5py change
1 contributor
10 lines (10 sloc) 169 Bytes
1 numpy==1.19.5
2 pillow
3 Keras==2.4.3
4 Keras-Preprocessing==1.1.2
5 tensorflow-cpu==2.4.1
6 tensorflow-estimator==2.4.0
7 Flask==1.1.2
8 Werkzeug==0.10.1
9 gunicorn==19.9.0
10 h5py==2.10.0
```

Steps to build the project

1. Explore the dataset

Our 'train' folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. With the help of the OS module, we iterate over all the classes and append images and their respective labels in the data and labels list. The PIL library is used to open image content into an array. Finally, we have stored all the images and their labels into lists (data and labels). We need to convert the list into numpy arrays for feeding to the model. The shape of data is (39209, 30, 30, 3) which means that there are 39,209 images of size 30×30 pixels and the last 3 means the data contains colored images (RGB value). With the sklearn package, we use the train_test_split() method to split training and testing data. From the keras.utils package, we use to_categorical method to convert the labels present in y_train and t_test into one-hot encoding.

```
print(data.shape, labels.shape)
(39209, 30, 30, 3) (39209,)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=0)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

Convert labels to onehot encoding

```
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

2. Build a CNN Model

To classify the images into their respective categories, we will build a CNN model. CNN is best for image classification purposes. The architecture of our model is:

- 2 Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- 2 Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- Flatten layer to squeeze the layers into 1 dimension

- Dense Fully connected layer (256 nodes, activation="relu")
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation="softmax")

As we have multiple classes to categorize, we compile the model using Adam's optimizer, which is performing well, with a loss of "categorical_crossentropy".

Build the model

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))

model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))

model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Dropout(rate=0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(rate=0.5))

# We have 43 classes that's why we have defined 43 in the dense
model.add(Dense(43, activation='softmax'))

# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

3. Train and validate the model

The model is then trained by using model.fit() after the model architecture has been constructed. The accuracy was stable after 20 epochs with a batch size of 64.

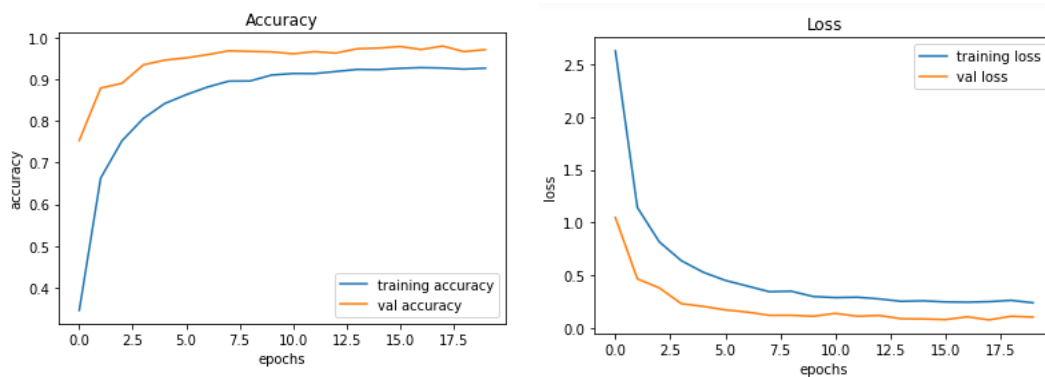
```

epochs = 20
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))

Epoch 1/20
491/491 [=====] - 89s 178ms/step - loss: 4.1388 - accuracy: 0.1861 - val_loss: 1.0454 - val_accuracy:
0.7534
Epoch 2/20
491/491 [=====] - 77s 157ms/step - loss: 1.2668 - accuracy: 0.6255 - val_loss: 0.4638 - val_accuracy:
0.8792
Epoch 3/20
491/491 [=====] - 77s 158ms/step - loss: 0.8696 - accuracy: 0.7360 - val_loss: 0.3779 - val_accuracy:
0.8907
Epoch 4/20
491/491 [=====] - 80s 163ms/step - loss: 0.6722 - accuracy: 0.7948 - val_loss: 0.2283 - val_accuracy:
0.9351
Epoch 5/20
491/491 [=====] - 78s 159ms/step - loss: 0.5613 - accuracy: 0.8304 - val_loss: 0.2021 - val_accuracy:
0.9464
Epoch 6/20
491/491 [=====] - 76s 155ms/step - loss: 0.4781 - accuracy: 0.8546 - val_loss: 0.1692 - val_accuracy:
0.9518
Epoch 7/20
491/491 [=====] - 74s 151ms/step - loss: 0.4087 - accuracy: 0.8775 - val_loss: 0.1485 - val_accuracy:
0.9596
Epoch 8/20
491/491 [=====] - 78s 159ms/step - loss: 0.3515 - accuracy: 0.8916 - val_loss: 0.1178 - val_accuracy:
0.9688
Epoch 9/20
491/491 [=====] - 75s 153ms/step - loss: 0.3540 - accuracy: 0.8940 - val_loss: 0.1179 - val_accuracy:
0.9674
Epoch 10/20
491/491 [=====] - 77s 156ms/step - loss: 0.3126 - accuracy: 0.9068 - val_loss: 0.1091 - val_accuracy:
0.9662
Epoch 11/20
491/491 [=====] - 77s 156ms/step - loss: 0.2751 - accuracy: 0.9154 - val_loss: 0.1360 - val_accuracy:
0.9616
Epoch 12/20
491/491 [=====] - 78s 159ms/step - loss: 0.2859 - accuracy: 0.9124 - val_loss: 0.1094 - val_accuracy:
0.9668
Epoch 13/20
491/491 [=====] - 73s 149ms/step - loss: 0.2674 - accuracy: 0.9198 - val_loss: 0.1162 - val_accuracy:
0.9633
Epoch 14/20
491/491 [=====] - 75s 153ms/step - loss: 0.2562 - accuracy: 0.9216 - val_loss: 0.0851 - val_accuracy:
0.9737
Epoch 15/20
491/491 [=====] - 73s 148ms/step - loss: 0.2554 - accuracy: 0.9235 - val_loss: 0.0833 - val_accuracy:
0.9753

```

On the training dataset, our model had an accuracy of 94%. We plot the graph for accuracy and loss with matplotlib.



4. Test the model with the test dataset

We have details regarding the image path and the corresponding class label in a test.csv file in our dataset. Pandas was used to extract image labels and path information. Our images need to be resized to 30*30 pixels, and we need a numpy array containing our images' data. Our model predicted actual labels based on exporting accuracy_score from the sklearn.metrics.

Our model achieved a ~94% accuracy rate. Our final step will be to save the trained model with the Keras model.save() function in .h5 format.

Accuracy with the test data

```
from sklearn.metrics import accuracy_score
print(accuracy_score(label, Y_pred))

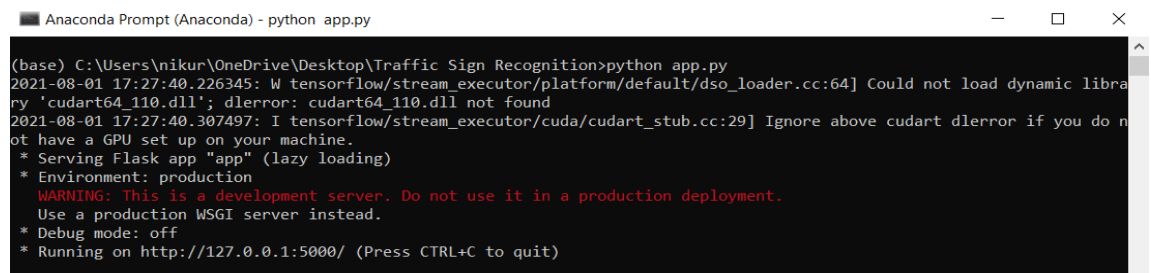
0.9391132224861442
```

Save the model

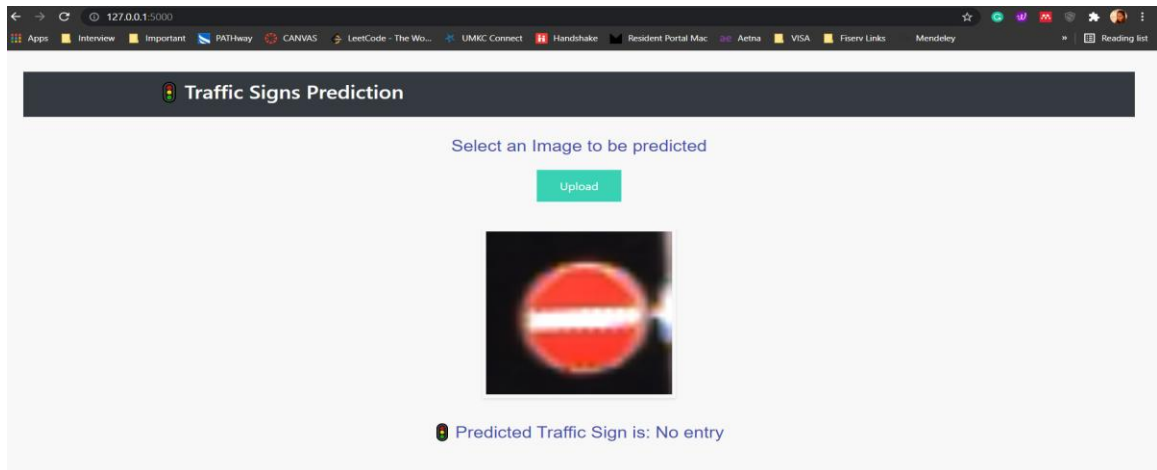
```
model.save("./training/TrafficSignPrediction.h5")
```

5. Debugging and deploying on local system with Flask

We would like to test and deploy our final code using Flask on our local system before deploying it to Heroku. Using Flask, we can build web-applications with Python. We first created our templates containing web HTML files and a static folder which has css and javascripts to host the web application. App.py contains the request rendering of urls of GET and POST methods on the server. Once we are finished doing the changes, we can run our code locally using flask service. Instantiate Anaconda prompt to run the application. Below is the screenshot of local debugging -



```
Anaconda Prompt (Anaconda) - python app.py
(base) C:\Users\nikur\OneDrive\Desktop\Traffic Sign Recognition>python app.py
2021-08-01 17:27:40.226345: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not found
2021-08-01 17:27:40.307497: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



6. Application deployment on Heroku

Once we are debugged and tested our application code on the local system, we are ready to deploy on Heroku. The Heroku platform as a service (PaaS) is a container-based cloud platform. Modern apps are deployed, managed, and scaled using Heroku. We create a Procfile which is used to specify the app.py file and web to be used. First connect the github repository to your github repo and deploy the code from the branch using the Deploy tab. Below are the logs when our application is successfully deployed.

```

Personal > traffic-sign-predict
GitHub Kalyani1309/Traffic-Sign-Recognition

Overview Resources Deploy Metrics Activity Access Settings

Activity Feed > Build Log

-----> Building on the Heroku-20 stack
-----> Using buildpack: heroku/python
-----> Python app detected
-----> Using Python version specified in runtime.txt
! Python has released a security update! Please consider upgrading to python-3.8.11
  Learn More: https://devcenter.heroku.com/articles/python-runtimes
-----> No change in requirements detected, installing from cache
-----> Using cached install of python-3.8.9
-----> Installing pip 20.2.4, setuptools 47.1.1 and wheel 0.36.2
-----> Installing SQLite3
-----> Installing requirements with pip
-----> Discovering process types
Procfile declares types -> web
-----> Compressing...
Done: 268.3M
-----> Launching...
Released v11
https://traffic-sign-predict.herokuapp.com/ deployed to Heroku

```

In this Python project with source code, we have successfully classified the traffic signs classifier with 94% accuracy and also visualized how our accuracy and loss changes with time, which is pretty good from a simple CNN model. We have also created a web-based application where a user can upload a traffic sign image and our model successfully predicts the classifier for the uploaded image.

1. Dataset Link: <https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>
2. Simplest way to deploy a React App to Heroku-
<https://www.youtube.com/watch?v=dn4mmfbletg&t=1s>
3. Deploy Flask App to Heroku | ML App | Flask-
<https://www.youtube.com/watch?v=e775cwGri6w&t=606s>

4. How to Deploy Node App to Heroku from GitHub-
<https://www.youtube.com/watch?v=gPOa0LvIwHQ>
5. Github Link: <https://github.com/Kalyani1309/Traffic-Sign-Recognition>
6. Heroku App Link: <http://traffic-sign-predict.herokuapp.com/>