TRAFFIC SIGNS RECONITION

COMP 4613 X2

MACHINE LEARNING

DEEPESH BELANI

100145274

Introduction

Image classification is the most basic task in computer vision, and it is also the task of almost all benchmark models for comparison. As the name suggests, image classification is a pattern classification problem. Its goal is to divide different images into different categories to achieve the smallest classification error. It is mainly for giving a better understanding of the differences between the computer vision and human vision by training computer with the data. It teaches the computer to recognize images and label images by computations according to predefined categories. Based on this, the algorithm learns which class the test images belong to and can then predict the correct class of future image data inputs and can even evaluate how accurate the test predictions are.

Problem Definition

Image classification is used many areas, one of them being the automotive industry. The automotive industry is moving towards smart vehicles where the vehicles use AI to navigate through the environment. The vehicles AI needs to recognize many different aspects of its surrounding environment, one of them being traffic signs. the recognition of proper traffic signs is important because they tell the vehicle weather it must stop, wait, drive at a specific speed etc. this can be achieved with the help of a traffic sign recognition algorithm which uses CNNs.

Expected Success Criteria

CNNs are used for image-based Machine Learning Problems. We will be using CNN to examine and extract the features of the image.

Most problems of this type have proved to work better with CNNs. Expected training accuracy is around 90%. Expected testing accuracy is around 95%.

Overview of Approach

The data is examined. It is found that the dataset has already been sorted, which makes it easier to understand. The data is then prepped for training and testing. The results are recorded, and a graph will be created to show the results after training.

Data Description

The GTSRB Dataset was obtained via Kaggle.com. The dataset contains 3 folders – Meta, Train and Test. The train folder consists of 43 (0 to 42) subfolders which each contain images of different traffic signs. The images are varied, i.e., they are taken at different times during the day. This is done for all the different traffic signs. the total image count is 39210 images across 43 folders.

The test folder contains 12631 images within it. The images in the folder are used for testing. The traffic sign images are varied, i.e., they are also taken at different time during the day.

All images are of different sizes. They will be resized in the (30, 30, 3) format. This means that all images are 30 x 30 pixels in size and 3 means that they are RGB.

Data Preparation

The train folder contains 43 folders within it (0 to 42). We use the OS module to explore the dataset. We iterate over all the classes and append the images and their respective labels in the data and labels list

(done in the second code block, second for loop – try section in Exhibit 2).

The Pillow library (formally known as PIL, can still be used as PIL when coding) is used to open the image contents into an array.

All the images and their labels are stored into lists (data and labels)

The lists are then converted into numpy arrays so that it can be fed to the model.

Training and testing

We use the train_test_splti() method from the sklearn package to train and test the data. The to_categorical method from the keras.utils package is use to convert the labels present in the y_train and t_test into one-hot encoding.

A CNN model is created to classify the images into their respective categories.

After building the model, is was trained using model.fit(). The model was tested with 2 batch sizes – 32 and 64. The model preformed better with a batch size of 64. A stable accuracy was obtained after 15 epochs.

The dataset contains a test folder and a test.csv file which contains details related to the image path and their respective class labels. The image path and labels are extracted using pandas. Images are then resized into 30 x 30 pixels and a numpy array is created, containing all the image data. accuracy_score is imported from sklearn.metrics. We then observe how accurately the labels are being predicted.

The model was saved using keras model.save() function.

Machine Learning Algorithms

Algorithm 1 – CNN with Keras

A CNN was used to train and test the model. The dataset contains multiple classes (43). The CNN model was designed to run with an Adam Optimizer which is an algorithm that has very good performance and its losses are “categorical_crossentropy”.

Adam Optimizer is a special optimization algorithm for the stochastic gradient descent used for training deep learning models. It is very easy to configure. Its default configuration parameters do well on most problems. (default configuration is used here)

CNN is basically used for image classifications and identifying if an image is a bird, a plane, or a car etc. It scans images from left to right and top to bottom to pull out important features from the image and combines the features to classify images. It can handle the images that have been translated, rotated scaled and changes in perspective.

The use of CNNs is perfect for a problem like this as CNNs will yield the best possible results.

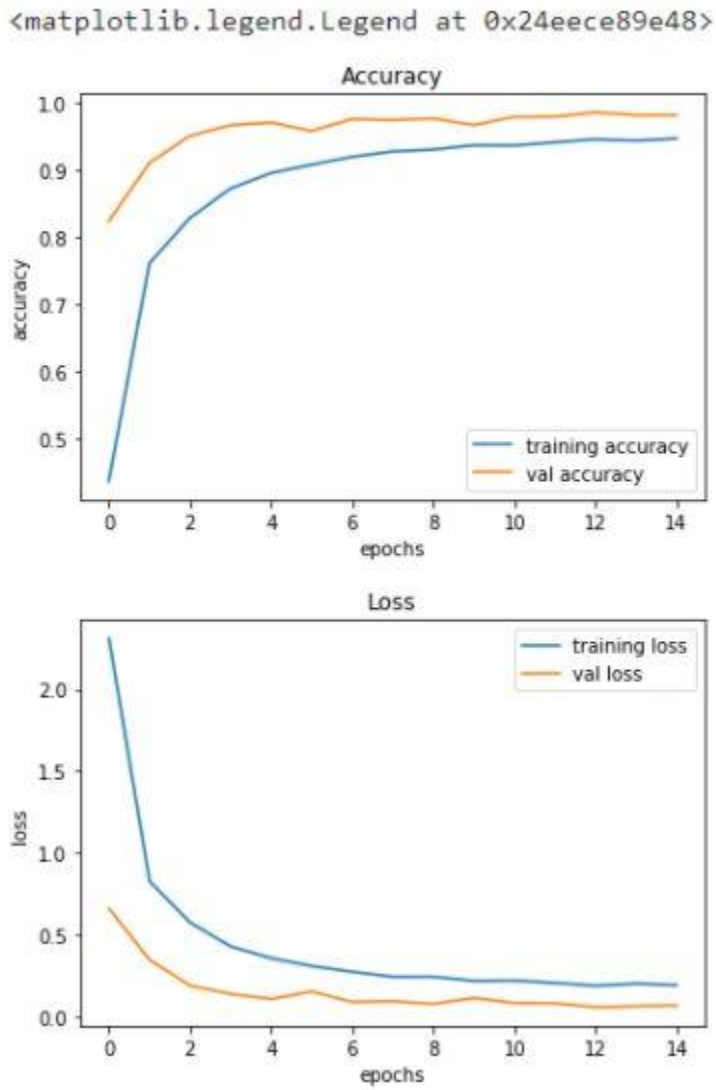
Evaluation

Graphs

2 graphs were created to show the accuracy and loss of the trained model.

The first graph shows the training accuracy in blue and validation accuracy in orange. As we can see from the graph, the training accuracy obtained was 95%. The validation accuracy was 98%.

The second graph shows us the loss over the course of 15 epochs. The training loss is roughly 2% whereas the validation loss is roughly 1 percent.



Comparison to Expected results

The results obtained in this project are in line with the expected results. The training accuracy obtained was 95% with a loss of roughly 2%. This is better than the expected training accuracy result which is 90%.

The testing accuracy obtained is 95%, which is in line with the expected testing accuracy of 95%.

Conclusion

Form the following results, the CNN is doing a good job in classifying the different types of traffic signs. The code is efficient and stable. It yields consistent results with multiple runs.

References

https://keras.io/api/optimizers/adam/#:~:text=Optimizer%20that%20implements%20the%20Adam,order%20and%20second%2Dorder%20moments.&text=The%20exponential%20decay%20rate%20for%20the%201st%20moment%20estimates.

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Exhibits (code)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
import tqdm
import warnings
```

```
data = []
labels = []
classes = 43
#category = ["Meta", "Test", "Train"]
#dir = r'C:/Users/deepe/Desktop/Acadia Y4/COMP 4613 - Machine Learning/archive'
#os.listdir()
#os.chdir(r"C:\Users\deepe\Desktop\Acadia Y4\COMP 4613 - Machine Learning")
cur_path = os.getcwd()

#for ct in category:
 # path = os.path.join(dir, ct, str(i))

#Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path, 'train', str(i))
    #os.path.exists(path)
    #try:
    #  images = os.listdir(path)
    #except FileNotFoundError:
    #  print("error")
      #continue
    images = os.listdir(path)

    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            #image = Image.convert('RGB')
            image = image.resize((30,30))
            image = np.array(image)
            #sim = Image.fromarray(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")

#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
```

```
print(data.shape, labels.shape)

#Splitting training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

```
#Building 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))
model.add(Dense(43, activation='softmax'))
#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
epochs = 15
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs,validation_data=(X_test, y_test))
```

```
Train on 31367 samples, validate on 7842 samples
Epoch 1/15
31367/31367 [==============================] - 82s 3ms/step - loss: 2.3108 - accuracy: 0.4369 - val_lo
ss: 0.6590 - val_accuracy: 0.8234
Epoch 2/15
31367/31367 [==============================] - 82s 3ms/step - loss: 0.8266 - accuracy: 0.7606 - val_lo
ss: 0.3468 - val_accuracy: 0.9100
Epoch 3/15
31367/31367 [==============================] - 83s 3ms/step - loss: 0.5738 - accuracy: 0.8283 - val_lo
ss: 0.1882 - val_accuracy: 0.9504
Epoch 4/15
31367/31367 [==============================] - 85s 3ms/step - loss: 0.4282 - accuracy: 0.8720 - val_lo
ss: 0.1373 - val_accuracy: 0.9661
Epoch 5/15
31367/31367 [==============================] - 84s 3ms/step - loss: 0.3565 - accuracy: 0.8950 - val_lo
ss: 0.1068 - val_accuracy: 0.9702
Epoch 6/15
31367/31367 [==============================] - 81s 3ms/step - loss: 0.3081 - accuracy: 0.9074 - val_lo
ss: 0.1527 - val_accuracy: 0.9575
Epoch 7/15
31367/31367 [==============================] - 81s 3ms/step - loss: 0.2730 - accuracy: 0.9192 - val_lo
ss: 0.0888 - val_accuracy: 0.9753
Epoch 8/15
31367/31367 [==============================] - 81s 3ms/step - loss: 0.2429 - accuracy: 0.9271 - val_lo
ss: 0.0934 - val_accuracy: 0.9737
Epoch 9/15
31367/31367 [==============================] - 84s 3ms/step - loss: 0.2429 - accuracy: 0.9299 - val_lo
ss: 0.0772 - val_accuracy: 0.9763
Epoch 10/15
31367/31367 [==============================] - 81s 3ms/step - loss: 0.2176 - accuracy: 0.9364 - val_lo
ss: 0.1133 - val_accuracy: 0.9663
Epoch 11/15
31367/31367 [==============================] - 82s 3ms/step - loss: 0.2200 - accuracy: 0.9360 - val_lo
ss: 0.0823 - val_accuracy: 0.9786
Epoch 12/15
31367/31367 [==============================] - 80s 3ms/step - loss: 0.2046 - accuracy: 0.9406 - val_lo
ss: 0.0806 - val_accuracy: 0.9787
Epoch 13/15
31367/31367 [==============================] - 80s 3ms/step - loss: 0.1876 - accuracy: 0.9452 - val_lo
ss: 0.0569 - val_accuracy: 0.9852
Epoch 14/15
31367/31367 [==============================] - 81s 3ms/step - loss: 0.2007 - accuracy: 0.9430 - val_lo
ss: 0.0629 - val_accuracy: 0.9811
Epoch 15/15
31367/31367 [==============================] - 81s 3ms/step - loss: 0.1914 - accuracy: 0.9463 - val_lo
ss: 0.0676 - val_accuracy: 0.9813
```

```
#plotting graphs for accuracy
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
plt.figure(1)

plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

```
#testing accuracy on test dataset
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('Test.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values
data=[]
for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
X_test=np.array(data)
pred = model.predict_classes(X_test)

#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
model.save('traffic_classifier.h5')
```