SUMMER INTERNSHIP REPORT

Area Of Online Internship	**AI/ML/DL**
Intern Name	**STANZIN NURBOO**
Name Of Institution	**INDIAN INSTITUTE OF TECHNOLOGY, INDORE**
Faculty Mentor Name	**Dr. Vimal Bhatia**
Duration	**2 MONTHS (23/06/2021 TO 23/08/2021)**
Date Of Submission	**17/09/2021**

Table Of Contents

INTRODUCTION

What is Traffic Sign Recognition?

Traffic-sign recognition (TSR) is a technology by which a vehicle is able to recognize the traffic sign put on the road e.g, "speed limit" or "children" or "turn ahead". This is part of the features collectively called ADAS. The technology is being developed by a variety of automotive suppliers. It uses image processing techniques to detect the traffic signs. The detection methods can be generally divided into color based, shape based and learning based methods.

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.

OBJECTIVE OF THIS PROJECT:

This project is made using the TENSORFLOW libraries and its high level API KERAS. The main objective of this project is to detect traffic sign and recognize it with higher accuracy.

ABOUT THIS PROJECT:

In this project, we will build a deep neural network model that can classify traffic signs present in the image into different categories. With this model, we are able to read and understand traffic signs which are a very important task for all autonomous vehicles.

DATASET:

The dataset we have used for this project is the GTSRB (German traffic sign recognition benchmark). It contains a Train folder that has traffic sign images in 43 different classes, a Test folder that has over 12,000 images for testing purposes. A test.csv file that contains the path of the test images along with their respective classes.

The GTSRB dataset consists of **43 traffic sign classes** and **nearly 50,000 images.**

BUILDING THE PROJECT:

Our approach to building this traffic sign classification model is discussed in four steps:

1. **Exploring the dataset**
2. **Building a CNN model**
3. **Train and validate the model**
4. **Test the model with test dataset**

1.Exploring 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.

In [3]:
```
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
os.chdir("D:\gtsrb")
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

```
Using TensorFlow backend.
```

In [4]:
```
data = []
labels = []
classes = 43
cur_path = os.getcwd()
```

In [5]:
```
#Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path,'train',str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
```

In [6]:
```
#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
```

Finally, we have stored all the images and their labels into lists (data and labels).

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.

In [7]:
```
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)
```

```
(39209, 30, 30, 3) (39209,)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

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).

2. Building a CNN model

To classify the images into their respective categories, we will build a CNN model (***Convolutional Neural Network***). 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")

We compile the model with Adam optimizer which performs well and loss is "categorical_crossentropy" because we have multiple classes to categorise.

In [8]:
```
#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'])
```

3. Train and validate the model

After building the model architecture, we then train the model using model.fit(). I tried with batch size 32 and 64. Our model performed better with 64 batch size. And after 15 epochs the accuracy was stable.

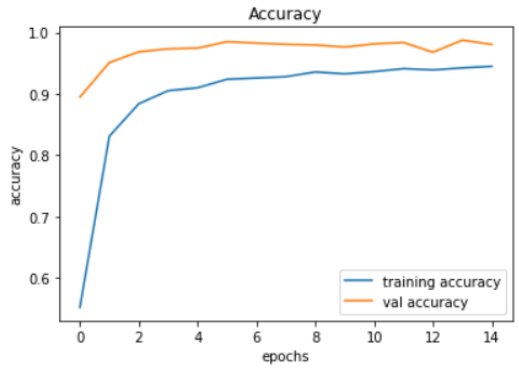
In [9]:
```
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
```

```
Epoch 1/15
981/981 [==============================] - 78s 79ms/step - loss: 1.7808 - accuracy: 0.5525 - val_loss: 0.4208 - val_accuracy:
0.8949
Epoch 2/15
981/981 [==============================] - 78s 80ms/step - loss: 0.5852 - accuracy: 0.8311 - val_loss: 0.2142 - val_accuracy:
0.9510
Epoch 3/15
981/981 [==============================] - 81s 83ms/step - loss: 0.4034 - accuracy: 0.8840 - val_loss: 0.1239 - val_accuracy:
0.9685
Epoch 4/15
981/981 [==============================] - 79s 80ms/step - loss: 0.3268 - accuracy: 0.9052 - val_loss: 0.1169 - val_accuracy:
0.9735
Epoch 5/15
981/981 [==============================] - 81s 82ms/step - loss: 0.3269 - accuracy: 0.9100 - val_loss: 0.0962 - val_accuracy:
0.9746
Epoch 6/15
981/981 [==============================] - 86s 88ms/step - loss: 0.2721 - accuracy: 0.9238 - val_loss: 0.0596 - val_accuracy:
0.9851
Epoch 7/15
981/981 [==============================] - 93s 94ms/step - loss: 0.2661 - accuracy: 0.9259 - val_loss: 0.0641 - val_accuracy:
0.9827
Epoch 8/15
981/981 [==============================] - 76s 77ms/step - loss: 0.2591 - accuracy: 0.9280 - val_loss: 0.0728 - val_accuracy:
0.9807
Epoch 9/15
981/981 [==============================] - 74s 76ms/step - loss: 0.2332 - accuracy: 0.9358 - val_loss: 0.0790 - val_accuracy:
0.9797
Epoch 10/15
981/981 [==============================] - 73s 75ms/step - loss: 0.2424 - accuracy: 0.9326 - val_loss: 0.0856 - val_accuracy:
0.9763
Epoch 11/15
981/981 [==============================] - 70s 72ms/step - loss: 0.2337 - accuracy: 0.9364 - val_loss: 0.0689 - val_accuracy:
0.9815
Epoch 12/15
981/981 [==============================] - 74s 76ms/step - loss: 0.2183 - accuracy: 0.9412 - val_loss: 0.0623 - val_accuracy:
0.9837
Epoch 13/15
981/981 [==============================] - 84s 85ms/step - loss: 0.2359 - accuracy: 0.9390 - val_loss: 0.1244 - val_accuracy:
0.9679
Epoch 14/15
981/981 [==============================] - 76s 77ms/step - loss: 0.2316 - accuracy: 0.9424 - val_loss: 0.0529 - val_accuracy:
0.9876
Epoch 15/15
981/981 [==============================] - 75s 77ms/step - loss: 0.2158 - accuracy: 0.9449 - val_loss: 0.0650 - val_accuracy:
0.9805
```

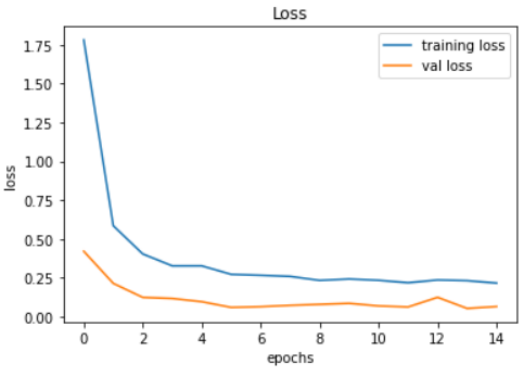
Our model got a 98.05% accuracy on the training dataset.

With matplotlib, we plot the graph for accuracy and the loss.

In [10]:
```
#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()
```



In [11]:
```
#Loss
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()
```



4. Test our model with test dataset

Our dataset contains a test folder and in a test.csv file, we have the details related to the image path and their respective class labels. We extract the image path and labels using pandas. Then to predict the model, we have to resize our images to 30×30 pixels and make a numpy array containing all image data. From the sklearn.metrics, we imported the accuracy_score and observed how our model predicted the actual labels. We achieved a 94.88% accuracy in this model.

In [12]:
```
def testing(testcsv):
    y_test = pd.read_csv(testcsv)
    label = 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)
    return X_test,label
X_test, label = testing('Test.csv')
Y_pred = model.predict_classes(X_test)
Y_pred
```

```
WARNING:tensorflow:From <ipython-input-12-4a067e9ae3a1>:13: Sequential.predict_classes (from tensorflow.python.keras.engine.seq
uential) is deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses
a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification
(e.g. if it uses a `sigmoid` last-layer activation).
```

Out[12]:
```
array([16,  1, 38, ..., 15,  7, 10], dtype=int64)
```

In [13]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(label, Y_pred))
```

```
0.9488519398258115
```

CONCLUSION:

We covered how deep learning can be used to classify traffic signs with high accuracy, employing a variety of pre-processing and regularization techniques (e.g. dropout), and trying different model architectures. We built highly configurable code and developed a flexible way of evaluating multiple architectures. Our model reached close to 98.05% accuracy on the test set, achieving 95% on the validation set.

REFERENCES:

https://www.slideshare.net/avijitrai/traffic-sign-detection

http://vision.soic.indiana.edu/b657/sp2016/projects/aditnaga/paper.pdf

https://en.wikipedia.org/wiki/Traffic-sign_recognition

https://ieeexplore.ieee.org/document/8997240

Traffic Signs Classification with a Convolutional Neural Network | by Raj Uppala | Medium