TRAFFIC SIGN DETECTION USING DEEP LEARNING
-Configuration Manual

MSc Research Project
Data Analytics

Sanika Khandalkar
Student ID: x19233655

School of Computing
National College of Ireland

Supervisor: Dr. Cristian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student ID:	X19266355
Programme:	MSc Data Analytics
Year	2021-2022
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date	15/08/2022
Project Title	TRAFFIC SIGN DETECTION USING DEEP LEARNING ALGORITHMS
Word Count	1022
Page Count	27

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …………SANIKA ATUL KHANDALKAR………

Date: ………15/08/2022…………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

TRAFFIC SIGN DETECTION USING DEEP LEARNING
-Configuration Manual

Sanika Khandalkar
X19233655

1. Introduction

The configuration handbook contains information on the research project's implementation phase, as well as the environment setup that was used and worked on. As a result, the research is being conducted on a traffic sign detection dataset using a deep learning Algorithmic model. The next sections include hardware and software specs, data sources, and implementation.

2. System Configuration

The system setup section offers thorough information on the hardware and software specs for the research project.

2.1. Hardware Configuration

This section includes the existing system configuration.

FEATURES	VERSION
Operating system	Windows 10 Home Single Language
Processor	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
RAM	8.00 GB
System Type	64-bit operating system, x64-based processor
Hard Drive	1TB

Table1: Hardware Configurations

2.2 Software Configuration

The research investigation was done out utilizing a single software application known as Google Collab. As a result, Google Collab's environment is Python 3.7.11. Few libraries are required for research purposes. As a result, the libraires utilized are as follows:

NumPy: 1.21.0

Keras: 2.9.0

Matplotlib: 3.5

Sklearn: 0.24

TensorFlow: 2.8.2

Seaborn: 0.11.2

3. Data Sources

The dataset for traffic sign detection comprises over 50,000 photos with 43 annotated classes. The dataset has a relatively low pixel resolution of around 32 × 32. Which represents the N number of traffic signs located on German roadways. As a consequence, four deep learning techniques, CNN, VGG16, VGG19, and Resnet 50, were used in this research study to improve accuracy. (Stallkamp, Schlipsing, Salmen and Igel, 2011) utilized this dataset for German traffic sign identification but used a different model.

4. Implementation

The first step of implementation is Mounting the google collab in the drive and followed by the unzip dataset folder and loading the dataset for the further implementation

```
[1] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

Figure 1 : Mount the drive from the google collab

```
!unzip /content/drive/MyDrive/traffic_sign/archive.zip -d /content/drive/MyDrive/traffic_sign/Data
```

Figure2: Import drive and Load the dataset after unzipping.

Now the next Image includes all the necessary libraries which will be used in the implementation. Where the tensor flow version has also been checked.

4.1 Data Pre-processing

The initial stage in data preparation is to import libraries, which is followed by partitioning the data into train test, improving traffic signal pictures, and the 43 classes.

```
[2] import os
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from PIL import Image
    import tensorflow as tf
    from tensorflow.keras.utils import to_categorical
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.models import Model
    from sklearn.metrics import accuracy_score
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.applications import Xception
    from keras.layers import Softmax, GlobalAvgPool2D
    from keras.preprocessing import image
    from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPool2D, Dropout
    from tensorflow.keras.applications.vgg16 import preprocess_input
    from tensorflow.keras.applications.vgg19 import preprocess_input
    from tensorflow.keras.preprocessing import image
    from sklearn.metrics import classification_report,confusion_matrix
    import seaborn as sns
    from tensorflow.keras.applications.vgg16 import VGG16
    from keras.applications.vgg19 import VGG19
```

Figure 3: Importing libraries

```
[3] import tensorflow as tf
    import keras
    print(keras.__version__)

    2.8.0
```

```
path = "/content/drive/MyDrive/traffic_sign/Data/Train/0/00000_00004_00029.png"
img = Image.open(path)
img = img.resize((32, 32))
sr = np.array(img)
plt.imshow(img)
plt.show()
```

Figure4: Path Directory

```
[7] dir = '/content/drive/MyDrive/traffic_sign/Data'

    plt.figure(figsize=(10, 10))
    for i in range (0,43):
        plt.subplot(7,7,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        path = dir + "/Meta/{0}.png".format(i)
        img = plt.imread(path)
        plt.imshow(img)
        plt.xlabel(i)
```

Figure5: Importing the classes

```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(data, labels, test_size= 0.1, random_state=10)

print("training_shape: ", x_train.shape,y_train.shape)
print("testing_shape: ", x_test.shape,y_test.shape)

y_train = tf.one_hot(y_train,43)
y_test = tf.one_hot(y_test,43)
```

```
training_shape:  (35288, 32, 32, 3) (35288,)
testing_shape:  (3921, 32, 32, 3) (3921,)
```

Figure 6: Splitting the data and one hot encoding

4.2 Model Implementation and evaluation

Deep learning methods are employed in this stage to recognize and detect traffic signs. CNN with Adam optimiser was the first model implemented. Restnet 50, VGG16, and VGG19 are all available. As a consequence, CNN with Adam optimiser produced the best results with high accuracy.

4.2.1 CNN Model with Adam Optimiser (Model 1)

We employed Adam optimiser and relu activation in convolutional neural network, and the accompanying snippets comprise the model building scripts and results.

```
model = tf.keras.Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu", input_shape= x_train.shape[1:]))
model.add(Conv2D(filters=64, kernel_size=(5,5), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))
model.add((Conv2D(filters=128,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(1024, activation="relu"))
model.add(Dropout(rate=0.40))
model.add(Dense(43, activation="softmax"))
```

Figure 7 : Model 1 Building

```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Figure 8: compiling model

```
epochs = 10
history = model.fit(x_train, y_train, epochs=epochs, batch_size=64, validation_data=(x_test, y_test))
```

```
Epoch 1/10
552/552 [==============================] - 236s 427ms/step - loss: 3.3413 - accuracy: 0.2087 - val_loss: 0.9817 - val_accuracy: 0.7564
Epoch 2/10
552/552 [==============================] - 235s 425ms/step - loss: 0.8204 - accuracy: 0.7601 - val_loss: 0.3378 - val_accuracy: 0.9156
Epoch 3/10
552/552 [==============================] - 238s 432ms/step - loss: 0.5137 - accuracy: 0.8461 - val_loss: 0.2226 - val_accuracy: 0.9447
Epoch 4/10
552/552 [==============================] - 238s 430ms/step - loss: 0.3407 - accuracy: 0.8976 - val_loss: 0.1142 - val_accuracy: 0.9684
Epoch 5/10
552/552 [==============================] - 236s 428ms/step - loss: 0.3065 - accuracy: 0.9098 - val_loss: 0.0832 - val_accuracy: 0.9773
Epoch 6/10
552/552 [==============================] - 244s 443ms/step - loss: 0.2239 - accuracy: 0.9346 - val_loss: 0.0823 - val_accuracy: 0.9778
Epoch 7/10
552/552 [==============================] - 235s 426ms/step - loss: 0.2067 - accuracy: 0.9385 - val_loss: 0.0592 - val_accuracy: 0.9844
Epoch 8/10
552/552 [==============================] - 235s 426ms/step - loss: 0.1926 - accuracy: 0.9446 - val_loss: 0.0687 - val_accuracy: 0.9788
Epoch 9/10
552/552 [==============================] - 235s 426ms/step - loss: 0.1694 - accuracy: 0.9521 - val_loss: 0.0632 - val_accuracy: 0.9821
Epoch 10/10
552/552 [==============================] - 234s 424ms/step - loss: 0.1466 - accuracy: 0.9567 - val_loss: 0.0654 - val_accuracy: 0.9824
```

Figure 9 : Epoch

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Training accuracy")
plt.plot(history.history['val_accuracy'], label="val accuracy")
plt.title("Accuracy Graph")
plt.xlabel("epochs")
plt.ylabel("accuracy (0,1)")
plt.legend()

plt.figure(1)
plt.plot(history.history['loss'], label="training loss")
plt.plot(history.history['val_loss'], label="val loss")
plt.title("Loss Graph")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Figure 10: Acc and loss graph code

```
print(classification_report(test_labels,classes_x))
```

```
    accuracy                           0.95     12630
   macro avg       0.93      0.94      0.93     12630
weighted avg       0.96      0.95      0.95     12630
```

Figure 11: Classification report for model 1

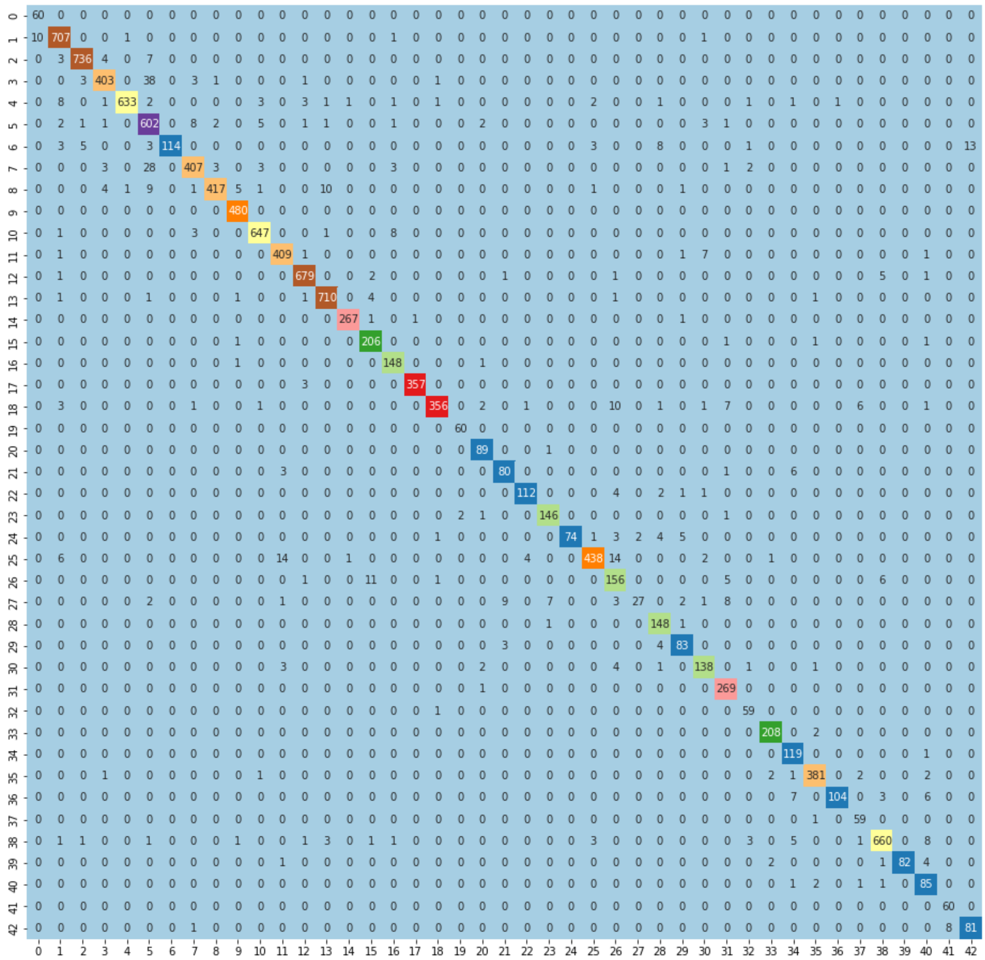

Figure 12 : Heatmap for model 1

Resnet50(Model 2)

The transfer learning foundation model is RESNET 50. Resnet 50 models are convolutional neural networks with 50 layers. And the accompanying snippets comprise the model building scripts and results.

```
import tensorflow as tf
print('TensoFl8ow Version: ', tf.__version__)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D, BatchNormalization, Dropout
#from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.applications import ResNet50

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
```

```
TensoFlow Version:  2.8.2
```

Figure 13: Importing the Restnet 50 model

```
n_epochs = 10
history =  model.fit(x_train, y_train,  batch_size = 64, epochs = n_epochs, verbose = 1,
            validation_data = (x_test, y_test), callbacks = [model_check, early, reduce_lr, csv_logger])
```

```
Epoch 1/10
552/552 [==============================] - 37s 54ms/step - loss: 3.3166 - accuracy: 0.2997 - val_loss: 162.2758 - val_accuracy: 0.0969 - lr: 0.0010
Epoch 2/10
552/552 [==============================] - 26s 47ms/step - loss: 1.5132 - accuracy: 0.6572 - val_loss: 35.1005 - val_accuracy: 0.0676 - lr: 0.0010
Epoch 3/10
552/552 [==============================] - 28s 50ms/step - loss: 1.1437 - accuracy: 0.7312 - val_loss: 0.5456 - val_accuracy: 0.8286 - lr: 0.0010
Epoch 4/10
552/552 [==============================] - 28s 50ms/step - loss: 1.0605 - accuracy: 0.7778 - val_loss: 0.5193 - val_accuracy: 0.8470 - lr: 0.0010
Epoch 5/10
552/552 [==============================] - 28s 50ms/step - loss: 0.3976 - accuracy: 0.8972 - val_loss: 0.1875 - val_accuracy: 0.9383 - lr: 0.0010
Epoch 6/10
552/552 [==============================] - 27s 49ms/step - loss: 0.1508 - accuracy: 0.9540 - val_loss: 0.1519 - val_accuracy: 0.9526 - lr: 0.0010
Epoch 7/10
552/552 [==============================] - 27s 49ms/step - loss: 0.0985 - accuracy: 0.9696 - val_loss: 0.0835 - val_accuracy: 0.9748 - lr: 0.0010
Epoch 8/10
552/552 [==============================] - 26s 47ms/step - loss: 0.1089 - accuracy: 0.9714 - val_loss: 0.1346 - val_accuracy: 0.9615 - lr: 0.0010
Epoch 9/10
552/552 [==============================] - 28s 50ms/step - loss: 0.0846 - accuracy: 0.9759 - val_loss: 0.0772 - val_accuracy: 0.9753 - lr: 0.0010
Epoch 10/10
552/552 [==============================] - 26s 47ms/step - loss: 0.4748 - accuracy: 0.8869 - val_loss: 1.1475 - val_accuracy: 0.6929 - lr: 0.0010
```

Figure 14: epoch for model 2

```
loss, acc = model.evaluate(x_test, y_test)
print('Accuracy: ', acc, '\nLoss    : ', loss)
```

```
123/123 [==============================] - 2s 14ms/step - loss: 1.1475 - accuracy: 0.6929
Accuracy:  0.6929354667663574
Loss     :  1.147536277770996
```

Figure 15: Evaluation

```
q = len(list(history.history['loss']))
plt.figure(figsize=(12, 6))
sns.lineplot(x = range(1, 1+q), y = history.history['accuracy'], label = 'Accuracy')
sns.lineplot(x = range(1, 1+q), y = history.history['loss'], label = 'Loss')
plt.xlabel('#epochs')
plt.ylabel('Training')
plt.legend()
```

Figure 16: Loss model

```
plt.figure(figsize=(12, 6))
sns.lineplot(x = range(1, 1+q), y = history.history['accuracy'], label = 'Train')
sns.lineplot(x = range(1, 1+q), y = history.history['val_accuracy'], label = 'Validation')
plt.xlabel('#epochs')
plt.ylabel('Accuracy')
plt.legend();
```

Figure 17: Training and validation model

```python
cmat = confusion_matrix(test_labels,classes_x)
plt.figure(figsize=(16,16))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```

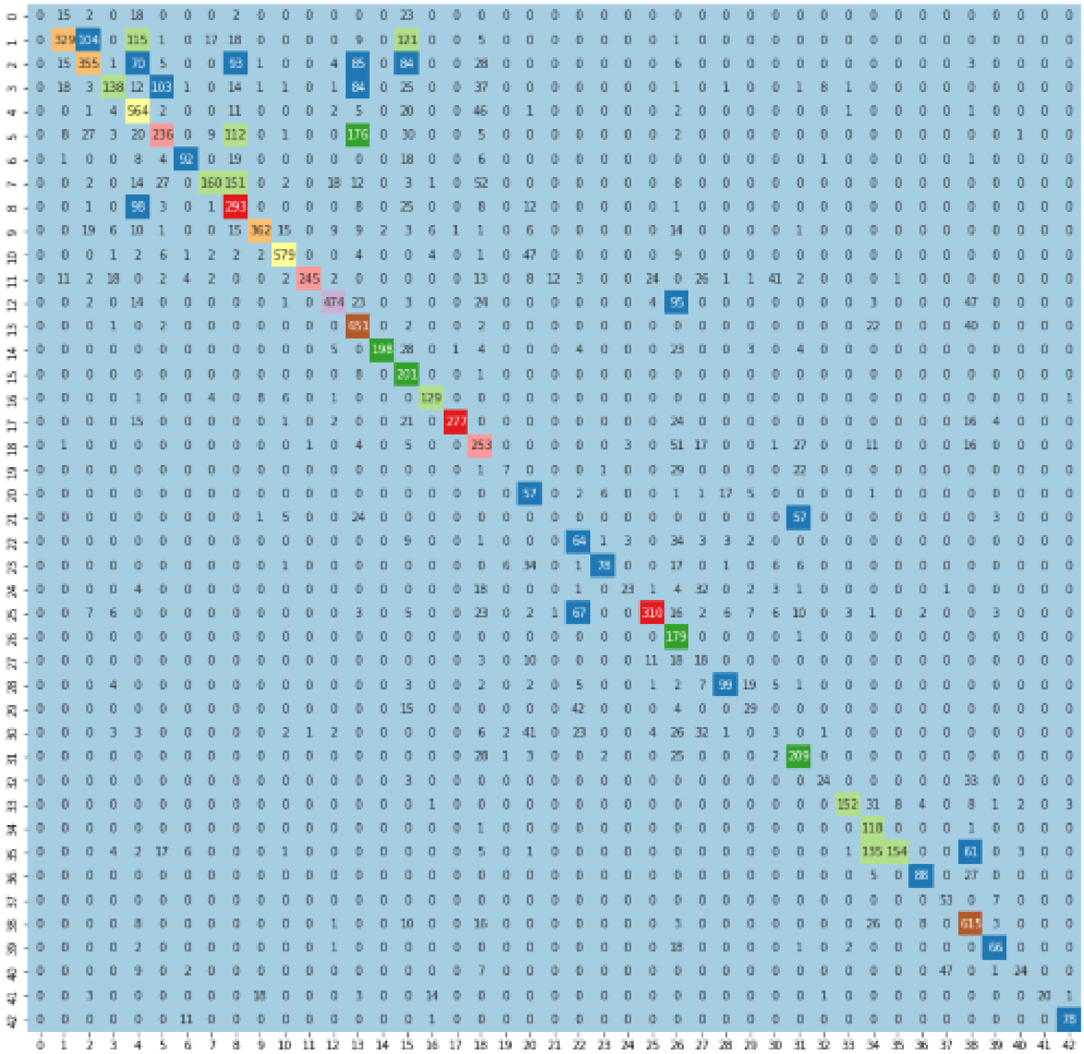

Figure 18: Heatmap Model 2

VGG16 (Model 3)

The first 13 levels of VGG16 are convolution networks, whereas the latter three layers are fully linked layers. The input shape in the VGG16 model is about 32*32, and the code and result are shown in the following excerpts.

```
vgg16 = VGG16(input_shape=(32,32,3), weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

Figure 19: VGG16 model building (Model 3)

```
add_model = Sequential()
add_model.add(Flatten(input_shape=vgg16.output_shape[1:]))

add_model.add(Dense(1024, activation='relu'))



add_model.add(Dense(y_train.shape[1], activation='softmax'))

model = Model(inputs=vgg16.input, outputs=add_model(vgg16.output))
learning_rate = 0.0001
def results(model):
  adam = Adam(lr=learning_rate)
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

model.summary()
```

Figure 20 : Further model for Model 3

```
epochs = 10
history = model.fit(x_train, y_train, batch_size=128, epochs=epochs,
validation_data=(x_test, y_test))
```

```
Epoch 1/10
276/276 [==============================] - 9s 26ms/step - loss: 2.3033 - accuracy: 0.6139 - val_loss: 0.9372 - val_accuracy: 0.7603
Epoch 2/10
276/276 [==============================] - 6s 23ms/step - loss: 0.6633 - accuracy: 0.8173 - val_loss: 0.7204 - val_accuracy: 0.8148
Epoch 3/10
276/276 [==============================] - 6s 23ms/step - loss: 0.4505 - accuracy: 0.8708 - val_loss: 0.5852 - val_accuracy: 0.8483
Epoch 4/10
276/276 [==============================] - 6s 22ms/step - loss: 0.3097 - accuracy: 0.9066 - val_loss: 0.5758 - val_accuracy: 0.8541
Epoch 5/10
276/276 [==============================] - 6s 22ms/step - loss: 0.2606 - accuracy: 0.9223 - val_loss: 0.5074 - val_accuracy: 0.8715
Epoch 6/10
276/276 [==============================] - 6s 22ms/step - loss: 0.2282 - accuracy: 0.9316 - val_loss: 0.5091 - val_accuracy: 0.8755
Epoch 7/10
276/276 [==============================] - 6s 22ms/step - loss: 0.2053 - accuracy: 0.9384 - val_loss: 0.5021 - val_accuracy: 0.8832
Epoch 8/10
276/276 [==============================] - 6s 22ms/step - loss: 0.2042 - accuracy: 0.9388 - val_loss: 0.4884 - val_accuracy: 0.8873
Epoch 9/10
276/276 [==============================] - 6s 24ms/step - loss: 0.2098 - accuracy: 0.9399 - val_loss: 0.4817 - val_accuracy: 0.8949
Epoch 10/10
276/276 [==============================] - 6s 23ms/step - loss: 0.1742 - accuracy: 0.9473 - val_loss: 0.4927 - val_accuracy: 0.8921
```

Figure 21 : Epoch for model 3

```
loss, acc = model.evaluate(x_test, y_test)
print('Accuracy: ', acc, '\nLoss     : ', loss)
```

```
123/123 [==============================] - 2s 12ms/step - loss: 0.4927 - accuracy: 0.8921
Accuracy:  0.8921193480491638
Loss     :  0.4926791489124298
```

ıble click (or enter) to edit

Figure 22 : Evaluation

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Training accuracy")
plt.plot(history.history['val_accuracy'], label="val accuracy")
plt.title("Accuracy Graph")
plt.xlabel("epochs")
plt.ylabel("accuracy (0,1)")
plt.legend()

plt.figure(1)
plt.plot(history.history['loss'], label="training loss")
plt.plot(history.history['val_loss'], label="val loss")
plt.title("Loss Graph")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Figure 23 : Accuracy and Loss model

```python
cmat = confusion_matrix(test_labels,classes_x)
plt.figure(figsize=(16,16))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```

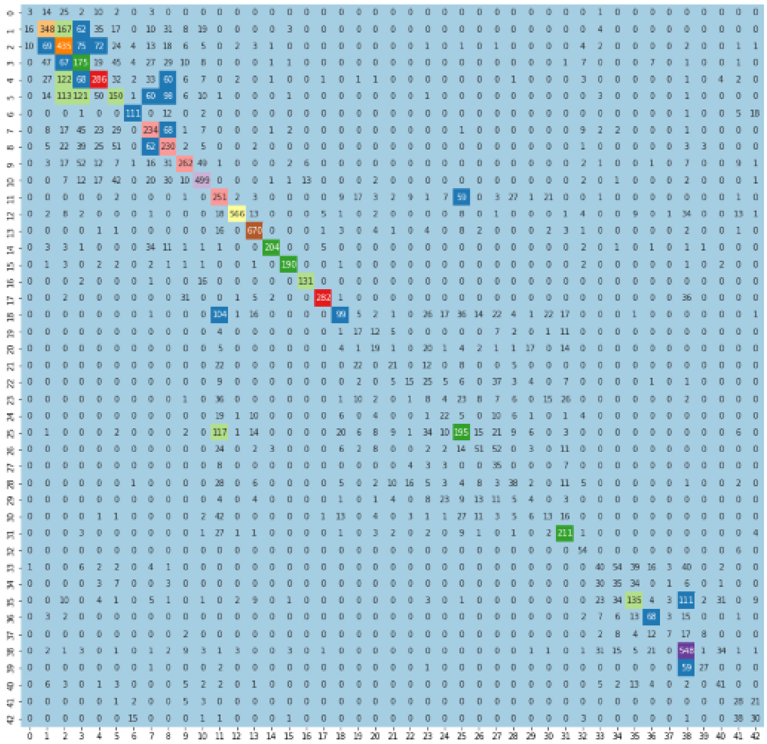

Figure 24 : Heatmap for model 3

VGG19(Model 4)

The VGG19 model is a CNN model as well, with about 19 layers of convolution layers and a fully linked layer divided into 16 and 3 layers. The code and the outcome are shown in the following excerpts.

```python
vgg19 = VGG19(input_shape=(32,32,3), weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
```

Figure 25: Model buiding for Model 4

```
add_model = Sequential()
add_model.add(Flatten(input_shape=vgg19.output_shape[1:]))

add_model.add(Dense(1024, activation='relu'))



add_model.add(Dense(y_train.shape[1], activation='softmax'))

model = Model(inputs=vgg19.input, outputs=add_model(vgg19.output))
learning_rate = 0.0001
def results(model):
  adam = Adam(lr=learning_rate)
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

model.summary()
```

Figure 26: Model summary code

```
epochs = 10
history = model.fit(x_train, y_train, batch_size=128, epochs=epochs,
validation_data=(x_test, y_test))

Epoch 1/10
276/276 [==============================] - 9s 29ms/step - loss: 2.6151 - accuracy: 0.5740 - val_loss: 1.2608 - val_accuracy: 0.6968
Epoch 2/10
276/276 [==============================] - 7s 26ms/step - loss: 0.7811 - accuracy: 0.7803 - val_loss: 0.7425 - val_accuracy: 0.8013
Epoch 3/10
276/276 [==============================] - 7s 26ms/step - loss: 0.5136 - accuracy: 0.8458 - val_loss: 0.7103 - val_accuracy: 0.8166
Epoch 4/10
276/276 [==============================] - 8s 28ms/step - loss: 0.3881 - accuracy: 0.8827 - val_loss: 0.5611 - val_accuracy: 0.8454
Epoch 5/10
276/276 [==============================] - 7s 26ms/step - loss: 0.3077 - accuracy: 0.9041 - val_loss: 0.5508 - val_accuracy: 0.8432
Epoch 6/10
276/276 [==============================] - 7s 26ms/step - loss: 0.2766 - accuracy: 0.9145 - val_loss: 0.5336 - val_accuracy: 0.8600
Epoch 7/10
276/276 [==============================] - 7s 26ms/step - loss: 0.2408 - accuracy: 0.9279 - val_loss: 0.5299 - val_accuracy: 0.8661
Epoch 8/10
276/276 [==============================] - 7s 26ms/step - loss: 0.2214 - accuracy: 0.9328 - val_loss: 0.5394 - val_accuracy: 0.8699
Epoch 9/10
276/276 [==============================] - 8s 28ms/step - loss: 0.2018 - accuracy: 0.9381 - val_loss: 0.5365 - val_accuracy: 0.8740
Epoch 10/10
276/276 [==============================] - 7s 26ms/step - loss: 0.1889 - accuracy: 0.9434 - val_loss: 0.5366 - val_accuracy: 0.8743
```

Figure 27: epoch

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Training accuracy")
plt.plot(history.history['val_accuracy'], label="val accuracy")
plt.title("Accuracy Graph")
plt.xlabel("epochs")
plt.ylabel("accuracy (0,1)")
plt.legend()

plt.figure(1)
plt.plot(history.history['loss'], label="training loss")
plt.plot(history.history['val_loss'], label="val loss")
plt.title("Loss Graph")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Figure 28 : Loss and accuracy model

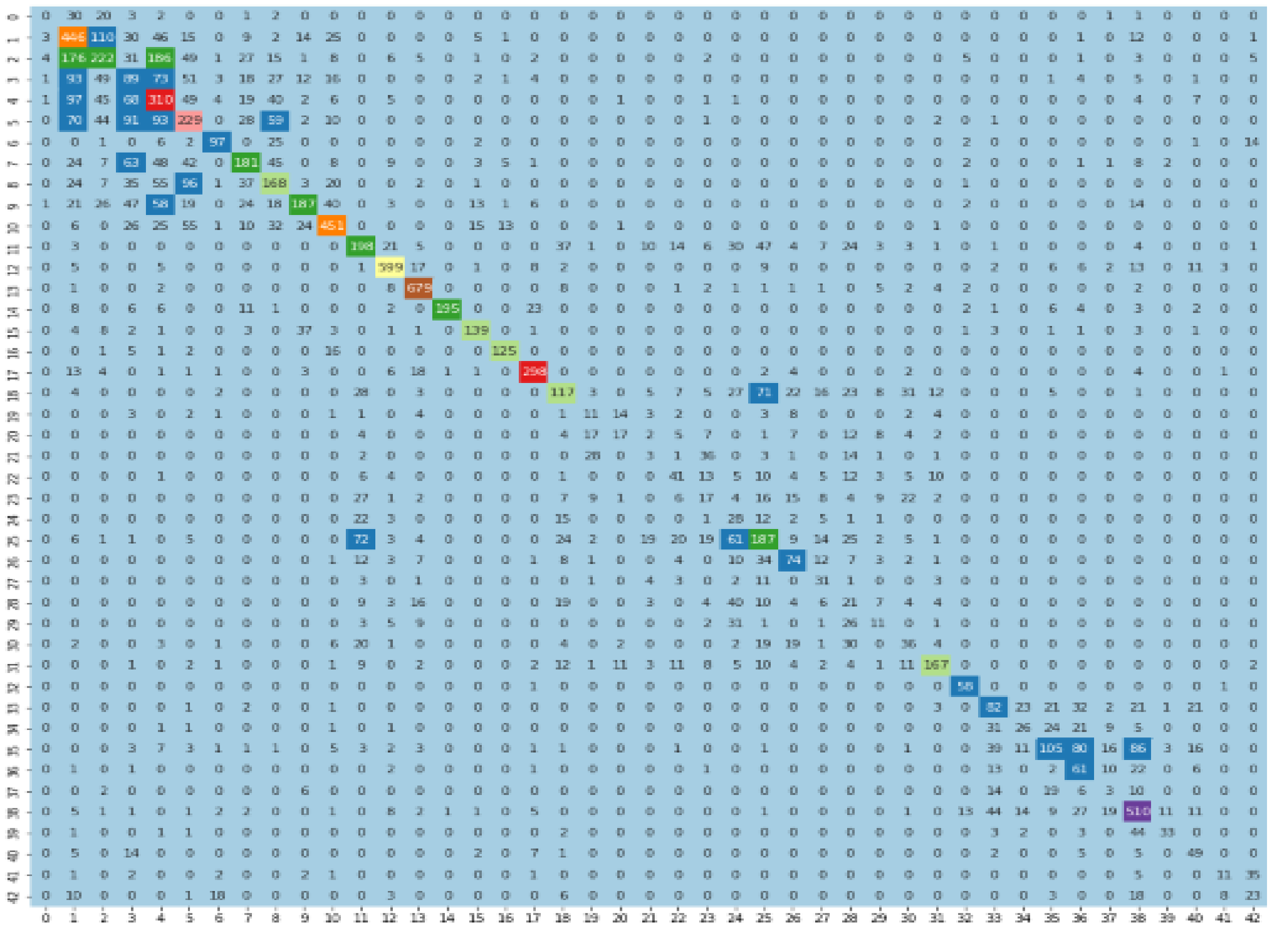

Figure 29: Heatmap model 4

References

Stallkamp, J., Schlipsing, M., Salmen, J. and Igel, C., 2011. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. *The 2011 International Joint Conference on Neural Networks*,.

.