Traffic Sign Recognition

Fai Aladhyani

Shahad Alotaibi

Maram Alshehri

Abstract

This project applies deep learning to a traffic-related problem, specifically Traffic Sign Recognition. We designed and implemented a model using a chosen dataset. Our results demonstrate the potential of deep learning in solving traffic-related problems and highlight the importance of careful data preprocessing and model selection

Table of Contents

Introduction

As part of the Project: Traffic-Related Deep Learning Project, our objective is to design and implement a deep learning model that can effectively solve a traffic-related problem. We have chosen to focus on Traffic sign recognition, a critical aspect of traffic management that has significant implications for road safety and efficiency.

Objective

The objectives of this project are:

• To select a traffic-related problem and implement a deep learning model to address it.

• To collect and preprocess a dataset relevant to the chosen problem.

• To design and implement a suitable deep learning model using a chosen framework (e.g., TensorFlow, PyTorch).

• To train and evaluate the model using relevant performance metrics.

• To fine-tune hyperparameters to optimize model performance.

• To document the entire workflow, including data collection, preprocessing, model selection, implementation, training, and evaluation.

• To present the project's findings, including results, challenges, and conclusions

Our Team :

Name	Tasks
Shahad Alotaibi	Collecting data, Code, trying Experiments on our model, edit presentation.
Maram Alshehri	Collecting data, Code, trying Experiments on our model, prepare presentation.
Fai Aladhyani	Collecting data, trying Experiments on our model, Write the Report

Conclusion

Through this project, we gained hands-on experience with deep learning frameworks and tools, including TensorFlow, PyTorch, Keras ,We also developed our skills in data preprocessing, model design, and evaluation

List All Figures

Traffic Sign Recognition using Deep Learning(CNN)

Traffic sign recognition using deep learning leverages Convolutional Neural Networks (CNNs) to classify traffic signs from images. CNNs automatically extract features from images and learn spatial hierarchies, making them highly effective for image classification tasks.

Import libraries

This script outlines a basic image classification workflow using Keras for deep learning. It includes data processing, model building, and model training.

9]:

```
import os
import cv2
import numpy as np
import pickle
import random
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.losses import categorical_crossentropy
from keras.utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from sklearn.metrics import classification_report
```

Configuration Variables for Traffic Sign Recognition

This section defines various configuration variables used for setting up and managing the traffic sign recognition project using deep learning.

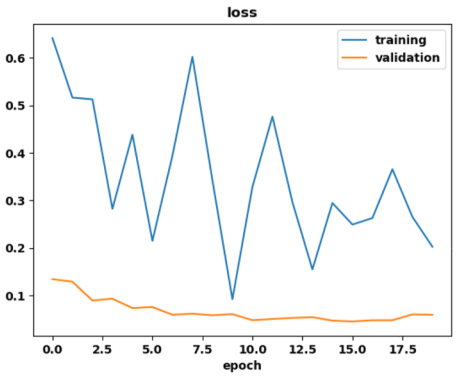
7]:

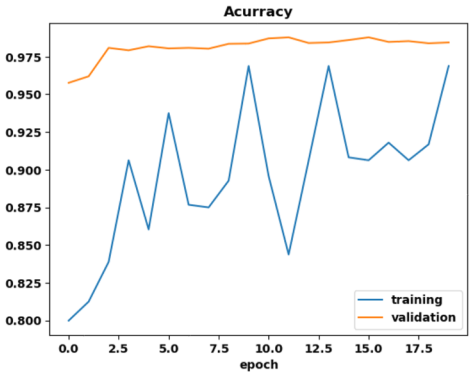
```
path = "Dataset"
labelFile = 'labels.csv'
batch_size_val=32
epochs_val=20
imageDimesions = (32,32,3)
testRatio = 0.2
validationRatio = 0.2
```

```python
# Get the predicted labels
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Generate and print classification report
report = classification_report(y_true, y_pred_classes)
print("Classification Report:")
print(report)
```

```
218/218 ━━━━━━━━━━━━━━━━━━━━ 7s 31ms/step
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        39
           1       0.98      1.00      0.99       354
           2       0.99      0.98      0.99       394
           3       0.99      0.96      0.98       230
           4       0.98      0.99      0.98       326
           5       0.96      0.97      0.97       339
           6       0.99      1.00      0.99        70
           7       0.91      0.99      0.95       257
           8       1.00      0.91      0.95       263
           9       1.00      0.99      0.99       264
          10       1.00      1.00      1.00       370
          11       0.99      1.00      0.99       209
          12       0.99      1.00      0.99       364
          13       1.00      1.00      1.00       384
          14       1.00      0.99      1.00       132
          15       0.99      0.96      0.98       110
          16       1.00      1.00      1.00        65
          17       1.00      1.00      1.00       208
          18       0.94      0.99      0.97       241
          19       1.00      0.81      0.89        31
          20       1.00      0.99      0.99        68
          21       0.94      1.00      0.97        48
          22       1.00      1.00      1.00        74
          23       1.00      0.97      0.99        69
          24       0.98      1.00      0.99        46
          25       1.00      1.00      1.00       293
          26       0.94      0.87      0.90       100
          27       0.94      0.98      0.96        46
          28       0.92      0.99      0.96        98
          29       0.96      0.85      0.90        53
          30       0.99      0.95      0.97        85
          31       0.99      0.98      0.99       139
          32       1.00      1.00      1.00        42
          33       1.00      0.98      0.99       131
          34       1.00      1.00      1.00        92
          35       1.00      0.99      1.00       248
          36       0.97      1.00      0.98        62
          37       1.00      1.00      1.00        32
          38       0.99      1.00      1.00       387
          39       0.98      0.98      0.98        54
          40       1.00      0.99      0.99        67
          41       1.00      0.98      0.99        42
          42       1.00      0.97      0.99        34

    accuracy                           0.98      6960
   macro avg       0.98      0.98      0.98      6960
weighted avg       0.98      0.98      0.98      6960
```

```
plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('loss')
plt.xlabel('epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Acurracy')
plt.xlabel('epoch')
plt.show()
```





Model Evaluation:

```
score =model.evaluate(X_test,y_test,verbose=0)
print('Test Loss:',score[0])
print('Test Accuracy:',score[1])
```

```
Test Loss: 0.06176169216632843
Test Accuracy: 0.9833333492279053
```

```
history=model.fit(dataGen.flow(X_train,y_train,batch_size=batch_size_val),steps_per_
```

```
Epoch 1/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 68s 98ms/step - accuracy: 0.7891 - loss: 0.6722 - val
_accuracy: 0.9576 - val_loss: 0.1340
Epoch 2/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.8125 - loss: 0.5161 - val_a
ccuracy: 0.9619 - val_loss: 0.1287
Epoch 3/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 65s 94ms/step - accuracy: 0.8369 - loss: 0.5256 - val
_accuracy: 0.9808 - val_loss: 0.0890
Epoch 4/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9062 - loss: 0.2819 - val_a
ccuracy: 0.9792 - val_loss: 0.0929
Epoch 5/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 67s 96ms/step - accuracy: 0.8526 - loss: 0.4613 - val
_accuracy: 0.9819 - val_loss: 0.0733
Epoch 6/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9375 - loss: 0.2146 - val_a
ccuracy: 0.9804 - val_loss: 0.0754
Epoch 7/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 66s 94ms/step - accuracy: 0.8779 - loss: 0.3991 - val
_accuracy: 0.9808 - val_loss: 0.0592
Epoch 8/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8750 - loss: 0.6019 - val_a
ccuracy: 0.9802 - val_loss: 0.0613
Epoch 9/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 66s 94ms/step - accuracy: 0.8931 - loss: 0.3404 - val
_accuracy: 0.9835 - val_loss: 0.0581
Epoch 10/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9688 - loss: 0.0918 - val_a
ccuracy: 0.9837 - val_loss: 0.0605
Epoch 11/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 77s 111ms/step - accuracy: 0.8934 - loss: 0.3355 - va
l_accuracy: 0.9871 - val_loss: 0.0476
Epoch 12/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.8438 - loss: 0.4760 - val_a
ccuracy: 0.9878 - val_loss: 0.0502
Epoch 13/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 63s 90ms/step - accuracy: 0.9076 - loss: 0.2897 - val
_accuracy: 0.9840 - val_loss: 0.0524
Epoch 14/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9688 - loss: 0.1546 - val_a
ccuracy: 0.9844 - val_loss: 0.0540
Epoch 15/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 66s 95ms/step - accuracy: 0.9084 - loss: 0.2919 - val
_accuracy: 0.9860 - val_loss: 0.0467
Epoch 16/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9062 - loss: 0.2489 - val_a
ccuracy: 0.9878 - val_loss: 0.0451
Epoch 17/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 67s 96ms/step - accuracy: 0.9183 - loss: 0.2575 - val
_accuracy: 0.9847 - val_loss: 0.0474
Epoch 18/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9062 - loss: 0.3653 - val_a
ccuracy: 0.9853 - val_loss: 0.0475
Epoch 19/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 76s 109ms/step - accuracy: 0.9177 - loss: 0.2689 - va
l_accuracy: 0.9838 - val_loss: 0.0599
Epoch 20/20
695/695 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9688 - loss: 0.2022 - val_a
ccuracy: 0.9844 - val_loss: 0.0592
```

```
print(model.summary())
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 60)	1,560
conv2d_1 (Conv2D)	(None, 24, 24, 60)	90,060
max_pooling2d (MaxPooling2D)	(None, 12, 12, 60)	0
conv2d_2 (Conv2D)	(None, 10, 10, 30)	16,230
conv2d_3 (Conv2D)	(None, 8, 8, 30)	8,130
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 30)	0
dropout (Dropout)	(None, 4, 4, 30)	0
flatten (Flatten)	(None, 480)	0
dense (Dense)	(None, 500)	240,500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 43)	21,543

display

Total params: 378,023 (1.44 MB)
Trainable params: 378,023 (1.44 MB)
Non-trainable params: 0 (0.00 B)
None

Data Augmentation:

```
dataGen= ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, zoom_range=0.2, shear_range=0.1, rotat
```

```
dataGen.fit(X_train)
batches= dataGen.flow(X_train,y_train,batch_size=20)
X_batch,y_batch = next(batches)
```

Converting Labels to Categorical Format:

```
y_train = to_categorical(y_train,noOfClasses)
y_validation = to_categorical(y_validation,noOfClasses)
y_test = to_categorical(y_test,noOfClasses)
```

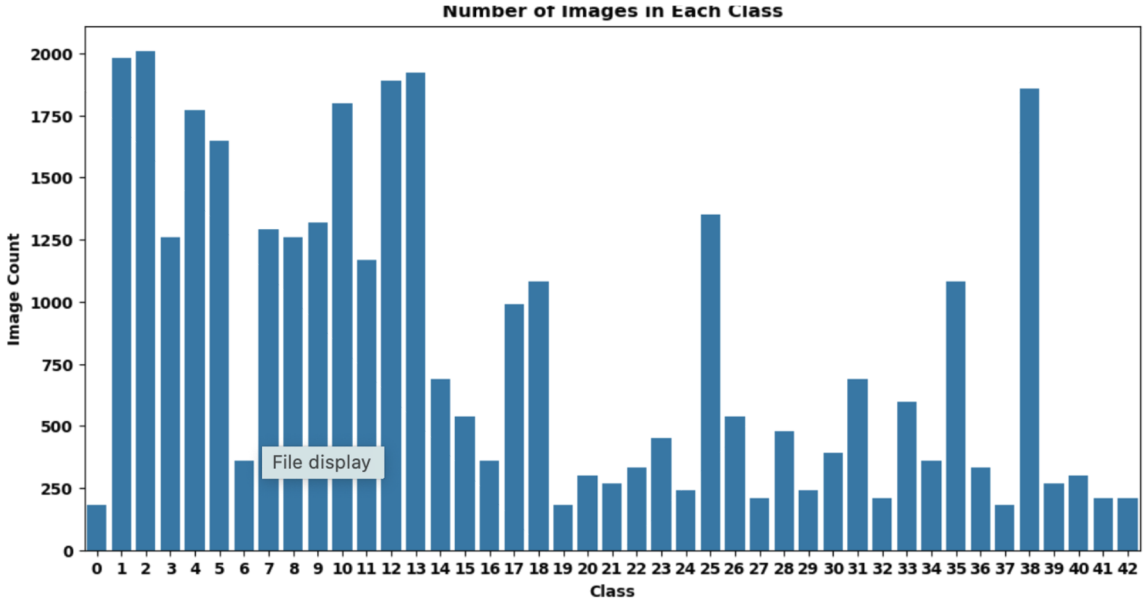
CNN Model Creation:

This code creates, compiles, trains, and evaluates a Convolutional Neural Network (CNN) model using TensorFlow/Keras. The model is designed for image classification, likely for tasks such as Traffic Sign Recognition, as suggested by the previous context. The code also includes visualizations of training progress and evaluation on the test set.

```
model= Sequential()
model.add((Conv2D(60,(5,5),input_shape=(imageDimesions[0],imageDimesions[1],1),activation='relu')))  # ADDING MOR
model.add((Conv2D(60, (5,5), activation='relu')))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add((Conv2D(30, (3,3),activation='relu')))
model.add((Conv2D(30, (3,3), activation='relu')))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(500,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(noOfClasses,activation='softmax'))
```

```
c:\Users\HP\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass a
n `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` objec
t as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(optimizer=Adam(learning_rate=0.001), loss=categorical_crossentropy, metrics=['accuracy'])
```



Initializing Preprocessing Functions:

```
def grayscale(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    return img
def equalize(img):
    img =cv2.equalizeHist(img)
    return img
def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img
```

Preprocessing Training, Validation, and Test Sets:

```
X_train=np.array(list(map(preprocessing,X_train)))
X_validation=np.array(list(map(preprocessing,X_validation)))
X_test=np.array(list(map(preprocessing,X_test)))
```

Reshaping the Data:

```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_validation=X_validation.reshape(X_validation.shape[0],X_validation.shape[1],X_validation.shape[2],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)
```

Printing the Shapes of Train, Validation, and Test Data

This block of code is used to display the shapes (dimensions) of the training, validation, and testing datasets for both features (images) and labels (class numbers). It helps ensure that the data has been split correctly into the respective sets.

```
print("Data Shapes")
print("Train",end = "");print(X_train.shape,y_train.shape)
print("Validation",end = "");print(X_validation.shape,y_validation.shape)
print("Test",end = "");print(X_test.shape,y_test.shape)
```

```
Data Shapes
Train(22271, 32, 32, 3) (22271,)
Validation(5568, 32, 32, 3) (5568,)
Test(6960, 32, 32, 3) (6960,)
```

Data Preprocessing and Augmentation for Training, Validation, and Test Data

Loading Label Data:

```
data=pd.read_csv(labelFile)
print("data shape ",data.shape,type(data))
```

```
data shape  (43, 2) <class 'pandas.core.frame.DataFrame'>
```

```
import os

# Path to the directory containing the class folders
dataset_dir = "Dataset"


# Initialize a dictionary to store the count of images for each class
class_image_count = {}

# Iterate over the class folders (assuming they are named from 0 to 42)
for class_name in range(43):  # Assuming class names are from 0 to 42
    class_dir = os.path.join(dataset_dir, str(class_name))
    if os.path.exists(class_dir) and os.path.isdir(class_dir):
        # Count the number of files (images) in the class directory
        num_images = len(os.listdir(class_dir))
        class_image_count[class_name] = num_images

# Create a DataFrame from the class_image_count dictionary
df = pd.DataFrame(list(class_image_count.items()), columns=['Class', 'Image Count'])

# Plotting the countplot using Seaborn
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Class', y='Image Count')
plt.title('Number of Images in Each Class')
plt.xlabel('Class')
plt.ylabel('Image Count')
plt.show()
```


30
100
100
70
30

Data Loading and Preparation

This section of the code handles loading and preparing the dataset of traffic signs for further processing and model training.

```
count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
noOfClasses=len(myList)
print("Importing Classes.....")
for x in range (0,len(myList)):
    myPicList = os.listdir(path+"/"+str(count))
    for y in myPicList:
        curImg = cv2.imread(path+"/"+str(count)+"/"+y)
        images.append(curImg)
        classNo.append(count)
    print(count, end =" ")
    count +=1
print(" ")
images = np.array(images)
classNo = np.array(classNo)
```

```
Total Classes Detected: 43
Importing Classes.....
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42
```

Data Splitting for Training, Testing, and Validation

```
X_train, X_test, y_train, y_test = train_test_split(images, classNo, test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validationRatio)
```

Displaying Random Images from the Dataset

In this block of code, we are displaying a grid of 25 random images from the combined training and testing datasets.

```
# Combine the X_train and X_test sets for displaying images
all_images = np.concatenate((X_train, X_test))

# Choose 25 random images
random_images = random.sample(range(all_images.shape[0]), 25)

# Display the random images
plt.figure(figsize=(10, 10))
for i, idx in enumerate(random_images):
    image = all_images[idx]

    plt.subplot(5, 5, i+1)
    plt.imshow(image)
    plt.axis('off')
```

Model Saving:

```
model.save("model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
s.saving.save_model(model)`. This file format is considered legacy. We recommend u
sing instead the native Keras format, e.g. `model.save('my_model.keras')` or `kera
s.saving.save_model(model, 'my_model.keras')`.
```

Conclusion

The CNN model achieved strong performance on the image classification task through effective preproc
improvements could focus on fine-tuning the model or exploring deeper architectures for better accura

Questions

What challenges did you face during data collection and how did you overcome them?

Data Collecting took a lot of the time to find the appropriate data size and of our target.

Why did you choose your specific model architecture?

CNN because it is good with object recognition.

How does the performance of your chosen model compare to alternative approaches, and what factors contributed to the differences?

We did not try another model to compare, we only try and used CNN because for time constrain.

What did you learn from storing the data in a database, and how did it impact your Workflow?

We did not used a Database.

If you were to extend this project, what additional features or improvements would you consider?

Future improvements:

- Experiment with over-sampling and under-sampling techniques
- Add more signs.
- Improve model accuracy.
- Real-time traffic sign detection