|  |  |  |
| --- | --- | --- |
| **University of Petra** | شعار جامعة البترا5 - | Text, letter  Description automatically generated |
| Faculty of Information Technology | كلية تكنولوجيا المعلومات |
| Department of Data Science and Artificial Intelligence | قسم علم البيانات والذكاء الاصطناعي |

# 

**Deep learning**

**Traffic Sign Classification**

Done by:

Faris Anki 202010569

Omar mousa 202020036

Abdalrahman Radwan 202020030

Table of contents

Description\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2

Importing the required libraries and modules \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2

Loading and preprocessing the datasets \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2-3

Model architecture \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

Model training\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4

Model evaluation and visualization \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4

Model saving \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4

Layers \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5-6

**German traffic signs recognition benchmark(gtsrb)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

Description\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

Datasets details \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

Dataset structure \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7-8

**Data usages** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8-9

Data loading\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

Data preprocessing \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

Data splitting \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

Model training \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

Model evaluation \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

Accuracy and loss\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

Code \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11-14

**Description:**

This code performs traffic sign classification using a Convolutional Neural Network (CNN) model. It trains the model on a dataset of traffic sign images, evaluates its performance, and saves the trained model for future use.

1. **Importing the required libraries and modules:**

- numpy: For numerical computations and array operations.

- pandas: For data manipulation and analysis.

- matplotlib: For plotting and visualization.

- cv2: OpenCV library for image processing.

- tensorflow: Deep learning framework for building and training neural networks.

- PIL: Python Imaging Library for image manipulation.

- os: For interacting with the operating system.

- sklearn: For model evaluation and data preprocessing.

1. **Loading and Preprocessing the Dataset:**

- The code loops over the classes (traffic sign labels) and loads the images for each class from the 'train' directory.

- Each image is resized to a fixed size of 30x30 pixels using PIL.

- Images with a shape of (30, 30, 3) are added to the 'data' list along with their corresponding labels in the 'labels' list.

- The 'data' and 'labels' lists are converted to NumPy arrays for further processing.

- The dataset is split into training and testing sets using the train\_test\_split function from sklearn.

1. **Model Architecture:**

- The code builds a sequential CNN model using the Keras API.

- The model consists of the following layers:

- Two Conv2D layers with 32 filters, kernel size (5, 5), and ReLU activation function.

- A MaxPool2D layer with pool size (2, 2) for downsampling.

- A Dropout layer with a rate of 0.25 to prevent overfitting.

- Two Conv2D layers with 64 filters, kernel size (3, 3), and ReLU activation function.

- Another MaxPool2D layer and Dropout layer.

- A Flatten layer to flatten the output for dense layers.

- A Dense layer with 256 units and ReLU activation function.

- A Dropout layer with a rate of 0.5.

- A Dense output layer with 43 units (corresponding to the number of traffic sign classes) and softmax activation function.

1. **Model Training:**

- The model is compiled with the categorical\_crossentropy loss function, adam optimizer, and accuracy metric.

- The model is trained on the training dataset for a specified number of epochs (30 in this code).

- The training history is stored in the 'history' variable for later visualization.

1. **Model Evaluation and Visualization:**

- The code plots the accuracy and loss curves for training and validation datasets using matplotlib.

- The code loads the test dataset from a CSV file.

- The images in the test dataset are preprocessed similarly to the training images.

- The model predicts the classes for the test dataset and calculates the accuracy.

- The accuracy score is printed to the console.

1. **Model Saving:**

- The trained model is saved as 'traffic\_classifier.h5' using the save method of the Keras model.

1. **Layers:**

Input Layer:

This layer is not explicitly defined in the code but is implicitly present when passing input data to the model.

**Convolutional Layer 1:**

Conv2D layer with 32 filters, kernel size (5, 5), and ReLU activation.

Responsible for extracting low-level features from the input images.

**Convolutional Layer 2:**

Conv2D layer with 32 filters, kernel size (5, 5), and ReLU activation.

Extracts higher-level features from the input images.

**Max Pooling Layer 1:**

MaxPool2D layer with pool size (2, 2).

Performs downsampling, reducing the spatial dimensions of the feature maps.

**Dropout Layer 1:**

Dropout layer with a dropout rate of 0.25.

Introduces regularization by randomly dropping out a fraction of the units to prevent overfitting.

**Convolutional Layer 3:**

Conv2D layer with 64 filters, kernel size (3, 3), and ReLU activation.

Continues to extract more complex and abstract features.

**Convolutional Layer 4:**

Conv2D layer with 64 filters, kernel size (3, 3), and ReLU activation.

Further enhances the feature extraction process.

**Max Pooling Layer 2:**

MaxPool2D layer with pool size (2, 2).

Performs additional downsampling to reduce the spatial dimensions of the feature maps.

**Dropout Layer 2:**

Dropout layer with a dropout rate of 0.25.

Provides further regularization to prevent overfitting.

**Flatten Layer:**

Converts the 2D feature maps into a 1D vector to prepare for the fully connected layers.

**Fully Connected Layer 1:**

Dense layer with 256 units and ReLU activation.

Learns high-level representations and patterns from the flattened feature vector.

**Dropout Layer 3:**

Dropout layer with a dropout rate of 0.5.

Adds regularization to prevent overfitting.

**Output Layer:**

Dense layer with 43 units (corresponding to the number of traffic sign classes) and softmax activation.

Produces the final predictions by generating a probability distribution over the classes.

**German Traffic Sign Recognition Benchmark (GTSRB) Dataset**

**Description:**

This dataset is a widely used dataset for training and evaluating traffic sign recognition models. It consists of images of various traffic signs captured under different conditions and viewpoints. The dataset is primarily intended for developing and evaluating algorithms for traffic sign classification tasks.

**Dataset Details:**

Number of Classes: 43

Total Images: Approximately 50,000

Training Set: 39,209 images

Testing Set: 12,630 images

**Dataset Structure:**

The GTSRB dataset is organized into two main folders:

1. **Meta folder:**

Contains a CSV file named 'Meta.csv' that provides metadata about each traffic sign class.

The CSV file includes information such as class ID, sign name, and number of training and testing samples for each class.

1. **Train folder:**

-Contains subfolders named from '0' to '42', representing the class labels of traffic signs.

-Each subfolder contains images corresponding to the respective traffic sign class.

**Dataset Usage:**

1. **Data Loading:**

The code loads the images from the 'Train' folder.

It loops over the class folders and reads the images for each class.

1. **Data Preprocessing:**

The images are resized to a fixed size (30x30 pixels in the provided code) using the PIL library.

The resized images are converted to NumPy arrays and added to the 'data' list.

The corresponding labels for each image are also stored in the 'labels' list.

1. **Data Splitting:**

The dataset is split into training and testing sets using the train\_test\_split function from the sklearn library.

The specified test\_size (0.2 in the provided code) determines the proportion of images reserved for testing.

1. **Model Training:**

The code builds a CNN model using the Keras API.

The model architecture includes convolutional layers, pooling layers, dropout layers, and fully connected layers.

The model is compiled with appropriate loss function, optimizer, and metrics.

It is trained on the training dataset for a specified number of epochs.

1. **Model Evaluation:**

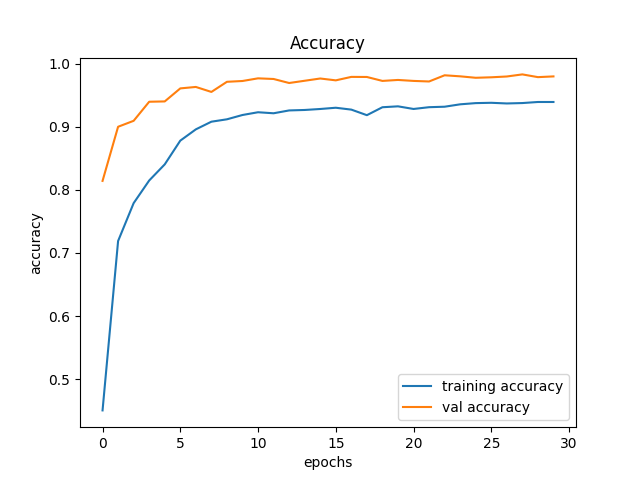
The code evaluates the trained model on the testing dataset.

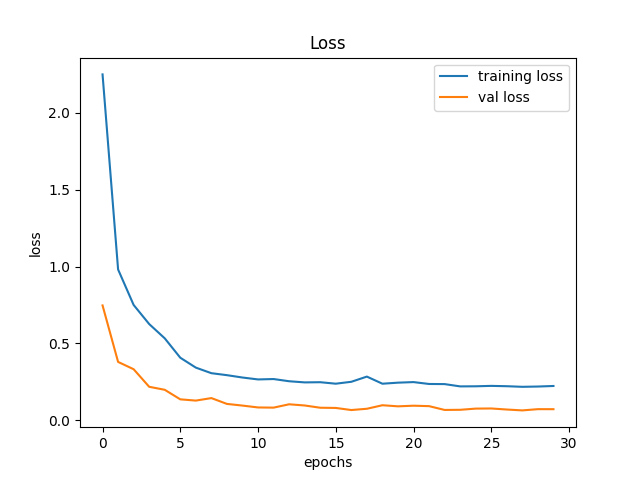
It loads the test dataset from a CSV file named 'Test.csv' in the same directory.

The images in the test dataset are preprocessed in a similar way to the training images.

The model predicts the classes for the test images and calculates the accuracy score.

**Here is the accuracy and the loss**





**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

data = []

labels = []

classes = 43

cur\_path = os.getcwd()

# 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(os.path.join(path, a))

image = image.resize((30, 30))

image = np.array(image)

if image.shape == (30, 30, 3): # Check image shape

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 = 30

history = model.fit(X\_train, y\_train, batch\_size=32, epochs=epochs, validation\_data=(X\_test, y\_test))

model.save("my\_model.h5")

# 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 the test dataset

from sklearn.metrics import accuracy\_score

test\_data = pd.read\_csv('Test.csv')

test\_labels = test\_data["ClassId"].values

test\_images = test\_data["Path"].values

test\_data = []

for img in test\_images:

image = Image.open(os.path.join(cur\_path, img))

image = image.resize((30, 30))

test\_data.append(np.array(image))

test\_data = np.array(test\_data)

# Make predictions

pred\_probabilities = model.predict(test\_data)

pred\_classes = np.argmax(pred\_probabilities, axis=1)

# Accuracy with the test data

accuracy = accuracy\_score(test\_labels, pred\_classes)

print("Accuracy:", accuracy)

# Save the model

model.save('traffic\_classifier.h5')