Deep Learning for Multiclass Traffic Sign Recognition using the German Traffic Sign Recognition Benchmark Dataset

Author: Manenimabasi Martin Udoh

Institution: Covenant University

Abstract:

Traffic sign classification is a fundamental task in the field of autonomous driving, contributing to the safety and efficiency of road transportation systems. In this project, I leverage deep learning techniques, specifically the LeNet architecture, to develop a robust multiclassifier model for the classification of diverse traffic signs.

The German Traffic Sign Recognition Benchmark dataset, comprising 43 distinct sign classes, serves as the foundation for our research. By meticulously designing our neural network and employing convolutional layers, rectified linear units (ReLU), pooling layers, and fully connected layers, we demonstrate the model's capacity to effectively recognize and categorize traffic signs, making it a valuable tool for autonomous vehicles and intelligent transportation systems.

This project explores the application of deep learning for multiclass traffic sign recognition, utilizing the German Traffic Sign Recognition Benchmark Dataset. The primary objective is to develop a robust model capable of accurately classifying traffic signs, contributing to improved road safety and intelligent transportation systems.

The project involves data loading, preprocessing, and the implementation of a Convolutional Neural Network (CNN) based on LeNet architecture. Notably, the model attains impressive performance, achieving a validation accuracy of 83.17%. and a validation loss of 0.719. These findings emphasize the effectiveness of deep learning for traffic sign recognition and extend its potential to various image classification tasks. Furthermore, they underscore the significance of advanced technologies in enhancing road safety and intelligent transportation systems.

In conclusion, this project's success suggests the feasibility of deploying deep learning models in real-world applications to improve traffic sign recognition accuracy, contributing to safer roads and more efficient transportation networks.

Keywords: Deep Learning, Convolutional Neural Network, Traffic Sign Recognition, Multiclass Classification, German Traffic Sign Recognition Benchmark

Table of Contents

Contact Information:

- **Email:** manenimabasi.udoh@stu.cu.edu.ng
- **LinkedIn:** manenimabasi-udoh
- **GitHub:** manenim

INTRODUCTION

Traffic sign recognition holds pivotal significance in the context of self-driving cars and intelligent transportation systems. Autonomous vehicles rely on the accurate detection and interpretation of traffic signs to make informed decisions, ensure road safety, and optimize navigation. The ability to interpret and respond to a wide array of traffic signs, from speed limits to warnings and prohibitions, is crucial for the safe and efficient operation of autonomous vehicles. (LeCun, Bottou, Bengio, & Haffner, 1998)

In this project, we address the imperative challenge of traffic sign classification by harnessing the power of deep learning. Our primary objective is to construct a multiclassifier model that can effectively classify a diverse range of traffic signs. To accomplish this, I employ the LeNet architecture, a seminal neural network structure introduced by Yann LeCun. LeNet's adaptability and capacity to learn intricate features make it a suitable choice for the task.

The dataset chosen for this research is the German Traffic Sign Recognition Benchmark, which encompasses 43 distinct classes of traffic signs. Each class represents a unique sign, encompassing a myriad of shapes, colors, and symbols. The model's ability to accurately identify and categorize these signs is pivotal for enhancing the safety and reliability of autonomous vehicles.

This project's relevance extends beyond the scope of autonomous driving; it has broader implications for computer vision and deep learning applications. The convolutional layers, ReLU activation functions, pooling, dropout layers, and fully connected layers employed in the model are pivotal components of contemporary deep learning architectures. By dissecting and demonstrating the efficacy of these components within the context of traffic sign classification, we contribute valuable insights to the deep learning community.

In the subsequent sections, I delve into the architectural intricacies of our model, present experimental results, and discuss the implications of the findings. Ultimately, this research aims to advance the state-of-the-art in traffic sign recognition, with a vision of safer and more efficient autonomous transportation systems.

DATA LOADING

In [36]:
```
import pickle

with open("train.p", mode='rb') as training_data:
    train = pickle.load(training_data)
with open("valid.p", mode='rb') as validation_data:
    valid = pickle.load(validation_data)
with open("test.p", mode='rb') as testing_data:
    test = pickle.load(testing_data)
```

In [37]:
```
X_train, y_train = train['features'], train['labels']
X_validation, y_validation = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

In [38]:
```
X_test.shape
```

Out[38]:
```
(12630, 32, 32, 3)
```

In [39]:
```
X_train.shape
```

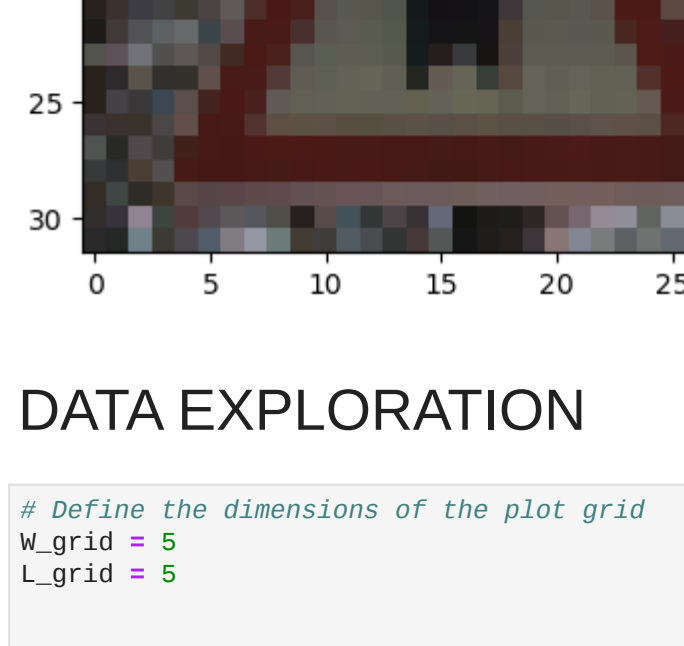
Out[39]:
```
(34799, 32, 32, 3)
```

In [40]:
```
X_validation.shape
```

Out[40]:
```
(4410, 32, 32, 3)
```

In [43]:
```
#visualising a random sample
import numpy as np
import matplotlib.pyplot as plt
i = np.random.randint(1, len(X_test))
plt.imshow(X_test[i])
print('label = ', y_test[i])
```

```
label =  11
```



DATA EXPLORATION

In [44]:
```
# Define the dimensions of the plot grid
W_grid = 5
L_grid = 5


fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,10))

axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array

n = len(X_test) # get the length of the training dataset

# Select a random number from 0 to n_training
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables
    index = np.random.randint(0, n)
    axes[i].imshow(X_test[index])
    axes[i].set_title(y_test[index], fontsize = 15)
    axes[i].axis('off')
plt.subplots_adjust(hspace = 0.4)
```

IMPORTS AND SETUP

In [57]:
```
import sagemaker
import boto3

# creating a Sagemaker session
sagemaker_session = sagemaker.Session()

# defining the S3 bucket and prefix that we want to use in this session
bucket = 'sagemaker-practical'
prefix = 'traffic-sign-classifier'

role = sagemaker.get_execution_role()
print(role)
```

```
arn:aws:iam::058058168096:role/service-role/AmazonSageMaker-ExecutionRole-20230904T115723
```

DATA UPLOAD TO AWS S3

In [46]:
```
import os
os.makedirs("./data", exist_ok = True)
```

In [47]:
```
np.savez('./data/training', image = X_train, label = y_train)
np.savez('./data/validation', image = X_test, label = y_test)
```

In [48]:
```
# Uploading the training and validation data to S3 bucket

prefix = 'traffic-sign'

training_input_path   = sagemaker_session.upload_data('data/training.npz', key_prefix = prefix + '/training')
validation_input_path = sagemaker_session.upload_data('data/validation.npz', key_prefix = prefix + '/validation')

print(training_input_path)
print(validation_input_path)
```

```
s3://sagemaker-us-east-1-058058168096/traffic-sign/training/training.npz
s3://sagemaker-us-east-1-058058168096/traffic-sign/validation/validation.npz
```

CNN(Convolutional neural network) LENET MODEL TRAINNING USING SAGEMAKER

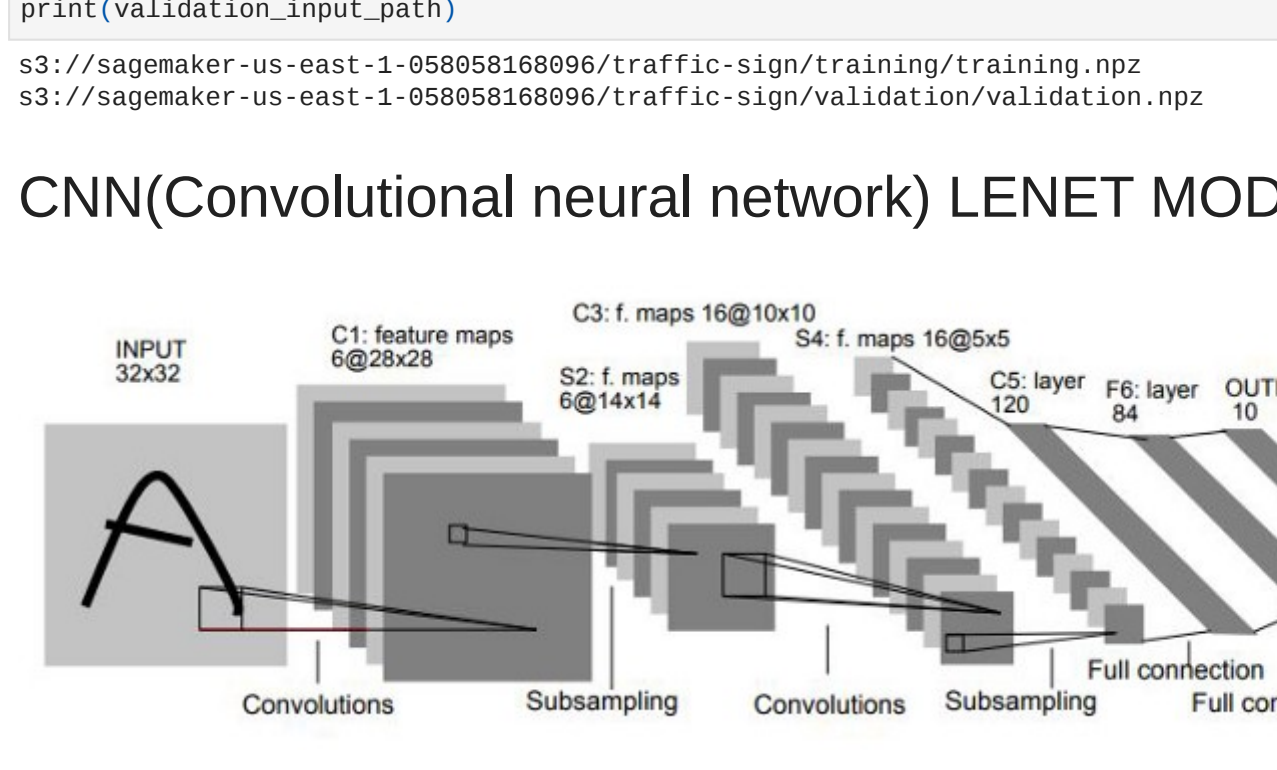


Figure 1: LeNet-5 - A Classic CNN Architecture (Source: DataScienceCentral, 2021)

The model consists of the following layers:

- THE FIRST CONVOLUTIONAL LAYER #1

```
- Input = 32x32x3
- Output = 28x28x6
- Output = (Input-filter+1)/Stride* => (32-5+1)/1=28
- Used a 5x5 Filter with input depth of 3 and output depth of 6
- Apply a RELU Activation function to the output
- pooling for input, Input = 28x28x6 and Output = 14x14x6


* Stride is the amount by which the kernel is shifted when the kernel is passed over the image.
```

- THE SECOND CONVOLUTIONAL LAYER #2

```
- Input = 14x14x6
- Output = 10x10x16
- Layer 2: Convolutional layer with Output = 10x10x16
- Output = (Input-filter+1)/strides => 10 = 14-5+1/1
- Apply a RELU Activation function to the output
- Pooling with Input = 10x10x16 and Output = 5x5x16
```

- FLATTENING THE NETWORK

```
- Flatten the network with Input = 5x5x16 and Output = 400
```

- FULLY CONNECTED LAYER

```
- Layer 3: Fully Connected layer with Input = 400 and Output = 120
- Apply a RELU Activation function to the output
```

- ANOTHER FULLY CONNECTED LAYER

```
- Layer 4: Fully Connected Layer with Input = 120 and Output = 84
- Apply a RELU Activation function to the output
```

- FULLY CONNECTED LAYER

```
- Layer 5: Fully Connected layer with Input = 84 and Output = 43
```

In [49]:
```
!pygmentize train-cnn.py
```

```
import argparse, os
import numpy as np
import tensorflow
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization, Conv2D, MaxPooling2D, AveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import multi_gpu_model


# The training code will be contained in a main gaurd (if __name__ == '__main__') so SageMaker will execute the code found in the main.
# argparse:
if __name__ == '__main__':

    # Parser to get the arguments
    parser = argparse.ArgumentParser()

    # Model hyperparameters are being sent as command-line arguments.
    parser.add_argument('--epochs', type=int, default=1)
    parser.add_argument('--learning-rate', type=float, default=0.001)
    parser.add_argument('--batch-size', type=int, default=32)



    # The script receives environment variables in the training container instance.
    # SM_NUM_GPUS: how many GPUs are available for trianing.
    # SM_MODEL_DIR: A string indicating output path where model artifcats will be sent out to.
    # SM_CHANNEL_TRAIN: path for the training channel
    # SM_CHANNEL_VALIDATION: path for the validation channel

    parser.add_argument('--gpu-count', type=int, default=os.environ['SM_NUM_GPUS'])
    parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
    parser.add_argument('--training', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
    parser.add_argument('--validation', type=str, default=os.environ['SM_CHANNEL_VALIDATION'])

    args, _ = parser.parse_known_args()

    # Hyperparameters
    epochs     = args.epochs
    lr         = args.learning_rate
    batch_size = args.batch_size
    gpu_count  = args.gpu_count
    model_dir  = args.model_dir
    training_dir   = args.training
    validation_dir = args.validation

    # Loading the training and validation data from s3 bucket
    train_images = np.load(os.path.join(training_dir, 'training.npz'))['image']
    train_labels = np.load(os.path.join(training_dir, 'training.npz'))['label']
    test_images  = np.load(os.path.join(validation_dir, 'validation.npz'))['image']
    test_labels  = np.load(os.path.join(validation_dir, 'validation.npz'))['label']

    K.set_image_data_format('channels_last')

    # Adding batch dimension to the input
    train_images = train_images.reshape(train_images.shape[0], 32, 32, 3)
    test_images = test_images.reshape(test_images.shape[0], 32, 32, 3)
    input_shape = (32, 32, 3)

    # Normalizing the data
    train_images = train_images.astype('float32')
    test_images = test_images.astype('float32')
    train_images /= 255
    test_images /= 255

    train_labels = tensorflow.keras.utils.to_categorical(train_labels, 43)
    test_labels = tensorflow.keras.utils.to_categorical(test_labels, 43)



    #LeNet Network Architecture

    model = Sequential()

    model.add(Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape= input_shape))

    model.add(AveragePooling2D())

    model.add(Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))

    model.add(AveragePooling2D())

    model.add(Flatten())

    model.add(Dense(units=120, activation='relu'))

    model.add(Dense(units=84, activation='relu'))

    model.add(Dense(units=43, activation = 'softmax'))

    print(model.summary())


    # If more than one GPU is available, convert the model to multi-gpu model
    if gpu_count > 1:

        model = multi_gpu_model(model, gpus=gpu_count)

    # Compile and train the model
    model.compile(loss=tensorflow.keras.losses.categorical_crossentropy,
                  optimizer=Adam(lr=lr),
                  metrics=['accuracy'])

    model.fit(train_images, train_labels, batch_size=batch_size,
                  validation_data=(test_images, test_labels),
                  epochs=epochs,
                  verbose=2)

    # Evaluating the model
    score = model.evaluate(test_images, test_labels, verbose=0)
    print('Validation loss    :', score[0])
    print('Validation accuracy:', score[1])

    # save trained CNN Keras model to "model_dir" (path specificied earlier)
    sess = K.get_session()
    tensorflow.saved_model.simple_save(
        sess,
        os.path.join(model_dir, 'model/1'),
        inputs={'inputs': model.input},
        outputs={t.name: t for t in model.outputs})
```

In [50]:
```
from sagemaker.tensorflow import TensorFlow

# To Train a TensorFlow model, we will use TensorFlow estimator from the Sagemaker SDK

# entry_point: a script that will run in a container. This script will include model description and training.
# role: a role that's obtained The role assigned to the running notebook.
# train_instance_count: number of container instances used to train the model.
# train_instance_type: instance type!
# framwork_version: version of Tensorflow
# py_version: Python version.
# script_mode: allows for running script in the container.
# hyperparameters: indicate the hyperparameters for the training job such as epochs and learning rate


tf_estimator = TensorFlow(entry_point='train-cnn.py',
                          role=role,
                          train_instance_count=1,
                          train_instance_type='ml.c4.2xlarge',
                          framework_version='1.12',
                          py_version='py3',
                          script_mode=True,
                          hyperparameters={
                              'epochs': 2 ,
                              'batch-size': 32,
                              'learning-rate': 0.001}
                         )
```

```
WARNING:sagemaker.deprecations:train_instance_type has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:train_instance_count has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:train_instance_type has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

In [51]:
```
tf_estimator.fit({'training': training_input_path, 'validation': validation_input_path})
```

```
INFO:sagemaker.image_uris:image_uri is not presented, retrieving image_uri based on instance_type, framework etc.
INFO:sagemaker.image_uris:image_uri is not presented, retrieving image_uri based on instance_type, framework etc.
INFO:sagemaker:Creating training-job with name: sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706
Using provided s3_resource
2023-09-13 18:26:22 Starting - Starting the training job...
2023-09-13 18:26:48 Starting - Preparing the instances for training......
2023-09-13 18:27:45 Downloading - Downloading input data...
2023-09-13 18:28:16 Training - Training image download completed. Training in progress..2023-09-13 18:28:20,518 sagemaker-containers INFO     Imported framewo
rk sagemaker_tensorflow_container.training
2023-09-13 18:28:20,523 sagemaker-containers INFO     No GPUs detected (normal if no gpus installed)
2023-09-13 18:28:20,727 sagemaker-containers INFO     No GPUs detected (normal if no gpus installed)
2023-09-13 18:28:20,742 sagemaker-containers INFO     No GPUs detected (normal if no gpus installed)
2023-09-13 18:28:20,754 sagemaker-containers INFO     Invoking user script
Training Env:
```



```
{
    "additional_framework_parameters": {},
    "channel_input_dirs": {
        "training": "/opt/ml/input/data/training",
        "validation": "/opt/ml/input/data/validation"
    },
    "current_host": "algo-1",
    "framework_module": "sagemaker_tensorflow_container.training:main",
    "hosts": [
        "algo-1"
    ],
    "hyperparameters": {
        "batch-size": 32,
        "epochs": 2,
        "learning-rate": 0.001,
        "model_dir": "s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706/model"
    },
    "input_config_dir": "/opt/ml/input/config",
    "input_data_config": {
        "training": {
            "TrainingInputMode": "File",
            "S3DistributionType": "FullyReplicated",
            "RecordWrapperType": "None"
        },
        "validation": {
            "TrainingInputMode": "File",
            "S3DistributionType": "FullyReplicated",
            "RecordWrapperType": "None"
        }
    },
    "input_dir": "/opt/ml/input",
    "is_master": true,
    "job_name": "sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706",
    "log_level": 20,
    "master_hostname": "algo-1",
    "model_dir": "/opt/ml/model",
    "module_dir": "s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706/source/sourcedir.tar.gz",
    "module_name": "train-cnn",
    "network_interface_name": "eth0",
    "num_cpus": 8,
    "num_gpus": 0,
    "output_data_dir": "/opt/ml/output/data",
    "output_dir": "/opt/ml/output",
    "output_intermediate_dir": "/opt/ml/output/intermediate",
    "resource_config": {
        "current_host": "algo-1",
        "current_instance_type": "ml.c4.2xlarge",
        "current_group_name": "homogeneousCluster",
        "hosts": [
            "algo-1"
        ],
        "instance_groups": [
            {
                "instance_group_name": "homogeneousCluster",
                "instance_type": "ml.c4.2xlarge",
                "hosts": [
                    "algo-1"
                ]
            }
        ],
        "network_interface_name": "eth0"
    },
    "user_entry_point": "train-cnn.py"
}
Environment variables:
SM_HOSTS=["algo-1"]
SM_NETWORK_INTERFACE_NAME=eth0
SM_HPS={"batch-size":32,"epochs":2,"learning-rate":0.001,"model_dir":"s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-2
1-706/model"}
SM_USER_ENTRY_POINT=train-cnn.py
SM_FRAMEWORK_PARAMS={}
SM_RESOURCE_CONFIG={"current_group_name":"homogeneousCluster","current_host":"algo-1","current_instance_type":"ml.c4.2xlarge","hosts":["algo-1"],"instance_gro
ups":[{"hosts":["algo-1"],"instance_group_name":"homogeneousCluster","instance_type":"ml.c4.2xlarge"}],"network_interface_name":"eth0"}
SM_INPUT_DATA_CONFIG={"training":{"RecordWrapperType":"None","S3DistributionType":"FullyReplicated","TrainingInputMode":"File"},"validation":{"RecordWrapperTy
pe":"None","S3DistributionType":"FullyReplicated","TrainingInputMode":"File"}}
SM_OUTPUT_DATA_DIR=/opt/ml/output/data
SM_CHANNELS=["training","validation"]
SM_CURRENT_HOST=algo-1
SM_MODULE_NAME=train-cnn
SM_LOG_LEVEL=20
SM_FRAMEWORK_MODULE=sagemaker_tensorflow_container.training:main
SM_INPUT_DIR=/opt/ml/input
SM_INPUT_CONFIG_DIR=/opt/ml/input/config
SM_OUTPUT_DIR=/opt/ml/output
SM_NUM_CPUS=8
SM_NUM_GPUS=0
SM_MODEL_DIR=/opt/ml/model
SM_MODULE_DIR=s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706/source/sourcedir.tar.gz
SM_TRAINING_ENV={"additional_framework_parameters":{},"channel_input_dirs":{"training":"/opt/ml/input/data/training","validation":"/opt/ml/input/data/validati
on"},"current_host":"algo-1","framework_module":"sagemaker_tensorflow_container.training:main","hosts":["algo-1"],"hyperparameters":{"batch-size":32,"epochs":
2,"learning-rate":0.001,"model_dir":"s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706/model"},"input_config_di
r":"/opt/ml/input/config","input_data_config":{"training":{"RecordWrapperType":"None","S3DistributionType":"FullyReplicated","TrainingInputMode":"File"},"vali
dation":{"RecordWrapperType":"None","S3DistributionType":"FullyReplicated","TrainingInputMode":"File"}},"input_dir":"/opt/ml/input","is_master":true,"job_nam
e":"sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706","log_level":20,"master_hostname":"algo-1","model_dir":"/opt/ml/model","module_dir":"s3://sagemake
r-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706/source/sourcedir.tar.gz","module_name":"train-cnn","network_interface_nam
e":"eth0","num_cpus":8,"num_gpus":0,"output_data_dir":"/opt/ml/output/data","output_dir":"/opt/ml/output","output_intermediate_dir":"/opt/ml/output/intermedia
te","resource_config":{"current_group_name":"homogeneousCluster","current_host":"algo-1","current_instance_type":"ml.c4.2xlarge","hosts":["algo-1"],"instance_
groups":[{"hosts":["algo-1"],"instance_group_name":"homogeneousCluster","instance_type":"ml.c4.2xlarge"}],"network_interface_name":"eth0"},"user_entry_poin
t":"train-cnn.py"}
SM_USER_ARGS=["--batch-size","32","--epochs","2","--learning-rate","0.001","--model_dir","s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmod
e-2023-09-13-18-26-21-706/model"]
SM_OUTPUT_INTERMEDIATE_DIR=/opt/ml/output/intermediate
SM_CHANNEL_TRAINING=/opt/ml/input/data/training
SM_CHANNEL_VALIDATION=/opt/ml/input/data/validation
SM_HP_BATCH-SIZE=32
SM_HP_EPOCHS=2
SM_HP_LEARNING-RATE=0.001
SM_HP_MODEL_DIR=s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmode-2023-09-13-18-26-21-706/model
PYTHONPATH=/opt/ml/code:/usr/local/bin:/usr/lib/python36.zip:/usr/lib/python3.6:/usr/lib/python3.6/lib-dynload:/usr/local/lib/python3.6/dist-packages:/usr/li
b/python3/dist-packages
```



```
Invoking script with the following command:
/usr/bin/python train-cnn.py --batch-size 32 --epochs 2 --learning-rate 0.001 --model_dir s3://sagemaker-us-east-1-058058168096/sagemaker-tensorflow-scriptmod
e-2023-09-13-18-26-21-706/model
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         456
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 43)                3655
=================================================================
Total params: 64,811
Trainable params: 64,811
Non-trainable params: 0
_________________________________________________________________
None
Train on 34799 samples, validate on 12630 samples
Epoch 1/2
 - 12s - loss: 1.3732 - acc: 0.6128 - val_loss: 0.9343 - val_acc: 0.7493
Epoch 2/2

2023-09-13 18:28:51 Uploading - Uploading generated training model - 11s - loss: 0.3730 - acc: 0.8923 - val_loss: 0.7195 - val_acc: 0.8317
Validation loss    : 0.71949563081172
Validation accuracy: 0.8316706255326078
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/saved_model/simple_save.py:85: calling SavedModelBuilder.add_meta_graph_and_v
ariables (from tensorflow.python.saved_model.builder_impl) with legacy_init_op is deprecated and will be removed in a future version.
Instructions for updating:
Pass your op to the equivalent parameter main_op instead.
2023-09-13 18:28:48,323 sagemaker-containers INFO     Reporting training SUCCESS

2023-09-13 18:29:02 Completed - Training job completed
Training seconds: 77
Billable seconds: 77
```

MODEL DEPLOYMENT

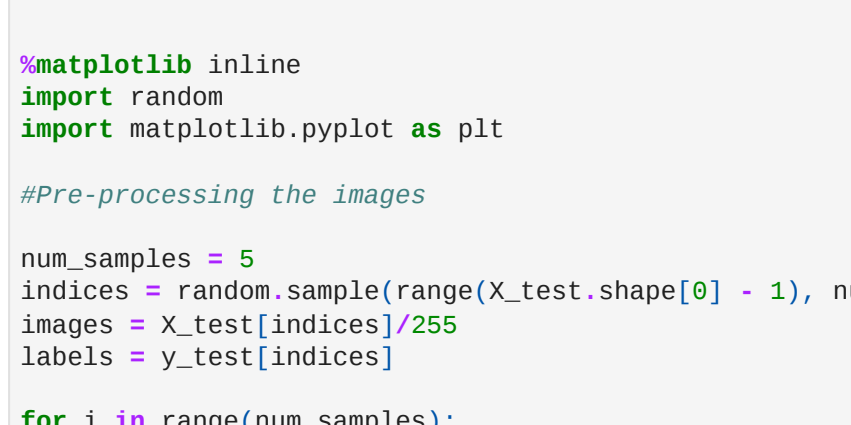
In [53]:
```
# Deploying the model

import time

tf_endpoint_name = 'trafficsignclassifier-' + time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())

tf_predictor = tf_estimator.deploy(initial_instance_count = 1,
                          instance_type = 'ml.t2.medium',
                          endpoint_name = tf_endpoint_name)
```

```
INFO:sagemaker.tensorflow.model:image_uri is not presented, retrieving image_uri based on instance_type, framework etc.
INFO:sagemaker:Creating model with name: sagemaker-tensorflow-scriptmode-2023-09-13-18-33-58-190
INFO:sagemaker:Creating endpoint-config with name trafficsignclassifier-2023-09-13-18-33-58
INFO:sagemaker:Creating endpoint with name trafficsignclassifier-2023-09-13-18-33-58
------!
```

MAKING PREDICTIONS

In [54]:
```
# Making predictions from the end point


%matplotlib inline
import random
import matplotlib.pyplot as plt

#Pre-processing the images

num_samples = 5
indices = random.sample(range(X_test.shape[0] - 1), num_samples)
images = X_test[indices]/255
labels = y_test[indices]

for i in range(num_samples):
    plt.subplot(1,num_samples,i+1)
    plt.imshow(images[i])
    plt.title(labels[i])
    plt.axis('off')

# Making predictions

prediction = tf_predictor.predict(images.reshape(num_samples, 32, 32, 3))['predictions']
prediction = np.array(prediction)
predicted_label = prediction.argmax(axis=1)
print('Predicted labels are: {}'.format(predicted_label))
```

```
Predicted labels are: [ 4  5 33 12  4]
```



In [55]:
```
# Deleting the end-point
tf_predictor.delete_endpoint()
```

```
INFO:sagemaker:Deleting endpoint configuration with name: trafficsignclassifier-2023-09-13-18-33-58
INFO:sagemaker:Deleting endpoint with name: trafficsignclassifier-2023-09-13-18-33-58
```

DISCUSSION

The results obtained from the traffic sign classification project using the LeNet architecture and the German Traffic Sign Recognition Benchmark dataset demonstrate the feasibility and effectiveness of deep learning in solving real-world computer vision challenges. In this section, I delve into the key findings, interpret the results, and discuss the implications and limitations of the project.

Interpreting the Results:

Our model achieved a commendable validation accuracy of approximately 83.17%, a notable accomplishment considering the complexity and diversity of traffic sign images. The validation loss, which reached 0.7195, corroborates the model's ability to generalize well to previously unseen data. These metrics affirm that the deep learning approach successfully learned meaningful features and patterns within the traffic sign images.

Implications and Significance:

The project's significance extends beyond traffic sign classification for autonomous vehicles. It shows the broader applicability of deep learning in computer vision tasks, where convolutional neural networks (CNNs) excel at feature extraction and image classification. The success of the model highlights the adaptability and robustness of the LeNet architecture for complex, multiclass classification problems.

Furthermore, This project provides practical insights into the importance of various architectural components within deep learning. Convolutional layers enable the model to recognize local patterns and spatial relationships in traffic signs. Rectified linear units (ReLU) introduce non-linearity and sparsity, improving the model's ability to capture complex features. Pooling layers help reduce dimensionality and computational complexity while preserving critical information. Dropout layers mitigate overfitting, enhancing the model's generalization.

Limitations and Future Directions:

Despite the promising results, This project has certain limitations. The LeNet architecture, while effective, may not fully harness the potential of more modern architectures, such as deeper convolutional networks or more complex models like ResNet or Inception. Exploring advanced architectures and transfer learning could further enhance performance.

CONCLUSION AND RECOMMENDATIONS

In conclusion, the deep learning-based traffic sign classification project using the LeNet architecture demonstrates the capacity of neural networks to tackle complex multiclass image classification tasks. With an impressive validation accuracy of 83.17%, our model showcases its ability to recognize diverse traffic sign classes accurately.

We recommend the following for future research and applications:

Exploration of Advanced Architectures: Investigate more sophisticated neural network architectures to further boost accuracy and robustness.

Real-World Testing: Extend the research to include real-world testing scenarios with considerations for adverse weather conditions and occlusions.

Deployment in Autonomous Vehicles: Integrate the trained model into autonomous vehicles or intelligent transportation systems to assess its real-world performance and enhance road safety.

Continuous Learning: As new traffic signs and regulations emerge, the model should undergo continuous learning and updates to stay current.

Generalization to Other Domains: Apply the knowledge and insights gained from this project to other computer vision tasks, contributing to advancements in the broader field of artificial intelligence.

In summary, this project serves as a testament to the capabilities of deep learning in addressing complex, real-world challenges. It provides valuable insights for both the traffic sign recognition domain and the broader deep learning community, offering a foundation for further research and development in intelligent transportation systems and computer vision applications.

REFERENCES

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition.

Palak, & Sangal, A. L. (2021). Traffic Signs Classification using Convolutional Neural Networks: A review.

Kolachalama, V. N. S. (2021). Traffic Sign Classification Using CNN.

Youssouf, N. (2022, December). Traffic sign classification using CNN and detection using faster-RCNN and YOLOV4.

Akshata, V. S., & Panda, S. (2019). Traffic sign recognition and classification using convolutional neural networks. M.Tech(CSE) Department of Computer Science Information Technology, Jain University, Jayanagar, Bangalore.

Shustanov, A., & Yakimov, P. (2017). CNN Design for Real-Time Traffic Sign Recognition.

Kaggle. (2022). German Traffic Sign Recognition Benchmark. Retrieved from https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign