Element 3: Traffic Sign Detection using Amazon SageMaker

Abstract
In today's world, almost everything we do has been simplified by automated tasks. In an attempt to focus on the road while driving, drivers often miss out on signs on the side of the road, which could be dangerous for them and for the people around them. This problem can be avoided if there was an efficient way to notify the driver without having them to shift their focus.
Traffic Sign Detection plays an important role here by detecting a sign, thus notifying the driver of any upcoming signs. This not only ensures road safety, but also allows the driver to be at little more ease while driving on tricky or new roads Traffic signs are utilized as a method of warning and guiding drivers, helping to regulate the flow of traffic among vehicles, pedestrians, motorcycles, bicycles and others who travel the streets, highways and other roadways effectively.

In this detection we have used the Object Detection Algorithm of Amazon Sagemaker to detect the traffic sign board by first labelling the dataset and training it using the AWS Management Console

We have achieved higher than 90% recognition accuracies for signs trained on the data sets.

Team Members:

Name	Roll No.	Email
Aradhita Menghal	04	menghalar@rknec.edu (mailto:menghalar@rknec.edu)
Urvi Negi	16	negiu@rknec.edu (mailto:negiu@rknec.edu)
Akshat Chandak	35	chandaka@rknec.edu (mailto:chandaka@rknec.edu)
Shreyas Nemani	77	nemanism@rknec.edu (mailto:nemanism@rknec.edu)
Shubham Jha	78	jhasm@rknec.edu (mailto:jhasm@rknec.edu)

Amazon SageMaker is a fully managed machine learning service. With SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers.
It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it

Defining S3 Bucket and Location of data

In [11]:
```
1 PREFIX = 'input'
2 BUCKET = 'project-sagemaker-virginia'
```

Importing Libraries

In [12]:
```
!pip -q install --upgrade pip
!pip -q install jsonlines
import jsonlines
import boto3
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import numpy as np
import json
from itertools import cycle
```

Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python, Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

Boto3: Boto3 is the name of the Python SDK for AWS. It allows you to directly create, update, and delete AWS resources from your Python scripts.

Jsonlines: jsonlines is a Python library to simplify working with jsonlines and ndjson data.

PIL: The Python Imaging Library adds image processing capabilities to your Python interpreter, This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Setting up client using boto3

In [ ]:
```
client = boto3.client('sagemaker')
```

Labeling with SageMaker Ground Truth

Amazon SageMaker Ground Truth is a fully managed data labeling service that makes it easy to build highly accurate training datasets for machine learning In addition, Ground Truth offers automatic data labeling which uses a machine learning model to label your data.

We have labelled the whole dataset of 761 images by using Amazon Sagemaker Groundtruth through the console

In [ ]:
```
labeling_job_name = 'sign-labelling-job'
```

Training an Object Detection Model

We are now ready to use the labeled dataset in order to train a Machine Learning model using the SageMaker built-in Object Detection algorithm (https://docs.aws.amazon.com/sagemaker/latest/dg/object-detection.html).

For this, we would need to split the full labeled dataset into a training and a validation datasets. Out of the total of 761 images we are going to use 609 for training and 152 for validation. The algorithm will use the first one to train the model and the latter to estimate the accuracy of the model, trained so far. The augmented manifest file from the full labeling job was included as `output.manifest` .

In [13]:
```
with jsonlines.open('output.manifest', 'r') as reader:
    lines = list(reader)
    # Shuffle data in place.
    np.random.shuffle(lines)

dataset_size = len(lines)
num_training_samples = round(dataset_size*0.8)

train_data = lines[:num_training_samples]
validation_data = lines[num_training_samples:]

augmented_manifest_filename_train = 'train.manifest'

with open(augmented_manifest_filename_train, 'w') as f:
    for line in train_data:
        f.write(json.dumps(line))
        f.write('\n')

augmented_manifest_filename_validation = 'validation.manifest'

with open(augmented_manifest_filename_validation, 'w') as f:
    for line in validation_data:
        f.write(json.dumps(line))
        f.write('\n')

print(f'training samples: {num_training_samples}, validation samples: {len(lines)-num_training_samples}')
```

```
training samples: 609, validation samples: 152
```

Next, let's upload the two manifest files to S3 in preparation for training. We will use the same bucket you created earlier.

In [14]:
```
pfx_training = PREFIX + '/training' if PREFIX else 'training'
# Defines paths for use in the training job request.
s3_train_data_path = 's3://{}/{}/{}'.format(BUCKET, pfx_training, augmented_manifest_filename_train)
s3_validation_data_path = 's3://{}/{}/{}'.format(BUCKET, pfx_training, augmented_manifest_filename_validation)

!aws s3 cp train.manifest s3://$BUCKET/$pfx_training/
!aws s3 cp validation.manifest s3://$BUCKET/$pfx_training/
```

```
upload: ./train.manifest to s3://project-sagemaker-virginia/input/training/train.manifest
upload: ./validation.manifest to s3://project-sagemaker-virginia/input/training/validation.manifest
```

We are now ready to kick off the training. We will do it from the SageMaker console, but alternatively, you can just run this code in a new cell using SageMaker Python SDK:

Code option

If you do not want to use the AWS Console to create the training job the below code could do it for you, We have done it using AWS Management Console

In [ ]:

```
import time
import sagemaker

role = sagemaker.get_execution_role()
sess = sagemaker.Session()

training_image = sagemaker.amazon.amazon_estimator.get_image_uri(
    boto3.Session().region_name, 'object-detection', repo_version='latest')
s3_output_path = 's3://{}/{}/output'.format(BUCKET, pfx_training)

# Create unique job name
training_job_name = 'sign-labelling-job'

training_params = \
    {
        "AlgorithmSpecification": {

            "TrainingImage": training_image,
            "TrainingInputMode": "Pipe"
        },
        "RoleArn": role,
        "OutputDataConfig": {
            "S3OutputPath": s3_output_path
        },
        "ResourceConfig": {
            "InstanceCount": 1,
            "InstanceType": "ml.p2.xlarge",
            "VolumeSizeInGB": 50
        },
        "TrainingJobName": training_job_name,
        "HyperParameters": {
            "base_network": "resnet-50",
            "use_pretrained_model": "1",
            "num_classes": "1",
            "mini_batch_size": "1",
            "epochs": "100",
            "learning_rate": "0.001",
            "lr_scheduler_step": "",
            "lr_scheduler_factor": "0.1",
            "optimizer": "sgd",
            "momentum": "0.9",
```

```
            "weight_decay": "0.0005",
            "overlap_threshold": "0.5",
            "nms_threshold": "0.45",
            "image_shape": "300",
            "label_width": "350",
            "num_training_samples": str(num_training_samples)
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": 86400
        },
        "InputDataConfig": [
            {
                "ChannelName": "train",
                "DataSource": {
                    "S3DataSource": {
                        "S3DataType": "AugmentedManifestFile",  #  Augmented Manifest
                        "S3Uri": s3_train_data_path,
                        "S3DataDistributionType": "FullyReplicated",
                        #  This must correspond to the JSON field names in your augmented manifest.
                        "AttributeNames": ['source-ref', 'sign-labelling-job']
                    }
                },
                "ContentType": "application/x-recordio",
                "RecordWrapperType": "RecordIO",
                "CompressionType": "None"
            },
            {
                "ChannelName": "validation",
                "DataSource": {
                    "S3DataSource": {
                        "S3DataType": "AugmentedManifestFile",  #  Augmented Manifest
                        "S3Uri": s3_validation_data_path,
                        "S3DataDistributionType": "FullyReplicated",
                        #  This must correspond to the JSON field names in your augmented manifest.
                        "AttributeNames": ['source-ref', 'sign-labelling-job']
                    }
                },
                "ContentType": "application/x-recordio",
                "RecordWrapperType": "RecordIO",
                "CompressionType": "None"
            }
        ]
```

```
    }

# Now we create the SageMaker training job.
client = boto3.client(service_name='sagemaker')
client.create_training_job(**training_params)

# Confirm that the training job has started
status = client.describe_training_job(TrainingJobName=training_job_name)['TrainingJobStatus']
print('Training job current status: {}'.format(status))
```

To check the progess of the training job, you can refresh the console or repeatedly evaluate the following cell. When the training job status reads `'Completed'`, we move on to the next part of the process

In [17]:
```
training_job_name = 'sign-training-job'
training_info = client.describe_training_job(TrainingJobName=training_job_name)
print("Training job status: ", training_info['TrainingJobStatus'])
print("Secondary status: ", training_info['SecondaryStatus'])
```

```
Training job status:  Completed
Secondary status:  Completed
```

Review of Training Results

First, let's create the SageMaker model out of model artifacts

In [18]:

```
import time
timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
model_name = training_job_name + '-model' + timestamp

training_image = training_info['AlgorithmSpecification']['TrainingImage']
model_data = training_info['ModelArtifacts']['S3ModelArtifacts']

primary_container = {
    'Image': training_image,
    'ModelDataUrl': model_data,
}

from sagemaker import get_execution_role

role = get_execution_role()

create_model_response = client.create_model(
    ModelName = model_name,
    ExecutionRoleArn = role,
    PrimaryContainer = primary_container)

print(create_model_response['ModelArn'])
```

```
arn:aws:sagemaker:us-east-1:080900144494:model/sign-training-job-model-2021-04-24-06-56-44
```

In [19]:
```
timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
endpoint_config_name = training_job_name + '-epc' + timestamp
endpoint_config_response = client.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.t2.medium',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])

print('Endpoint configuration name: {}'.format(endpoint_config_name))
print('Endpoint configuration arn:  {}'.format(endpoint_config_response['EndpointConfigArn']))
```

```
Endpoint configuration name: sign-training-job-epc-2021-04-24-06-56-53
Endpoint configuration arn:  arn:aws:sagemaker:us-east-1:080900144494:endpoint-config/sign-training-job-epc-2021-04-24-06-56-53
```

Create Endpoint

The next cell creates an endpoint that can be validated and incorporated into production applications. This takes about 10 minutes to complete.

In [20]:
```
timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
endpoint_name = training_job_name + '-ep' + timestamp
print('Endpoint name: {}'.format(endpoint_name))

endpoint_params = {
    'EndpointName': endpoint_name,
    'EndpointConfigName': endpoint_config_name,
}
endpoint_response = client.create_endpoint(**endpoint_params)
print('EndpointArn = {}'.format(endpoint_response['EndpointArn']))
```

```
Endpoint name: sign-training-job-ep-2021-04-24-06-57-12
EndpointArn = arn:aws:sagemaker:us-east-1:080900144494:endpoint/sign-training-job-ep-2021-04-24-06-57-12
```

In [25]:
```
endpoint_name="sign-training-job-ep-2021-04-24-06-57-12"
# get the status of the endpoint
response = client.describe_endpoint(EndpointName=endpoint_name)
status = response['EndpointStatus']
print('EndpointStatus = {}'.format(status))
```

```
EndpointStatus = InService
```

Perform inference

We will invoke the deployed endpoint to detect traffic signs in the 10 test images that were inside the `test` folder

In [41]:
```
import glob
test_images = glob.glob('test/*')
print(*test_images, sep="\n")
```

```
test/00859.jpg
test/wsi-imageoptim-French-Road-Signs-No-right-turn-813x730.jpg
test/1.jpg
test/00797.jpg
test/00023.jpg
test/00659.jpg
test/00480.jpg
test/00446.jpg
```

In [42]:
```
def prediction_to_bbox_data(image_path, prediction):
    class_id, confidence, xmin, ymin, xmax, ymax = prediction
    width, height = Image.open(image_path).size
    bbox_data = {'class_id': class_id,
                 'height': (ymax-ymin)*height,
                 'width': (xmax-xmin)*width,
                 'left': xmin*width,
                 'top': ymin*height}
    return bbox_data
```

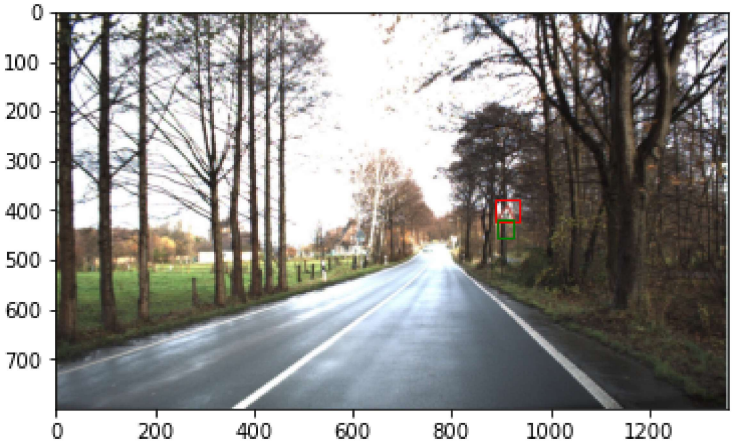
Finally, for each of the test images, the following cell transforms the image into the appropriate format for realtime prediction, repeatedly calls the endpoint, receives back the prediction, and displays the result.

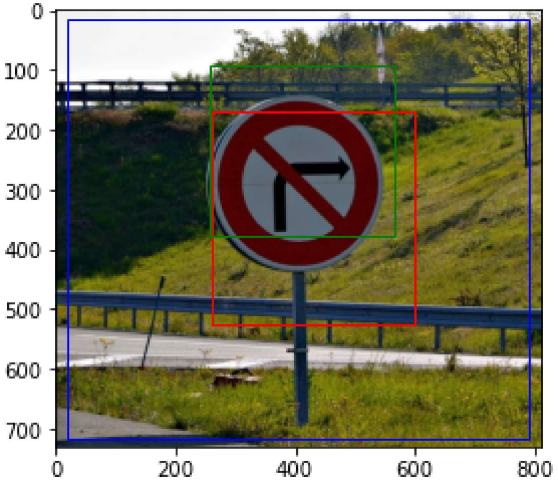
In [43]:

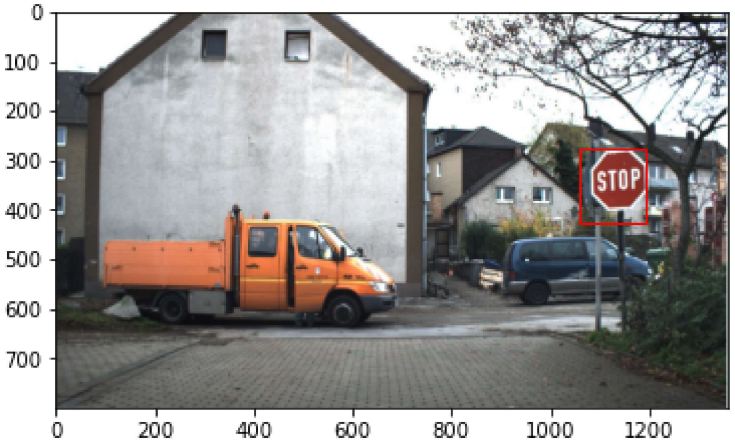
```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import numpy as np
from itertools import cycle

def show_annotated_image(img_path, bboxes):
    im = np.array(Image.open(img_path), dtype=np.uint8)

    # Create figure and axes
    fig,ax = plt.subplots(1)

    # Display the image
    ax.imshow(im)

    colors = cycle(['r', 'g', 'b', 'y', 'c', 'm', 'k', 'w'])

    for bbox in bboxes:
        # Create a Rectangle patch
        rect = patches.Rectangle((bbox['left'],bbox['top']),bbox['width'],bbox['height'],linewidth=1,edgecolor=next(

        # Add the patch to the Axes
        ax.add_patch(rect)

    plt.show()
```

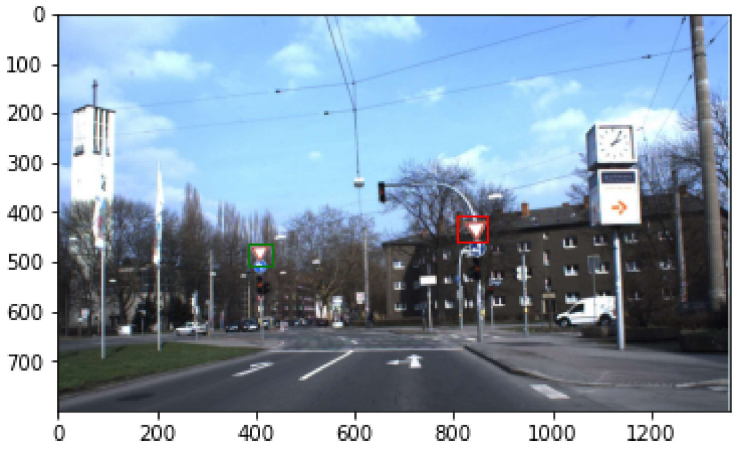
In [44]:

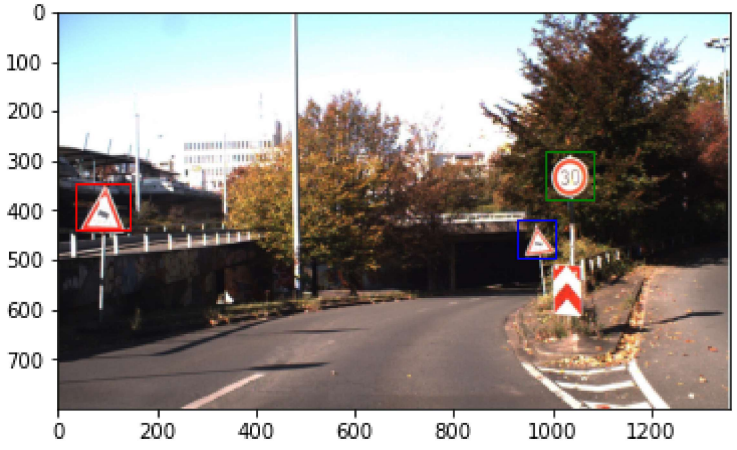
```
import matplotlib.pyplot as plt

runtime_client = boto3.client('sagemaker-runtime')

# Call SageMaker endpoint to obtain predictions
def get_predictions_for_img(runtime_client, endpoint_name, img_path):
    with open(img_path, 'rb') as f:
        payload = f.read()
        payload = bytearray(payload)

    response = runtime_client.invoke_endpoint(EndpointName=endpoint_name,
                                      ContentType='application/x-image',
                                      Body=payload)

    result = response['Body'].read()
    result = json.loads(result)
    return result


# wait until the status has changed
client.get_waiter('endpoint_in_service').wait(EndpointName=endpoint_name)
endpoint_response = client.describe_endpoint(EndpointName=endpoint_name)
status = endpoint_response['EndpointStatus']
if status != 'InService':
    raise Exception('Endpoint creation failed.')

for test_image in test_images:
    result = get_predictions_for_img(runtime_client, endpoint_name, test_image)
    confidence_threshold = .2
    best_n = 3
    # display the best n predictions with confidence > confidence_threshold
    predictions = [prediction for prediction in result['prediction'] if prediction[1] > confidence_threshold]
    predictions.sort(reverse=True, key = lambda x: x[1])
    bboxes = [prediction_to_bbox_data(test_image, prediction) for prediction in predictions[:best_n]]
    show_annotated_image(test_image, bboxes)
```

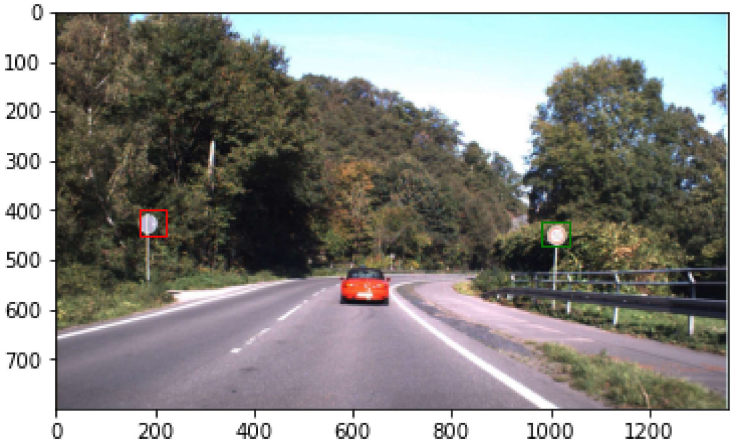

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

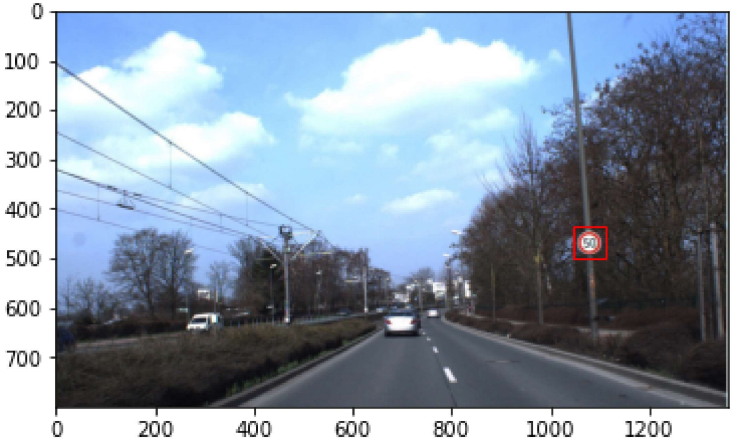

0
100
200
300
400
500
600
700
0
200
400
600
800

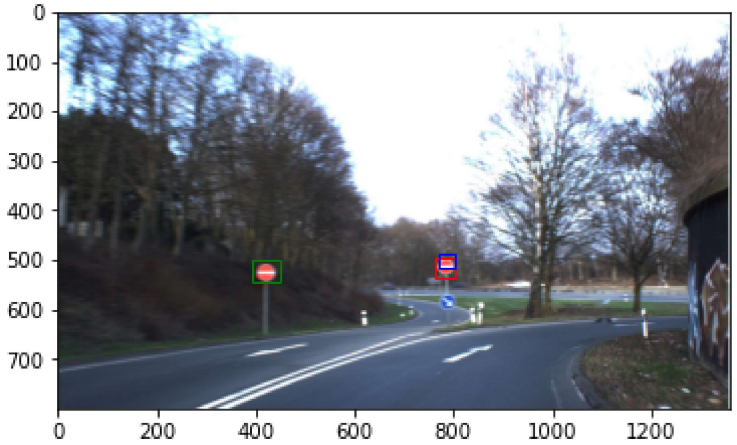

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
STOP


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Cleanup

At the end of the lab we would like to delete the real-time endpoint, as keeping a real-time endpoint around while being idle is costly and wasteful.

In [ ]:
```
client.delete_endpoint(EndpointName=endpoint_name)
```