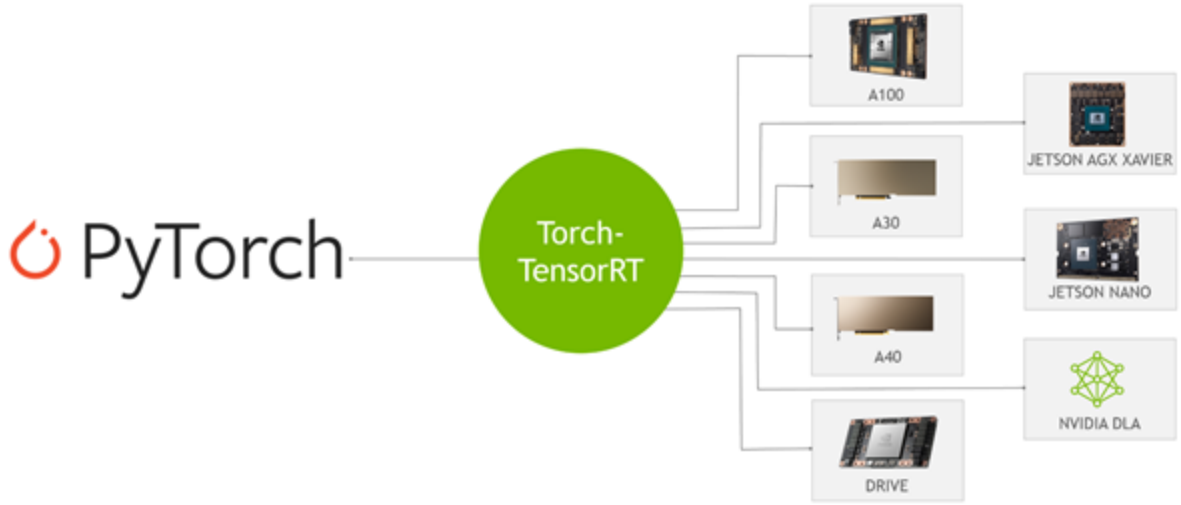
TensorRT

Torch-TensorRT



- **PyTorch/TorchScript/FX용 컴파일러**
  - PyTorch와 NIVIDIA의 TensorRT를 통합하여 딥러닝 모델 최적화 및 가속화하는 데 사용
- **목적**

    - NVIDIA GPU에서 모델을 더 빠르게 실행하기 위한 최적화된 런타임 엔진으로, 특히 딥러닝 모델을 배포 환경에서 더 효율적으로 실행하고 추론(inference) 성능을 향상시키는 데 사용

- **장점**
    - **모델 최적화**: Torch-TensorRT는 PyTorch 모델을 TensorRT 호환 형식으로 변환하여 TensorRT의 최적화 기능을 활용할 수 있음. 이로써 모델의 연산 그래프가 최적화되고, GPU 가속을 위한 특정 연산 커널로 대체됨.
    - **TensorRT 모델 생성**: Torch-TensorRT를 사용하여 최적화된 모델을 생성할 수 있음. 이렇게 생성된 모델은 TensorRT 엔진을 사용하여 효율적으로 추론을 수행할 수 있음.
    - **End-to-end 가속화**: Torch-TensorRT를 사용하면 PyTorch 모델을 가져와 TensorRT 모델로 변환하고 추론을 수행하는 전체 엔드 투 엔드 파이프라인을 구축할 수 있음.
    - **동적 그래프 지원**: PyTorch의 동적 그래프 특성을 유지하면서 TensorRT로의 최적화가 가능하므로, 다양한 모델 아키텍처에 대한 지원을 제공합니다.

YoloV5 with TensorRT

- **참고한 youtube 링크:** TensorRT for Beginners: A Tutorial on Deep Learning Inference Optimization

라이브러리 다운로드

```
!pip install torch torch-tensorrt tensorrt --extra-index-url htt
!pip install opencv-python
!pip install numpy
!pip install pandas
!pip install matplotlib
```

YoloV5 Pretrained 모델 불러오기

```
import torch
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrai

# 테스트 진행할 경우 GPU 사용
device = torch.device('cuda' if torch.cuda.is_available() else
model.to(device)
print(device)
```

COCO 2017 데이터셋 다운로드

```
# COCO 데이터셋 다운로드
!wget http://images.cocodataset.org/zips/val2017.zip
!unzip val2017.zip
```

이미지 한 장 불러오는 함수

```
import os
import cv2

def prepare_images(image_paths):

    images = []
    for image_path in image_paths:
      image = cv2.imread(image_path)

      # 원하는 이미지로 변형
      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
      image = cv2.resize(image, (640, 640))

      image = image.transpose((2, 0, 1)) # W, H, C => C, H, W
      image = torch.from_numpy(image).to(device)
      image = image.float() / 255.0 # Normalize [0;1]
      images.append(image)
```

```
    return images
```

COCO 데이터셋 불러오기

```
# COCO 데이터셋 로드
image_dir = 'val2017'
image_paths = [os.path.join(image_dir, img) for img in os.listdi

# COCO의 일부분만 사용
IMAGE_NUMS = 128
images = prepare_images(image_paths[:IMAGE_NUMS])
```

테스트

```
import time

def run_inference(model, images):
  times = []
  results = []
  for image in images:
    # 이미지를 GPU로 옮김
    image = image.unsqueeze(0) # (1,3,640,640)
    image_gpu = image.to('cuda')

    # Gradient 계산이 없는 모델 테스트
    start = time.time()
    with torch.no_grad():
        result = model(image_gpu)

    results.append(result)
    times.append(time.time() - start)
```

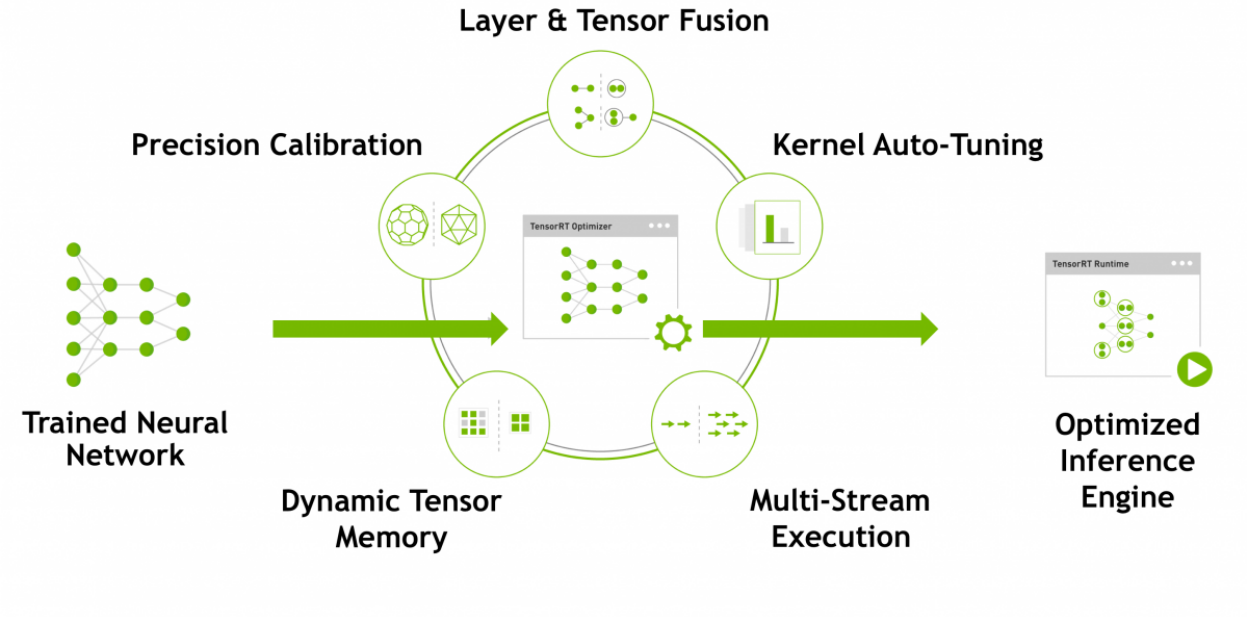
```
  return results, times
```

```
NUM_TRIALS = 10
avgs = []

for i in range(NUM_TRIALS):
    results, times = run_inference(model, images)
    avg = sum(times)/len(times)
    print("Average inference time (s):", avg)
    avgs.append(avg)

mat = sum(avgs)/NUM_TRIALS
print("Mean average inference time (s):", mat)
```

TensorRT를 적용하여 최적화 진행




```
import torch_tensorrt
```

```
model.eval()  # 모델을 test 모드로만 사용
input_data = torch.randn(1, 3, 640, 640)


ts_model =  torch.jit.trace(model,input_data,strict=False)

trt_model = torch_tensorrt.compile(ts_model, inputs=[torch_tens
                                    ir='ts',
                                    enabled_precisions={torch.fl
                                    truncate_long_and_double=Tru
```

TensorRT 최적화 결과

```
print("TensorRT optimized")

avgs = []
for i in range(NUM_TRIALS):
    trt_results, trt_time = run_inference(trt_model, images)
    avg = sum(trt_time)/len(images)
    print("Average inference time (s):", avg)
    avgs.append(avg)

trt_mat = sum(avgs)/NUM_TRIALS
print("Mean average inference time (s):", trt_mat)
```

시간 비교

```
speed_up = mat / trt_mat
print("Speed up:", speed_up)
```

Batch 단위로 계산

이미지들을 하나의 batch로 묶어서 계산

```
import torch_tensorrt
import torch


BATCH_SIZE = 16
DYNAMIC_BATCH_ENABLED = True
input_data = torch.randn(BATCH_SIZE, 3, 640, 640)

if DYNAMIC_BATCH_ENABLED:
    # 동적인 batch size 설정 가능
    inputs = [torch_tensorrt.Input(
        min_shape=[1, 3, 640, 640],
        opt_shape=[BATCH_SIZE, 3, 640, 640],
        max_shape=[32, 3, 640, 640],  # 최대 batch 크기의 예시
        dtype=torch.float32  # 모델의 입력 타입과 같아야함
    )]
else:
    inputs = [input_data]

ts = torch.jit.trace(model, input_data, strict=False)
trt_model = torch_tensorrt.compile(ts,
                                   inputs=inputs,
                                   enabled_precisions={torch.flo
                                   truncate_long_and_double=True
```

테스트

```
def run_inference_in_batches(model, images, batch_size=BATCH_SIZ
    times = []
    results = []

    for i in range(0, len(images), batch_size):
        batch_images = images[i:i + batch_size]
```

```
        # Stack images to create a batch
        batch = torch.stack(batch_images).to('cuda')

        start = time.time()
        with torch.no_grad():
            result = model(batch)

        results.append(result)
        times.append(time.time() - start)

    return results, times
```

```
print("TensorRT optimized - Batched ", BATCH_SIZE)
avgs = []
for i in range(NUM_TRIALS):
    batched_results, batched_time = run_inference_in_batches(tr
    avg = sum(batched_time)/len(images)
    print("Average inference time (s):", avg)
    avgs.append(avg)

batched_mat = sum(avgs)/NUM_TRIALS
print("Mean average inference time (s):", batched_mat)
```

```
mat/batched_mat
```

시각화

```
%matplotlib inline
import matplotlib.pyplot as plt

speed_up_values = [mat / mat, mat / trt_mat, mat / batched_mat]
labels = ['Baseline', 'TRT', 'Batched-TRT (16)']
```

```
plt.bar(labels, speed_up_values, color=['blue', 'orange', 'green

# Adding labels and title
plt.ylabel('Speed Up in X')
plt.title('Speed Up Comparison')

# Display the plot
plt.show()
```