ONNX 轉換及模型加速推論

ONNX

- COCO dataset
- Cifar10/Cifar100
- NMIST
- ImageNet
- Cityscapes …

- ResNet
- AlexNet
- GoogleNet
- VGG…

- Tensorflow TensorFlow
- PyTorch PYTORCH

- PC
- Smartphone
- edge computing devices…

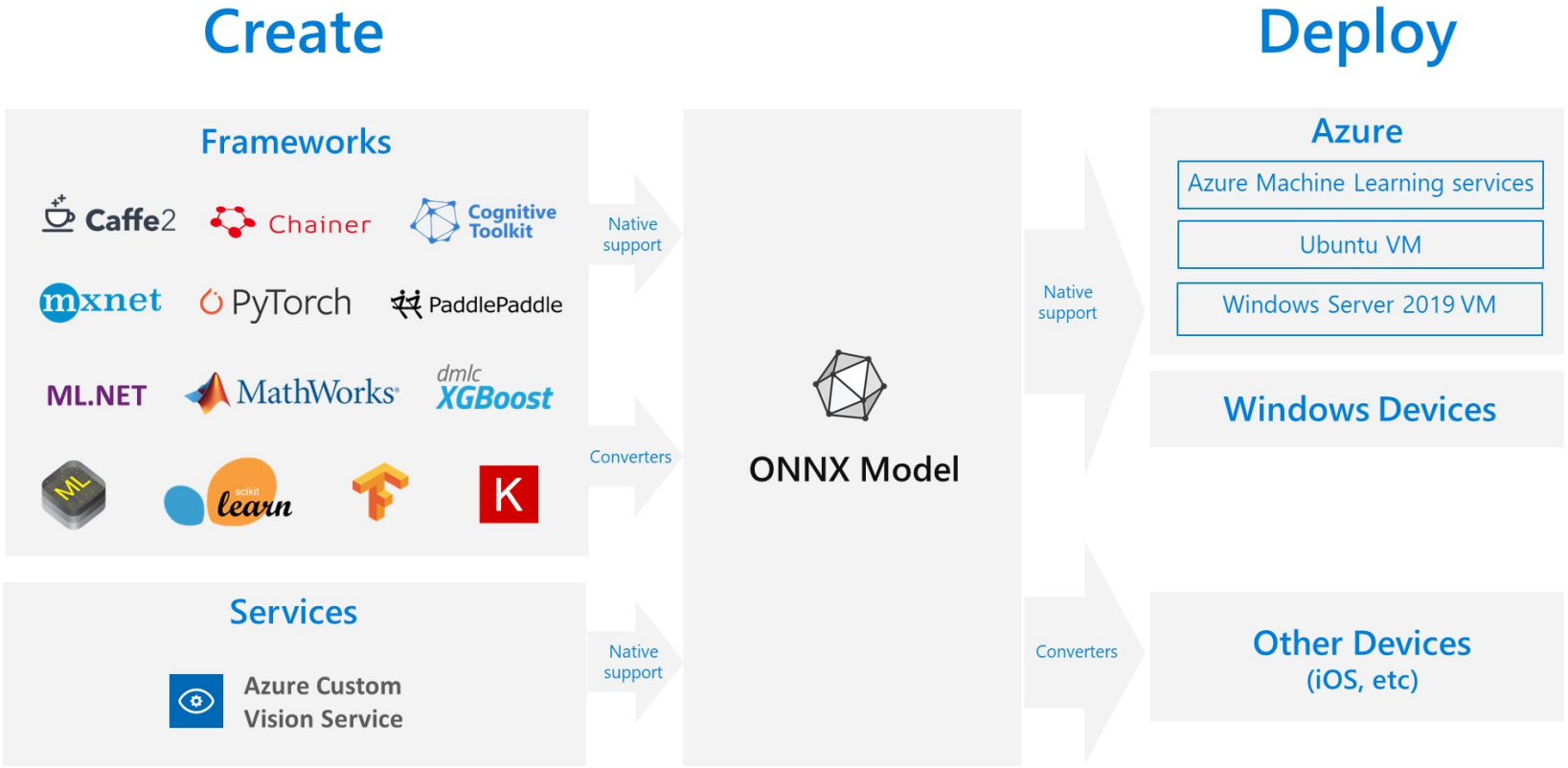
- Tensorflow TensorFlow
- PyTorch PYTORCH
- ONNX ONNX

ONNX

ONNX （Open Neural Network Exchange）是一個開放標準，主要用於讓不同深度學習框架之間能夠共享和使用模型。
ONNX 的目標之一是提供一個標準的模型表示形式，以便於模型在不同框架之間的轉換和部署

ONNX

pip install onnx
pip install onnxruntime



模型轉換

PyTorch ➡ ONNX

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(12544, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


package_dir = os.path.dirname(os.path.abspath(__file__))
default_model_path = os.path.join(package_dir, 'cifar10_model.pt')

parser = argparse.ArgumentParser(description='PyTorch cifar10 Predictor')
parser.add_argument('--model', type=str, default=default_model_path,
                    help='model for prediction (default: {})'.format(default_model_path))
args = parser.parse_args()

# 使用指定的模型檔案路徑
model = Net()
model_r = torch.load(default_model_path, map_location="cuda")
model.load_state_dict(model_r)
model.eval()

x = torch.randn(1, 3, 32, 32,  requires_grad=True)
torch.onnx.export(model, x, 'cifar10.onnx', input_names=['input'], output_names=['output'], verbose=False)
```

定義網路模型

模型轉換

PyTorch ⟹ ONNX

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(12544, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


package_dir = os.path.dirname(os.path.abspath(__file__))
default_model_path = os.path.join(package_dir, 'cifar10_model.pt')

parser = argparse.ArgumentParser(description='PyTorch cifar10 Predictor')
parser.add_argument('--model', type=str, default=default_model_path,
                    help='model for prediction (default: {})'.format(default_model_path))
args = parser.parse_args()

# 使用指定的模型檔案路徑
model = Net()
model_r = torch.load(default_model_path, map_location="cuda")
model.load_state_dict(model_r)
model.eval()

x = torch.randn(1, 3, 32, 32,  requires_grad=True)
torch.onnx.export(model, x, 'cifar10.onnx', input_names=['input'], output_names=['output'], verbose=False)
```

→ 使用 onnx.export 輸出 ONNX 網路

模型推論

PyTorch inference

```python
def load_image(image_path):
    image = Image.open(image_path).convert('RGB')
    transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    image = transform(image).unsqueeze(0)
    return image

def main():
    parser = argparse.ArgumentParser(description='PyTorch CIFAR-10 Inference')
    parser.add_argument('--img', type=str, required=True, help='path to the input image')
    args = parser.parse_args()

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = Net().to(device)
    model.load_state_dict(torch.load("cifar10_model.pt", map_location=device))
    model.eval()

    with torch.no_grad():
        image = load_image(args.img).to(device)
        # 進行推理
        t0 = time.time()
        for i in range(1000):
            output = model(image)
        t1 = time.time()
        probabilities = F.softmax(output, dim=1)
        predicted_class = torch.argmax(probabilities, dim=1).item()

    labels = ["飛機", "汽車", "鳥", "貓", "鹿", "狗", "青蛙", "馬", "船", "卡車"]
    print("概率分佈：", probabilities)
    print("預測類別：", labels[predicted_class])
    print('PyTorch推論5000次消耗時間', int(t1-t0), 's')

if __name__ == '__main__':
    main()
```

調整影像

- **將圖片調整為指定的尺寸 (32, 32)**
  transforms.Resize((32, 32))
- **將圖片轉換為 PyTorch Tensor**
  transforms.ToTensor()
- **對每個通道進行標準化。**
  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

模型推論

PyTorch inference

```python
def load_image(image_path):
    image = Image.open(image_path).convert('RGB')
    transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    image = transform(image).unsqueeze(0)
    return image

def main():
    parser = argparse.ArgumentParser(description='PyTorch CIFAR-10 Inference')
    parser.add_argument('--img', type=str, required=True, help='path to the input image')
    args = parser.parse_args()

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = Net().to(device)
    model.load_state_dict(torch.load("cifar10_model.pt", map_location=device))
    model.eval()

    with torch.no_grad():
        image = load_image(args.img).to(device)
        # 進行推理
        t0 = time.time()
        for i in range(1000):
            output = model(image)
        t1 = time.time()
        probabilities = F.softmax(output, dim=1)
        predicted_class = torch.argmax(probabilities, dim=1).item()

    labels = ["飛機", "汽車", "鳥", "貓", "鹿", "狗", "青蛙", "馬", "船", "卡車"]
    print("概率分佈：", probabilities)
    print("預測類別：", labels[predicted_class])
    print('PyTorch推論5000次消耗時間', int(t1-t0), 's')

if __name__ == '__main__':
    main()
```

模型初始化：
建模型的實例，將其移到所選擇的裝置，
載入預訓練權重

模型推論

PyTorch inference

```python
def load_image(image_path):
    image = Image.open(image_path).convert('RGB')
    transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    image = transform(image).unsqueeze(0)
    return image

def main():
    parser = argparse.ArgumentParser(description='PyTorch CIFAR-10 Inference')
    parser.add_argument('--img', type=str, required=True, help='path to the input image')
    args = parser.parse_args()

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = Net().to(device)
    model.load_state_dict(torch.load("cifar10_model.pt", map_location=device))
    model.eval()

    with torch.no_grad():
        image = load_image(args.img).to(device)
        # 進行推理
        t0 = time.time()
        for i in range(1000):
            output = model(image)
        t1 = time.time()
        probabilities = F.softmax(output, dim=1)
        predicted_class = torch.argmax(probabilities, dim=1).item()

    labels = ["飛機", "汽車", "鳥", "貓", "鹿", "狗", "青蛙", "馬", "船", "卡車"]
    print("概率分佈：", probabilities)
    print("預測類別：", labels[predicted_class])
    print('PyTorch推論5000次消耗時間', int(t1-t0), 's')

if __name__ == '__main__':
    main()
```

模型推論：

跑 1000 次，觀察推論時間

模型推論

ONNX inference

```python
def load_image(image_path):
    image = Image.open(image_path).convert('RGB')
    transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    image = transform(image).unsqueeze(0)
    return image

def main():
    parser = argparse.ArgumentParser(description='ONNX CIFAR-10 Inference')
    parser.add_argument('--img', type=str, required=True, help='path to the input image')
    args = parser.parse_args()

    # 載入 ONNX 模型並指定 GPU 運行
    session = onnxruntime.InferenceSession("./cifar10.onnx", providers=['CUDAExecutionProvider'])

    # 讀取並預處理圖片
    image = load_image(args.img).to("cuda")
    input_dict = {session.get_inputs()[0].name: image.cpu().numpy()}

    # 進行推理
    t0 = time.time()
    for i in range(1000):
        output = session.run(None, input_dict)
    t1 = time.time()

    # 將原始分數轉換為概率
    probabilities = F.softmax(torch.tensor(output[0]), dim=1)

    # 獲取最大概率對應的類別
    predicted_class = torch.argmax(probabilities, dim=1).item()
    labels = ["飛機", "汽車", "鳥", "貓", "鹿", "狗", "青蛙", "馬", "船", "卡車"]
    print("概率分佈：", probabilities.numpy())
    print("預測類別：", labels[predicted_class])
    print('ONNX推論1000次消耗時間', int(t1-t0), 's')

if __name__ == '__main__':
    main()
```

調整影像

ONNX 模型設定

Providers=[
‘TensorrtExecutionProvider’ ,
‘CUDAExecutionProvider’ ,
‘CPUExecutionProvider’]

https://onnxruntime.ai/docs/get-started/with-pytho

模型推論

比較 PyTorch 及 ONNX 推論時間差異

python3 pytorch_infer.py --img xxx.jpg

```
概率分佈： tensor([[8.2289e-01, 3.4841e-04, 3.2062e-02, 1.8582e-03, 2.3031e-03, 2.9601e-04,
         9.9342e-05, 7.6376e-05, 1.3842e-01, 1.6463e-03]], device='cuda:0')
預測類別： 飛機
PyTorch推論1000次消耗時間 4 s
```

1000 times

```
概率分佈： tensor([[8.2289e-01, 3.4841e-04, 3.2062e-02, 1.8582e-03, 2.3031e-03, 2.9601e-04,
         9.9342e-05, 7.6376e-05, 1.3842e-01, 1.6463e-03]], device='cuda:0')
預測類別： 飛機
PyTorch推論10000次消耗時間 11 s
```

10000 times

python3 onnx_infer.py --img xxx.jpg

```
概率分佈： [[8.2290685e-01 3.4846656e-04 3.2054279e-02 1.8576067e-03 2.3028094e-03
  2.9585965e-04 9.9300894e-05 7.6347642e-05 1.3841213e-01 1.6463229e-03]]
預測類別： 飛機
ONNX推論1000次消耗時間 2 s
```

1000 times

```
概率分佈： [[8.2290685e-01 3.4846656e-04 3.2054279e-02 1.8576067e-03 2.3028094e-03
  2.9585965e-04 9.9300894e-05 7.6347642e-05 1.3841213e-01 1.6463229e-03]]
預測類別： 飛機
ONNX推論10000次消耗時間 9 s
```

10000 times

目標

1. 比較 PyTorch 及 ONNX 推論時間差異

報告形式：
將程式碼與詳細註解以文字形式或圖片貼入 Word 檔，連同執行結果截圖，轉成 PDF 檔 (week3 、 week4) 。
檔案名稱以 HW2_ 學號命名，例如 HW2_M11201234.pdf 。
將程式碼與 PDF 打包成 zip 上傳至 Moodle 2 作業區。