Университет ИТМО

Лабораторная работа №3
«Конвертация разработанной нейронной сети с помощью ONNX»

по дисциплине: Технологии нейросетевых вычислений

вариант: Классификатор снимков с пневмонией

Выполнил: Неграш Андрей, P34301

Преподаватель: Старобыховская Анастасия Александровна

Санкт-Петербург
2023

1. Цель

Экспортировать созданную в рамках лабораторной работы №1 или №2 в ONNX, проверить запускаемость через onnxruntime и проанализировать граф вычислений.

2. Результаты работы

В качестве исходной модели я выбрал модель, выполненную в лабораторной работе №1 и добавил следующий блок кода после обучения:

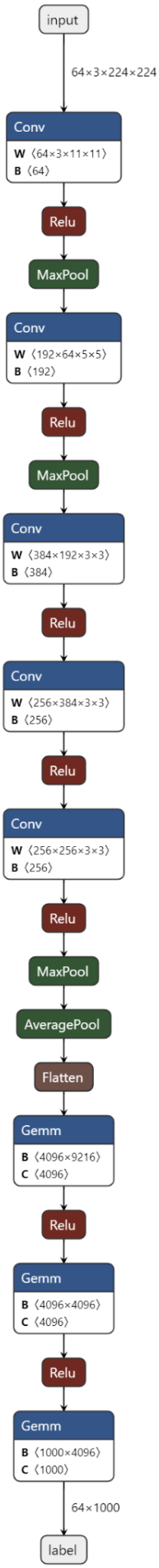
```
cmodel_path = base_path+"/cust_model.onnx"
input_size=(64, 3, 224, 224)
dummy_input = Variable(torch.randn([64, 3, 224, 224]))

onnx_program = torch.onnx.export(cnn,
                                 dummy_input.cuda(),
                                 cmodel_path,
                                 input_names = ['input'],
                                 output_names = ['label'])
```

Благодаря этому коду был получен файл *cust_model.onnx*, который я проанализировал с помощью сервиса https://netron.app/.

В результате был получен граф вычислений, представленный в графическом виде ниже. Он представляет собой графовое вид модели AlexNet, которая как раз и использовалась для обучения.

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```


input
64×3×224×224
Conv
W 〈64×3×11×11〉
B 〈64〉
Relu
MaxPool
Conv
W 〈192×64×5×5〉
B 〈192〉
Relu
MaxPool
Conv
W 〈384×192×3×3〉
B 〈384〉
Relu
Conv
W 〈256×384×3×3〉
B 〈256〉
Relu
Conv
W 〈256×256×3×3〉
B 〈256〉
Relu
MaxPool
AveragePool
Flatten
Gemm
B 〈4096×9216〉
C 〈4096〉
Relu
Gemm
B 〈4096×4096〉
C 〈4096〉
Relu
Gemm
B 〈1000×4096〉
C 〈1000〉
64×1000
label

При тестировании на GPU были получены следующие результаты времени выполнения (поскольку входными данными были одни и те же, то скорость вычисления напрямую зависит только от времени):

```
import time

start_time = time.time()
outputs = session.run(None, {'input':  dummy_input.numpy()})
end_time = time.time()
print("Onnx: {}".format(end_time - start_time))

start_time_t = time.time()
torch_outputs = cnn(dummy_input.cuda())
end_time_t = time.time()
print("Normal: {}".format(end_time_t - start_time_t))
```

```
Onnx: 1.2778558731079102
Normal: 0.01175236701965332
```

По данному скриншоту видно, что запущенная с помощью onnxruntime модель существенно проигрывает изначальной модели по скорости выполнения – разница более чем в 100 раз!

Однако на CPU ситуация иная:

```
import time

start_time = time.time()
outputs = session.run(None, {'input':  dummy_input.numpy()})
end_time = time.time()
print("Onnx: {}".format(end_time - start_time))

start_time_t = time.time()
torch_outputs = cnn(dummy_input)
end_time_t = time.time()
print("Normal: {}".format(end_time_t - start_time_t))
```

```
Onnx: 1.5630879402160645
Normal: 2.026010274887085
```

Таким образом можно заметить, что инференс модели с использованием ONNX на CPU, то есть более дешёвом средстве выполнения, принёс преимущество в скорости вычислений.

3. Вывод

Итак, в процессе данной лабораторной работы я провёл экспорт разработанной ранее нейронной сети, классифицирующей рентгеновские снимки по признаку наличия на них пневмонии, в формат ONNX и проанализировал получившийся граф и значения скорости работы на инференсе.