




23년도 1학기 딥러닝 개론

Programming assignment

황지현 (aribae@etri.re.kr)
2023.06.12



What to submit

1. This **.ipynb file** (with all the code) (24 pts)
2. a printout version of this file (with all the code execution results shown)
3. **A presentation file** for explaining what you have done (may include purpose of each experiment, results, analysis, your opinion)

Data preparation

- google colab 사용, 드라이브 연동 문제로 kaggle API 활용하여 pokemon dataset download

```
[14] 1 # ! cp kaggle.json ~/.kaggle/  
      2 # ! chmod 600 ~/.kaggle/kaggle.json  
      3 # ! kaggle datasets list  
      4  
      5 !kaggle datasets download -d lantian773030/pokemonclassification
```

```
Downloading pokemonclassification.zip to /content  
100% 416M/417M [00:22<00:00, 23.2MB/s]  
100% 417M/417M [00:22<00:00, 19.5MB/s]
```

```
[15] 1 !unzip pokemonclassification.zip
```

3. Assignment

- Assignments 1 to 9 should start with `model_o` .
- Assignments 10-12 start with Resnet 34
- Good presentation: Through the assignments, you should present the results effectively by using the best visualization methods for the given assignments, for example, either text printout, graph, charts, or whatever that's most appropriate for the results.
- You can modify `model_o` for each assignment as necessary
- Put your opinion on the results

base model_0 구성과 성능 (epoch 10)

```
3 # model_0
4 class ConvolutionalNetwork(nn.Module):
5     def __init__(self):
6         super().__init__()
7         self.conv1 = nn.Conv2d(3, 6, 3, 1)
8         self.conv2 = nn.Conv2d(6, 12, 3, 1)
9         self.conv3 = nn.Conv2d(12, 24, 3, 1)
10        self.fc1 = nn.Linear(28*28*24, 512)
11        self.fc2 = nn.Linear(512, 256)
12        self.fc3 = nn.Linear(256, num_classes)
13
14    def forward(self, X):
15        X = F.relu(self.conv1(X)) # conv1
16        X = F.max_pool2d(X, 2, 2)
17        X = F.relu(self.conv2(X)) # conv2
18        X = F.max_pool2d(X, 2, 2)
19        X = F.relu(self.conv3(X)) # conv3
20        X = F.max_pool2d(X, 2, 2)
21        X = X.view(-1, 28*28*24)
22        X = F.relu(self.fc1(X))
23        X = F.relu(self.fc2(X))
24        X = self.fc3(X)
25        return F.log_softmax(X, dim=1)
```

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

1. Several Techniques (2pts)

Apply Xavier weight initialization, L2 regularization, Dropout, Batch Normalization.
Each should start with `model_o`

```
# your code for Xavier weight initialization
```

```
# your code for L2 regularization
```

```
# your code for Dropout
```

```
# your code for Batch Normalization
```

1-1. Xavier weight initialization

- `init.xavier_uniform_(m.weight)`를 통해 Xavier 가중치 초기화 수행
- 기존 `model_0` class에 `init.xavier` 추가 (`model_0`보다 **loss & 정확도 둘 다 감소**)

```
def xavier_init(self):  
    for module in self.modules():  
        if isinstance(module, nn.Conv2d) or isinstance(module, nn.Linear):  
            nn.init.xavier_uniform_(module.weight)  
            if module.bias is not None:  
                nn.init.constant_(module.bias, 0)
```

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

Xavier result (epoch=10)		
	loss	accuracy
validation	0.6219	0.4146
test	0.6314	0.4314

1-2. L2 regularization

- **L2 regularization:** 모델의 가중치에 대한 제약 조건을 추가하여 과적합을 줄이는 방식
- `model_0`의 구조는 변경하지 않고, `weight_decay` 매개변수 설정 (`model_0`보다 성능 감소)

```
1 model = ConvolutionalNetwork()
2 model.to(device)
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0.001)
6
7 train(model, 10, train_loader, val_loader)
8 torch.save(model.state_dict(), 'PokemonModel_L2.pt')
9
10 test(model, test_loader)
```

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

L2 result (epoch=10)		
	loss	accuracy
validation	0.7472	0.2537
test	0.7581	0.2437

1-3. Dropout

- **Dropout:** 신경망의 과적합을 줄이기 위해 사용되는 정규화(regularization) 기법
- Dropout 확률 0.5로 설정(학습 중 뉴런의 출력을 50%의 확률로 0으로 만듦으로써 과적합 방지)
 - 0.2와도 비교

```
class ConvolutionalNetwork_drop5(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.conv1 = nn.Conv2d(3, 6, 3, 1)  
        self.conv2 = nn.Conv2d(6, 12, 3, 1)  
        self.conv3 = nn.Conv2d(12, 24, 3, 1)  
        self.fc1 = nn.Linear(28*28*24, 512)  
        self.fc2 = nn.Linear(512, 256)  
        self.fc3 = nn.Linear(256, num_classes)  
  
        # dropout layer  
        self.dropout = nn.Dropout(p=0.5)
```

```
15 def forward(self, X):  
16     X = F.relu(self.conv1(X))  
17     X = F.max_pool2d(X, 2, 2)  
18     X = F.relu(self.conv2(X))  
19     X = F.max_pool2d(X, 2, 2)  
20     X = F.relu(self.conv3(X))  
21     X = F.max_pool2d(X, 2, 2)  
22     X = X.view(-1, 28*28*24)  
23     X = F.relu(self.fc1(X))  
24  
25     # dropout  
26     X = self.dropout(X)  
27     X = F.relu(self.fc2(X))  
28     X = self.dropout(X)  
29     X = self.fc3(X)  
30  
31     return F.log_softmax(X, dim=1)
```

1-3. Dropout

Result

- 0.5로 dropout을 설정하니 성능이 매우 낮게 나옴
 - 데이터셋 이미지 수가 적어 반을 과적합이 발생했을 가능성이 있다고 보여 0.2와 비교 진행
- 예상과 동일하게 0.2로 설정하였을 때 비교적 더 높은 성능을 보임
 - 충분한 데이터셋이 학습에 정말 중요하다는 것을 다시 한 번 느낌

Dropout 0.5 result (epoch=10)		
	loss	accuracy
validation	0.9531	0.1333
test	0.9011	0.1778

Dropout 0.2 result (epoch=10)		
	loss	accuracy
validation	0.8201	0.2358
test	0.7417	0.2943

1-4. Batch Normalization

- **Batch Normalization:** 학습 중 레이어의 입력을 정규화하여 학습을 안정화시키는 기법
- Convolution 레이어(마지막) 다음에 Batch Normalization을 추가한다.

```
1 class ConvolutionalNetwork_bn(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.conv1 = nn.Conv2d(3, 6, 3, 1)
5         self.conv2 = nn.Conv2d(6, 12, 3, 1)
6         self.conv3 = nn.Conv2d(12, 24, 3, 1, bias=False)
7
8         # Batch norm 적용
9         self.batchnorm = nn.BatchNorm2d(24)
10        self.fc1 = nn.Linear(28*28*24, 512)
11        self.fc2 = nn.Linear(512, 256)
12        self.fc3 = nn.Linear(256, num_classes)
```

```
14 def forward(self, X):
15     X = F.relu((self.conv1(X))) # conv1
16     X = F.max_pool2d(X, 2, 2)
17     X = F.relu((self.conv2(X))) # conv2
18     X = F.max_pool2d(X, 2, 2)
19
20     # Batch norm 적용
21     X = F.relu(self.batchnorm(self.conv3(X))) # conv3
22     X = F.max_pool2d(X, 2, 2)
23     X = X.view(-1, 28*28*24)
24     X = F.relu((self.fc1(X)))
25     X = F.relu(self.fc2(X))
26     X = self.fc3(X)
27
28     return F.log_softmax(X, dim=1)
```

1-4. Batch Normalization

Result

- 앞선 모델들과 동일하게 epoch를 10으로 설정하였으나, 성능이 눈에 띄게 낮아 보완하기 위한 방안 모색
 - epoch 조정: 10, 20, 20 각각 비교
- epoch가 증가함에 따라 성능이 향상하는 것을 볼 수 있었음
- 실험을 통해 적절한 epoch를 설정하는 것이 중요하겠음

BN result (epoch=10)		
	loss	accuracy
val	1.1437	0.0244
test	1.0736	0.0246

BN result (epoch=20)		
	loss	accuracy
val	1.0065	0.1675
test	0.8063	0.2357

BN result (epoch=30)		
	loss	accuracy
val	0.9692	0.4358
test	0.9897	0.4268

학습 방식 별 Best Accuracy 비교

- 동일한 dataset, 각각 다른 학습 방식에서의 validation/test 결과 비교
 - 정확도가 높아진다고 항상 loss가 줄어드는 것은 아님
 - 과적합 발생 가능, 과적합 없이 최적의 성능을 낼 수 있는 방안을 찾는 것이 중요하겠음

Validation	VAL_LOSS	VAL_ACC
Xavier	0.6219	0.4146
L2	0.7472	0.2537
Dropout (0.2)	0.8201	0.2358
Batch normalization (epoch=30)	0.9692	0.4358

TEST	TEST_LOSS	TEST_ACC
Xavier	0.6314	0.4314
L2	0.7581	0.2437
Dropout (0.2)	0.7417	0.2943
Batch normalization (epoch=30)	0.9897	0.4268

2. Optimization variants (1pts)

Apply SGD, Momentum+SGD, maybe others optimizers can be tried as well

```
# your code for SGD
```

```
# your code for Momentum+SGD
```

2-1. SGD

- model_0의 구조는 변경하지 않고, **optimizer만 SGD로 변경**하여 학습
- **model_0보다 낮은 성능**을 보임

```
1 # your code for SGD
2 model = ConvolutionalNetwork()
3 model.to(device)
4
5 criterion = nn.CrossEntropyLoss()
6 optimizer = torch.optim.SGD(model.parameters(), lr=0.01) # SGD를 적용한 optimizer
7
8 train(model, 10, train_loader, val_loader)
9 test(model, test_loader)
```

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

SGD result (epoch=10)		
	loss	accuracy
validation	0.6998	0.3528
test	0.6968	0.3475

2-2. Momentum SGD

- **momentum**: 경사하강법에 관성을 부여, 학습을 안정화 할 수 있다.
- SGD와 동일하게 **model_0**의 구조는 변경하지 않고, **optimizer SGD로 변경 + 모멘텀 매개변수** 설정
 - 설정한 **momentum = 0.9**
- **관성을 부여한 경우** 부여하지 않았던 경우에 비해 **성능 향상을 보임**

```
# your code for Momentum+SGD
model = ConvolutionalNetwork()
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9) # momentum을 적용한 SGD optimizer

train(model, 10, train_loader, val_loader)
test(model, test_loader)
```

SGD result (epoch=10)		
	loss	accuracy
validation	0.6998	0.3528
test	0.6968	0.3475

Momentum SGD result (epoch=10)		
	loss	accuracy
validation	0.5511	0.4504
test	0.5641	0.4254

3. Learning rate annealing (1pts)

Apply StepLR, MultistepLR, Exponential LR

```
# your code for StepLR
```

```
# your code for MultistepLR,
```

```
# your code for Exponential LR
```

3-1. Step LR

- **Step LR:** 일정한 에폭마다(step_size) 학습률 감소 (gamma* = 0.1)
 - step_size : 10(감소하지 않은 경우), 5, 2 비교

gamma* : Multi-step LR까지 gamma를 잘 못 설정함.. Exponential model에서 gamma 설정에 따른 차이 비교 (시간이 된다면 0.9로 모두 수정할 예정.....)

```
1 # your code for StepLR
2 model = ConvolutionalNetwork()
3 model.to(device)
4
5 criterion = nn.CrossEntropyLoss()
6 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
7
8 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1) # StepLR을 적용한 scheduler
9
10 train(model, 10, train_loader, val_loader)
11 test(model, test_loader)
```

3-1. Step LR

Result

- 2epoch마다 학습률 감소한 경우 나머지 경우들과 비교하였을 때 loss는 비슷했지만 가장 높은 정확도를 보임
- 학습률 감소 주기 외에도 epoch, batch size, learning rate 등 최적의 파라미터를 찾는 것이 중요하겠음

Step 10 result (epoch=10)		
	loss	accuracy
val	0.7472	0.3886
test	0.7162	0.3901

Step 5 result (epoch=10)		
	loss	accuracy
val	0.7893	0.3073
test	0.7875	0.3189

Step 2 result (epoch=10)		
	loss	accuracy
val	0.6307	0.4228
test	0.6381	0.3921

3-2. Multi-step LR

- **Step LR:** 주어진 에폭마다(step_size) 학습률 감소 (gamma = 0.1)
- 다른 문제들과 동일하게 epoch 10으로 해보려 했는데, 감소주기가 짧은 듯 싶어 30과 비교
 - momentum in epoch 10 : 3, 5, 7
 - momentum in epoch 30 : 10, 20

```
1 # your code for MultistepLR_1
2 model = ConvolutionalNetwork()
3 model.to(device)
4
5 criterion = nn.CrossEntropyLoss()
6 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
7
8 milestones = [3,5,7] # 학습률을 감소시킬 epochs
9 gamma = 0.1 # 학습률을 감소시킬 비율
10
11 scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=milestones, gamma=gamma) # MultiStepLR을 적용한 scheduler
12
13 train(model, 10, train_loader, val_loader)
14 test(model, test_loader)
```

3-2. Multi-step LR

Result

- Batch Normalization을 적용한 모델과 동일하게 epoch가 늘어났을 때 성능 증가를 보임
- 하지만 동일한 bs에서 multi-step간 비교를 진행해보지 못하여 이 또한 실험을 통해 적절한 step size를 설정한다면 더 높은 성능을 보일 수 있을 것이라 생각됨

MsLR result (epoch=10 / 3,5,7)		
	loss	accuracy
validation	0.6871	0.3545
test	0.6879	0.3602

MsLR result (epoch=30 / 10,20)		
	loss	accuracy
validation	0.9692	0.4358
test	0.9897	0.4268

3-3. Exponential LR

- Exponential LR: **매 epoch마다** 학습률 감소($\text{gamma} = 0.9$)
 - 직전 코드까지 계산을 반대로 해서 gamma 를 0.1로 설정
 - 근데 이번 코드에서 0.9로 설정한 것과 성능 차이가 크지 않음...(크게 낮지 않음)

```
1 # your code for Exponential LR
2 model = ConvolutionalNetwork()
3 model.to(device)
4
5 criterion = nn.CrossEntropyLoss()
6 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
7
8 gamma = 0.9 # 학습률을 감소시킬 비율
9
10 scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=gamma) # ExponentialLR을 적용한 scheduler
11
12 train(model, 10, train_loader, val_loader)
13 test(model, test_loader)
```

3-3. Exponential LR

Result

- gamma 값을 잘못 설정하였을 때, 눈에 띄는 성능 저하가 있는 것은 아니었다
- 하지만 0.9와 비교하였을 때는 낮은 성능을 보임
- 앞 경우들에도 0.9를 사용했다면 보다 높은 성능이 나왔을 것으로 보임

ELR result (epoch=10, gamma=0.9)			ELR result (epoch=10, gamma=0.1)		
	loss	accuracy		loss	accuracy
validation	0.7198	0.3642	validation	0.7225	0.3415
test	0.6995	0.3735	test	0.7319	0.3395

4. Dilated Conv (1pts)

we can use dilated 3x3 conv to enlarge the receptive field. So this time, let's try dilated 3x3 conv to see what difference it makes.

- 확장된 convolution network를 사용하면 입력 이미지에 대해 더 많은 정보를 얻어낼 수 있음

```
2 class ConvolutionalNetwork_33(nn.Module):
3     def __init__(self):
4         super().__init__()
5         self.conv1 = nn.Conv2d(3, 6, 3, stride=1, dilation=2) # Dilated convolution with dilation rate of 2
6         self.conv2 = nn.Conv2d(6, 12, 3, stride=1, dilation=2) # Dilated convolution with dilation rate of 2
7         self.conv3 = nn.Conv2d(12, 24, 3, stride=1, dilation=2) # Dilated convolution with dilation rate of 2
8         self.fc1 = nn.Linear(26*26*24, 512)
9         self.fc2 = nn.Linear(512, 256)
10        self.fc3 = nn.Linear(256, num_classes)
11
12    def forward(self, X):
13        X = F.relu(self.conv1(X))
14        X = F.max_pool2d(X, 2, 2)
15        X = F.relu(self.conv2(X))
16        X = F.max_pool2d(X, 2, 2)
17        X = F.relu(self.conv3(X))
18        X = F.max_pool2d(X, 2, 2)
19        X = X.view(-1, 26*26*24)
20        X = F.relu(self.fc1(X))
21        X = F.relu(self.fc2(X))
22        X = self.fc3(X)
23        return F.log_softmax(X, dim=1)
```


4. Dilated 3*3 Conv

Result

- model_0과 성능 비교: loss 값은 비슷하지만, accuracy는 비교적 작았음
- 그림이 아닌 사진이었다면 이미지 정보가 더 많으니, 성능 향상을 보였을지 궁금

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

Dilated result (epoch=10)		
	loss	accuracy
validation	0.6891	0.3724
test	0.7326	0.3375

5. 1x1 Conv (2pts)

As we learned in the class 1x1 conv can be utilized to reduce the computation. To see the computation reduction, lets first change one or more of the three 3x3 convs to be 7x7 (you may have to adjust other layers as well in order to the NN work). Then apply 1x1 technique to make it faster

```
1 # your code here
2 class ConvolutionalNetwork_11(nn.Module):
3     def __init__(self):
4         super().__init__()
5         self.conv1 = nn.Conv2d(3,6,7,1) # convert to 7*7
6         self.conv2 = nn.Conv2d(3,12,7,1) # convert to 7*7
7         self.conv3 = nn.Conv2d(6,24,7,1) # convert to 7*7
8
9         self.conv1_1 = nn.Conv2d(6,3,1,1) # convert to 1*1
10        self.conv2_1 = nn.Conv2d(12,6,1,1) # convert to 1*1
11        self.conv3_1 = nn.Conv2d(24,12,1,1) # convert to 1*1
12
13        self.fc1 = nn.Linear(24*24*12, 512)
14        self.fc2 = nn.Linear(512, 256)
15        self.fc3 = nn.Linear(256, num_classes)
16
```

```
16
17     def forward(self, X):
18         X = F.relu(self.conv1(X))
19         X = F.max_pool2d(X,2,2)
20         X = F.relu(self.conv1_1(X))
21
22         X = F.relu(self.conv2(X))
23         X = F.max_pool2d(X,2,2)
24         X = F.relu(self.conv2_1(X))
25
26         X = F.relu(self.conv3(X))
27         X = F.max_pool2d(X,2,2)
28         X = F.relu(self.conv3_1(X))
29
30         X = X.view(-1, 24*24*12)
31         X = F.relu(self.fc1(X))
32         X = F.relu(self.fc2(X))
33         X = self.fc3(X)
34         return F.log_softmax(X,dim=1)
35
```

5. 1*1 Conv

Result

- model_0에 비해 성능 감소,, (비교할 필요도 없이 너무 낮음)
- 레이어 구조를 바꿔보아도 동일하거나 비슷한 성능 나옴
- 어떻게 사용해야 하는지 더 알아볼 필요가 있겠음

epoch 바꿔서 해보고 싶는데 시간부족으로 못 해봤습니다.. ㅜㅜ!!

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

11 conv result (epoch=10)		
	loss	accuracy
validation	0.1.2317	0.0098
test	0.1.2520	0.0093

6. Instable NN (1pts)

Let's modify the model so that the model has softmax output inside the model and then apply log and NLLLoss to test whether it really has numerical instability

```
1 class ConvolutionalNetwork_nn2(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.conv1 = nn.Conv2d(3, 6, 3, 1)
5         self.conv2 = nn.Conv2d(6, 12, 3, 1)
6         self.conv3 = nn.Conv2d(12, 24, 3, 1)
7         self.fc1 = nn.Linear(28*28*24, 512)
8         self.fc2 = nn.Linear(512, 256)
9         self.fc3 = nn.Linear(256, num_classes)
10        self.softmax = nn.LogSoftmax(dim=1) # Apply softmax and log inside the model
11        self.loss_fn = nn.NLLLoss() # Define the NLLLoss
12
13    def forward(self, x):
14        x = nn.functional.relu(self.conv1(x))
15        x = nn.functional.max_pool2d(x, 2, 2)
16        x = nn.functional.relu(self.conv2(x))
17        x = nn.functional.max_pool2d(x, 2, 2)
18        x = nn.functional.relu(self.conv3(x))
19        x = nn.functional.max_pool2d(x, 2, 2)
20        x = x.view(-1, 28*28*24)
21        x = nn.functional.relu(self.fc1(x))
22        x = nn.functional.relu(self.fc2(x))
23        x = self.fc3(x)
24        x = self.softmax(x) # Apply softmax and log
25
26    return x
```

6. Instable NN

Result

- 찾아보니 criterion에 NLLLoss 사용할 수 있어 Crossentropy와 둘 다 사용해 봄
 - **CrossEntropyLoss**를 사용했을 때 성능 증가
- 두 손실함수간의 차이가 크다고 할 만큼 벌어지지 않아서 어떤 방식을 쓰는 것이 더 적절한지 공부해볼 필요가 있음

```
5 criterion = nn.CrossEntropyLoss()
```

CrossEntropyLoss result (epoch=10)		
	loss	accuracy
validation	0.7420	0.3512
test	0.7403	0.3422

```
4 criterion = nn.NLLLoss()
```

NNNLoss result (epoch=10)		
	loss	accuracy
validation	0.8070	0.2423
test	0.8271	0.2170

7. variable sized input (2pts)

The current model requires the input to be of 240x240 or more precisely 238x238 (both work with current model). Let's make the model takes any size input by using adaptive pooling. You may need to remove one or more pooling layers to make the minimum input size (you need to calculate this considering the operations in your model) less smaller than 240(or 238) so that you don't need to resize many of the images to make it bigger to fit the minimum input size. Still there maybe some images smaller than the minimum size, in that case you may have to reisze the image to the minimum size.

7. Variable sized input

- Adaptive pooling layer를 추가하여 모델이 모든 크기의 이미지를 입력받을 수 있도록 함
- 크기가 더 작은 이미지에 대해서는 최소 크기로 조정 (fc1)

```
class ConvolutionalNetwork_input(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 12, 3, 1)
        self.conv3 = nn.Conv2d(12, 24, 3, 1)

        self.adaptive_pool = nn.AdaptiveAvgPool2d((7, 7)) # Adjust output size dynamically
        self.fc1 = nn.Linear(24 * 7 * 7, 512) # Adjust input size

        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))

        x = self.adaptive_pool(x) # Apply adaptive pooling

        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

7. Variable sized input

Result

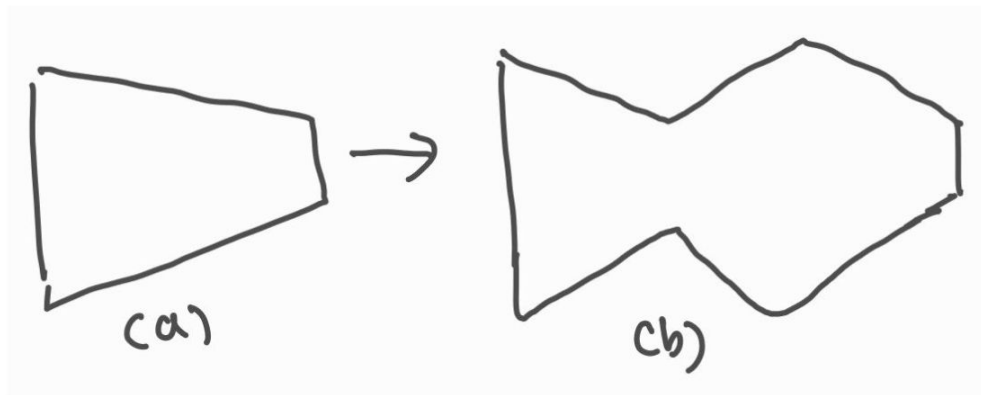
- model_0과 비교해보았을 때 성능 감소를 보임
- 다양한 크기의 이미지 데이터에서 비교해 보아야 할 듯 함

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

Variable result (epoch=10)		
	loss	accuracy
validation	0.7175	0.3187
test	0.7330	0.2969

8. Upsampling by transposed conv (2pts)

the model so far will look like (a) below. In order to get familiar with upsampling. Let's just apply the transposed conv here so that the model looks like (b).



```
x = x.unsqueeze(2).unsqueeze(3) # Expand dimensions to 4D
x = self.upsample(x)
x = x.squeeze(2).squeeze(2) # Squeeze dimensions back to 2D
```

8. Upsampling

- squeeze 사용해 upsampling 수행
- 배치사이즈 에러 발생
- train 함수 선언할 때 배치사이즈 설정을 바꿔도 안됨..

```
] 1 batch_size1 = 16
2 train_loader1 = DataLoader(train_data, batch_size=batch_size1, shuffle=True)
3 val_loader1 = DataLoader(val_data, batch_size=batch_size1, shuffle=False)
4 test_loader1 = DataLoader(test_data, batch_size=batch_size1, shuffle=False)
5
6 def train1(model, epochs, train_loader1, val_loader1=None):
7     model.train()
8     start_time = time.time()
9
10    # Trackers
```

```
class ConvolutionalNetwork8(nn.Module):
    def __init__(self):
        super(ConvolutionalNetwork8, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 12, 3, 1)
        self.conv3 = nn.Conv2d(12, 24, 3, 1)
        self.fc1 = nn.Linear(7 * 7 * 24, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, num_classes)
        self.upsample = nn.Upsample(scale_factor=2, mode='nearest')

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv3(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 7 * 7 * 24)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = x.unsqueeze(2).unsqueeze(3) # Expand dimensions to 4D
        x = self.upsample(x)
        x = x.squeeze(2).squeeze(2) # Squeeze dimensions back to 2D
        y = F.log_softmax(x, dim=1)
        return x
```

ValueError: Expected input batch_size (3) to match target batch_size (4).

[SEARCH STACK OVERFLOW](#)

8. Upsampling

- 레이어 추가하고 아무리 바꿔도 해결 안됨 ..
- 레이어 추가하면 입력 크기 안 맞는다고 나옴

```
-----  
RuntimeError                                Traceback (most recent call last)  
<ipython-input-100-846ab87af0df> in <cell line: 40>()  
    38 criterion = nn.CrossEntropyLoss()  
    39 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)  
--> 40 train(model, 10, train_loader, val_loader)  
    41 test(model, test_loader)  
    42
```

```
----- 2 frames -----  
<ipython-input-100-846ab87af0df> in forward(self, x)  
    24     x = F.relu(self.conv3(x))  
    25     x = F.max_pool2d(x, 2, 2)  
--> 26     x = x.view(-1, 7 * 7 * 512)  
    27     x = F.relu(self.fc1(x))  
    28     x = F.relu(self.fc2(x))
```

RuntimeError: shape '[-1, 25088]' is invalid for input of size 32448

SEARCH STACK OVERFLOW

```
self.conv2 = nn.Conv2d(6, 12, 3, 1)  
  
self.conv22 = nn.Conv2d(12, 24, 3, 1)  
  
self.conv3 = nn.Conv2d(24, 48, 3, 1)  
self.fc1 = nn.Linear(7 * 7 * 512, 1024)  
  
x = F.relu(self.conv2(x))  
x = F.max_pool2d(x, 2, 2)  
  
x = F.relu(self.conv22(x))  
x = F.max_pool2d(x, 2, 2)  
  
x = F.relu(self.conv3(x))  
x = F.max_pool2d(x, 2, 2)
```

8. Upsampling 해결..

- torch에서 제공하는 Upsample을 사용했는데, 문제에 ConvTranspose라고 나와있음
- 바꿔주고 입출력 크기 맞춰주니 실행되었다 ... !!
 - 주어진 문제를 잘 봐야함

```
class ConvolutionalNetwork8(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.conv1 = nn.Conv2d(3, 6, 3, 1)  
        self.conv2 = nn.Conv2d(6, 12, 3, 1)  
        self.conv3 = nn.Conv2d(12, 24, 2, 1)  
        self.convt = nn.ConvTranspose2d(12, 12, 2, 2)  
        self.fc1 = nn.Linear(18816, 512)  
        self.fc2 = nn.Linear(512, 256)  
        self.fc3 = nn.Linear(256, num_classes)
```

```
def forward(self, X):  
    X = F.relu(self.conv1(X))  
    X = F.max_pool2d(X, 2, 2)  
    X = F.relu(self.conv2(X))  
    X = F.max_pool2d(X, 2, 2)  
    X = F.relu(self.convt(X))  
    X = F.max_pool2d(X, 2, 2)  
    X = F.relu(self.conv3(X))  
    X = F.max_pool2d(X, 2, 2)  
    X = X.view(X.size(0), -1)  
    X = F.relu(self.fc1(X))  
    X = self.fc2(X)  
    X = self.fc3(X)  
    return F.log_softmax(X, dim=1)
```

8. Upsampling

Result

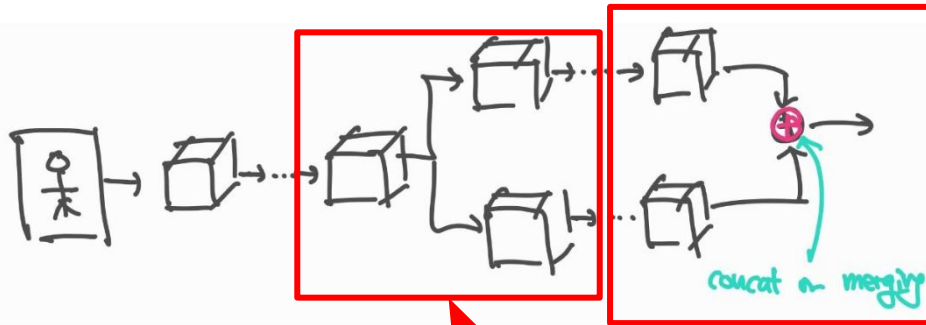
- Upsampling을 적용하면 데이터가 기존에 비해 정렬되어 성능향상을 보일 것이라 예상했는데, 오히려 감소를 보임
 - 아마 내가 변수를 잘못 설정해서..
 - 하지만 loss값에서는 큰 차이가 없었다.

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

Upsampling result (epoch=10)		
	loss	accuracy
validation	0.6213	0.3593
test	0.6353	0.4035

9. Two branch network (3pts)

Let's make a two-branch model, which looks like below.



```
class ConvolutionalNetwork_twobranch(nn.Module):
    def __init__(self):
        super().__init__()
        self.shared_conv = nn.Sequential(
            nn.Conv2d(3, 6, 3, 1),
            nn.ReLU(), nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 12, 3, 1),
            nn.ReLU(), nn.MaxPool2d(2, 2),
            nn.Conv2d(12, 24, 3, 1),
            nn.ReLU(), nn.MaxPool2d(2, 2))
```

```
        self.branch1_fc = nn.Sequential(
            nn.Linear(28 * 28 * 24, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, num_classes))

        self.branch2_fc = nn.Sequential(
            nn.Linear(28 * 28 * 24, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, num_classes))
```

```
    def forward(self, x):
        x = self.shared_conv(x)
        x = x.view(-1, 28 * 28 * 24)
        output1 = self.branch1_fc(x)
        output2 = self.branch2_fc(x)
        x = torch.cat((output1, output2), dim=1)
        return F.log_softmax(x, dim=1)
```

9. Two branch Network

Result

- model_0과 비교해보았을 때, loss는 비슷 & 정확도는 감소
- 본 모델에서는 적합하지 않은 구조인가?
 - 그건 알 수 없음..

Model_0 result (epoch=10)		
	loss	accuracy
validation	0.6892	0.4340
test	0.6417	0.4055

Two branch result (epoch=10)		
	loss	accuracy
validation	0.6943	0.3642
test	0.7166	0.3602

10. Pretrained Model (3pts)

Let's use pretrained model instead of the custom model we defined. Use a pre-trained ResNet-34 to train the model. Things to consider here is for example, up to which layers of the Resnet you are going to keep and what layers to drop. Up to which layers to freeze. Also need to adjust final classifier layers.

- torchvision.models resnet34 를 사용해 모델 구조를 불러와 pokemon dataset으로 학습

```
1 # your code here
2 from torchvision.models import resnet34
3 # Load the pre-trained ResNet-34 model
4 pretrained_model = resnet34(pretrained=True)
5
6 # Modify the model
7 num_classes = num_classes # Number of classes in your dataset / 같은데 굳이 안 해도 될 듯?
8 pretrained_model.fc = nn.Linear(pretrained_model.fc.in_features, num_classes)
9
10 pretrained_model.to(device)
11
12 criterion = nn.CrossEntropyLoss().to(device)
13 optimizer = torch.optim.Adam(pretrained_model.parameters(), lr=0.001)
14
15 train(pretrained_model, 10, train_loader, val_loader)
16 test(pretrained_model, test_loader)
```


10. Pretrained model

Result

- 실행 테스트(구동1) 후, 노트북 병합을 위해 최종 파일로 다시 실행(구동2)하니 결과가 다르게
나옴
 - 구동 1의 경우 검증과 테스트 성능이 비슷하게 나온 반면, 구동2의 경우 검증에서 낮은 성능을 보이고 테스트에서 높은 성능을 보임
- 같은 조건에서 실행을 하더라도 매번 다른 결과가 나올 수 있음을 알게 되었음

Resnet34 result (epoch=10) 구동 1		
	loss	accuracy
validation	0.6891	0.3724
test	0.7326	0.3375

Resnet34 result (epoch=10) 구동 2		
	loss	accuracy
validation	1.0225	0.1984
test	0.5102	0.4714

11. Visualization by CAM (3pts)

Since this is Resnet, meaning that it has global average pooling. Let's make use of it and visualize using Class Activation Map (although we skipped this part in the class but you can easily understand it and there are lots of code that you can refer.)

```
3 resnet = resnet34(pretrained=True)
4 param = list(resnet.parameters())[-2]
5 weight = param.detach().numpy()
6
7 class ExtractFeature(nn.Module):
8     def __init__(self):
9         super(ExtractFeature, self).__init__()
10        self.feature = nn.Sequential(
11            *list(resnet.children())[0:-2])
12
13    def forward(self, x):
14        out = self.feature(x)
15        return out
16
17
18 def getCam(weight, feature):
19
20     # 가중치와 내적 계산
21     map = weight.dot(feature)
22     # 피쳐맵과 크기가 일치하도록 변경
23     map = map.reshape(7, 7)
24     # 정규화
25     map = map - np.min(map)
26     map = map / np.max(map)
27     # 피쳐맵 정수 변환
28     map = np.uint8(255 * map)
29
30     heatmap = cv2.applyColorMap(map, cv2.COLORMAP_JET)
31
32     return cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB) # 색상 채널 변환
33
34 net = ExtractFeature()
35 raw = cv2.cvtColor(cv2.imread('/content/input/train/Abra/5c9ca320656b4f2fadea7aefeb80da53.jpg'), cv2.COLOR_BGR2RGB)
36 resizedImage = cv2.resize(raw, (224, 224))
37 inputImage = torch.tensor(resizedImage, dtype=torch.float)
38 inputImage = inputImage.permute(2, 1, 0)
39 inputImage = inputImage.view(1, 3, 224, 224)
40 plt.imshow(resizedImage)
```

12 Visualization by GradCAM (3pts)

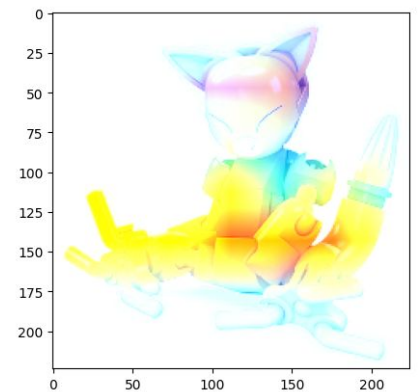
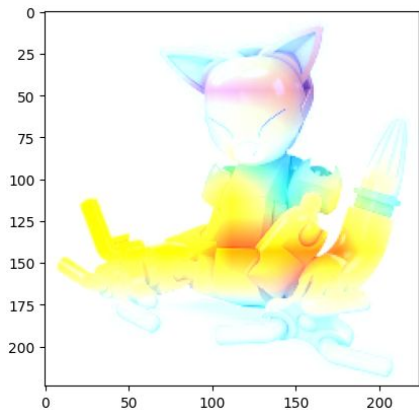
While CAM only allows us to visualize the layer connected to the global average pooling, GradCAM allows to visualize any layers. So try to visualize other layers as well by using GradCAM. You can do some search for the GradCAM code and use it.

```
1 class Flatten(nn.Module):
2     def __init__(self):
3         super(Flatten, self).__init__()
4     def forward(self, x):
5         return x.view(x.size(0), -1)
6
7 featuresFn = nn.Sequential(*list(resnet.children())[:-2])
8 classifierFn = nn.Sequential(*(list(resnet.children())[-2:-1] + [Flatten()] + list(resnet.children())[-1:]))
9
10 # 피쳐 추출
11 feats = featuresFn(inputImage)
12
13 out = classifierFn(feats)
14
15 c_score = out[0, 94]
16
17 # 선택 class에 대한 grad 계산
18 grads = torch.autograd.grad(c_score, feats)
19 # 가중치와 특성 내적 계산
20 w = grads[0][0].mean(-1).mean(-1)
21
22 map = torch.matmul(w, feats.view(512, 7*7))
23 map = map.view(7, 7).cpu().detach().numpy()
24 # map = np.maximum(map, 0)
25 plt.imshow(map)
```

11-12. CAM(우측 상단) & Grad CAM(우측 하단)

Result

- 오픈소스에 있는 여러 코드를 봤는데, 수정해도 실행 안 되는 것이 많아 가장 간단하고 실행 잘 되는 것으로 가져옴
- CAM의 단점을 보완하기 위해 나온 것이 Grad CAM이라고 하는데, 간단하게 구현된 코드를 실행해봐서 그런지 더 간단해졌다거나 이미지 출력이 눈에 띄게 좋아졌다는 느낌은 안 들었음
- 많이 활용해보고 단점을 직접 느껴야 차이를 알 수 있을 것이라 생각됨



결론 및 고찰

1. 수업에서 배웠던 딥러닝에 사용되는 다양한 모델을 직접 코드로 구현해 볼 수 있어 어려웠지만 유익했다.
 - 코드를 직접 구성해야하니 어떻게 사용하는지 더 자세히 찾아보게 됨, 퀴즈 만들 때 보다 더 자세히 보게 된다.
2. train 코드를 수정하거나 새로 만들어서 구동해보지는 못하였는데, 실력이 된다면 수정하여 train accuracy & loss를 보고싶다.
 - train 성능이 중요한 지표인데, val과 test 성능만 확인할 수 있어 아쉬웠다.
3. 노트북으로 한 번에 다 수행하기에는 런타임 오류가 있어 코드를 각각 수행하고 완성된 후 하나의 노트북에 옮겨 실행했는데, 실행할 때 마다 성능 결과가 다르게 나와서 ppt파일 수정하기 번거로웠다.
 - 실행할 때 마다 split이 다르게 되어서 그랬을 것 같음, 다음엔 돌아가는대로 옮겨서 한 번에 돌려야겠다.
 - split이 다르게 설정되더라도 데이터셋의 양이 많았다면 보다 적은 차이를 보였을 것이라 생각된다.