Dog Classification Kaggle Competition

2016145101 안장환

Summary

이번 프로젝트는 '딥러닝 기초 및 응용' 수업 기말고사로 진행되었고, 주어진 dog image dataset에 대하여 120가지의 종으로 구분하는 task이다. 주어진 image dataset은 train dataset, validation dataset, test dataset으로 미리 분류되어 있었다. 모델 구성은 첫 번째로 직접 설계한 CNN 모델인 DogNet으로 학습을 진행했다. 그리고 transfer learning을 적용하기 위해서 좋은 성능을 보여 널리 알려진 VGG16, Inception v3, Densenet 모델로 classification task를 진행했다. 그리고 이 모델들의 hyperparameter를 변경해보며 test accuracy와 validation accuracy를 측정해 성능을 비교했다.

1. DogNet

DogNet은 VGG16에서와 유사하도록 직접 설계한 모델로 convolution * 2 + maxpool을 두 번 진행한 뒤, convolution * 3 + maxpooling을 세 번 진행하는 모델이다. 그리고 그 뒤의 fully connected layer는 512 * 7 * 7개의 feature를 4096개로 줄여주고 그 후에 120개의 class probability로 계산해 준다.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
ReLU-2	[-1, 64, 224, 224]	0
Conv2d-3	[-1, 64, 224, 224]	36,928
ReLU-4	[-1, 64, 224, 224]	0
MaxPool2d-5	[-1, 64, 112, 112]	0
Conv2d-6	[-1, 128, 112, 112]	73,856
ReLU-7	[-1, 128, 112, 112]	0
Conv2d-8	[-1, 128, 112, 112]	147,584
ReLU-9	[-1, 128, 112, 112]	0
MaxPool2d-10	[-1, 128, 56, 56]	0
Conv2d-11	[-1, 256, 56, 56]	295,168
ReLU-12	[-1, 256, 56, 56]	0
Conv2d-13	[-1, 256, 56, 56]	590,080
ReLU-14	[-1, 256, 56, 56]	0
Conv2d-15	[-1, 256, 56, 56]	590,080
ReLU-16	[-1, 256, 56, 56]	0
MaxPool2d-17	[-1, 256, 28, 28]	0
Conv2d-18	[-1, 512, 28, 28]	1,180,160
ReLU-19	[-1, 512, 28, 28]	0
Conv2d-20	[-1, 512, 28, 28]	2,359,808
ReLU-21	[-1, 512, 28, 28]	0
Conv2d-22	[-1, 512, 28, 28]	2,359,808
ReLU-23	[-1, 512, 28, 28]	0
MaxPool2d-24	[-1, 512, 14, 14]	0
Conv2d-25	[-1, 512, 14, 14]	2,359,808
ReLU-26	[-1, 512, 14, 14]	0
Conv2d-27	[-1, 512, 14, 14]	2,359,808
ReLU-28	[-1, 512, 14, 14]	0
Conv2d-29	[-1, 512, 14, 14]	2,359,808
ReLU-30	[-1, 512, 14, 14]	0
MaxPool2d-31	[-1, 512, 7, 7]	0
Linear-32	[-1, 4096]	102,764,544
ReLU-33	[-1, 4096]	0
Dropout-34	[-1, 4096]	0
Linear-35	[-1, 120]	491,640

Total params: 117,970,872
Trainable params: 117,970,872
Non-trainable params: 0

Input size (MB): 0.57
Forward/backward pass size (MB): 218.49
Params size (MB): 450.02
Estimated Total Size (MB): 669.09

Fig 1. DogNet model specification

약 1억 2천만개의 parameter가 사용되는 것을 Fig 1을 통해 확인할 수 있다. Convolution kernel에 사용되는 parameter에 비해 fully connected layer에서 input과 output를 맞춰주기 위해 사용되는 matrix에서 약 1억 개 정도로 전체 parameter의 개수 중 약 80%의 비중을 차지하는 것을 확인할 수 있다.

```
transform_train = transforms.Compose([
    transforms.RandomResizedCrop((224,224)),
    transforms.RandomRotation((-90,90)),
    transforms.RandomAffine((-72,72)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(), #이미지를 tensor자료형으로 변환
    transforms.Normalize(
                mean=torch.tensor([0.485, 0.456, 0.406]),
                std=torch.tensor([0.229, 0.224, 0.225]))]) # 정규화
```

Fig 2. Data augmentation

모델을 학습시키는데 많은 시간이 소요되므로 다른 방법론을 적용할 때 어떻게, 얼마나 향상되는지를 모든 경우에 대해 실험할 수는 없었다. 따라서 overfitting이 발생할 수 있는 문제를 미리 해결해주기 위해서 첫 모델 실험부터 data augmentation을 train set와 validation set에 해주었다. Pytorch framework에서 제공해주는 RandomResizedCrop, RandomRotation, RandomAffine, RandomHorizontalFlip을 해주었다.

Loss function은 sigmoid과 cross entropy 계산을 같이 해주는 pytorch의 CrossEntropyLoss를 사용하였다. Optimizer는 "Adagrad", learning rate = 0.001로 실험을 진행하였다. Batch size는 gpu memory가 최대한 허용하는 범위에서 값을 설정하였고, 128로 해주었을 때 'out of memory' error가 발생하여 64로 설정해주었다.

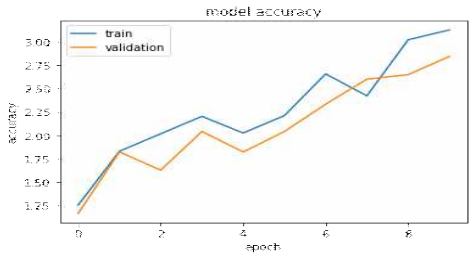

Fig 3. DogNet accuracy

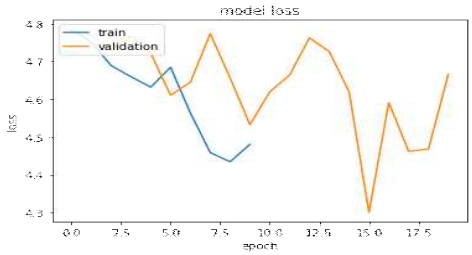

Fig 4. DogNet loss

	Train	Validation
Accuracy	**3.12%**	**2.84%**
Loss	4.4348	4.3

Table 1. DogNet learning accuracy / loss

Fig 5. VGG16 transfer learning Kaggle submission

직접 설계하고 학습시킨 위 모델로는 여러 epoch을 거쳐도 accuracy가 크게 향상되지 않았다. Fig 3과 Table 1에서 볼 수 있듯이 한 자리 수의 % 정확도를 보였기 때문에 더 이상의 학습에는 의미가 없을 것으로 판단하여 학습진행을 중단하였다.

2. Transfer learning

Transfer learning이란 ImageNet과 같은 큰 데이터셋으로 모델을 미리 학습(pre-trained model)을 진행한 뒤에 실제 학습을 원하는 dataset으로 fine-tuning하는 것을 의미한다. 본 dog classification task에서는 VGG16과 Inception v3로 transfer learning을 수행해보았고, pre-trained 모델을 fine-tuning하여 성능을 비교했다.

2.1 VGG16

Transfer learning의 첫 번째로 VGG16으로 실험을 진행했다. VGG16 model의 input size는 $(3 * 224 * 224)$로 dog classification에서 사용하는 input size와 같았다. Data augmentation은 DogNet에서와 같다.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                 [-1, 1000]       4,097,000
================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.78
Params size (MB): 527.79
Estimated Total Size (MB): 747.15
----------------------------------------------------------------
```

Fig 6. VGG16 model layer specification

```
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Fig 7. Original VGG16 classifier

```
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=120, bias=True)
  )
)
```

Fig 8. VGG16 classifier for the task

VGG16은 1000개의 class로 구분하는 모델이기 때문에

classifier에서 120개의 class로 구분하도록 모델을 변경해주었다. Fig 7의 마지막 줄을 보면 기존의 VGG16 모델은 “out_features =1000“로 되어있지만, dog classifiation에 맞도록 설계한 모델 Fig 8에서는 ”out_features=120”로 바뀐 것을 확인할 수 있다.

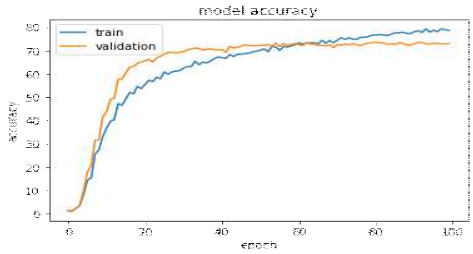

Fig 9. VGG16 transfer learning accuracy

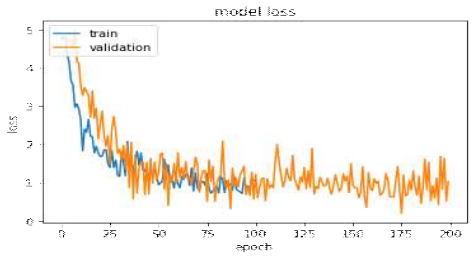

Fig 10. VGG16 transfer learning loss

	Train	Validation
Accuracy	**79.36 %**	**73.66 %**
Loss	0.7115	0.1906

Table 2. VGG16 transfer learning accuracy / loss

Fig 11. VGG16 transfer learning Kaggle submission

학습 epoch은 100epochs로 해주었고, Fig 9를 확인했을 때 train set에 대해서는 지속적으로 accuracy가 상승하지만 validation set에서 40epoch이후에는 성능에 큰 차이가 없는 것을 확인할 수 있다. Fig 10을 보면 overfitting문제가 발생하지는 않았고, Table 2을 통해 maximum train accuracy는 79.36%, maximum validation accuracy는 73.66%인 것을 볼 수 있다. 이 후 test set으로 만든 결과를 csv파일을 통해 kaggle에 제출한 결과 정확도가 0.73469(약 73.5%)로 나왔다. 이는 validation set에서 측정한 accuracy과 거의 같은 결과였다.

2.2 VGG16 fine tuning

```
model_2.classifier =  nn.Sequential(
        nn.Linear(512 * 7 * 7, 4096),
        nn.ReLU(True),
        nn.Dropout(0.5),
        nn.Linear(4096, 2048),
        nn.ReLU(True),
        nn.Linear(2048, 512)),
        nn.ReLU(True),
        nn.Dropout(0.5),
        nn.Linear(512, 120),
        nn.Dropout(0.5)
        )
```

Fig 12. Fine tuned VGG16 classifier

2.1에서 VGG16 transfer learning 모델에서 73.5%의 성능을 보여주어 fine tuning을 통해 성능을 더 향상시키고자 하였다. 따라서 Fig 12에서처럼 4096에서 2048, 512, 120으로 축소시키며 class score를 얻고자 하였다. 이외의 모든 optimizer와 hyperparameter는 같게 실험하였다.

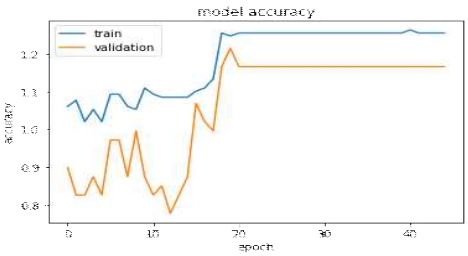

Fig 13. Fine tuned VGG16 accuracy

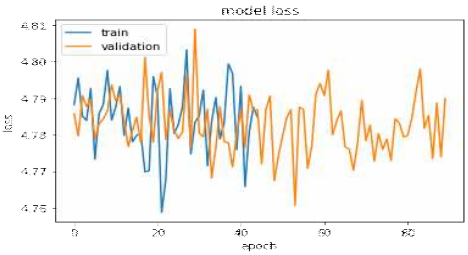

Fig 14. Fine tuned VGG16 loss

	Train	Validation
Accuracy	**1.26 %**	**1.21%**
Loss	4.7587	4.7607

Table 3. Fine tuned VGG16 accuracy / loss

Fine tuning을 한 모델은 2.1에서 보여준 성능에 더해 더 좋은 성능을 나타낼 것으로 기대했지만 전혀 학습이 진행되지 않는 결과를 보였다.

2.3 Inception v3

```
model.aux_logits = False
model.fc = nn.Linear(2048, 120)
```

Fig 15. Inception v3 c:assifier

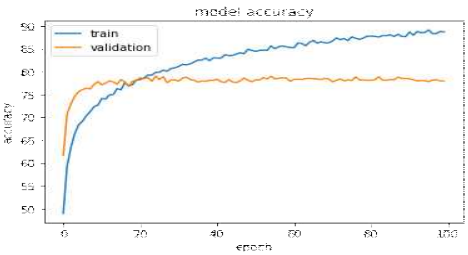

Fig 16. Inception v3 accuracy

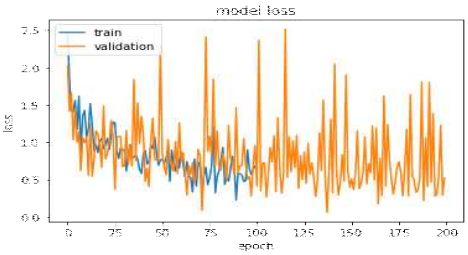

Fig 17. Inception v3 loss

	Train	Validation
Accuracy	**89.1 %**	**79%**
Loss	0.2267	0.0625

Table 4. Inception v3 accuracy / loss

Fig 18. Inception v3 Kaggle submission

Inception v3 마지막 classifier부분에서 Fig 15처럼 120 class에 맞도록 변경해주었다. 그리고 Inception v3 모델은 input image size가 299 * 299이므로 transformer에서 input 할 dog image size를 resize해주었다.

이 모델에서는 train accuracy는 계속해서 상승하지만 20epoch정도 되었을 때, validation accuracy는 수렴하는 것을 Fig 16에서 확인할 수 있다.

Test set을 활용하여 output을 만들고 kaggle에 제출하여 accuracy를 확인한 결과 약 78.5%의 성능을 얻을 수 있는 것을 확인했다. 이 역시도 validadtion set에서 보인 성능과 매우 유사하다는 것을 알 수 있다.

2.4 Inception v3 (Fine tuning with He initialization)

```
model.aux_logits = False

linear1 = nn.Linear(2048, 512)
linear2 = nn.Linear(512, 120)

nn.init.kaiming_uniform_(linear1.weight)
nn.init.kaiming_uniform_(linear2.weight)

model.fc =  nn.Sequential(
            linear1,
            nn.ReLU(True),
            nn.Dropout(0.5),
            linear2,
            nn.Dropout(0.5),
            )
```

Fig 19. Inception v3 (He) c:assifier

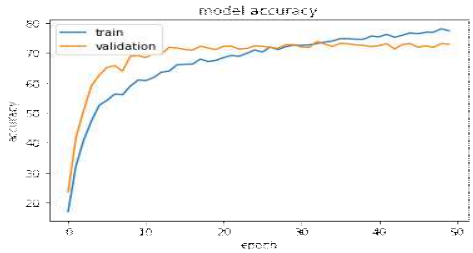

Fig 20. Inception v3 (He) accuracy

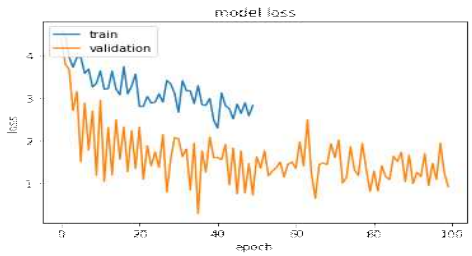

Fig 21. Inception v3 (He) loss

	Train	Validation
Accuracy	**78.1 %**	**73.8%**
Loss	2.2894	0.2811

Table 5. Inception v3 (He) accuracy / loss

Fig 22. Fine tuned Inception v3 Kaggle submission

2.3에서 구성한 모델이 VGG16보다 우수한 성능을 보였기 때문에 이를 더 향상시키고자 weight initialization과 fine tuning을 해보았다. Fully connected layer 2048에서 바로 120으로 줄이는 2.3과 달리 layer 하나를 더 추가하여 2048, 512, 120으로 단계적으로 줄였다. 그리고 ReLU nonliearity를 사용하기 때문에 weight initialization 중에서 He

initializaition을 해주었다. Fig 20을 보면 알 수 있듯이 20epoch정도가 되자 validation accuracy는 70% 정도에서 수렴하였고, 결과적으로는 기대했던 것과 달리 2.3에서보다 조금 성능이 하락한 73.25%(Table 5)의 test accuracy를 기록하였다.

2.5 Inception v3 & Adam optimizer

```
model.aux_logits = False

linear1 = nn.Linear(2048, 120)

nn.init.kaiming_uniform_(linear1.weight)

model.fc =  nn.Sequential(
            linear1
            )

trainer = Trainer(model, train_loader, "Adam",epoch_size=30,learning_rate=0.001)
```

Fig 23. Inception v3 (Adam) classifier

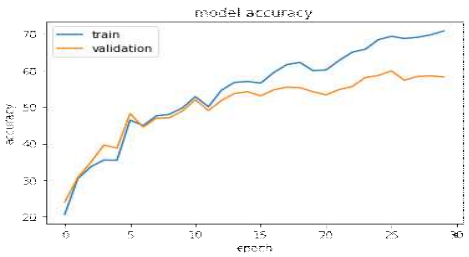

Fig 24. Inception v3 (Adam) accuracy

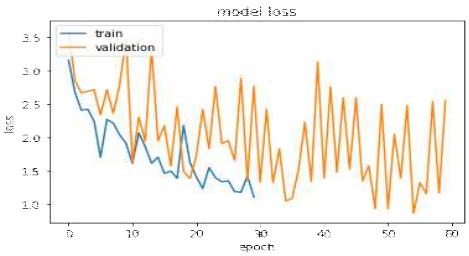

Fig 25. Inception v3 (Adam) loss

	Train	Validation
Accuracy	**70.8%**	**59.8%**
Loss	1.1056	0.8611

Table 5. Inception v3 (Adam) accuracy / loss

Fig 26. Inception v3 (Adam) Kaggle submission

2.5에서는 2.3에서 optimizer를 변경하는 방법을 취했다. 기존에 사용했던 Adagrad optimizer에서 Adam으로 변경하였고, hyperparameter, classifier는 2.3과 모두 같았다.

이 경우에는 Fig 24에서 학습속도가 느리고 Table 5에서 validation accuracy가 높지 않을 것을 확인했다. Adagrad에 비해 성능이 저하되는 것을 확인했다. 이로써 Adam optimizer가 가장 흔히 사용하는 optimizer로 알려져 있지만, 모든 경우에 해당하는 것은 아님을 알 수 있었다.

2.6 DenseNet

```
  (classifier): Linear(in_features=1664, out_features=1000, bias=True)
)

model.classifier = nn.Linear(1664,120)

(classifier): Linear(in_features=1664, out_features=120, bias=True)
```

Fig 27. DenseNet classifier

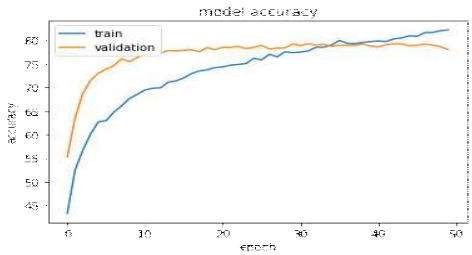

Fig 28. DenseNet accuracy

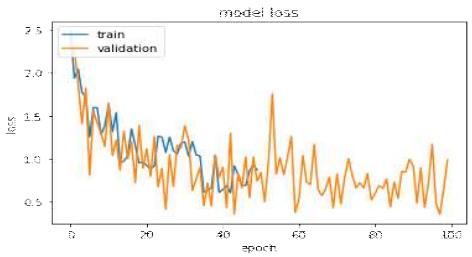

Fig 29. DenseNet loss

	Train	Validation
Accuracy	**82.2%**	**79.3%**
Loss	0.6048	0.3540

Table 5. DenseNet accuracy / loss

Fig 30. DenseNet Kaggle submission

Transfer learning의 마지막으로 Densenet을 학습시켰다. epoch 50, optimizer = Adagrad, learning rate = 0.001로 설정해주었다. Fig 27처럼 pretrained된 모델을 가져와 output을 120 class로 변경하고 학습을 진행했다. 정확도는 20 epoch정도에서 수렴하였다(Fig 28). Validation set에서 79.3%의 정확도를 보여 Inception v3보다 아주 조금 정확도

가 향상했다는 것을 확인했고, kaggle submission을 통해 측정한 결과 79.2%로 실험했던 모델 중 가장 높은 정확도를 달성하였다.

3. 결론

본 프로젝트를 수행하면서 직접 구성한 CNN model로는 훌륭한 classification 성능을 보이기 어려웠다. 따라서 여러 가지 Transfer learning을 통해 수행한 결과 약 80% 정도의 성능을 보이는 것을 알 수 있었다.

다만 기존의 pretrained된 모델을 참고하여 fine tuning을 통해 80%이상의 정확도를 보이도록 성능을 향상시키고자 하였지만 그 목적을 달성하지 못하였다. 하나의 모델을 실험하는 데에 수 시간이 소요되기 때문에 빠르게 변인을 조작하며 실험할 수 없었다.

참고문헌

(1) 연세대학교 '딥러닝 기초 및 응용' 수업, Lecture 7 "Data augmentation & CNN 모델들" 강의자료.

(2) 연세대학교 '딥러닝 기초 및 응용' 수업, Lecture 9 "Data augmentation & CNN 모델들" 강의자료.

(3) 연세대학교 '딥러닝 기초 및 응용' 수업, Lecture 10 "Data augmentation & CNN 모델들" 강의자료.

(4) https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

[Appendix]

1. DogNet

```
class DogNet(nn.Module):
    def __init__(self):
        super(DogNet, self).__init__()


        self.avgpool = nn.AvgPool2d((7,7)) #512 7 7

        self.features = nn.Sequential(
            #3 224 224
            nn.Conv2d(3, 64, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2),
            #64 112 112
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2),
            #128 56 56
            nn.Conv2d(128, 256, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2),
            #256 28 28
            nn.Conv2d(256, 512, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2),
            #512 14 14
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2)
        )
```

```
model = DogNet()
model.cuda()
trainer = Trainer(model, train_loader, "Adagrad",epoch_size=10,learning_rate=0.001)
trainer.train(val_loader) #training and validation
```

```
==========================================
tensor(4.6323, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 5/10 | ACC 2.024620
Val Acc | Epoch 5/10 | ACC 1.822157
==========================================
tensor(4.6850, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 6/10 | ACC 2.210884
Val Acc | Epoch 6/10 | ACC 2.040816
==========================================
tensor(4.5636, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 7/10 | ACC 2.656301
Val Acc | Epoch 7/10 | ACC 2.332361
==========================================
tensor(4.4592, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 8/10 | ACC 2.421445
Val Acc | Epoch 8/10 | ACC 2.599611
==========================================
tensor(4.4348, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 9/10 | ACC 3.020732
Val Acc | Epoch 9/10 | ACC 2.648202
==========================================
tensor(4.4810, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 10/10 | ACC 3.126013
Val Acc | Epoch 10/10 | ACC 2.842566
==========================================
```

```
# plot train, val accuracy
plt.plot(trainer.history['train_acc'])
plt.plot(trainer.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

print(f"Maximum Train Accuracy : {max(trainer.history['train_acc'])}")
print(f"Maximum Validation Accuracy : {max(trainer.history['val_acc'])}")

# plot train, val loss
plt.plot(trainer.history['train_loss'])
plt.plot(trainer.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

print(f"Minimum Train Loss : {min(trainer.history['train_loss'])}")
print(f"Minimum Validation Loss : {min(trainer.history['val_loss'])}")
```

2. VGG16

```
from torchvision import models

model_ = models.vgg16(pretrained=True)
```

```
linear1 = nn.Linear(512 * 7 * 7, 4096)
linear2 = nn.Linear(4096, 120)
nn.init.kaiming_uniform_(linear1.weight)
nn.init.kaiming_uniform_(linear2.weight)

model_.classifier =  nn.Sequential(
        linear1,
        nn.ReLU(True),
        nn.Dropout(0.5),
        linear2
        )
```

```
==========================================
tensor(0.7115, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 94/100 | ACC 79.348885
Val Acc | Epoch 94/100 | ACC 72.959183
==========================================
tensor(1.1206, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 95/100 | ACC 77.939751
Val Acc | Epoch 95/100 | ACC 72.837708
==========================================
tensor(0.8501, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 96/100 | ACC 78.741493
Val Acc | Epoch 96/100 | ACC 73.129250
==========================================
tensor(0.9101, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 97/100 | ACC 78.101715
Val Acc | Epoch 97/100 | ACC 73.032066
==========================================
tensor(0.8419, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 98/100 | ACC 79.365082
Val Acc | Epoch 98/100 | ACC 72.959183
==========================================
tensor(1.1255, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 99/100 | ACC 78.976357
Val Acc | Epoch 99/100 | ACC 72.886299
==========================================
tensor(1.0480, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 100/100 | ACC 78.701004
Val Acc | Epoch 100/100 | ACC 73.104950
==========================================
```

3. Inception v3

```
from torchvision import models

model = models.inception_v3(pretrained=True) #True는 weight만 가져오기
model.cuda()
```

```
model.aux_logits = False
model.fc = nn.Linear(2048, 120)


trainer = Trainer(model, train_loader, "Adagrad",epoch_size=100,learning_rate=0.001)
trainer.train(val_loader) #training and validation
```

```
==========================================
tensor(0.4743, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 94/100 | ACC 88.516357
Val Acc | Epoch 94/100 | ACC 78.036926
==========================================
tensor(0.4828, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 95/100 | ACC 88.597343
Val Acc | Epoch 95/100 | ACC 78.012634
==========================================
tensor(0.8975, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 96/100 | ACC 89.139946
Val Acc | Epoch 96/100 | ACC 77.818268
==========================================
tensor(0.6393, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 97/100 | ACC 88.289604
Val Acc | Epoch 97/100 | ACC 78.158409
==========================================
tensor(0.5676, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 98/100 | ACC 88.354393
Val Acc | Epoch 98/100 | ACC 78.207001
==========================================
tensor(0.6347, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 99/100 | ACC 88.807907
Val Acc | Epoch 99/100 | ACC 77.988342
==========================================
tensor(0.6837, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 100/100 | ACC 88.710724
Val Acc | Epoch 100/100 | ACC 77.891159
==========================================
```

4. Inception v3 (He initialization)

```
model = models.inception_v3(pretrained=True)
model.cuda()
```

```
model.aux_logits = False

linear1 = nn.Linear(2048, 512)
linear2 = nn.Linear(512, 120)

nn.init.kaiming_uniform_(linear1.weight)
nn.init.kaiming_uniform_(linear2.weight)

model.fc =  nn.Sequential(
            linear1,
            nn.ReLU(True),
            nn.Dropout(0.5),
            linear2,
            nn.Dropout(0.5),
            )
```

```
tensor(3.1196, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 42/50 | ACC 76.214775
Val Acc | Epoch 42/50 | ACC 73.129250
==========
tensor(2.8194, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 43/50 | ACC 75.251053
Val Acc | Epoch 43/50 | ACC 71.331390
==========
tensor(2.7459, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 44/50 | ACC 75.931328
Val Acc | Epoch 44/50 | ACC 72.813408
==========
tensor(2.5092, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 45/50 | ACC 76.676384
Val Acc | Epoch 45/50 | ACC 73.177841
==========
tensor(2.8560, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 46/50 | ACC 76.441536
Val Acc | Epoch 46/50 | ACC 71.914482
==========
tensor(2.6373, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 47/50 | ACC 76.911240
Val Acc | Epoch 47/50 | ACC 72.400391
==========
tensor(2.8885, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 48/50 | ACC 76.951736
Val Acc | Epoch 48/50 | ACC 71.865883
==========
tensor(2.5811, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 49/50 | ACC 78.085526
Val Acc | Epoch 49/50 | ACC 73.153549
==========
tensor(2.8276, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 50/50 | ACC 77.437645
Val Acc | Epoch 50/50 | ACC 72.934891
==========
```

5. Inception v3 (Adam optimizer)

```
from torchvision import models

model = models.inception_v3(pretrained=True)
model.cuda()
```

```
model.aux_logits = False

linear1 = nn.Linear(2048, 120)

nn.init.kaiming_uniform_(linear1.weight)

model.fc =  nn.Sequential(
            linear1
            )

trainer = Trainer(model, train_loader, "Adam",epoch_size=30,learning_rate=0.001)
trainer.train(val_loader) #training and validation
```

```
==========
tensor(1.5466, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 23/30 | ACC 64.957886
Val Acc | Epoch 23/30 | ACC 55.563652
==========
tensor(1.3966, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 24/30 | ACC 65.751541
Val Acc | Epoch 24/30 | ACC 58.017494
==========
tensor(1.3347, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 25/30 | ACC 68.375443
Val Acc | Epoch 25/30 | ACC 58.576286
==========
tensor(1.3491, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 26/30 | ACC 69.290573
Val Acc | Epoch 26/30 | ACC 59.863941
==========
tensor(1.1864, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 27/30 | ACC 68.691292
Val Acc | Epoch 27/30 | ACC 57.264336
==========
tensor(1.1774, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 28/30 | ACC 69.031425
Val Acc | Epoch 28/30 | ACC 58.333332
==========
tensor(1.4251, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 29/30 | ACC 69.687401
Val Acc | Epoch 29/30 | ACC 58.479107
==========
tensor(1.1056, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 30/30 | ACC 70.764503
Val Acc | Epoch 30/30 | ACC 58.187561
==========
```

6. DenseNet

```
model = models.densenet169(pretrained=True)
model.cuda()
```

```
model.classifier = nn.Linear(1664,120)
```

```
tensor(0.6048, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 43/50 | ACC 80.296410
Val Acc | Epoch 43/50 | ACC 79.227409
==========
tensor(0.9200, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 44/50 | ACC 80.490768
Val Acc | Epoch 44/50 | ACC 79.227409
==========
tensor(0.8159, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 45/50 | ACC 80.944283
Val Acc | Epoch 45/50 | ACC 78.838676
==========
tensor(0.6933, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 46/50 | ACC 80.822807
Val Acc | Epoch 46/50 | ACC 78.960159
==========
tensor(0.6892, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 47/50 | ACC 81.616455
Val Acc | Epoch 47/50 | ACC 79.178818
==========
tensor(0.8574, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 48/50 | ACC 81.656952
Val Acc | Epoch 48/50 | ACC 79.008743
==========
tensor(0.9296, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 49/50 | ACC 82.045677
Val Acc | Epoch 49/50 | ACC 78.644318
==========
tensor(0.8704, device='cuda:0', grad_fn=<NllLossBackward0>)
Train Acc | Epoch 50/50 | ACC 82.199547
Val Acc | Epoch 50/50 | ACC 78.061226
==========
```