<8~20 page 이내로 작성>

2022 년도 전자공학부 캡스톤디자인2 보고서

팀명	잘생긴 아이들		
과제명	강아지 코디네이터		
구성원	학과(전공)	학번	이름
	전자공학부	2017007456	주상현
	전자공학부	2017006753	신준하
	전자공학부	2017007692	황순규
과제유형	HW ( ) SW (O) HW/SW ( )	지도교수	이민석(인)
과제개요	인터넷 쇼핑이 일상이 된 지금 강아지의 옷을 인터넷을 이용하여 구매를 하는 경우가 빈번해졌습니다. 그 중 옷이라는 특성에 의해 직접 입어보지 못하여 본인의 강아지가 입었을 경우 어떤 이미지일지 상상하기가 힘들고, 구매를 했을 시 마음에 들지 않아 반품 혹은 환불을 하는 경우가 발생할 수 있습니다. 따라서 이 과제를 통하여 이 문제를 해결하고자 합니다.		

1. 과제의 목표 및 개요

저희 캡스톤의 목표는 아무것도 입고 있지 않은 강아지 이미지를 input으로 설정합니다. 딥러닝 모델 중 cycle GAN을 활용해 강아지가 옷을 입었을 때의 이미지를 학습시켜, input 이미지가 옷을 입었을 때는 어떤 모습일지 이미지로 만들어 내는 것입니다. 이를 위하여, 순수한 강아지 이미지를 1200장, 강아지가 옷을 입고 있는 이미지를 총 1500장 정도를 모아 데이터 셋을 구성하였습니다. 이 때, 강아지가 옷을 입었을 때의 데이터 중 다른 종류의 옷들이 많았기에 이를 옷의 색과 특색에 따라 구분 지었습니다. 그리고 이를 이용하여 GAN을 학습시켜 Model을 만들었습니다.

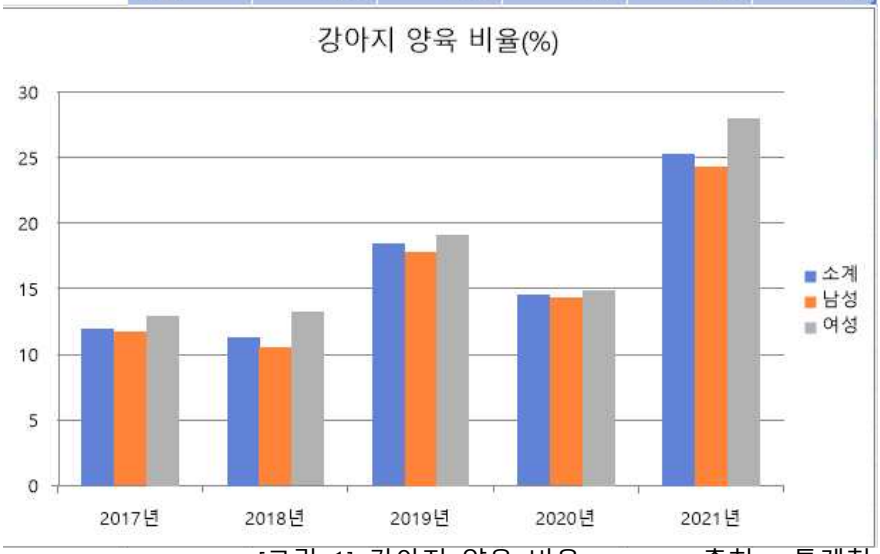
GAN은 Generator와 Discriminator로 이루어져 Generator에서 새로운 강아지 이미지를 만들어 내면, Discriminator에서 원래 있던 이미지와 비교를 하면서 점점 실제 강아지가 옷을 입고 있는 이미지와 가까워지도록 만드는 방식입니다.

이를 이용하여, 순수한 강아지 이미지에서 Generator model을 통하여 강아지가 옷을 입었을 때의 이미지를 연상할 수 있도록 구상하였습니다.

2. 과제의 필요성 및 독창성

(1) 과제의 필요성

	열1	2017년	2018년	2019년	2020년	2021년
전체	소계	12	11.3	18.4	14.5	25.3
성별	남성	11.7	10.5	17.8	14.3	24.3
	여성	12.9	13.2	19.1	14.9	28



[그림 1] 강아지 양육 비율 출처 - 통계청

해가 지날수록 강아지를 양육하는 사람들의 비율이 점점 증가하고 있습니다. 강아지가 가족이라는 인식이 늘어나게 되고, 강아지는 산책을 시켜야 하는 동물이기에 추운 겨울옷을 입혀 따뜻하게 해주거나, 옷을 입혀 예쁘게 꾸며주려고 하는 사람들 또한 증가하게 되었습니다. 이에 따라 인터넷으로 강아지 옷을 구매하는 사람도 점차 생겨났습니다. 하지만, 강아지 옷을 구매 할 때, 대부분의 사람들은 강아지를 데려가서 옷을 입혀보기 보단, 인터넷으로 구매를 한 후 입혀봅니다. 그렇게 되면, 사이즈가 맞지 않거나 기대한 모습이 아니어서 환불을 하거나 교환을 하게 되는 일이 발생하게 됩니다.

저희들은 본인들의 강아지 사진을 cycle GAN 모델에 넣어 강아지가 옷을 입었을 때의 이미지를 미리 볼 수 있다면 이러한 문제를 해결할 수 있지 않을까 하는 생각에 구상하게 되었습니다.

(2) 과제의 독창성

위의 필요성에서 이야기했듯이, 강아지 양육과 옷 구매량이 생김과 동시에 발생하는 다음과 같은 문제를 해결할 수 있는 방안을 생각해 보았습니다. 이를 해결하는 방법으로는 "내 강아지가 이 옷을 입으면 어떨까?"라는 생각을 가지게 되었습니다. 그러나 저희가 만들고자 하는 강아지 이미지 생성을 해주는 것은 아직 찾아볼 수 없었습니다. 사과를 오렌지로 바꾸거나 말을 얼룩말로 바꾸는 예시들이나 데이터는 많이 있었지만, 저희가 직접적으로 원하는 데이터는 찾아볼 수 없었습니다. 이에 따라 저희들이 직접 데이터를 모아 구성을 하고, cycleGAN 모델에 구현 하여 새로운 결과물을 만들어 내고자 하였습니다.

3. 캡스톤 디자인1에서의 변경점

(1) Data Set 변경

기존의 사람의 사진과 옷 사진들을 순수한 강아지 사진 1200장과 강아지가 옷을 입고 있는 사진 약 1500장으로 변경하였습니다. 이 때, 강아지가 옷을 입고 있는 사진은 모두 1500장이지만 진행 상황 중, 문제점을 발견하여 1500장을 비슷한 옷으로 분류하여 red 약 700장 stripe 400장 yellow 200장 black 200장 등으로 분류하였습니다.

(2) ViTon에서 CycleGAN으로 변경

사람을 대상으로 하여 VITon을 이용하려 했지만, 강아지로 변경됨에 따라 우리가 이용할 수 있는 모델을 찾아보던 중 CNN 기반의 생성 알고리즘인 Cycle GAN을 이용하여 강아지가 옷을 입었을 때의 이미지를 생성해내도록 하였습니다.

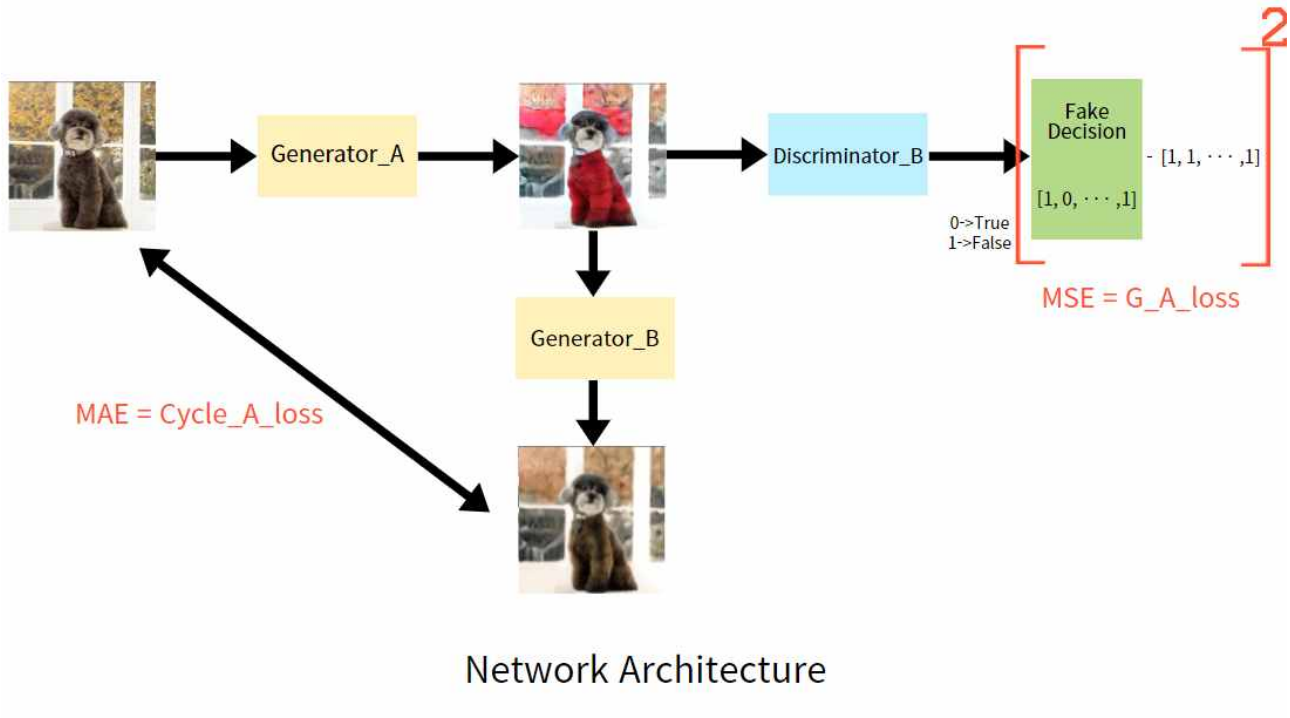
4. 진행 과정

	1월	2월	3월	4월	5월
주제 대상 및 모델 변경					
데이터 셋 수집					
코드 수정					
데이터 학습					
보고서 작성					

[그림 2] 캡스톤 디자인2 진행 과정

초기에 옷을 입고 있지 않은 강아지 이미지를 약 1200장 정도를 수집하였고, 강아지가 옷을 입고 있는 이미지를 약 1200장 정도를 수집하였습니다. 이를 이용하여 train 코드에 input으로 활용하여 train_model을 생성하였습니다.

학습 시키는 데이터 셋 중에서 옷을 입고 있는 데이터를 보았을 때, 이를 색상 또는 패턴에 따라 분류를 할 필요성을 느껴, 색상과 패턴에 따라 분류하였습니다. 분류된 데이터 셋은 빨간색 600장, 줄무늬 패턴 400장 등으로 나눌 수 있었으며, 데이터의 개수가 많은 것들을 위주로 학습시켜 model을 생성하였습니다. 모델 학습 시, epoch은 200, 300, 400 모두 해보았습니다.



[그림 3] Network 학습 구조

[그림 3]은 저희가 사용할 cycle GAN의 학습 구조를 도식화하여 나타낸 것입니다.
아래는 이 학습을 위한 training코드와 Generator, Discriminator의 모델입니다.

[1] model

```
class Generator(torch.nn.Module):
def __init__(self, input_dim, num_filter, output_dim, num_resnet):
super(Generator, self).__init__()

# Reflection padding
 self.pad = torch.nn.ReflectionPad2d(3)
# Encoder
 self.conv1 = ConvBlock(input_dim, num_filter, kernel_size=7, stride=1, padding=0)
self.conv2 = ConvBlock(num_filter, num_filter * 2)
self.conv3 = ConvBlock(num_filter * 2, num_filter * 4)
# Resnet blocks
 self.resnet_blocks = []
for i in range(num_resnet):
self.resnet_blocks.append(ResnetBlock(num_filter * 4))
self.resnet_blocks = torch.nn.Sequential(*self.resnet_blocks)
# Decoder
 self.deconv1 = DeconvBlock(num_filter * 4, num_filter * 2)
self.deconv2 = DeconvBlock(num_filter * 2, num_filter)
self.deconv3 = ConvBlock(num_filter, output_dim,
 kernel_size=7, stride=1, padding=0, activation='tanh', batch_norm=False)

def forward(self, x):
# Encoder
 enc1 = self.conv1(self.pad(x))
enc2 = self.conv2(enc1)
enc3 = self.conv3(enc2)
# Resnet blocks
 res = self.resnet_blocks(enc3)
# Decoder
 dec1 = self.deconv1(res)
dec2 = self.deconv2(dec1)
out = self.deconv3(self.pad(dec2))
```

```
return out

class Discriminator(torch.nn.Module):
def __init__(self, input_dim, num_filter, output_dim):
super(Discriminator, self).__init__()

conv1 = ConvBlock(input_dim, num_filter, kernel_size=4, stride=2, padding=1, activation='lrelu', batch_norm=False)
conv2 = ConvBlock(num_filter, num_filter * 2, kernel_size=4, stride=2, padding=1, activation='lrelu')
conv3 = ConvBlock(num_filter * 2, num_filter * 4, kernel_size=4, stride=2, padding=1, activation='lrelu')
conv4 = ConvBlock(num_filter * 4, num_filter * 8, kernel_size=4, stride=1, padding=1, activation='lrelu')
conv5 = ConvBlock(num_filter * 8, output_dim, kernel_size=4, stride=1, padding=1, activation='no_act', batch_norm=False)

self.conv_blocks = torch.nn.Sequential(
conv1,
 conv2,
 conv3,
 conv4,
 conv5
)
```

[2] train code

```
# Models
G_A = Generator(3, params.ngf, 3, params.num_resnet)
G_B = Generator(3, params.ngf, 3, params.num_resnet)
D_A = Discriminator(3, params.ndf, 1)
D_B = Discriminator(3, params.ndf, 1)
G_A.normal_weight_init(mean=0.0, std=0.02)
G_B.normal_weight_init(mean=0.0, std=0.02)
D_A.normal_weight_init(mean=0.0, std=0.02)
D_B.normal_weight_init(mean=0.0, std=0.02)
G_A.cuda()
G_B.cuda()
D_A.cuda()
D_B.cuda()

# Loss function
MSE_loss = torch.nn.MSELoss().cuda()
L1_loss = torch.nn.L1Loss().cuda()

# Training GAN
D_A_avg_losses = []
D_B_avg_losses = []
G_A_avg_losses = []
G_B_avg_losses = []
cycle_A_avg_losses = []
cycle_B_avg_losses = []

# training
 for i, (real_A, real_B) in enumerate(zip(train_data_loader_A, train_data_loader_B)):

# input image data
real_A = Variable(real_A.cuda())
real_B = Variable(real_B.cuda())

# Train generator G
# A -> B
 fake_B = G_A(real_A)
D_B_fake_decision = D_B(fake_B)
G_A_loss = MSE_loss(D_B_fake_decision, Variable(torch.ones(D_B_fake_decision.size()).cuda()))

# forward cycle loss
```

```
 recon_A = G_B(fake_B)
cycle_A_loss = L1_loss(recon_A, real_A) * params.lambdaA

# B -> A
 fake_A = G_B(real_B)
D_A_fake_decision = D_A(fake_A)
G_B_loss = MSE_loss(D_A_fake_decision, Variable(torch.ones(D_A_fake_decision.size()).cuda()))

# backward cycle loss
 recon_B = G_A(fake_A)
cycle_B_loss = L1_loss(recon_B, real_B) * params.lambdaB

# Back propagation
 G_loss = G_A_loss + G_B_loss + cycle_A_loss + cycle_B_loss
G_optimizer.zero_grad()
G_loss.backward()
G_optimizer.step()

# Train discriminator D_A
 D_A_real_decision = D_A(real_A)
D_A_real_loss = MSE_loss(D_A_real_decision, Variable(torch.ones(D_A_real_decision.size()).cuda()))
fake_A = fake_A_pool.query(fake_A)
D_A_fake_decision = D_A(fake_A)
D_A_fake_loss = MSE_loss(D_A_fake_decision, Variable(torch.zeros(D_A_fake_decision.size()).cuda()))

# Back propagation
 D_A_loss = (D_A_real_loss + D_A_fake_loss) * 0.5
 D_A_optimizer.zero_grad()
D_A_loss.backward()
D_A_optimizer.step()

# Train discriminator D_B
 D_B_real_decision = D_B(real_B)
D_B_real_loss = MSE_loss(D_B_real_decision, Variable(torch.ones(D_B_real_decision.size()).cuda()))
fake_B = fake_B_pool.query(fake_B)
D_B_fake_decision = D_B(fake_B)
D_B_fake_loss = MSE_loss(D_B_fake_decision, Variable(torch.zeros(D_B_fake_decision.size()).cuda()))

# Back propagation
 D_B_loss = (D_B_real_loss + D_B_fake_loss) * 0.5
 D_B_optimizer.zero_grad()
D_B_loss.backward()
D_B_optimizer.step()

# loss values
 D_A_losses.append(D_A_loss.data)
D_B_losses.append(D_B_loss.data)
G_A_losses.append(G_A_loss.data)
G_B_losses.append(G_B_loss.data)
cycle_A_losses.append(cycle_A_loss.data)
cycle_B_losses.append(cycle_B_loss.data)

print('Epoch [%d/%d], Step [%d/%d], D_A_loss: %.4f, D_B_loss: %.4f, G_A_loss: %.4f, G_B_loss: %.4f'
 % (epoch+1, params.num_epochs, i+1, len(train_data_loader_A), D_A_loss.data, D_B_loss.data, G_A_loss.data, G_B_loss.data))

# Show result for test image
 test_real_A = Variable(test_real_A_data.cuda())
test_fake_B = G_A(test_real_A)
test_recon_A = G_B(test_fake_B)

test_real_B = Variable(test_real_B_data.cuda())
test_fake_A = G_B(test_real_B)
```

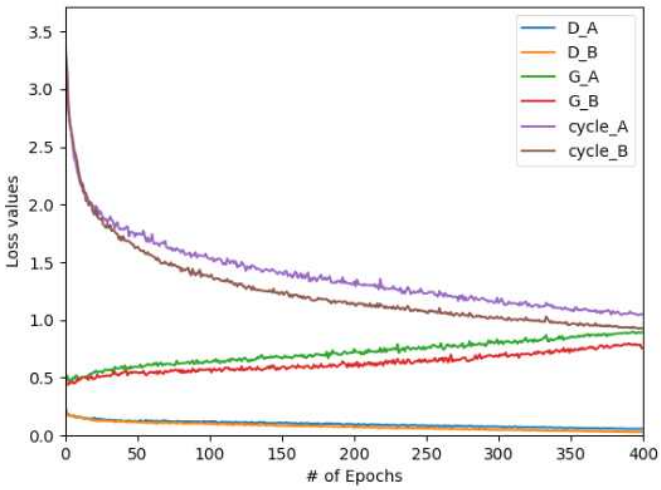
```
test_recon_B = G_A(test_fake_A)

utils.plot_train_result([test_real_A, test_real_B], [test_fake_B, test_fake_A], [test_recon_A, test_recon_B],
 epoch, save=True, save_dir=save_dir)

# log the images
 result_AtoB = np.concatenate((utils.to_np(test_real_A), utils.to_np(test_fake_B), utils.to_np(test_recon_A)), axis=3)
result_BtoA = np.concatenate((utils.to_np(test_real_B), utils.to_np(test_fake_A), utils.to_np(test_recon_B)), axis=3)

# Save trained parameters of model
torch.save(G_A.state_dict(), model_dir + 'generator_A_yellow_param.pkl')
torch.save(G_B.state_dict(), model_dir + 'generator_B_yellow_param.pkl')
torch.save(D_A.state_dict(), model_dir + 'discriminator_A_yellow_param.pkl')
torch.save(D_B.state_dict(), model_dir + 'discriminator_B_yellow_param.pkl')
```

[그림 4] 중간 결과

처음 train 코드를 이용하여 학습할 때의 데이터 셋은 순수한 강아지 이미지 1200장과 모든 종류의 옷을 모은 데이터 셋을 함께 학습시켰을 때의 결과 이미지입니다. 이 때, 옷을 입은 이미지가 분류되어 있지 않아 학습하는 부분에 있어 정확도가 떨어진다고 판단을 하게 되어 이 데이터 셋을 색상과 무늬에 따라 각각 분류를 하게 되었습니다.

5. 과제 결과

(1) red

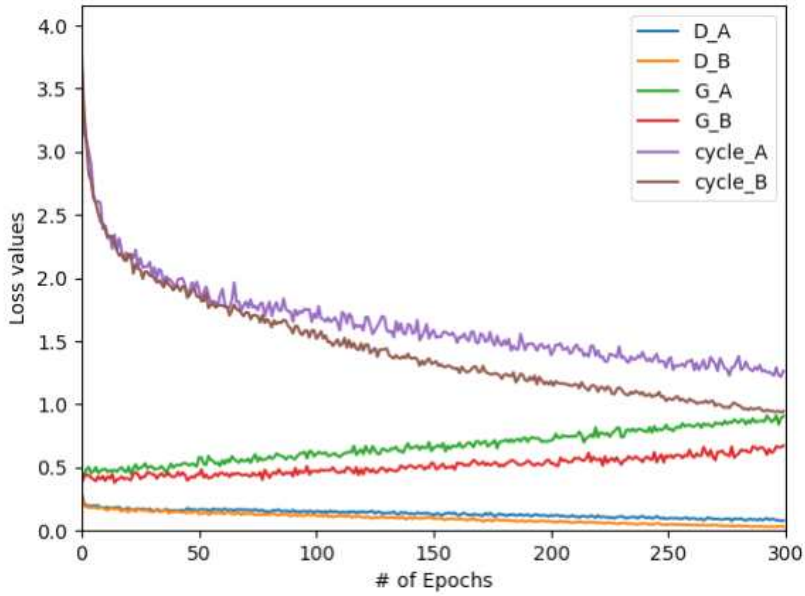


[그림 5] loss values

[그림 6] 결과 예시 사진

loss values가 Epoch가 증가 할수록 1에 점점 근사하는 모습을 볼 수 있고, 학습이 되고 잘 되고 있음을 보여줍니다. [그림 5]는 test한 사진들 중 예시 사진으로 강아지가 옷을 입었을 때의 예시를 보여주며 마지막 모습은 다시 원래대로 복구를 하였을 때의 모습을 보여줍니다.

(2) stripe

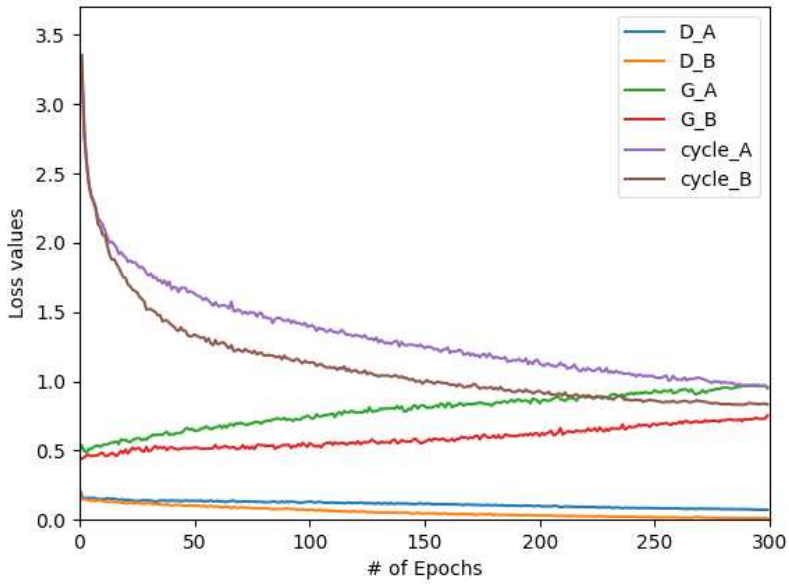


[그림 7] loss value

[그림 8] 결과 예시 사진

loss value가 Epoch가 증가하면서 1에 가까워지는 그래프가 되고는 있지만, 데이터셋이 부족하여 [그림 5]와는 차이를 보이고 있습니다. 그럼에도 약간의 줄무늬 패턴을 보여주는 것으로 보았을 때, 더 많은 데이터 셋을 구하여 학습을 시킨다면 더 정확한 그림을 얻을 수 있을 것으로 보입니다.

(3) yellow



[그림 9] value loss

[그림 10] 결과 예시 사진

중반까지는 loss value 가 1에 가까워지고는 있지만, 부족한 데이터셋에 의해 오히려 loss가 증가하는 것을 확인 할 수 있습니다. Overfitting이 된 것으로 보이는데, 이는 데이터셋을 늘리고 적당한 epoch으로 줄이면 적절한 모델을 만들 수 있을 것으로 보입니다.

6. 토의 및 개선방향

(1) 데이터셋의 다양성

저희가 구한 데이터 셋을 자세하게 살펴보겠습니다. 우선 순수한 강아지 사진을 보게 되면, 강아지의 종류가 아주 다양합니다. 학습을 시킬 때는 같이 이용을 하였지만, 더 정확한 패턴을 인식하기 위해서는 강아지도 분류를 할 필요가 있습니다. 또한 강아지가 옷을 입었을 때의 사진들이 상당히 다양하여, 각 옷들의 모양, 색상에 따른 각 각의 데이터 셋 구분이 필요할 것으로 보입니다. 특히 줄무늬와 같이 복잡한 패턴은 더 많은 데이터를 필요로 합니다.

(2) 패턴인식의 문제

흰 색의 강아지의 경우를 보면 색이 없어 정확하게 인식을 하지 못하는 것을 볼 수 있습니다. 이는 강아지의 dog-segmentation을 활용하여, 강아지의 신체부분을 이용하면 더 정확한 결과를 찾을 수 있을 것 같습니다.

(3) 판매사이트 적용

이를 판매 사이트에서 적용하기 위해서는 강아지의 색상(흰색, 갈색, 검은색 등) 별로 입었을 한 가지의 옷을 입었을 때의 사진을 연속촬영을 이용하여 각 1000장~2000장을 찍습니다. 각 사진들 별로 model을 만들 개인 강아지 사진을 input으로 활용하여 만들어낸 이미지를 구매자에게 보여준다면 판매 사이트에서 활용이 가능할 것으로 보입니다.

7. 참고문헌 및 부록

https://arxiv.org/pdf/1703.10593.pdf
https://arxiv.org/pdf/1512.03385v1.pdf