Back to the Future

GAN을 이용한 흑백사진 Colorization

KUBIG 장기프로젝트 |
12기 이나윤
13기 윤정현
14기 구은아

01. 프로젝트 소개

GAN을 이용한 흑백 사진 Colorization

; 과거의 흑백 사진에 색을 입히다.

02. 프로젝트 진행 과정

실전! GAN 프로젝트를 통한
GAN 기초 개념 학습

논문 reading

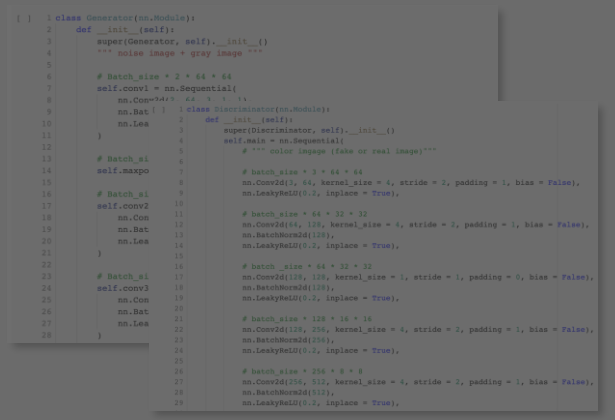
주제 선정 후 Base Code 구현

02. 프로젝트 진행 과정

실전! GAN 프로젝트를 통한
GAN 기초 개념 학습

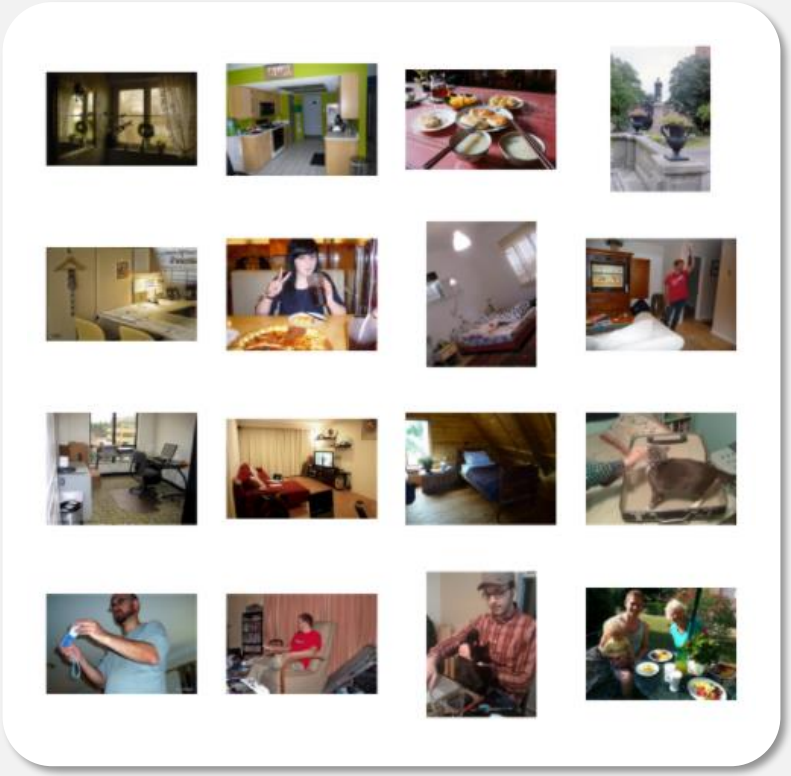
논문 reading

Parameter 조절과 Train의 반복



주제 선정 후 Base Code 구현

03. 모델 1 | 사용 데이터



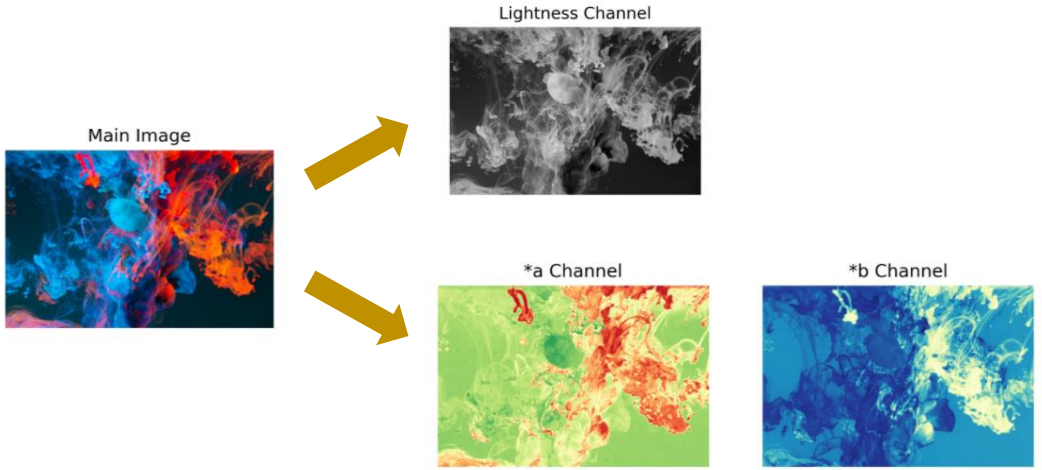
- **COCO Datasets**을 사용하여 모델 학습
- **8000개의 이미지**는 training set
  2000개의 이미지는 validation set으로 사용

시대별 이미지 데이터 구축 후 적용

민주화운동 | 일제강점기 |한국전쟁

03. 모델 1 | Data Loader

```
SIZE = 256
class ColorizationDataset(Dataset):
    def __init__(self, paths, split='train'):
        if split == 'train':
            self.transforms = transforms.Compose([
                transforms.Resize((SIZE, SIZE),  Image.BICUBIC),
                transforms.RandomHorizontalFlip(), # A little data augmentation!
            ])
        elif split == 'val':
            self.transforms = transforms.Resize((SIZE, SIZE),  Image.BICUBIC)

        self.split = split
        self.size = SIZE
        self.paths = paths

    def __getitem__(self, idx):
        img = Image.open(self.paths[idx]).convert("RGB")
        img = self.transforms(img)
        img = np.array(img)
        img_lab = rgb2lab(img).astype("float32") # Converting RGB to L*a*b
        img_lab = transforms.ToTensor()(img_lab)
        L = img_lab[[0], ...] / 50. - 1. # Between -1 and 1
        ab = img_lab[[1, 2], ...] / 110. # Between -1 and 1

        return {'L': L, 'ab': ab}

    def __len__(self):
        return len(self.paths)

def make_dataloaders(batch_size=16, n_workers=4, pin_memory=True, **kwargs): # A handy
    dataset = ColorizationDataset(**kwargs)
    dataloader = DataLoader(dataset, batch_size=batch_size, num_workers=n_workers,
                            pin_memory=pin_memory)
    return dataloader
```

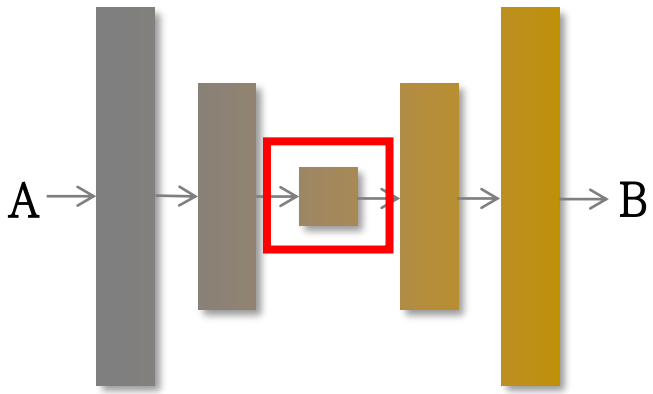


- 256*256 크기로 Resize
- Training set: **수평으로 뒤집은 버전** 추가
- RGB ➡ Lab Space로 변환 후,
  채널 L은 Input으로, 채널 a / b는 Target으로 분리

03. 모델 1 | pix2pix

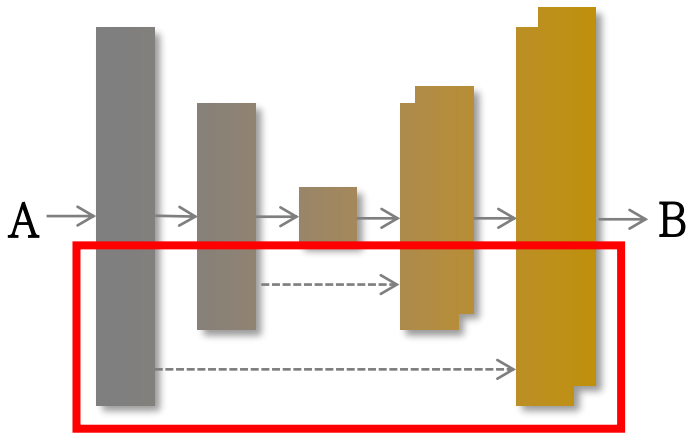
GAN 모델의 한 종류 | 이미지 간 형식 변환에 사용 | Image Translation 학습

03. 모델 1 | Generator 구조



Bottleneck 레이어에서
입력 이미지의 일부 정보를 잃어버림

Encoder-decoder

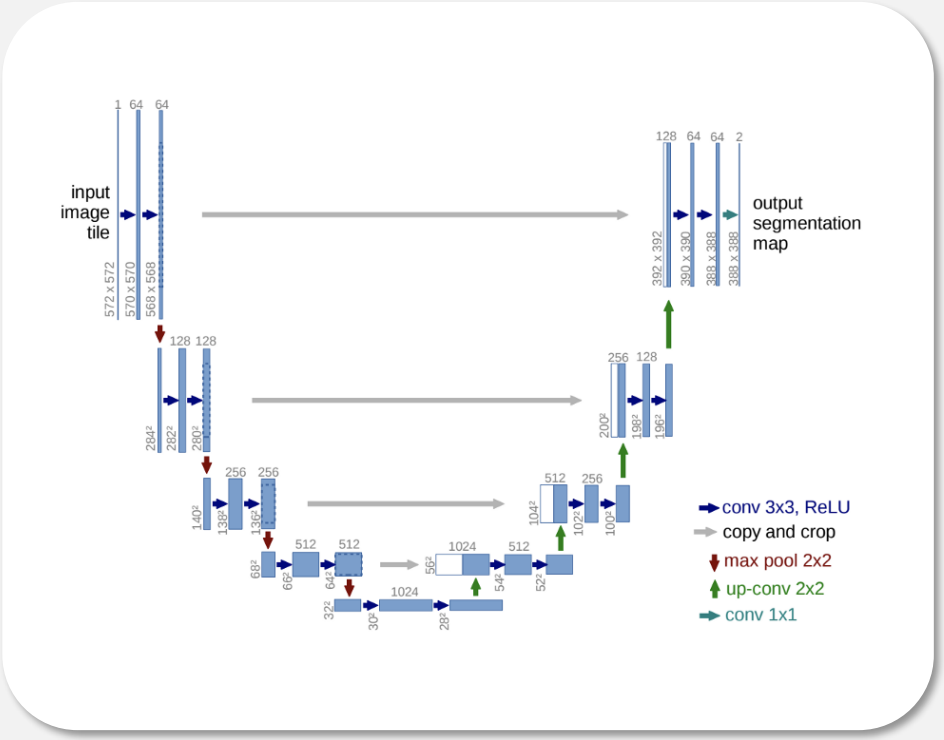


U-Net 구조 이용: Skip Connection
Bottleneck 레이어 우회

U-Net

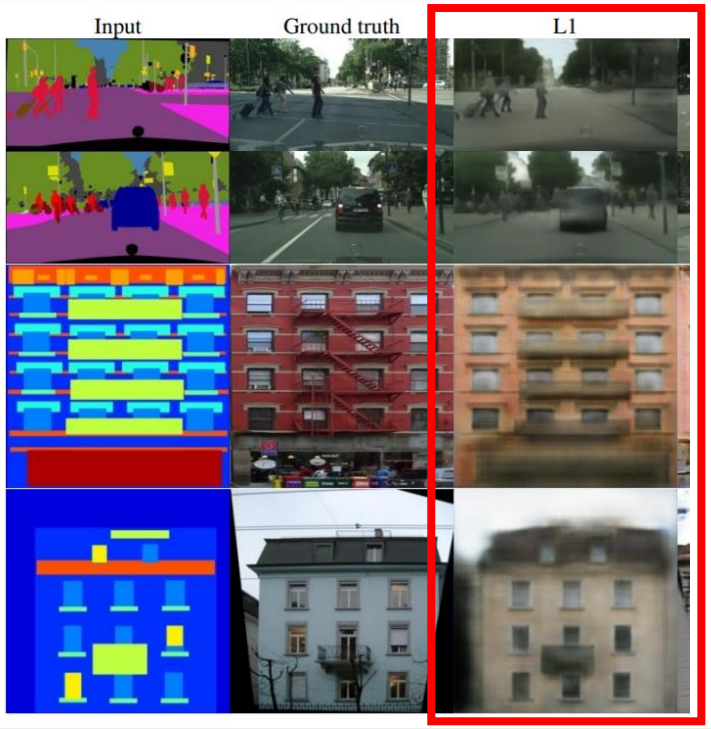
Image Translation: 고해상도의 입력이미지 ➡ 고해상도의 출력이미지

03. 모델 1 | Generator 구조

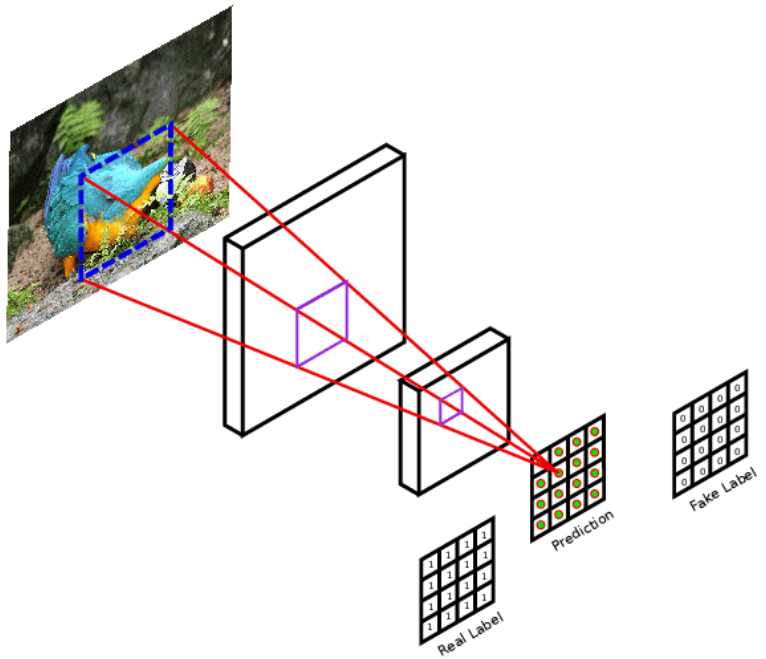



```python
class UnetBlock(nn.Module):
    def __init__(self, nf, ni, submodule=None, input_c=None, dropout=False,
                 innermost=False, outermost=False):
        super().__init__()
        self.outermost = outermost
        if input_c is None: input_c = nf
        downconv = nn.Conv2d(input_c, ni, kernel_size=4,
                             stride=2, padding=1, bias=False)
        downrelu = nn.LeakyReLU(0.2, True)
        downnorm = nn.BatchNorm2d(ni)
        uprelu = nn.ReLU(True)
        upnorm = nn.BatchNorm2d(nf)

        if outermost:
            upconv = nn.ConvTranspose2d(ni * 2, nf, kernel_size=4,
                                        stride=2, padding=1)
            down = [downconv]
            up = [uprelu, upconv, nn.Tanh()]
            model = down + [submodule] + up
        elif innermost:
            upconv = nn.ConvTranspose2d(ni, nf, kernel_size=4,
                                        stride=2, padding=1, bias=False)
            down = [downrelu, downconv]
            up = [uprelu, upconv, upnorm]
            model = down + up
        else:
            upconv = nn.ConvTranspose2d(ni * 2, nf, kernel_size=4,
                                        stride=2, padding=1, bias=False)
            down = [downrelu, downconv, downnorm]
            up = [uprelu, upconv, upnorm]
            if dropout: up += [nn.Dropout(0.5)]
            model = down + [submodule] + up
        self.model = nn.Sequential(*model)

    def forward(self, x):
        if self.outermost:
            return self.model(x)
        else:
            return torch.cat([x, self.model(x)], 1)

class Unet(nn.Module):
    def __init__(self, input_c=1, output_c=2, n_down=8, num_filters=64):
        super().__init__()
        unet_block = UnetBlock(num_filters * 8, num_filters * 8, innermost=True)
        for _ in range(n_down - 5):
            unet_block = UnetBlock(num_filters * 8, num_filters * 8, submodule=unet_block, dropout=True)
        out_filters = num_filters * 8
        for _ in range(3):
            unet_block = UnetBlock(out_filters // 2, out_filters, submodule=unet_block)
            out_filters //= 2
        self.model = UnetBlock(output_c, out_filters, input_c=input_c, submodule=unet_block, outermost=True)

    def forward(self, x):
        return self.model(x)
```

03. 모델 1 | Discriminator 구조



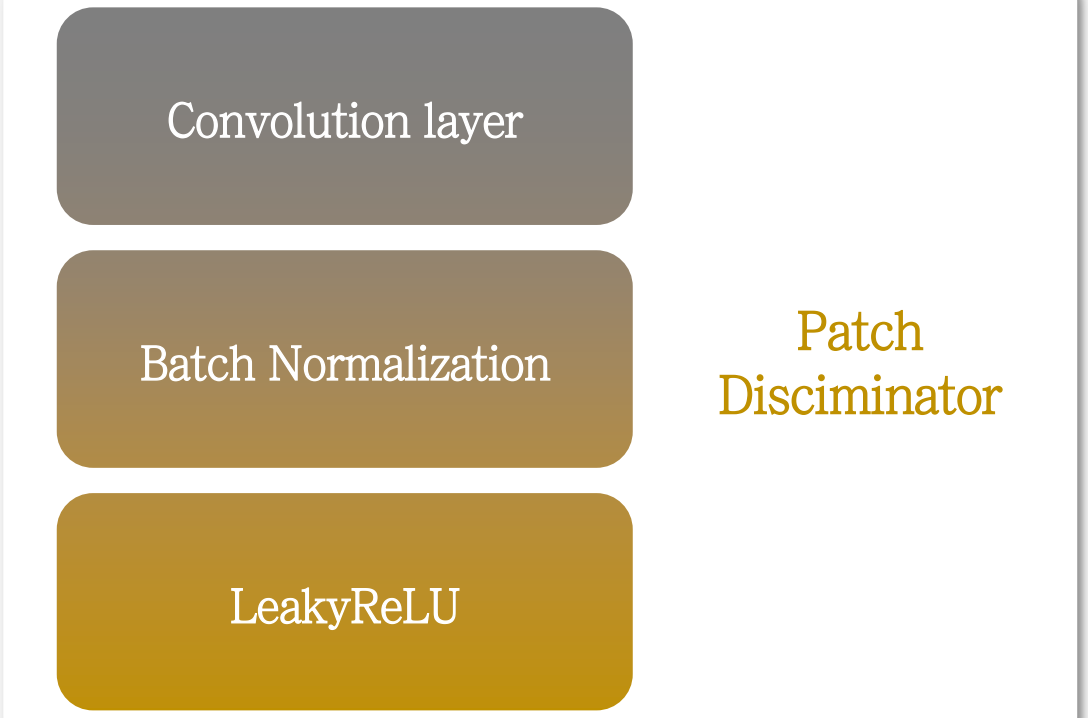
Generator의 최적화 함수: L1 loss 사용
➡ 영상의 저주파 선명도를 잘 Capture
Discriminator는 영상의 고주파 선명도에 집중
➡ 진위 여부를 판단하는 것이 유리 ➡ **PatchGAN 사용**



PatchGAN
N*N 크기의 patch의 fake/real 여부 판단
➡ 모든 정답을 평균하여 궁극적인 답 도출

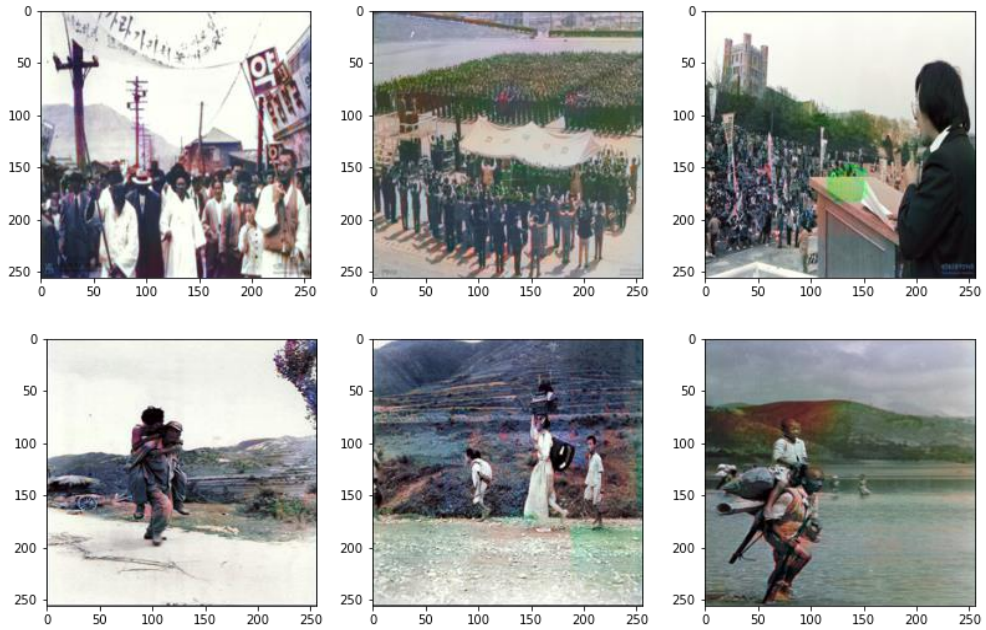
03. 모델 1 | Discriminator 구조

```
PatchDiscriminator(
  (model): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (3): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (4): Sequential(
      (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
    )
  )
)
```



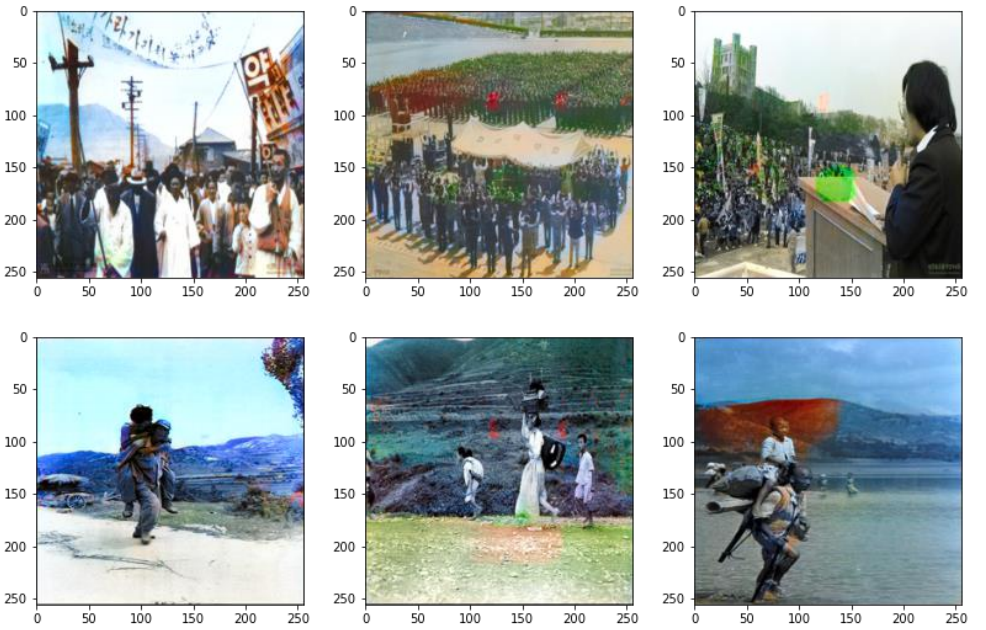
70*70 크기의 patch 사용하여 각 patch마다 fake/real 여부 판단에 대한 숫자를 얻음

03. 모델 1 | 결과



20 epoch

- 바랜 사진처럼 색이 많이 입혀지지 않음
- 20 epoch만 학습하는 것은 충분하지 않음을 확인



100 epoch

- 20 epoch만 학습했을 때보다 더 다양한 색깔로 컬러화 됨
- 인물보다 배경의 컬러화가 효과적
- 작은 크기의 다수의 인물 이미지를 더 자연스럽게 컬러화

04. 모델 2 | 사용 데이터

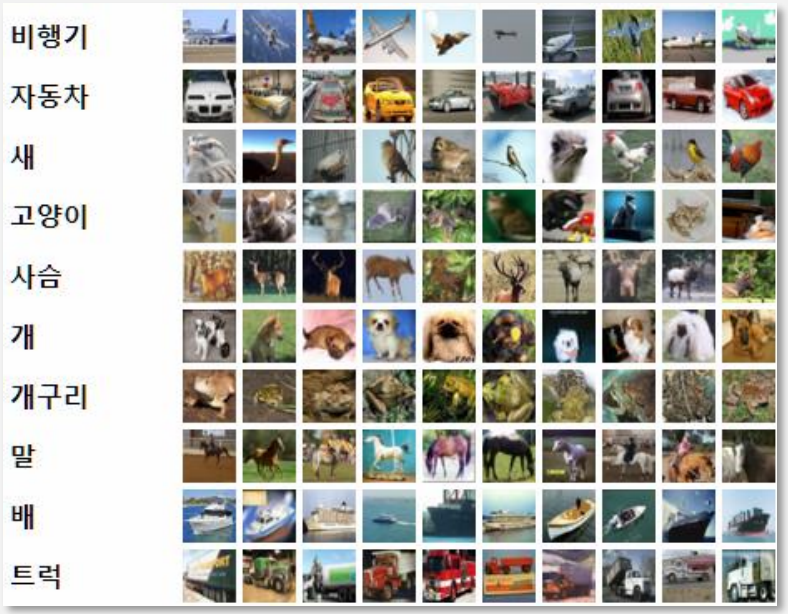


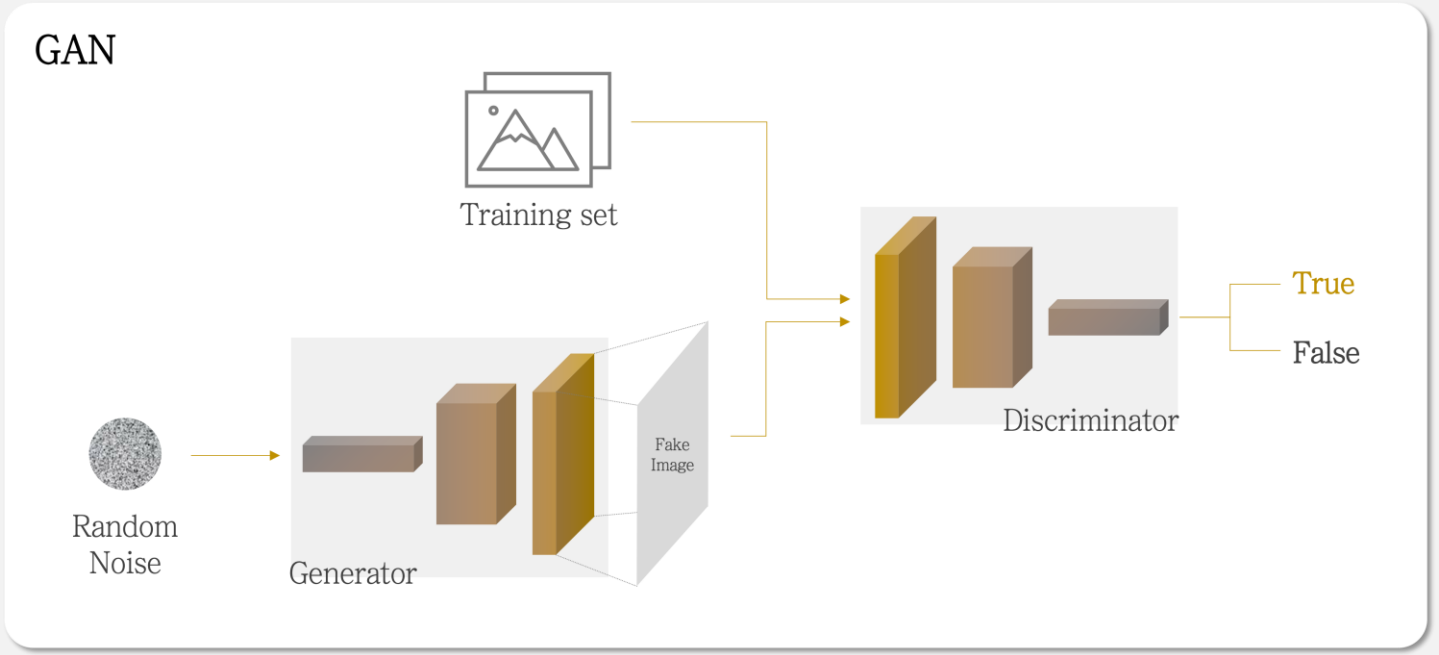
image-net의 이미지를 통해 train 진행

1,000만개 이상의 데이터 존재
총 1,000개의 class로 구성

시대별 이미지 데이터 구축 후 적용

민주화운동 | 일제강점기 |한국전쟁

04. 모델 2 | 사용 모델



Generator를 Unsuperivised가 아닌, Supervised로 따로 Pretrain 시키는 게 목적

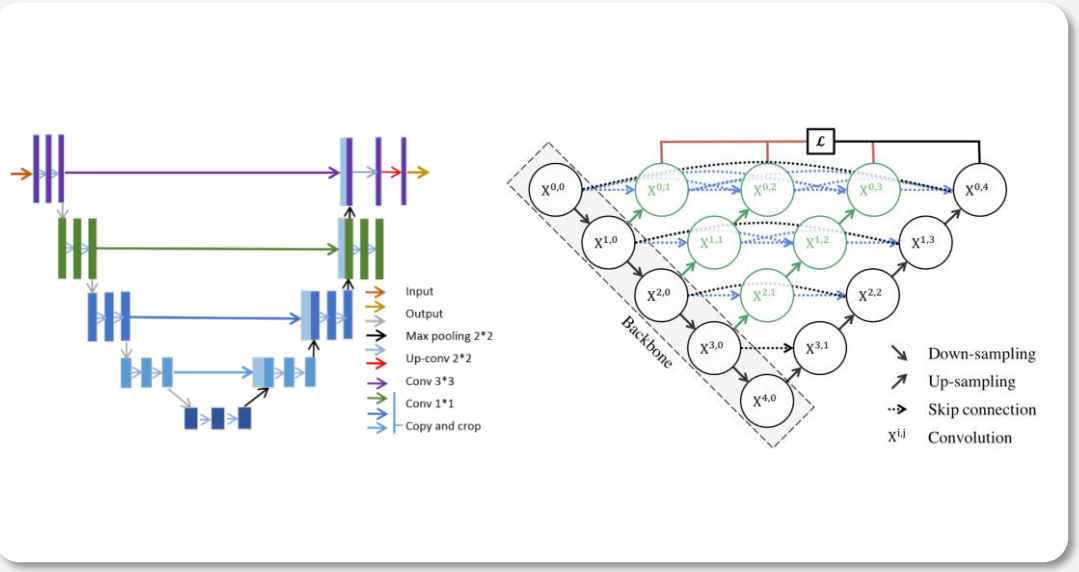
➡ GAN을 unsuperivised하게 훈련: blind leading the blind 문제 발생(아무것도 모르는 상태로 G와 D 시작)

➡ G: Backbone으로 Pretrained Model인 **RESNET18** 사용, L1 loss을 사용하여 Train

04. 모델 2 | 코드 소개

```
[ ]  from fastai.vision.learner import create_body
     from torchvision.models.resnet import resnet18
     from fastai.vision.models.unet import DynamicUnet

[ ]  def build_res_unet(n_input=1, n_output=2, size=256):
         device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
         body = create_body(resnet18, pretrained=True, n_in=n_input, cut=-2)
         net_G = DynamicUnet(body, n_output, (size, size)).to(device)
         return net_G

[ ]  net_G = build_res_unet(n_input=1, n_output=2, size=256)
     opt = optim.Adam(net_G.parameters(), lr=1e-4)
     criterion = nn.L1Loss()
```

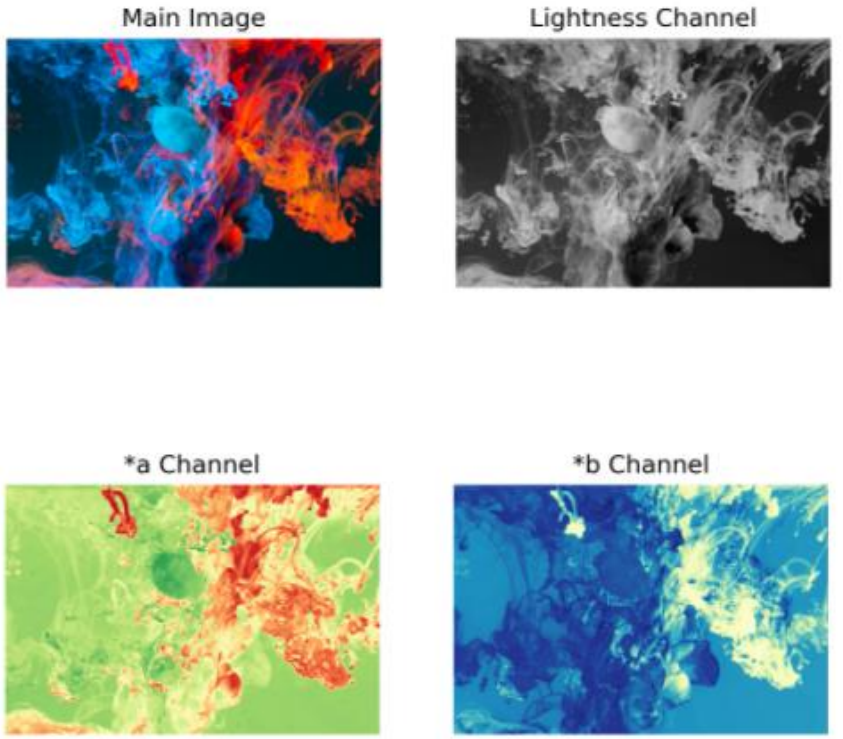
- Body로 설정한 resnet을 backbone으로 사용하게 설정
- Input size는 256, Output channel는 2로 설정
- Adam을 사용하여 최적 hyperparameter를 찾은 후 적용
- L1 Loss 사용



04. 모델 2 | 코드 소개

```
[ ]  def pretrain_generator(net_G, train_dl, opt, criterion, epochs):
         for e in range(epochs):
             loss_meter = AverageMeter()
             for data in tqdm(train_dl):
                 L, ab = data['L'].to(device), data['ab'].to(device)
                 preds = net_G(L)
                 loss = criterion(preds, ab)
                 opt.zero_grad()
                 loss.backward()
                 opt.step()

                 loss_meter.update(loss.item(), L.size(0))

             print(f"Epoch {e + 1}/{epochs}")
             print(f"L1 Loss: {loss_meter.avg:.5f}")
[ ]  pretrain_generator(net_G, train_dl, opt, criterion, 60)
```

- 훈련 데이터인 imagenet을 L과 ab로 나누어 generator training 진행
- 4시간 ~ 5시간 30분 소요

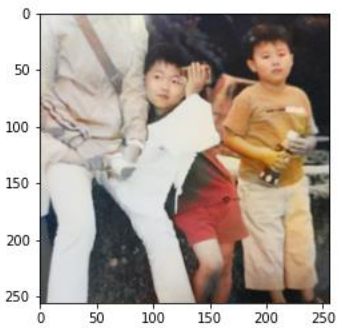
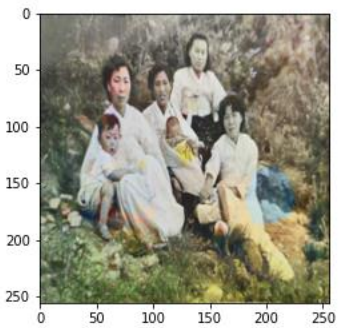
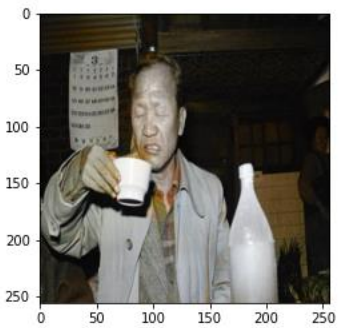
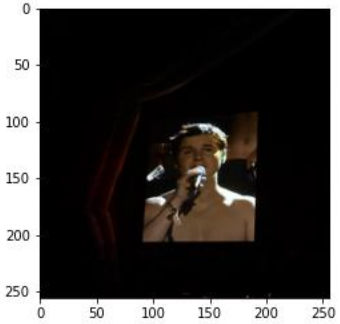
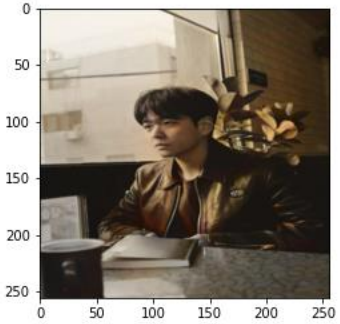
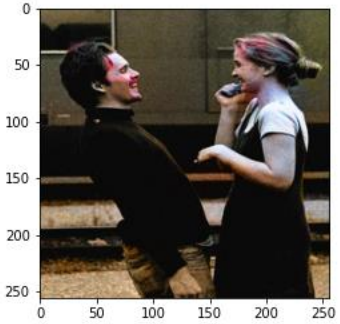
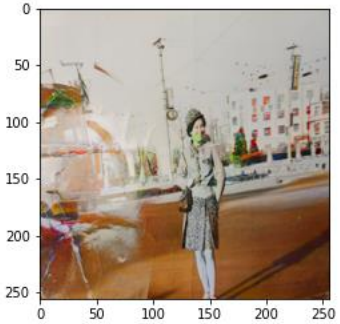
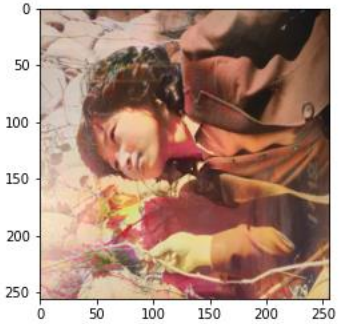
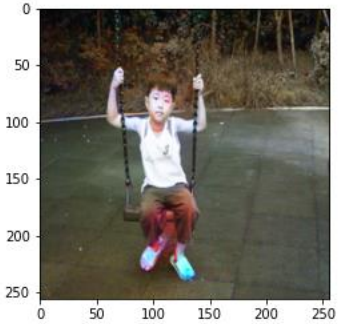


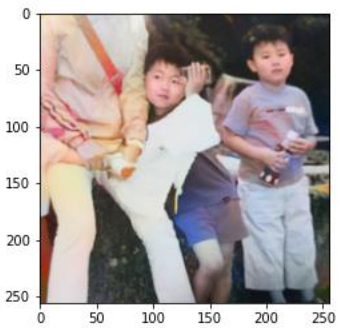
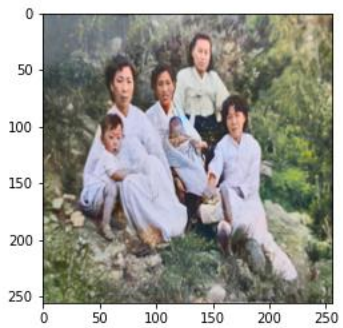
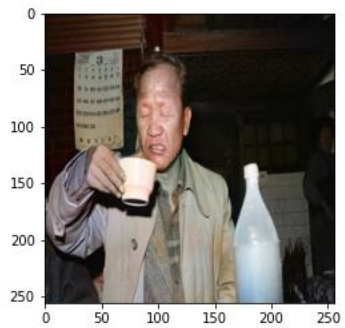
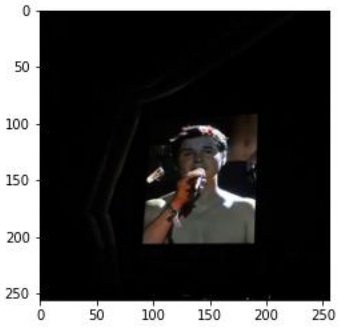
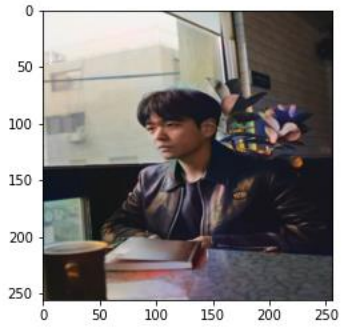
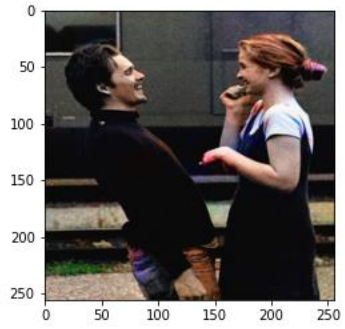
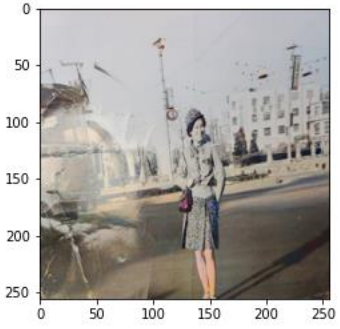
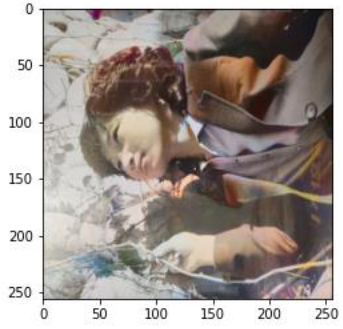
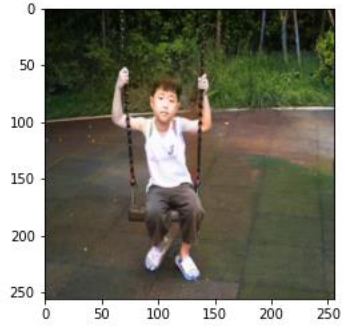
04. 모델 2 | 결과

04. 모델 2 | 결과





05. 후기

1. 한정적인 참고 자료와 주제
2. 더 다양한 이미지에 대한 colorization
3. 처음 다뤄보는 CV
4. 제한된 시간으로 인한 모델의 성능

Happy New Year!