CDAL Research

# Yosemite_summer2winter

## Image Classification, GAN, Diffusion Model

# CONTENTS

1

## Dataset

· Train, Test Dataset
· Feature 분석

2

## CNN Model

· Data Pipeline 설계
· Image Augmentation
· ResNet Transfer Learning

3

## GAN Model

· Data Pipeline & Image Augmentation
· Generator & Discriminator Class
· Train, Model save

4

## Diffusion Model

· Ran out of GPU

# Dataset

This dataset was obtained from UC Berkeley's official directory of [CycleGAN Datasets]

## File Path 구성

```
summer2winter_yosemite
├─test
│  ├─test_summer
│  └─test_winter
└─train
   ├─train_summer
   └─train_winter
```

## Train, Test Split

Summer: 1540 images
Winter: 1200 images

train_summer: 1231 images
train_winter: 962 images
test_summer: 309 images
test_winter: 238 images

## Image Size

Size: 256 X 256

# Dataset

This dataset was obtained from UC Berkeley's official directory of [CycleGAN Datasets]

## File Path 구성

```
summer2winter_yosemite
├─test
│  ├─test_summer
│  └─test_winter
└─train
   ├─train_summer
   └─train_winter
```

## Train, Test Split

Summer: 1540 images
Winter: 1200 images

train_summer: 1231 images
train_winter: 962 images
test_summer: 309 images
test_winter: 238 images

## Image Size

Size: 256 X 256

# CNN Model - Objective

Binary image classification using Yosemite_summer2winter Dataset

STEP 1

## Data Pipeline

Custom Image Dataset 설계

Constructor
__getitem__
__len__

STEP 2

## Data Augmentation

Albumentation Module

Training Dataset에 적용

STEP 3

## Dataset, DataLoader

Dataset Creation (Labeling)

Dataset Concat (train, test)

Validation Dataset

DataLoader

STEP 4

## Transfer Learning

Pretrained Model

Layer Parameter Freeze

FC Layer 조정

STEP 5

## Train, Validation, Test

Epoch 50으로 설정

Tensorboard에 기록

# CNN Model – Data Pipeline

```python
class CustomImageDataset(Dataset):
    def __init__(self, img_dir, transform=None, label=0):
        self.img_dir = img_dir
        self.transform = transform
        self.label = label
        self.img_paths = [
            os.path.join(img_dir, img_file)
            for img_file in os.listdir(img_dir)
            if img_file.lower().endswith(("png", "jpg", "jpeg"))
        ]

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

    def __getitem__(self, idx):
        img_path = self.img_paths[idx]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image=np.array(image))['image']
        return image, self.label
```

**PyTorch Dataset** – **3 components**: (data, label) for each index, length

· img_dir: current folder path

· transform: data augmentation method

· label: data labeling (0 – summer, 1 – winter)

· img_paths: [ file path in specific folder ]

# CNN Model – Data Augmentation

```python
# 데이터 증강 및 전처리
transform_train = A.Compose([
    A.Resize(256, 256),
    A.HorizontalFlip(p=0.85),
    A.RandomRotate90(p=0.85),
    A.VerticalFlip(p=0.85),
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    A.RandomCrop(width=224, height=224),  # 이미지 크기에 맞게 조정 필요
    A.OneOf([
        A.MotionBlur(p=0.85),
        A.OpticalDistortion(p=0.85),
        A.GaussNoise(p=0.85)
    ], p=0.85),
    A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    A.pytorch.transforms.ToTensorV2(),
])


transform_test = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    A.pytorch.transforms.ToTensorV2(),
])
```

**Albumentation Library: Data Augmentation**

*Torchvision 에서 제공하는 transform 보다 더 빠르고 다양함*

- **Filp** – Vertical(상하), Horizontal(좌우)
- **ColorJitter** – Brightness(밝기), contrast(대비), saturation(채도)
- **RandomCrop** (자르기)
- **Noise**(소음) – GaussianNoise, MotionBlur, OpticalDistortion
- **Normalize**(정규화) – mean & standard deviation for each channel

# CNN Model – DataSet, DataLoader

```python
# 데이터셋 생성
train_dataset_summer = CustomImageDataset(
    os.path.join(root_dir_train, "train_summer"), transform=transform_train, label=0
)
train_dataset_winter = CustomImageDataset(
    os.path.join(root_dir_train, "train_winter"), transform=transform_train, label=1
)

test_dataset_summer = CustomImageDataset(
    os.path.join(root_dir_test, "test_summer"), transform=transform_test, label=0
)
test_dataset_winter = CustomImageDataset(
    os.path.join(root_dir_test, "test_winter"), transform=transform_test, label=1
)

# 데이터셋 합치기
train_dataset = ConcatDataset([train_dataset_summer, train_dataset_winter])
test_dataset = ConcatDataset([test_dataset_summer, test_dataset_winter])
```

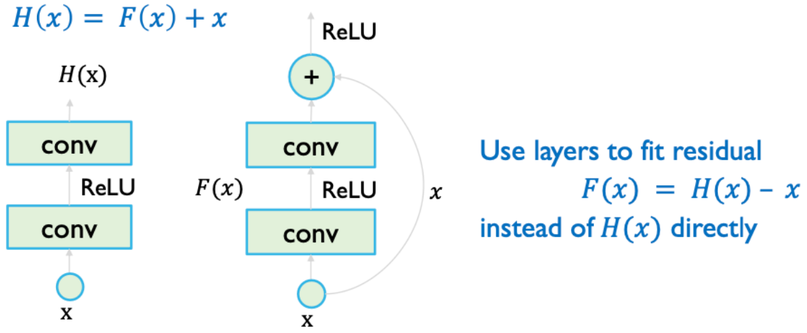
```python
# 검증 데이터셋 분할
val_split = 0.2
train_size = int((1 - val_split) * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset_split, val_dataset_split = random_split(
    train_dataset, [train_size, val_size]
)

# DataLoader 생성
train_loader = DataLoader(train_dataset_split, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset_split, batch_size=16, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

- Dataset Creation & Labeling (Summer – 0, Winter – 1)
- Dataset Concatenation – Train & Test
- Validation Set & DataLoader

# CNN Model – ResNet

**Residual Network (ResNet)**

- Main Idea: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping




[He et al., 2015]

93

· **Degradation Problem**: Network가 깊어질수록 train & test error가 커지는 문제

· **Residual Learning**: H(x) = F(x) + x로 최종 Output을 구성하고 F(x) = H(x) – x가 된다

이를 Residual 이라 하며 결국 x는 학습대상이 아니므로 Residual을 학습

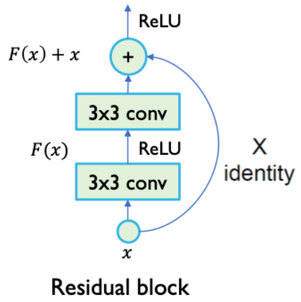
· **Skip Connection:** Residual이 0 되도록 학습, 미분을 할 때 F(x) = 0 으로 소실 되더라도,

H(x) = 0 + x 에서의 1로 gradient descent 문제를 해결 & '+' Low Computational Problem

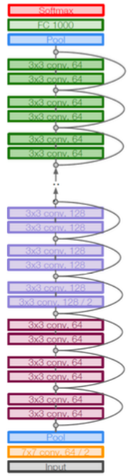
# CNN Model – ResNet

**Residual Network (ResNet)**

- Very deep networks using residual connections
  - 152-layer model for ImageNet
  - ILSVRC'15 classification winner (3.57% top 5 error)
  - Swept all classification and detection competitions in ILSVRC'15 and COCO'15!




Residual block



[He et al., 2015]

94

· **ILSVRC 2015 Winner:** Residual Block이라는 Module을 깊게 쌓아 152개의 Layer를 가짐

· **vs VGGNet:** VGGNet 보다 8배나 많은 Layer 개수를 가지지만 lower complexity

· **Model Architecture:** Residual Block으로 구성됨

Identity Mapping에 Zero-padding or 1 X 1 Convolution을 사용해 Channel의 개수 증가

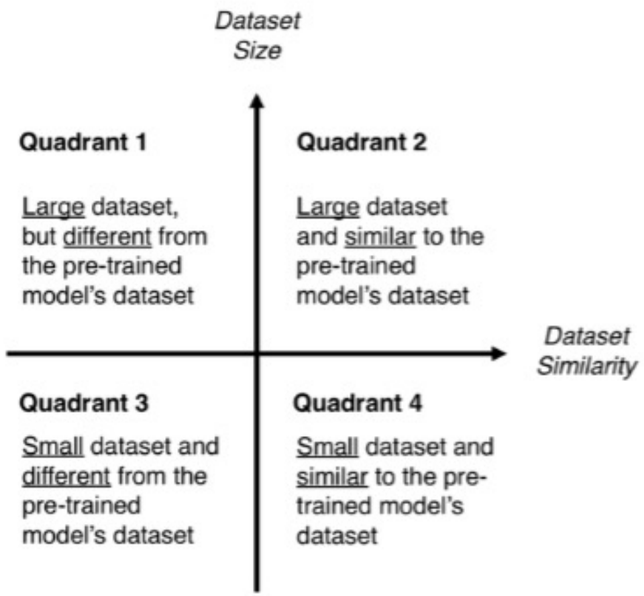
Feature Map size를 줄이기 위하여 Stride = 2를 사용

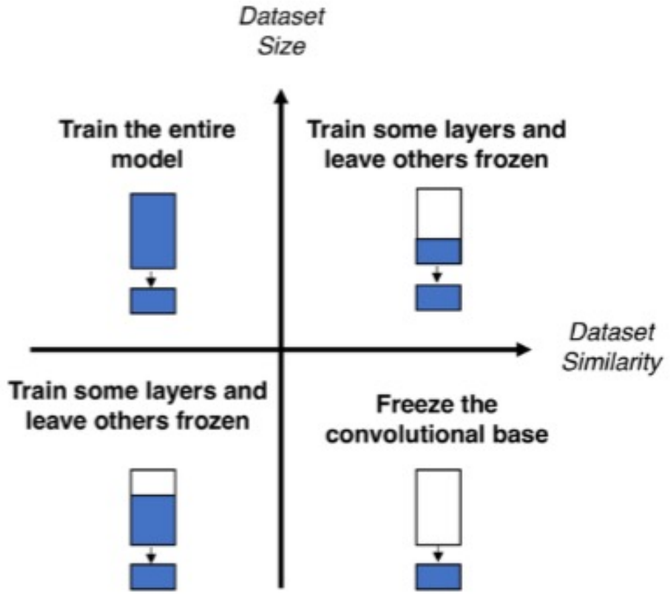
# CNN Model – Transfer Learning

**· CNN – 2 Image Classification Method**

| | 구분 | 설명 |
|---|---|---|
| 1. | 사용자가 직접 CNN 생성 | - 직접 ConvNet2D를 여러개 쌓아서 신경망 구축<br>- 단점 : 정교한 모형 구축하는데에 많은 시간 / 데이터 / 컴퓨팅 파워가 필요하다 |
| 2. | 사전학습 모델 사용 | • 사전 학습 모델이란 이미 방대한 양의 학습 데이터를 이용해서 학습을 완료한 모델이다. 해당 모델의 구조와 파라미터 최적값이 학습을 통해서 정해져 있다.<br><br>• 성능이 좋은 정교한 모형을 사용할 수 있고, 오랜 시간과 컴퓨팅 파워를 사용해서 대용량 데이터를 학습한 모델을 효율적으로 사용할 수 있다. |
| | 2-1. 학습모델 그대로 사용 | • 원래 모델을 그대로 사용하는 방식으로, 새로운 task가 사전 학습 모형의 task와 유사한 경우에 적용할 수 있다.<br><br>•하지만, 사전학습 모형은 대부분 ImageNet 대회에서 우승한 모델이기 때문에 대부분의 종속변수 클래스가 동물이라는 점을 유의해야 한다. |
| | 2-2. 전이학습<br>(Transfer Learning) | • 사전학습 모델의 파라미터나 구조를 일부 변형해서 사용하는 방법<br>새로운 학습 데이터가 필요하지만, 완전히 모델을 새로 만드는 것 만큼 많은 데이터가 필요하지는 않다. |

# CNN Model – Transfer Learning




**· Transfer Learning**

ImageNet으로 Pre-trained된 Backbone을 이용하여 feature map extraction

4-Category에 따라 fc-layer만 설계히여 fine-tuning

**· Current Task의 data size**
**· pretrained model과의 dataset or task similarity**

**· Fine Tuning Point**

- learning_rate를 매우 작게 설정 (1/10 수준)

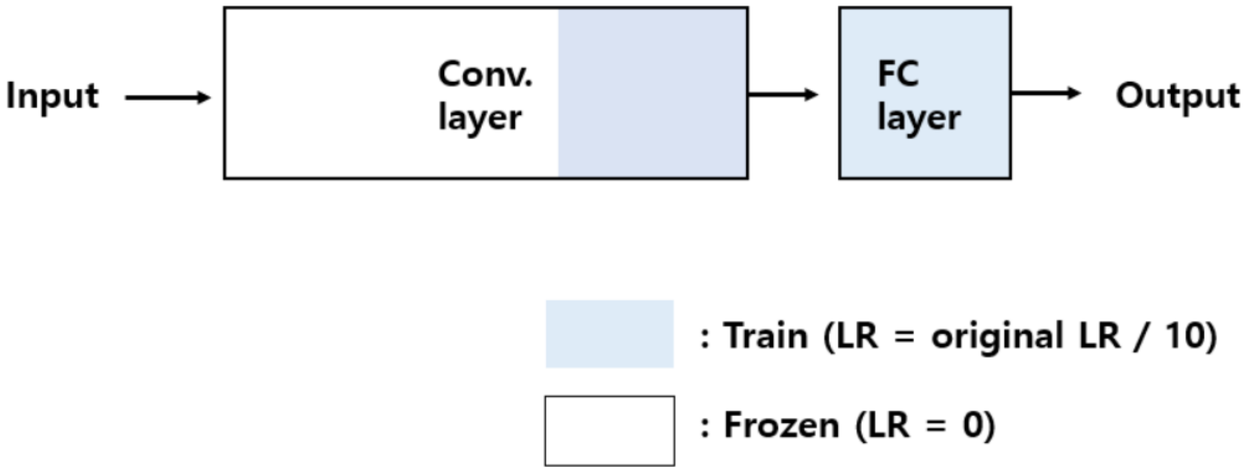
- Stable한 SGD Optimizer를 이용

- Bottleneck Feature만 따로 저장하여 FC Layer만 학습
(Data Augementation의 필요 없이 Dataset size가 큰 경우)

# CNN Model – Transfer Learning

· **Transfer Learning**

- Task Data Size ↓
- Dataset Similarity ↓




· **Last Convolution Layer: unfreeze**

· **New FC Layer w/ dropout**

· **Optimizer – learning_rate를 layer 별로 조정**

# CNN Model – Transfer Learning 1

```python
# ResNet50 모델 로드 및 수정
model = models.resnet50(pretrained=True).to(device)

# 모든 파라미터를 먼저 고정하고 특정 레이어의 파라미터를 학습 가능하게 설정
for param in model.parameters():
    param.requires_grad = False

for param in model.layer4.parameters():
    param.requires_grad = True

num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_ftrs, 2)
).to(device)
```

- **Pretrained Model – resnet(18, 34, 50, 101)**

- **Last Convolution Layer: unfreeze**

- **New FC Layer w/ dropout**

```python
#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)
```

- **Optimizer – learning_rate를 layer 별로 조정**

- **Scheduler– learning_rate를 active 하게 조정**

```python
scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader), eta_min=0, last_epoch=-1)
```

# CNN Model – Transfer Learning 2

```python
# 모든 파라미터를 먼저 고정하고 특정 레이어의 파라미터를 학습 가능하게 설정
for param in model.parameters():
    param.requires_grad = False

for name, param in model.named_parameters():
    if "layer3" in name or "layer4" in name:
        param.requires_grad = True

num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_ftrs, 2)
).to(device)

#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam([
    {'params': model.layer3.parameters(), 'lr': 5e-5},
    {'params': model.layer4.parameters(), 'lr': 5e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-4)

num_epochs = 50
scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs // 2, eta_min=1e-6)
```

- **Pretrained Model – resnet(18, 34, 50, 101)**
- **3(the layer before last), 4$^{th}$(last) Convolution Layer: unfreeze**
- **New FC Layer w/ dropout**
- **Optimizer – learning_rate를 layer 별로 더 세밀하게 조정 & weight decay 추가**
- **Scheduler– learning_rate를 active 하게 조정**

# CNN Model – Transfer Learning 3

```python
# ResNet50 모델 로드 및 수정
model = models.resnet50(pretrained=True).to(device)

# 모든 파라미터를 먼저 고정하고 특정 레이어의 파라미터를 학습 가능하게 설정
for param in model.parameters():
    param.requires_grad = False

num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_ftrs, 2)
).to(device)

#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)

scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader), eta_min=0, last_epoch=-1)

# 손실 함수 설정
criterion = nn.CrossEntropyLoss()
```

- **Pretrained Model – resnet(18, 34, 50, 101)**
- **All Layer: Freeze**
- **New FC Layer w/ dropout**
- **Optimizer – learning_rate를 layer 별로 조정**

# CNN Model – Train, Validation, Test

```python
# 학습 함수
def train(model, train_loader, device, criterion, optimizer, epoch, writer):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(output.data, 1)
        total += target.size(0)
        correct += (predicted == target).sum().item()

        train_accuracy = 100.0 * correct / total

        global_step = (epoch - 1) * len(train_loader) + batch_idx
        writer.add_scalar("Loss/train", loss.item(), global_step)
        writer.add_scalar("Accuracy/train", train_accuracy, global_step)

    train_loss = running_loss / len(train_loader)
    print(f"Train Epoch: {epoch}, Loss: {train_loss:.6f}, Acc: {train_accuracy:.2f}%")
```

```python
# 검증 함수
def validate(model, val_loader, device, criterion, epoch, writer):
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (data, target) in enumerate(val_loader):
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output, target)
            val_loss += loss.item()
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

            val_accuracy = 100.0 * correct / total

            global_step = (epoch - 1) * len(train_loader) + batch_idx
            writer.add_scalar("Loss/val", loss.item(), global_step)
            writer.add_scalar("Accuracy/val", val_accuracy, global_step)

    val_loss /= len(val_loader)

    print(f"Validation Epoch: {epoch}, Loss: {val_loss:.6f}, Acc: {val_accuracy:.2f}%")
```

# CNN Model – Train, Validation, Test

```python
# 테스트 함수
def test(model, device, test_loader, epoch, writer):
    model.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item()
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

    test_loss /= len(test_loader)
    test_accuracy = 100. * correct / total
    print(f'\nTest set: Accuracy: {test_accuracy:.2f}%\n')
```

```python
# 주어진 설정으로 모델 훈련
num_epochs = 50
for epoch in range(1, num_epochs + 1):
    train(model, train_loader, device, criterion, optimizer, epoch, writer)
    validate(model, val_loader, device, criterion, epoch, writer)
    scheduler.step()


test(model, device, test_loader, epoch, writer)
```

▶ TensorBoard 세션 시작

```
%load_ext tensorboard
```

▶ TensorBoard 세션 시작

```
%tensorboard --logdir=runs
```

```python
# 학습이 끝나면 SummaryWriter를 닫습니다.
writer.close()
```

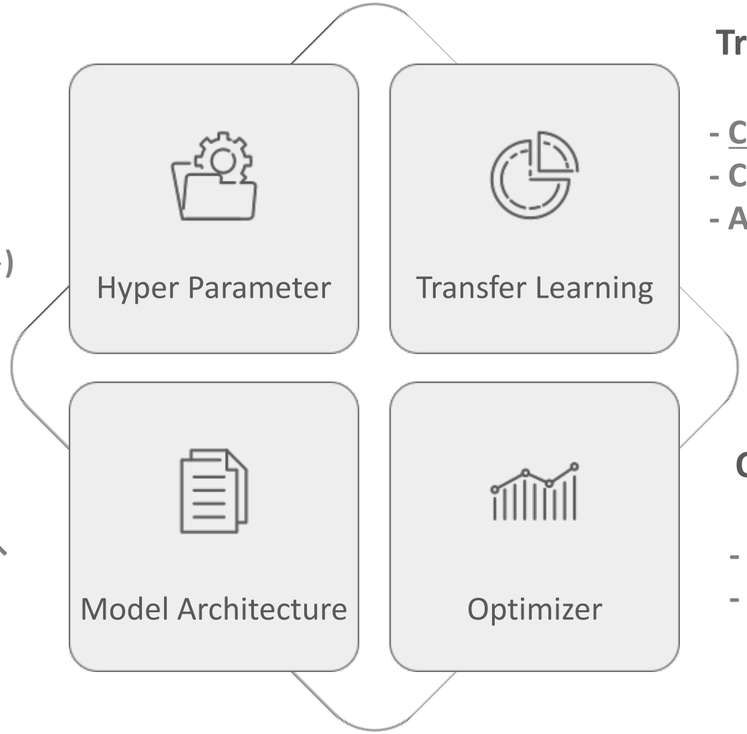
# CNN Model – Result Analysis

## Hyper Parameter

- batch_size: 16, 32, 64 성능 차이 x – 32로 선택
- learning_rate: Transfer Learning (Layer별로 다름)
- Conv4: , FC Layer: , weight_decay
- epoch: 50 넘어가면 overfitting (validation score ↓)

## Transfer Learning

- Convolutional Layer 4만 unfreeze: Test Acc - 85.07%
- Convolutional Layer 3, 4만 unfreeze: Test Acc - 82.27%
- All Layer freeze: Test Acc – 79.52%




## Model Architecture

- ResNet: 18. 34, 50, 101 성능 거의 차이 X - resnet34↑
- EfficientNet
- DenseNet

## Optimizer

- Adam
- SGD

# CNN Model – Result Analysis

**· learning_rate Comparison**

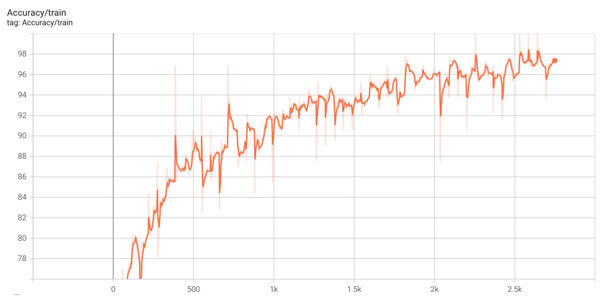
**[Fixed Conditions]**

- Model Architecture: ResNet34
- Hyperparameter : batch_size=32, epoch=50, CosineAneealingLR
- Transfer Learning: Conv4 Layer unfreeze + FC Layer
- Optimizer: Adam

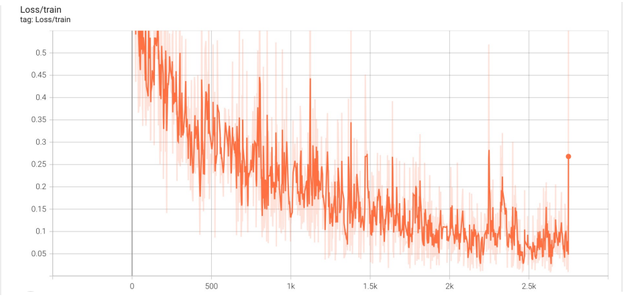
**[learning_rate]**

```python
#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)

scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader), eta_min=0, last_epoch=-1)
```

**Test Accuracy 85.07%**



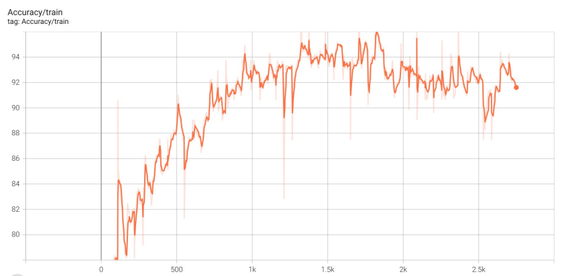

**· Train Accuracy_1**



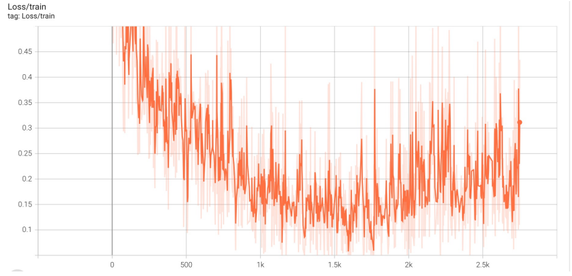

**· Train Loss_1**

```python
#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam(
    [
        {"params": model.layer3.parameters(), "lr": 5e-5},
        {"params": model.layer4.parameters(), "lr": 5e-4},
        {"params": model.fc.parameters(), "lr": 1e-3},
    ],
    lr=1e-4,
    weight_decay=1e-5,
)

num_epochs = 50
scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs // 2, eta_min=1e-6)
```

**Test Accuracy 81.90%**




**· Train Accuracy_2**




**· Train Loss_2**

# CNN Model – Result Analysis

**· Scheduler Comparison**

**[Fixed Conditions]**

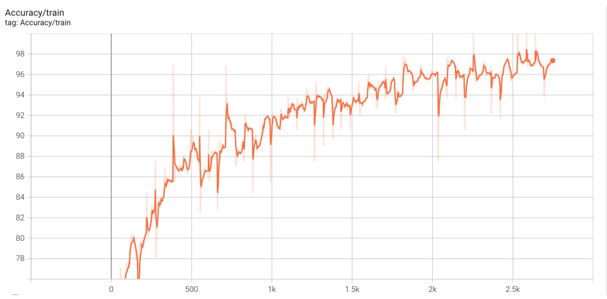
- Model Architecture: ResNet34
- Hyperparameter : batch_size=32, epoch=50, learning_rate
- Transfer Learning: Conv4 Layer unfreeze + FC Layer
- Optimizer: Adam

**[Scheduler] CosineAnnealingLR: Test Accuracy 85.07%**

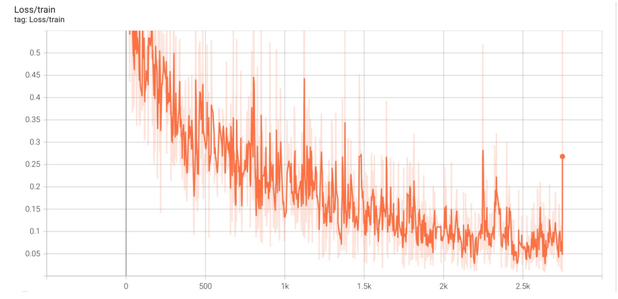
```
#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)

scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader), eta_min=0, last_epoch=-1)
```

```
#  옵티마이저와 스케줄러 설정
optimizer = optim.SGD([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)

scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=10, verbose=True)
```

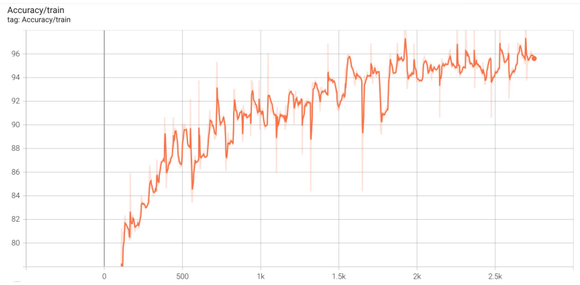
**ReduceLRonPlateau: Test Accuracy 84.19%**



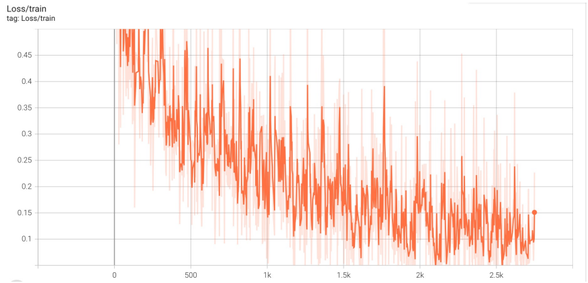

**· Train Accuracy_1**




**· Train Loss_1**




**· Train Accuracy_2**



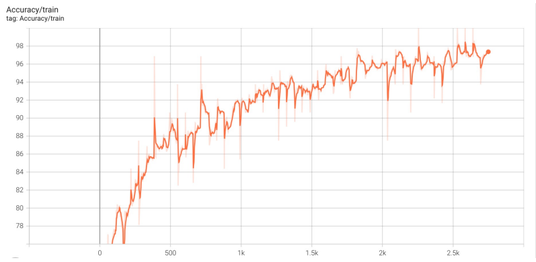

**· Train Loss_2**

# CNN Model – Result Analysis

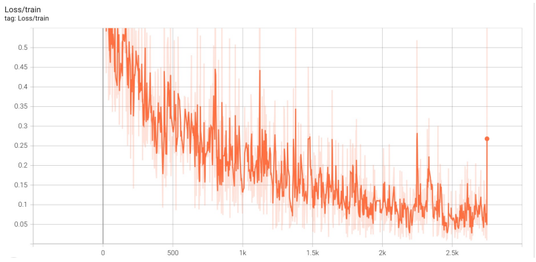
· **Transfer Learning Comparison**

**[Fixed Conditions]**
- Hyperparameter : batch_size=32, learning_rate_1, epoch=50
- Model Architecture: ResNet34
- Optimizer: Adam

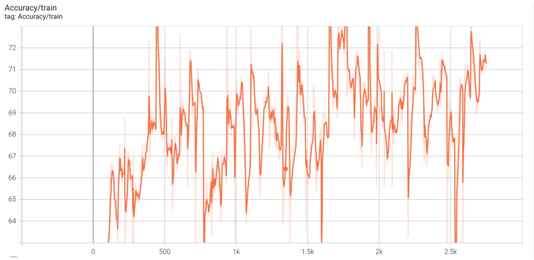
**[Transfer Learning]**
- Convolutional Layer 4만 unfreeze: Test Acc - 85.07%
- Convolutional Layer 3, 4만 unfreeze: Test Acc - 82.27%
- All Layer freeze: Test Acc – 79.52%



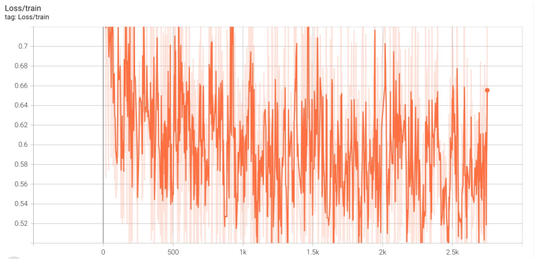

· **Train Accuracy_1**



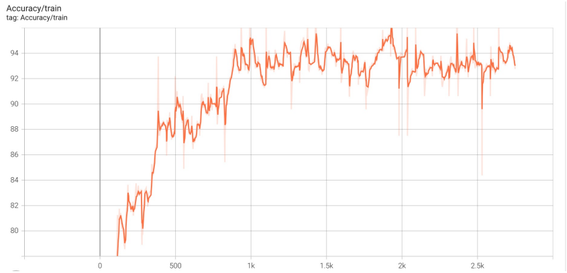

· **Train Loss_1**



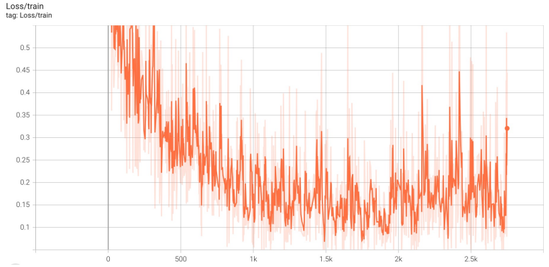

· **Train Accuracy_3**




· **Train Loss_3**




· **Train Accuracy_2**




· **Train Loss_2**

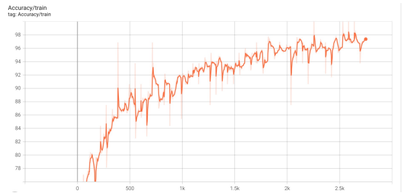
# CNN Model – Result Analysis

**· Model Architecture Comparison**

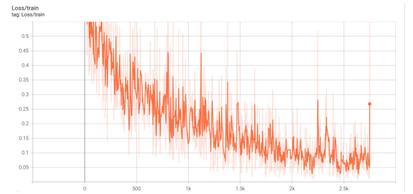
**[Fixed Conditions]**
- Hyperparameter : batch_size=32, learning_rate_1, scheduler, epoch-50
- Transfer Learning: Conv4 Layer unfreeze + FC Layer
- Optimizer: Adam

**[Model Architecture]**
- ResNet18, 34, 50, 101
- DenseNet

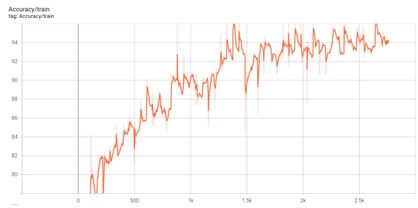
- ResNet Test Accuarcy: 84.46% , 85.07%, 84.46%, 84.64%
- DenseNet Test Accuarcy: 84.46%



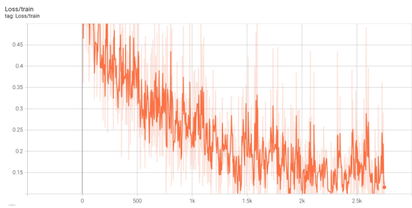
· Train Accuracy_resnet34



· Train Loss_resnet34



· Train Accuracy_densenet121



· Train Loss_densenet121

# CNN Model – Result Analysis

**· Optimizer Comparison**

**[Fixed Conditions]**

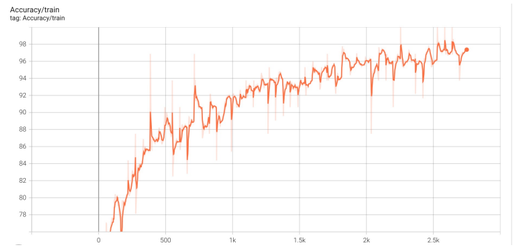
- Hyperparameter : batch_size=32, learning_rate_1, scheduler, epoch=50
- Model Architecture: ResNet34
- Convolutional Layer 4만 unfreeze

**[Optimizer]**

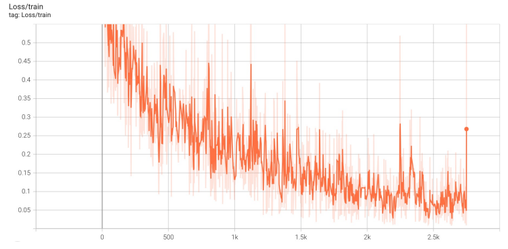
- Adam **Test Accuracy 85.07%**
- SGD (epoch 수를 더 늘렸으면 다를 가능성 존재) **Test Accuracy 79.16%**

```
#  옵티마이저와 스케줄러 설정
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)

scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader), eta_min=0, last_epoch=-1)
```

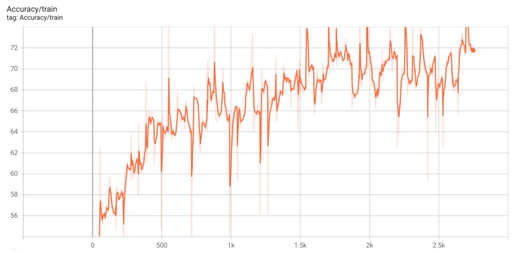
```
#  옵티마이저와 스케줄러 설정
optimizer = optim.SGD([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
], lr=1e-3)

scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader), eta_min=0, last_epoch=-1)
```



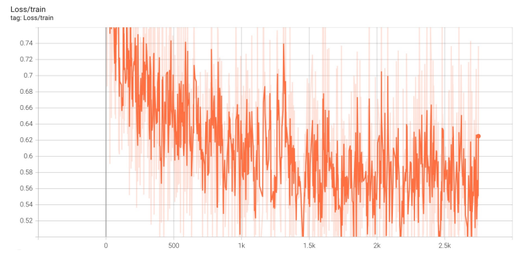

**· Train Accuracy_Adam**




**· Train Loss_Adam**




**· Train Accuracy_SGD**




**· Train Loss_SGD**

# Thank You!

경청해주셔서 감사합니다.

Manager. 김미리
E-Mail. miri@mcompany.com
Tel. 070.1234.5678