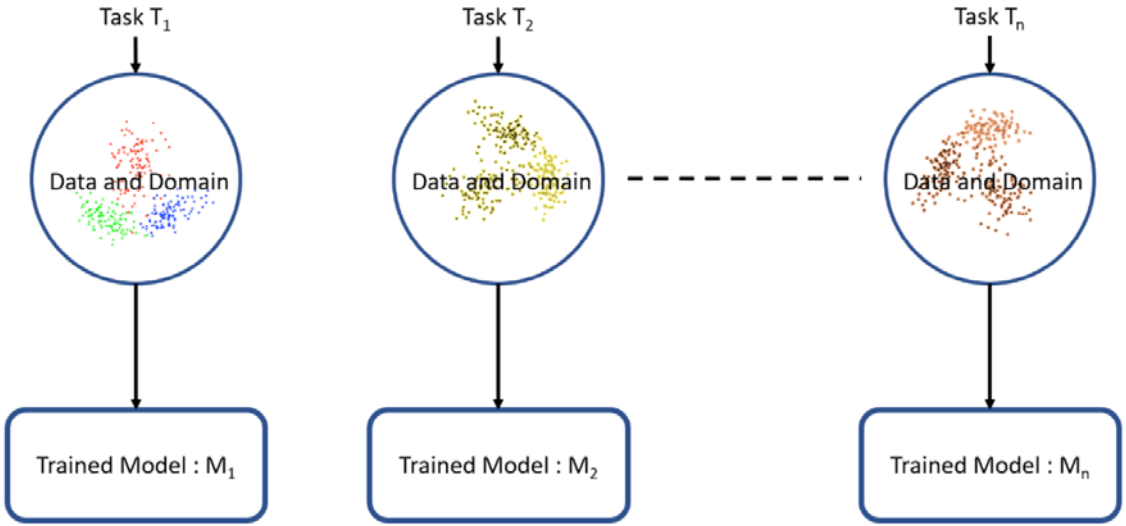
Transfer Learning

Prof. Hyunseok Oh

School of Mechanical Engineering
Gwangju Institute of Science and Technology

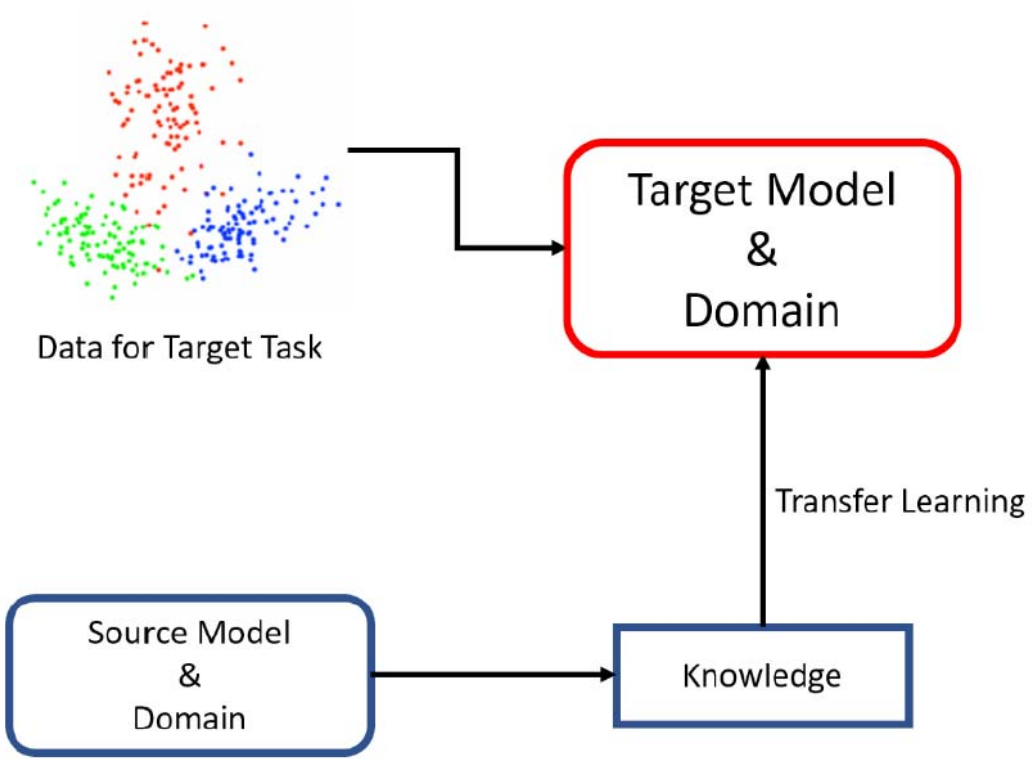
Conventional Machine Learning Model

- For a particular task, conventional machine learning models work well only under an assumption that the training and test data are drawn from the same distribution.
- However, if a new dataset has a different distribution, the trained model cannot be used any more. The machine learning model needs to be trained from scratch.



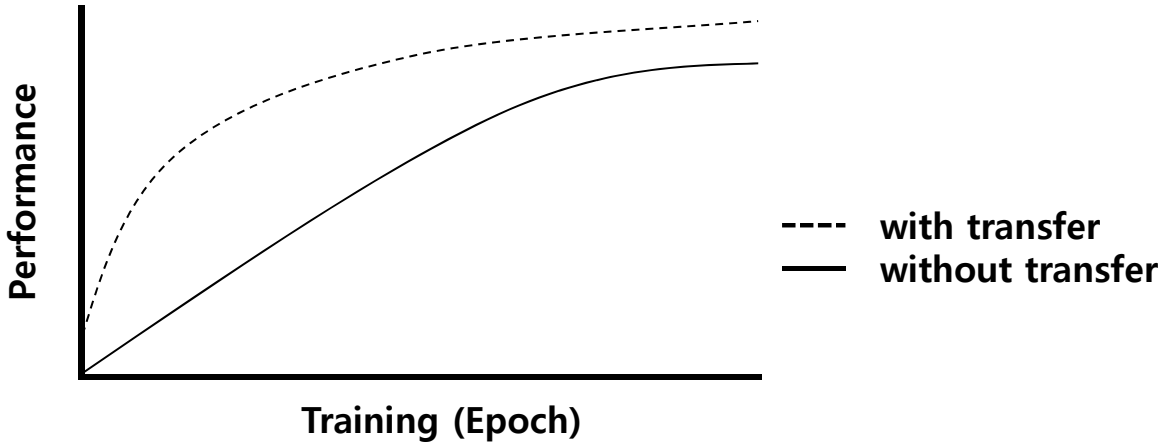
Transfer Learning

- 전이 학습: 소스 과제에서 습득한 지식을 타깃 과제 학습 시 활용



Transfer Learning

- 장점
  - 향상된 기본 성능 : 더 높은 정확도의 성능에서 학습 시작
  - 모델 개발 시간 단축 : 더 급해진 학습 곡선 기울기
  - 향상된 최종 성능 : 더 높아진 최종 정확도



Terms and Definitions

- Source와 Target
  - Source: 기존 대상
  - Target: 전이가 될 대상
- Domain과 Task
  - Domain $D = \{X, P(X)\}$

Feature space **Marginal probability distribution**

예, X는 가능한 이미지, P(*X*)는 특정 이미지의 확률 분포

  - Task $T = \{Y, P(y \mid X)\}$

Label space **Objective predictive function**

예, *Y*는 “신선”, “변질”등의 레이블, P(y|X)는 이미지가 주어졌을 때 “변질”일 확률
예, *Y*는 “스가루”, 홍로”, “후지”등의 레이블, P(y|X)는 이미지가 주어졌을 때 “후지”일 확률

Objective

- 전이 학습 목표
  - 소스 도메인 (D_S)과 소스 태스크 (T_S)의 지식을 활용하여 타겟 도메인 (D_T)의 태스크 성능($f_T(\cdot)$)을 향상
- $D_S \neq D_T$
  - $x_S \neq x_T$: 다른 장비(가시광선, 적외선, 초분광)로 촬영한 이미지
  - $P_S(X) \neq P_T(X)$: 다른 종류의 과일
- $T_S \neq T_T$
  - $y_S \neq y_T$: 소스와 타겟 클래스의 갯수가 서로 다름
  - $P(Y_S \mid X_S) \neq P(Y_T \mid X_T)$: 소스와 타겟 이미지의 클래스가 서로 매우 불균형

Strategy

- 가능한 시나리오
  - Inductive transfer learning
    - 타겟과 소스 태스크가 다름
      - 다중 작업 학습 (Multi-task learning) : 소스 도메인의 레이블이 지정되지 않은 데이터
      - 지도 학습 (Supervised learning) : 소스 데이터의 레이블이 지정된 데이터
  - Transductive transfer learning
    - 타겟과 소스 도메인이 다름
      - 일반적 변환 전이 (General transductive transfer learning) : 특성 공간 x가 다름
      - 도메인 적응 (Domain adaptation) : 확률 분포 P(x)가 다름
  - Unsupervised transfer learning
    - 클러스터링, 차원 축소, 오토 엔코더 등

Learning Settings		Source and Target Domains	Source and Target Tasks
Traditional Machine Learning		the same	the same
Transfer Learning	*Inductive Transfer Learning / Unsupervised Transfer Learning*	the same	different but related
		different but related	different but related
	Transductive Transfer Learning	different but related	the same

심층 전이 학습 유형 (1/2)

- 도메인 적응 (Domain adaptation)

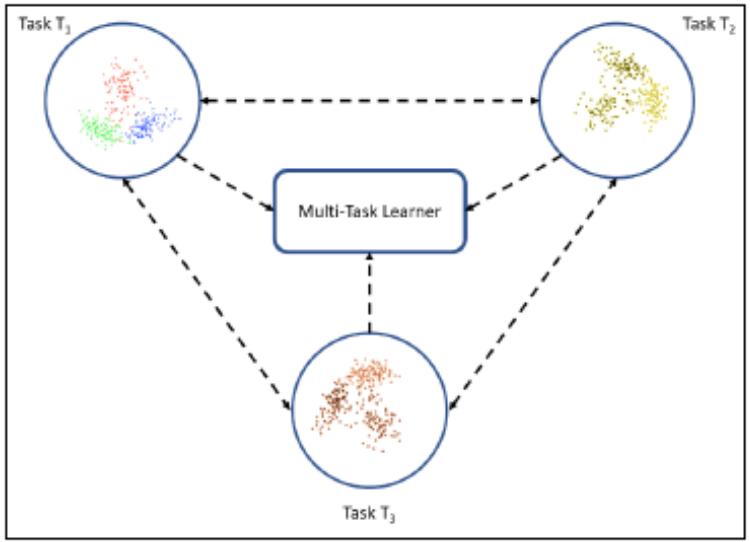
$$P(X_S) \neq P(X_T)$$

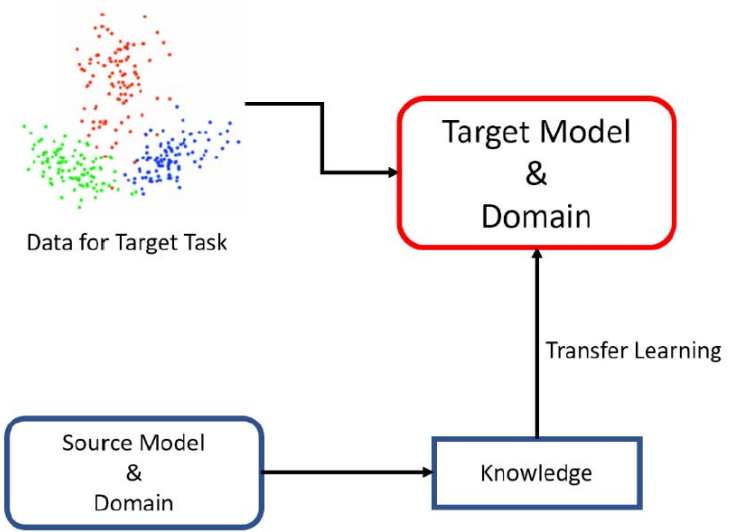
$$D = \{x, P(X)\}$$

특성 공간

주변부 확률

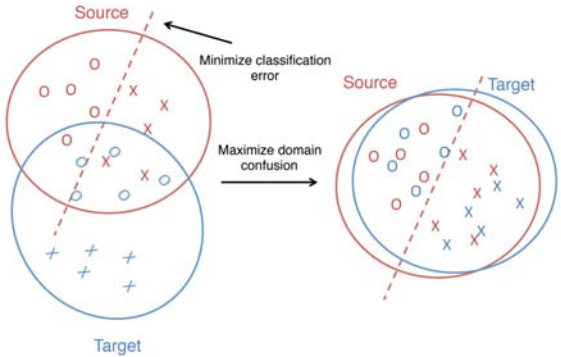
  - Ex) 영화 리뷰 데이터로 훈련된 모델을 제품 리뷰 데이터에 활용
- 다중 과제 학습 (Multitask learning)
  - 소스 태스크와 타겟 태스크를 구분하지 않고 여러 과제를 동시 학습





심층 전이 학습 유형 (2/2)

- 도메인 혼란 (Domain confusion)
  - 소스 도메인과 타겟 도메인의 편향 조절로 영역 혼동을 야기해서 유사성을 높이는 방법
- 원샷 학습 (Oneshot learning)
  - 적은 수의 데이터 학습과 베이즈 추론으로부터 카테고리화 하는 학습법
- 제로샷 학습 (Zeroshot learning)
  - 기존 입력 변수와 출력 변수 외에 임의 변수를 추가한 3가지 변수 사용
  - 미리 지도학습된 알고리즘의 예제로부터 레이블이 없는 예제를 학습하는 방법

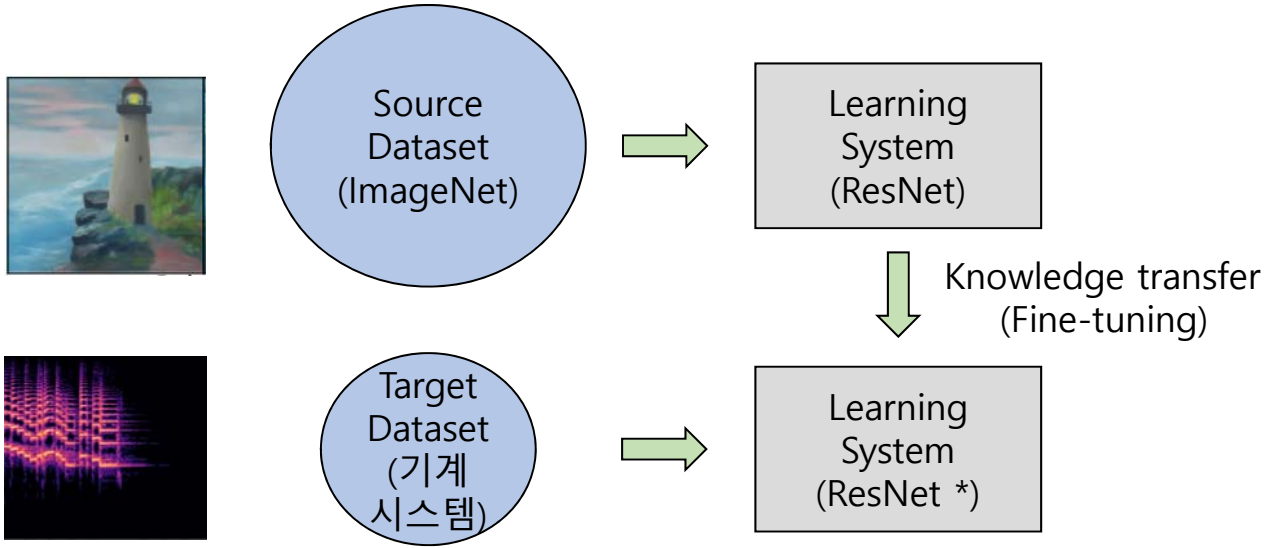


전이 학습 구현 방법

- 파라미터 전이 (Parameter transfer)
  - 타겟 태스크에 대한 모델의 일부 매개변수 및 하이퍼파라미터의 사전 분포를 공유한다는 가정
  - 타겟 도메인의 손실에 추가적인 가중치 적용
- 인스턴스 전이 (Instance transfer approach)
  - 소스 도메인 지식을 타겟 태스크에 재사용하는 것이 이상적
  - 대부분의 경우 소스 도메인 데이터 직접 재사용 불가
  - 타겟 데이터와 함께 소스 도메인의 특정 인스턴스 사용
- 특성 표현 전이 (Feature-representation transfer)
  - 도메인의 분산을 최소화하고 에러율을 줄이는 좋은 특성 표현 사용
- 관계형 지식 전이 (Relational-knowledge transfer)
  - 소스와 타겟 데이터의 서로 다른 데이터 포인트 간의 관계 정립 시도

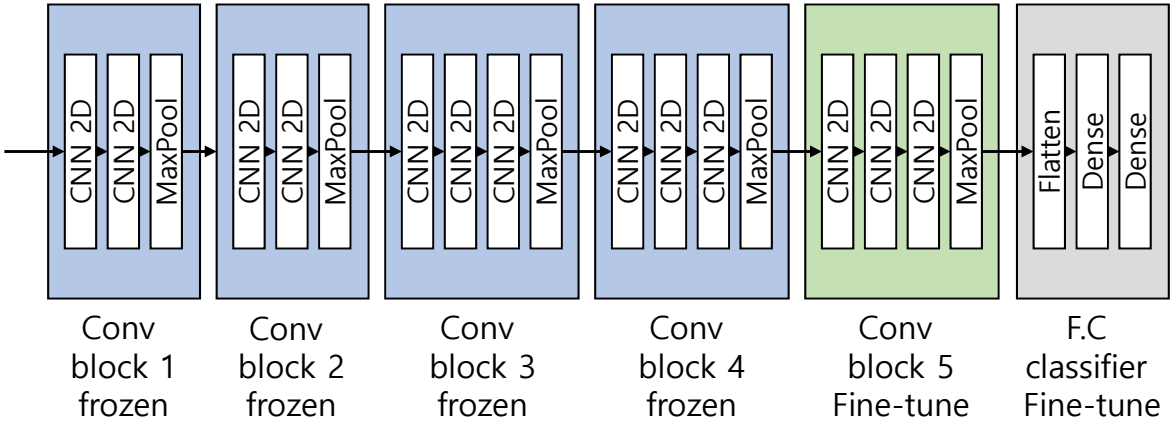
Fine Tuning

- 전이 학습 (Transfer Learning)
  - 하나의 문제를 해결하면서 얻은 지식을 다른 관련 문제에 적용하여 해결하는 연구 문제
  - 전이 학습의 목표는 상대적으로 규모가 큰 소스 작업 (Source Task)에서 얻은 지식을 사용하여 대상 작업 (Target Task)의 데이터 부족을 보완
- 미세 조정 (Fine-tuning)은 전이 학습의 예시



Fine Tuning

- 미세 조정 (Fine-tuning)
  - 사전 학습된 모델의 일부 하위 레이어를 고정 해제하고 모델의 새로 추가된 부분과 함께 훈련
  - 대상 문제와 관련성을 높이기 위해 재사용되는 모델의 추상 표현을 약간 조정하는 과정



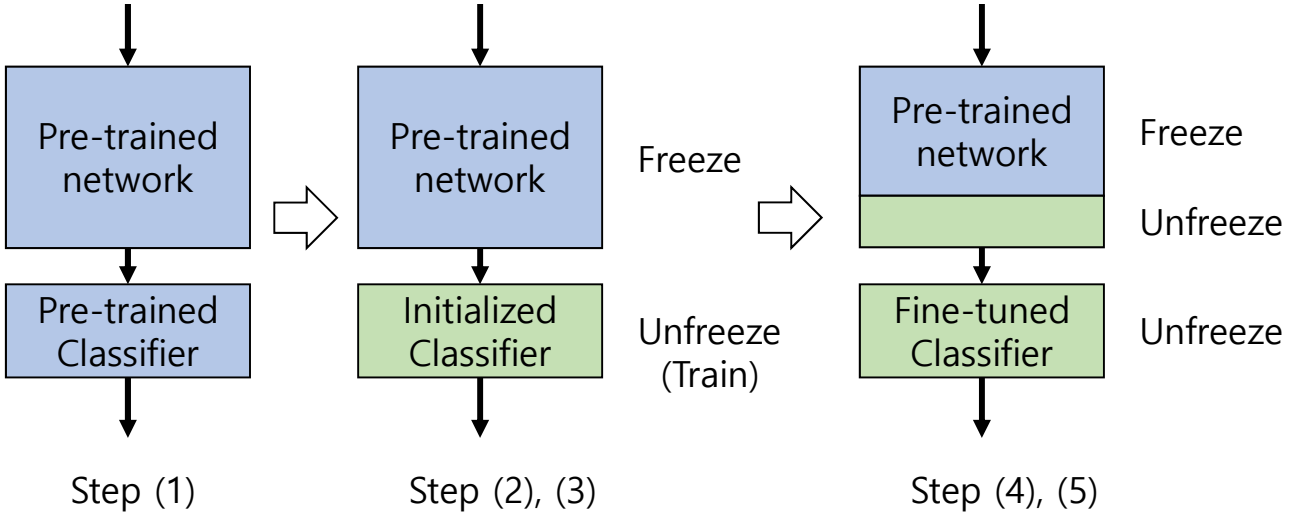
Fine-tuning with VGG16

Fine Tuning Procedure

- 네트워크를 미세 조정하는 과정
  - (1) 사전 학습된 모델에 새로운 분류기를 추가 및 변경
  - (2) 사전 학습된 네트워크를 고정 (Freeze)
  - (3) 추가된 새 분류기 학습
  - (4) 사전 학습된 네트워크의 일부 레이어를 고정 해제 (Unfreeze)
  - (5) 추가된 분류기와 고정 해제된 일부 레이어를 학습

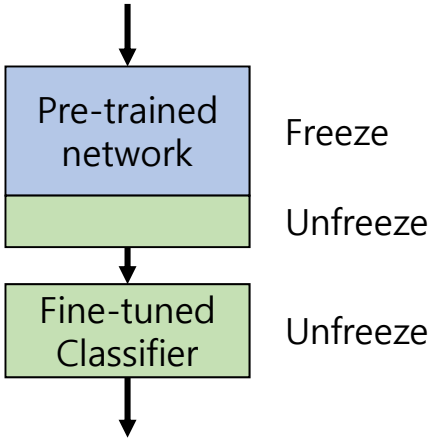
Feature extraction 단계

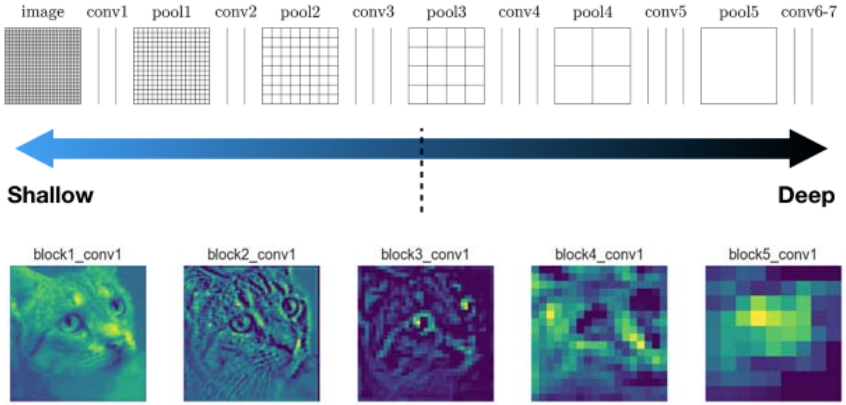
미세 조정 단계



Fine Tuning Procedure

- 왜 사전 학습된 모델의 모든 레이어를 미세 조정하지 않는가?
  - 오버 피팅 발생
    - 적은 데이터의 대상 데이터셋을 사용하여 전체 네트워크를 학습하면 오버 피팅 발생 가능
  - 깊은 신경망에서 전역적인 (Global) 특징 추출
    - 전역적인 특징을 미세하게 조정





Demo

Load VGG 16

- 분류기 없이 모델 불러오기

```
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Feature Extraction

- VGG16 모델 Feature extraction

→ Step 1: 사전 학습된 모델에 새로운 분류기를 추가 및 변경

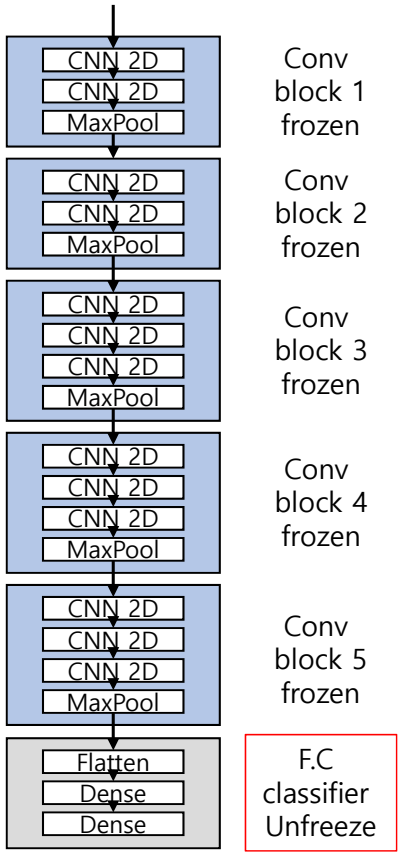
```
conv_base = VGG16(weights='imagenet', include_top=False,
                  input_shape=(150, 150, 3))

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
conv_base.trainable = False
```

→ Step 2: 사전 학습된 네트워크 고정

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5), metrics=['acc'])

history = model.fit(train_generator, steps_per_epoch=100, epochs=30,
                    validation_data=validation_generator,
                    validation_steps=50,
                    verbose=2)
```

→ Step 3: 추가된 새 분류기 학습



Fine Tuning

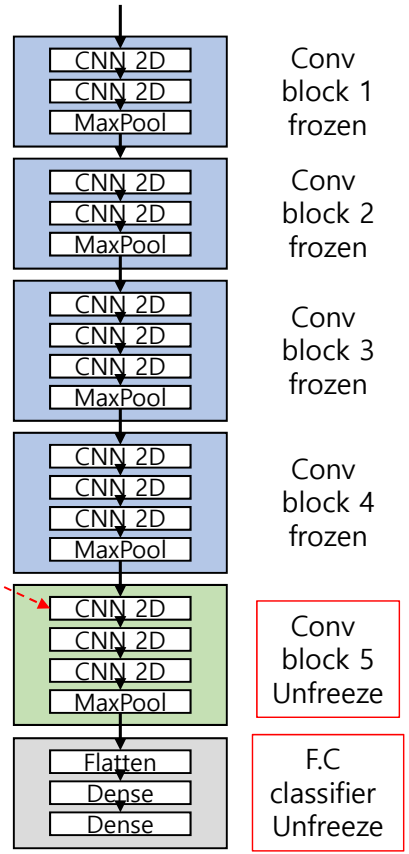
- VGG 16 모델 미세 조정

→ Step 4: 사전 학습된 네트워크의 일부 레이어를 고정 해제

```
model.trainable = True

set_trainable = False
for layer in model.layers[0].layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

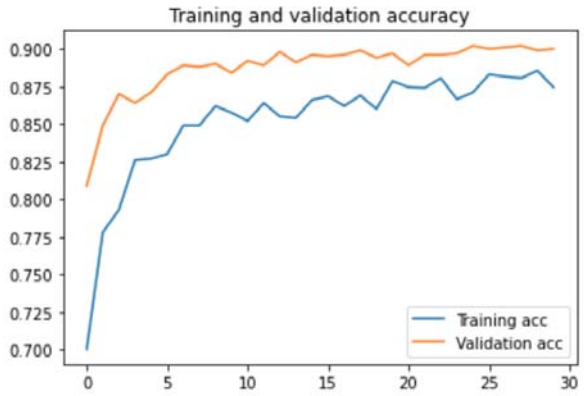
```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5), metrics=['acc'])

history = model.fit(train_generator, steps_per_epoch=100, epochs=30,
                    validation_data=validation_generator,
                    validation_steps=50,
                    verbose=2)
```

→ Step 5: 추가된 분류기와 고정 해제된 일부 레이어를 학습

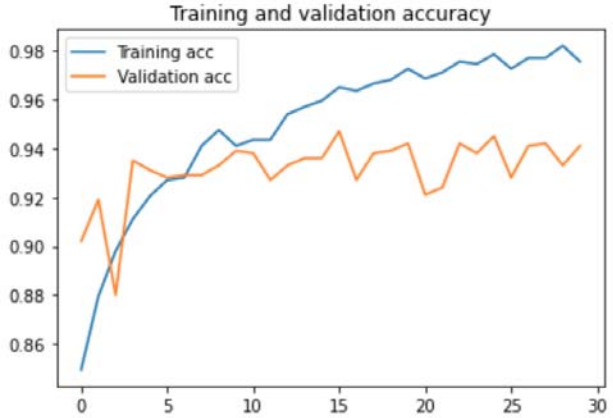


Results

- Feature extraction 정확도
  - 89.4 %
- 미세 조정 이후 정확도
  - 93.7 %
- 4.3 % 정확도 증가



After feature extraction



After fine-tuning

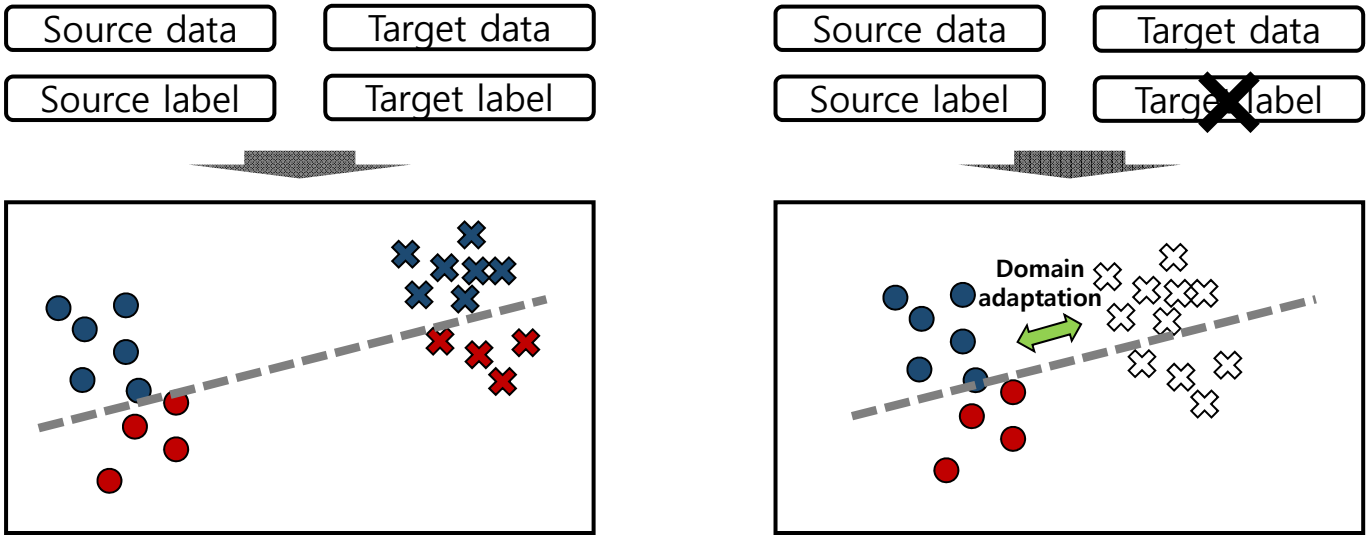
Domain Adaptation

Prof. Hyunseok Oh

School of Mechanical Engineering
Gwangju Institute of Science and Technology

Domain Adaptation

- 도메인 적응 학습은 타겟 데이터의 라벨이 필요하지 않음.
- 다른 조건에서 취득된 데이터에 대해 라벨 없이 학습이 가능함.

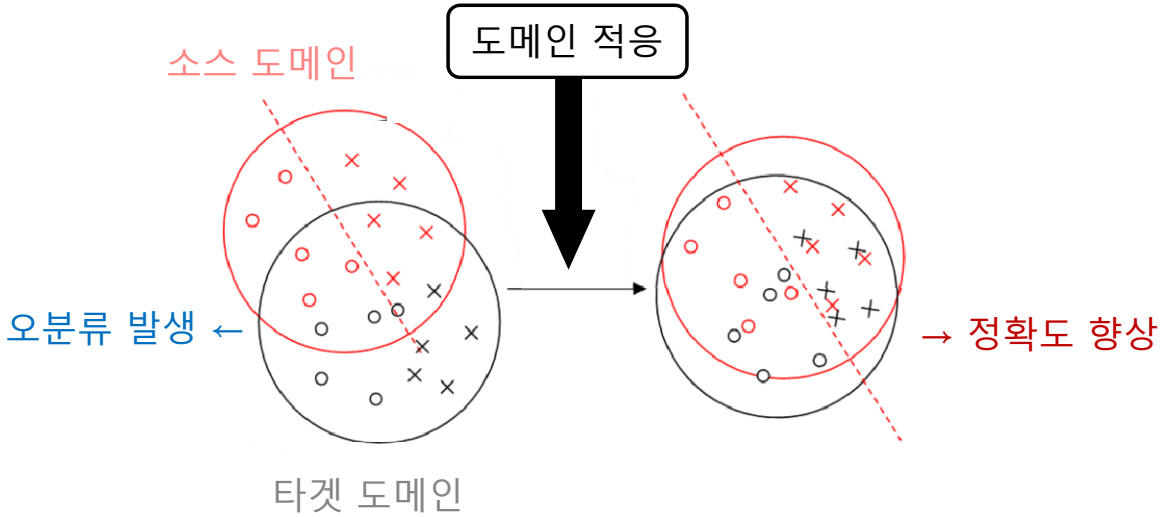


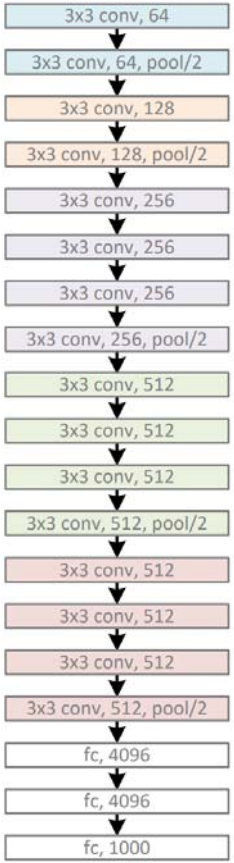
기존 CNN 이용한 지도 학습

도메인 적응 학습

Objective

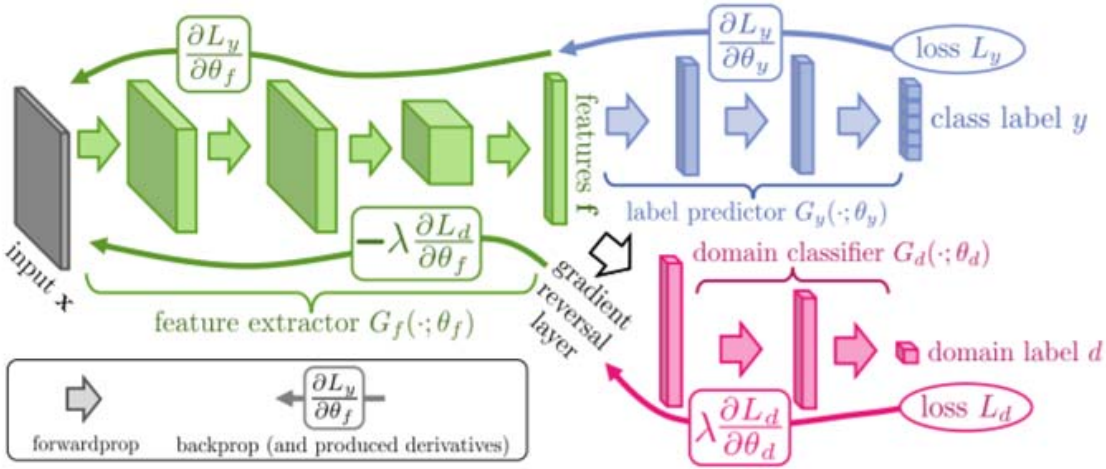
- 소스 도메인의 정보를 타겟 도메인에 적응시켜 예측 성능 향상
  - "**Feature space**" 상에서 도메인 간 데이터의 관계를 사용





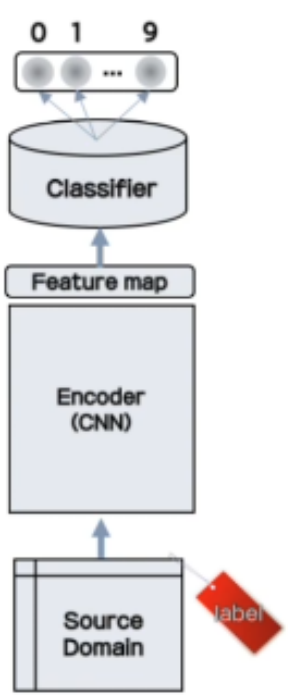
Domain Adversarial Neural Net (DANN)

- Ganin et al., “Unsupervised domain adaptation by backpropagation”, ICML, 2015.
- 타겟 도메인 label 없이 학습
- End to end 학습(태스크 분류기, 도메인 분류기)
- Backpropagation 변경 레이어 추가: GRL (Gradient reversal layer)

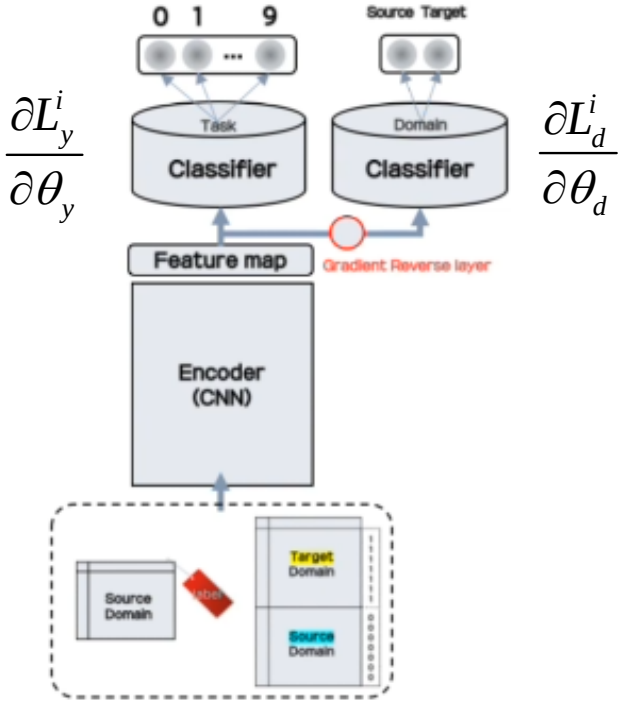


DANN Architecture

- 모델 아키텍쳐
  - 태스크 분류기 (Task classifier)
    - 데이터의 클래스 구분
  - 도메인 분류기 (Domain classifier)
    - 소스인지 타겟인지 도메인 구분
  - GRL (Gradient reversal layer)
    - 적대적 학습을 위한 가중치 음수화



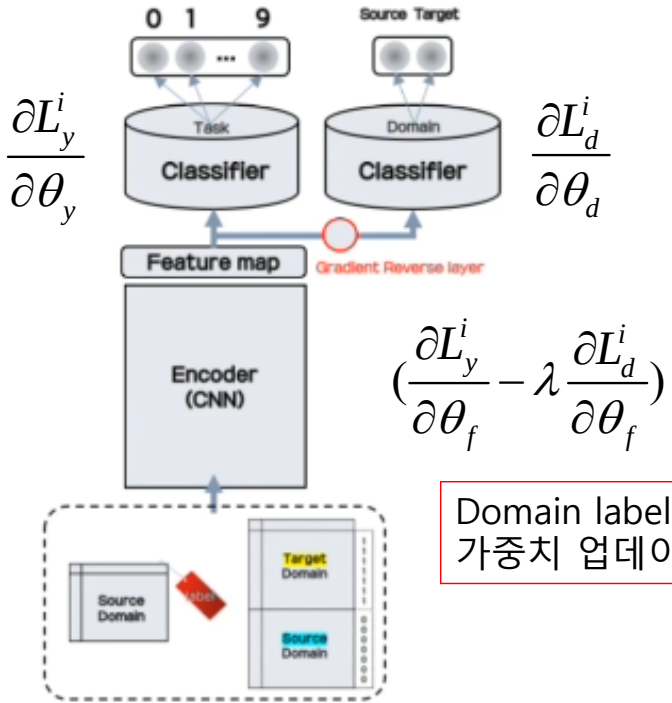
기존 CNN



DANN

DANN Architecture

- 역전파 과정
  - Domain label은 혼동하도록 학습
  - Task label은 잘 맞추도록 학습



$$\frac{\partial L_y^i}{\partial \theta_y}$$

$$\frac{\partial L_d^i}{\partial \theta_d}$$

$$\left(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f}\right)$$

Domain label 잘 분류 못하도록 가중치 업데이트

DANN

DANN Architecture

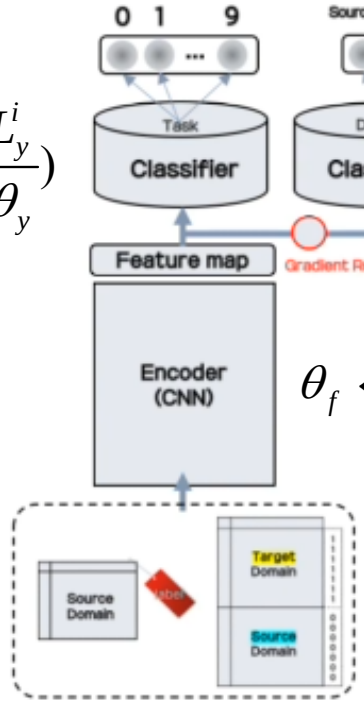
- 역전파 과정
  - Domain label은 혼동하도록 학습
  - Task label은 잘 맞추도록 학습

$$\theta_y \leftarrow \theta_y - \alpha\left(\frac{\partial L_y^i}{\partial \theta_y}\right)$$

$$\theta_d \leftarrow \theta_d - \alpha\left(-\lambda \frac{\partial L_d^i}{\partial \theta_d}\right)$$

$$\theta_f \leftarrow \theta_f - \alpha\left(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f}\right)$$

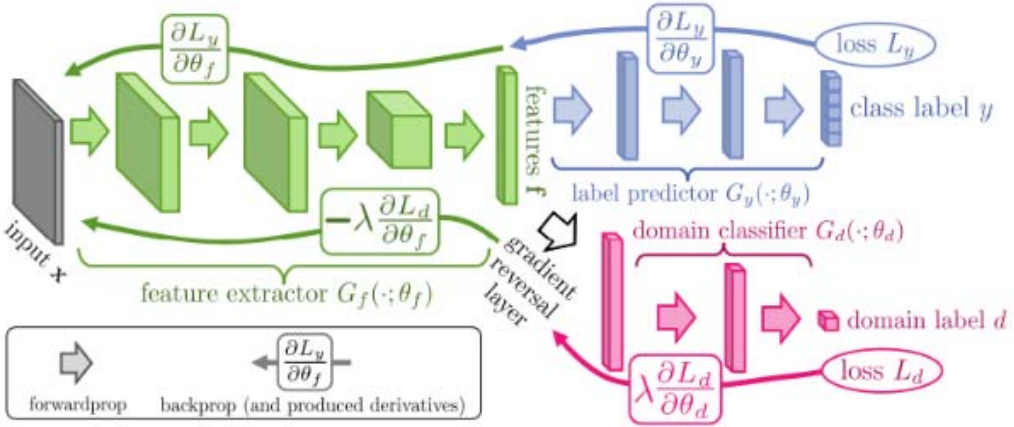
Domain label 잘 분류 못하도록
가중치 업데이트



DANN

DANN Loss Function

- Source domain에 대한 classification loss와 regularizer로 구성



$$\min_{W,b,V,c}\left[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(W,b,V,c)+\lambda\cdot R(W,b)\right]$$

Note: $\mathcal{L}_y^i(W,b,V,c)=\mathcal{L}_y(G_y(G_f(x_i;W,b);V,c),y_i)$; i-th example prediction loss.

DANN Loss Function

- Regularizer는 도메인을 혼동하는 방향으로 adversarial하게 학습

$$R(W,b) = \max_{u,z} \left[-\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_d^i(W,b,u,z) - \frac{1}{n'}\sum_{i=n+1}^{N} \mathcal{L}_d^i(W,b,u,z) \right]$$

$$\text{where } \mathcal{L}_d^i(W,b,u,z) = \mathcal{L}_d(G_d(G_f(x_i;W,b);u,z),d_i).$$

$$E(W,V,b,c,u,z)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_y^i(W,b,V,c) - \lambda \left(\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_d^i(W,b,u,z) + \frac{1}{n'}\sum_{i=n+1}^{N} \mathcal{L}_d^i(W,b,u,z) \right),$$

where we are seeking the parameters $\hat{W}, \hat{V}, \hat{b}, \hat{c}, \hat{u}, \hat{z}$ that deliver a saddle point given by

$$(\hat{W},\hat{V},\hat{b},\hat{c}) = \arg\min_{W,V,b,c} E(W,V,b,c,\hat{u},\hat{z}), \quad (1)$$

$$(\hat{u},\hat{z}) = \arg\max_{\hat{u},\hat{z}} E(\hat{W},\hat{V},\hat{b},\hat{c},u,z). \quad (2)$$

Demo

Load Data

- 라이브러리 불러오기

```
import tensorflow as tf
from tensorflow.python.keras.layers import Dense, Flatten, Conv2D, MaxPool2D
from tensorflow.python.keras import Model
import numpy as np
import matplotlib.pyplot as plt
```

- MNIST 데이터 불러오기

```
!wget https://url.kr/6ri2mk -O 'data.zip'
!unzip data.zip -d './data'
```

```
mnist_train = np.load('./data/train_mnist.npz')
x_train, y_train  = mnist_train['x'], mnist_train['y']
mnist_test = np.load('./data/test_mnist.npz')
x_test, y_test  = mnist_test['x'], mnist_test['y']

train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(100).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

Load Data

- SVHN 데이터 불러오기

```
svhn_train = np.load('./data/train_svhn.npz')
svhn_train_ls, svhn_train_y  = svhn_train['x'], svhn_train['y']
svhn_test = np.load('./data/test_svhn.npz')
svhn_test_ls, svhn_test_y  = svhn_test['x'], svhn_test['y']

svhn_train_ds = tf.data.Dataset.from_tensor_slices((svhn_train_ls,svhn_train_y)).batch(32)
svhn_test_ds = tf.data.Dataset.from_tensor_slices((svhn_test_ls,svhn_test_y)).batch(32)
```

- 표준화 정보 추출

```
all_train_domain_images = np.vstack((x_train, mnist_m_train_ls))
channel_mean = all_train_domain_images.mean((0,1,2))
channel_mean
```

```
array([73.41482781, 73.1164477 , 75.94361926])
```

Build Model

- 피쳐 생성기 클래스 생성

```
class FeatureGenerator(Model):
    def __init__(self):
        super(FeatureGenerator, self).__init__()
        self.normalise = lambda x: (tf.cast(x, tf.float64) - channel_mean) / 255.0
        self.conv1 = Conv2D(64, 5, activation='relu')
        self.conv2 = Conv2D(128, 5, activation='relu')
        self.maxpool = MaxPool2D(2)
        self.flatten = Flatten()

    def call(self, x):
        x = self.normalise(x)
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.maxpool(x)

        return self.flatten(x)

feature_generator = FeatureGenerator()
```

Build Model

- 라벨 예측기 클래스 생성

```
class LabelPredictor(Model):
    def __init__(self):
        super(LabelPredictor, self).__init__()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10, activation='softmax')

    def call(self, feats):
        feats = self.d1(feats)
        return self.d2(feats)

label_predictor = LabelPredictor()
```

Build Model

- 도메인 예측기 클래스 생성

```
class DomainPredictor(Model):
    def __init__(self):
        super(DomainPredictor, self).__init__()
        self.d3 = Dense(64, activation='relu')
        self.d4 = Dense(2, activation='softmax')

    def call(self, feats):
        feats = self.d3(feats)
        return self.d4(feats)

domain_predictor = DomainPredictor()
```

Define Loss Function

- 최적화 모듈 선언

```
d_optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
f_optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
```

- 손실 함수 선언

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()

train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')

m_test_loss = tf.keras.metrics.Mean(name='m_test_loss')
m_test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='m_test_accuracy')

conf_train_loss = tf.keras.metrics.Mean(name='c_train_loss')
conf_train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='c_train_accuracy')
```

Define Loss Function

- 도메인 예측 라벨 및 생성자 선언

```
x_train_domain_labels = np.ones([len(x_train)])
mnist_m_train_domain_labels = np.zeros([len(mnist_m_train_ls)])
all_train_domain_labels = np.hstack((x_train_domain_labels, mnist_m_train_domain_labels))

domain_train_ds = tf.data.Dataset.from_tensor_slices(
    (all_train_domain_images, tf.cast(all_train_domain_labels, tf.int8))).shuffle(60000).batch(32)
```

Training

- 학습 함수 선언

```
@tf.function
def train_step(images, labels, images2, domains, alpha):

    with tf.GradientTape(persistent=True) as tape:
        features = feature_generator(images)
        l_predictions = label_predictor(features)
        features = feature_generator(images2)
        d_predictions = domain_predictor(features)
        label_loss = loss_object(labels, l_predictions)
        domain_loss = loss_object(domains, d_predictions)

    f_gradients_on_label_loss = tape.gradient(label_loss, feature_generator.trainable_variables)
    f_gradients_on_domain_loss = tape.gradient(domain_loss, feature_generator.trainable_variables)
    f_gradients = [f_gradients_on_label_loss[i] - alpha*f_gradients_on_domain_loss[i] for i in range(
        len(f_gradients_on_domain_loss))]

    l_gradients = tape.gradient(label_loss, label_predictor.trainable_variables)
    f_optimizer.apply_gradients(zip(f_gradients+l_gradients,
                                    feature_generator.trainable_variables+label_predictor.trainable_variables))
```

Training

- 테스트 함수 선언

```
@tf.function
def test_step(mnist_images, labels, mnist_m_images, labels2):
    features = feature_generator(mnist_images)
    predictions = label_predictor(features)
    t_loss = loss_object(labels, predictions)

    test_loss(t_loss)
    test_accuracy(labels, predictions)

    features = feature_generator(mnist_m_images)
    predictions = label_predictor(features)
    t_loss = loss_object(labels2, predictions)

    m_test_loss(t_loss)
    m_test_accuracy(labels2, predictions)
```

Training

- 메트릭 초기화 함수 선언

```
def reset_metrics():
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()
    m_test_loss.reset_states()
    m_test_accuracy.reset_states()
```

Training

- 학습 함수 선언

```
with tf.GradientTape() as tape:
    features = feature_generator(images2)
    d_predictions = domain_predictor(features)
    domain_loss = loss_object(domains, d_predictions)

d_gradients = tape.gradient(domain_loss, domain_predictor.trainable_variables)
d_gradients = [alpha*i for i in d_gradients]
d_optimizer.apply_gradients(zip(d_gradients, domain_predictor.trainable_variables))

train_loss(label_loss)
train_accuracy(labels, l_predictions)
conf_train_loss(domain_loss)
conf_train_accuracy(domains, d_predictions)
```

Training

- 학습 시작

```
EPOCHS = 100
alpha = 1

train_acc = []
conf_train_acc = []
test_acc = []
m_test_acc = []
for epoch in tqdm(range(EPOCHS)):
    reset_metrics()

    for domain_data, label_data in zip(domain_train_ds, train_ds):

        try:
            train_step(label_data[0], label_data[1], domain_data[0], domain_data[1], alpha=alpha)

        #End of the smaller dataset
        except ValueError:
            pass
```

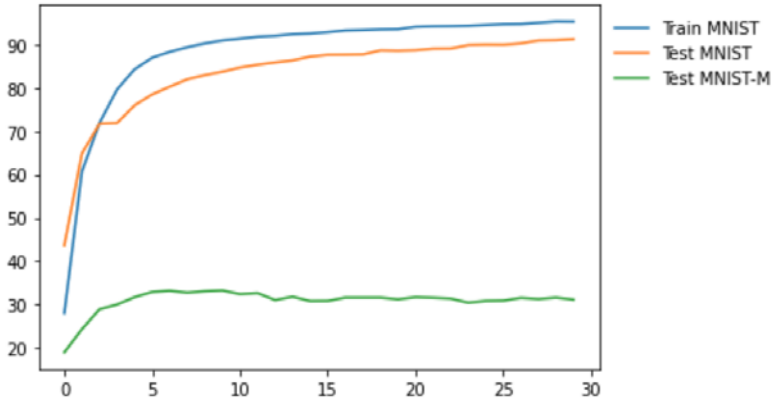
Training

- 학습 시작

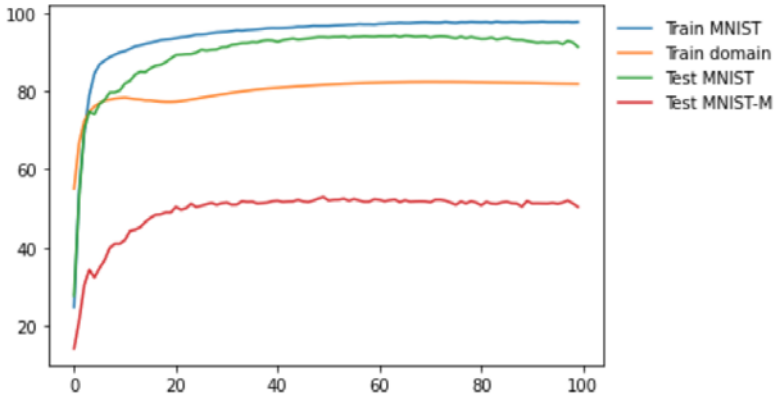
```
for test_data, m_test_data in zip(test_ds,mnist_m_test_ds):
    test_step(test_data[0], test_data[1], m_test_data[0], m_test_data[1])

template = 'Epoch {}, Train Accuracy: {}, Domain Accuracy: {}, Source Test Accuracy: {}, Target Test Accuracy: {}'
print (template.format(epoch+1,
                       train_accuracy.result()*100,
                       conf_train_accuracy.result()*100,
                       test_accuracy.result()*100,
                       m_test_accuracy.result()*100,))

train_acc.append(train_accuracy.result()*100)
conf_train_acc.append(conf_train_accuracy.result()*100)
test_acc.append(test_accuracy.result()*100)
m_test_acc.append(m_test_accuracy.result()*100)
```

Test

- Baseline CNN 정확도
  - 소스 데이터 테스트 정확도 : 91.27 %
  - 타겟 데이터 테스트 정확도 : 31.13 %
- DANN 정확도
  - 소스 데이터 테스트 정확도 : 91.30 %
  - 타겟 데이터 테스트 정확도 : 50.40 %



Baseline CNN



DANN