데이터->모델->평가->손실 줄이기(최적화)->다시 모델링->결과

Prediction/Logit: 각 class 별로 예측한 값

Loss/Cost : 얼마나 틀렸는지 계산

Optimization : loss 최소화

Result : 평가

**딥러닝 용어**

CNN(Convolution neuron network)

Convolution : 합성곱 /이미지와 합성해서 특징을 뽑아냄.

Weight-학습하려고 하는 대상

Y = wa + b(bias)

Pooling layer : 이미지가 가지고 있는 특징을 줄인다.(압축)

Activation function : 앞에서 특징을 뽑고, 불필요한 음수 부분 없애기(ReLU)

Softmax : 수치를 유도 /softmax를 거쳐 모든 값 합이 1이 되도록 만든다.

Loss /cost function : 얼마나 틀렸는지 계산

Optimization : loss function을 최소로

Learning rate : learning rate가 너무 낮아도 높아도 안좋다. / 적정 조절

Batch Size : 몇 장을 넣을건지 정하는 것

Epoch : batch size 다음 에폭 수만큼 다시 봐야한다.

Label/ground truth: 데이터를 받으면 데이터에 대한 정답(레이블)

**CNN모델 구조**

Feature extraction : 특징 추출->fully connected layer에서 결정을 내림

1.Convolution Layer : 특징을 합성

2.Pooling layer(Max Pooling) : 특성을 뽑은 것 중 가장 중요한 것 뽑기(가장 큰 특성 압축)

3.Activation function(ReLU) : 0미만 없애기

Tensor

np\_argmax(arr) : 가장 큰 인덱스의 위치

np\_unique(arr) : 중복 제외하고 유니크한 값

**시각화**

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline : 주피터 내부에 그래프를 띄우겠다.

**이미지**

이미지 합치기

Import cv2

dog\_image = cv2.resize(image, (275, 183)) #강아지 사이즈 조정

dog\_image.shape

out[] ((183, 275, 3), (183, 275, 3))

dog\_image.shape, cat\_image.shape #강아지랑 고양이랑 사이즈 같다.

이미지를 합칠 때 투명도(alpha)를 주면서 이미지 합치기

2-02 TensorFlow 기초 사용법

Tensor 생성 list -> Tensor / array->Tensor

tf.constant([1, 2, 3])

데이터 타입 적용 : tf.constant([1, 2, 3],dtype=tf.float32)

데이터 타입 변환 : tf.cast(tensor, dtype = tf.uInt8)

tf.random.normal([3, 3])

tf.random.uniform([4, 4])

Data Preprocess(MNIST)

from tensorflow.keras import datasets

mnist = datasets.mnist

(tranin\_x, train\_y), (test\_x, test\_y) = mnist.load\_data()

image = train\_x[0] //데이터 하나만 뽑기

image.shape

plt.imshow(image, ‘gray’) //시각화

plt.show()

**Channel 관련**

차원수 늘리기 : new\_train\_x = np.expand\_dims(train\_x, -1) # 맨뒤에 1붙이기

new\_train\_x.shape

disp = new\_train\_x[0, :, :, 0] / disp = np.squeeze(new\_train\_x[0])

disp.shape #결과 : (28, 28) shape를 줄이기

원 핫 인코딩 : 컴퓨터가 이해할 수 있는 형태로 변환해서 label을 주도록 함.

from tensorflow.keras.utils import to\_categorical

**Layer Explaination**

Input image

-os, glob, matplotlib

kernel\_size: filter(weight)의 사이즈

strides : 몇 개의 pixel을 skip하면서 훑어 지날지

padding : zero padding만들건지 ,VALID 는 padding 없고, SAME은 padding 있다.

activation: activation function만들건지

**Visualization**

output = layer(image)

pooling : 이미지가 줄어든다.

tf.keras.layers.MaxPool2D(pool\_size=(2, 2), strides=(2, 2), padding='SAME')

Fully Connected : y = wx+b

flatten

Optimization & Training(Beginner)(모델 학습)

from tensorflow.keras import layers

from tensorflow.keras import datasets

crossentropy : 2개 -> binary\_crossentropy / 2개 이상 : categorical\_crossentropy

원핫인코딩을 주지 않았을 때 : sparse\_categorical\_crossentropy

원핫인코딩을 주었을 때 : categorical\_crossentropy

Compile – Optimizer 적용

모델 평가 : accuracy를 이름으로 넣기

Training : 학습용 Hyperparameter 설정

epoch ->데이터를 하나씩 보는데 다 보면 한 epoch

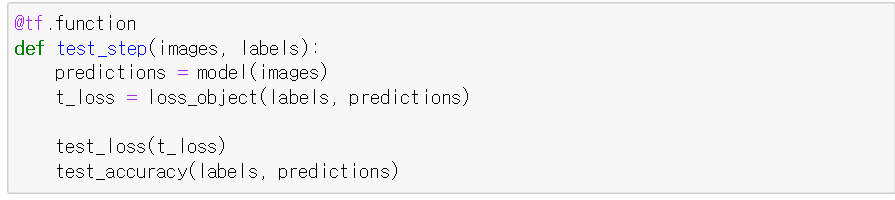
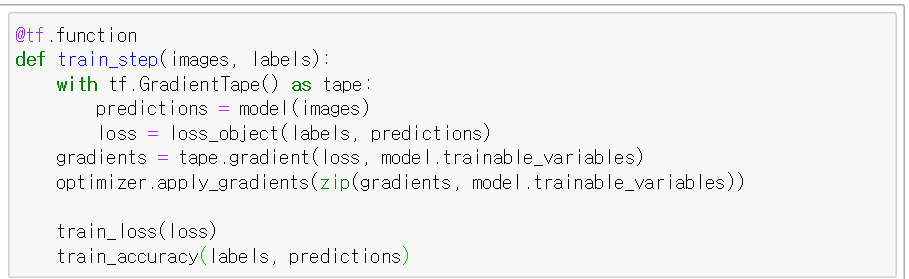
batch\_size만큼 한 모델에 넣어줘야함.(메모리의 효율 위해)

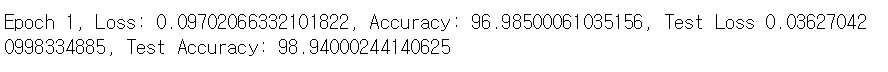
model.fit(train\_x, train\_y,

batch\_size=batch\_size,

shuffle=True,

epochs=num\_epochs)





에폭당 로그 hist.history

Evaluating ( 비교 대상이 있다.)

model.evaluate(test\_x, test\_y, batch\_size = batch\_size)

모델의 Input Data로 확인 할 이미지 데이터 넣기

pred = model.predict(test\_image.reshape(1, 28, 28, 1))

pred.shape ->(1, 10)

노드가 가장 높은 자극을 받은 것이 정답

pred가 가장 높은값 np.argmax(pred)

Test Batch->이미지를 테스트 배치로 넣기 (예측만 하기)

test\_batch = test\_x[:32]

test\_batch.shape

preds = model.predict(test\_batch)

preds.shape

np.argmax(preds, -1)

PyTorch

AutoGrad : 기울기를 주고 학습이 가능하도록

x = torch.ones(2, 2, requires\_grad = True)

DataLoader를 불러 model에 넣기

Convolution

in\_channels, out\_channels, kernel\_size, stride

weight는 detach() 해서 빼야한다.

weight.detach().numpy()

Optimization

model, Optimization 설정

model.train() #train mode

data, target = next(iter(train\_loader))

data, target = data.to(device), target.to(device)

gradients를 clear해서 새로운 최적화 값을 찾기 위해 준비

loss.backward() # 기울기 계산

optimizer.step() # 업데이트

Start Training

epochs = 1

log\_interval = 100

enumerate 사용

for epoch in range(1, epochs+1):

# Train Mode

model.train()

for batch\_idx, (data, target) in enumerate(train\_loader):

data, target = data.to(device), target.to(device)

optimizer.zero\_grad()

output = model(data)

loss = F.nll\_loss(output, target)

loss.backward()

optimizer.step()

if batch\_idx % log\_interval == 0:

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, batch\_idx \* len(data), len(train\_loader.dataset),

100 \* batch\_idx / len(train\_loader), loss.item()

))

model.eval()

test\_loss = 0

correct = 0

with torch.no\_grad():

for data, target in test\_loader:

data, target = data.to(device), target.to(device)

output = model(data)

test\_loss += F.nll\_loss(output, target, reduction='sum').item()

pred = output.argmax(dim=1, keepdim=True)

correct += pred.eq(target.view\_as(pred)).sum().item()

test\_loss /= len(test\_loader.dataset)

print('\nTest set: Average Loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

test\_loss, correct, len(test\_loader.dataset), 100. \* correct / len(test\_loader.dataset)))

데이터 만들기, 모델, 최적화, 컴파일, 트레이닝, 평가

torchvision : 이미지를 관리하기 위함.

**part 3** **이미지 분석으로 배우는 tensorflow**

Augmentation : 여러 환경에서 적응 가능하도록 트레이닝

callbacks : epoch or step 단위로 이벤트를 일으키는 옵션, 정해진 시간대에 running rate를 주는

모델 저장 및 불러오기

from glob import glob # 외부 파일 불러오기

PIL->이미지 열 때

os.getcwd() # 현재 경로

os.listdir()

os.listdir(‘dataset/mnist\_png/training/0/’) # 폴더 경로

glob(dataset/mnist\_png/training/0/\*.png’) # 경로가 포함된 모든 파일

data\_paths[-1] : 마지막에 있는 것 가져오기

path = data\_paths[0]

path # 첫번째에 있는 것 경로 불러오기

데이터 분석(MNIST)

os.listdir(‘dataset/mnist\_png/training/’) # 데이터 확인

데이터별 개수 비교

nums\_dataset = []

for lbl\_n in label\_nums:

data\_per\_class = os.listdir(‘dataset/mnist\_png/training/’ + lbl\_n) #각 레이블별 데이터

데이터 셋을 클래스 별로 확인

Pillow로 열기

image\_pil = Image.open(path)

image = np.array(image\_pil)

TensorFlow 로 열기

gfile = tf.io.read\_file(path)

image = tf.io.decode\_image(gfile)

일단 채널이 포함된 상태로 시작 TensorShape([28, 28, 1]) #1이 채널

Label얻기

path

path.split(‘\\’) # \기준으로 쪼개짐

int(label) : 숫자로 변환

데이터 이미지 사이즈 알기

input shape가 정해져 있어 사이즈를 통일 시켜야 함.

from tqdm import tqdm\_notebook

heights = []

widths = []

len(data\_paths[:10])

함수로 정의해서 이용

heights = []

widths = []

for path in tqdm\_notebook(data\_paths):

image\_pil = Image.open(path)

image = np.array(image\_pil)

h, w = image.shape

heights.append(h)

widths.append(w)

plt.figure(figsize=(20, 10))

plt.subplot(121)

plt.hist(heights)

plt.title(‘Heights’)

plt.axvline(np.mean(heights), color = ‘r’, linestyle=’dashed’, linewidth=2) # 평균값

plt.subplot(122)

plt.hist(widths)

plt.title(‘Widths’)

plt.show()

이미지를 배치 사이즈만큼 잘라서 넣어주기

배피 사이즈만큼 모델에 넣기

배치가 다 돌아가면 한 에폭

Images in List

batch\_image = []

for path in data\_paths[:8]: #데이터패스중 8개만 받기

image = read\_image(path)

batch\_image.append(image)

plt.imshow(batch\_images[0])

plt.show()

batch = tf.convert\_to\_tensor(batch\_images)

batch.shape

(batch\_size, height, width, channel)

데이터가 4차원

def make\_batch(batch\_paths):

batch\_images = []

for path in batch\_paths:

image = read\_image(path)

batch\_images.append(Image)

return tf.convert\_to\_tensor(batch\_images\_

batch\_size = 16;

for step in range[4]:

batch\_images = make\_batch(data\_paths[step\*batch\_size : (step + 1) \* batch\_size])

plt imshow(batch\_images[0])

plt.show() # 시각화

batch\_images.shape

**data generator** : 데이터 모델링 간편하게

data\_paths = os.listdir(‘dataset/mnist\_png/0/\*.png’)

data\_paths[0]

Load Image

gfile = tf.io.read\_file(path)

image = tf.io.decade\_image(gfile) #이미지 열기

image.shape

plt.imshow(image[:, :, 0], ‘gray’)

plt.show()

Set Data Generator

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator( #데이터에 변환을 주면서 이미지 학습

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

horizontal\_flip=True)

image.shape

inputs = image[tf.newaxis, …] #차원 수 늘리기

inputs.shape

image = next(iter(datagen.flow(inputs)))

image.shape

**Transformation**

**data generator - 변환 주기**

datagen = ImageDataGenerator(

width\_shift\_range=0.3 #0.3만큼 랜덤하게 변환

zom\_range = 0.3 #위로 옮겨지거나 아래로 옮겨지거나

)

outputs = next(iter(datagen.flow(inputs)))

rescale : trainset, testset 모두 해야한다.

train\_datagen = ImageDataGenerator(

zoom\_range = 0.7, #train에만 해준다.

rescale = 1./255.)

test\_datagen = ImageDataGenerator(

rescale=1./255

)

Preprocess

train\_datagen = ImageDataGenerator(

rescale = 1./255.,

width\_shift\_range = 0.3,

zoom\_range = 0.2,

horizontal\_flip = True

)

test\_datagen = ImageDataGenerator(rescale=1./255)

폴더 별로 데이터가 있어

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=input\_shape[:2], #채널 빼고 2개

batch\_size=batch\_size,

color\_mode='grayscale',

class\_mode = ‘categorical’

)

validation\_generator = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size=input\_shape[:2],

batch\_size=batch\_size,

color\_mode='grayscale'

class\_mode = ‘categorical’

)

Training

model.fit\_generator(

train\_generator,

steps\_per\_epoch=len(train\_generator),

epochs=num\_epochs,

validation\_data=validation\_generator,

validation\_steps=len(validation\_generator))

strip : 빈공간 없애기

os.path.basename(path) #파일명만 가져오기

os.path.exist(path) #path가 있는지 없는지

class 수 확인

classes\_name = []

for path in train\_paths:

cls\_name = get\_class\_name(path)

class\_names.append(cls\_name)

class\_names = [get\_class\_name(path) for path in train\_paths]

unique\_classes = np.unique(classes\_name, return\_counts=True) #클래스가 몇 개 있는지 (counts)

unique\_classes

plt.bar(\*unique\_classes)

plt.xticks(rotation=45)

plt.show()

DataFrame 생성

data\_ex = {'a':[1, 2, 3], 'b':[10, 20, 30], 'c':[100, 200, 300]}

df\_ex = pd.DataFrame(data\_ex)

df\_ex

data = {'path': train\_paths, 'class\_name': classes\_name}

df = pd.DataFrame(data)

df.head()

만들어진 DataFrame 저장

train\_csv\_path = ‘train\_dataset.csv’

test\_csv\_path = ‘test\_dataset.csv’

dataframe 이용해서 학습하기

preprocess

import pandas as pd

train\_df = pd.read\_csv('train\_dataset.csv')

test\_df = pd.read\_csv('test\_dataset.csv')

train\_df.head()

train\_datagen = ImageDataGenerator(

rescale=1./255,

width\_shift\_range=0.3,

zoom\_range=0.2,

horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_dataframe(

train\_df,

x\_col='path',

y\_col='class\_name',

target\_size=input\_shape[:2],

batch\_size=batch\_size

)

validation\_generator = test\_datagen.flow\_from\_dataframe(

test\_df,

x\_col='path',

y\_col='class\_name',

target\_size=input\_shape[:2],

batch\_size=batch\_size

)

tf.data

def read\_image(path):

gfile = tf.io.read\_file(path)

image = tf.io.decode\_image(gfile, dtype=tf.float32)

return image

dataset = tf.data.Dataset.from\_tensor\_slices(train\_paths)

dataset = dataset.map(read\_image, num\_parallel\_calls=AUTOTUNE)

배치로 묶어서 모델에 넣어주기

dataset = tf.data.Dataset.from\_tensor\_slices(train\_paths)

dataset = dataset.map(read\_image)

dataset = dataset.batch(4) #batch\_size: 4만큼 묶인다.

**Shuffle(섞기)**

Label하고 같이 넣기

dataset = tf.data.Dataset.from\_tensor\_slices((train\_paths, labels))

dataset = dataset.map(load\_data, num\_parallel\_calls=AUTOTUNE)

dataset = dataset.batch(4)

dataset = dataset.shuffle(buffer\_size=len(train\_paths))

dataset = dataset.repeat() # 반복적으로 돌아갈 수 있도록

tensorflow 함수로 label얻기

def onehot\_encoding(label):

return np.array(class\_names == label, np.uint8)

Data Preprocess

train\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_paths)

train\_dataset = train\_dataset.map(load\_image\_label, num\_parallel\_calls=AUTOTUNE)

train\_dataset = train\_dataset.map(image\_preprocess, num\_parallel\_calls=AUTOTUNE)

train\_dataset = train\_dataset.batch(batch\_size)

train\_dataset = train\_dataset.shuffle(buffer\_size=len(train\_paths))

train\_dataset = train\_dataset.repeat()

test\_dataset = tf.data.Dataset.from\_tensor\_slices(test\_paths)

test\_dataset = test\_dataset.map(load\_image\_label, num\_parallel\_calls=AUTOTUNE)

test\_dataset = test\_dataset.batch(batch\_size)

test\_dataset = test\_dataset.repeat()

training

steps\_per\_epoch = len(train\_paths) // batch\_size

validation\_steps = len(test\_paths) // batch\_size

model.fit\_generator(

train\_dataset,

steps\_per\_epoch=steps\_per\_epoch,

validation\_data=test\_dataset,

validation\_steps=validation\_steps,

epochs=num\_epochs

)

**callbacks:학습 도중 이벤트**

tensorboard열기

callbacks

logdir = os.path.join('logs', datetime.now().strftime("%Y%m%d-%H%M%S"))

tf.keras.callbacks.TensorBoard(

log\_dir = logdir,

write\_graph = True,

write\_images = True,

histogram\_freq = 1

)

%tensorboard --logdir logs --port 8008

Training

LamdaCallback : 맞춤형 그래프 가져오기(복붙)

# Define the per-epoch callback.

cm\_callback = tf.keras.callbacks.LambdaCallback(on\_epoch\_end=log\_confusion\_matrix)

Expert : 시간을 정해서 넣을 수 있다.

logdir = os.path.join('logs', datetime.now().strftime("%Y%m%d-%H%M%S"))

file\_writer = tf.summary.create\_file\_writer(logdir)

for epoch in range(num\_epochs):

for step, (images, labels) in enumerate(train\_dataset):

train\_step(images, labels)

with file\_writer.as\_default():

tf.summary.image('input\_image', images, step=step)

tf.summary.scalar('loss', train\_loss.result(), step=step)

for test\_images, test\_labels in test\_dataset:

test\_step(test\_images, test\_labels)

template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {}'

print (template.format(epoch+1,

train\_loss.result(),

train\_accuracy.result()\*100,

test\_loss.result(),

test\_accuracy.result()\*100))

**callbacks-learing rate schedule : 최적화된 위치까지 도달하게 하는 것**

def scheduler(epoch):

if epoch < 10:

return 0.001

else:

return 0.001 \* math.exp(0.1 \* (10 - epoch))

learning\_rate\_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler)

**checkpoint** : 모델이 학습하다가 weight를 저장시킴. 나중에 weight로 돌아갈 수 있도록

save\_path = 'checkpoints'

checkpoint = tf.keras.callbacks.ModelCheckpoint(save\_path, monitor='val\_accuracy', verbose=1, save\_best\_only=True, mode='max')

val\_accuracy가 올라가면 저장하고, 아니면 저장안함.(save\_best\_only=True)

val\_accuracy이면 mode = ‘max’ , loss이면 ‘min’

history 들여다보기

history.history.keys()

history.params

new\_model = history.model

plt.plot(history.history[‘accuracy’])

plt.plot(history.history[‘val\_accuracy’])

plt.title(“Model Accuracy”)

plt.ylabel(‘accuracy’)

plt.xlabel(‘epoch’)

plt.legend([‘tran’, ‘validation’])

plt.show()

plt.plot(history.history[‘loss’])

plt.plot(history.history[‘val\_loss’])

plt.title(‘Model Loss’)

plt.ylabel(‘loss’)

plt.xlabel(‘epoch’)

plt.legend([‘tran’, ‘validation’])

plt.show()

이미지를 load 직접load해서 넣는 방법

path = train\_paths[0]

test\_image, test\_label = load\_image\_label(path)

test\_image.shape

test\_image = test\_image[tf.newaxis, ...]

test\_image.shape

pred = model.predict(test\_image)

pred

generator에서 데이터 가져오는 방법

generator에 넣는 방법

pred = model.predict\_generator(test\_dataset.take(1)) #한 배치만 가져온다

evals = model.evaluate(image, label)

tf케라스로 저장하기

save\_path=’my\_model.h5’

model.save(save\_path, include\_optimizer=True)

tf.keras.models.load\_moel(‘my\_model.h5’) #모델 불러오기

모델을 weight만 저장하기

model.save\_weights(‘model\_weights.h3)

with open('model\_architecture.json', 'w') as f:

f.write(model.to\_json())

from tensorflow.keras.models import model\_from\_json

with open('model\_architecture.json', 'r') as f:

model = model\_from\_json(f.read())

model.load\_weights('model\_weights.h5')