데이터셋 불러오기

```
from google.colab import drive
drive.mount('Mydrive')
```

```
Drive already mounted at Mydrive; to attempt to forcibly remount, call drive.mount("Mydrive", force_remount=True).
```

```
pwd
```

```
/content/Mydrive/MyDrive/project/output
```

필요한 라이브러리를 가져오기

굵은 텍스트

```
# import package
# model
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchsummary import summary
from torch import optim
```

```
# dataset and transformation
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import os
```

```
# display images
from torchvision import utils
import matplotlib.pyplot as plt
%matplotlib inline

# utils
import numpy as np
from torchsummary import summary
import time
import copy
```

```
from tqdm.auto import tqdm
```

```
!pip install jcopdl
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: jcopdl in /usr/local/lib/python3.8/dist-packages (2.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from jcopdl) (1.21.6)
Requirement already satisfied: pillow in /usr/local/lib/python3.8/dist-packages (from jcopdl) (7.1.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from jcopdl) (3.2.2)
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages (from jcopdl) (1.13.0+cu116)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from jcopdl) (1.3.5)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->jcopdl) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->jcopdl) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->jcopdl) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->jcopdl) (3.0.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib->jcopdl) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->jcopdl) (2022.6)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torch->jcopdl) (4.4.0)
```

```
!pip install imutils
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: imutils in /usr/local/lib/python3.8/dist-packages (0.5.4)
```

```
from jcopdl.callback import Callback, set_config
```

```
from jcopdl.layers import linear_block, conv_block
```

```
from imutils import paths
```

데이터셋 불러오기

```
# specify path to data
path2data = '/content/Mydrive/MyDrive/project/data'

#load dataset
train_dir = '/content/Mydrive/MyDrive/project/output/train/'
validation_dir = '/content/Mydrive/MyDrive/project/output/val/'
test_dir = '/content/Mydrive/MyDrive/project/output/test/'
```

```
bs = 128
crop_size = 224
```

```
train_transform = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(crop_size, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

```
test_transform = transforms.Compose([
    transforms.Resize(230),
    transforms.CenterCrop(crop_size),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

```
print(train_transform)
print(test_transform)
```

```
Compose(
    RandomRotation(degrees=[-10.0, 10.0], interpolation=nearest, expand=False, fill=0)
    RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0), ratio=(0.75, 1.3333), interpolation=bilinear), antialias=None)
    RandomHorizontalFlip(p=0.5)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
Compose(
    Resize(size=230, interpolation=bilinear, max_size=None, antialias=None)
    CenterCrop(size=(224, 224))
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

```
train_set = datasets.ImageFolder(train_dir, transform=train_transform)
trainloader = DataLoader(train_set, batch_size=bs, shuffle=True, num_workers=2)
```

```
print(train_set)
print(trainloader)
```

```
Dataset ImageFolder
    Number of datapoints: 200
    Root location: /content/Mydrive/MyDrive/project/output/train/
    StandardTransform
Transform: Compose(
               RandomRotation(degrees=[-10.0, 10.0], interpolation=nearest, expand=False, fill=0)
               RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0), ratio=(0.75, 1.3333), interpolation=bilinear), antialias=None)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
<torch.utils.data.dataloader.DataLoader object at 0x7fb9aa7cf6d0>
```

```
test_set = datasets.ImageFolder(test_dir, transform=test_transform)
testloader = DataLoader(test_set, shuffle=True)
```

```
print(test_set)
print(testloader)
```

```
Dataset ImageFolder
    Number of datapoints: 25
    Root location: /content/Mydrive/MyDrive/project/output/test/
    StandardTransform
Transform: Compose(
               Resize(size=230, interpolation=bilinear, max_size=None, antialias=None)
               CenterCrop(size=(224, 224))
```

```
                ToTensor()
                Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
            )
    <torch.utils.data.dataloader.DataLoader object at 0x7fb92f78d7c0>
```

```
val_set = datasets.ImageFolder(validation_dir, transform=test_transform)
valloader = DataLoader(val_set, batch_size=bs, shuffle=True)
```

```
print(val_set)
print(valloader)
```

```
    Dataset ImageFolder
        Number of datapoints: 25
        Root location: /content/Mydrive/MyDrive/project/output/val/
        StandardTransform
    Transform: Compose(
                Resize(size=230, interpolation=bilinear, max_size=None, antialias=None)
                CenterCrop(size=(224, 224))
                ToTensor()
                Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
            )
    <torch.utils.data.dataloader.DataLoader object at 0x7fb92f78d850>
```

샘플이미지 확인

```
# check sample images
def show(img, y=None):
    npimg = img.numpy()
    npimg_tr = np.transpose(npimg, (1, 2, 0))
    plt.imshow(npimg_tr)

    if y is not None:
        plt.title('labels:' + str(y))

np.random.seed(10)
torch.manual_seed(0)
```

```
    <torch._C.Generator at 0x7fb933179350>
```

```
print(torch.manual_seed(0))
```

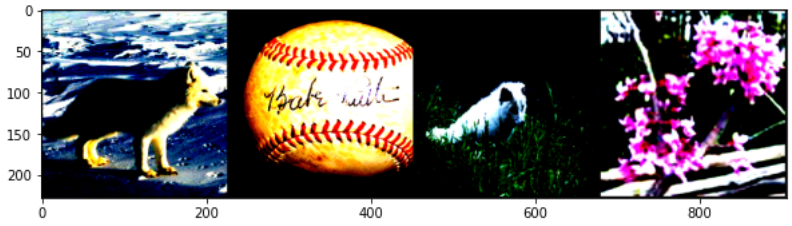
```
    <torch._C.Generator object at 0x7fb933179350>
```

```
grid_size=4
rnd_ind = np.random.randint(0, len(train_set), grid_size)
```

```
x_grid = [train_set[i][0] for i in rnd_ind]
```

```
x_grid = utils.make_grid(x_grid, nrow=grid_size, padding=2)
plt.figure(figsize=(10,10))
show(x_grid)
```

```
    WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



MobileNetV1 모델구축

MobileNetV1은 Depthwise separable convolution을 반복하여 쌓은 구조입니다. 따라서 Depthwise separable convolution 클래스를 정의합니다.

```
# Depthwise Separable Convolution
class Depthwise(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()

        self.depthwise = nn.Sequential(
            nn.Conv2d(in_channels, in_channels, 3, stride=stride, padding=1, groups=in_channels, bias=False),
            nn.BatchNorm2d(in_channels),
            nn.ReLU6(),
```

```
        )

        self.pointwise = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU6()
        )

    def forward(self, x):
        x = self.depthwise(x)
        x = self.pointwise(x)
        return x
```

BasicConv2d 클래스도 정의하면 편리하게 모델을 구축할 수 있습니다.

```
# Basic Conv2d
class BasicConv2d(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, **kwargs):
        super().__init__()

        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, **kwargs),
            nn.BatchNorm2d(out_channels),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.conv(x)
        return x
```

위에서 정의한 클래스를 활용해서 MobileNetV1을 구축하겠습니다. MobileNet은 하이퍼파라미터가 존재합니다. 채널 수를 조절하는 width_multiplier 파라미터를 신경써서 구축합니다.

```
# MobileNetV1
class MobileNet(nn.Module):
    def __init__(self, width_multiplier, num_classes=10, init_weights=True):
        super().__init__()
        self.init_weights=init_weights
        alpha = width_multiplier

        self.conv1 = BasicConv2d(3, int(32*alpha), 3, stride=2, padding=1)
        self.conv2 = Depthwise(int(32*alpha), int(64*alpha), stride=1)
        # down sample
        self.conv3 = nn.Sequential(
            Depthwise(int(64*alpha), int(128*alpha), stride=2),
            Depthwise(int(128*alpha), int(128*alpha), stride=1)
        )
        # down sample
        self.conv4 = nn.Sequential(
            Depthwise(int(128*alpha), int(256*alpha), stride=2),
            Depthwise(int(256*alpha), int(256*alpha), stride=1)
        )
        # down sample
        self.conv5 = nn.Sequential(
            Depthwise(int(256*alpha), int(512*alpha), stride=2),
            Depthwise(int(512*alpha), int(512*alpha), stride=1),
            Depthwise(int(512*alpha), int(512*alpha), stride=1),
            Depthwise(int(512*alpha), int(512*alpha), stride=1),
            Depthwise(int(512*alpha), int(512*alpha), stride=1),
            Depthwise(int(512*alpha), int(512*alpha), stride=1),
        )
        # down sample
        self.conv6 = nn.Sequential(
            Depthwise(int(512*alpha), int(1024*alpha), stride=2)
        )
        # down sample
        self.conv7 = nn.Sequential(
            Depthwise(int(1024*alpha), int(1024*alpha), stride=2)
        )

        self.avg_pool = nn.AdaptiveAvgPool2d((1,1))
        self.linear = nn.Linear(int(1024*alpha), num_classes)

        # weights initialization
        if self.init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.conv6(x)
        x = self.conv7(x)
```

```
        x = self.avg_pool(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x

    # weights initialization function
    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)

def mobilenet(alpha=1, num_classes=10):
    return MobileNet(alpha, num_classes)
```

구축한 모델을 확인합니다.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
x = torch.randn((3, 3, 224, 224)).to(device)
model = mobilenet(alpha=1).to(device)
output = model(x)
print('output size:', output.size())
```

```
output size: torch.Size([3, 10])
```

model summary를 출력합니다.

```
summary(model, (3, 224, 224), device=device.type)
```

```
# define loss function, optimizer, lr_scheduler
loss_func = nn.CrossEntropyLoss(reduction='sum')
opt = optim.Adam(model.parameters(), lr=0.01)

from torch.optim.lr_scheduler import ReduceLROnPlateau
lr_scheduler = ReduceLROnPlateau(opt, mode='min', factor=0.1, patience=10)
```

```
# get current lr
def get_lr(opt):
    for param_group in opt.param_groups:
        return param_group['lr']


# calculate the metric per mini-batch
def metric_batch(output, target):
    pred = output.argmax(1, keepdim=True)
    corrects = pred.eq(target.view_as(pred)).sum().item()
    return corrects
```

```
# calculate the loss per mini-batch
def loss_batch(loss_func, output, target, opt=None):
    loss_b = loss_func(output, target)
    metric_b = metric_batch(output, target)

    if opt is not None:
        opt.zero_grad()
        loss_b.backward()
        opt.step()

    return loss_b.item(), metric_b
```

```
# calculate the loss per epochs
def loss_epoch(model, loss_func, dataset_dl, sanity_check=False, opt=None):
    running_loss = 0.0
    running_metric = 0.0
    len_data = len(dataset_dl.dataset)

    for xb, yb in dataset_dl:
```

```
        xb = xb.to(device)
        yb = yb.to(device)
        output = model(xb)

        loss_b, metric_b = loss_batch(loss_func, output, yb, opt)

        running_loss += loss_b

        if metric_b is not None:
            running_metric += metric_b

        if sanity_check is True:
            break

    loss = running_loss / len_data
    metric = running_metric / len_data
    return loss, metric
```

```
# function to start training
def train_val(model, params):
    num_epochs=params['num_epochs']
    loss_func=params['loss_func']
    opt=params['optimizer']
    trainloader=params['trainloader']
    valloader=params['valloader']
    sanity_check=params['sanity_check']
    lr_scheduler=params['lr_scheduler']
    path2weights=params['path2weights']

    loss_history = {'train': [], 'val': []}
    metric_history = {'train': [], 'val': []}

    best_loss = float('inf')
    best_model_wts = copy.deepcopy(model.state_dict())
    start_time = time.time()

    for epoch in range(num_epochs):
        current_lr = get_lr(opt)
        print('Epoch {}/{}, current lr= {}'.format(epoch, num_epochs-1, current_lr))

        model.train()
        train_loss, train_metric = loss_epoch(model, loss_func, trainloader, sanity_check, opt)
        loss_history['train'].append(train_loss)
        metric_history['train'].append(train_metric)

        model.eval()
        with torch.no_grad():
            val_loss, val_metric = loss_epoch(model, loss_func, valloader, sanity_check)
        loss_history['val'].append(val_loss)
        metric_history['val'].append(val_metric)

        if val_loss < best_loss:
            best_loss = val_loss
            best_model_wts = copy.deepcopy(model.state_dict())
            torch.save(model.state_dict(), path2weights)
            print('Copied best model weights!')

        lr_scheduler.step(val_loss)
        if current_lr != get_lr(opt):
            print('Loading best model weights!')
            model.load_state_dict(best_model_wts)

        print('train loss: %.6f, val loss: %.6f, accuracy: %.2f, time: %.4f min' %(train_loss, val_loss, 100*val_metric, (time.time()-start_time)/60))
        print('-'*10)

    model.load_state_dict(best_model_wts)
    return model, loss_history, metric_history
```

학습에 필요한 파라미터를 설정합니다.

```
# define the training parameters
params_train = {
    'num_epochs':30,
    'optimizer':opt,
    'loss_func':loss_func,
    'trainloader':trainloader,
    'valloader':valloader,
    'sanity_check':False,
    'lr_scheduler':lr_scheduler,
    'path2weights':'./models/weights.pt',
}

# check the directory to save weights.pt
def createFolder(directory):
    try:
```

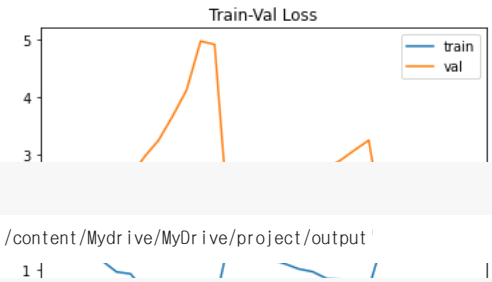
```
        if not os.path.exists(directory):
            os.makedirs(directory)
    except OSerror:
        print('Error')
createFolder('./models')
```

```
model, loss_hist, metric_hist = train_val(model, params_train)
```

```
Loading best model weights!
train loss: 0.376921, val loss: 4.909743, accuracy: 20.00, time: 9.0067 min
----------
Epoch 12/29, current lr= 0.001
train loss: 1.485146, val loss: 1.812876, accuracy: 20.00, time: 9.7635 min
----------
Epoch 13/29, current lr= 0.001
train loss: 1.308585, val loss: 1.972168, accuracy: 20.00, time: 10.5235 min
----------
Epoch 14/29, current lr= 0.001
train loss: 1.196778, val loss: 2.079031, accuracy: 20.00, time: 11.2750 min
----------
Epoch 15/29, current lr= 0.001
train loss: 1.177557, val loss: 2.173544, accuracy: 20.00, time: 12.0278 min
----------
Epoch 16/29, current lr= 0.001
train loss: 1.105074, val loss: 2.287086, accuracy: 20.00, time: 12.7750 min
----------
Epoch 17/29, current lr= 0.001
train loss: 1.014173, val loss: 2.440025, accuracy: 20.00, time: 13.5280 min
----------
Epoch 18/29, current lr= 0.001
train loss: 0.965161, val loss: 2.608122, accuracy: 20.00, time: 14.2840 min
----------
Epoch 19/29, current lr= 0.001
train loss: 0.846381, val loss: 2.762239, accuracy: 20.00, time: 15.1809 min
----------
Epoch 20/29, current lr= 0.001
train loss: 0.823012, val loss: 2.903336, accuracy: 20.00, time: 15.9546 min
----------
Epoch 21/29, current lr= 0.001
train loss: 0.754553, val loss: 3.078497, accuracy: 20.00, time: 16.7256 min
----------
Epoch 22/29, current lr= 0.001
Loading best model weights!
train loss: 0.676096, val loss: 3.247011, accuracy: 20.00, time: 17.4991 min
----------
Epoch 23/29, current lr= 0.0001
train loss: 1.496192, val loss: 1.821090, accuracy: 20.00, time: 18.2677 min
----------
Epoch 24/29, current lr= 0.0001
train loss: 1.490866, val loss: 2.007580, accuracy: 20.00, time: 19.0368 min
----------
Epoch 25/29, current lr= 0.0001
train loss: 1.425707, val loss: 2.150540, accuracy: 20.00, time: 19.8061 min
----------
Epoch 26/29, current lr= 0.0001
train loss: 1.368988, val loss: 2.255647, accuracy: 20.00, time: 20.5792 min
----------
Epoch 27/29, current lr= 0.0001
train loss: 1.349003, val loss: 2.333552, accuracy: 20.00, time: 21.3490 min
----------
Epoch 28/29, current lr= 0.0001
train loss: 1.305644, val loss: 2.393055, accuracy: 20.00, time: 22.1274 min
----------
Epoch 29/29, current lr= 0.0001
train loss: 1.274053, val loss: 2.438549, accuracy: 20.00, time: 22.9026 min
----------
```

loss, accuracy progress를 확인하겠습니다

```
# train-val progress
num_epochs = params_train['num_epochs']

# plot loss progress
plt.title('Train-Val Loss')
plt.plot(range(1, num_epochs+1), loss_hist['train'], label='train')
plt.plot(range(1, num_epochs+1), loss_hist['val'], label='val')
plt.ylabel('Loss')
plt.xlabel('Training Epochs')
plt.legend()
plt.show()
```



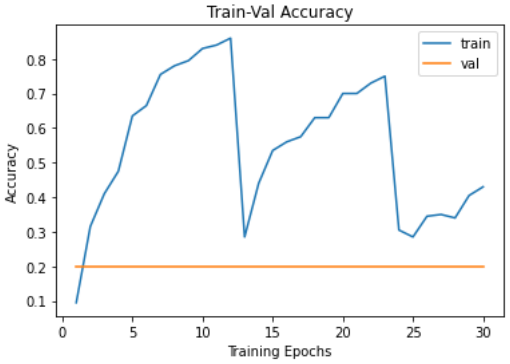

```
pwd
```

```
'/content/Mydrive/MyDrive/project/output'
```

```
plt.savefig("/content/Mydrive/MyDrive/project/output/result/mobilenet2_loss_121802.png")
```

```
<Figure size 432x288 with 0 Axes>
```

```
# plot accuracy progress
plt.title('Train-Val Accuracy')
plt.plot(range(1, num_epochs+1), metric_hist['train'], label='train')
plt.plot(range(1, num_epochs+1), metric_hist['val'], label='val')
plt.ylabel('Accuracy')
plt.xlabel('Training Epochs')
plt.legend()
plt.show()
```




```
plt.savefig("/content/Mydrive/MyDrive/project/output/result/mobilenet2_accuracy_121802.png")
```

```
<Figure size 432x288 with 0 Axes>
```

파이토치로 MobileNet2 생성

```
import torch.nn as nn
import math
import torch
from torchvision import datasets, transforms
# from torchvision.models import mobilenet_v2
from torch import nn, optim
from torch.utils.data import DataLoader, TensorDataset
from torchvision.utils import make_grid

from tqdm.auto import tqdm

import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cpu')
```

```
# 첫번째 layer에서 사용될 convolution 함수
def conv_bn(inp, oup, stride):
    return nn.Sequential(
        nn.Conv2d(inp, oup, 3, stride, 1, bias=False),
        nn.BatchNorm2d(oup),
        nn.ReLU6(inplace=True)
    )
```

```
# inverted bottleneck layer 바로 다음에 나오는 convolution에 사용될 함수
def conv_1x1_bn(inp, oup):
    return nn.Sequential(
        nn.Conv2d(inp, oup, 1, 1, 0, bias=False),
```

```
        nn.BatchNorm2d(oup),
        nn.ReLU6(inplace=True))
```

```
# channel수를 무조건 8로 나누어 떨어지게 만드는 함수
def make_divisible(x, divisible_by=8):
    import numpy as np
    return int(np.ceil(x * 1. / divisible_by) * divisible_by)
```

```
class InvertedResidual(nn.Module):

    def __init__(self, inp, oup, stride, expand_ratio):
        super(InvertedResidual,self).__init__()
        self.stride = stride
        assert stride in [1,2]

        hidden_dim = int(inp * expand_ratio)  # expansion channel
        self.use_res_connect = self.stride == 1 and inp == oup  # skip connection이 가능한지 확인 True or False
        '''
        self.stride == 1 ----> 연산 전 후의 feature_map size가 같다는 의미
        inp == oup ----> 채널수도 동일하게 유지된다는 의미
        즉 skip connection 가능
        '''
        if expand_ratio == 1:
            self.conv = nn.Sequential(
                # 확장시킬 필요가 없기 때문에 바로 depth wise conv
                nn.Conv2d(hidden_dim, hidden_dim, 3, stride, 1, groups=hidden_dim, bias=False),
                nn.BatchNorm2d(hidden_dim),
                nn.ReLU6(inplace=True),
                # pw-linear
                nn.Conv2d(hidden_dim, oup, 1, 1, 0, bias=False),
                nn.BatchNorm2d(oup)
            )
        else:
            self.conv = nn.Sequential(
                # pw(확장)
                nn.Conv2d(inp, hidden_dim, 1, 1, 0, bias=False),
                nn.BatchNorm2d(hidden_dim),
                nn.ReLU6(inplace=True),
                # dw
                nn.Conv2d(hidden_dim, hidden_dim, 3, stride, 1, groups=hidden_dim, bias=False),
                nn.BatchNorm2d(hidden_dim),
                nn.ReLU6(inplace=True),
                # pw-linear(축소)
                nn.Conv2d(hidden_dim, oup, 1, 1, 0, bias=False),
                nn.BatchNorm2d(oup)
            )

    def forward(self, x):
        if self.use_res_connect:
            return x + self.conv(x)  # skip connection (element wise sum)
        else:
            return self.conv(x)
```

```
class MobileNetV2(nn.Module):

    def __init__(self, n_class=1000, input_size=224, width_mult=1.):
        super(MobileNetV2, self).__init__()
        block = InvertedResidual
        input_channel = 32
        last_channel = 1280

        interverted_residual_setting = [
            # t, c, n, s
            # t : expand ratio
            # c : channel
            # n : Number of iterations
            # s : stride
            [1, 16, 1, 1],
            [6, 24, 2, 2],
            [6, 32, 3, 2],
            [6, 64, 4, 2],
            [6, 96, 3, 1],
            [6, 160, 3, 2],
            [6, 320, 1, 1]
        ]

        # building first layer
        assert input_size % 32 == 0
        # input_channel = make_divisible(input_channel * width_mult)
        self.last_channel = make_divisible(last_channel * width_mult) if width_mult > 1.0 else last_channel
        self.features = [conv_bn(3, input_channel, 2)]  # feature들을 담을 리스트에 first layer 추가
```

```
        # building inverted residual blocks
        for t, c, n, s in interverted_residual_setting:
            output_channel = make_divisible(c * width_mult) if t > 1 else c
            for i in range(n):
                if i == 0:
                    self.features.append(block(input_channel, output_channel, s, t))
                else:
                    self.features.append(block(input_channel, output_channel, 1, t))  # 반복되는 부분에서 skip connection 가능
                input_channel = output_channel
        # building last several layers
        self.features.append(conv_1x1_bn(input_channel, self.last_channel))  # (batch, 320, 7, 7) -> (batch, 1280, 7, 7)
        # make it nn.Sequential
        self.features = nn.Sequential(*self.features)

        # Average pooling layer
        self.avg = nn.AvgPool2d(7, 7)
        # building classifier
        self.classifier = nn.Linear(self.last_channel, n_class)

        self._initialize_weights()

    def forward(self, x):
        # pdb.set_trace()
        x = self.features(x)
        x = self.avg(x)
        x = x.view(-1, self.last_channel)
        x = self.classifier(x)
        return x

    # 초기 weight 설정
    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
                if m.bias is not None:
                    m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
            elif isinstance(m, nn.Linear):
                n = m.weight.size(1)
                m.weight.data.normal_(0, 0.01)
                m.bias.data.zero_()


def mobilenet_v2(pretrained=True):
    model = MobileNetV2(width_mult=1)

    if pretrained:
        try:
            from torch.hub import load_state_dict_from_url
        except ImportError:
            from torch.utils.model_zoo import load_url as load_state_dict_from_url
        state_dict = load_state_dict_from_url(
            'https://www.dropbox.com/s/47tyzpofuuyyv1b/mobilenetv2_1.0-f2a8633.pth.tar?dl=1', progress=True)

        model.load_state_dict(state_dict)
    return model
```

위 코드에서 아직 class수가 1000개로 되어있는 이유는 ImageNet 데이터에 대한 pretrained weights를 가져다 쓸 것이기 때문에 일단 모델 구조를 수정하지 않았다.

DataLoad and Hyperparameter adjustment

```
import os
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
import torchvision.datasets as datasets

device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
# specify path to data
path2data = '/content/Mydrive/MyDrive/project/data'

#load dataset
train_path = '/content/Mydrive/MyDrive/project/output/train/'
validation_path = '/content/Mydrive/MyDrive/project/output/val/'
test_path = '/content/Mydrive/MyDrive/project/output/test/'
```

```
train_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((.5, .5, .5), (.5, .5, .5))
])
```

```
test_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize((.5, .5, .5), (.5, .5, .5))
])
```

```
train_dataset = datasets.ImageFolder(train_path, transform=train_transform)
test_dataset = datasets.ImageFolder(test_path, transform=test_transform)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False, num_workers=2, pin_memory=True)
```

/usr/local/lib/python3.8/dist-packages/torch/utils/data/dataloader.py:554: UserWarning: 이 DataLoader는 총 6개의 작업자 프로세스를 생성합니다. 현재 시스템에서 권장하는 최대 작업자 수는 2이며, 이는 이 DataLoader가 생성하려는 것보다 작습니다. 작업자를 과도하게 생성하면 DataLoader가 느리게 실행되거나 정지될 수 있으므로 필요한 경우 잠재적인 느려짐/정지를 방지하기 위해 작업자 수를 줄이십시오.

```
classes = os.listdir(train_path)

model = mobilenet_v2(True)
model.classifier = nn.Linear(model.classifier.in_features, len(classes)).to(device)
model = model.to(device)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)
best_acc = 0
epochs = 10
```

```
def train(epoch):
    model.train()

    train_loss = 0
    correct = 0
    total = 0

    for index, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, targets)

        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += (predicted == targets).sum().item()
        if (index+1) % 20 == 0:
            print(f'[Train] | epoch: {epoch+1}/{epochs} | batch: {index+1}/{len(train_loader)}| loss: {loss.item():.4f} | Acc: {correct / total * 100:.4f}')
```

```
def test(epoch):
    global best_acc
    model.eval()
    test_loss = 0
    correct = 0
    total = 0

    with torch.no_grad():
        for index, (inputs, targets) in enumerate(test_loader):

            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
```

```
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += (predicted == targets).sum().item()

        print(f'[Test] epoch: {epoch+1} loss: {test_loss:.4f} | Acc: {correct / total * 100:.4f}')



    # Save checkpoint.
    acc = 100.*correct/total
    if acc > best_acc:
        print('Saving..')
        state = {
            'model': model.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state, './checkpoint/ckpt02.pth')
        best_acc = acc
```

```
for epoch in range(epochs):
    train(epoch)
```

```
for epoch in range(epochs):
    test(epoch)
```

```
[Test] epoch: 1 loss: 1.1959 | Acc: 64.0000
Saving..
[Test] epoch: 2 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 3 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 4 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 5 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 6 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 7 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 8 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 9 loss: 1.1959 | Acc: 64.0000
[Test] epoch: 10 loss: 1.1959 | Acc: 64.0000
```

Inference (Visualization)

```
import matplotlib.pyplot as plt
import numpy as np
import torchvision.transforms as transforms
from torch.utils.data import Dataset
from PIL import Image
import time
import torchvision
```

```
class Archive(Dataset):
    def __init__(self, path, transform=None):
        img_name = [f for f in os.listdir(path)]
        self.imgList = [os.path.join(path, i) for i in img_name]
        self.path = path
        self.transform = transform

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

    def __getitem__(self, idx):
        image = Image.open(self.imgList[idx]).convert('RGB')

        if self.transform is not None:
            image = self.transform(image)
        return image
```

```
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.5, 0.5, 0.5])
    std = np.array([0.5, 0.5, 0.5])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)

    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated
```

```
def visualize_model(model, num_images=12):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, inputs in enumerate(pred_loader):
            inputs = inputs.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                plt.figure(figsize=(20,20))
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('predicted: {}'.format(classes[preds[j]]))
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

```
train_path = '/content/Mydrive/MyDrive/project/output/train/'
validation_path = '/content/Mydrive/MyDrive/project/output/val/'
test_path = '/content/Mydrive/MyDrive/project/output/test/'
```

```
if __name__ == '__main__':

    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    train_path = '/content/Mydrive/MyDrive/project/output/train/'
    pred_path = '/content/Mydrive/MyDrive/project/data/test/'
    classes = sorted(os.listdir(train_path))

    pred_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize((.5, .5, .5), (.5, .5, .5))
    ])

    pred_dataset = Archive(pred_path, transform=pred_transform)
    pred_loader = DataLoader(pred_dataset, batch_size=64, shuffle=True, num_workers=2, pin_memory=True)

    model = mobilenet_v2(False)
    model.classifier = nn.Linear(model.classifier.in_features, len(classes)).to(device)  # 1000 -> 6

    checkpoint = torch.load('/content/checkpoint/ckpt02.pth')
    model.load_state_dict(checkpoint['model'])

    model = model.to(device)
    visualize_model(model)
```

```
pwd
```

```
'/content'
```

```
label2cat = train_dataset.classes
label2cat, len(label2cat)
```

```
(['Arctic fox', 'Judas tree', 'barn', 'baseball', 'seashore'], 5)
```

```
feature, target = next(iter(trainloader))
feature.shape
```

```
torch.Size([128, 3, 224, 224])
```

케라스를 통해 ImagenetV2 모델을 Convert하기

```
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from keras.models import Sequential

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import matplotlib.pyplot as plt
import numpy as np
import random
import cv2
import os
```

```
imagePaths = sorted(list(paths.list_images("/content/Mydrive/MyDrive/project/class5")))
```

```
cd /content/Mydrive/MyDrive/project/class5/barn
```

```
/content/Mydrive/MyDrive/project/class5/barn
```

```
ls -l | grep ^- | wc -l
```

```
50
```

```
# random shuffle
random.seed(42)
random.shuffle(imagePaths)

data = []
labels = []
image_dims = (224, 224, 3)
```

```
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (image_dims[1], image_dims[0]))
    image = img_to_array(image)
    data.append(image)
    l = label = imagePath.split(os.path.sep)[-2].split("_")
    labels.append(l)
```

```
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
print("{} images ({:.2f}MB)".format(len(imagePaths), data.nbytes / (1024 * 1000.0)))
```

```
250 images (294.00MB)
```

```
data = np.array(data)
label = np.array(labels)
print(data.shape)
```

```
(250, 224, 224, 3)
```

Custom MobileNetV2 만들기

```
class CustomMobileNetv2(nn.Module):
  def __init__(self, output_size):
    super().__init__()
    self.mnet = mobilenet_v2(pretrained=True)
    self.freeze()

    self.mnet.classifier = nn.Sequential(
        nn.Linear(1280, output_size),
        nn.LogSoftmax(1)
```

```
    )

  def forward(self, x):
    return self.mnet(x)

  def freeze(self):
    for param in self.mnet.parameters():
      param.requires_grad = False

  def unfreeze(self):
    for param in self.mnet.parameters():
      param.requires_grad = True
```

Adaptation

```
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from keras.models import Sequential
from tensorflow.keras.applications import MobileNetV2
```

```
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import matplotlib.pyplot as plt
import numpy as np
import random
import cv2
import os
```

```
imagePaths = sorted(list(paths.list_images("/content/Mydrive/MyDrive/project/class5")))
# random shuffle
random.seed(7)
random.shuffle(imagePaths)

data = []
labels = []
image_dims = (224, 224, 3)
```

```
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (image_dims[1], image_dims[0]))
    image = img_to_array(image)
    data.append(image)
    l = label = imagePath.split(os.path.sep)[-2].split("_")
    labels.append(l)


data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
print("{} images ({:.2f}MB)".format(len(imagePaths), data.nbytes / (1024 * 1000.0)))


data = np.array(data)
label = np.array(labels)
print(data.shape)

x_train, x_valid, y_train, y_valid = train_test_split(data, labels, test_size=0.20)
```

```
250 images (294.00MB)
(250, 224, 224, 3)
```

```
 mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)
# total 5 labels
print("class labels:")
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i + 1, label))
```

```
class labels:
1. Arctic fox
2. Judas tree
3. barn
4. baseball
5. seashore
```

```
bs = 32
lr = 0.0001
size = (224, 224)
shape = (224,224, 3)
epochs = 10
class_number = 5
```

```
from keras.callbacks import ReduceLROnPlateau

# 콜백 정의
reduceLR = ReduceLROnPlateau(
    monitor='val_loss',  # 검증 손실을 기준으로 callback이 호출됩니다
    factor=0.5,          # callback 호출시 학습률을 1/2로 줄입니다
    patience=10,         # epoch 10 동안 개선되지 않으면 callback이 호출됩니다
)
```

model.fit 이 왜 안되는지 원인 찾아야함

```
print("[INFO] training ...")
H = model.fit(x_train, y_train, batch_size=32,
              steps_per_epoch=len(x_train),
              validation_data=(x_valid, y_valid),
              validation_steps=len(x_valid),    epochs=10)
```

```
from keras.callbacks import ReduceLROnPlateau as callbacks

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=5, min_lr=0.001)
model.fit(x_train, y_train, callbacks=[reduce_lr])
```

Arsitektur & Config

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=5, min_lr=0.001)
model.fit(x_train, y_train, callbacks=[reduce_lr])
```

```
optimizer = optim.AdamW(model.parameters(), lr=0.001)
criterion = nn.NLLLoss()
```

```
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1,
patience=10, threshold=0.0001, threshold_mode='rel',
cooldown=0, min_lr=0, eps=1e-08, verbose=False)
```

```
<torch.optim.lr_scheduler.ReduceLROnPlateau at 0x7fb903fc8b80>
```

Adaptation¶

```
def loop_fn(mode, dataset, dataloader, model, criterion, optimizer, device):
  if mode == 'train':
    model.train()
  elif mode == 'val':
    model.eval()

  cost = correct = 0
  for feature, target in tqdm(dataloader, desc=mode.title()):
    feature, target = feature.to(device), target.to(device)
    output = model(feature)
    loss = criterion(output, target)

    if mode == 'train':
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()

    cost += loss.item() * feature.shape[0]
    correct += (output.argmax(1) == target).sum().item()
  cost = cost/len(dataset)
  acc = correct/len(dataset)
  return cost, acc
```

```
while True:
  train_cost, train_score = loop_fn('train', train_set, trainloader, model, criterion, optimizer, device)
  with torch.no_grad():
    test_cost, test_score = loop_fn('val', val_set, valloader, model, criterion, optimizer, device)
```

```
Train: 0%                                                    0/2 [00:00<?, ?it/s]
```

```
  # Logging
  callbacks.log(train_cost, test_cost, train_score, test_score)

  # Checkpoint
  callbacks.save_checkpoint()

  # Runtime Plotting
  callbacks.cost_runtime_plotting()
  callbacks.score_runtime_plotting()

  # Early Stopping
  if callbacks.early_stopping(model, monitor='test_score'):
    callbacks.plot_cost()
    callbacks.plot_score()
    break
```

Fine-Tuning

```
model.unfreeze()
optimizer = optim.AdamW(model.parameters(), lr=1e-5)

callbacks.reset_early_stop()
callbacks.early_stop_patience = 3
```

```
while True:
  train_cost, train_score = loop_fn('train', train_set, trainloader, model, criterion, optimizer, device)
  with torch.no_grad():
    test_cost, test_score = loop_fn('val', val_set, valloader, model, criterion, optimizer, device)

  # Logging
  callbacks.log(train_cost, test_cost, train_score, test_score)

  # Checkpoint
  callbacks.save_checkpoint()

  # Runtime Plotting
  callbacks.cost_runtime_plotting()
  callbacks.score_runtime_plotting()

  # Early Stopping
  if callbacks.early_stopping(model, monitor='test_score'):
    callbacks.plot_cost()
    callbacks.plot_score()
    break
```

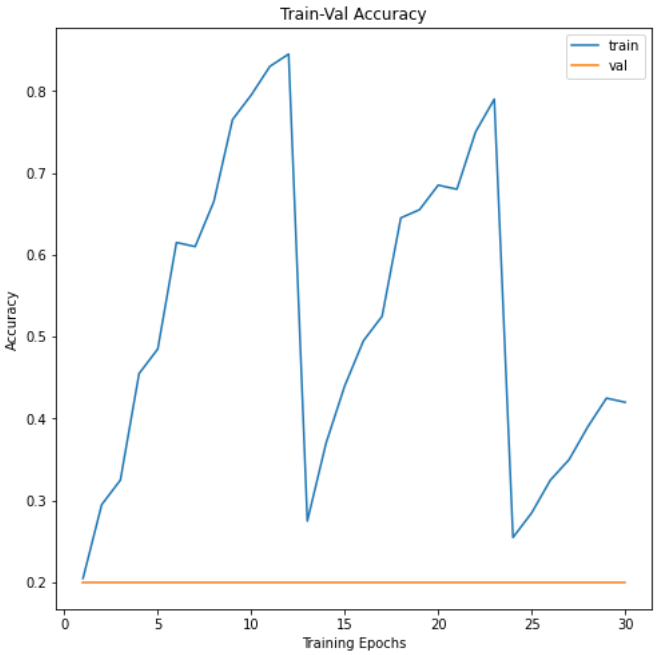
Predict

```
for feature, target in testloader:
  feature, target = feature.to(device), target.to(device)
  with torch.no_grad():
    model.eval()
    output = model(feature)
    preds = output.argmax(1)

fig, axes = plt.subplots(6, 6, figsize=(24, 24))
for img, label, pred, ax in zip(feature, target, preds, axes.flatten()):
  ax.imshow(img.permute(1,2,0).cpu())
  font = {"color":'r'} if label != pred else {"color": 'g'}
  label, pred = label2cat[label.item()], label2cat[pred.item()]
  ax.set_title(f"Label: {label}₩nPred: {pred}", fontdict=font);
  ax.axis("off");
```

Test Score¶

```
accuracy = []
for feature, target in testloader:
  feature, target = feature.to(device), target.to(device)
  with torch.no_grad():
    model.eval()
    output = model(feature)
    preds = output.argmax(1)
    if(target == preds):
      acc = 1
```

```
    else: acc = 0
    accuracy.append(acc)
accuracy = np.array(accuracy)
accuracy.mean()
```




```
plt.savefig("/content/Mydrive/MyDrive/project/output/mobilenet2_accuracy_1218.png")
```

```
<Figure size 576x576 with 0 Axes>
```

```
print('/content/Mydrive/MyDrive/project/output/model/weights.pt')
```

```
/content/Mydrive/MyDrive/project/output/model/weights.pt
```

```
pwd
```

```
'/content/Mydrive/MyDrive/project/output'
```

Convert labels as sparse matrix¶

```
mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)
# total 5 labels
print("class labels:")
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i + 1, label))
```

```
class labels:
1. Arctic fox
2. Judas tree
3. barn
4. baseball
5. seashore
```

Define model and layes by MobileNetV2¶

```
def MobileNetV2_model(learning_rate, input_shape,class_number):
    baseModel = MobileNetV2(include_top=False, input_tensor=Input(shape=input_shape))
    for layer in baseModel.layers[:-4]:
        layer.trainable = False

    model = Sequential()
    model.add(baseModel)
    model.add(AveragePooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(512, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(50, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(class_number, activation='softmax'))

    return model
```

Set parameters for training

```
bs = 32
lr = 0.0001
size = (224, 224)
shape = (224,224, 3)
epochs = 10
class_number = 5
```

Compile model¶

```
model = MobileNetV2_model(lr,shape,class_number)
model.compile(loss= "categorical_crossentropy", metrics=["accuracy"], optimizer="adam")
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as
```

더블클릭 또는 Enter 키를 눌러 수정

```
trainX, testX, trainY, testY = train_test_split(data, labels, test_size=0.20)
```

```
len(trainX)
```

```
200
```

```
len(testX)
```

```
50
```

Train the network¶

```
print("[INFO] training ...")
H = model.fit(trainX, trainY, batch_size=32,
              steps_per_epoch=len(trainX),
              validation_data=(testX, testY),
              validation_steps=len(testX),  epochs=20)
```

```
[INFO] training ...
Epoch 1/20
140/200 [====================>.........] - ETA: 1:07 - loss: 0.0522 - accuracy: 0.9840WARNING:tensorflow:Your input ran out of data; interrupting training. Make
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` b
200/200 [==============================] - 159s 792ms/step - loss: 0.0522 - accuracy: 0.9840 - val_loss: 7.6625e-04 - val_accuracy: 1.0000
```

140/200 [==========>.....] - ETA: 1:07 - 손실: 0.0522 - 정확도: 0.9840경고 :tensorflow:입력한 데이터가 부족합니다. 훈련 방해. 데이터세트 또는 생성기가 적어도 `steps_per_epoch * epochs` 배치(이 경우 4000 배치)를 생성할 수 있는지 확인하십시오. 데이터 세트를 만들 때 repeat() 함수를 사용해야 할 수도 있습니다. 경고: tensorflow: 입력 데이터가 부족합니다. 훈련 방해. 데이터세트 또는 생성기가 적어도 `steps_per_epoch * epochs` 배치(이 경우 배치 50개)를 생성할 수 있는지 확인하십시오. 데이터 세트를 만들 때 repeat() 함수를 사용해야 할 수도 있습니다.

Make predictions on the testing set

```
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=32)

# for each image in the testing set we need to find the index of the label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,target_names=mlb.classes_))
```

```
[INFO] evaluating network...
2/2 [==============================] - 3s 681ms/step
              precision    recall  f1-score   support

   Arctic fox       1.00      1.00      1.00         9
   Judas tree       1.00      1.00      1.00         6
         barn       1.00      1.00      1.00        12
     baseball       1.00      1.00      1.00        13
     seashore       1.00      1.00      1.00        10

     accuracy                           1.00        50
    macro avg       1.00      1.00      1.00        50
 weighted avg       1.00      1.00      1.00        50
```

Serialize the model to disk¶

```
#print("[INFO] saving orange detector model...")
#model.save("Orange_MobileNetV2.model", save_format="h5")
#model.save("Orange_MobileNetV2tf.model", save_format="tf")
```

Plot the training loss and accuracy¶

```
N = 20
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
```

```
#plt.savefig("mobilenet.png")
```

더블클릭 또는 Enter 키를 눌러 수정

적대적 샘플을 통해 accuracy 높이기

```
import tensorflow as tf
import matplotlib.pyplot as plt

pretrained_model = tf.keras.applications.MobileNetV2(weights='imagenet')
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224.h5
14536120/14536120 [==============================] - 1s 0us/step
```

```
import tensorflow as tf
```

```
def preprocess(image):
  image = tf.cast(image, tf.float32)
  image = image/255
  image = tf.image.resize(image, (224, 224))
  image = image[None, ...]
  return image
```

```
# 확률 벡터에서 레이블을 추출해주는 헬퍼 메서드
def get_imagenet_label(probs):
  return decode_predictions(probs, top=1)[0][0]
```

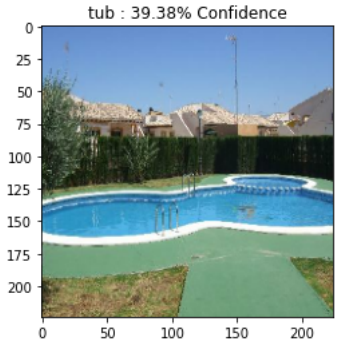
```
sample_image = '/content/Mydrive/MyDrive/project/data/test/ILSVRC2010_val_00048053.jpg'
```

```
image_raw = tf.io.read_file(sample_image)
image = tf.image.decode_image(image_raw)

image = preprocess(image)
image_probs = pretrained_model.predict(image)
```

```
1/1 [==============================] - 1s 1s/step
```

```
plt.figure()
plt.imshow(image[0])
_, image_class, class_confidence = get_imagenet_label(image_probs)
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 [==============================] - 0s 0us/step
```



적대적 샘플이란? 적대적 샘플이란 신경망을 혼란시킬 목적으로 만들어진 특수한 입력으로, 신경망으로 하여금 샘플을 잘못 분류하도록 합니다. 비록 인간에게 적대적 샘플은 일반 샘플과 큰 차이가 없어보이지만, 신경망은 적대적 샘플을 올바르게 식별하지 못합니다. 이와 같은 신경망 공격에는 여러 종류가 있는데, 본 튜토리얼에서는 화이트 박스(white box) 공격 기술에 속하는 FGSM을 소개합니다. 화이트 박스 공격이란 공격자가 대상 모델의 모든 파라미터값에 접근할 수 있다는 가정 하에 이루어지는 공격을 일컫습니다

FGSM FGSM은 신경망의 그래디언트(gradient)를 이용해 적대적 샘플을 생성하는 기법입니다. 만약 모델의 입력이 이미지라면, 입력 이미지에 대한 손실 함수의 그래디언트를 계산하여 그 손실을 최대화하는 이미지를 생성합니다. 이처럼 새롭게 생성된 이미지를 적대적 이미지(adversarial image)라고 합니다. 이 과정은 다음과 같은 수식으로 정리할 수 있습니다:

adv_x : 적대적 이미지. x : 원본 입력 이미지. y : 원본 입력 레이블(label). : 왜곡의 양을 적게 만들기 위해 곱하는 수. : 모델의 파라미터. : 손실 함수. 각 기호에 대한 설명은 다음과 같습니다.

여기서 흥미로운 사실은 입력 이미지에 대한 그래디언트가 사용된다는 점입니다. 이는 손실을 최대화하는 이미지를 생성하는 것이 FGSM의 목적이기 때문입니다. 요약하자면, 적대적 샘플은 각 픽셀의 손실에 대한 기여도를 그래디언트를 통해 계산한 후, 그 기여도에 따라 픽셀값에 왜곡을 추가함으로써 생성할 수 있습니다. 각 픽셀의 기여도는 연쇄 법칙(chain rule)을 이용해 그래디언트를 계산하는 것으로 빠르게 파악할 수 있습니다. 이것이 입력 이미지에 대한 그래디언트가 쓰이는 이유입니다. 또한, 대상 모델은 더 이상 학습하고 있지 않기 때문에 (따라서 신경망의 가중치에 대한 그래디언트는 필요하지 않습니다) 모델의 가중치값은 변하지 않습니다. FGSM의 궁극적인 목표는 이미 학습을 마친 상태의 모델을 혼란시키는 것입니다.

```
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams['figure.figsize'] = (8, 8)
mpl.rcParams['axes.grid'] = False
```

```
pretrained_model = tf.keras.applications.MobileNetV2(include_top=True,
                                                      weights='imagenet')
pretrained_model.trainable = False

# ImageNet 클래스 레이블
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions
```

```
plt.figure()
plt.imshow(image[0])
_, image_class, class_confidence = get_imagenet_label(image_probs)
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```

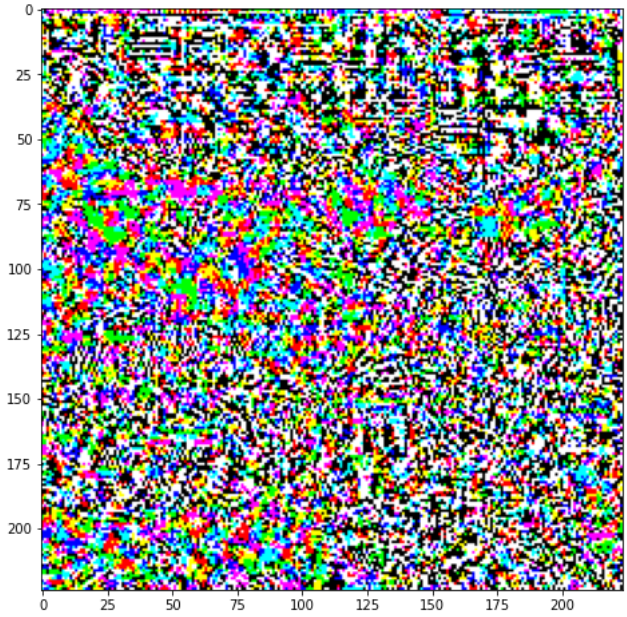
적대적 이미지 생성하기 FGSM 실행하기 첫번째 단계는 샘플 생성을 위해 원본 이미지에 가하게 될 왜곡을 생성하는 것입니다. 앞서 살펴보았듯이, 왜곡을 생성할 때에는 입력 이미지에 대한 그래디언트를 사용합니다.

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

def create_adversarial_pattern(input_image, input_label):
  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = pretrained_model(input_image)
    loss = loss_object(input_label, prediction)

  # 입력 이미지에 대한 손실 함수의 기울기를 구합니다.
  gradient = tape.gradient(loss, input_image)
  # 왜곡을 생성하기 위해 그래디언트의 부호를 구합니다.
  signed_grad = tf.sign(gradient)
  return signed_grad
```

생성한 왜곡을 시각화해 볼 수 있습니다.

```
# 이미지의 레이블을 원-핫 인코딩 처리합니다.
labrador_retriever_index = 208
label = tf.one_hot(labrador_retriever_index, image_probs.shape[-1])
label = tf.reshape(label, (1, image_probs.shape[-1]))

perturbations = create_adversarial_pattern(image, label)
plt.imshow(perturbations[0])
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
<matplotlib.image.AxesImage at 0x7f33120c83d0>
```



왜곡 승수 엡실론(epsilon)을 바꿔가며 다양한 값들을 시도해봅시다. 위의 간단한 실험을 통해 엡실론의 값이 커질수록 네트워크를 혼란시키는 것이 쉬워짐을 알 수 있습니다. 하지만 이는 이미지의 왜곡이 점점 더 뚜렷해진다는 단점을 동반합니다.

```
def display_images(image, description):
  _, label, confidence = get_imagenet_label(pretrained_model.predict(image))
  plt.figure()
  plt.imshow(image[0])
  plt.title('{} Wn {} : {:.2f}% Confidence'.format(description,
                                                   label, confidence*100))
  plt.show()
```

```
epsilons = [0, 0.01, 0.1, 0.15]
descriptions = [('Epsilon = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]

for i, eps in enumerate(epsilons):
  adv_x = image + eps*perturbations
  adv_x = tf.clip_by_value(adv_x, 0, 1)
  display_images(adv_x, descriptions[i])
```

```
len(train_set), len(test_set), len(val_set)
```

```
(200, 25, 25)
```

```
label2cat = train_set.classes
label2cat, len(label2cat)
```

```
(['Arctic fox', 'Judas tree', 'barn', 'baseball', 'seashore'], 5)
```

```
feature, target = next(iter(trainloader))
feature.shape
```

```
torch.Size([128, 3, 224, 224])
```

Arsitektur & Config

```
class CustomMobileNetv2(nn.Module):
  def __init__(self, output_size):
    super().__init__()
    self.mnet = mobilenet_v2(pretrained=True)
    self.freeze()

    self.mnet.classifier = nn.Sequential(
        nn.Linear(1280, output_size),
        nn.LogSoftmax(1)
    )

  def forward(self, x):
    return self.mnet(x)

  def freeze(self):
    for param in self.mnet.parameters():
      param.requires_grad = False

  def unfreeze(self):
    for param in self.mnet.parameters():
      param.requires_grad = True
```

이미지 train, test , val 폴더 비율 split-folders로 나누기

[ ] ↳ *숨겨진 셀 4개*