DNN_02_3. CNN_EfficientNet

생성일	@2022년 6월 14일 오후 4:24
유형	머신러닝/딥러닝
작성자	동훈 오

EfficientNet(2019) 논문

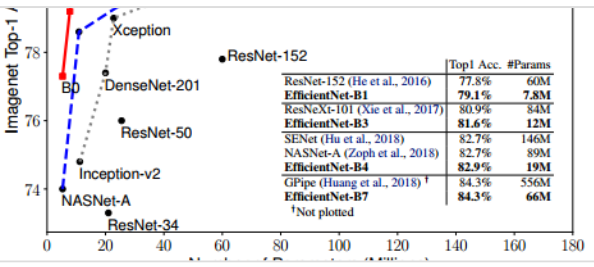
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/cf0f5c80-fcd6-48bd-aa97-9c73e6773cc3/EfficientNet.pdf

댓글에 올린 논문 한국어 리뷰

[논문 읽기] EfficientNet(2019) 리뷰, Rethinking Model Scaling for Convolutional Neural Networks

안녕하세요! 이번에 읽어볼 논문은 EfficientNet, Rethinking Model Scaling for Convolutional Neural Networks 입니다. 모델의 정확도를 높일 때, 일반적으로 (1) 모델의 깊이, (2) 너비, (3) 입력 이미지의 크기를 조절합니다. 기존에는 이 세 가지를 수동으로 조절하였기 때문에, 최적의 성능과 효율을 얻지 못했습니다. EfficientNet은 3가지를 효율적으로 조절

https://deep-learning-study.tistory.com/552



EfficientNet 은 MnasNet 구조와 유사하다.

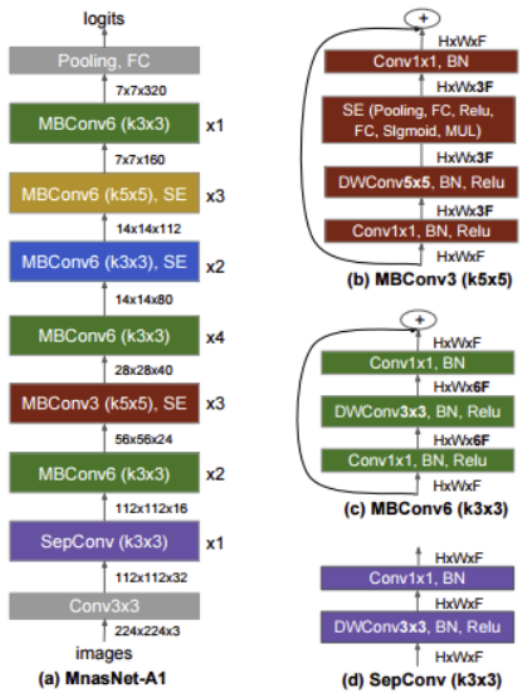


Figure 7: **MnasNet-A1 Architecture –** (a) is a representative model selected from Table 1; (b) - (d) are a few corresponding layer structures. *MBConv* denotes mobile inverted bottleneck conv, *DWConv* denotes depthwise conv, k3x3/k5x5 denotes kernel size, *BN* is batch norm, HxWxF denotes tensor shape (height, width, depth), and $\times 1/2/3/4$ denotes the number of repeated layers within the block.

Table 1. **EfficientNet-B0 baseline network –** Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Tensorflow에서 구현한 EfficientNet 설명 Document

tf.keras.applications.efficientnet.EfficientNetB0 | TensorFlow Core v2.9.1

Instantiates the EfficientNetB0 architecture.

https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0

텐서플로우에서 구현한 EfficientNet 은 구조에 따라서 하위 종류가 여러 가지로 나뉜다. 그중 가장 기초적인 EfficientNetB0 를 살펴보자.

```
def EfficientNetB0(include_top=True,
                   weights='imagenet',
                   input_tensor=None,
                   input_shape=None,
                   pooling=None,
                   classes=1000,
                   classifier_activation='softmax',
                   **kwargs):
  return EfficientNet(
      1.0,
      1.0,
      224,
      0.2,
      model_name='efficientnetb0',
      include_top=include_top,
      weights=weights,
      input_tensor=input_tensor,
      input_shape=input_shape,
      pooling=pooling,
      classes=classes,
      classifier_activation=classifier_activation,
      **kwargs)
```

매개변수에 대한 설명을 참조하면 다음과 같다.

- include_top : fully-connected layer를 네트워크 최상단에 포함시킬 것인지에 대한 여부. 디폴트는 True 이다.
- weights : 디폴트로 imagenet 을 설정
- input_tensor : input 으로 image 를 사용하기 위한 optional 한 keras tensor
- input_shape : optional shape tuple, 최상단에 fully-connected-layer 가 없을 경우 사용하기 위함.
- pooling : optional pooling mode, avg 또는 max 지정 가능.
- classes : 이미지 분류를 위한 클래스 개수 지정

- classifier_activation : 클래스 분류를 위한 활성화함수. = softmax

사실 핵심은 return 부분에 있는 설정값이다. EfficientNet 에서 B0 모델이 분류된건 아래에서 제시된 계수 때문이라고 생각할 수 있다.

```
return EfficientNet(
          1.0,
          1.0,
          224,
          0.2,
          ...
)
```

EfficientNet 함수를 찾아보면 다음과 같다.

```
def EfficientNet(
    width_coefficient,
    depth_coefficient,
    default_size,
    dropout_rate=0.2,
    drop_connect_rate=0.2,
    depth_divisor=8,
    activation='swish',
    blocks_args='default',
    model_name='efficientnet',
    include_top=True,
    weights='imagenet',
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation='softmax'):
```

인자로 갖는 처음 세 개의 변수를 각각 width 계수, depth 계수, default_size 값, dropout 비율을 의미한다.

EfficientNet을 이전에 만들어둔 CNN 모델처럼 사용하려면 configuration 부분과 모델 부분만 바꿔주면 된다. 아래의 코드를 붙여 넣으면 완성된다.

```
_efficient_finetune_cfg: dict = {
    "efficient_net_model_name" : "EfficientNetB0",
    "classes" : 10,
    "efficient_net_weight_trainable" : False,
    "kwargs" : {
        "include_top" : False,
        "weights" : "imagenet",
    }
}

_efficient_finetune_cfg = OmegaConf.create(_efficient_finetune_cfg)

# generate model
class EfficientNetFinetune(tf.keras.Model):
def __init__(self, cfg: DictConfig = _efficient_finetune_cfg):
```