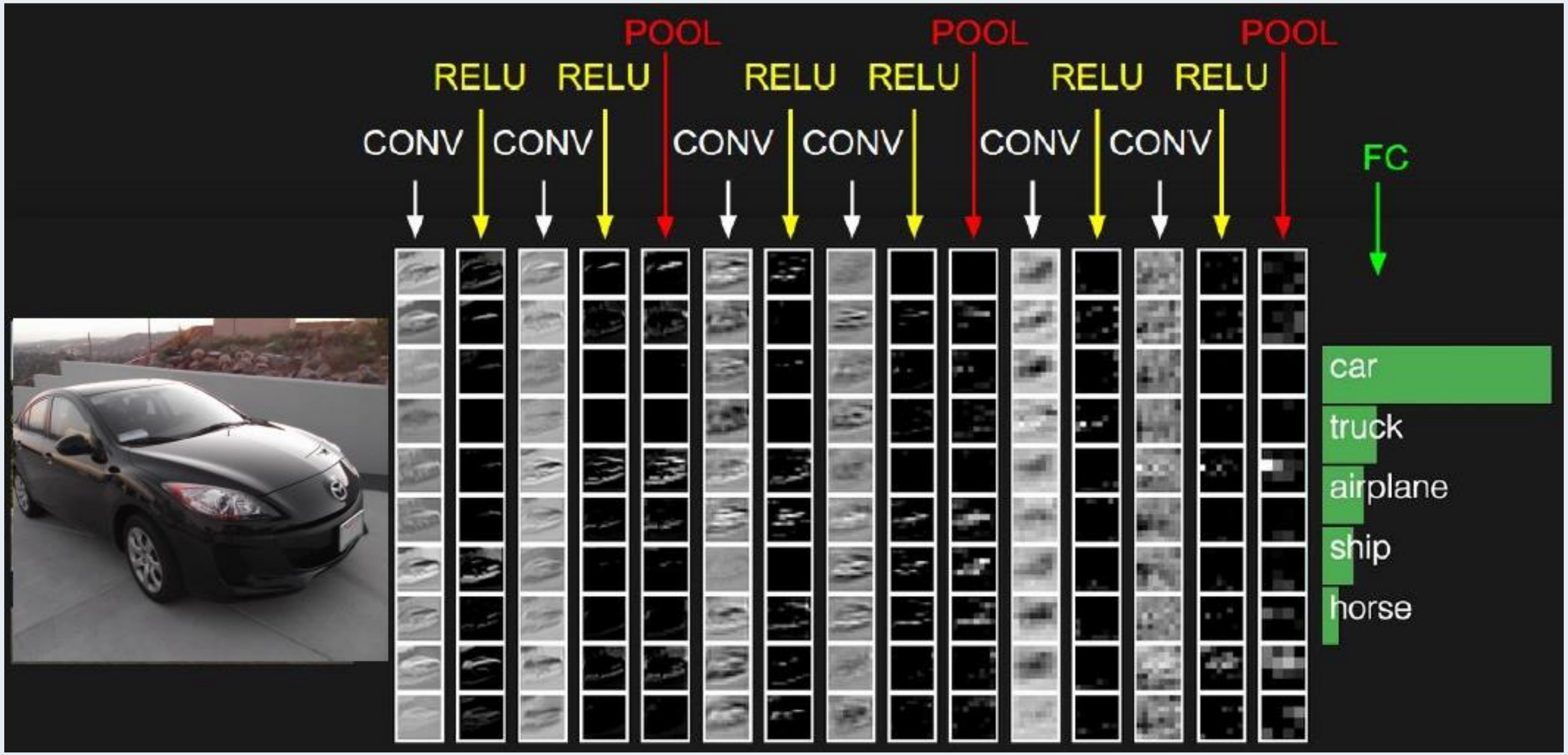
CNN 아키텍처

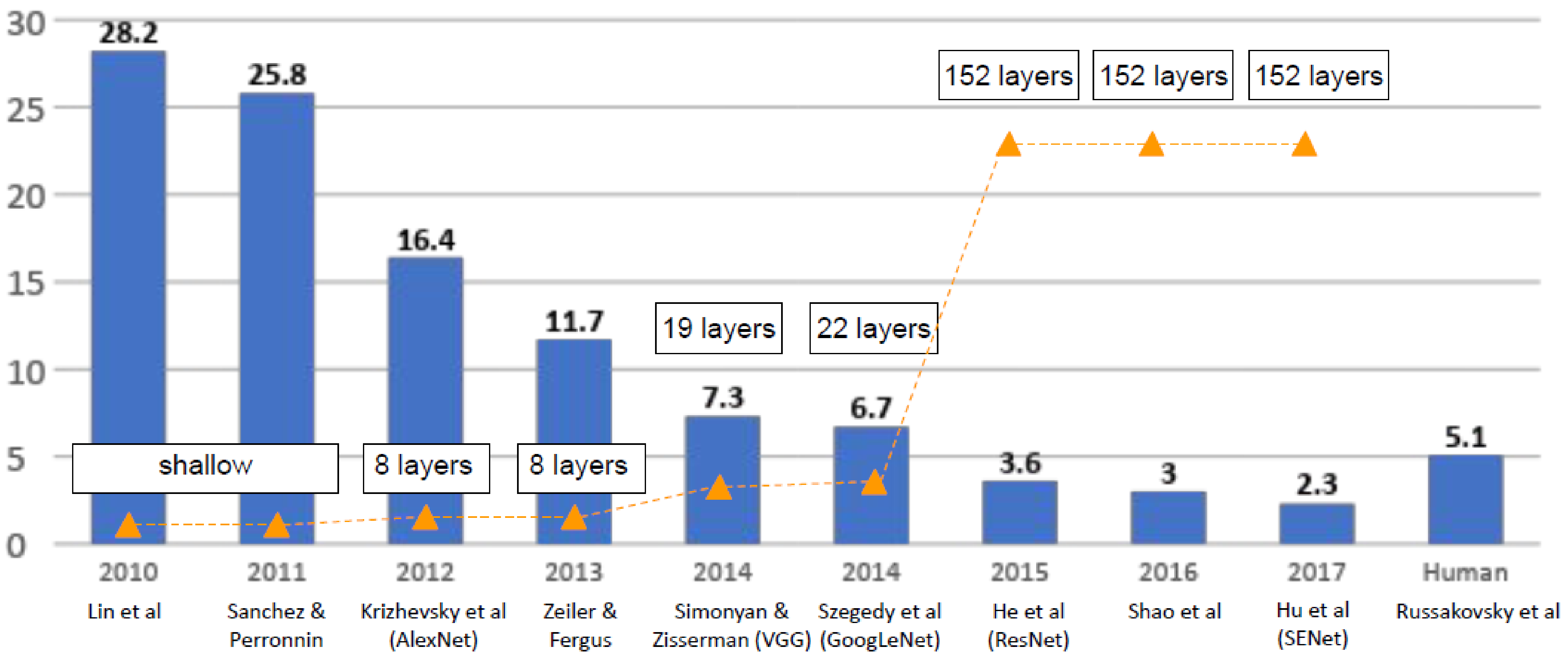

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

박경규

ImageNet

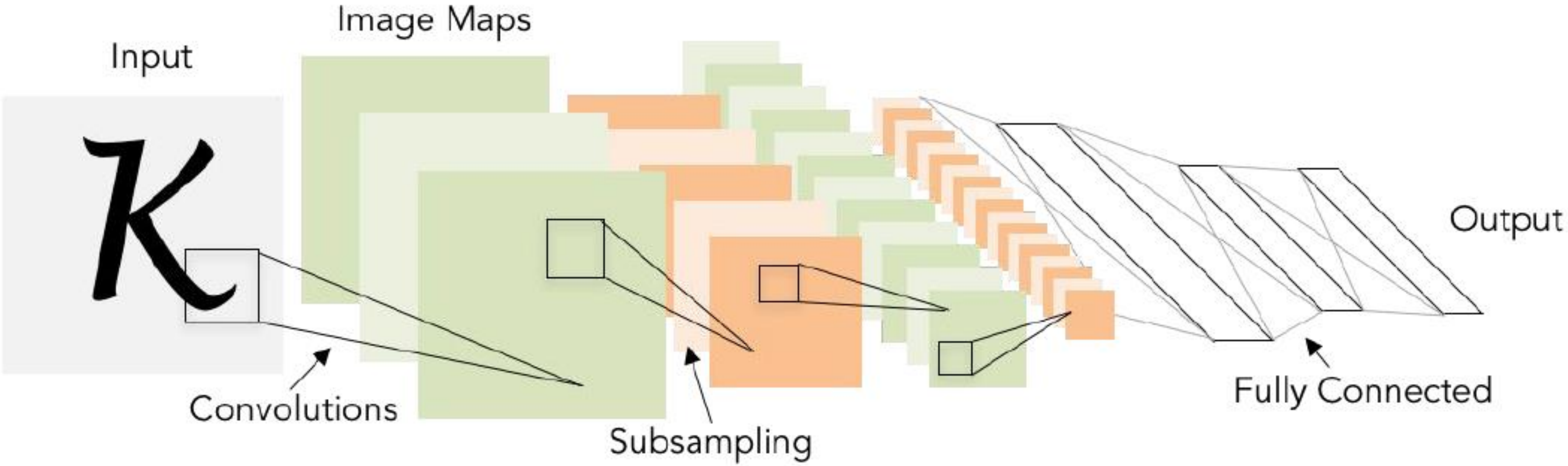
이미지 분류 모델을 측정하기 위한 데이터셋, 학습데이터셋 138G, 2만개 이상의 클래스, 약 1,400만장의 이미지

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



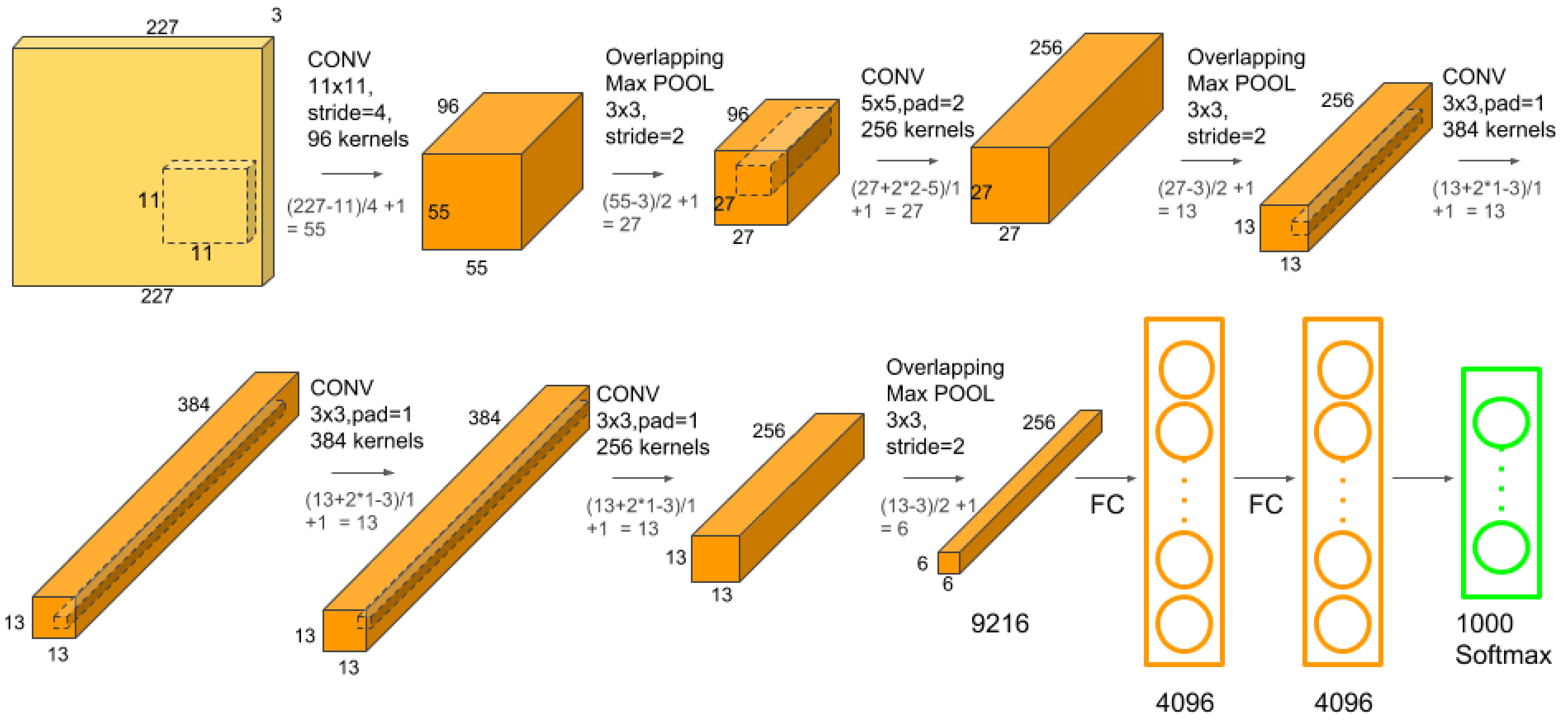
LeNet-5

[LeCun et al., 1998]



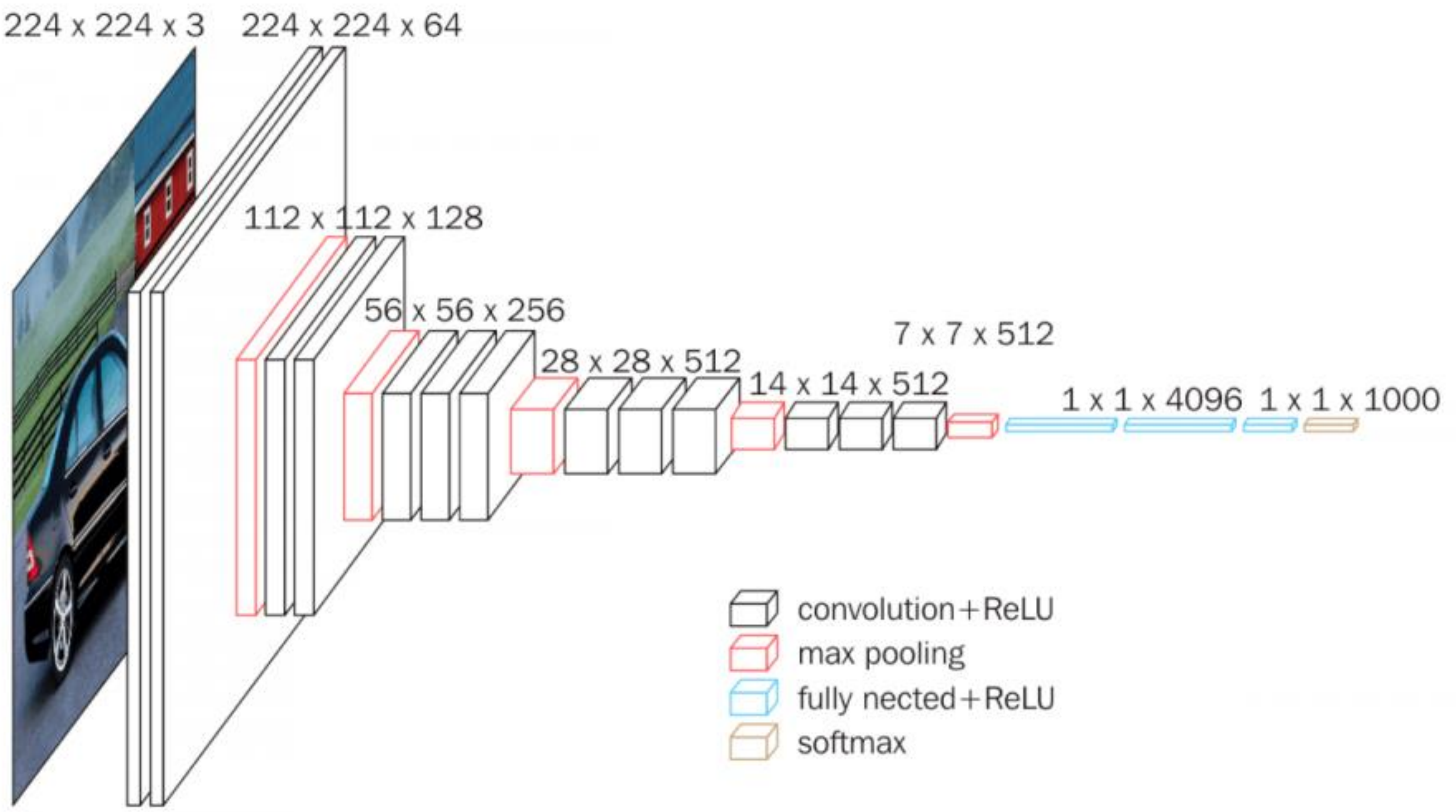
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

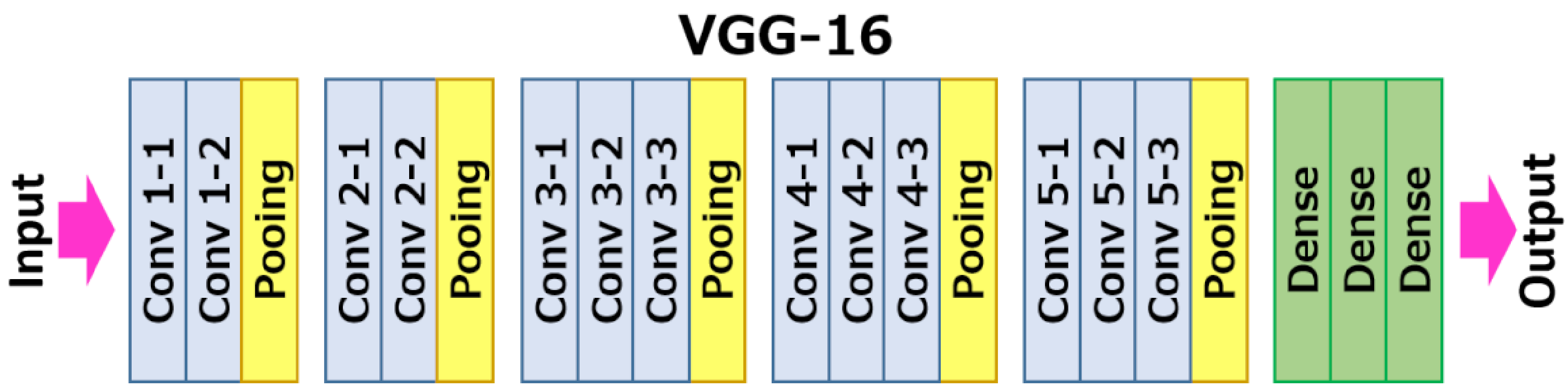
AlexNet



https://medium.com/analytics-vidhya/concept-of-alexnet-convolutional-neural-network-6e73b4f9ee30

VGG





https://neurohive.io/en/popular-networks/vgg16/

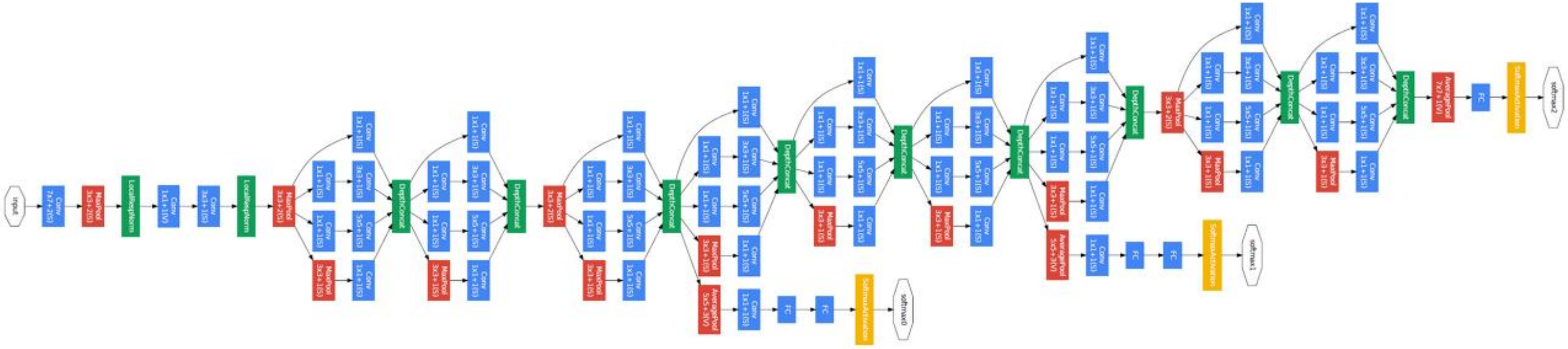
VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

```
Conv2D(64, (3,3))
```

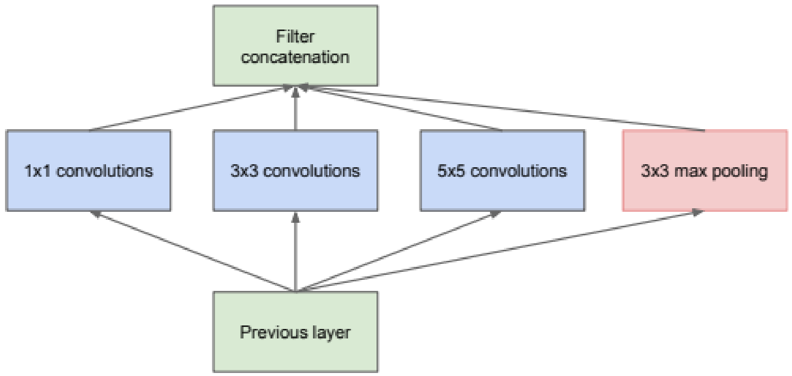
https://neurohive.io/en/popular-networks/vgg16/

GoogLeNet

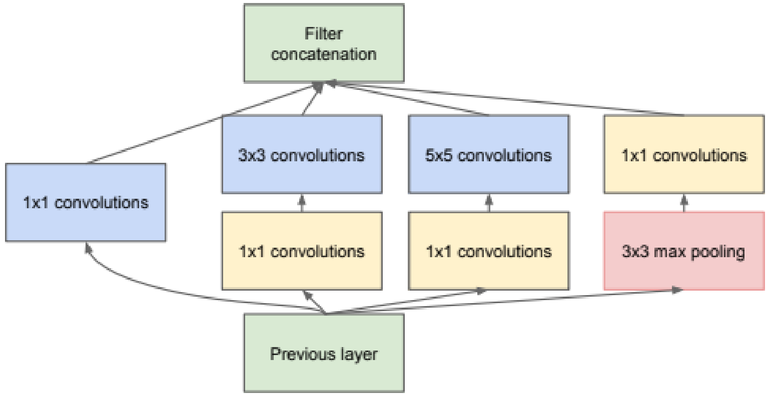


Deeper networks, with computational efficiency

- 22 layers
- Efficient "Inception" module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC'14 classification winner (6.7% top 5 error)



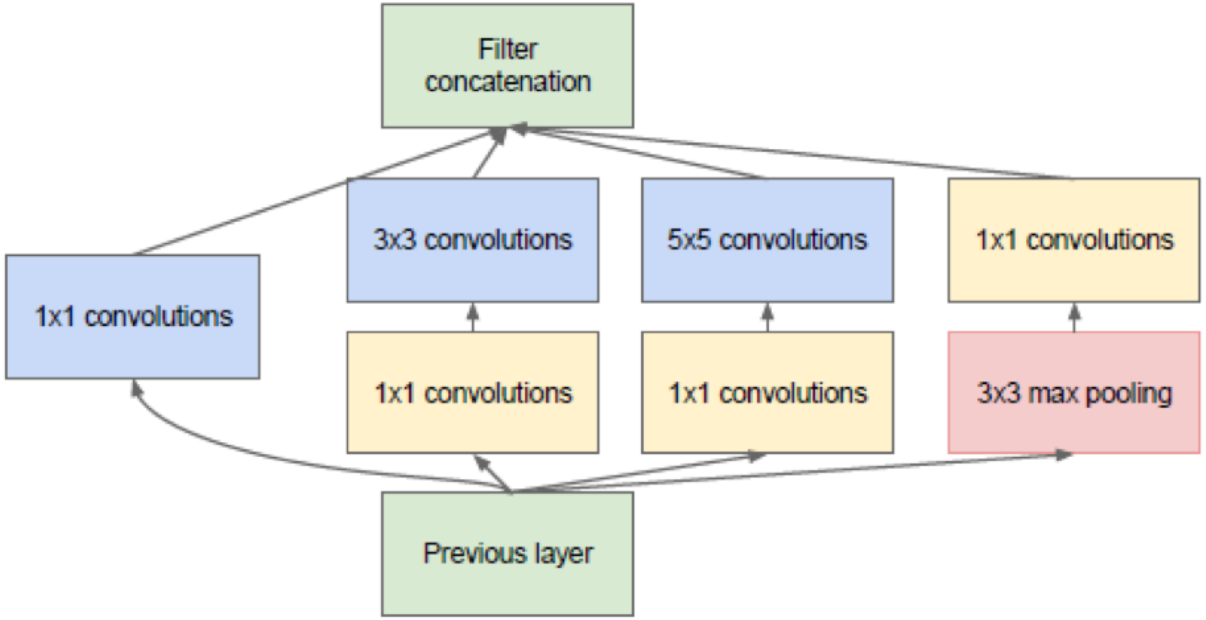
(a) Inception module, naïve version



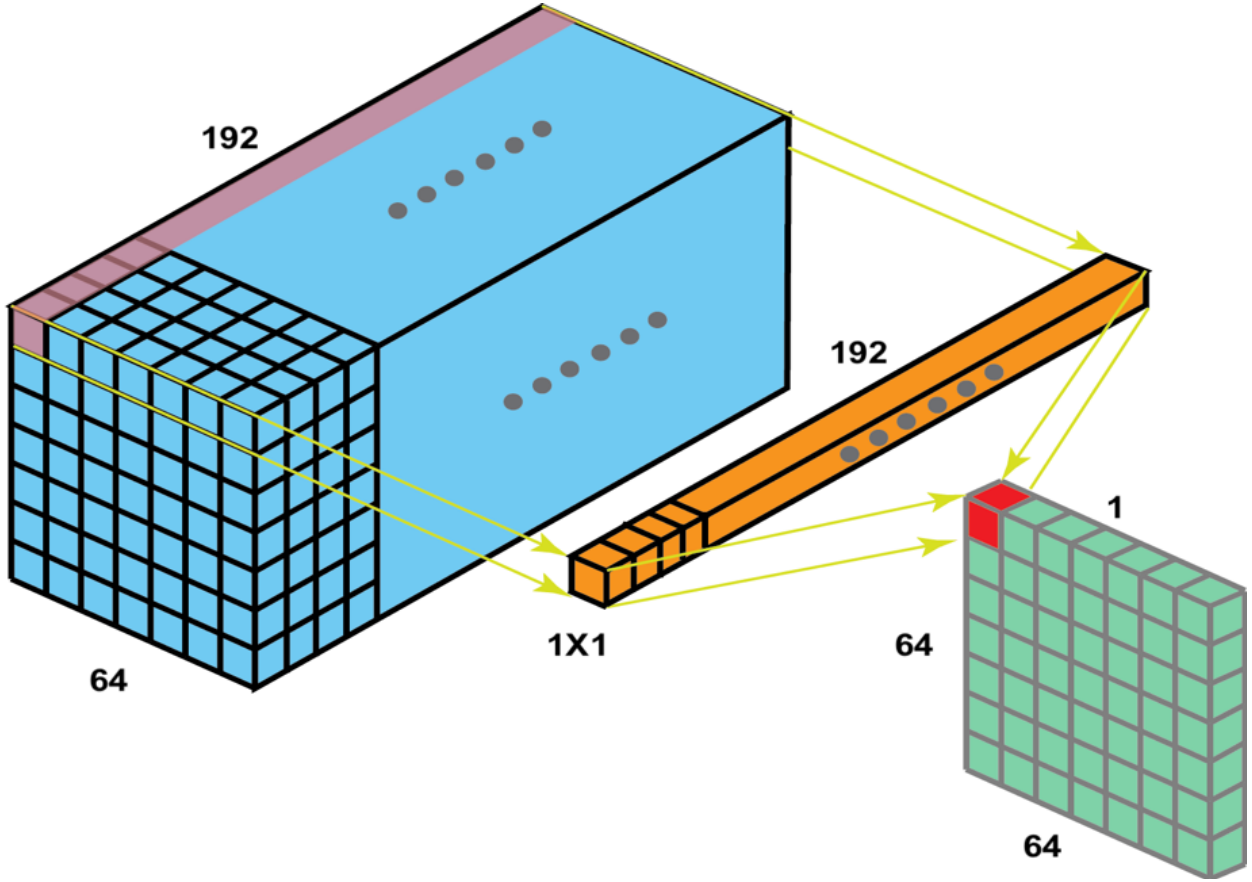
(b) Inception module with dimension reductions

https://bit.ly/3aZyi4S

GoogLeNet

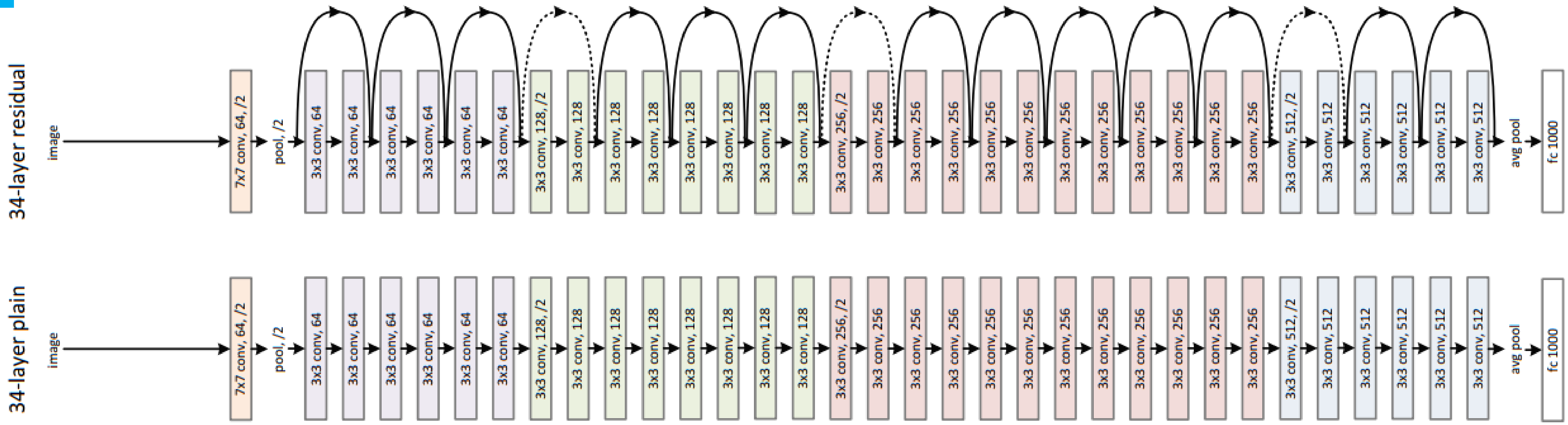


(b) Inception module with dimensionality reduction

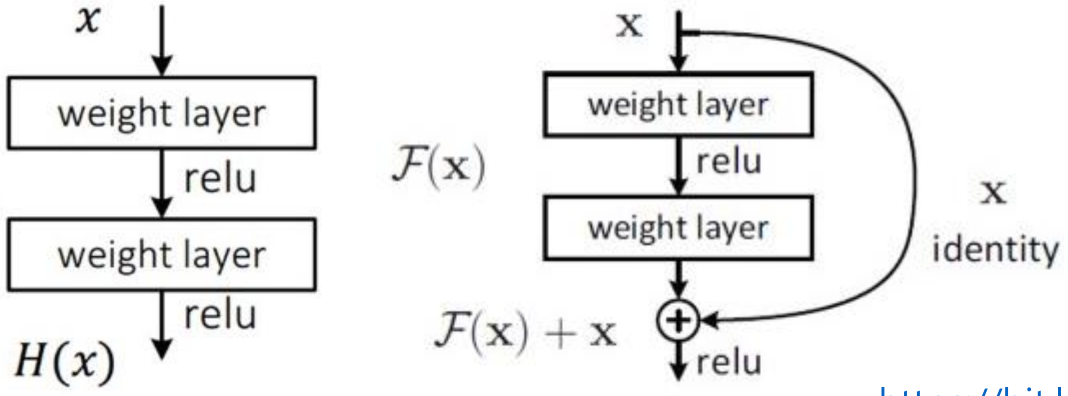


https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578

ResNet



■ 기존 네트워크와 ResNet의 구조



Very deep networks using residual connections

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

https://bit.ly/3cyhAKE

EfficientNet

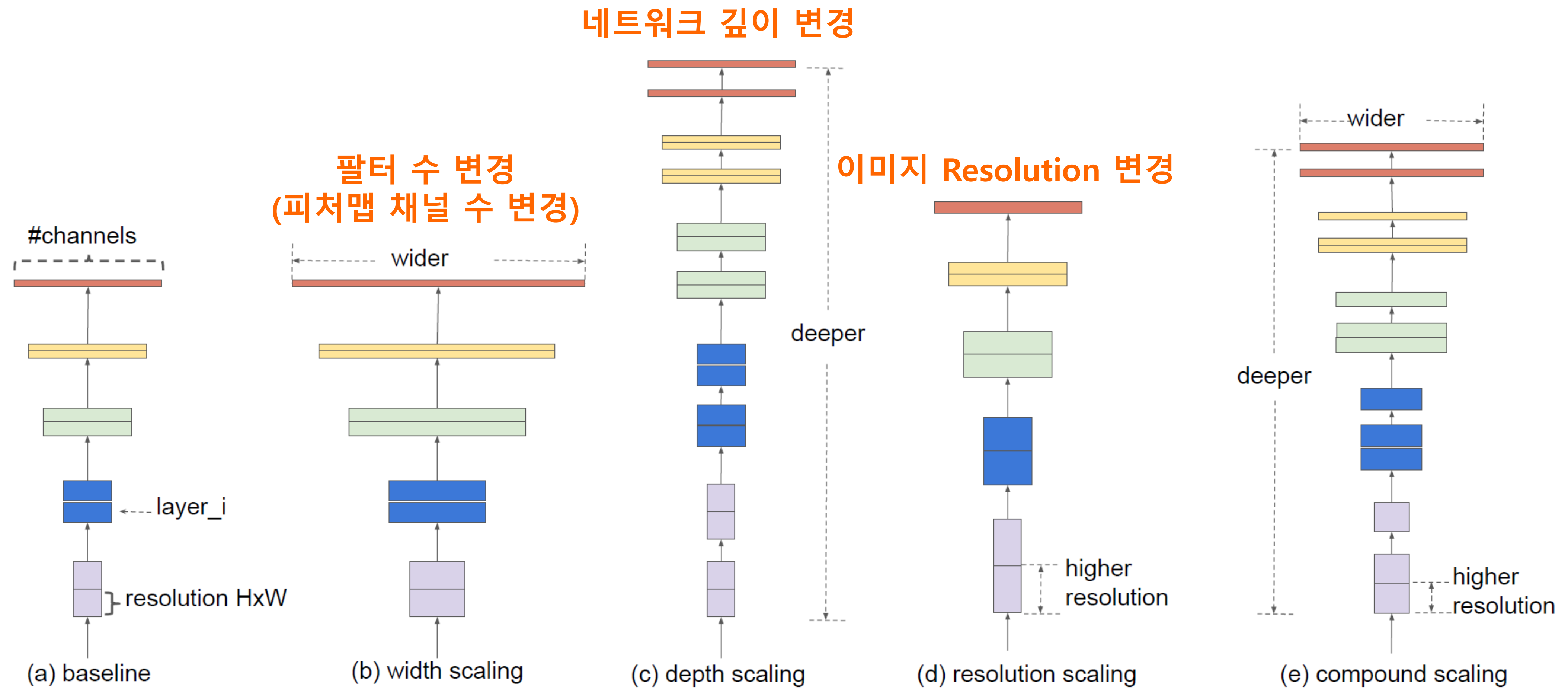
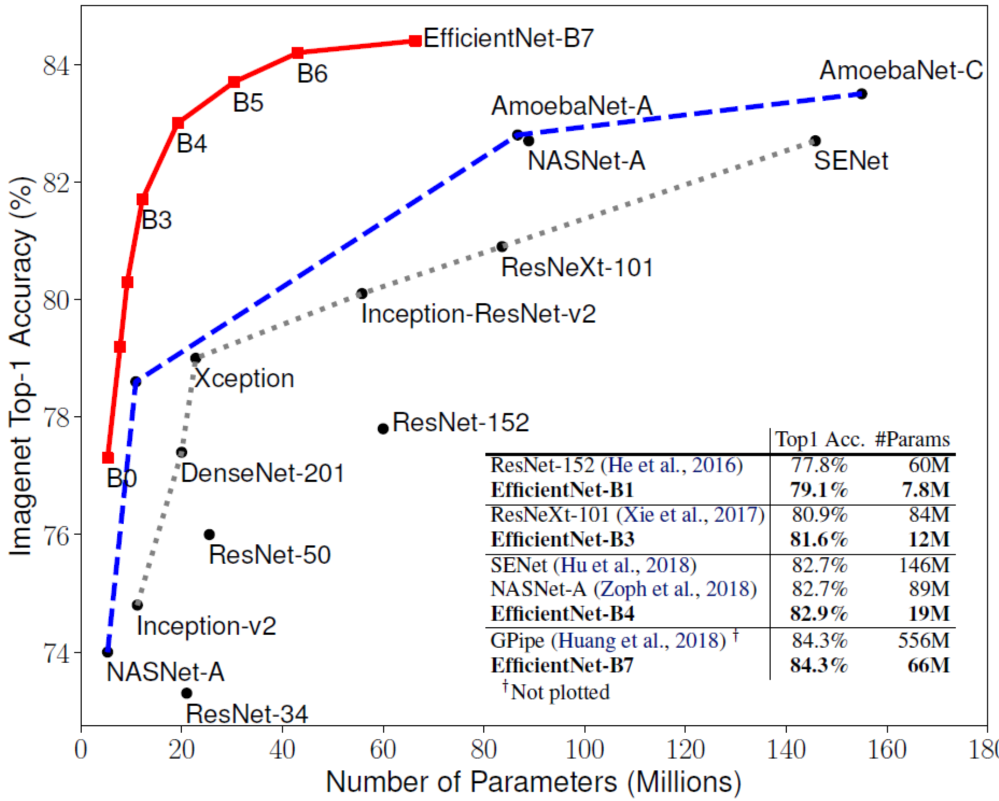


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

https://greeksharifa.github.io/computer%20vision/2022/03/01/EfficientNet/ https://velog.io/@iissaacc/EfficientNet

EfficientNet



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

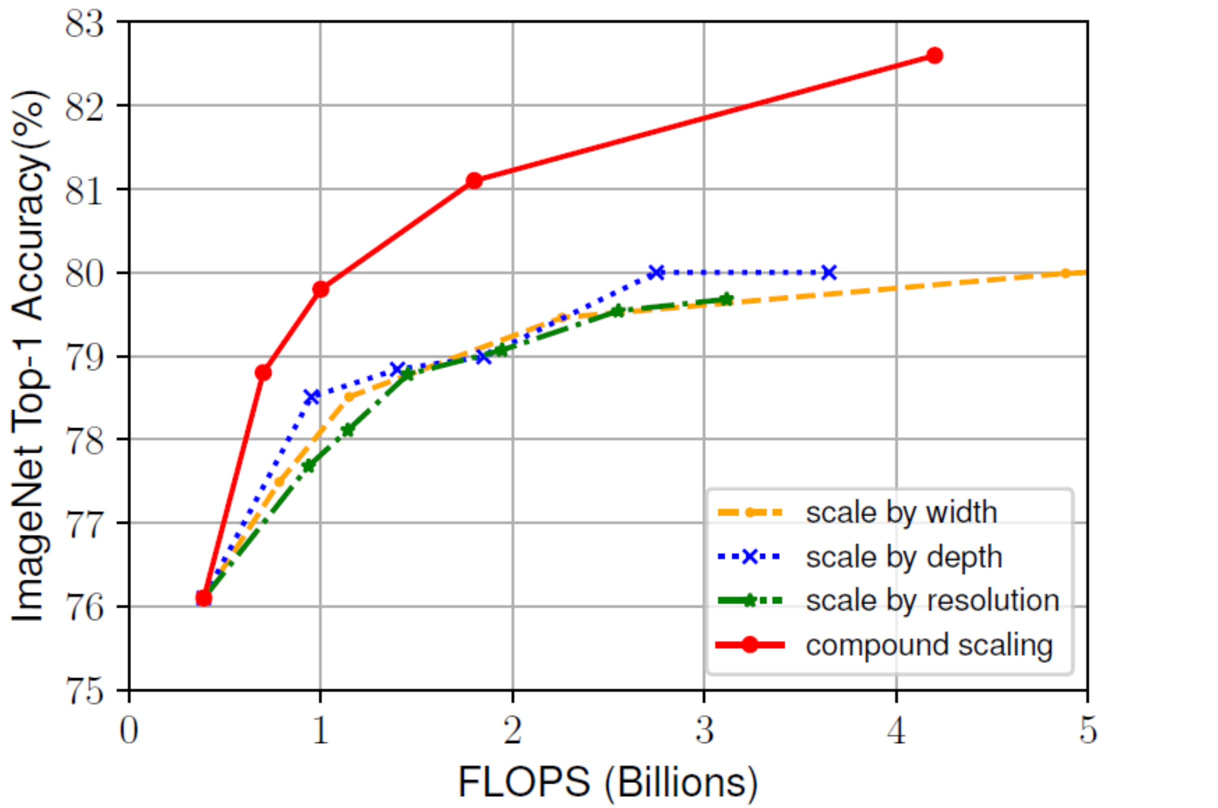
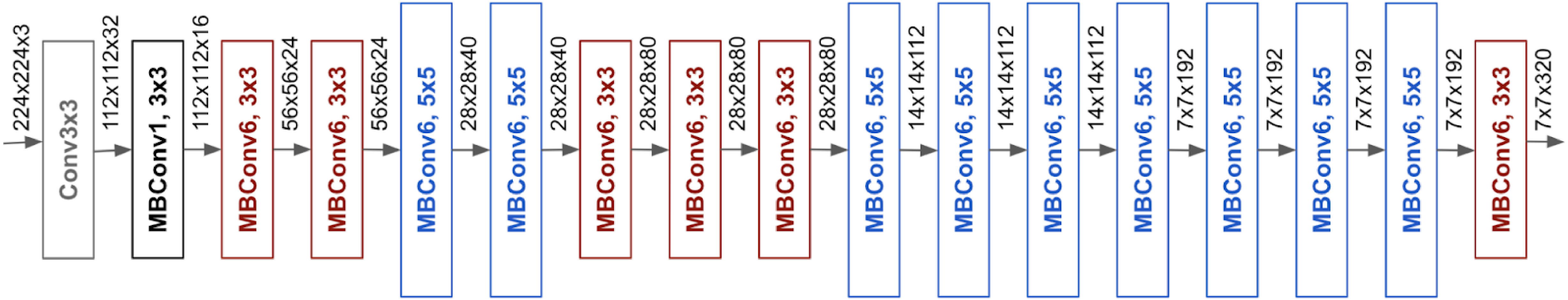


Figure 8. **Scaling Up EfficientNet-B0 with Different Methods.**

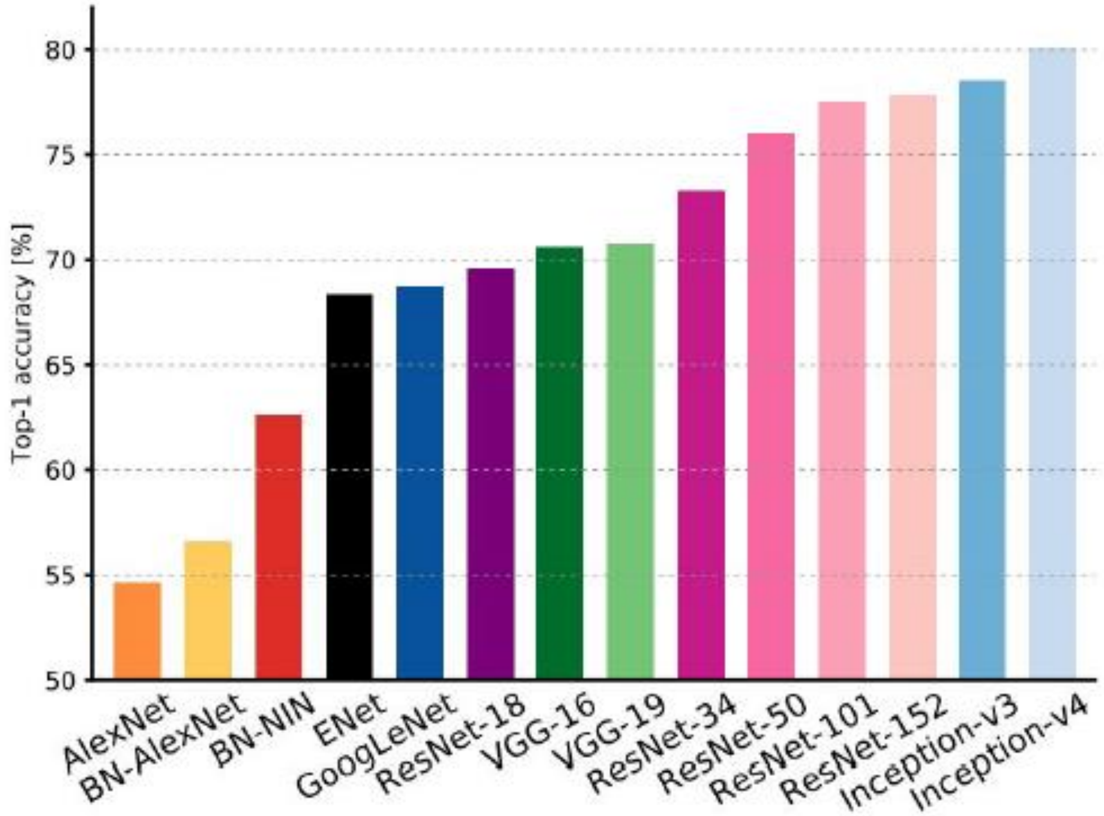
https://greeksharifa.github.io/computer%20vision/2022/03/01/EfficientNet/ https://velog.io/@iissaacc/EfficientNet

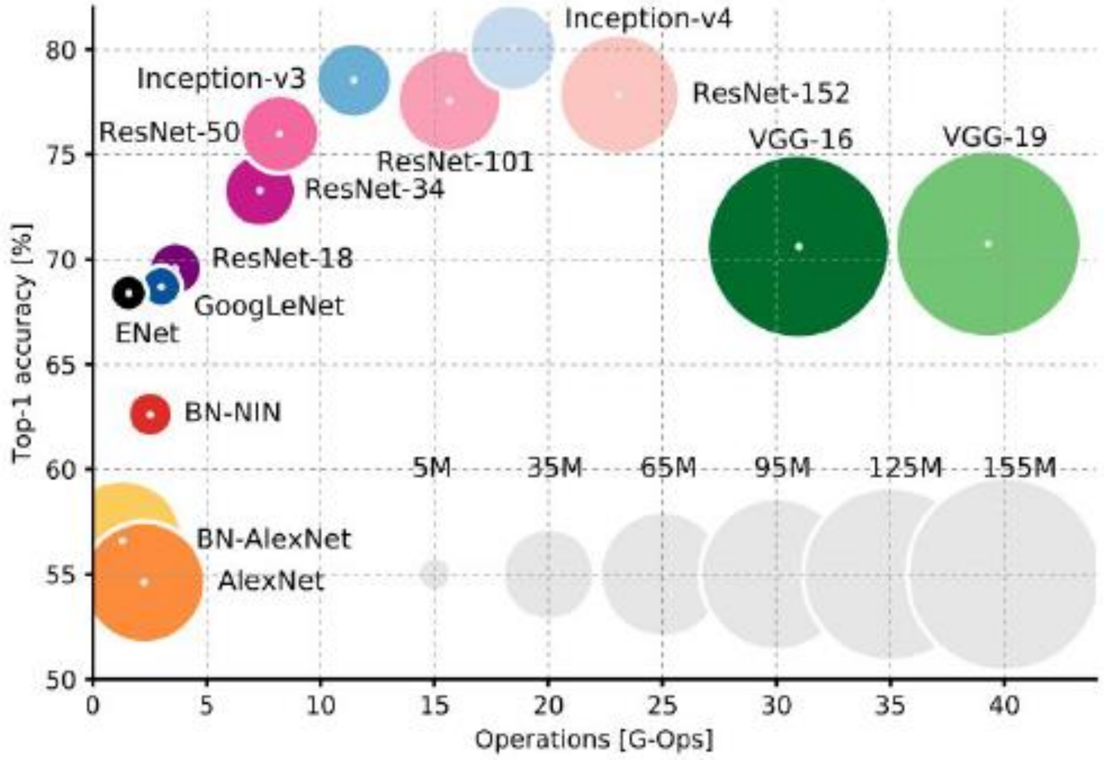
EfficientNet



https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html
https://github.com/keras-team/keras-applications/blob/master/keras_applications/efficientnet.py

Comparing complexity...





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

AlexNet 구현

데이터셋 로드

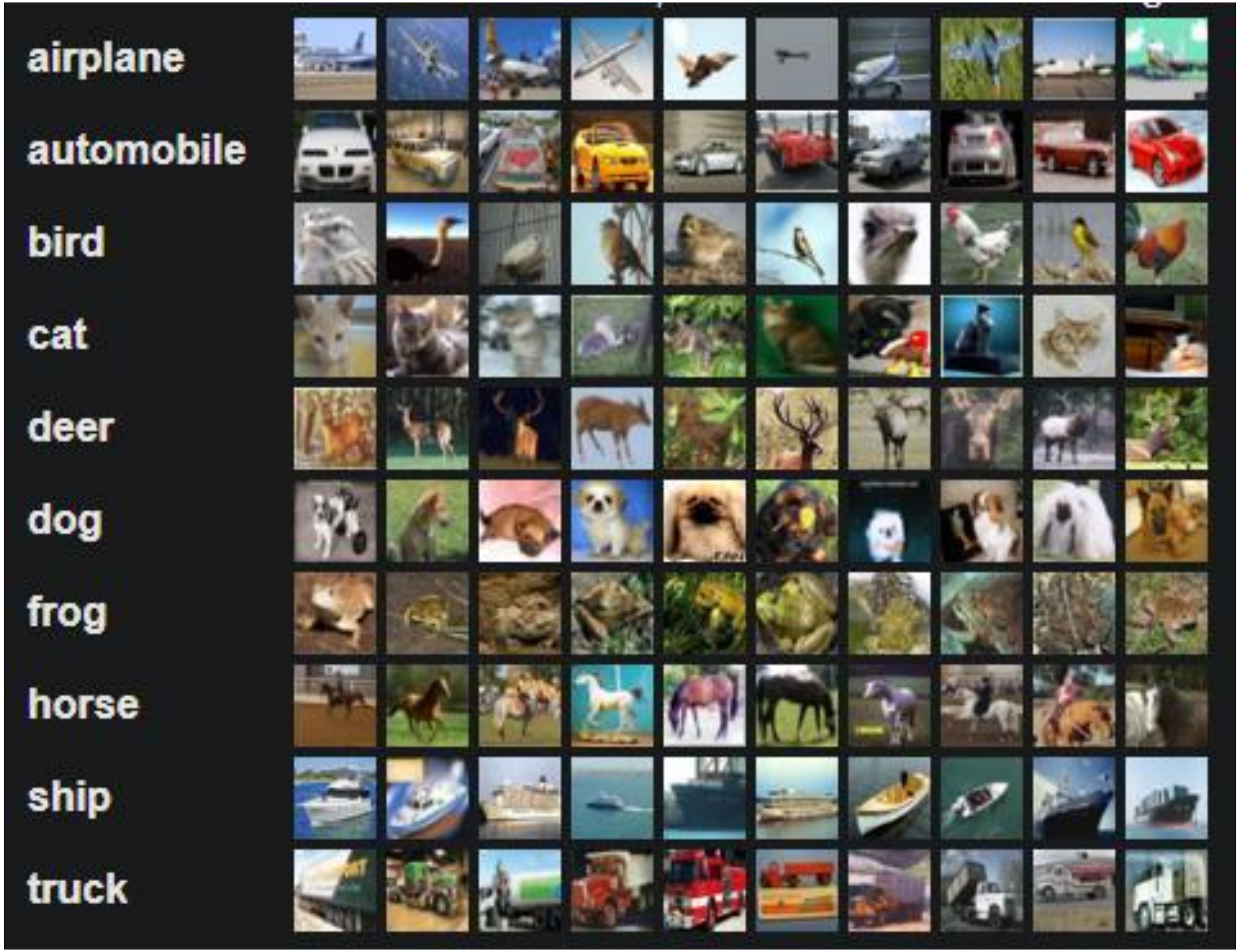
- CIFAR-10 dataset은 32x32픽셀의 60000개 컬러이미지가 포함되어 있습니다.
- 각 이미지는 10개의 클래스로 라벨링이 되어있습니다.
- MNIST와 같이 머신러닝 연구에 가장 널리 사용되는 dataset중 하나입니다.

```
[2] (train_images, train_labels), (test_images, test_labels) = keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
```

```
[3] CLASS_NAMES= ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

AlexNet 구현


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

라이브러리 임포트

```
[1]  import os
     import time
     import matplotlib.pyplot as plt

     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Activation, Dense, Dropout
     from tensorflow.keras.layers import Flatten, Conv2D, MaxPooling2D, BatchNormalization
     from tensorflow.keras.optimizers import SGD
     from tensorflow.keras.callbacks import TensorBoard
```

데이터셋 로드

- The CIFAR-10 dataset contains 60,000 colour images, each with dimensions 32x32px.
- The content of the images within the dataset is sampled from 10 classes.

```
[2]  (train_images, train_labels), (test_images, test_labels) = keras.datasets.cifar10.load_data()
```

```
[3]  CLASS_NAMES= ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

데이터셋 분리

```
[4]  train_images = train_images[:-5000]
     train_labels = train_labels[:-5000]
     validation_images  = train_images[-5000:]
     validation_labels = train_labels[-5000:]
```

```
[5]  train_ds = tf.data.Dataset.from_tensor_slices((train_images, train_labels))
     test_ds = tf.data.Dataset.from_tensor_slices((test_images, test_labels))
     validation_ds = tf.data.Dataset.from_tensor_slices((validation_images, validation_labels))
```

이미지 확인

```
[6]  plt.figure(figsize=(20,20))
     for i, (image, label) in enumerate(train_ds.take(5)):
         ax = plt.subplot(5,5,i+1)
         plt.imshow(image)
         plt.title(CLASS_NAMES[label.numpy()[0]])
         plt.axis('off')
```

데이터 전처리 함수

```
[7] def process_images(image, label):
        # Normalize images to have a mean of 0 and standard deviation of 1
        image = tf.image.per_image_standardization(image)
        # Resize images from 32x32 to 277x277
        image = tf.image.resize(image, (227,227))
        return image, label
```

데이터셋 준비

```
[8] train_ds_size = tf.data.experimental.cardinality(train_ds).numpy()
    validation_ds_size = tf.data.experimental.cardinality(validation_ds).numpy()
    test_ds_size = tf.data.experimental.cardinality(test_ds).numpy()

    print("Training dataset size:", train_ds_size)
    print("Validation dataset size:", validation_ds_size)
    print("Test dataset size:", test_ds_size)
```

```
Training dataset size: 45000
Validation dataset size: 5000
Test dataset size: 10000
```

```
[9] batch_size = 32

    train_ds = (train_ds
                     .map(process_images)
                     .shuffle(buffer_size=10000)
                     .batch(batch_size=batch_size, drop_remainder=True))

    validation_ds = (validation_ds
                     .map(process_images)
                     .shuffle(buffer_size=10000)
                     .batch(batch_size=batch_size, drop_remainder=True))

    test_ds = (test_ds
                     .map(process_images)
                     .batch(batch_size=batch_size, drop_remainder=True))
```

```
[10] for d in train_ds:
         print(d)
         break
```

```python
[11] model = Sequential()

     # 1st Convolutional Layer
     model.add(Conv2D(filters=96, input_shape=(227,227,3), kernel_size=(11,11), strides=(4,4), padding='valid'))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
     model.add(BatchNormalization())

     # 2nd Convolutional Layer
     model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same'))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
     model.add(BatchNormalization())

     # 3rd Convolutional Layer
     model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
     model.add(Activation('relu'))
     model.add(BatchNormalization())

     # 4th Convolutional Layer
     model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
     model.add(Activation('relu'))
     model.add(BatchNormalization())
```

```python
# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(BatchNormalization())

# Passing it to a dense layer
model.add(Flatten())

# 1st Dense Layer
model.add(Dense(4096, input_shape=(227*227*3,)))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())

# 2nd Dense Layer
model.add(Dense(4096))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())

#  output Layer
model.add(Dense(10))

model.add(Activation('softmax'))
```

```
[12] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
activation (Activation)      (None, 55, 55, 96)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
```

····································

```
_________________________________________________________________
dense_2 (Dense)              (None, 10)                40970
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 58,360,586
Trainable params: 58,341,450
Non-trainable params: 19,136
_________________________________________________________________
```

TensorBoard 로깅 디렉토리 설정

```
[13] tensorboard = TensorBoard('logs/alexnet')
```

모델 컴파일

```
[14] model.compile(loss='sparse_categorical_crossentropy',
                   optimizer=SGD(learning_rate=0.001),
                   metrics=['accuracy'])
```

모델 훈련

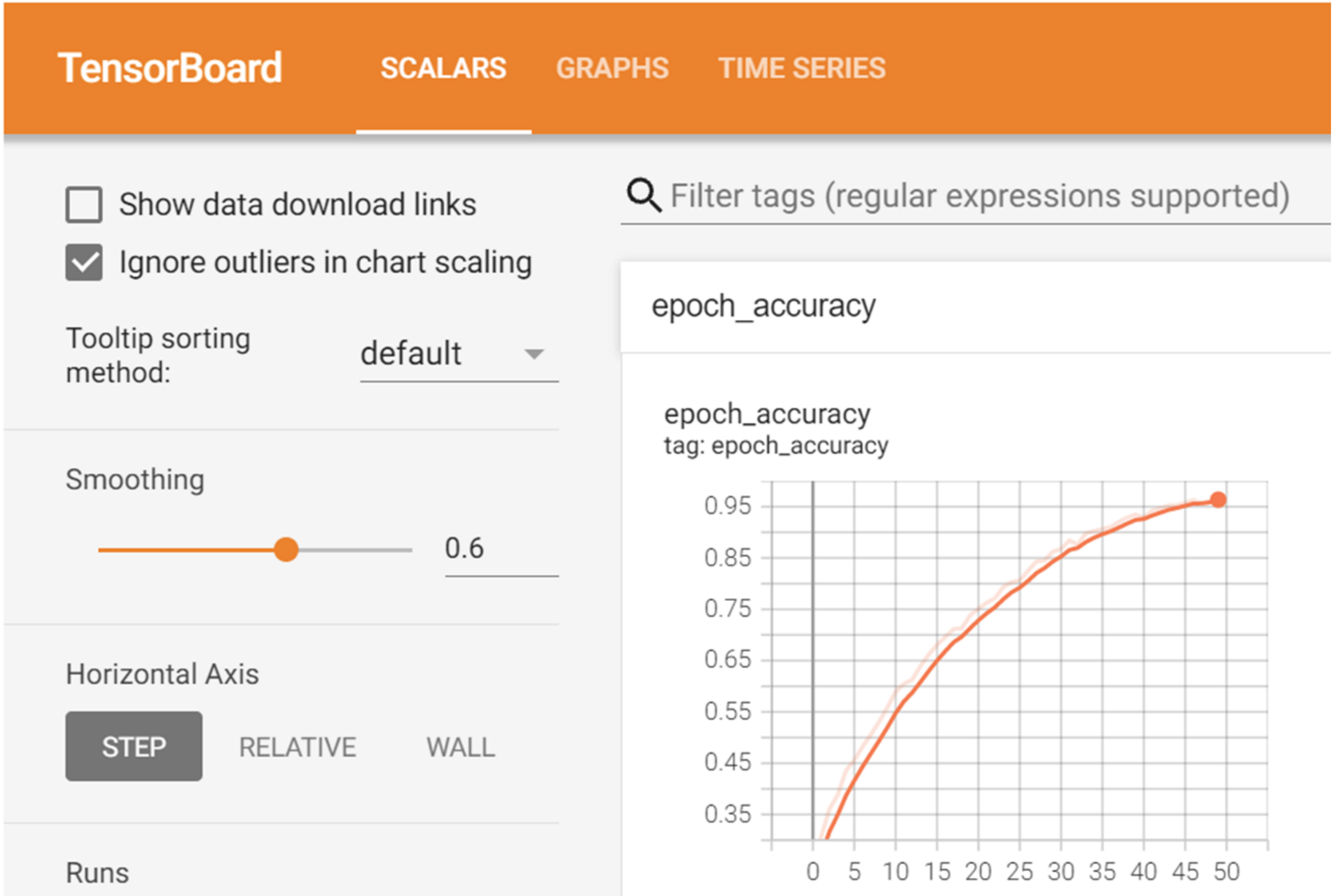
```
epochs = 30

model.fit(train_ds,
          epochs=epochs,
          validation_data=validation_ds,
          callbacks=[tensorboard])
```

```
Epoch 29/30
1406/1406 [==============================] - 82s 56ms/step - loss: 0.1929 - accuracy: 0.9334 - val_loss: 0.0358 - val_accuracy: 0.9964
Epoch 30/30
1406/1406 [==============================] - 81s 56ms/step - loss: 0.1751 - accuracy: 0.9413 - val_loss: 0.0350 - val_accuracy: 0.9954
```

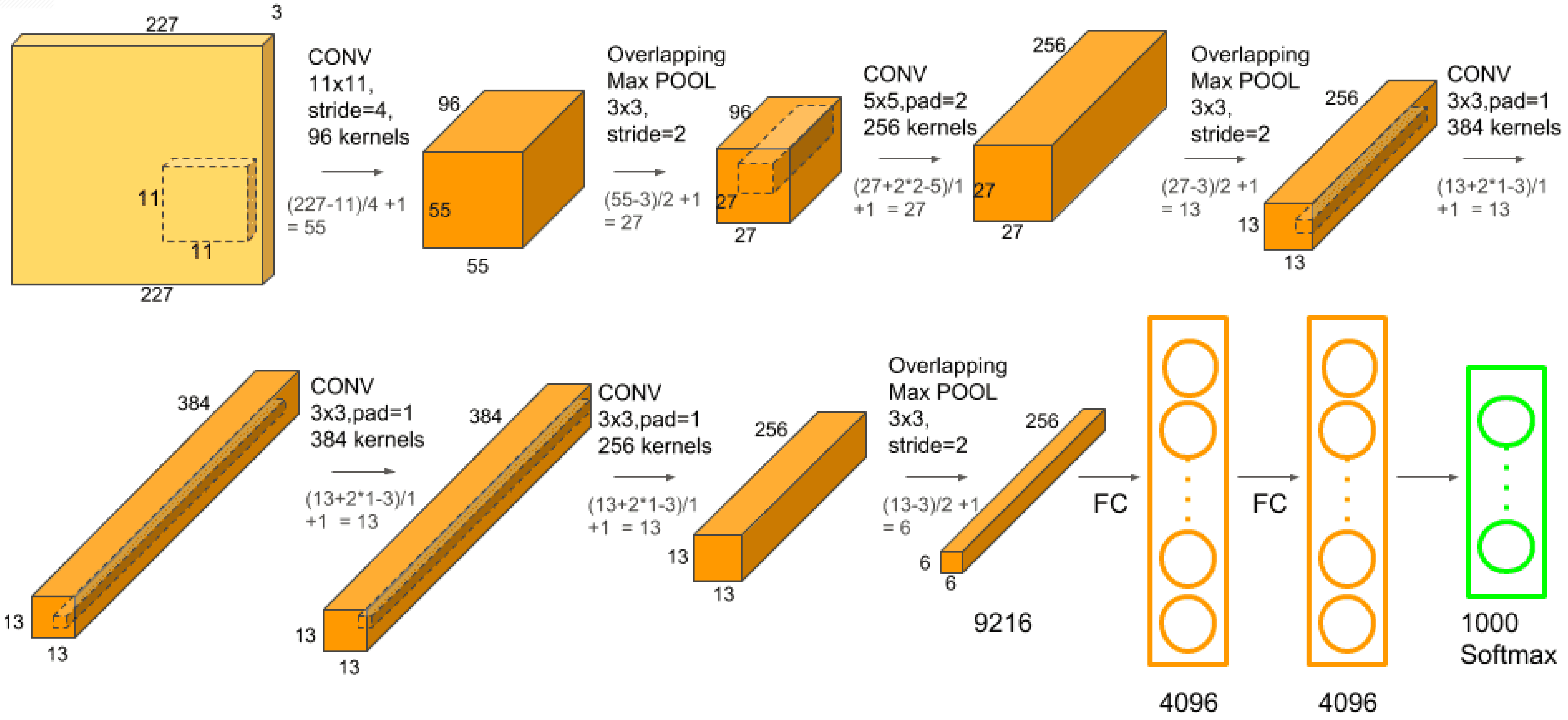
```
%load_ext tensorboard
```

```
%tensorboard --logdir logs/alexnet
```



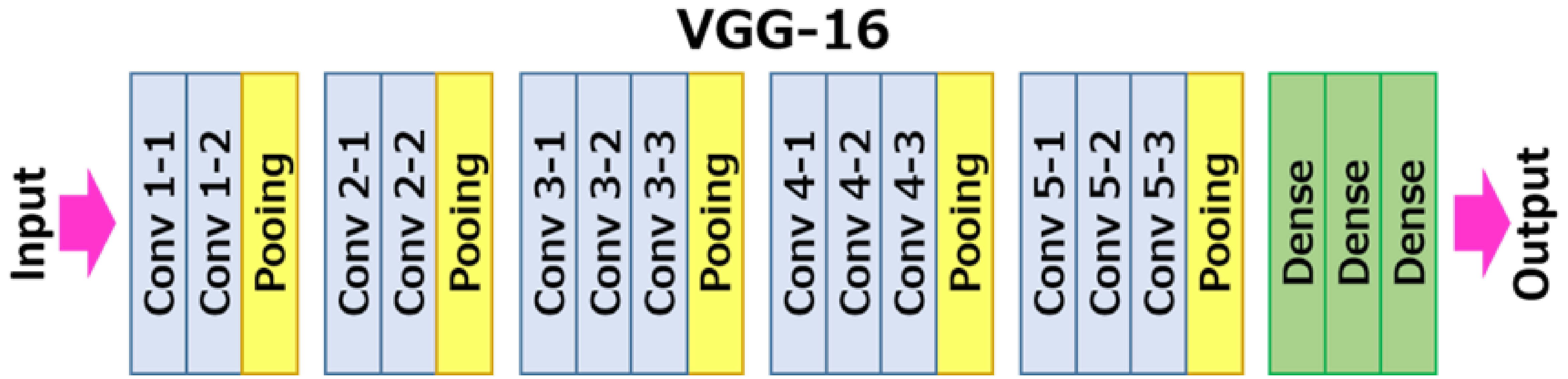
AlexNet 구현 실습

cnn_alexnet.ipynb



VGG-16 구현 실습

cnn_vgg.ipynb



THANK YOU

kgpark88@gmail.com