Keras

- TensorFlow나 PyTorch보다 사용자 친화적이다.
- 사용하기 쉬운 고수준의 API를 제공한다.

Keras 주요 Packages

- Layers :: 모델을 구현하는데 필요한 레이어를 생성하는 패키지로 **Input(), Dense(), Conv2D(), SimpleRNN()** 등 많은 클래스를 지원
  Layers Documentations: (https://www.tensorflow.org/api_docs/python/tf/keras/layers)
- Models :: 구현한 딥러닝 모델을 생성하는 패키지로 **Model(), Sequential()** 클래스를 지원
  Models Documentations: (https://www.tensorflow.org/api_docs/python/tf/keras/models)
- Losses :: 모델 학습에 필요한 Loss 값을 계산하는 패키지로 **BinaryCrossentropy(), MeanSquaredError()** 등 다양한 클래스를 지원
  Losses Documentations: (https://www.tensorflow.org/api_docs/python/tf/keras/losses)
- Optimizers :: 모델 학습시의 OPtimizers를 생성하는 패키지로 **SGD(), Adam(), RMSprop()** 등 다양한 클래스를 지원
  Optimizers Documentations: (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)

In [ ]:

```
# python==3.7  tensorflow==2.2.0 에서 진행하는 실습입니다.
# 사용할 모듈 import 하는 부분입니다.

# 저희는 tensorflow에 있는 keras를 사용합니다.
# 단순히 영어로 번역하시는 겁니다.
# tensorflow에서 keras를 import 하겠다.-> from tensorflow import keras
from tensorflow import keras ## or

# keras 모듈이 크기때문에 특정 레이어나 모델을 import 해서 사용하셔도 됩니다.
# tensorflow의 keras의 layers 중에 Input, Dense, Activation... 등을 import 하겠다.
# => from tensorflow.keras.layers import Input, Dense, Activation, ....
from tensorflow.keras.layers import Input, Dense, Activation, SimpleRNN, LSTM, #... 등

# 모델도 동일합니다.
# tensorflow의 keras의 models 중에 Model을 import 하겠다.
# => from tensorflow.keras.models import Model
from tensorflow.keras.models import Model, Sequential

import numpy as np
from pprint import pprint

# 아래 3줄은 구현한 모델의 시각화를 위한 모듈을 import 하는겁니다.
from IPython.display import SVG ## 모델을 시각화 하기위한 모듈
from tensorflow.keras.utils import model_to_dot ## 모델을 시각화 하기위한 모듈
from tensorflow.keras.utils import plot_model
```

Keras Layers

- ***Input()***, ***InputLayer()*** :: 구현할 네트워크의 entry point가 되는 Layer
  주로 bold 체의 파라미터들을 건드립니다.

tf.keras.layers.Input (**shape=None**, **batch_size=None**, dtype=None, input_tensor=None, sparse=False, name=None, ragged=False, **kwargs)

tf.keras.layers.InputLayer(**input_shape=None**, **batch_size=None**, dtype=None, input_tensor=None, sparse=False, name=None, ragged=False, **kwargs)

In [ ]:

```
## Input Layer
## Input() 혹은 InputLayer를 통과하게 되면,
## 변수의 데이터 타입이 ndarray 등에서 Tensor로 변하게 됩니다.

temp_input = np.arange(-5, 6, dtype='float32')
print(type(temp_input))
print('''------------------------------------------------------------------------''')
## Input(), InputLayer() 생성 ##
# inp = keras.layers.Input(shape=(6,)) # 입력을 Tensor로 사용 시
input_layer = keras.layers.InputLayer(input_shape=(6, )) # 입력을 레이어로 사용 시

## InputLayer를 거친 input 데이터
print(input_layer(temp_input))
print('''------------------------------------------------------------------------''')
## InputLayer를 거친 input 데이터의 타입
print(type(input_layer(temp_input)))
```

- ***Dense()*** :: 기본적인 NN Layer
  주로 bold 체의 파라미터들을 건드립니다.

tf.keras.layers.Dense (**units**, **activation=None**, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)

In [ ]:

```
## Dense Layer
## Dense Layer는 쉽게 생각하시면, 입력을 units의 개수 바꿔 출력해주는 Layer 입니다.
dense_input = np.random.randn(1, 10)

## DenseLayer 생성 ::
## 본 예시에서 dense_input의 개수는 10개이고, Dense의 units의 개수는 32개 입니다.
input_layer = keras.layers.InputLayer()
dense_layer = keras.layers.Dense(units=32)

## dense_input을 inputLayer를 거쳐 DenseLayer에 입력하는 부분입니다.
dense_output = dense_layer(input_layer(dense_input))

## dense_input의 shape와 dense_output의 shape을 확인하는 부분입니다.
## 확인해 보시면 (1, 10)이 (1, 32)로 변경된것을 보실 수 있습니다.
print(dense_input.shape)
print(dense_output.shape)
```

- ***Activation()*** :: 이전 레이어의 출력값에 activation 함수를 적용하는 Layer
  주로 bold 체의 파라미터들을 건드립니다.

tf.keras.layers.Activation (**activation**, **kwargs)

activation으로는 주로 'sigmoid', 'tanh', 'relu', 등을 사용합니다.

In [ ]:

```
## Activation Layer
activation_input = np.linspace(-1, 1, 10).astype('float')
print('Input: ₩n', activation_input)
print('''----------------------------------------------------------------------''')

input_layer = keras.layers.InputLayer()
activation_layer_sigmoid = keras.layers.Activation('sigmoid')
activation_layer_tanh = keras.layers.Activation('tanh')
activation_layer_relu = keras.layers.Activation('relu')
activation_layer_softmax = keras.layers.Activation('softmax')

sigmoid_output = activation_layer_sigmoid(input_layer(activation_input))
print('sigmoid_output: ₩n', sigmoid_output)
print('''----------------------------------------------------------------------''')

tanh_output = activation_layer_tanh(input_layer(activation_input))
print('tanh_output: ₩n', tanh_output)
print('''----------------------------------------------------------------------''')

relu_output = activation_layer_relu(input_layer(activation_input))
print('relu_output: ₩n', relu_output)
print('''----------------------------------------------------------------------''')
```

- ***SimpleRNN()*** :: 가장 기본적인 RNN Layer.
  Input_shape은 무조건 3차원으로 이루어 집니다.
  EX) (samples, seq_length, features) = (문장의 개수, 문장의 길이(단어의 개수), 단어의 벡터)

tf.keras.layers.SimpleRNN (**units**, activation='tanh', use_bias=True, kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initializer='zeros', kernel_regularizer=None, recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0, **return_sequences=False**, **return_state=False**, go_backwards=False, **stateful=False**, unroll=False, **kwargs)

In [ ]:

```
## SimpleRNN Layer
simpleRNN_input = np.random.randn(2, 5, 10)

## SimpleRNN Layer를 생성하는 부분입니다.
## Dense와 비슷하게, 입력의 units의 수만큼 바꾸어 출력합니다.
input_layer = keras.layers.InputLayer()
simpleRNN_layer = keras.layers.SimpleRNN(units=4)

## input과 output을 비교해 보시면, shape이 (2, 5, 10)에서 (2, 4)로 변경된걸
## 보실 수 있습니다. 일반적으로 input shape의 제일 앞부분은
## 문장의 개수 혹의 이미지의 개수 (sample의 수 혹은 batch의 수)를
## 나타내기 때문에 계산에 영향을 받지 않습니다.
simpleRNN_output = simpleRNN_layer(input_layer(simpleRNN_input))
print(simpleRNN_input.shape)
print(simpleRNN_output.shape)
print(simpleRNN_output)
print('''----------------------------------------------------------------------''')

## SimpleRNN Layer에 return_sequences 파라미터를 True로 변경한 레이어입니다.
input_layer = keras.layers.InputLayer()
simpleRNN_layer = keras.layers.SimpleRNN(units=4, return_sequences=True)

## 입력과 출력의 shape을 확인해 보시면, (2, 5, 10)에서 (2, 5, 4)로 변한것을
## 보실 수 있습니다. return_sequences=True가 되면 단어 하나가 들어갈때마다
## output을 출력해 주기때문에 문장의 길이(단어의 개수)가 유지됩니다.
simpleRNN_output = simpleRNN_layer(input_layer(simpleRNN_input))
print(simpleRNN_input.shape)
print(simpleRNN_output.shape)
print(simpleRNN_output)
print('''----------------------------------------------------------------------''')

## SimpleRNN Layer에 return_state 파라미터를 True로 변경한 레이어입니다.
input_layer = keras.layers.InputLayer()
simpleRNN_layer = keras.layers.SimpleRNN(units=4, return_state=True)

## return_state=True가 되면, RNN Layer는 output 뿐만 아니라 hidden_state도
## 함께 반환하게 됩니다.
simpleRNN_output, state_h = simpleRNN_layer(input_layer(simpleRNN_input))
print(simpleRNN_input.shape)
print(simpleRNN_output.shape)
print(state_h.shape)
print('''----------------------------------------------------------------------''')

## return_sequences=True와 return_state=True를 함께 사용할 때도 있습니다.
input_layer = keras.layers.InputLayer()
simpleRNN_layer = keras.layers.SimpleRNN(units=4, return_sequences=True, return_state=True)
simpleRNN_output, state_h = simpleRNN_layer(input_layer(simpleRNN_input))
print(simpleRNN_output.shape)
print(state_h.shape)
```

- ***LSTM()*** :: 메모리 셀이라는 세로운 state가 추가된 RNN => LSTM (Long Short term memory)
  tf.keras.layers.LSTM (**units**, activation='tanh', recurrent_activation='sigmoid', use_bias=True, kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initializer='zeros', unit_forget_bias=True, kernel_regularizer=None, recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0, implementation=2, **return_sequences=False**, **return_state=False**, go_backwards=False, **stateful=False**, time_major=False, unroll=False, **kwargs)

In [ ]:

```
## LSTM Layer
## SimpleRNN과 LSTM의 차이점은 두가지 정도로
## 첫번째로 LSTM의 내부구조(계산식)가 SimpleRNN 보다 복잡하다는 점과
## 두번째로는 Memory cell 이라는 state가 추가된 점입니다.
## 결과적으로는 SimpleRNN과 동일한 output의 shape을 보입니다.
LSTM_input = np.random.randn(2, 5, 10)

input_layer = keras.layers.InputLayer()
LSTM_layer = keras.layers.LSTM(units=4)

LSTM_output = LSTM_layer(input_layer(LSTM_input))
print(LSTM_output.shape)
print('''----------------------------------------------------------------------''')

LSTM_layer = keras.layers.LSTM(units=4, return_sequences=True)
LSTM_output = LSTM_layer(input_layer(LSTM_input))
print(LSTM_output)
print('''----------------------------------------------------------------------''')

## 유일하게 다른 점은 return_state=True가 되었을때
## return 되는 state가 state_h와 state_c로 2개 라는 점입니다.
LSTM_layer = keras.layers.LSTM(units=4, return_state=True)
LSTM_output, state_h, state_c = LSTM_layer(input_layer(LSTM_input))
print(LSTM_output)
print(state_h)
print(state_c)
```

- ***Bidirectional()*** :: RNN 계열의 레이어에 적용해 양방향으로 반드는 레이어

tf.keras.layers.Bidirectional(**layer**, **merge_mode='concat'**, weights=None, backward_layer=None, **kwargs)

In [ ]:

```
## Bidirectional Layer
LSTM_input = np.random.randn(2, 5, 10)

## Bidirectional Layer를 사용하게 되면 RNN계열의 레이어가 정방향 뿐만 아니라 역방향으로도
## 계산을 진행하게 됩니다.
## 예를들어 "가 나 다 라"를 Bidirectional-RNN을 사용해 입력한다면
## ['가', '나', '다', '라']와 ['라', '다', '나', '가'] 총 2개를 입력한것과 동일한
## 기능을 하게 됩니다.

## merge_mode는 두개의 결과를 어떻게 반환할것인지를 결정하는 파라미터입니다.
## merge_mode=None 이라면 두개를 따로 반환하며
## merge_mode=concat 이라면, 두개를 Concatenate해서 반환합니다.
input_layer = keras.layers.InputLayer()
bidirectional_layer = keras.layers.Bidirectional(keras.layers.LSTM(units=4), merge_mode=None)

bidirectional_output_forward, bidirectional_output_backward = bidirectional_layer(
                                                                input_layer(LSTM_input))
print(bidirectional_output_forward)
print(bidirectional_output_backward)
print('''--------------------------------------------------------------------------''')

input_layer = keras.layers.InputLayer()
bidirectional_layer = keras.layers.Bidirectional(keras.layers.LSTM(4), merge_mode='concat')

bidirectional_output = bidirectional_layer(input_layer(LSTM_input))
print(bidirectional_output)
```

- ***Embedding()*** :: 파라미터 input_dim의 개수만큼 고정된 크기(output_dim)의 벡터를 생성해 index로 접근하는 레이어

tf.keras.layers.Embedding(**input_dim**, **output_dim**, embeddings_initializer='uniform', embeddings_regularizer=None, activity_regularizer=None, embeddings_constraint=None, **mask_zero=False**, input_length=None, **kwargs)

In [ ]:

```
## Embedding Layer
embedding_input = np.arange(0, 6)

## Embedding Layer 생성
## Embedding Layer에서 input_dim은 몇 개의 벡터가 필요한지를 나타내며, 예를들면 단어의 개수가 됩니다
## output_dim은 벡터를 크기를 몇개로 할것인지를 설정하는 벡터로, 임의로 입력하시면 됩니다.
input_layer = keras.layers.InputLayer()
embedding_layer = keras.layers.Embedding(input_dim=len(embedding_input), output_dim=5)
```

In [ ]:

```
## 입력과 출력을 확인해 보시면, 하나의 인덱스에 고유의 벡터가 생성된것을 볼 수 있습니다.
## 입력은 무조건 array로 들어가야 하기 때문에 reshape(-1)을 해줍니다.
for input_data in embedding_input:
    input_data = input_data.reshape(-1)
    print(input_data, " : ", embedding_layer(input_layer(input_data)))

## input_dim 보다 큰값을 입력하면 Error가 발생합니다.
# print('6', " : ", embedding_layer(input_layer(np.asarray([6]))))
```

- ***Dot()*** :: 두 벡터를 축(axse)에 맞춰 내적연산을 하는 레이어
  tf.keras.layers.Dot (**axes**, normalize=False, **kwargs)

In [ ]:

```
## Dot
## Dot Layer는 두 벡터를 내적하는 Layer 입니다.
## 자주 쓰는 Layer는 아니기에 존재만 알고 넘어가시면 됩니다.
input_layer = keras.layers.InputLayer()
dot_input_1=np.arange(4).reshape(2, 2)
dot_input_2=np.arange(4).reshape(2, 2)
pprint(dot_input_1)
pprint(dot_input_2)
print('''----------------------------------------------------------------------''')

dot_ax1 = keras.layers.Dot(axes=1)
pprint(dot_ax1([input_layer(dot_inp_1), input_layer(dot_inp_2)]))
print('''----------------------------------------------------------------------''')

dot_ax2 = keras.layers.Dot(axes=2)
dot_inp_1=np.arange(4).reshape(1, 2, 2)
dot_inp_2=np.arange(4).reshape(1, 2, 2)
pprint(dot_ax2([input_layer(dot_inp_1), input_layer(dot_inp_2)]))
```

- ***Dropout()*** :: NN의 전체 연결 중 일정 비율(rate)을 제거하는 레이어
  tf.keras.layers.Dropout (**rate**, noise_shape=None, seed=None, **kwargs)

In [ ]:

```
## Dropout
dropout_input = np.arange(3).reshape(1, 3)

## Dropout Layer를 사용해기 위해 Dense Layer도 함께 생성 해 줍니다.
## Dense Layer의 16개 출력 중 25%가 Dropout Layer에 의해 0으로 변경됩니다.
input_layer = keras.layers.InputLayer()
dense_layer = keras.layers.Dense(16)
dropout_layer = keras.layers.Dropout(0.25)

dense_output = dense_layer(input_layer(dropout_input))
dropout_output = dropout_layer(dense_output, training=True)

## Dense Layer의 output과 dropout Layer를 거친 output을 비교해 보시면
## 16의 25%인 4개의 출력이 0 혹은 -0으로 변경된것을 보실 수 있습니다.
print('Dense_output: ₩n', dense_output)
print('Dtopout_output: ₩n', dropout_output)
```

- ***Conv2D()*** :: 2D convolution 연산을 진행하는 레이어 (for CNN)
  tf.keras.layers.Conv2D (**filters**, **kernel_size**, **strides=(1, 1)**, **padding='valid'**, data_format=None, dilation_rate=(1, 1), **activation=None**, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)

In [ ]:

```
# Conv2D                          # (batch_size, rows, cols, channels)
x = np.random.randn(10, 28, 28, 3) # (samples, height, width, features)
print(x.shape)
# filters: 출력의 개수, kernel_size: kernel의 shape
conv2D = keras.layers.Conv2D(filters=4, kernel_size=3, activation='relu')
conv_out = conv2D(inp(x))
print(conv_out.shape)
conv2D = keras.layers.Conv2D(filters=4, kernel_size=3, activation='relu', padding='same')
conv_out = conv2D(inp(x))
print(conv_out.shape)
```

- ***AveragePooling2D()*** :: 2D에서 pooling size 내부의 값들의 평균을 구하는 레이어 (for CNN)
  tf.keras.layers.AveragePooling2D(**pool_size=(2, 2)**, **strides=None**, **padding='valid'**, data_format=None, **kwargs)

In [ ]:

```
# output_shape = (input_shape - pool_size + 1) / strides)
x = np.random.randn(10*28*28*3).astype('float').reshape(10, 28, 28, 3)
avg_pooling = keras.layers.AveragePooling2D(pool_size=(2, 2), padding='same')
avg_out = avg_pooling(inp(x))
print(avg_out.shape)
```

- ***MaxPooling2D()*** :: 2D에서 poolling size 내부의 값들 중 최대값을 구하는 레이어 (for CNN)
  tf.keras.layers.MaxPool2D(**pool_size=(2, 2)**, **strides=None**, **padding='valid'**, data_format=None, **kwargs)

In [ ]:

```
# output_shape = (input_shape - pool_size + 1) / strides)
x = np.random.randn(10*28*28*3).astype('float').reshape(10, 28, 28, 3)
avg_pooling = keras.layers.MaxPooling2D(pool_size=(2, 2))
avg_out = avg_pooling(inp(x))
print(avg_out.shape)
```

- ***Flatten()*** :: 다차원의 입력을 일차원으로 만드는 레이어
  tf.keras.layers.Flatten (data_format=None, **kwargs)

In [ ]:

```
# Flatten
x = np.random.randn(3, 28, 28, 3)
print(x.shape)
flatten = keras.layers.Flatten()
print(flatten(inp(x)).shape)
print(28*28*3)
```

- ***Concatenate()*** :: 두 벡터를 축에 맞춰 하나의 벡터로 묶어주는 레이어
  tf.keras.layers.Concatenate (axis=-1, **kwargs)

In [ ]:

```
# Concatenate
x_1 = np.arange(20).reshape(2, 5, 2)
x_2 = np.arange(30). reshape(3, 5, 2)
print(x_1.shape)
print(x_2.shape)
concat = keras.layers.Concatenate(axis=0)
print(concat([inp(x_1), inp(x_2)]).shape, '₩n')

x_1 = np.arange(20).reshape(2, 2, 5)
x_2 = np.arange(40). reshape(2, 4, 5)
print(x_1.shape)
print(x_2.shape)
concat = keras.layers.Concatenate(axis=1)
print(concat([inp(x_1), inp(x_2)]).shape, '₩n')

x_1 = np.arange(30).reshape(2, 5, 3)
x_2 = np.arange(20). reshape(2, 5, 2)
print(x_1.shape)
print(x_2.shape)
concat = keras.layers.Concatenate(axis=2)
print(concat([inp(x_1), inp(x_2)]).shape, '₩n')
```

Keras Models

- ***Sequential()*** :: 단순히 이전 레이어의 출력이 다음 레이어의 입력이 되는 선형적 흐름 모델 생성에 사용
- ***Model()*** :: 선형적 흐름 모델이 아는 경우의 모델 생성에 사용

In [ ]:

```
# Sequential()
seq_model = keras.models.Sequential()
seq_model.add(keras.layers.InputLayer(input_shape=(2,)))
seq_model.add(keras.layers.Dense(32, activation='sigmoid'))
seq_model.add(keras.layers.Dense(16, activation='relu'))
seq_model.add(keras.layers.Dropout(0.2))
seq_model.add(keras.layers.Dense(2, activation='softmax'))

seq_inp = np.arange(20).reshape(-1, 2)
seq_model.predict(seq_inp)

%matplotlib inline
SVG(model_to_dot(seq_model, show_shapes=True, dpi=65).create(prog='dot', format='svg'))
```

In [ ]:

```
# Model()
inp_layer_1 = keras.layers.Input(shape=(2,))
inp_layer_2 = keras.layers.Input(shape=(2,))
dense_layer_1_1 = keras.layers.Dense(32)
dense_layer_1_2 = keras.layers.Dense(8)
dense_layer_2 = keras.layers.Dense(4)
concat = keras.layers.Concatenate(axis=-1)
output_layer = keras.layers.Dense(2, activation='softmax')

dense_1_1_out = dense_layer_1_1(inp_layer_1)
dense_1_2_out = dense_layer_1_2(inp_layer_2)
concat_out = concat([dense_1_1_out, dense_1_2_out])
den_2_out = dense_layer_2(concat_out)
output = output_layer(den_2_out)

model = keras.models.Model([inp_layer_1, inp_layer_2], output)

%matplotlib inline
SVG(model_to_dot(model, show_shapes=True, dpi=65).create(prog='dot', format='svg'))
```

Keras Loss & Optimizer

자주 사용되는 Losses

- ***BinaryCrossentropy()*** :: 데이터를 True/False로 나눌때 사용되는 손실함수, 'binary_crossentropy'
- ***CategoricalCrossentropy()*** :: 데이터를 다수의 카테고리로 나눌때 사용되는 손실함수, 'categorical_crossentropy'
- ***CosineSimilarity()*** :: 데이터간의 코사인 유사도를 Loss로 사용하는 손실함수, 'cosine_similarity'
- ***MeanSquaredError()*** :: square(y_true - y_pred)를 Loss로 사용하는 손실함수, 'mean_squared_error', 'mse'

자주 사용되는 Optimizers

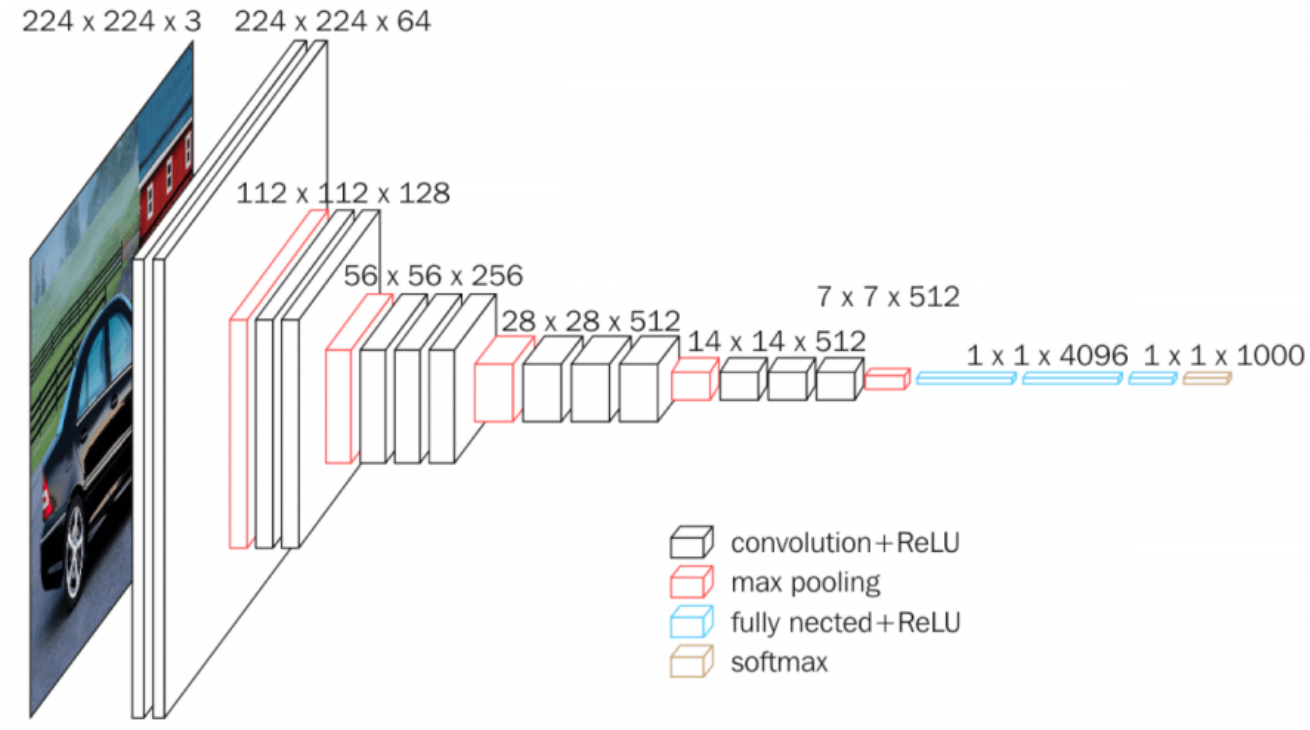
- ***SGD()***, 'SGD'
- ***Adam()***, 'Adam'
- ***RMSprop()***, 'RMSprop'

In [ ]:

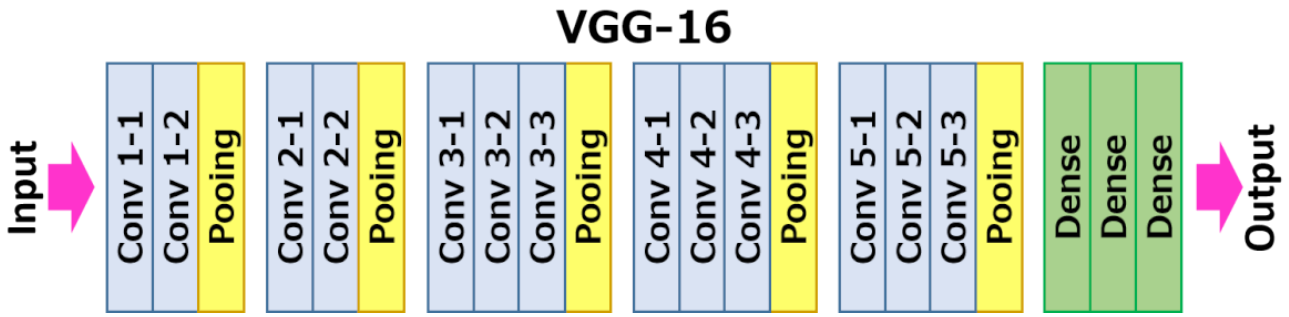
```
## 사용법
# model.compile(loss='binary_crossentropy', optimizer='SGD')
# model.compile(loss='categorical_crossentropy', optimizer='Adam')
model.compile(loss='mean_squared_error', optimizer='RMSprop')
## 필요에 따라 셋 중 하나만 사용하시는 겁니다.
```

모델 구현하기 문제 1.

1. VGG16 모델을 만들어 보세요. (CNN 모델)
2. VGG16 Model



- VGG16 Archtecture



이미지 출처: https://neurohive.io/en/popular-networks/vgg16/ (https://neurohive.io/en/popular-networks/vgg16/)

In [ ]:

```
model = Sequential()
model.add(keras.layers.InputLayer(input_shape=(224,224,3)))
model.add(keras.layers.Conv2D(filters=64//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=64//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Conv2D(filters=128//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=128//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Conv2D(filters=256//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=256//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=256//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Conv2D(filters=512//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=512//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=512//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Conv2D(filters=512//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=512//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.Conv2D(filters=512//2, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(7*7*512//2, activation='relu'))
model.add(keras.layers.Dense(7*7*512//2, activation='relu'))
model.add(keras.layers.Dense(7*7*512//2, activation='relu'))
model.add(keras.layers.Dense(1000, activation='softmax'))
```

In [ ]:

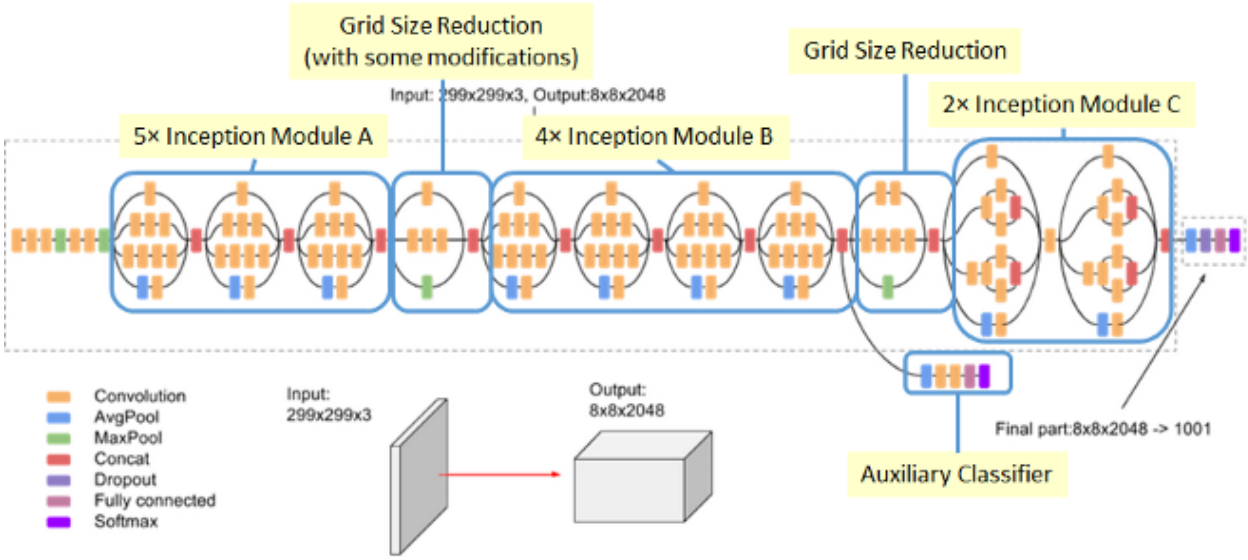
```
%matplotlib inline
## SVG는 jupyter notebook에 바로 모델을 그려주는 모듈입니다.
SVG(model_to_dot(model, show_shapes=True, dpi=65).create(prog='dot', format='svg'))

## plot_model은 to_file='file_name'으로 모델의 그림을 저장해주는 모듈입니다.
# plot_model(model, to_file='./vgg_model.png', show_shapes=True)
```

모델 구현하기 문제 2.

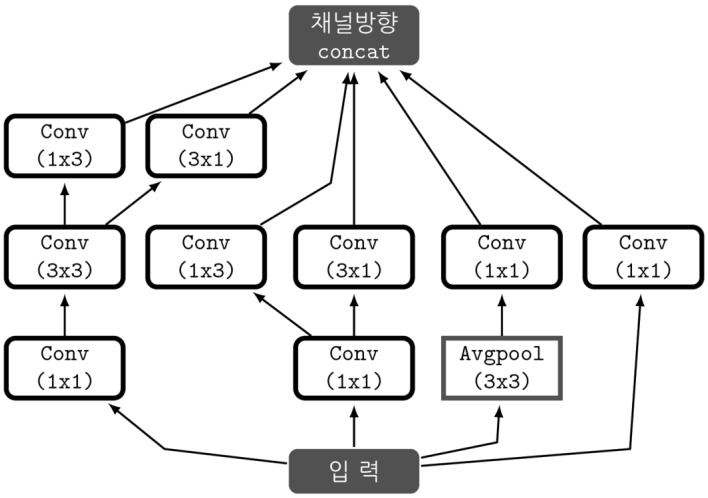
2. Google Inception v3 모델 중 제일 마지막에 있는 Inception Module C를 만들어 보세요. (CNN 모델)
3. Inception Module C 다음의 출력층을 만들고 연결해보세요.

- Input의 shape은 (8, 8, 1280)
- Inception Module C 내부의 filters = {1: 768, 3:(384, 768), 5:(96, 256), pool:256}

- Inception V3 Full Architecture



- Inception Module C (왼쪽부터 5, 3, pool, 1)



이미지 출처(Architecture): https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c (https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c)

이미지 출처(Inception Module C): https://datascienceschool.net/view-notebook/8d34d65bcced42ef84996b5d56321ba9/ (https://datascienceschool.net/view-notebook/8d34d65bcced42ef84996b5d56321ba9/)

In [ ]:

```
def inception_module_c(x):
    # 1, 3, 5, pool
    # 384, (192, 384), (48, 128), 128

    ## inception_module_c의 레이어를 생성하는 부분입니다.

    ## 5, 그림에서 제일 왼쪽 부분입니다.
    conv_01 = keras.layers.Conv2D(filters=48*2, kernel_size=(1, 1),
                                  activation='relu', padding='same')
    conv_02 = keras.layers.Conv2D(filters=128*2, kernel_size=(3, 3),
                                  activation='relu', padding='same')
    conv_03 = keras.layers.Conv2D(filters=128*2, kernel_size=(1, 3),
                                  activation='relu', padding='same')
    conv_04 = keras.layers.Conv2D(filters=128*2, kernel_size=(3, 1),
                                  activation='relu', padding='same')

    ## 3, 그림에서 중간 왼쪽 부분입니다.
    conv_11 = keras.layers.Conv2D(filters=192*2, kernel_size=(1, 1),
                                  activation='relu', padding='same')
    conv_12 = keras.layers.Conv2D(filters=384*2, kernel_size=(1, 3),
                                  activation='relu', padding='same')
    conv_13 = keras.layers.Conv2D(filters=384*2, kernel_size=(3, 1),
                                  activation='relu', padding='same')

    ## Pool, 그림에서 중간 오른쪽 부분입니다.
    avg_pool_21 = keras.layers.AveragePooling2D(pool_size=(3, 3), strides=1,
                                                padding='same')
    conv_22 = keras.layers.Conv2D(filters=128*2, kernel_size=(1, 1),
                                  activation='relu', padding='same')

    ## 1, 그림에서 제일 오른쪽 부분입니다.
    conv_31 = keras.layers.Conv2D(filters=384*2, kernel_size=(1, 1),
                                  activation='relu', padding='same')

    ## out
    concat = keras.layers.Concatenate(axis=-1)

    ## inception_module_c의 레이어를 연결해 주는 부분입니다.
    x0_0 = conv_04(conv_02(conv_01(x)))
    x0_1 = conv_03(conv_02(conv_01(x)))
    x1_0 = conv_13(conv_11(x))
    x1_1 = conv_12(conv_11(x))
    x2 = conv_22(avg_pool_21(x))
    x3 = conv_31(x)
    out = concat([x0_0, x0_1, x1_0, x1_1, x2, x3])

    return out


## 출력층 레이어 생성부분 입니다.
avg_pool = keras.layers.AveragePooling2D(pool_size=(7, 7))
dense_fc = keras.layers.Dense(1000)
dense_out = keras.layers.Dense(1000, activation='softmax')


## 전체 레이어를 연결해 주는 부분입니다.
input_layer = keras.layers.Input(shape=(8, 8, 1280))
imc_out_01 = inception_module_c(input_layer)
imc_out_02 = inception_module_c(imc_out_01)
```

```
output = dense_out(dense_fc(avg_pool(imc_out_02)))
```

In [ ]:

```
model = keras.models.Model(input_layer, output)
SVG(model_to_dot(model, show_shapes=True, dpi=50).create(prog='dot', format='svg'))
# plot_model(model, to_file='./imc_model.png', show_shapes=True)
```

In [ ]:

```

```