CNN(Convolution Neural Network) 추가 실습

학습 내용

- CNN의 추가 실습 해보기
  - adam 0.9919
  - rmsprop -> adam으로 변경 [] - [0.9876]
  - adam, 마지막 pooling 없애기 [0.9876] - [0.9901]
  - sigmoid, 마지막 pooling 없애기 [0.9901]- [0.1134]
  - leakyRelu 적용, 마지막 pooling 없애기 [0.9901]- [0.990]
  - Adamax, relu [L123전부 사용]- [0.9929]

In [1]:

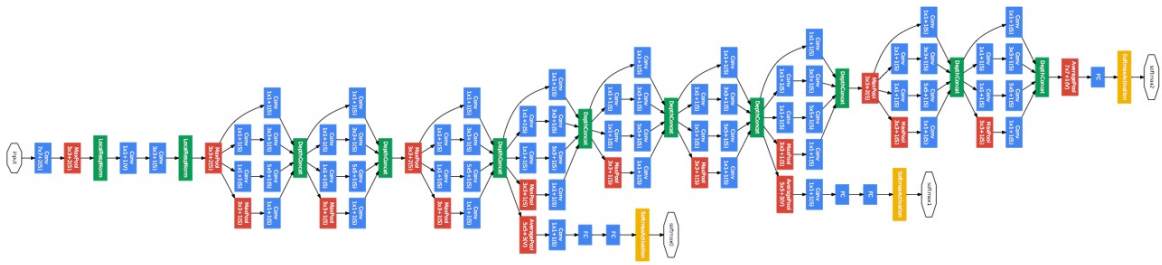
```
from IPython.display import display, Image
```

구글의 인셉션 모델

- 2014년 간신히 VGG 모델을 제치고 ImageNet에서 1등을 차지한 모델

In [2]:

```
display(Image(filename="img/google-inception.jpg"))
```



구글 인셉션 모델의 계층

- 작은 컨볼루션 계층을 매우 많이 연결.
- 구성이 꽤 복잡하다. 구현이 조금 까다로움.
  - 파란색-컨볼루션
  - 빨간색-풀링계층

01 데이터 가져오기

In [3]:

```
# 이미지 처리 분야에서 가장 유명한 신경망 모델인 CNN 을 이용
from keras import layers
from keras import models

from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

In [4]:

```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

02 신경망 모델 구성

- (28, 28, 1) 가로 세로 : 28 픽셀, RGB 단색
- (28, 28, 3) 가로 세로 : 28 픽셀, RGB 3색
- MNIST는 회색조의 이미지로 색상이 한개

In [5]:

```
# 기존 모델에서는 입력 값을 28x28 하나의 차원으로 구성하였으나,
# CNN 모델을 사용하기 위해 2차원 평면과 특성치의 형태를 갖는 구조로 만듭니다.
# None는 한번의 학습당 사용할 입력데이터의 개수,
# 마지막 차원 1은 특징의 개수. MNIST는 회색조의 이미지로 색상이 한개

model = models.Sequential()

### L1 계층
model.add(layers.Conv2D(32, (3, 3),
                        activation='relu',
                        padding="same",  #  기본값 : valid
                        input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

### L2 계층
model.add(layers.Conv2D(64, (3, 3),
                        activation='relu',
                        padding="same") )
model.add(layers.MaxPooling2D((2, 2)))

### L3 계층
model.add(layers.Conv2D(64, (3, 3),
                        activation='relu',
                       padding="same"))
model.add(layers.MaxPooling2D((2, 2)))
```

In [6]:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

In [7]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

비용함수, 최적화 함수

In [8]:

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 40s 42ms/step - loss: 0.4418 - accuracy:
0.8622
Epoch 2/5
938/938 [==============================] - 44s 47ms/step - loss: 0.0542 - accuracy:
0.98320s - loss:
Epoch 3/5
938/938 [==============================] - 42s 44ms/step - loss: 0.0337 - accuracy:
0.9898
Epoch 4/5
938/938 [==============================] - 39s 42ms/step - loss: 0.0286 - accuracy:
0.9915
Epoch 5/5
938/938 [==============================] - 49s 52ms/step - loss: 0.0204 - accuracy:
0.9936
```

Out[8]:

```
<tensorflow.python.keras.callbacks.History at 0x1cb2db58b20>
```

결과 확인

In [9]:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_acc)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0286 - accuracy: 0.
9912
0.9911999702453613
```

In [10]:

```
from keras.layers.advanced_activations import LeakyReLU
```

In [13]:

```
# 기존 모델에서는 입력 값을 28x28 하나의 차원으로 구성하였으나,
# CNN 모델을 사용하기 위해 2차원 평면과 특성치의 형태를 갖는 구조로 만듭니다.
# None는 한번의 학습당 사용할 입력데이터의 개수,
# 마지막 차원 1은 특징의 개수. MNIST는 회색조의 이미지로 색상이 한개

model = models.Sequential()

### L1 계층
### Convolution
model.add(layers.Conv2D(32, (3,3),
                        padding='same',  #  기본값 : valid
                        input_shape = (28,28,1) ))
model.add(LeakyReLU(0.2))
### Pooling - 다운 샘플링
model.add(layers.MaxPooling2D( (2,2) ))  # 2x2


### L2 계층
### Convolution 3x3, 64필터수, padding (같은 크기의 특징맵)
model.add(layers.Conv2D(64, (3,3),
                        padding='same')  )
model.add(LeakyReLU(0.2))
### Pooling - 다운 샘플링
model.add(layers.MaxPooling2D( (2,2) ))


### L3 계층
### Convolution 3x3, 64필터수, padding (같은 크기의 특징맵)
model.add(layers.Conv2D(64, (3,3),
                        padding='same')  )
model.add(LeakyReLU(0.2))

### Pooling - 다운 샘플링
# model.add(layers.MaxPooling2D( (2,2) ))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

In [14]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 7, 7, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                200768
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 257,162
Trainable params: 257,162
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
%%time

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 43s 45ms/step - loss: 0.3298 - accuracy:
0.8984
Epoch 2/5
938/938 [==============================] - 44s 46ms/step - loss: 0.0444 - accuracy:
0.9863
Epoch 3/5
938/938 [==============================] - 42s 44ms/step - loss: 0.0308 - accuracy:
0.9906
Epoch 4/5
938/938 [==============================] - 44s 47ms/step - loss: 0.0233 - accuracy:
0.9929
Epoch 5/5
938/938 [==============================] - 48s 52ms/step - loss: 0.0165 - accuracy:
0.9948
Wall time: 3min 40s
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x1cb2f1c2370>
```

In [16]:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_acc)
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0403 - accuracy: 0.
9879
0.9879000186920166
```

REF

- 컨볼루션 참고 :
  - https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59 (https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59)
  - https://mlblr.com/includes/research/index.html (https://mlblr.com/includes/research/index.html) : Convolution 이해, 다양한 논문
  - http://taewan.kim/post/cnn/#1-cnn%EC%9D%98-%EC%A3%BC%EC%9A%94-%EC%9A%A9%EC%96%B4-%EC%A0%95%EB%A6%AC (http://taewan.kim/post/cnn/#1-cnn%EC%9D%98-%EC%A3%BC%EC%9A%94-%EC%9A%A9%EC%96%B4-%EC%A0%95%EB%A6%AC)
- 인셉션 관련 : https://sunghan-kim.github.io/ml/3min-dl-ch11/# (https://sunghan-kim.github.io/ml/3min-dl-ch11/)

In [ ]: