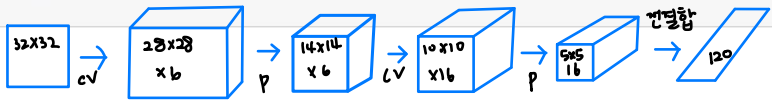
기본 합성곱 신경망 구현

입력층 → 컨볼루션층 → 풀링층 → 전결합 은닉층 → 전결합 출력층

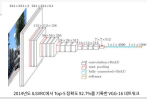
In [1]:
```
import tensorflow as tf
import numpy as np
```

하이퍼 파라미터



In [2]:
```
EPOCHS = 10
```

모델 정의 (VGG-16)



In [3]:
```
class ConvNet(tf.keras.Model):
    def __init__(self):
        super(ConvNet, self).__init__()
        conv2d = tf.keras.layers.Conv2D
        maxpool = tf.keras.layers.MaxPool2D
        self.sequence = list()
        self.sequence.append(conv2d(16, (3, 3), padding='same', activation='relu')) # 28x28x16
        self.sequence.append(conv2d(16, (3, 3), padding='same', activation='relu')) # 28x28x16
        self.sequence.append(maxpool((2,2))) # 14x14x16
        self.sequence.append(conv2d(32, (3, 3), padding='same', activation='relu')) # 14x14x32
        self.sequence.append(conv2d(32, (3, 3), padding='same', activation='relu')) # 14x14x32
        self.sequence.append(maxpool((2,2))) # 7x7x32
        self.sequence.append(conv2d(64, (3, 3), padding='same', activation='relu')) # 7x7x64
        self.sequence.append(conv2d(64, (3, 3), padding='same', activation='relu')) # 7x7x64
        self.sequence.append(tf.keras.layers.Flatten()) # 1568
        self.sequence.append(tf.keras.layers.Dense(128, activation='relu'))
        self.sequence.append(tf.keras.layers.Dense(10, activation='softmax'))

    def call(self, x, training=False, mask=None):
        for layer in self.sequence:
            x = layer(x)
        return x
```

① 출력 채널 수. 입력 채널 수는 적지 않음.
② 사용될 커널 크기
③ padding='same' → 동일한 크기 유지

학습, 테스트 루프 정의

In [4]:
```
# Implement training loop
@tf.function
def train_step(model, images, labels, loss_object, optimizer, train_loss, train_accuracy):
    with tf.GradientTape() as tape:
        predictions = model(images)
        loss = loss_object(labels, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)

    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    train_loss(loss)
    train_accuracy(labels, predictions)

# Implement algorithm test
@tf.function
def test_step(model, images, labels, loss_object, test_loss, test_accuracy):
    predictions = model(images)

    t_loss = loss_object(labels, predictions)
    test_loss(t_loss)
    test_accuracy(labels, predictions)
```

데이터셋 준비

In [5]:
```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# x_train : (NUM_SAMPLE, 28, 28) -> (NUM_SAMPLE, 28, 28, 1)
x_train = x_train[..., tf.newaxis].astype(np.float32)
x_test = x_test[..., tf.newaxis].astype(np.float32)

train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(10000).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

0~255 → 0~1 (정규화)
배치수 28x28
채널 추가
epoch

학습 환경 정의

모델 생성, 손실함수, 최적화 알고리즘, 평가지표 정의

In [6]:
```
# Create model
model = ConvNet()

# Define loss and optimizer
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()

# Define performance metrics
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

학습 루프 동작

In [7]:
```
for epoch in range(EPOCHS):
    for images, labels in train_ds:
        train_step(model, images, labels, loss_object, optimizer, train_loss, train_accuracy)

    for test_images, test_labels in test_ds:
        test_step(model, test_images, test_labels, loss_object, test_loss, test_accuracy)

    template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {}'
    print(template.format(epoch + 1,
                          train_loss.result(),
                          train_accuracy.result() * 100,
                          test_loss.result(),
                          test_accuracy.result() * 100))
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()
```

```
Epoch 1, Loss: 0.11901089549064636, Accuracy: 96.29833221435547, Test Loss: 0.04793301969766617, Test Accuracy: 98.37999725341797
Epoch 2, Loss: 0.08064260333776474, Accuracy: 97.5041656494141, Test Loss: 0.04420175403356552, Test Accuracy: 98.56999969482422
Epoch 3, Loss: 0.06300631165504456, Accuracy: 98.0455551147461, Test Loss: 0.04434952512383461, Test Accuracy: 98.60333251953125
Epoch 4, Loss: 0.05299808084964752, Accuracy: 98.35166931152344, Test Loss: 0.04470251500606537, Test Accuracy: 98.63249969482422
Epoch 5, Loss: 0.04613211750984192, Accuracy: 98.55500030517578, Test Loss: 0.04394772648811340, Test Accuracy: 98.6760025024414
Epoch 6, Loss: 0.04120549187064171, Accuracy: 98.69944763183594, Test Loss: 0.04168213158845901, Test Accuracy: 98.7550048828125
Epoch 7, Loss: 0.03739384934306145, Accuracy: 98.81832885742188, Test Loss: 0.04067239165306091, Test Accuracy: 98.78571319580078
Epoch 8, Loss: 0.03435363620519638, Accuracy: 98.91478729248047, Test Loss: 0.03910746797919273, Test Accuracy: 98.84874725341797
Epoch 9, Loss: 0.03191001340746879, Accuracy: 98.99130249023438, Test Loss: 0.03902928531169891, Test Accuracy: 98.87333679199219
Epoch 10, Loss: 0.0297947209328413, Accuracy: 99.05616760253906, Test Loss: 0.04044229909777641, Test Accuracy: 98.8740005493164
```