Deep Learning Image Processing

Case Study: LeNet-5 & AlexNet

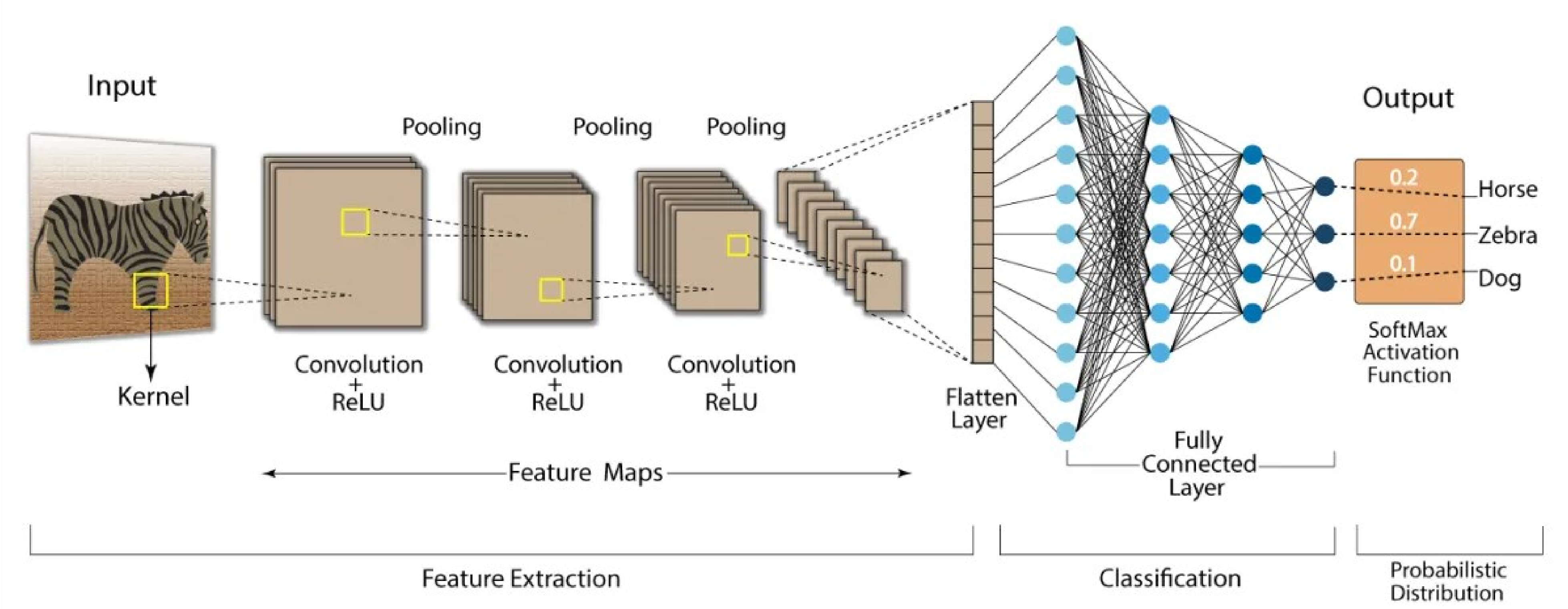
2023-1

Young-Keun Kim
School of Mechanical & Control Engineering

Overview

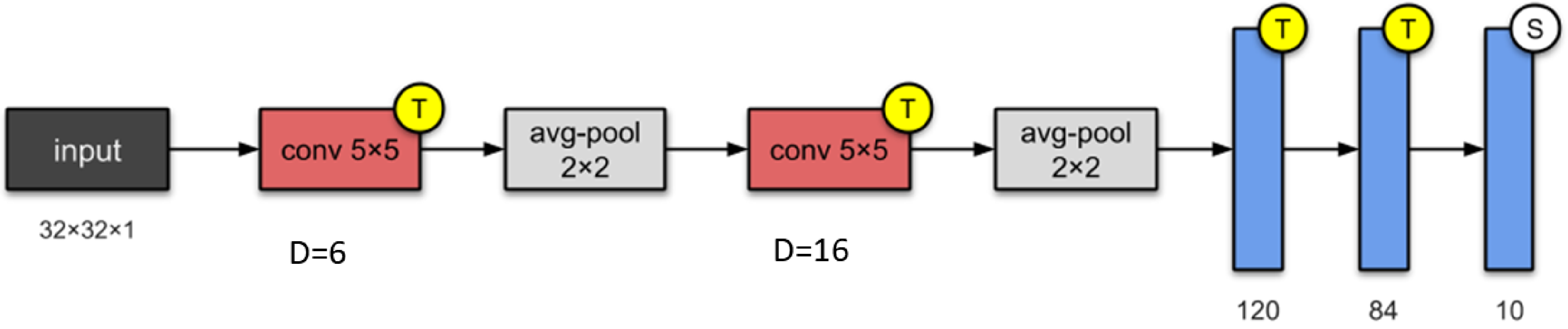
● CNN architecture

- For Classification Prediction of an image
  - Feature Extraction - Classification - Probability Distribution
- Basic CNN architecture components
  - convolutional layer, activation function, pooling layer, fully-connected layer



LeNet-5

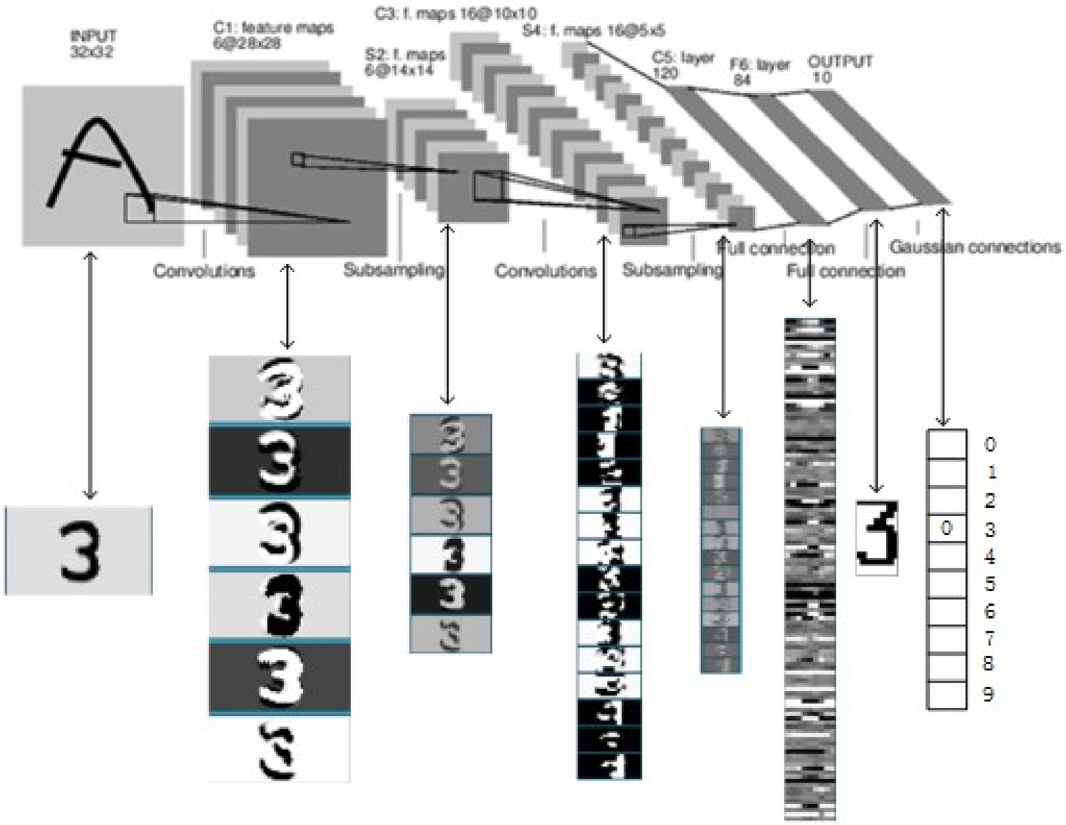
- **Case study of LeNet-5**
  - LeCun, Yann et al., Gradient-based learning applied to document recognition, 1998
    - For handwriting recognition, MNIST
  - A standard template of CNN
  - 2 CONV layers + 3 F.C layers
  - About 60,000 parameters



[CNN architecture of LeNet-5]

LeNet-5

- **Case study of LeNet-5**
  - A standard template of CNN
  - Activation Function: TanH
    - Now, ReLU is often used
  - Pooling: Avg. pooling
    - Now, MaxPooling is often used
  - No Padding
  - F.C: softmax
    - Originally used RBF(Radial Basis Function)
  - Loss Function: MSE
  - Input: 32x32x1
    - MNIST image is 28x28
    - MNIST is padded to 32



[CNN architecture of LeNet-5]

Case study : LeNet-5

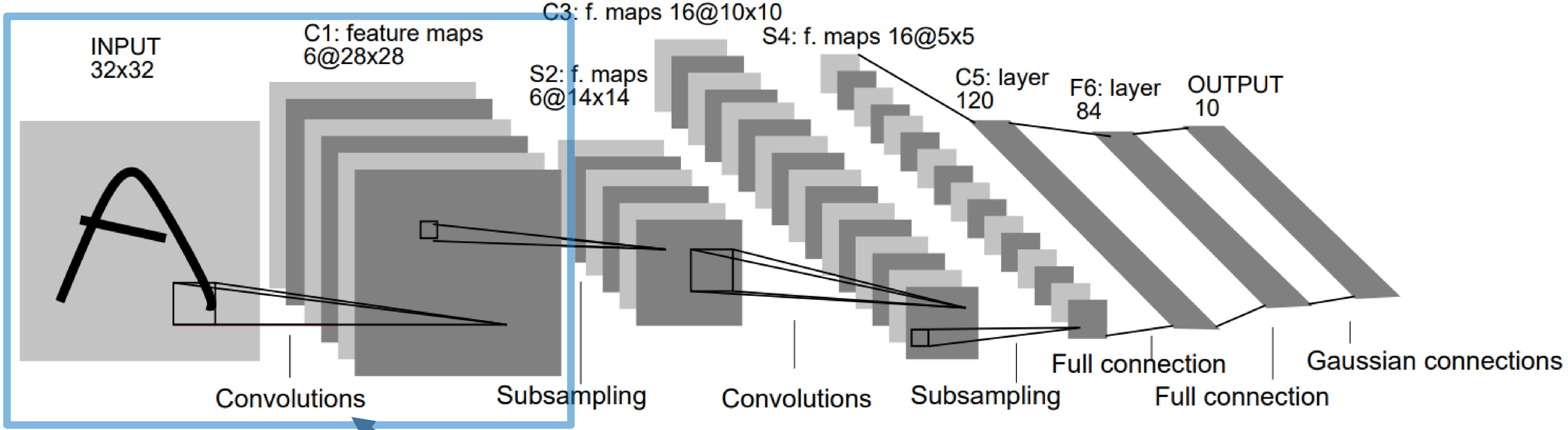


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

without zero-padding & stride=1

*If zero-padding->(5*5+1)*32*32*6*

Step	No. of free parameters	No. of connections
Input→C1 : 6 of convolution filter kernel (5x5x1)	156=(5*5+1(bias))*6	122,304=(5*5+1)*28*28*6

**MLP: (32*32+1)*28*28*6=4,821,600* **MLP: (32*32+1)*28*28*6=4,821,600*

Case study : LeNet-5

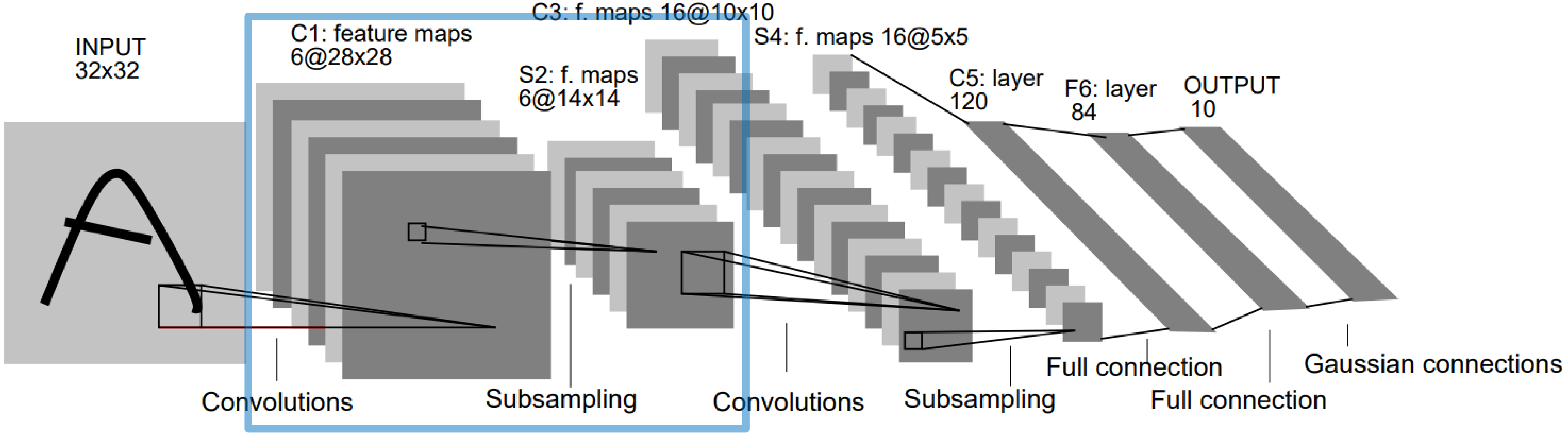
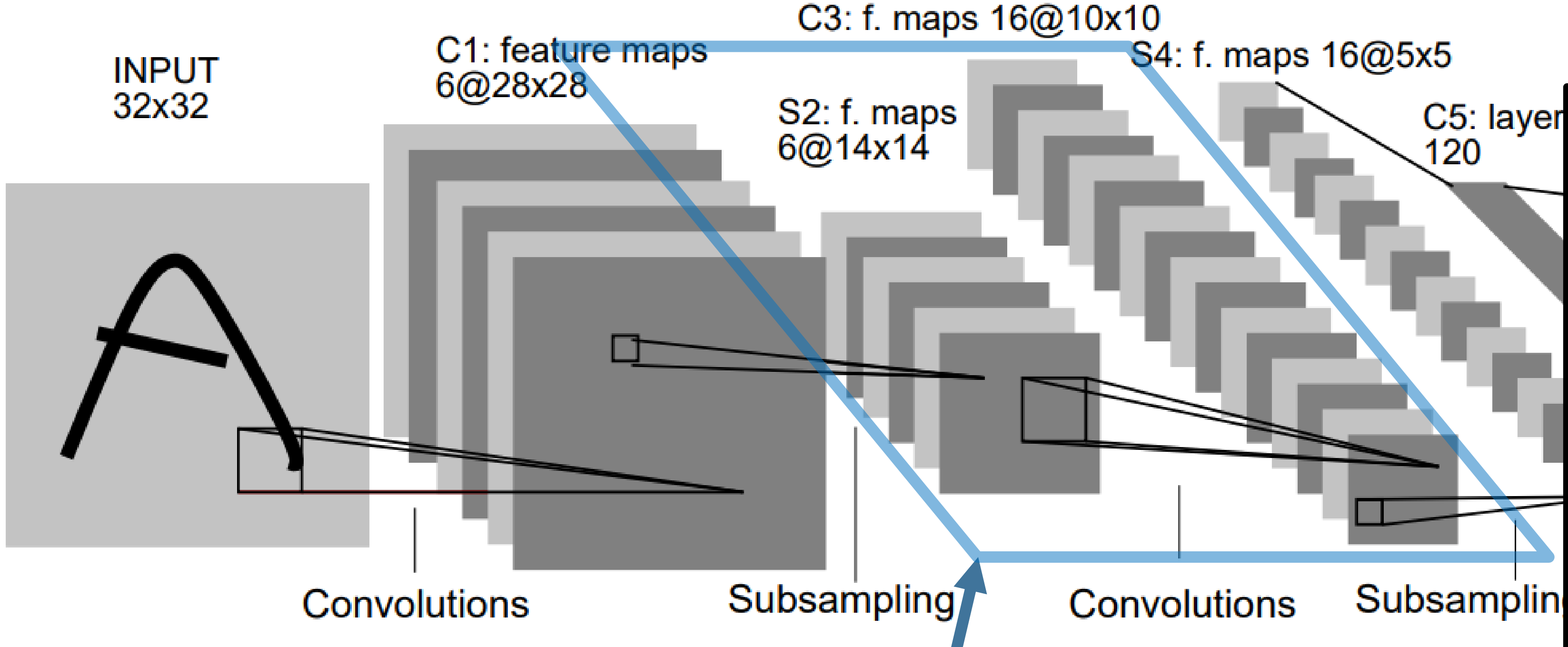


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Step	No. of free parameters	No. of connections
C1→S2 : average pooling (2x2), stride 2	12=(1(weight)+1(bias))*6 (* usually, pooling has no weight)	5880=(2*2+1)*14*14*6

Case study : LeNet-5



To give some variation in convolution, disconnect some links

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

without zero-padding

Step	No. of free parameters	No. of connections
C2→S3 : convolution kernel (5x5)	1516=5*5*60+16	151600=(5*5*60+16)*10*10

Case study : LeNet-5

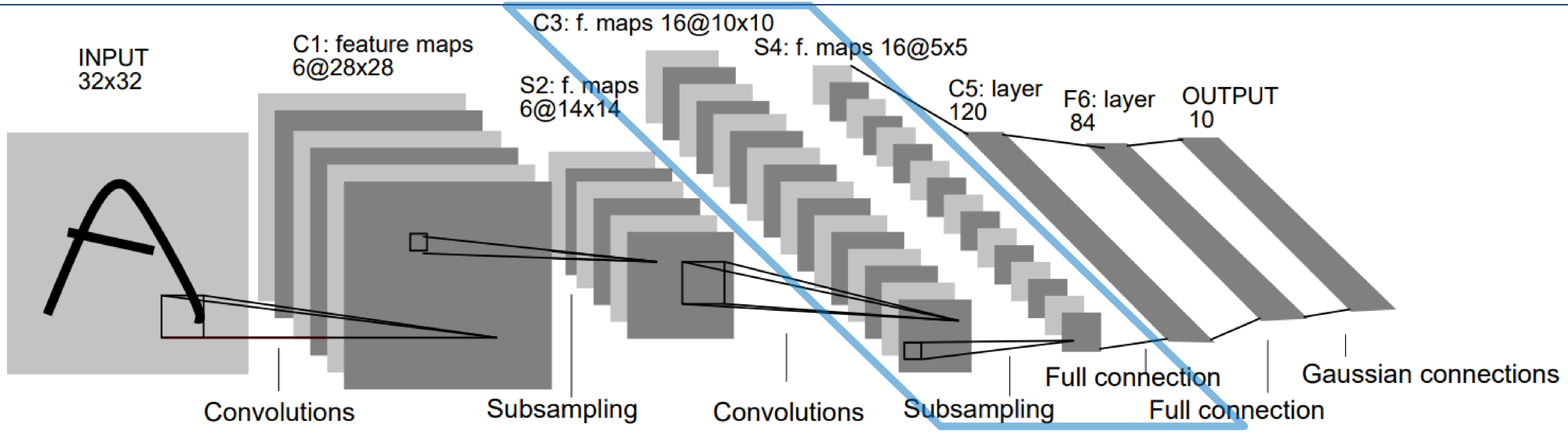


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Step	No. of free parameters	No. of connections
C3→S4 : average pooling (2x2)	32=(1(weight)+1(bias))*16	2000=(2*2+1)*5*5*16

Case study : LeNet-5

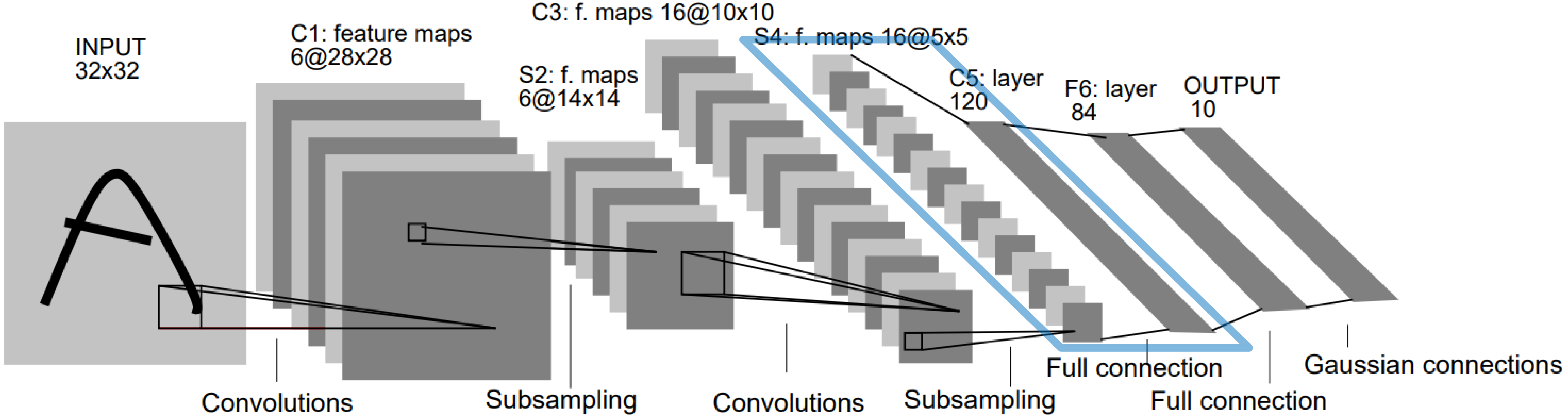


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Step	No. of free parameters	No. of connections
S4→C5 : convolution kernel (5x5)	48,120=5*5*16*120+120(bias)	Same as 'No. of free parameters'

Case study : LeNet-5

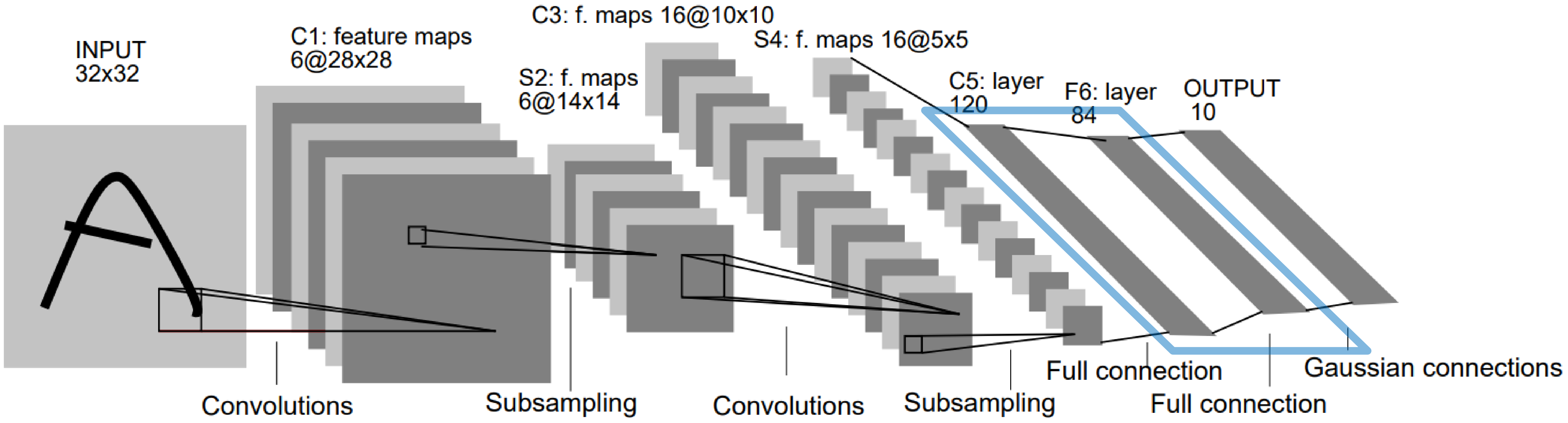


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Step	No. of free parameter	No. of connection
C5→F6 : fully connected	10,164=(120+1)*84	Same as 'No. of free parameters'

Case study : LeNet-5

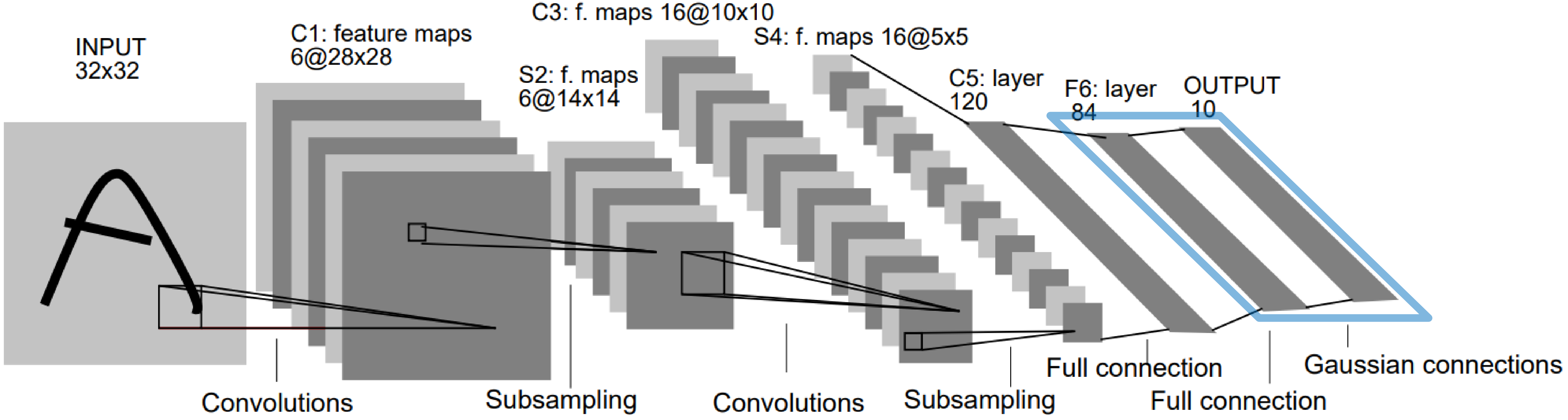


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Step	No. of free parameter	No. of connection
F6→OUTPUT : Gaussian connected	850=(84+1)*10	Same as 'No. of free parameters'

Case study : LeNet-5

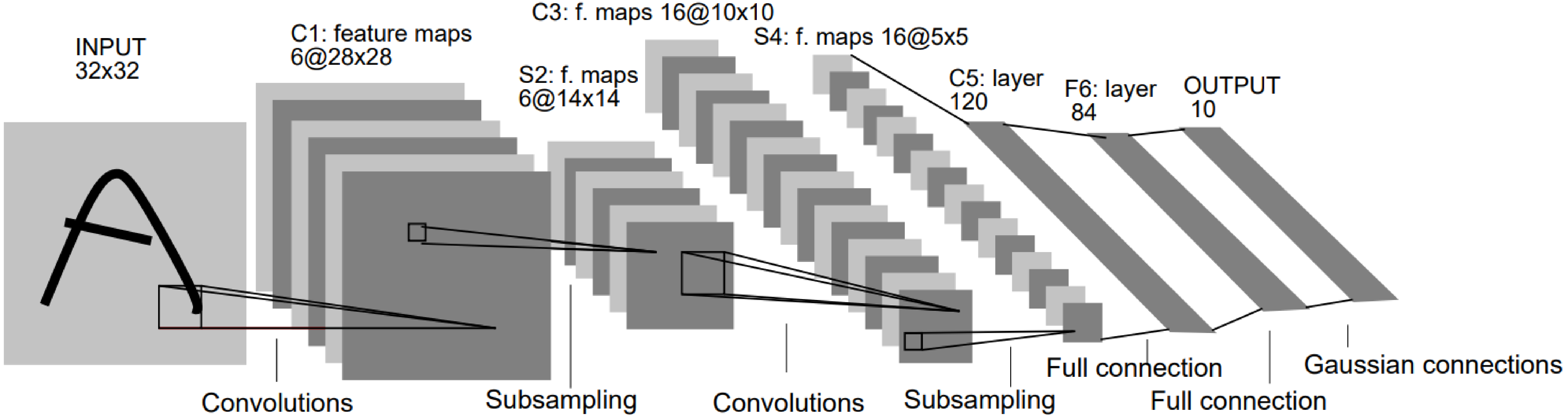


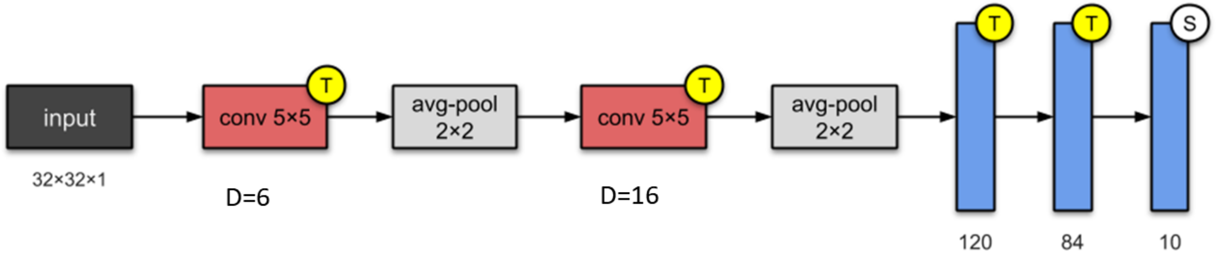
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

To train one image of 32*32*1

Total No. of free parameter	Total No. of connection (per image)
60,850	340,918

LeNet-5: Pytorch

- Pytorch implementation
  - Go to Tutorial_LeNet for full description

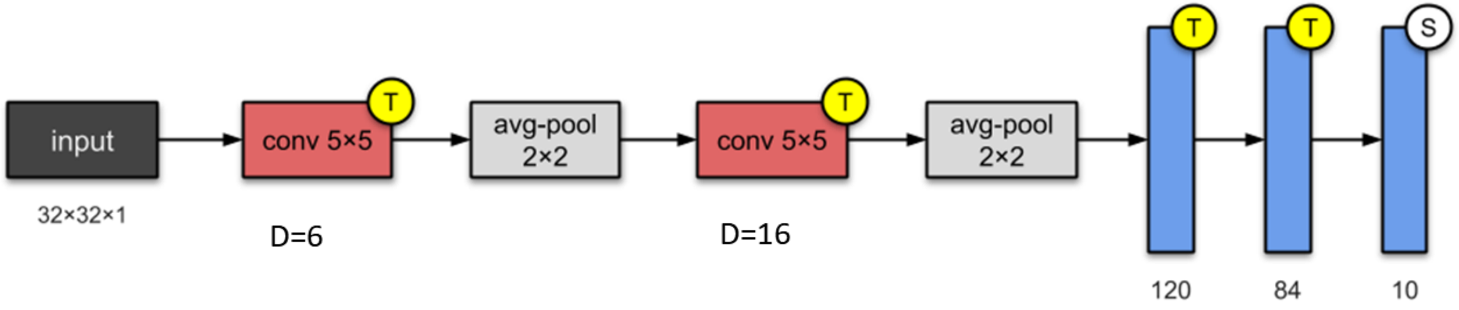



```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 1 input channel, 6 output channels, 5x5 convolution kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        # 6*6 from image dimension
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```python
def num_flat_features(self, x):
    size = x.size()[1:]
    # all dimensions except the batch dimension
    num_features = 1
    for s in size:
        num_features *= s
    return num_features
```

LeNet-5: Keras

- Keras implementation
  - Go to Tutorial_LeNet for full description




```
model = keras.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(units=10, activation = 'softmax'))
```

LeNet-5

Tutorial

- Implement LeNet-5 in PyTorch

Download module: My_DLIP.py

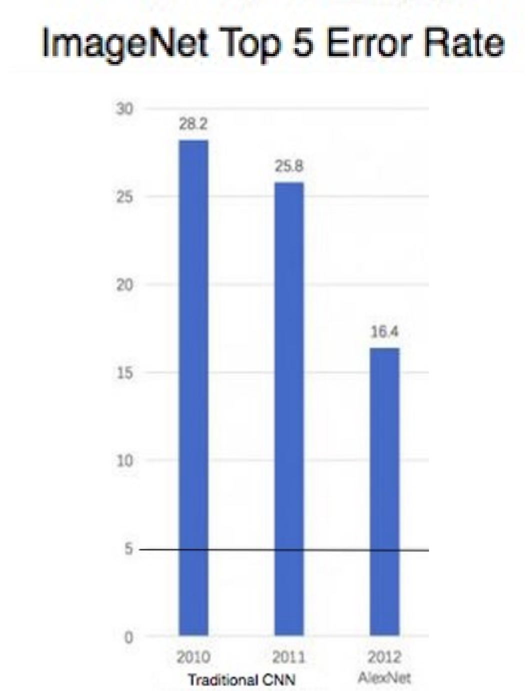
- T2-1: Create LeNeT CNN model and Train with opendataset (CIFAR10)
- T2-2: Test with loading trained model(LeNet-5)

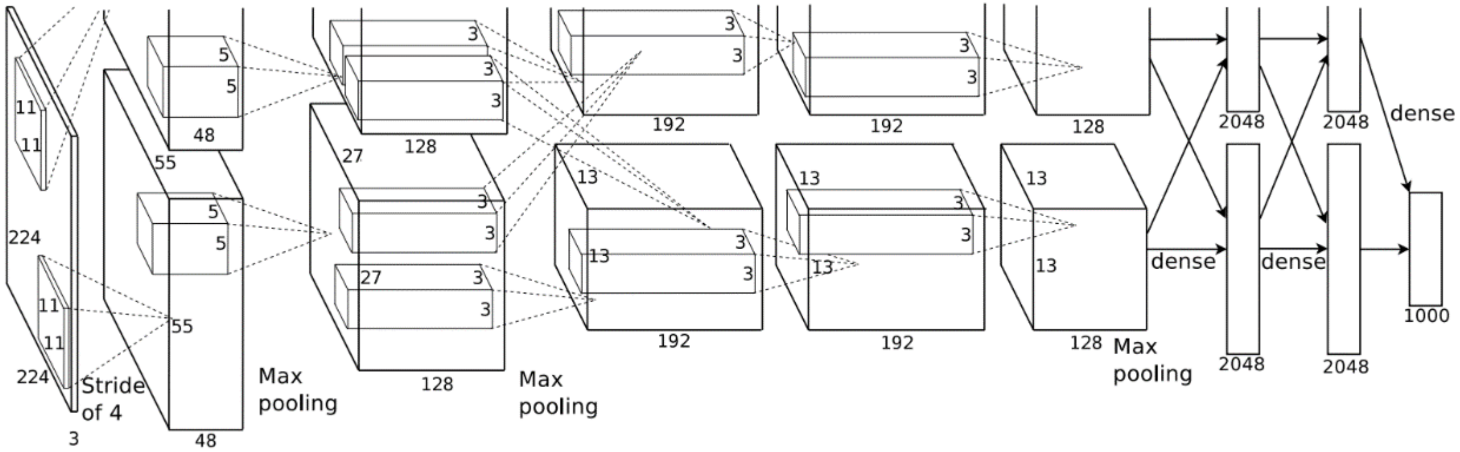
Deep Learning Image Processing

Case Study: AlexNet

Case study : AlexNet

- **AlexNet**
  - Krizhevsky, Sutskever, Hinton, “Imagenet classification with deep convolutional neural networks”. NIPS 2012
  - Winner of ILSVRC 2012
    - Top-5 test error of 15.4% (2nd place was 26.2%)
    - A breakthrough in CNN for high classification prediction

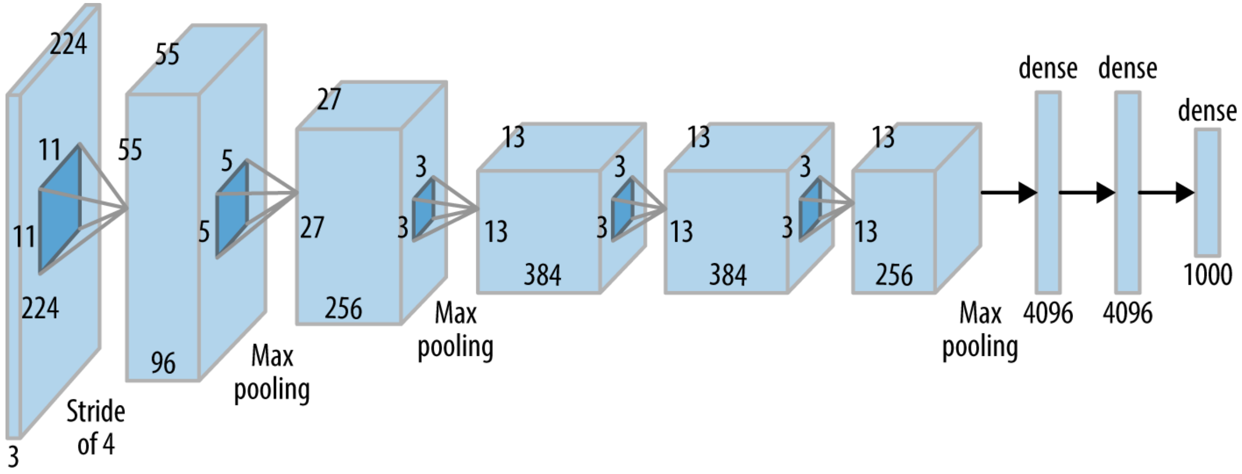




Case study : AlexNet

- **AlexNet features**
  - Multiple GPU
  - **ReLU** is introduced
    - instead of TanH
  - Local Response Normalization
    - Nowadays use batch normalization
  - Overlapping Pooling
  - Dropout
  - Data augmentation

Input Layer : 224 Width x 224 Height x 3 Channel Image
Convolution 1 : 96 of 11x11x3 Filter + LRN + Pooling
Convolution 2 : 256 of 5x5x48 Filter + LRN + Pooling
Convolution 3 : 384 of 3x3x128 Filter
Convolution 4 : 384 of 3x3x192 Filter
Convolution 5 : 256 of 3x3x192 Filter + Pooling
Fully-Connected 1 : 4096
Fully-Connected 2 : 4096
Output Layer : 1000 Class

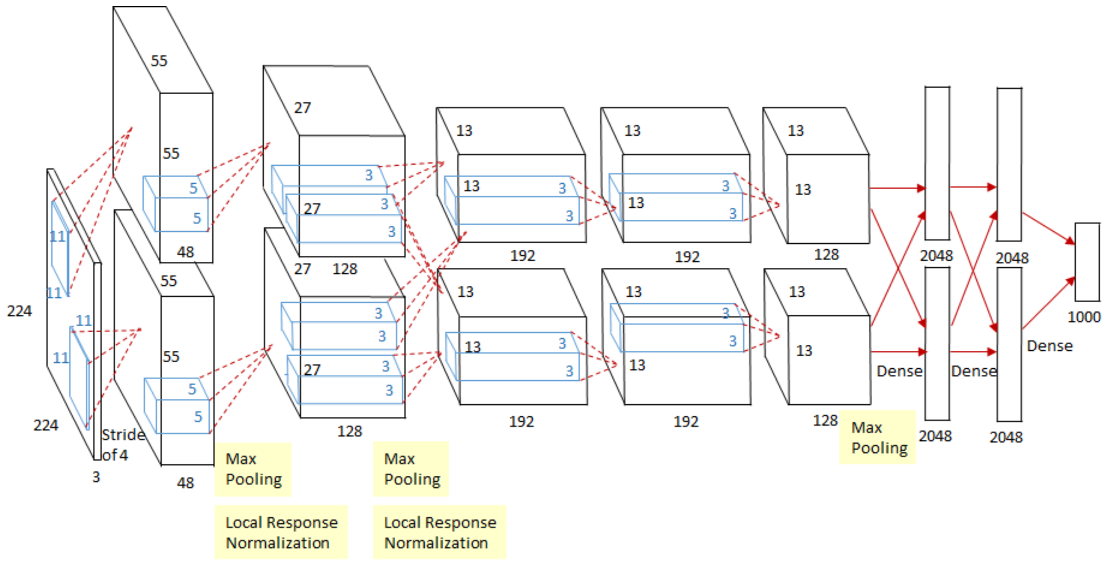


Group convolution is applied. From 2nd layer, number of kernels are divided by 2 for each group. e.g.
256 of 5x5x48 -->
(128 of 5x5x48) *2

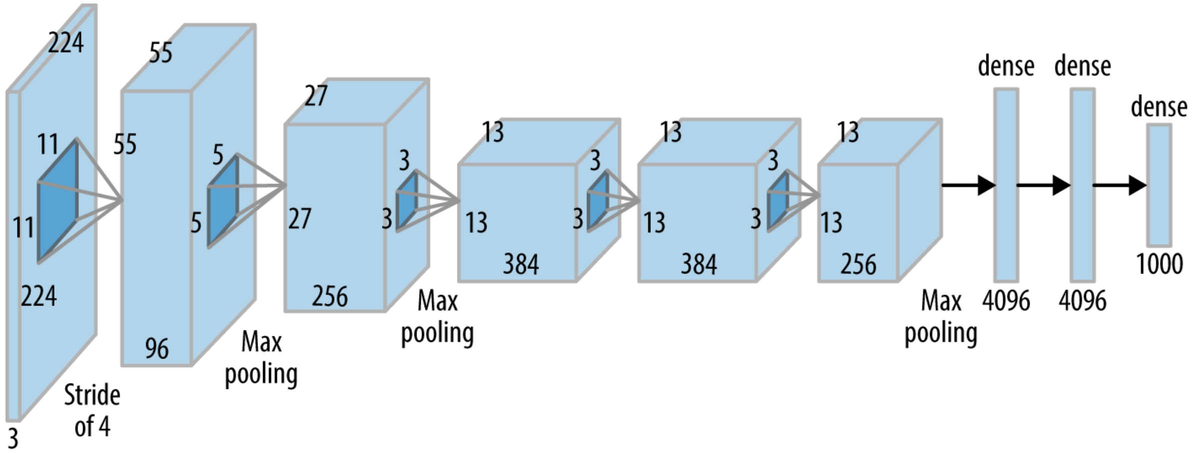
Case study : AlexNet

- **AlexNet training**
  - Dataset
    - ImageNet of 256x256x3
  - Input: 224×224×3 image
  - Data augmentation
  - Initial
    - Bias: 1 for 2,4,5th conv. 0 for others
  - SGD
    - Batch size: 128
    - Momentum v: 0.9
    - Weight Decay: 0.0005
    - Learning rate : 0.01

Parallel GPU Processings



1 GPU



Case study : AlexNet

- **Pytorch implementation**
  - Class Alexnet

```
class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
            nn.ReLU(inplace=True),
            nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, padding=2, groups=2),
            nn.ReLU(inplace=True),
            nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1, groups=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1, groups=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
```

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

```
def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), 256 * 6 * 6)
    x = self.classifier(x)
    return x
```

Case study : AlexNet

- **Keras implementation**
  - Class Alexnet: see class website

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

```python
model = keras.Sequential()
#1st Convolutional Layer
model.add(Conv2D(filters=96, input_shape=(32,32,3), kernel_size=(11,11), strides=(4,4), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

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

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

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

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

```python
#Passing it to a Fully Connected layer
model.add(Flatten())
# 1st Fully Connected Layer
model.add(Dense(4096, input_shape=(32,32,3,)))
model.add(BatchNormalization())
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))

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

#3rd Fully Connected Layer
model.add(Dense(1000))
model.add(BatchNormalization())
model.add(Activation('relu'))
#Add Dropout
model.add(Dropout(0.4))

#Output Layer
model.add(Dense(10))
model.add(BatchNormalization())
model.add(Activation('softmax'))

#Model Summary
model.summary()
```