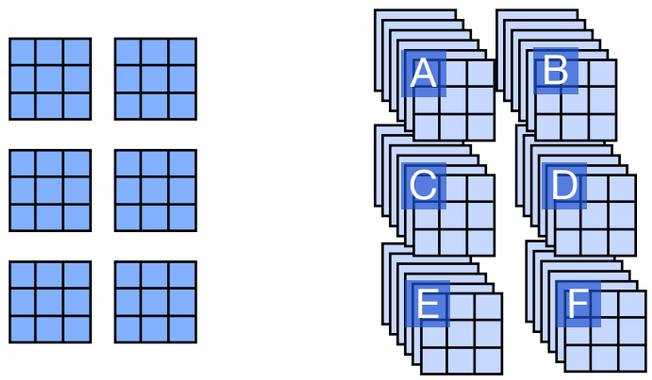
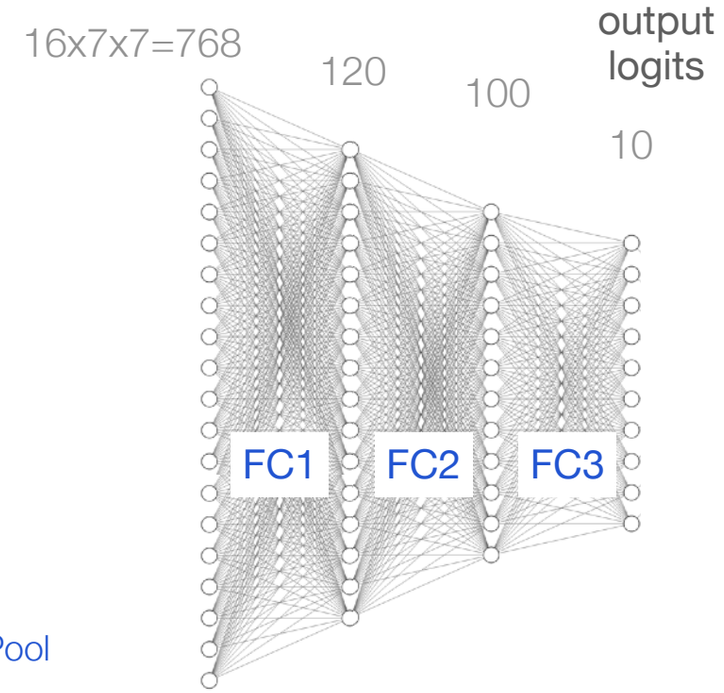
Exploring **Features Learned** from **Convolutional Networks**

Outline

- A **simple CNN** trained on fashion-MNIST dataset
- **Convolutional filters** learned from CNNs
- **Feature Vectors** learned in the **Last Layer** | **UMAP** visualization
- Transfer Learning

A simple ConV Net





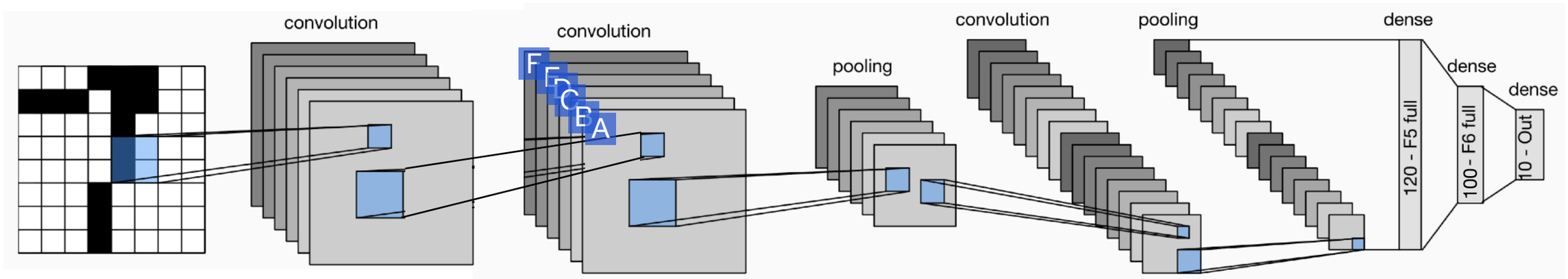
ConV1
3x3

ConV2
3x3

MaxPool

ConV3
3x3

MaxPool



1@28x28 6@28x28 6@28x28 6@14x14 16@14x14 16@7x7 120 100 10

ConV Net : feature extractor

MLP : classifier

Architecture

ConV1 → ReLU → ConV2 → ReLU → Pool → ConV3 → ReLU → Pool → Flatten → FC1 → ReLU → FC2 → ReLU → FC3 → SoftMax

Fashion-MNIST

0 : T-shirt

1 : Trouser

2 : Pullover

3 : Dress

4 : Coat

5 : Sandal

6 : Shirt

7 : Sneaker

8 : Bag

9 : Ankle boot

Image size : 28x28

Training Set : 60,000
Testing Set : 10,000

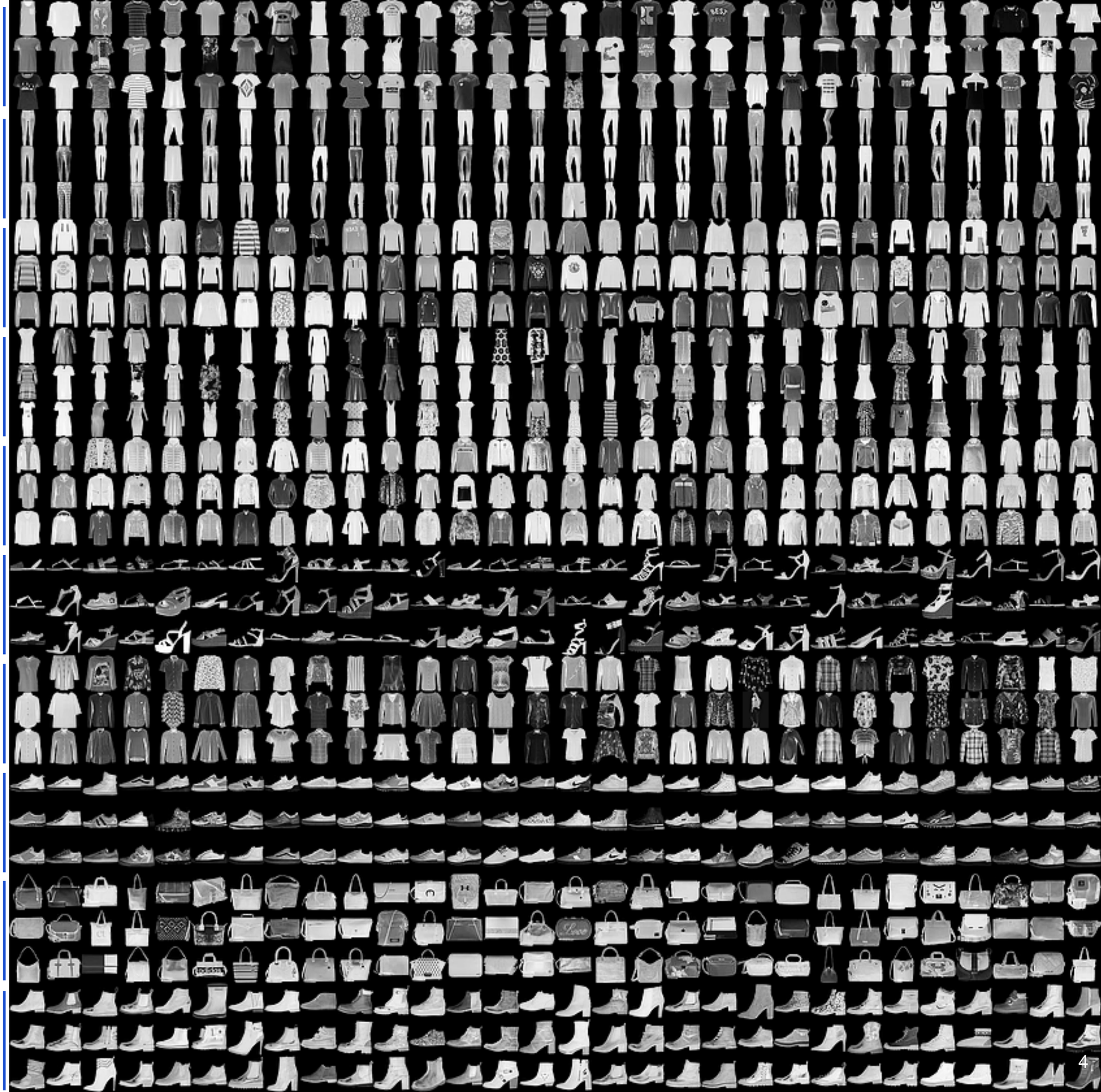
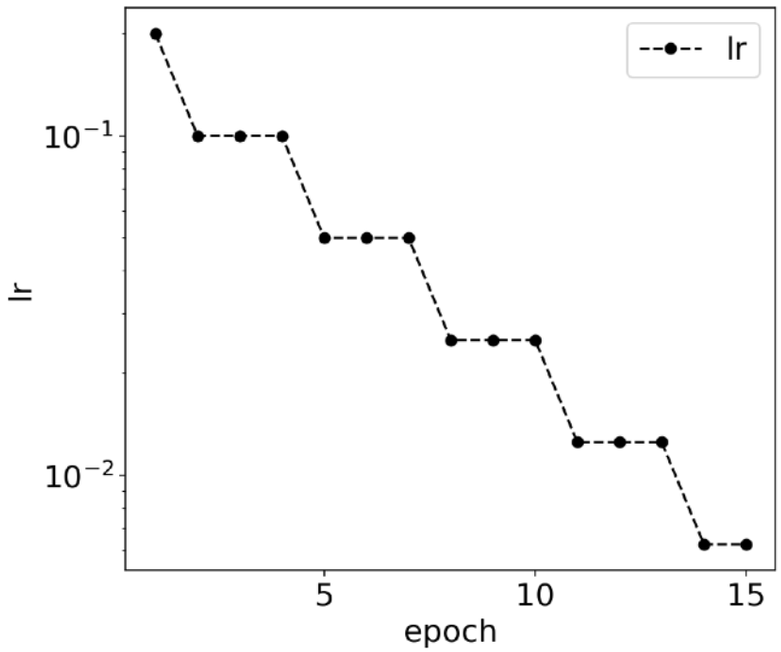
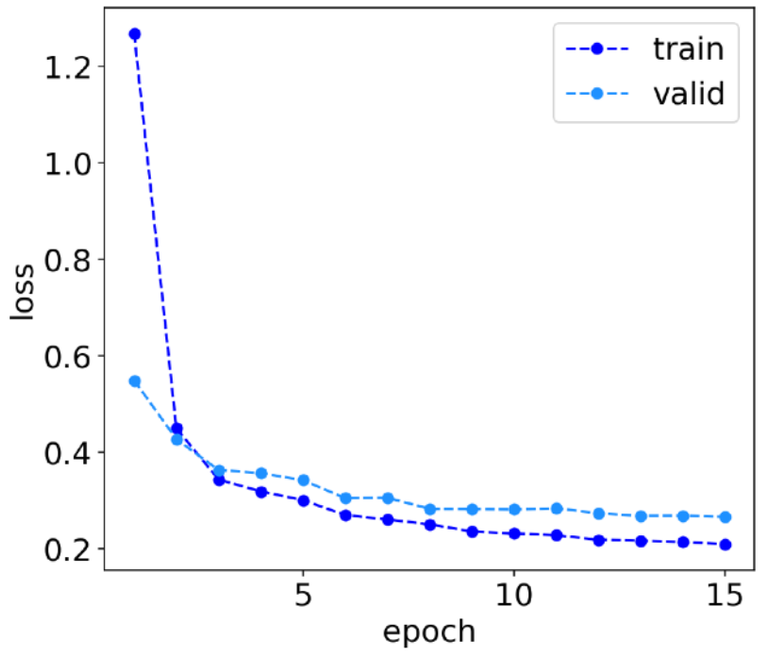


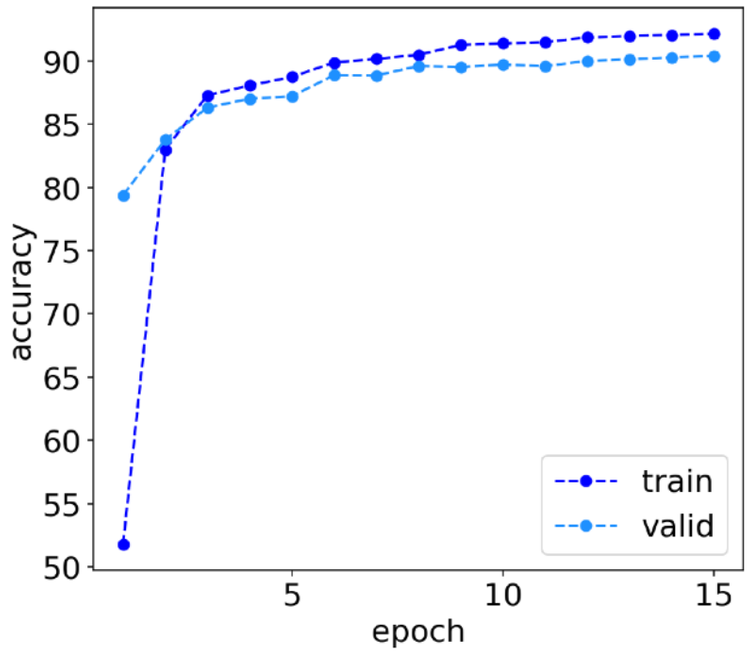
Image Credit :
https://github.com/zalandoresearch/fashion-mnist

Training

```python
# Optimizer
learning_rate = 0.2
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[2, 5, 8, 11, 14], gamma=0.5)
```

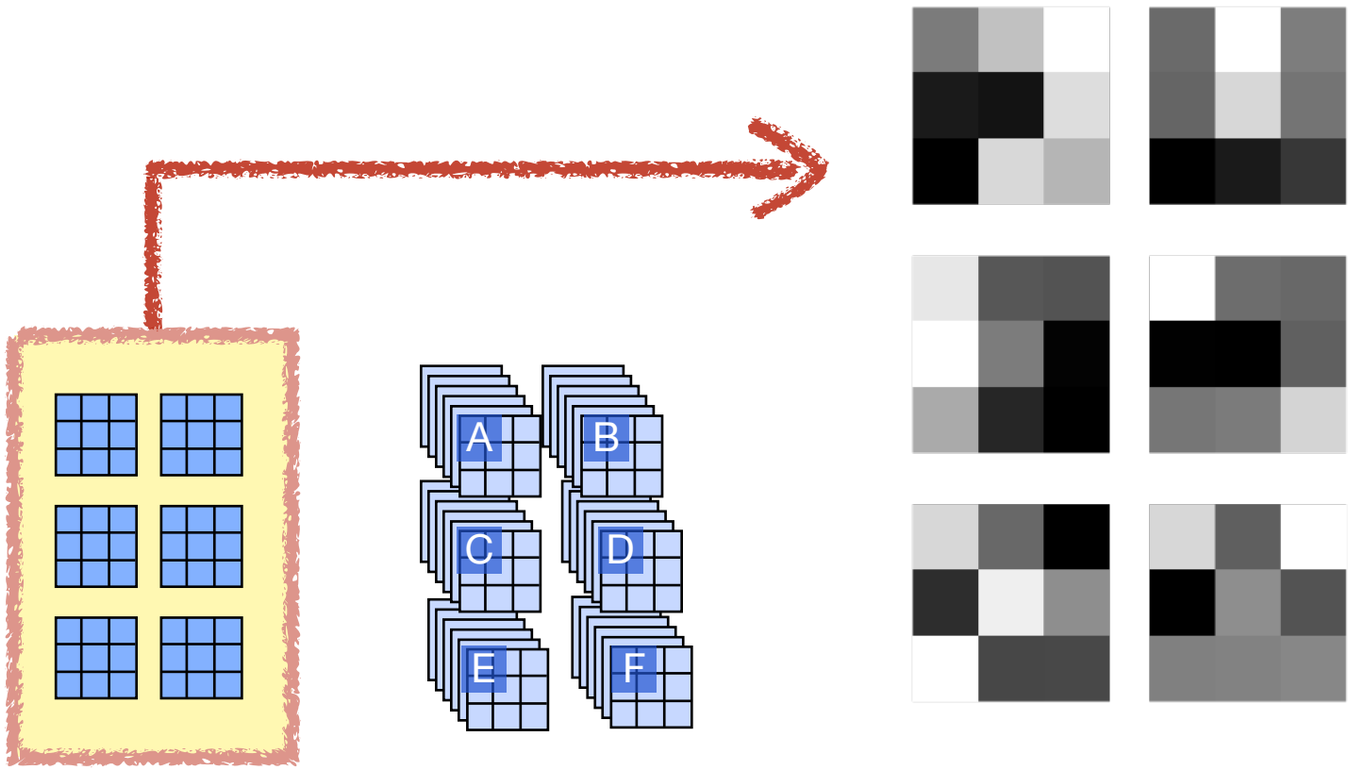


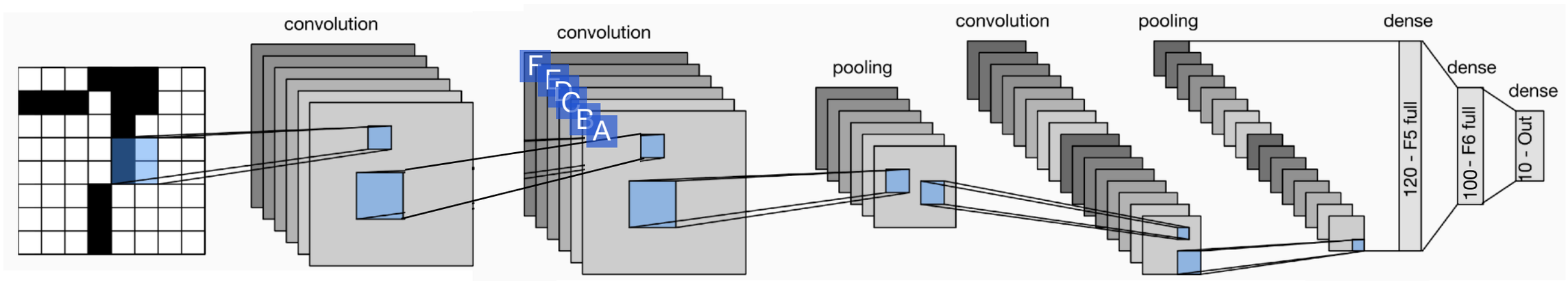





```
>>> End epoch 1, train accuracy = 51.77%, train avgLoss = 1.2684
>>>              valid accuracy = 79.38%, valid avgLoss = 0.5493
>>> End epoch 2, train accuracy = 83.00%, train avgLoss = 0.4497
>>>              valid accuracy = 83.78%, valid avgLoss = 0.4274
>>> End epoch 3, train accuracy = 87.29%, train avgLoss = 0.3438
>>>              valid accuracy = 86.31%, valid avgLoss = 0.3634
>>> End epoch 4, train accuracy = 88.06%, train avgLoss = 0.3190
>>>              valid accuracy = 87.02%, valid avgLoss = 0.3569
>>> End epoch 5, train accuracy = 88.74%, train avgLoss = 0.3009
>>>              valid accuracy = 87.21%, valid avgLoss = 0.3424
>>> End epoch 6, train accuracy = 89.88%, train avgLoss = 0.2700
>>>              valid accuracy = 88.87%, valid avgLoss = 0.3050
>>> End epoch 7, train accuracy = 90.16%, train avgLoss = 0.2609
>>>              valid accuracy = 88.85%, valid avgLoss = 0.3059
>>> End epoch 8, train accuracy = 90.49%, train avgLoss = 0.2507
>>>              valid accuracy = 89.60%, valid avgLoss = 0.2830
>>> End epoch 9, train accuracy = 91.26%, train avgLoss = 0.2361
>>>              valid accuracy = 89.53%, valid avgLoss = 0.2827
>>> End epoch 10, train accuracy = 91.39%, train avgLoss = 0.2317
>>>              valid accuracy = 89.70%, valid avgLoss = 0.2818
>>> End epoch 11, train accuracy = 91.48%, train avgLoss = 0.2283
>>>              valid accuracy = 89.59%, valid avgLoss = 0.2839
>>> End epoch 12, train accuracy = 91.86%, train avgLoss = 0.2188
>>>              valid accuracy = 90.00%, valid avgLoss = 0.2739
>>> End epoch 13, train accuracy = 91.97%, train avgLoss = 0.2167
>>>              valid accuracy = 90.13%, valid avgLoss = 0.2684
>>> End epoch 14, train accuracy = 92.05%, train avgLoss = 0.2143
>>>              valid accuracy = 90.28%, valid avgLoss = 0.2687
>>> End epoch 15, train accuracy = 92.17%, train avgLoss = 0.2093
>>>              valid accuracy = 90.39%, valid avgLoss = 0.2662
```

1st layer convolutional filters





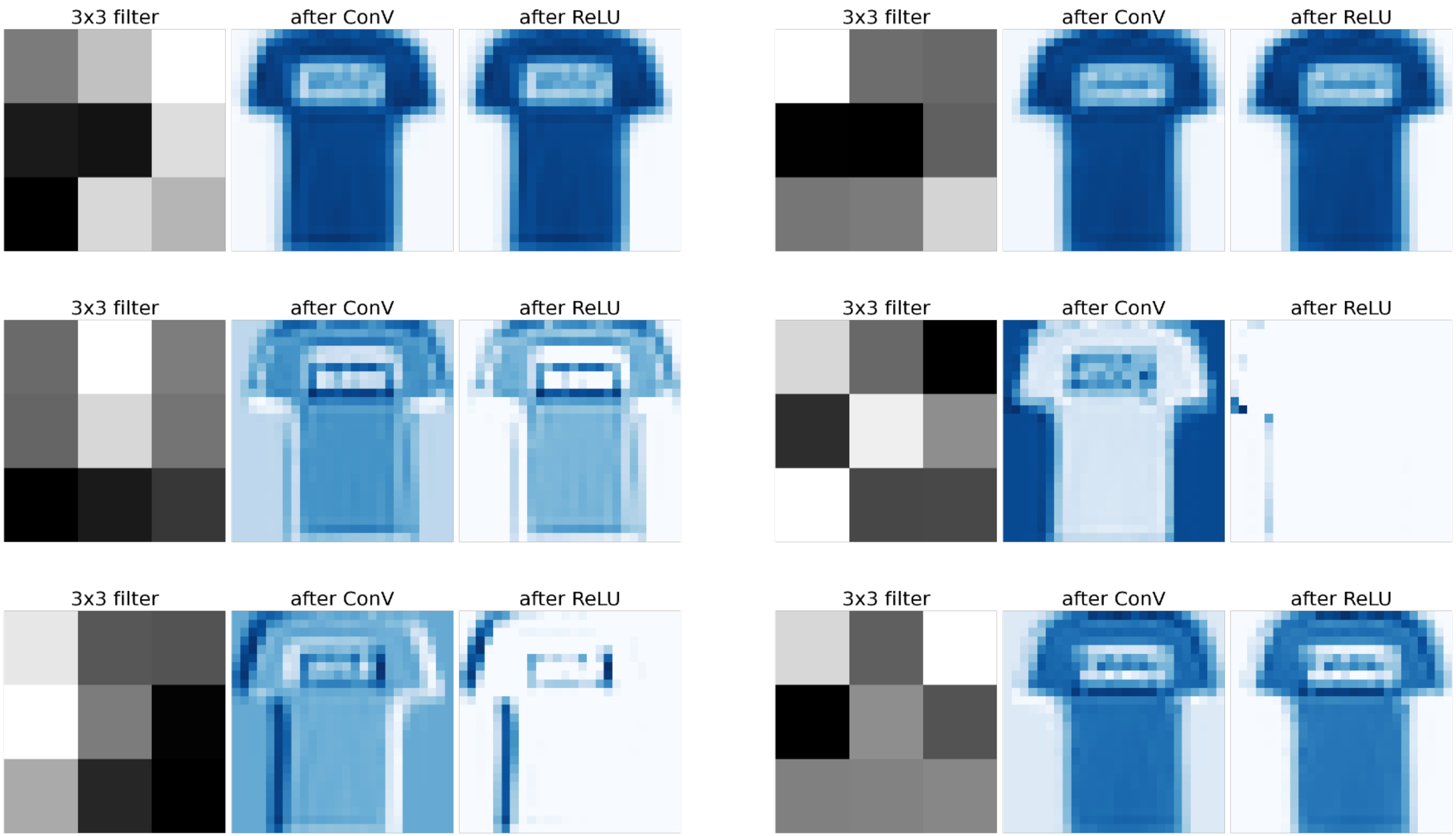
ConV1 3x3 | ConV2 3x3 | MaxPool | ConV3 3x3 | MaxPool

1@28x28 | 6@28x28 | 6@28x28 | 6@14x14 | 16@14x14 | 16@7x7 | 120 | 100 | 10

ConvNet : feature extractor

MLP : classifier

1st layer convolutional filters

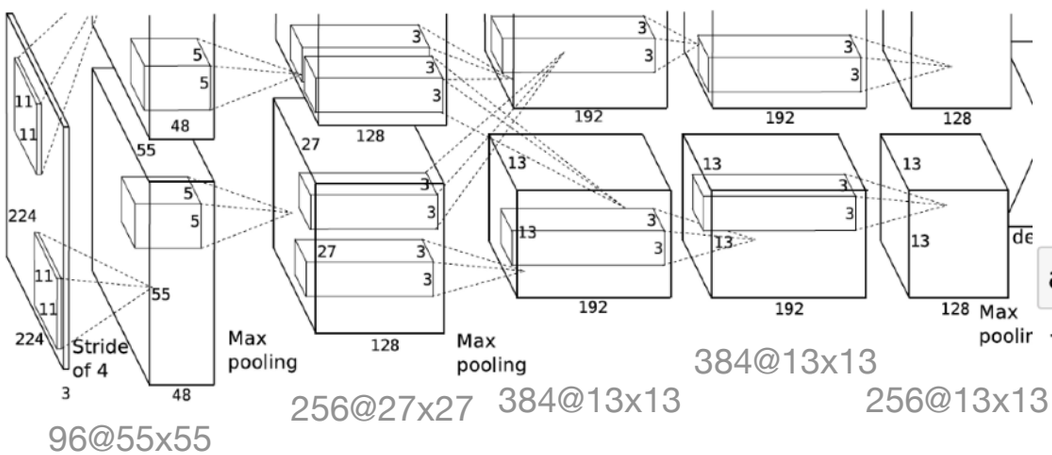


1st layer convolutional filters of AlexNet trained on the ImageNet dataset

- Load pre-trained AlexNet in Pytorch from torchvision.models

ImageNet dataset
~ 1.3 million images to train
50,000 images for validation
1000 categories

```
Input shape:     torch.Size([1, 3, 224, 224])
Conv2d output shape:     torch.Size([1, 64, 55, 55])
ReLU output shape:       torch.Size([1, 64, 55, 55])
MaxPool2d output shape:  torch.Size([1, 64, 27, 27])
Conv2d output shape:     torch.Size([1, 192, 27, 27])
ReLU output shape:       torch.Size([1, 192, 27, 27])
MaxPool2d output shape:  torch.Size([1, 192, 13, 13])
Conv2d output shape:     torch.Size([1, 384, 13, 13])
ReLU output shape:       torch.Size([1, 384, 13, 13])
Conv2d output shape:     torch.Size([1, 256, 13, 13])
ReLU output shape:       torch.Size([1, 256, 13, 13])
Conv2d output shape:     torch.Size([1, 256, 13, 13])
ReLU output shape:       torch.Size([1, 256, 13, 13])
MaxPool2d output shape:  torch.Size([1, 256, 6, 6])
```




```
import torchvision.models as models

alexnet = models.alexnet(pretrained=True)
```

```
alexnet
```

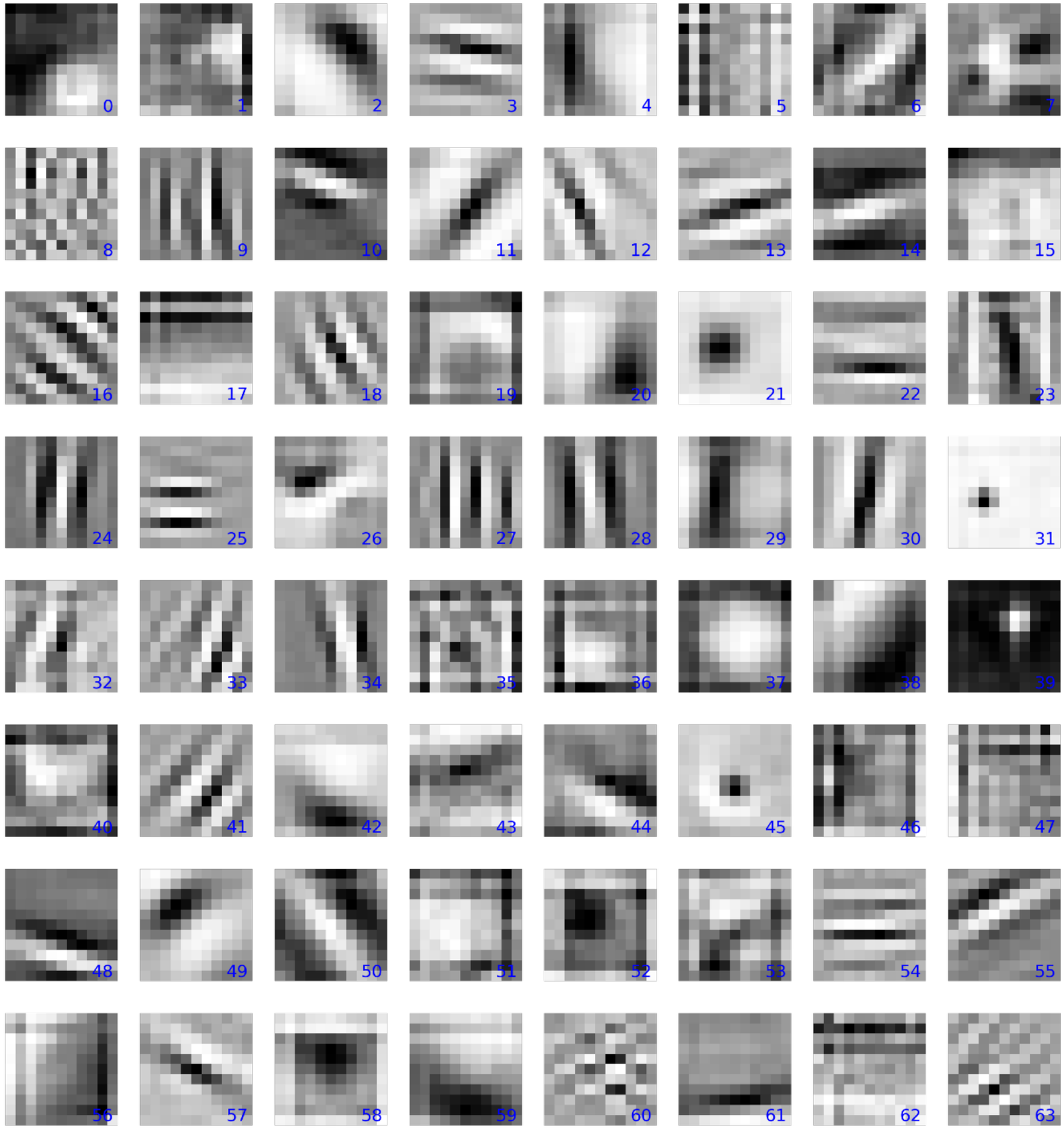
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

```
alexnet.features[0].weight.data.shape
```

```
torch.Size([64, 3, 11, 11])
```

1st layer convolutional filters of AlexNet trained on ImageNet dataset

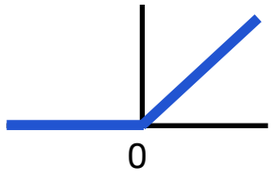
1st layer ConV weight matrix :
64 x 3 x 11 x 11
kernel size



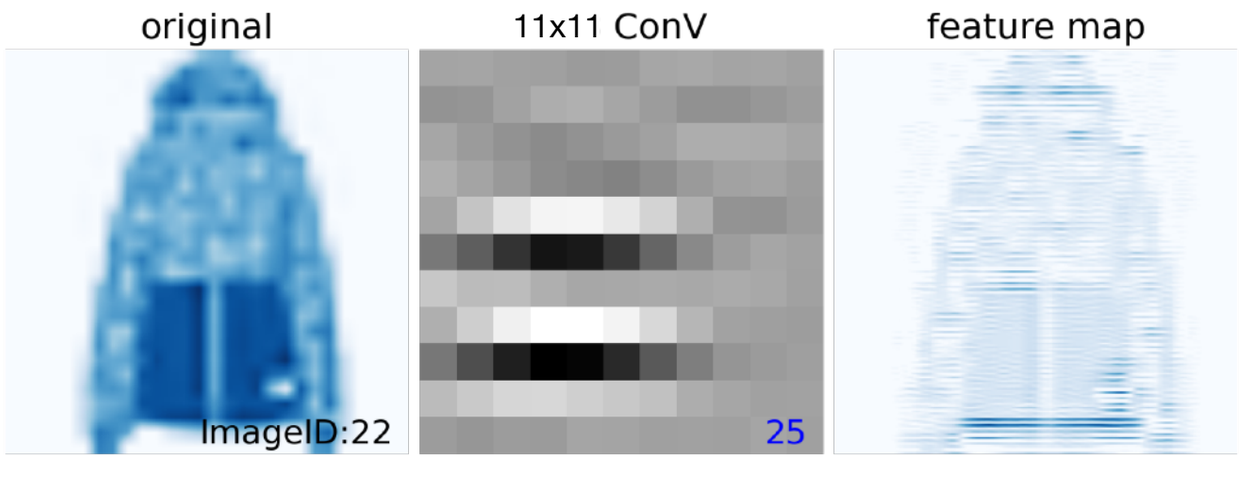

```
Input shape:     torch.Size([1, 3, 224, 224])
Conv2d output shape:     torch.Size([1, 64, 55, 55])
ReLU output shape:       torch.Size([1, 64, 55, 55])
MaxPool2d output shape:  torch.Size([1, 64, 27, 27])
Conv2d output shape:     torch.Size([1, 192, 27, 27])
ReLU output shape:       torch.Size([1, 192, 27, 27])
MaxPool2d output shape:  torch.Size([1, 192, 13, 13])
Conv2d output shape:     torch.Size([1, 384, 13, 13])
ReLU output shape:       torch.Size([1, 384, 13, 13])
Conv2d output shape:     torch.Size([1, 256, 13, 13])
ReLU output shape:       torch.Size([1, 256, 13, 13])
Conv2d output shape:     torch.Size([1, 256, 13, 13])
ReLU output shape:       torch.Size([1, 256, 13, 13])
MaxPool2d output shape:  torch.Size([1, 256, 6, 6])
```

1st layer convolutional filters of **AlexNet** trained on **ImageNet** dataset and applied on **Fashion-MNIST**

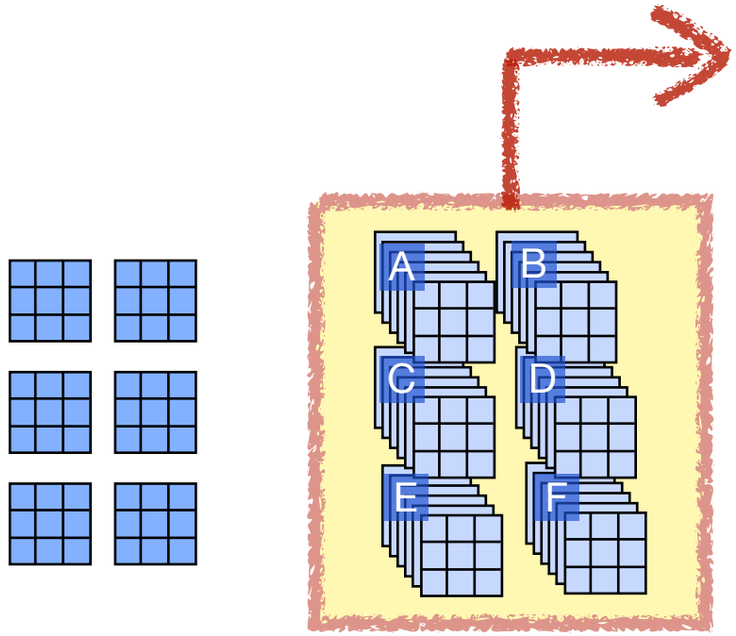
1st layer convolutional filters of AlexNet trained on ImageNet dataset and applied on Fashion-MNIST

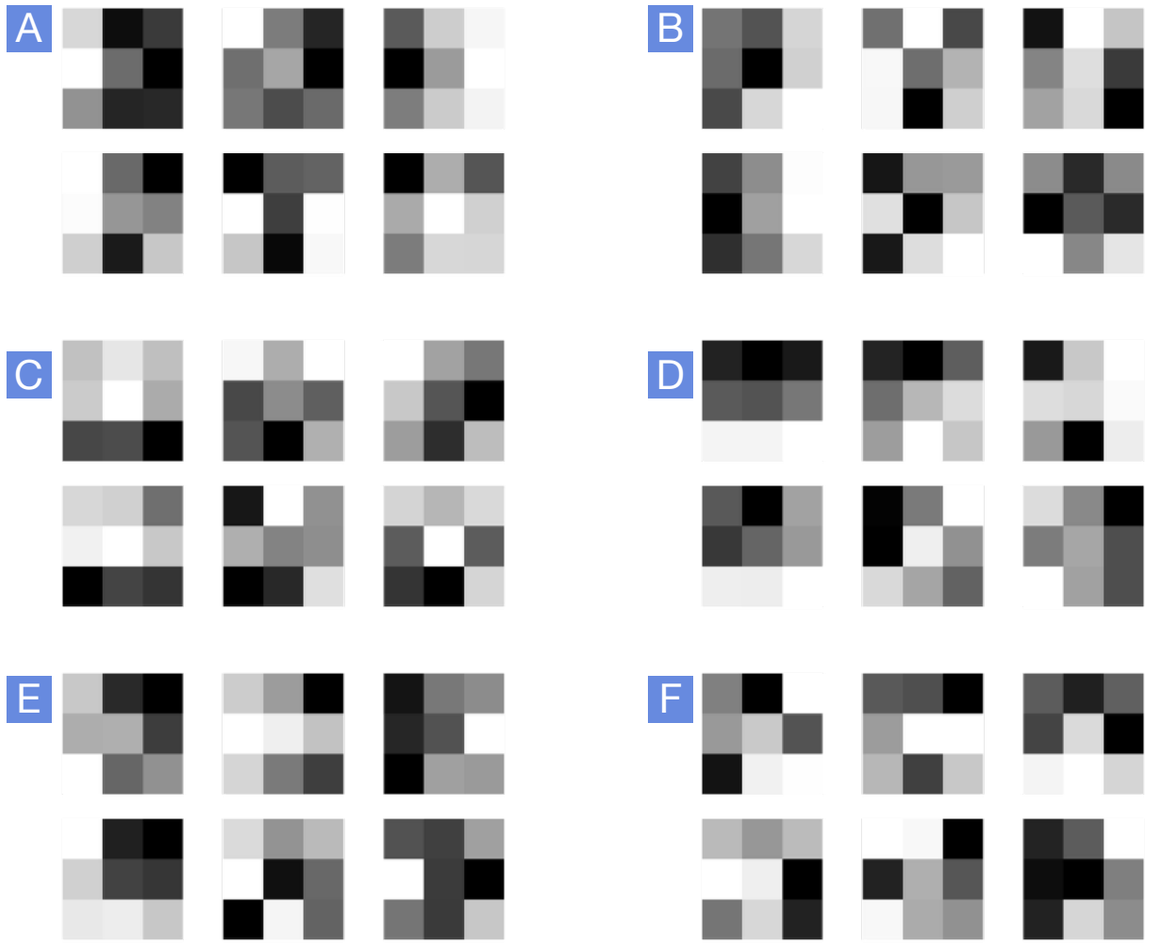


after applying **ReLU** activation …

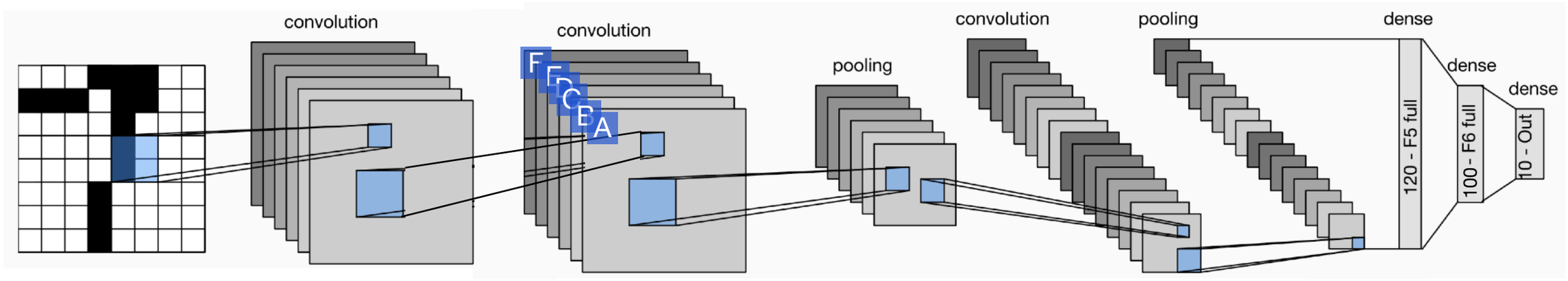


2nd layer convolutional filters





ConV1 3x3 | ConV2 3x3 | MaxPool | ConV3 3x3 | MaxPool

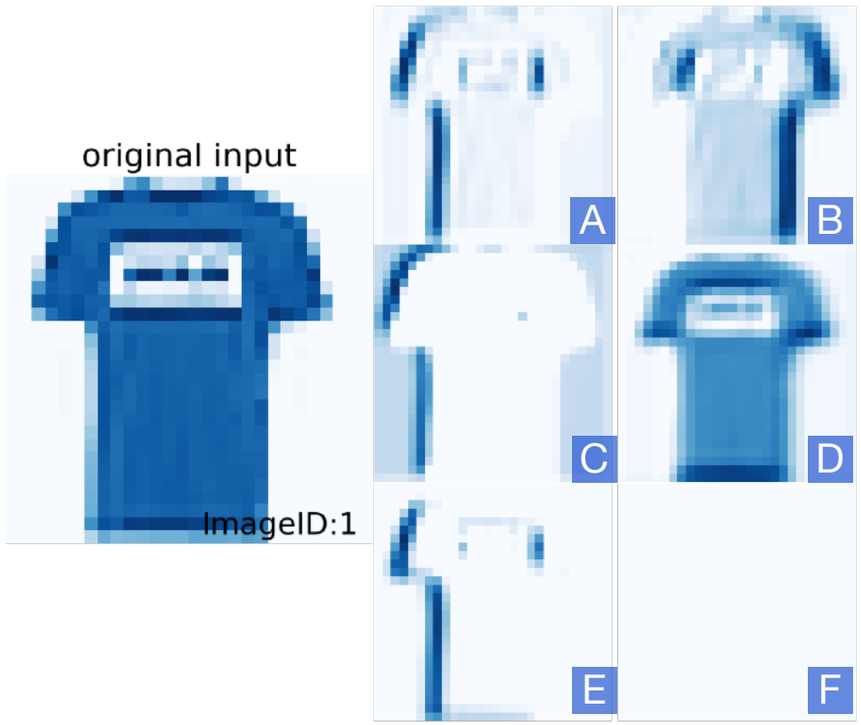


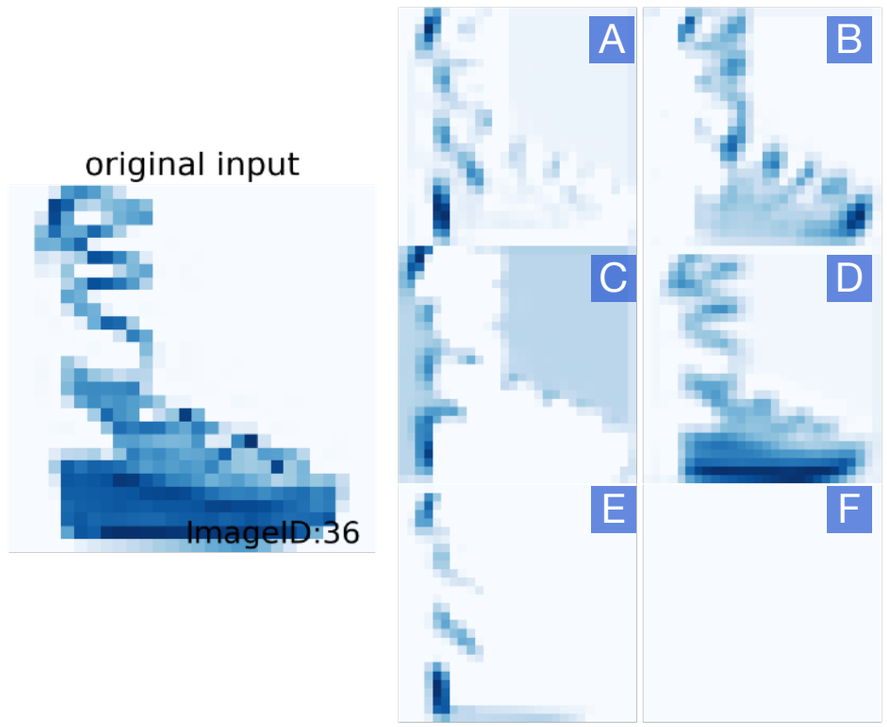
1@28x28 | 6@28x28 | 6@28x28 | 6@14x14 | 16@14x14 | 16@7x7 | 120 | 100 | 10

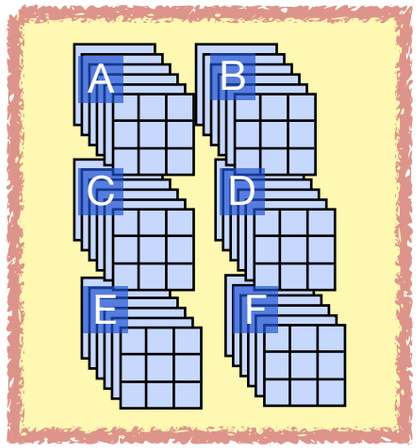
ConvNet : feature extractor

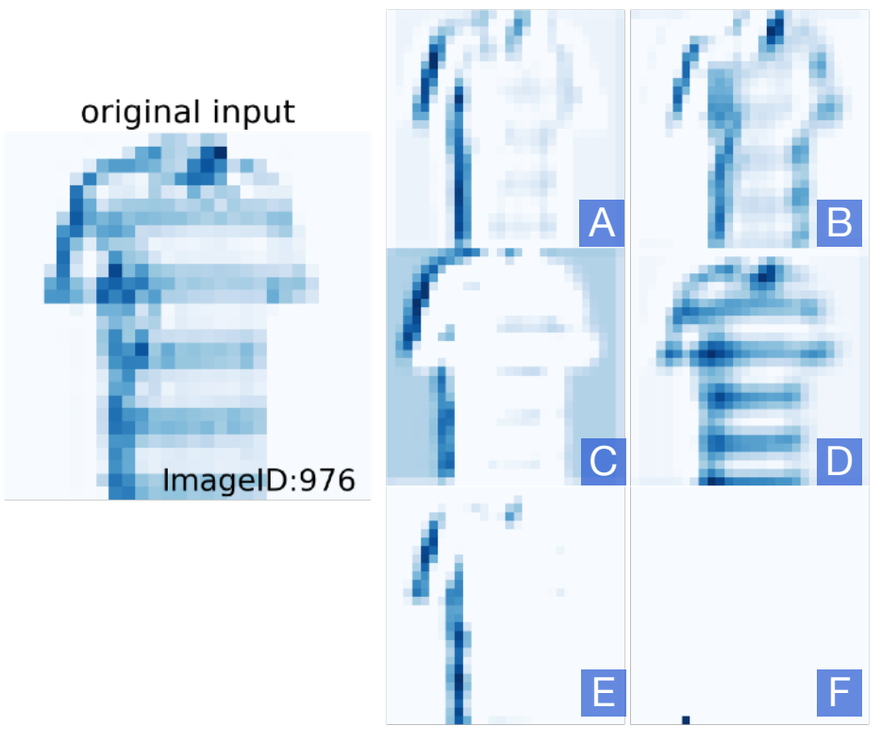
MLP : classifier

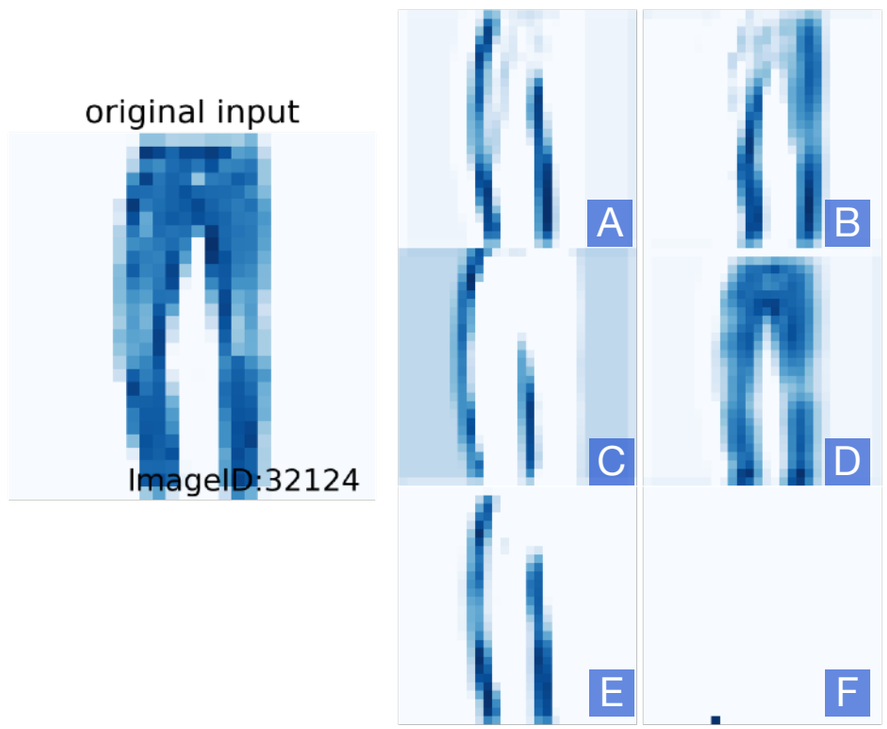
Feature Maps out from **ConV2** filters + **ReLU**

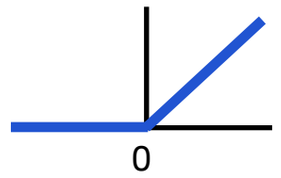












Dying ReLU Problem

When ReLU neurons become inactive and **only output 0 for any input.**

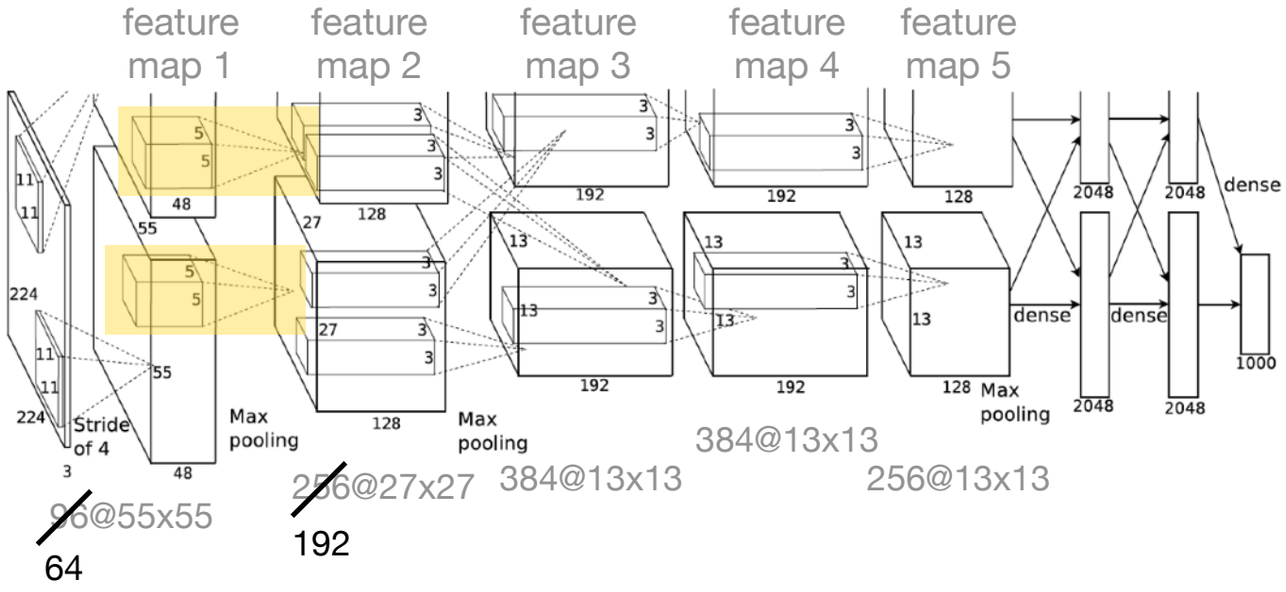
Happens when

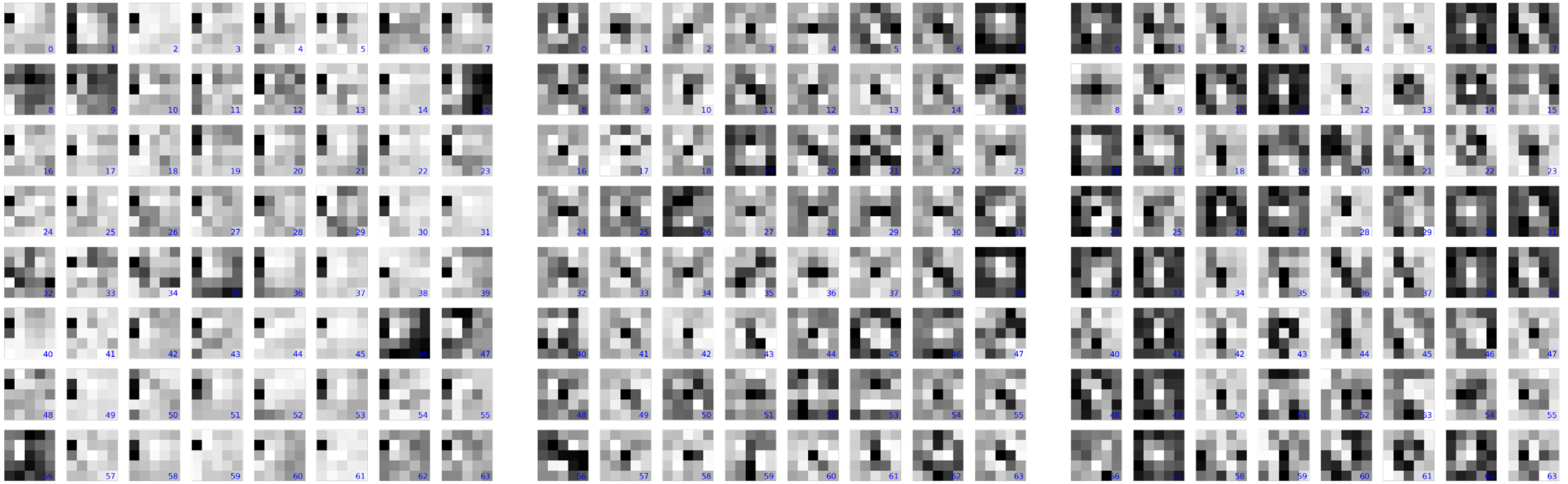
1. learning rate is too high.
2. large negative bias

$$\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} L_{\text{mini}}(\mathbf{w})$$

2nd layer convolutional filters of AlexNet trained on ImageNet dataset

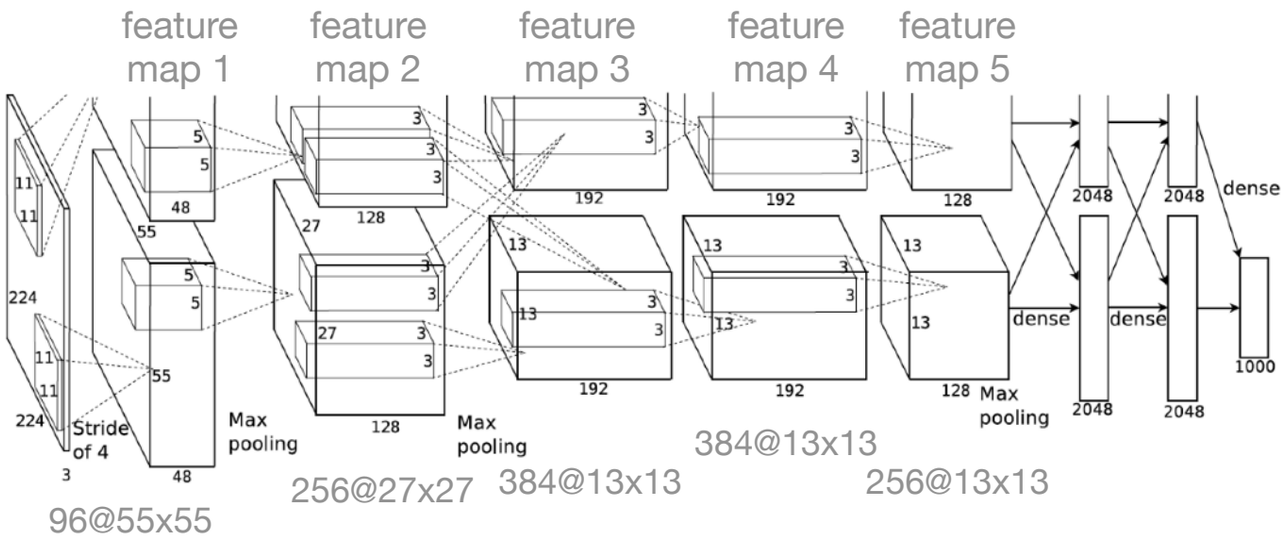
2nd layer ConV weight matrix :
192 x 64 x 5 x 5
kernel size

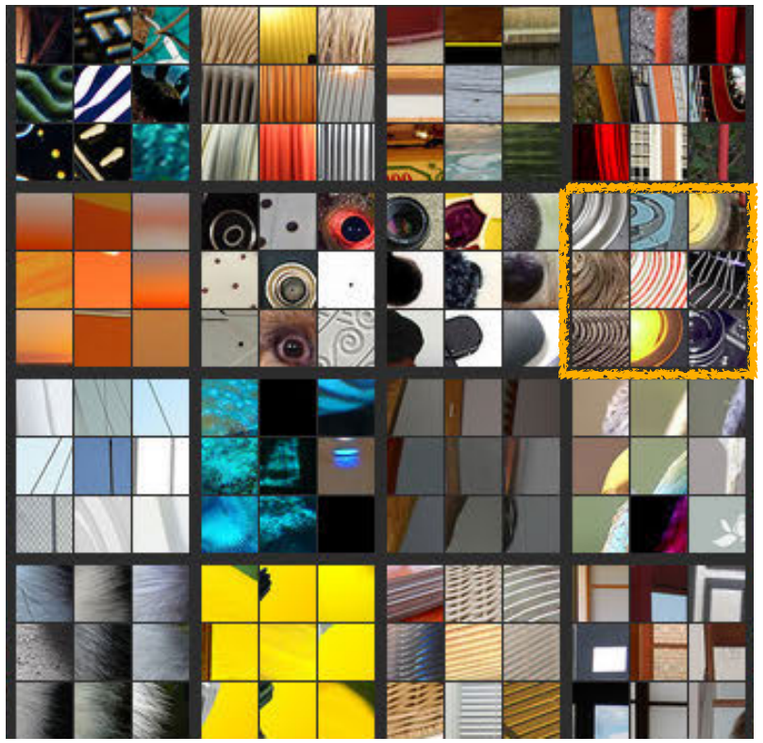


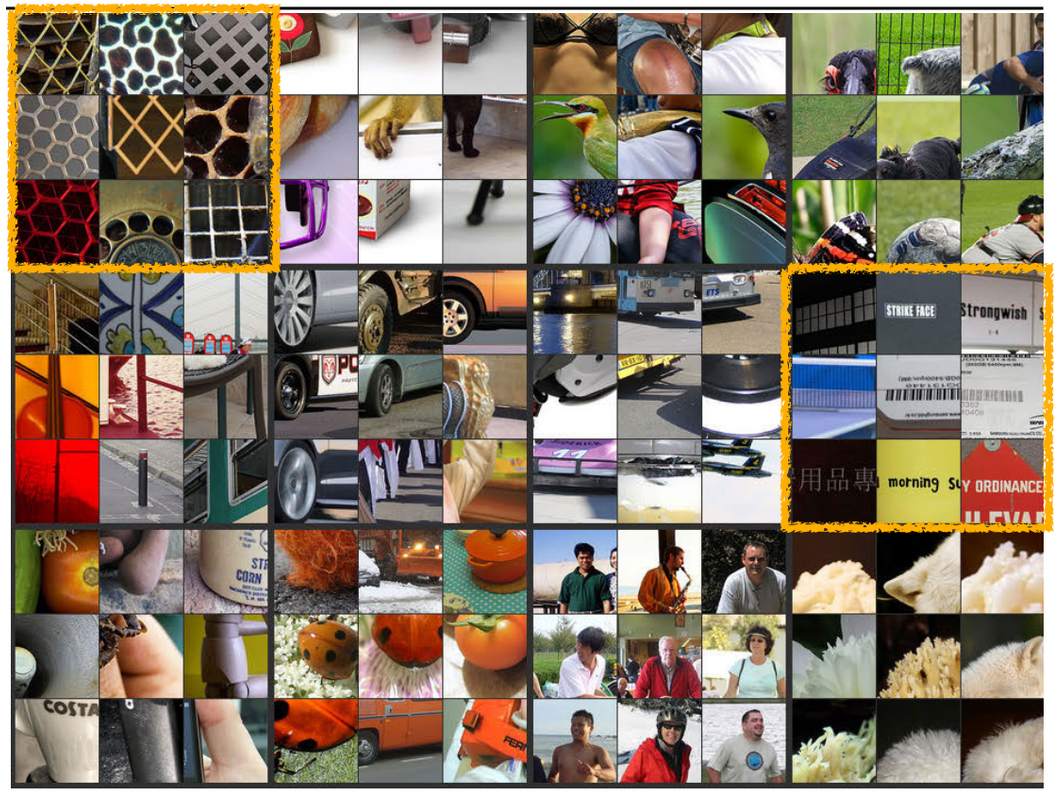


… and another 189 of the 64x5x5 filters

Maximally activated image patches (for **AlexNet** trained on ImageNet)





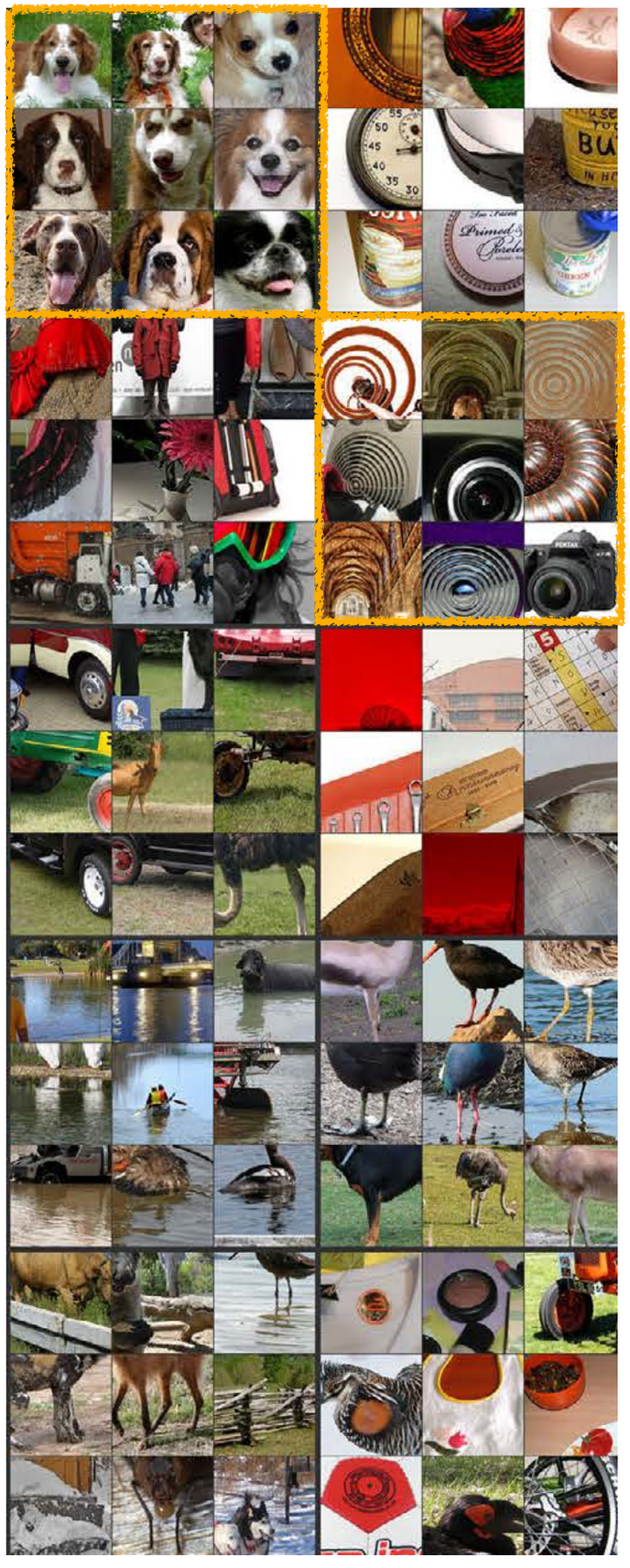


layer 1 layer 2 layer 3

Image Credit :
Fig.2 of Zeiler et al. 2013 (arXiv:1311.2901)

Maximally activated image patches (for **AlexNet** trained on ImageNet)

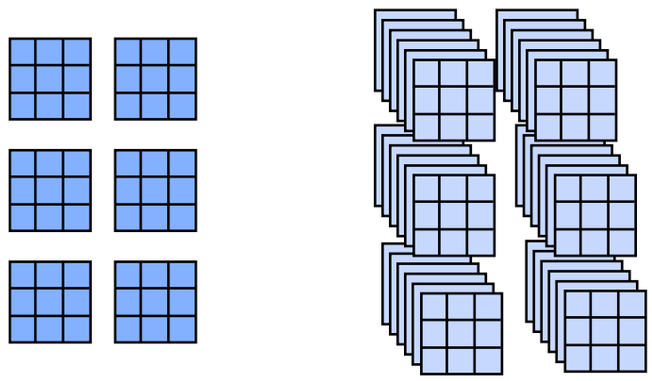
layer 4



layer 5

Image Credit :
Fig.2 of Zeiler et al. 2013 (arXiv:1311.2901)

Last Layer feature vector



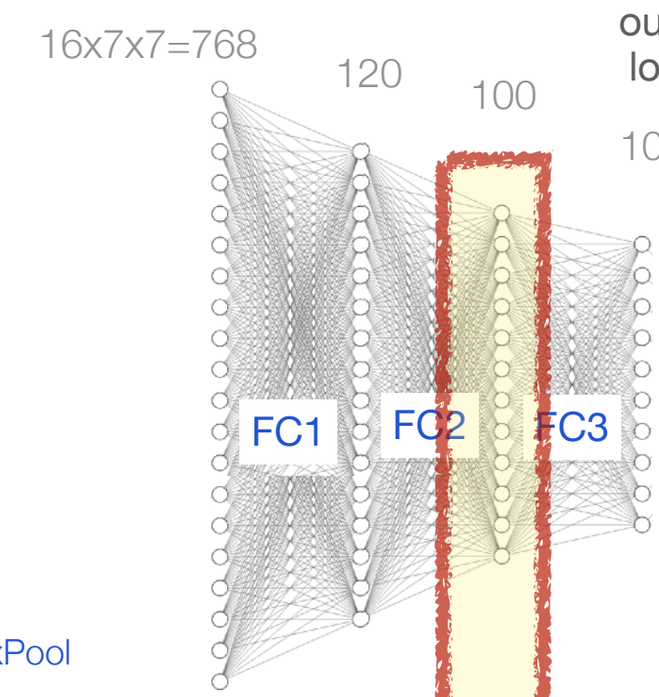
ConV1
3x3

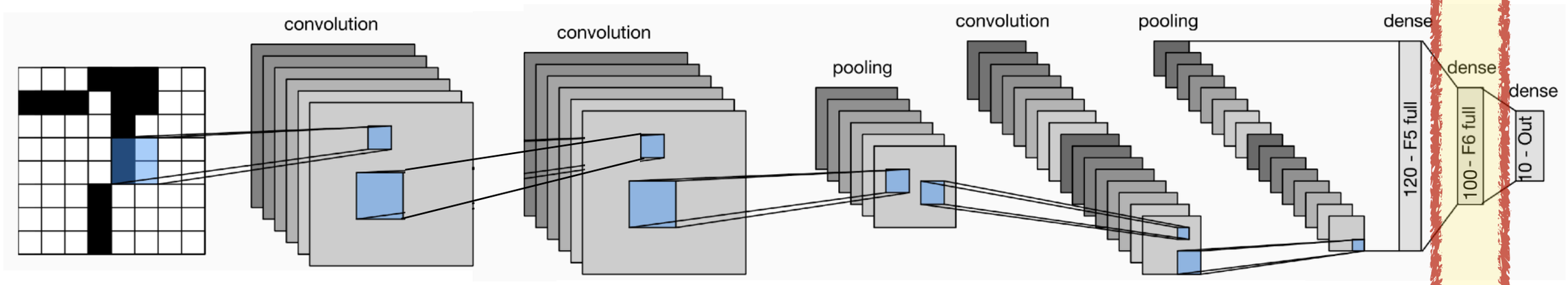
ConV2
3x3

MaxPool

ConV3
3x3

MaxPool





1@28x28 6@28x28 6@28x28 6@14x14 16@14x14 16@7x7 120 100 10

ConvNet : feature extractor

MLP : classifier

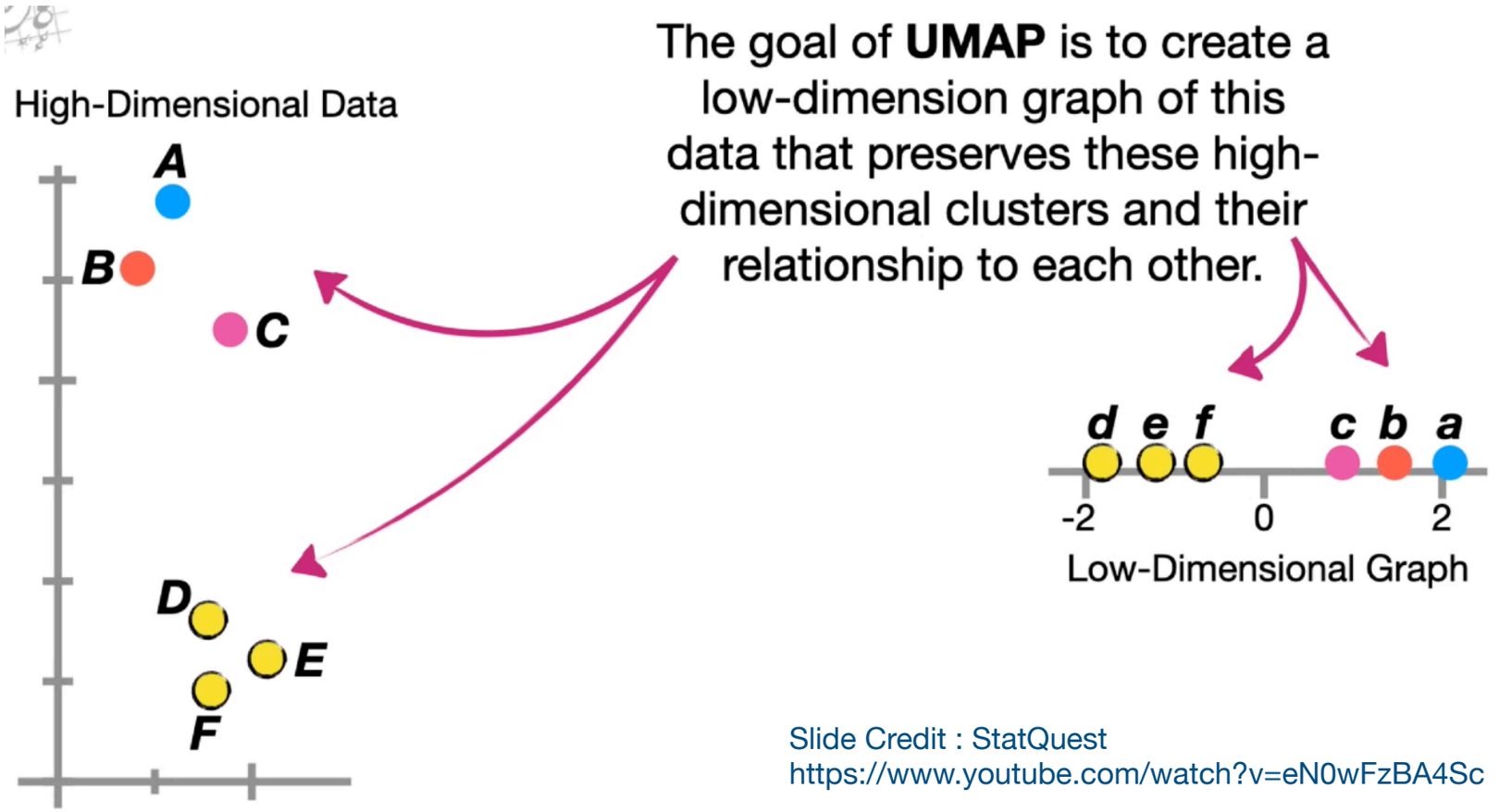
Architecture

ConV1 → ReLU → ConV2 → ReLU → Pool → ConV3 → ReLU → Pool → Flatten → FC1 → ReLU → FC2 → ReLU → FC3 → SoftMax

UMAP — Uniform Manifold Approximation and Projection

- Our **Last Layer Feature Vector** has a dimension of **100**, for each input image (28x28). How to better visualize such a high dimensional vector ?

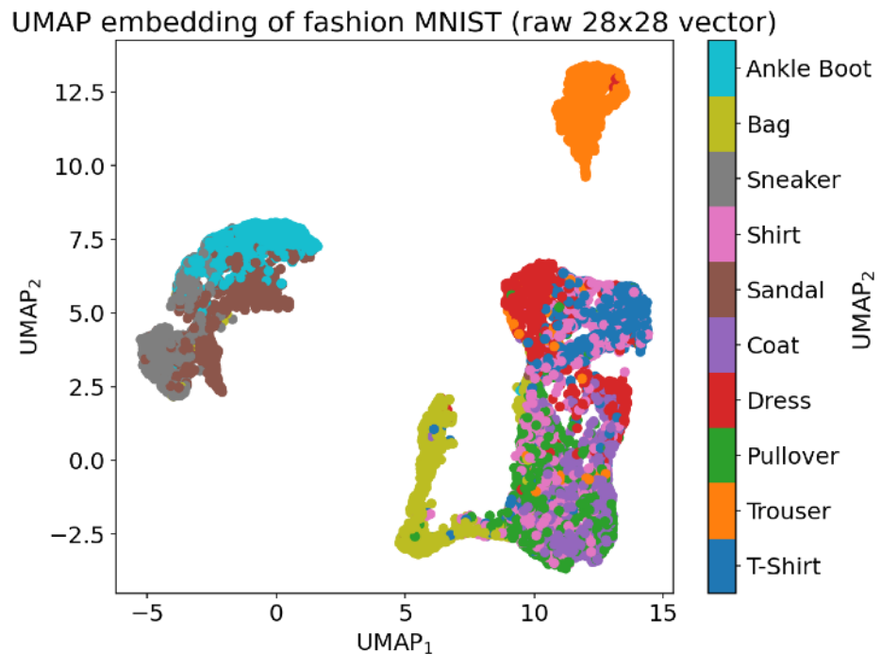
- **UMAP** : an algorithm for **dimension reduction** that maps **high-dimensional vectors** to **low-dimension** while preserving the “**similarity**” or the "**clustering properties**” of the original vector.



Slide Credit : StatQuest
https://www.youtube.com/watch?v=eN0wFzBA4Sc

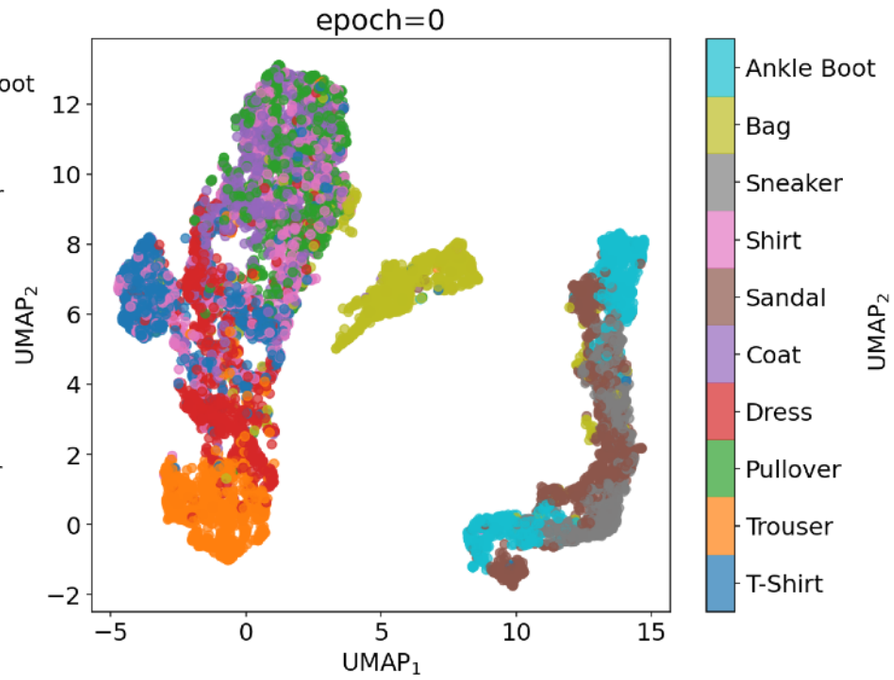
Clustering of the the Last Layer feature vectors

28x28 = 784 $\xrightarrow{\text{map}}$ 2

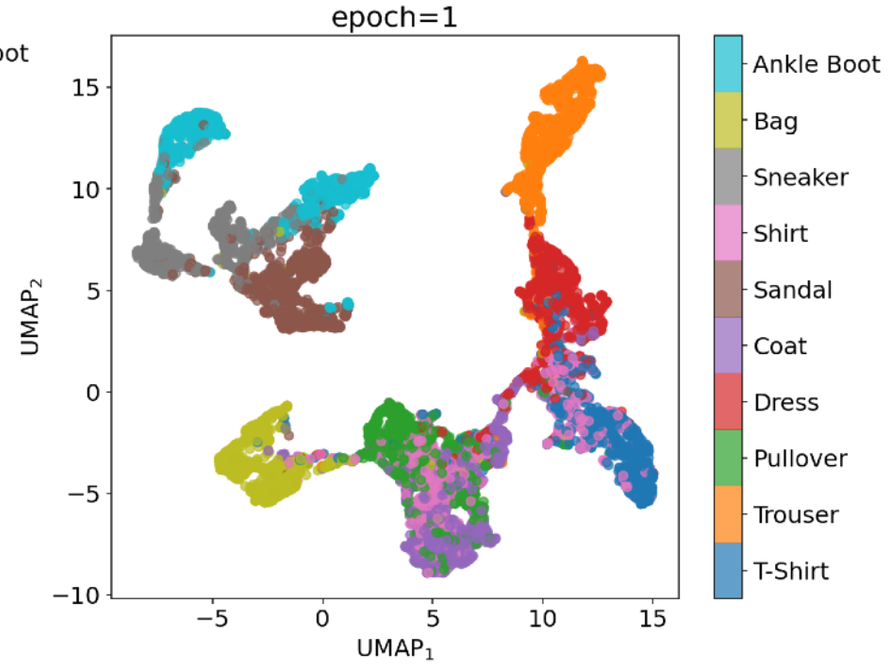


before training

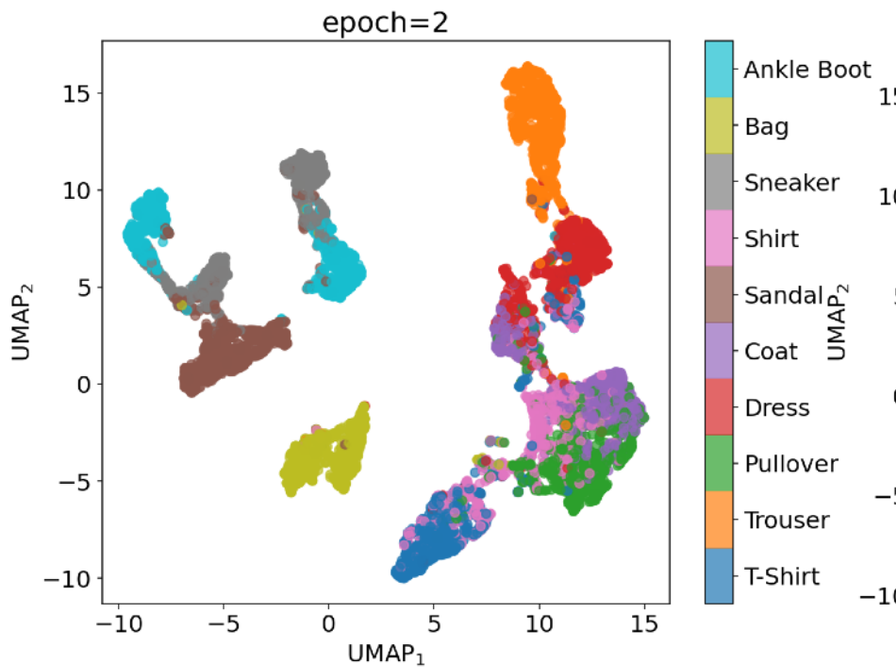
100 $\xrightarrow{\text{map}}$ 2



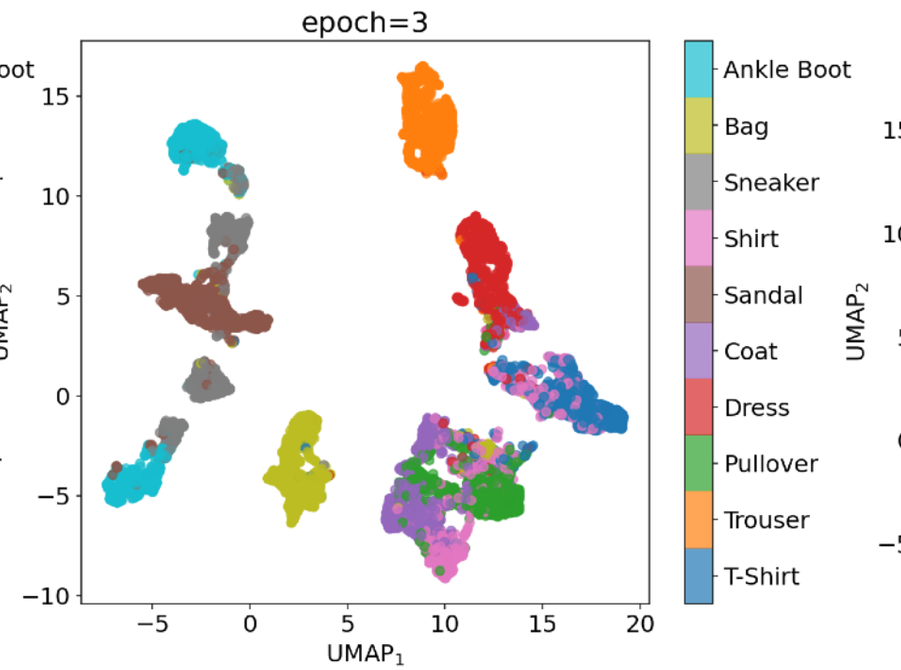
trained after 1 epoch
test accuracy 79%



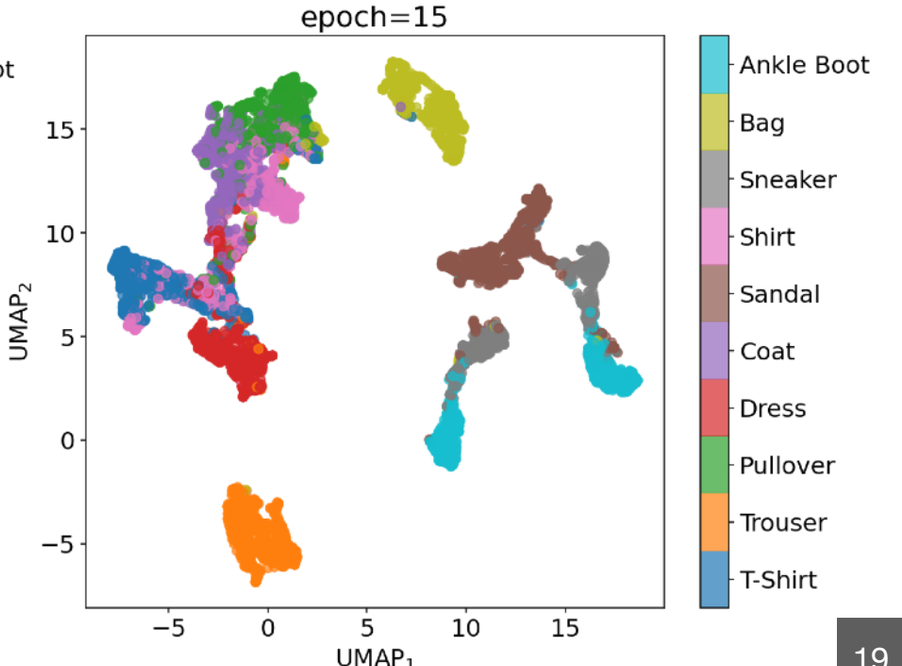
trained after 2 epochs
test accuracy 84%



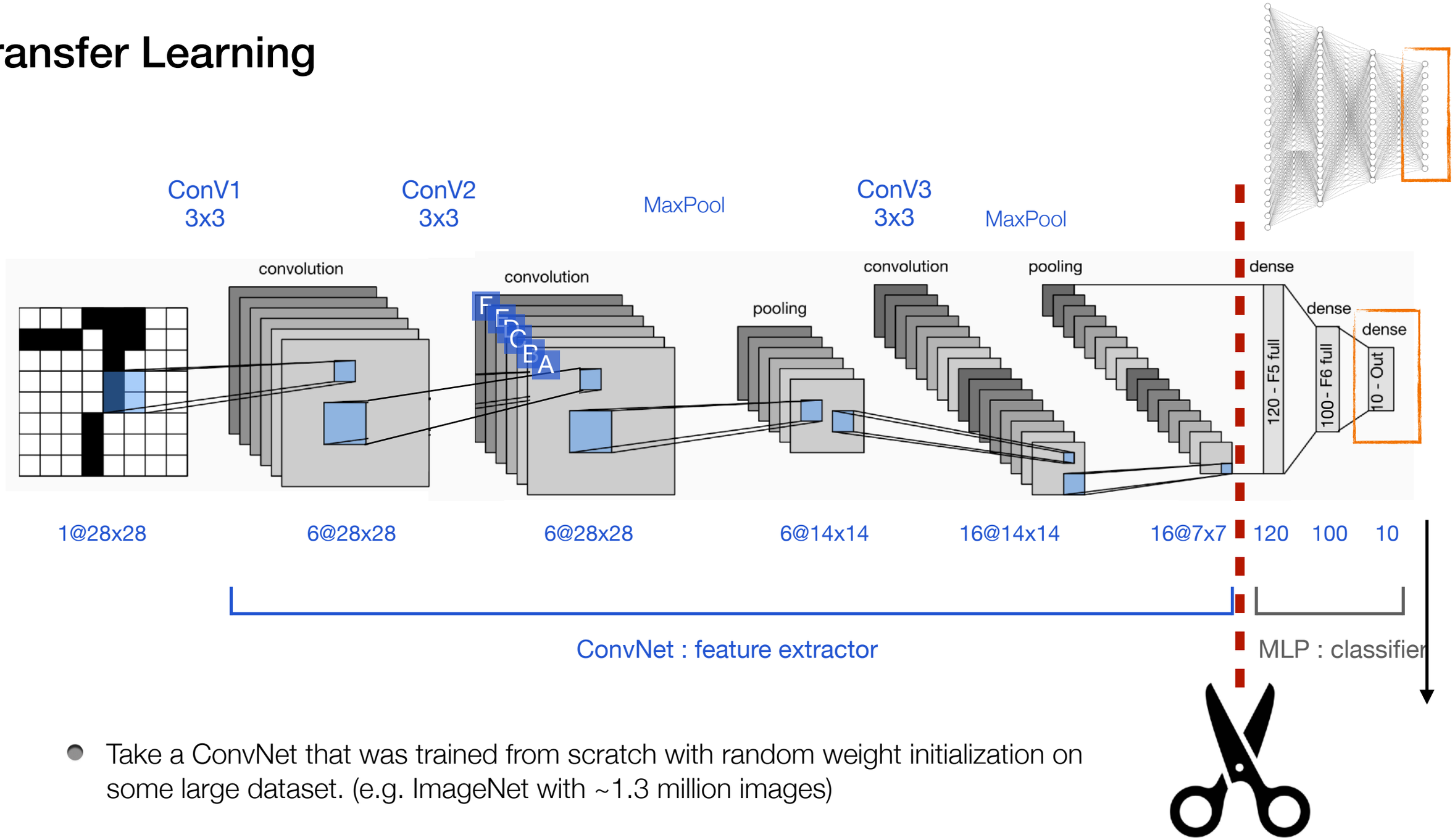
trained after 3 epochs
test accuracy 86%



trained after 15 epochs
test accuracy 90%



Transfer Learning



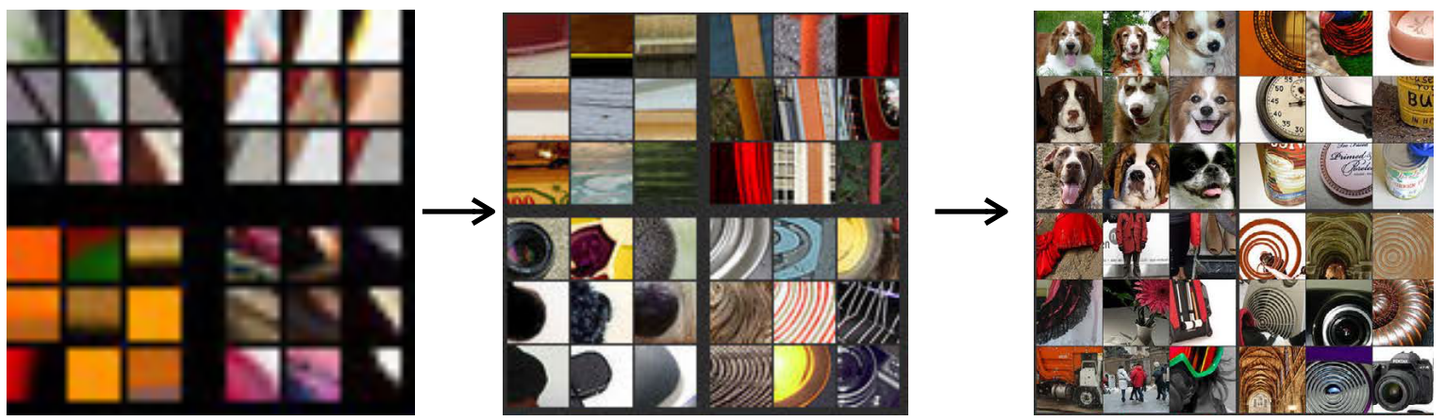
- Take a ConvNet that was trained from scratch with random weight initialization on some large dataset. (e.g. ImageNet with ~1.3 million images)

- Use the convolutional parts of the network as feature extractor.
  Freeze the network weight parameters in the convolutional parts.

- **Retrain the MLP classification part** with the **new dataset** that you care about. Modify the MLP network structure if needed (especially the number of output neurons to match the number of outputs for your task).

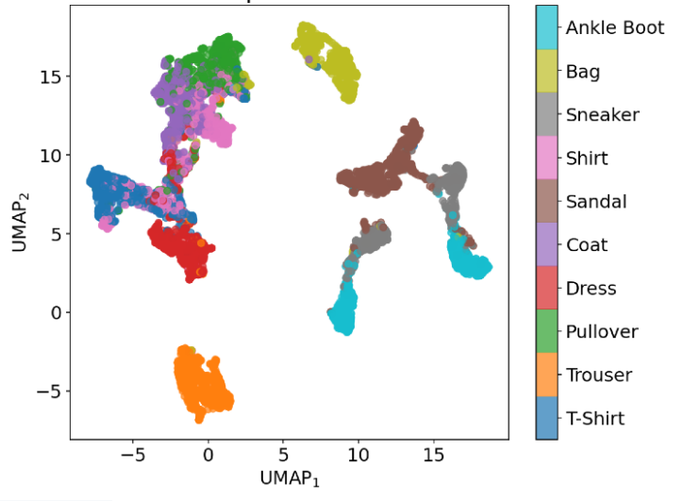
Summary

- Convolutional Nets learn image patterns **hierarchically**.
  Earlier layers detect simple **edges** and **boundaries.**
  Composing the lower level features, the **later layers** learn more **complex structures**.



- We can view the **last layer feature vector** (before the output logits) as a **final condensed summary vector** for each image.

  Visualize the clustering of these high dimensional vectors with **UMAP**.



- Transfer Learning :
  Utilizing the pre-trained ConvNet as an effective feature extractor.