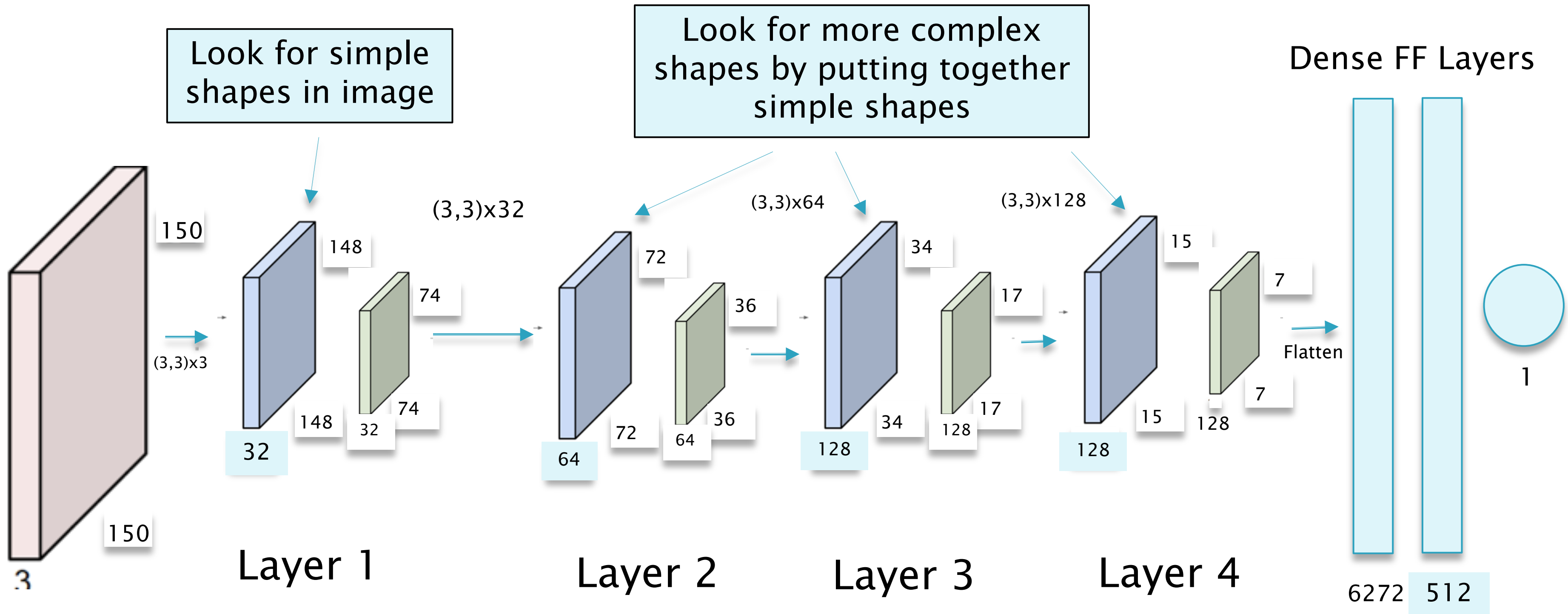
Convolutional Neural Networks: Part 2

Lecture 11

Subir Varma

CNN Architecture



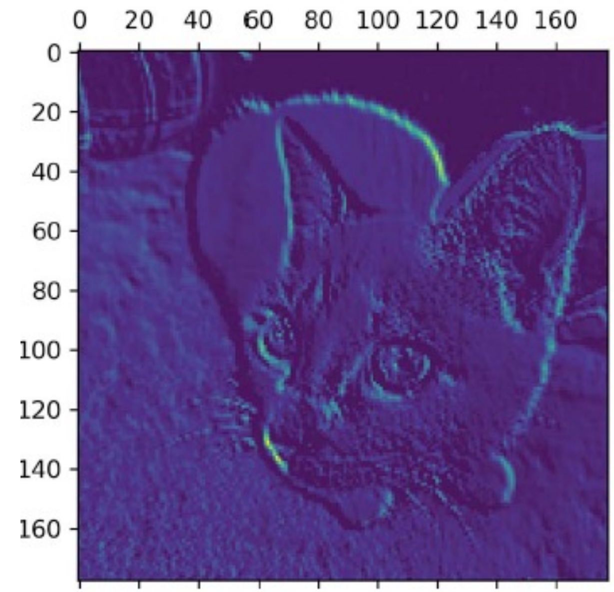
The number of Activation Maps in each Layer is a Hyper Parameter (32,64,128,128)

A Node in a Dense Feed Forward Network ←→ An Activation Map in a ConvNet

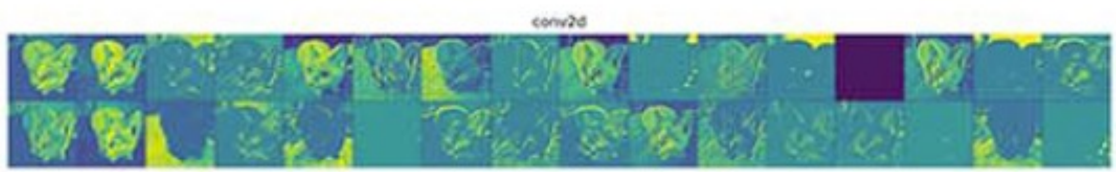
Visualizing an Activation Map

Input Image

Activation Map

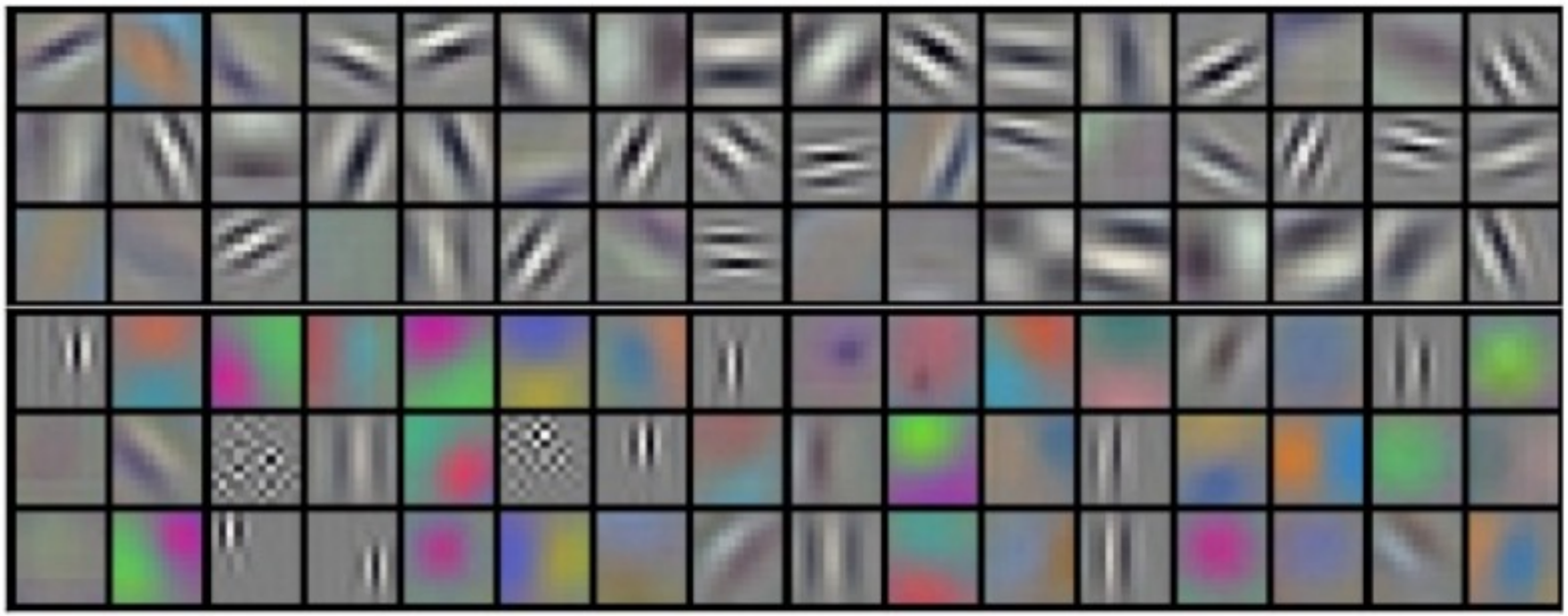


Detects Edges



Chollet P. 266

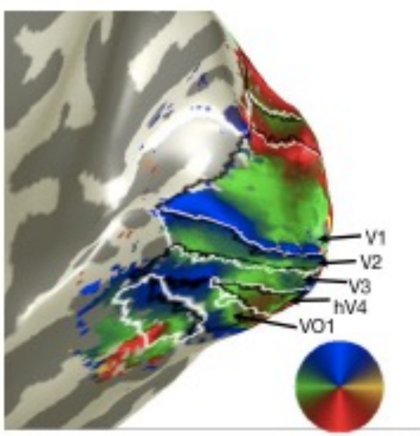
Visualizing the Local Filters – First Layer

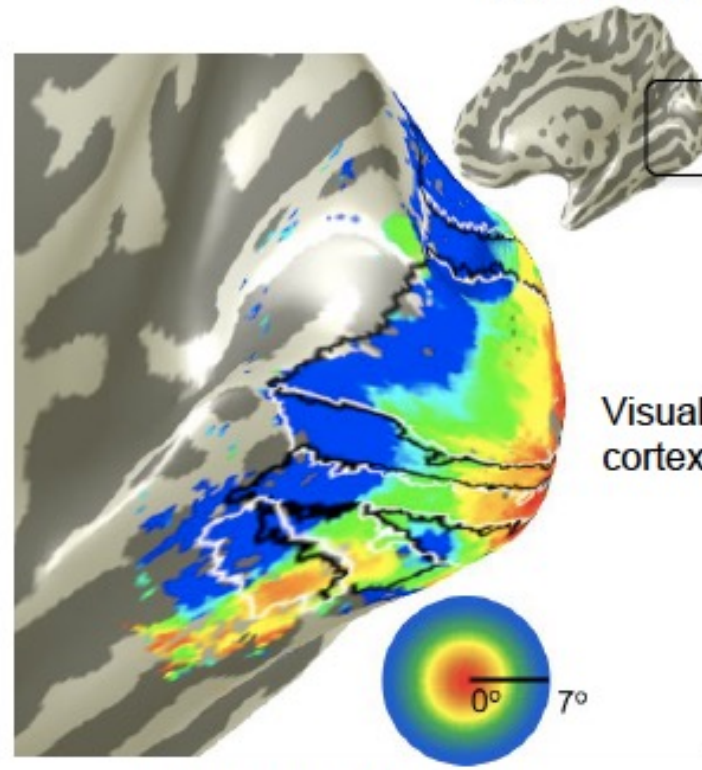


96 Local Filters, looking for simple shapes

Visual Processing in the Brain

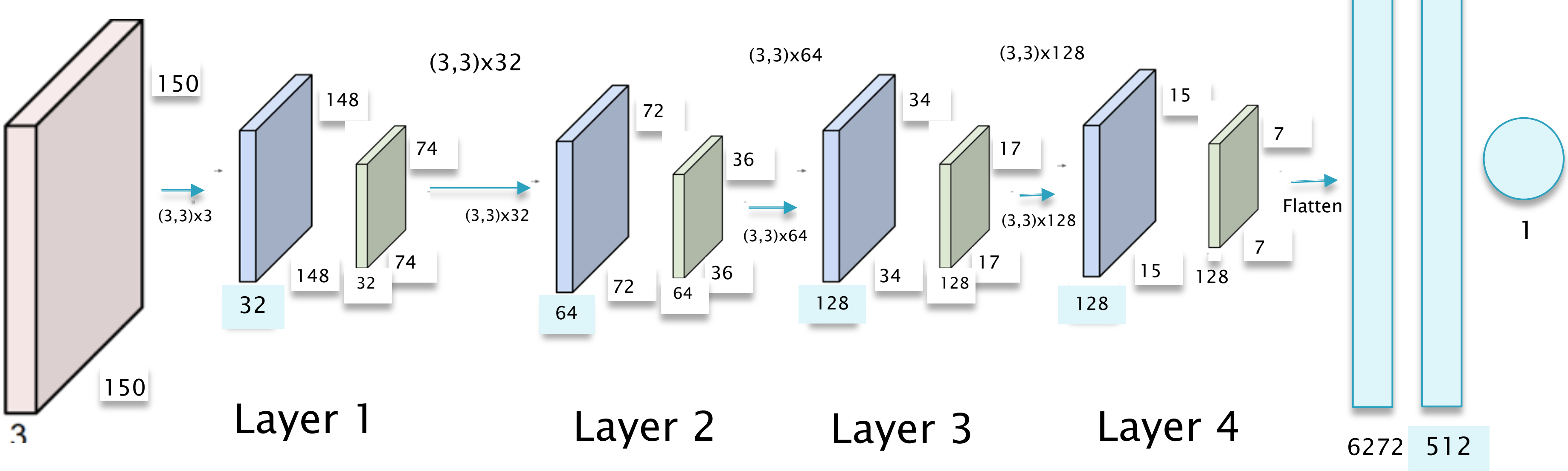
Topographical mapping in the cortex: nearby cells in cortex represent nearby regions in the visual field





Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

CNNs in Keras




```python
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Layer 1

Layer 2

Layer 3

Layer 4

For each Conv Layer, specify:
1. Number of Activation Maps
2. Size of Filter used to generate Activation Map
3. Activation Function

For Max Pooling Layer only need to specify Filter Size

CNNs in Keras

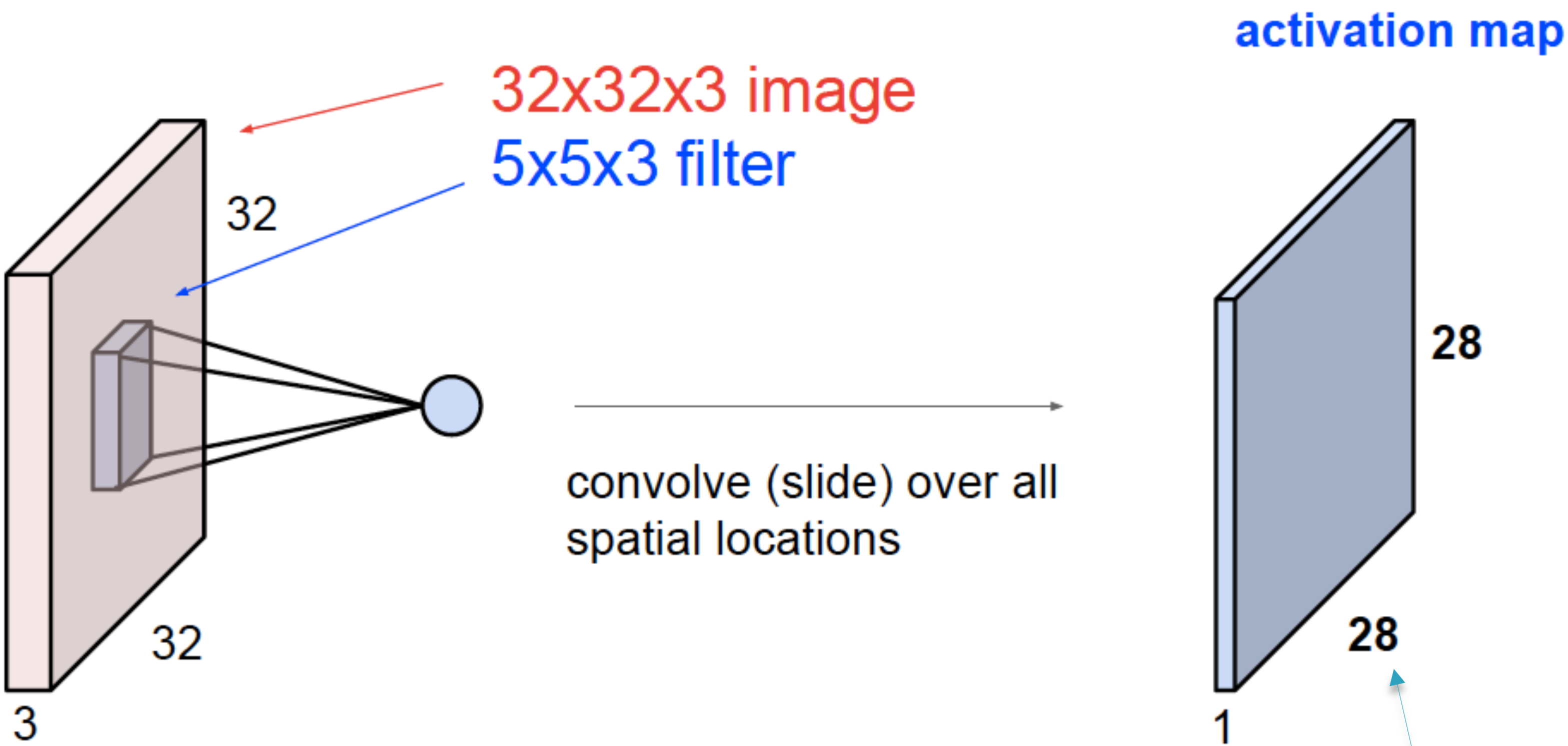
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

How to compute these numbers?

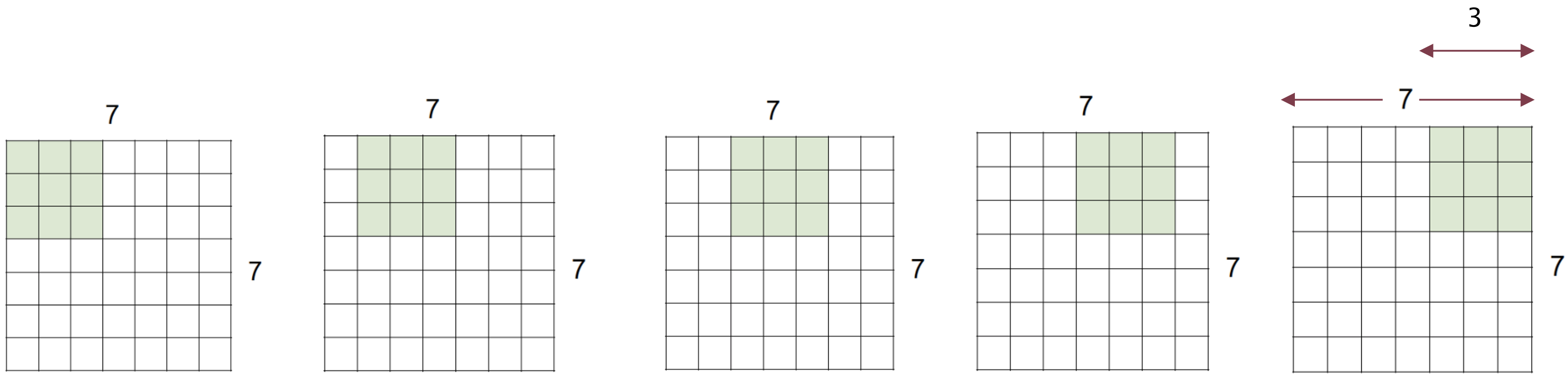
Sizing CNNs

Size of Convolutional Layer



How did we get this number?

Example: Stride = 1



(a) 7x7 Input, 3x3 Filter, S = 1: Results in 5x5 Output

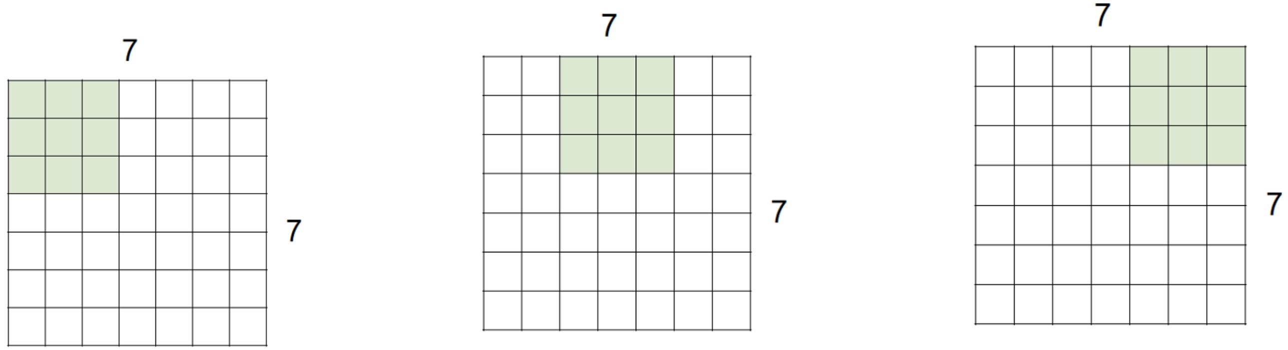
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

$$5 = (7 - 3) + 1$$

$$N' = (N - F) + 1$$

Stride = 2



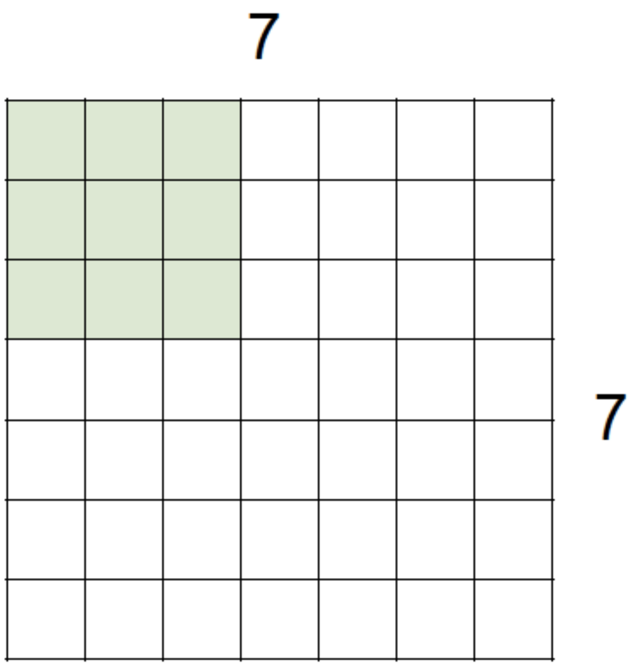
(b) 7x7 Input, 3x3 Filter, S = 2: Results in 3x3 Output

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

$$3 = (7-3)/2 + 1$$

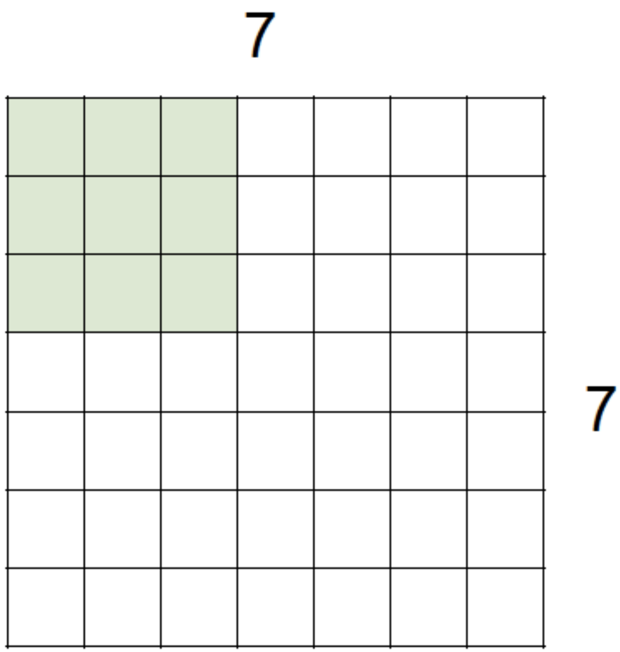
$$N' = \frac{(N-F)}{S} + 1$$

Stride=3



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

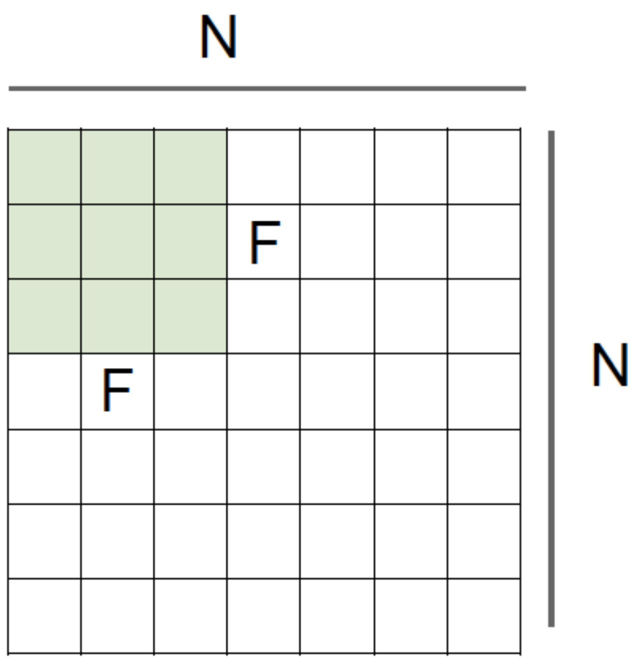
Stride=3



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

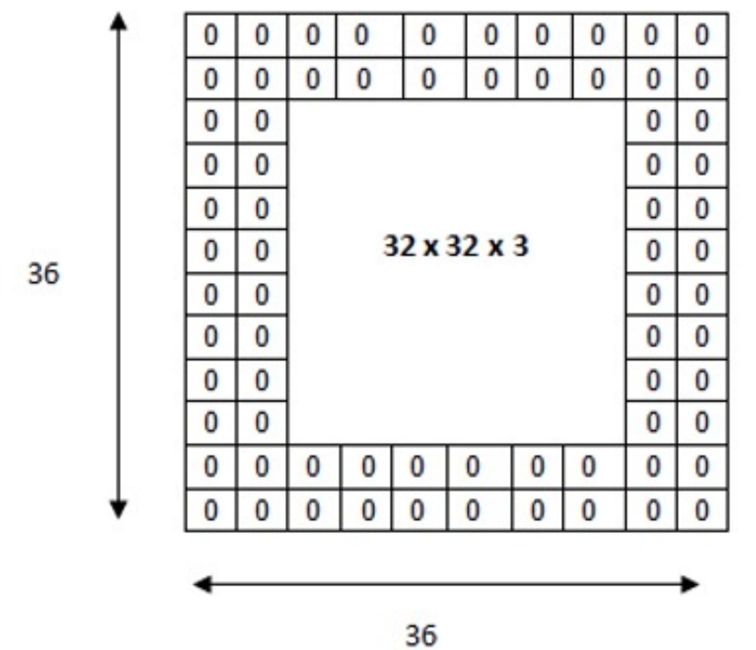
Size of Next Activation Map



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

With Zero Padding



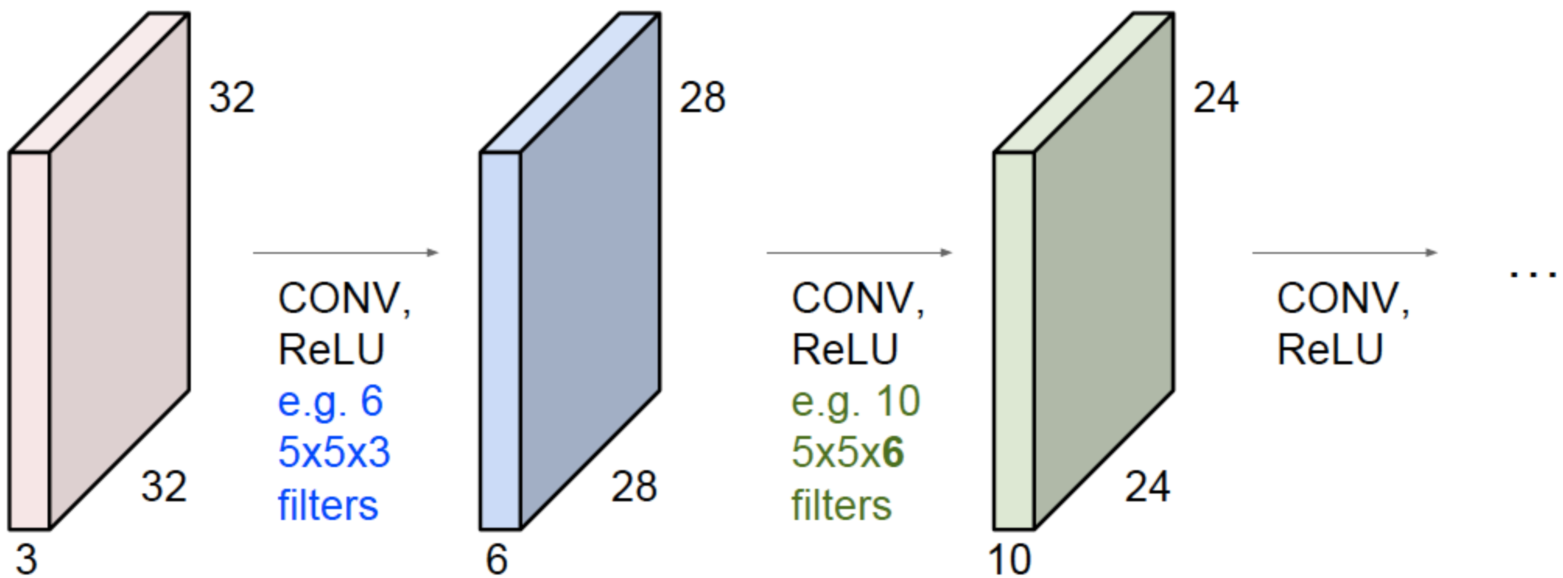
Zero padding of size P increases
The dimensions of the Activation Map
By 2P

Final Formula:

$$N2 = \frac{N1 - F + 2P}{S} + 1$$

Repeated Convolutions → Shrinking Volumes

Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Constant Volume Networks

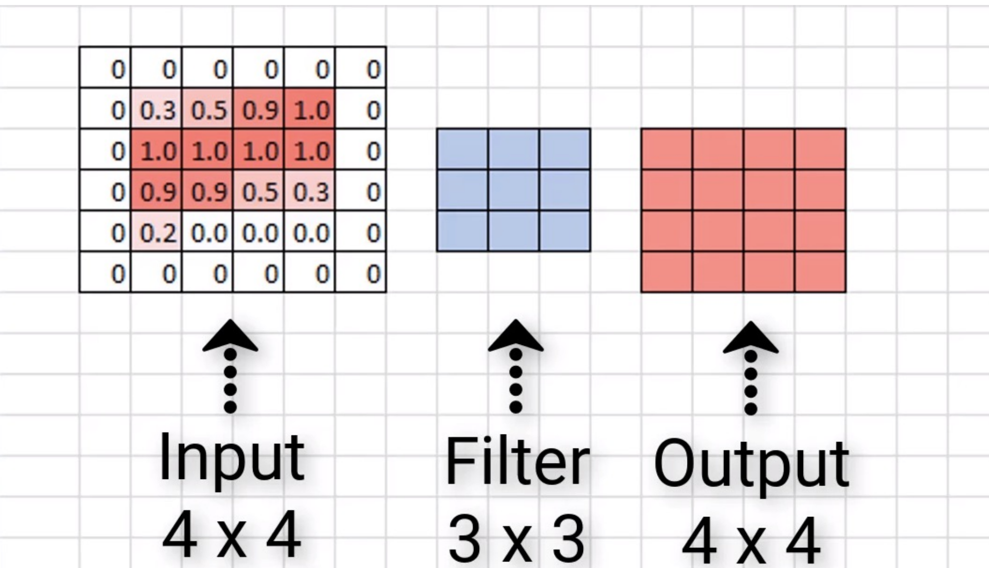
$$N2 = \frac{N1 - F + 2P}{S} + 1$$

Assume S = 1

N2 = N1 - F +2P +1

If N1 = N2, then

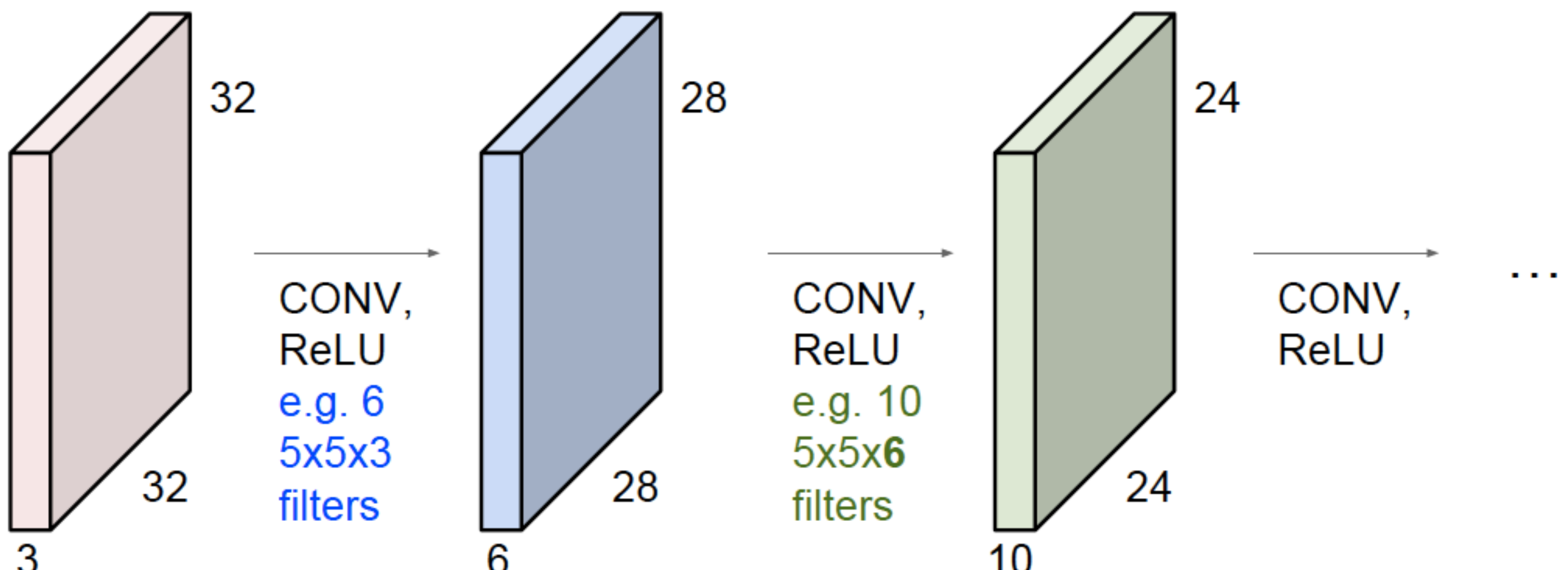
$$P = \frac{F-1}{2}$$



e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

```
conv = tf.keras.layers.Conv2D(6, (3,3), strides=2, padding='same')
```

Computing Number of Parameters



Layer 1

6 * (5*5*3 +1) = 456 N2 = (32 – 5) + 1 = 28

3072 * 6 = 18,432 (for Dense Feed Forward case)

Computations: 5*5*3 * 28*28 * 6 = 352,800

6 * 3072 = 18,500 (For Dense Feed Forwards case)

Layer 2

10 * (5*5*6 + 1) = 1510 N3 = (28 – 5) + 1 = 24

Summary of Volume Computation

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$

- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

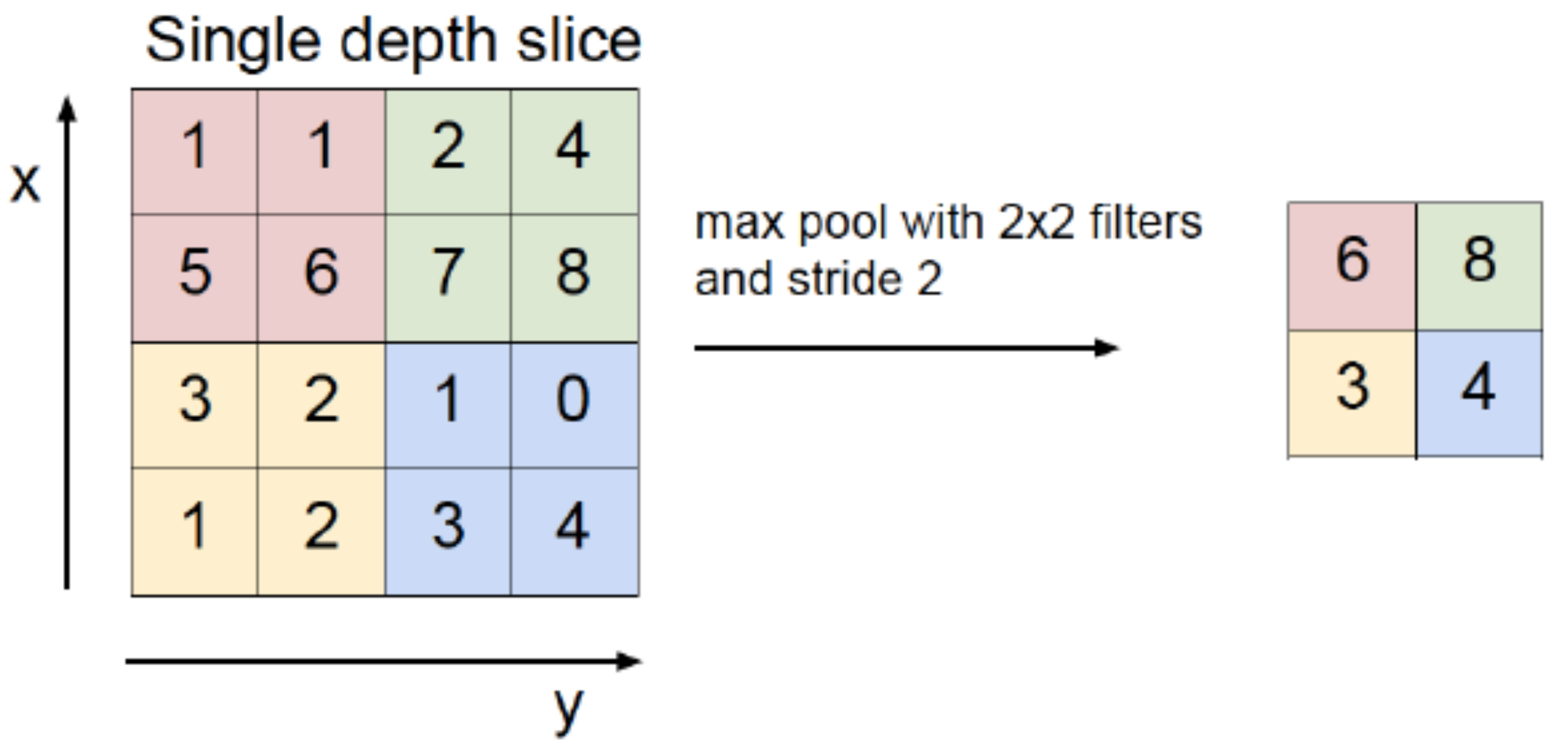
Summary of Volume Computation

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Max Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Size of Pooling Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input

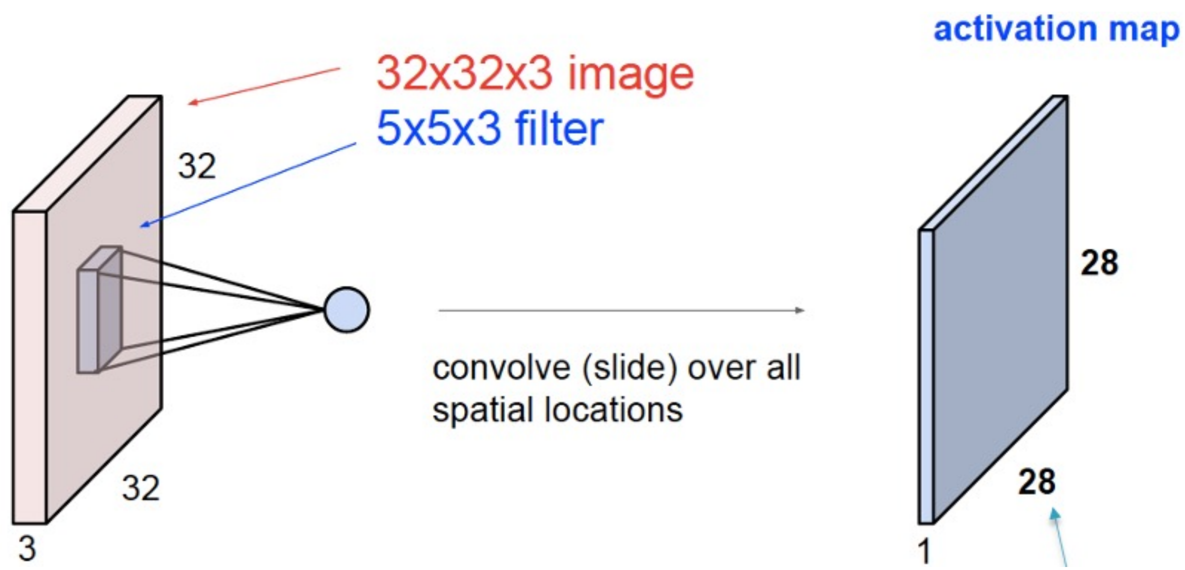
Common settings:

F = 2, S = 2
F = 3, S = 2

Number of Computations

- Number of computations needed to generate all the activations in ConvNet layer (r+1)

$$Mult = F_r F_r D_r \times W_{r+1} H_{r+1} \times D_{r+1}$$



Number of Computations

- Number of computations needed to generate all the activations in ConvNet layer (r+1)

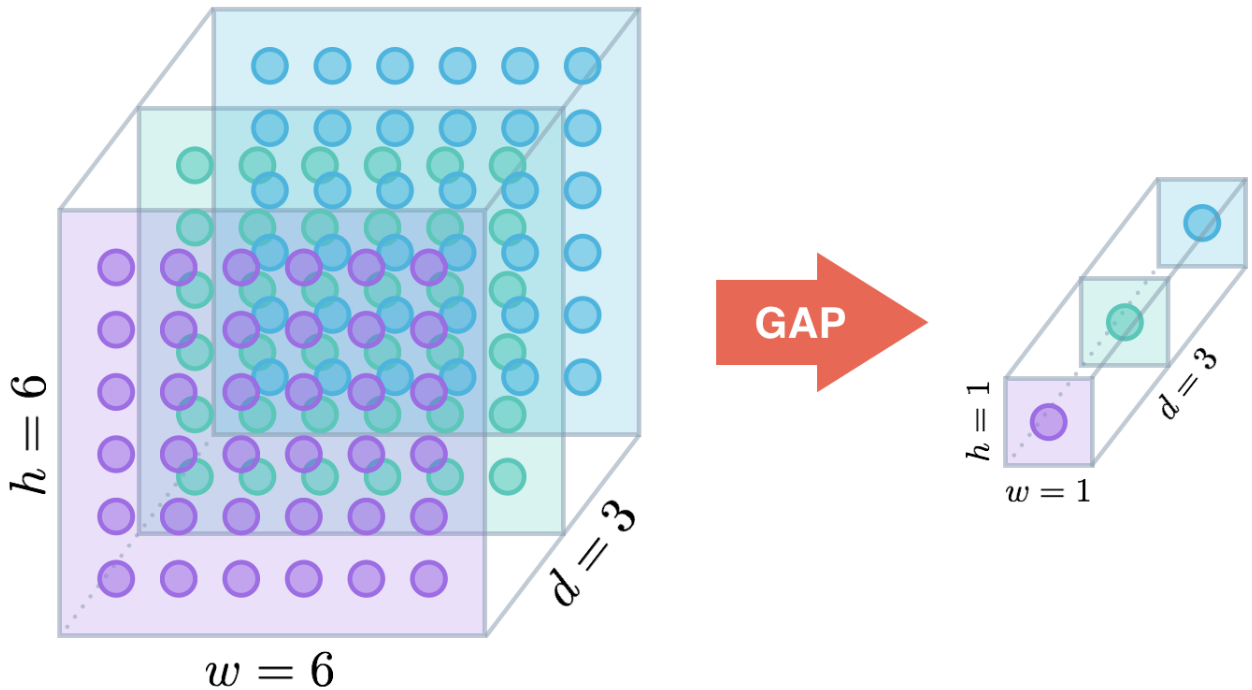
$$Mult = F_r F_r D_r \times W_{r+1} H_{r+1} \times D_{r+1}$$

- Number of computations needed to generate all the activations in Dense Feed Forward layer (r+1) $= D_r D_{r+1}$

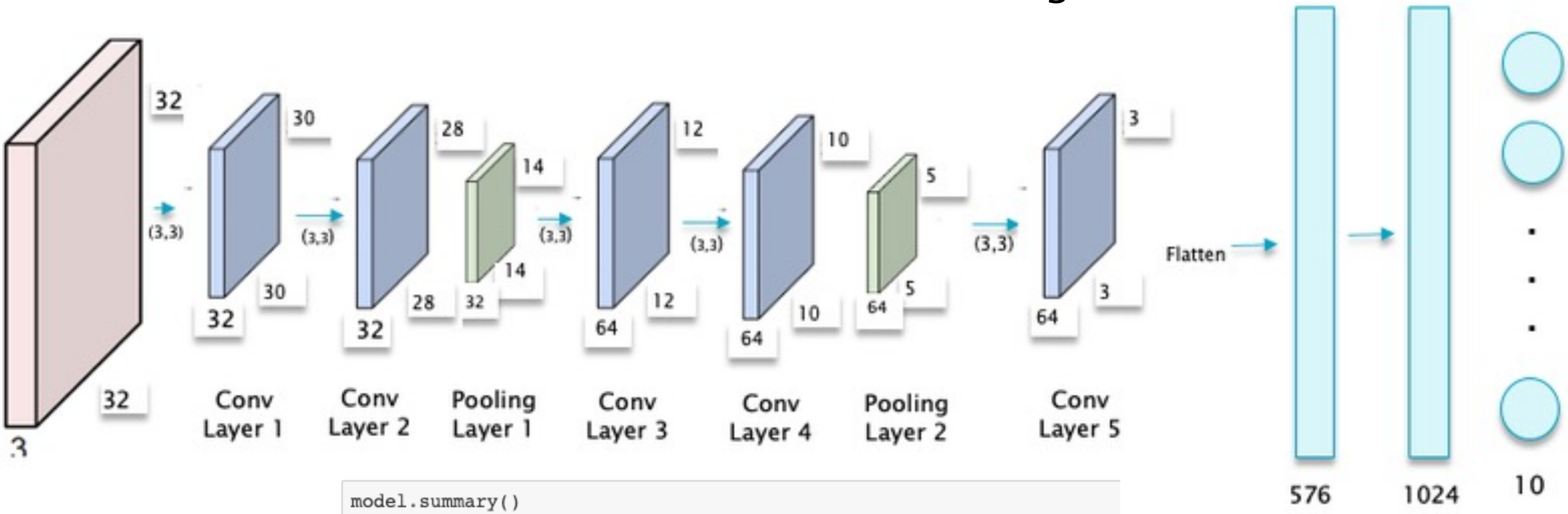
- ConvNet computations greater by a factor of $F_r F_r W_{r+1} H_{r+1}$!!

Global Max Pooling

Global Max Pooling



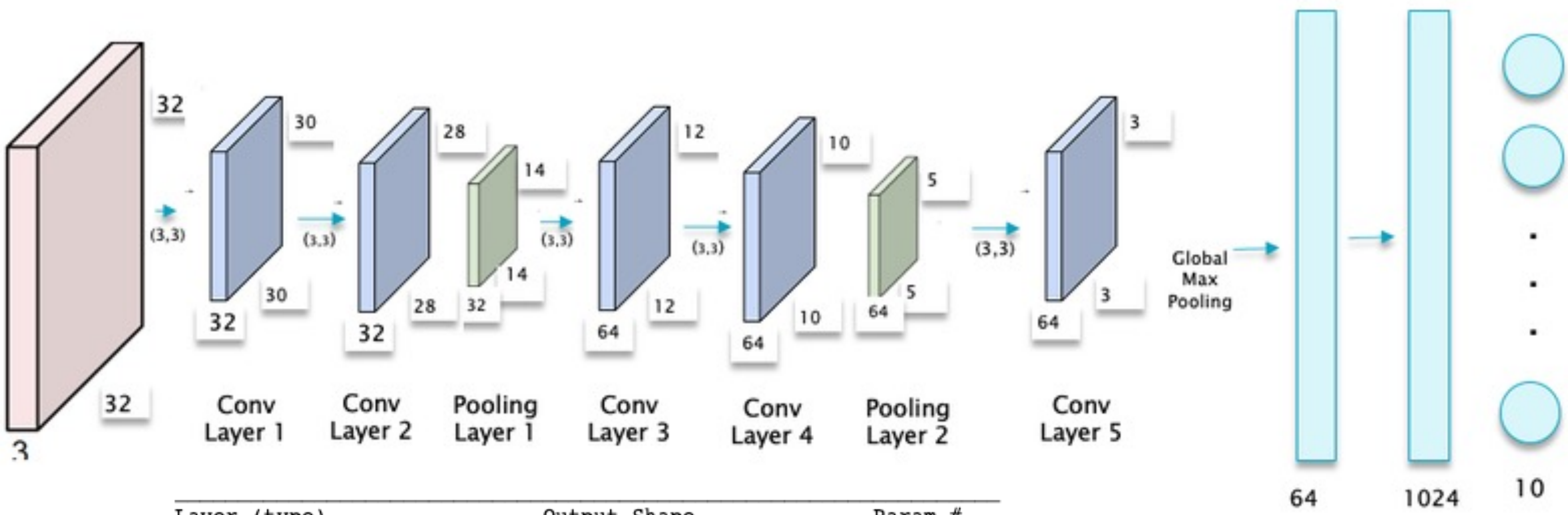
Without Global Max Pooling




```
model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              590848
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250
=================================================================
Total params: 703,594
Trainable params: 703,594
Non-trainable params: 0
_________________________________________________________________
```

With Global Max Pooling



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 32)	896
conv2d_2 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_3 (Conv2D)	(None, 12, 12, 64)	18496
conv2d_4 (Conv2D)	(None, 10, 10, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 64)	36928
global_max_pooling2d_4 (Glob	(None, 64)	0
dense_7 (Dense)	(None, 1024)	66560
dense_8 (Dense)	(None, 10)	10250

Total params: 179,306
Trainable params: 179,306
Non-trainable params: 0

Transfer Learning

Example from F. Chollet:
“8.3-using-a-pretrained-convnet”

Transfer Learning: Motivation

Modern CNNs have tens of millions of parameters

They correspondingly need very large Training Datasets

Example: ImageNet has 1 Million images

Training can be very expensive and time consuming

Transfer Learning: Motivation

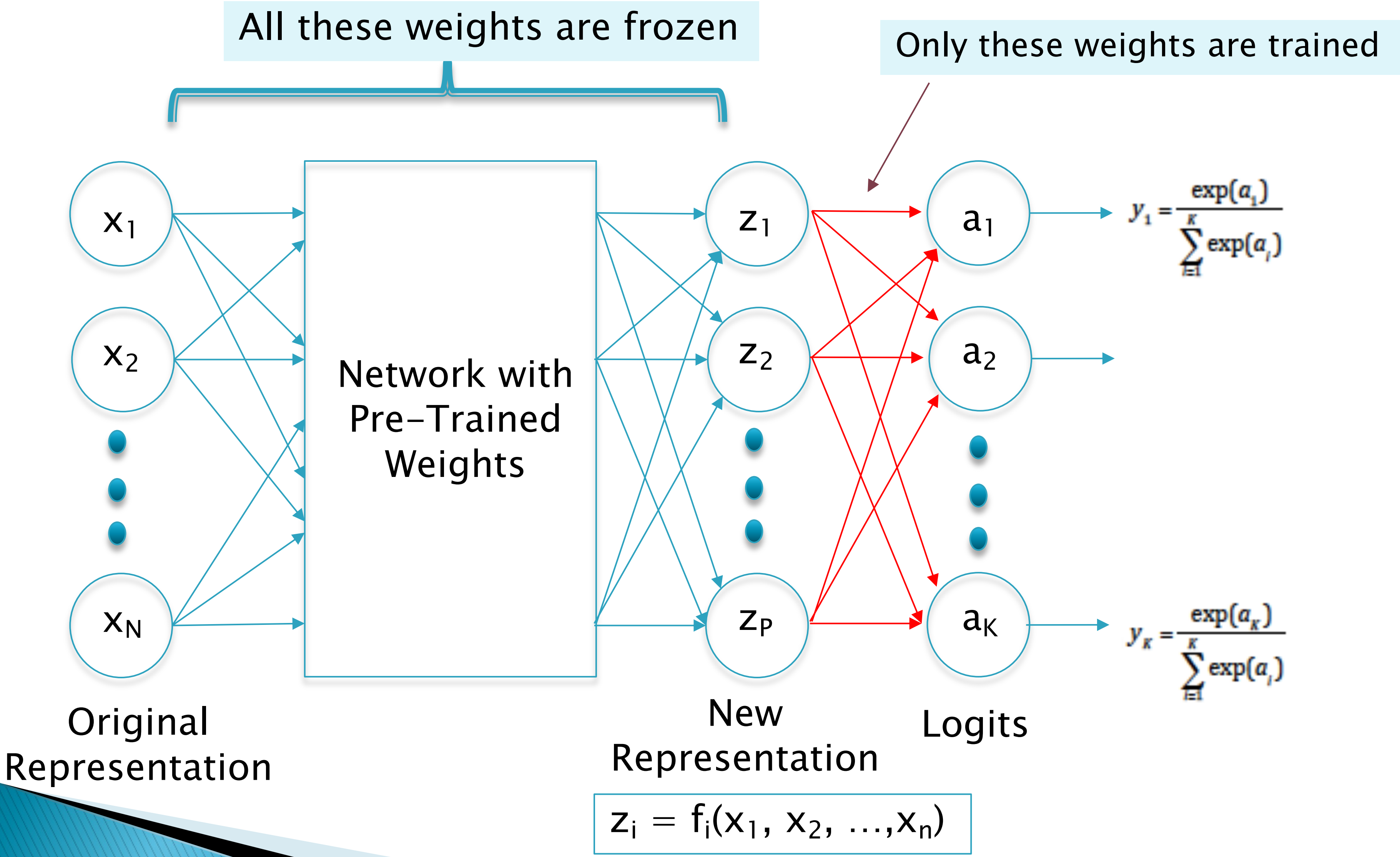
What if we use Small Dataset instead?

Small Dataset and Large Model → Overfitting

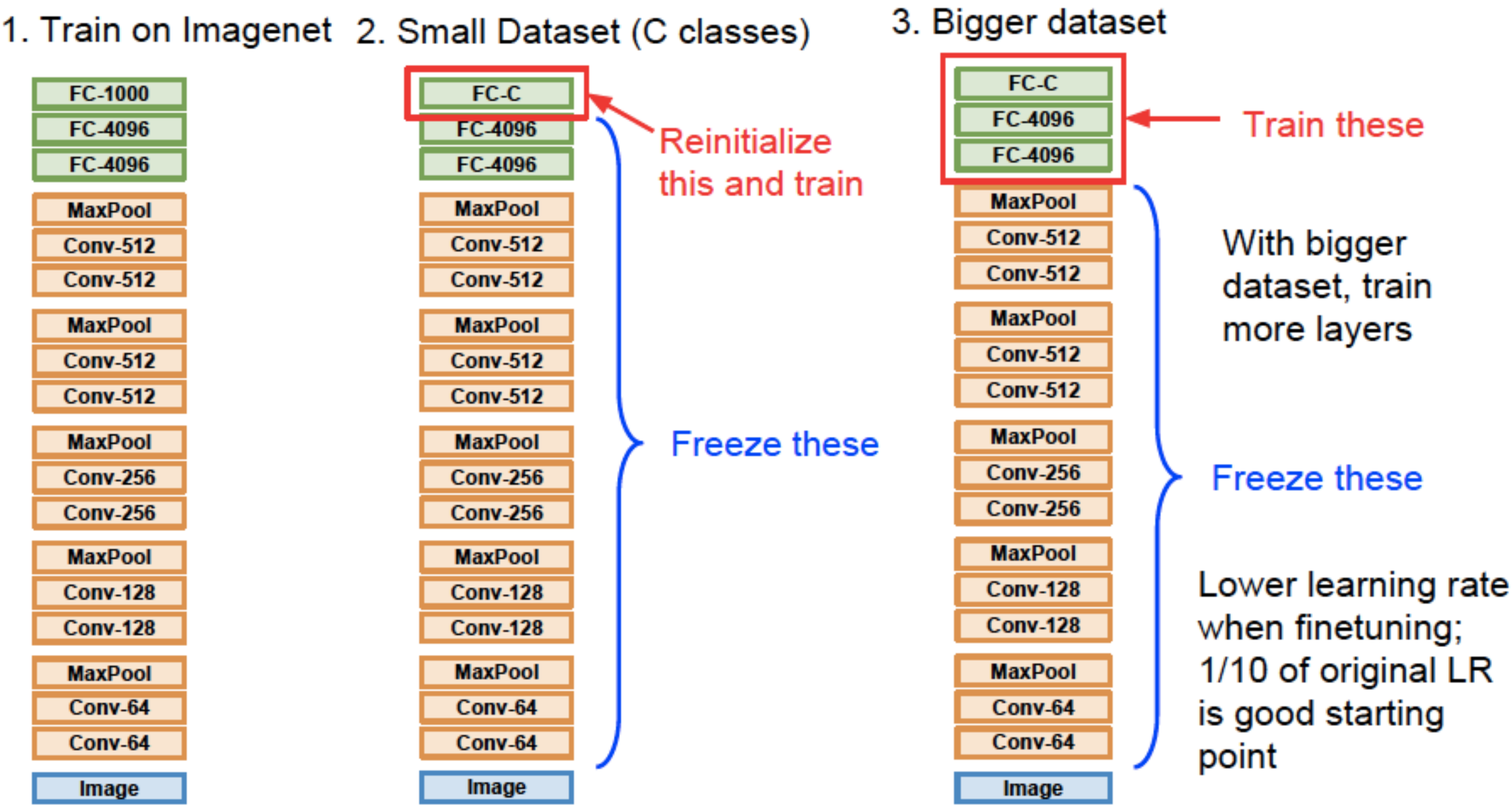
Solution: Transfer Learning

Reduces the number of Parameters
(and training time)

Transfer Learning



Transfer Learning Using ConvNets



Why Transfer Learning Works?

Why does Transfer Learning work so well?

- A CNN trained on a large dataset such as ImageNet, learns to recognize generic patterns and shapes that also occur in non-ImageNet data.
  - Even non-image data, such as audio wave signals, can be classified well with the patterns that are learnt with ImageNet.

- A general rule of thumb is that the earlier Hidden Layers in the CNN contain the more generic portion that can be re-used in other contexts, and the model becomes more and more specific to the particular dataset, as we go deeper into the network.

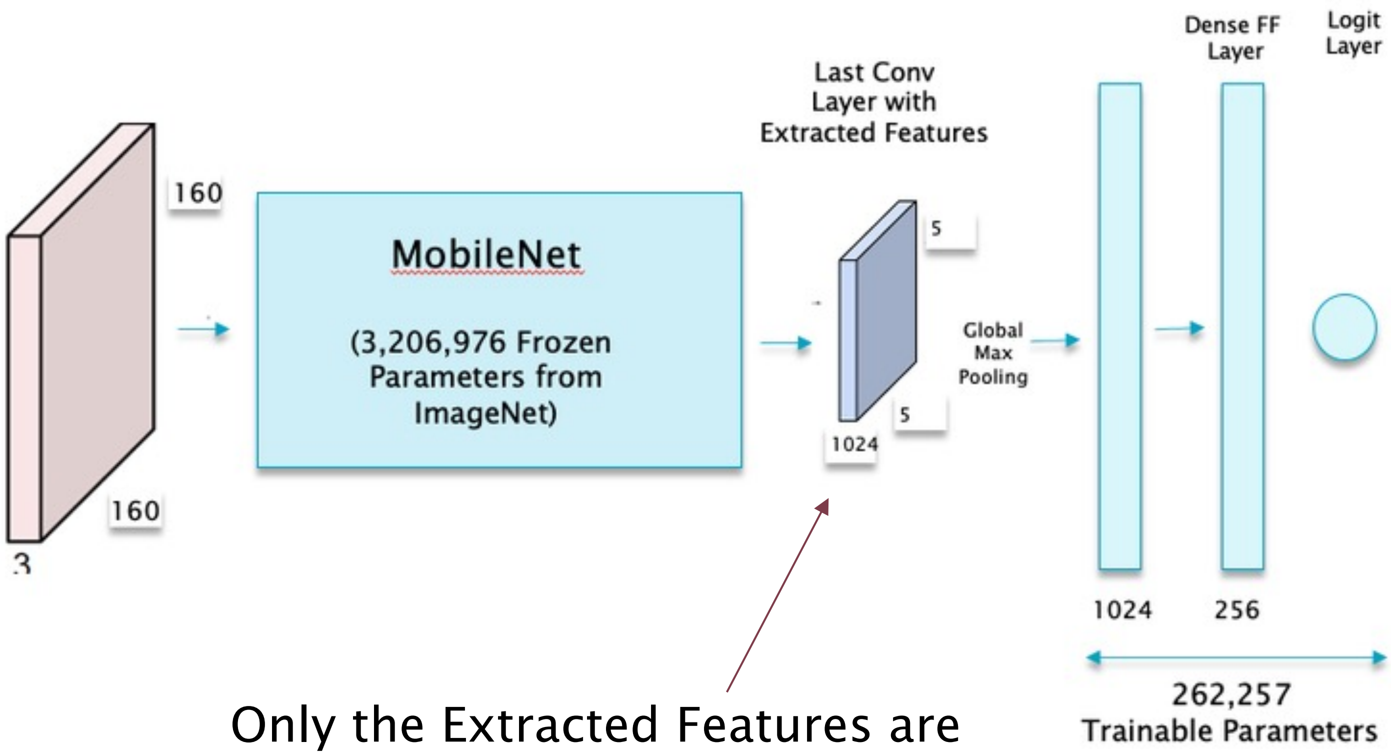
Transfer Learning in Keras

Feature Extraction: Only the Dense part is Trained. Two Methods

1. Method 1: Fast Feature Extraction Without Data Augmentation (Chollet Ch. 8, p 289)
2. Method 2: Feature Extraction Together with Data Augmentation (Chollet Ch. 8, p. 231)

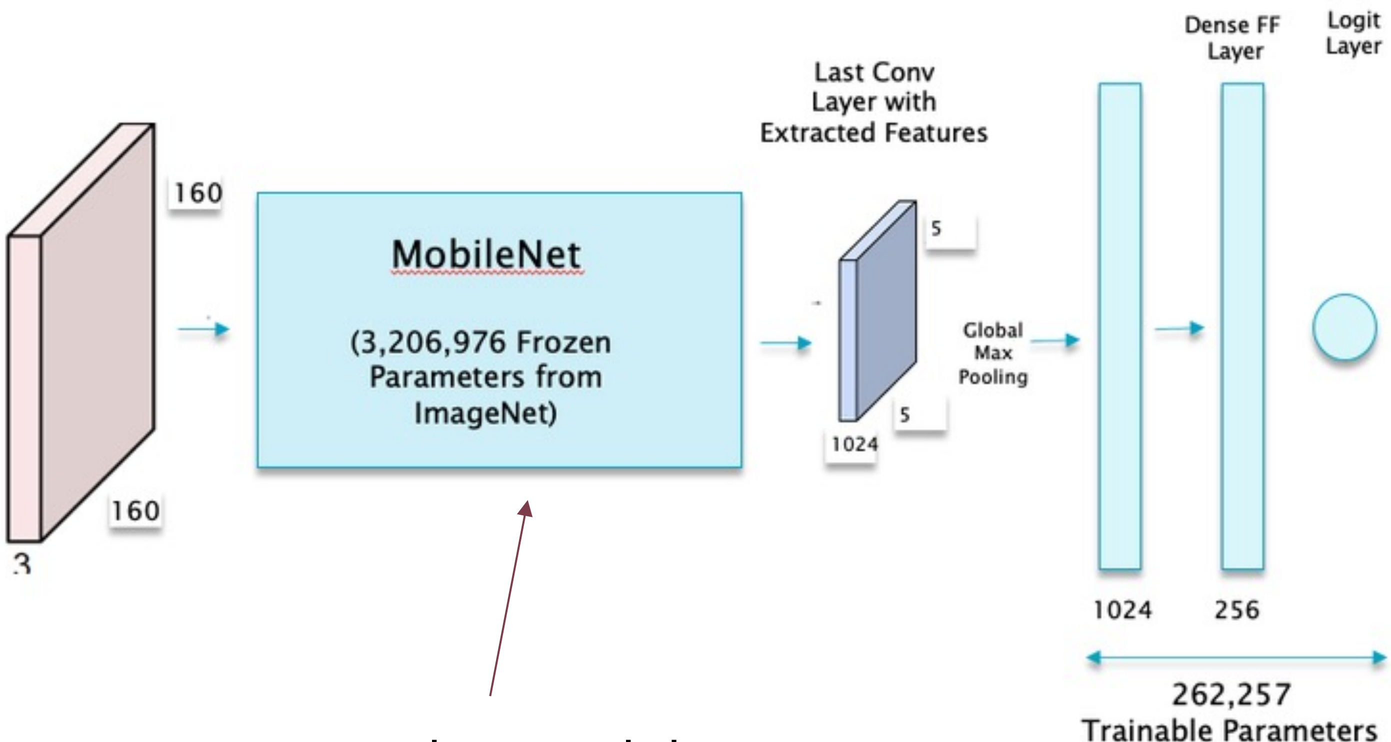
Fine Tuning: Several Convolutional Layers at the top of the network are trained

Fast Feature Extraction: Method 1



Only the Extracted Features are used for Training

Fast Feature Extraction: Method 2



Entire Network is used during Training
with the Base Model Frozen

Fast Feature Extraction : Method 1

Read in the Base Model

```
from tensorflow.keras.applications import MobileNet

conv_base = MobileNet(weights='imagenet',
                      include_top=False,
                      input_shape=(160, 160, 3))
```

Create a 'New' Dataset by passing existing Data through the Base Model

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(160, 160),
    batch_size=20)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(160, 160),
    batch_size=20)

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = tensorflow.keras.applications.mobilenet.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
```

Fast Feature Extraction : Method 1

```
inputs = keras.Input(shape=(5, 5, 1024))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

history = model.fit(
    train_features, train_labels,
    epochs=100,
    validation_data=(val_features, val_labels)
    )
```

Fast Feature Extraction : Method 2

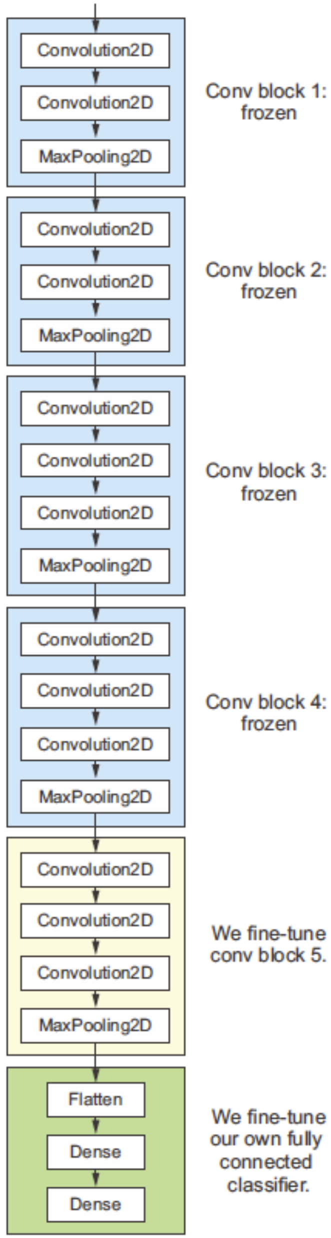
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)                          <- Apply data augmentation.
x = keras.applications.vgg16.preprocess_input(x)       <- Apply input value scaling.
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Transfer Learning with Fine Tuning

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```



Further Reading

- Chapter 12 (Sections 12.2 to 12.7) of "Deep Learning" by Das and Varma
- Chollet Chapter, Section 8.2, 8.3