Convolutional Neural Network

Reference: https://www.coursera.org/learn/convolutional-neural-networks/home/welcome

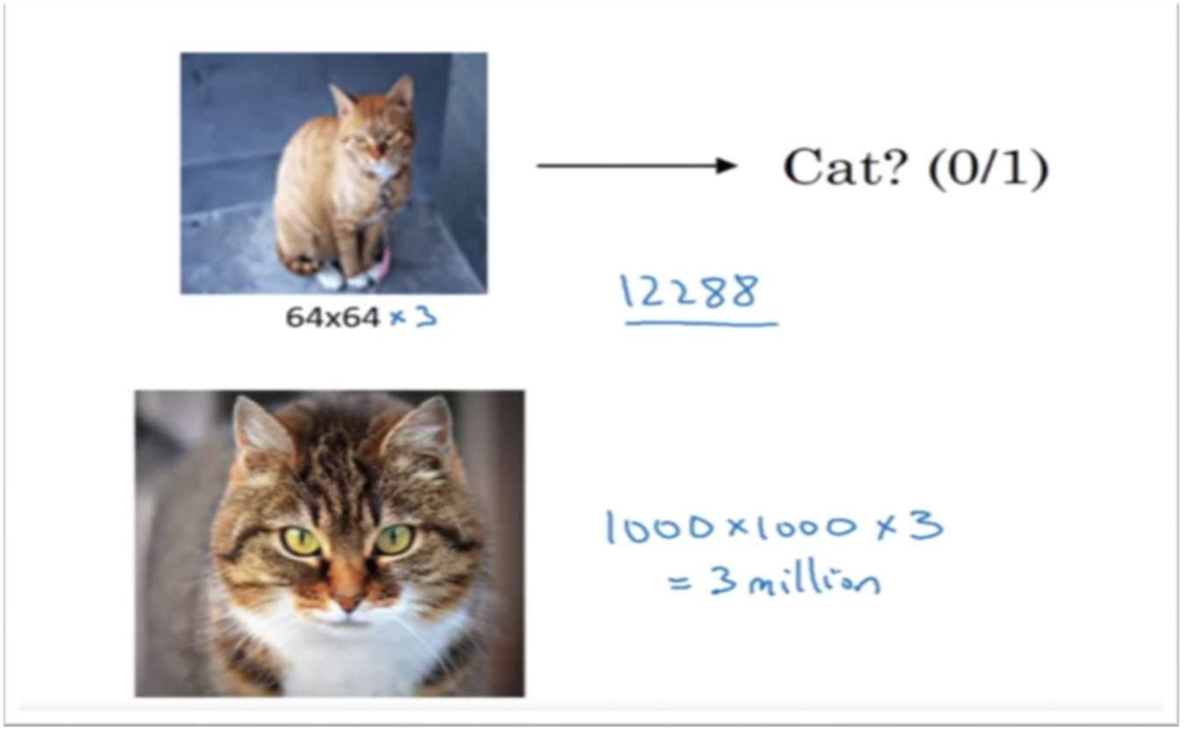
Application

Image Classification

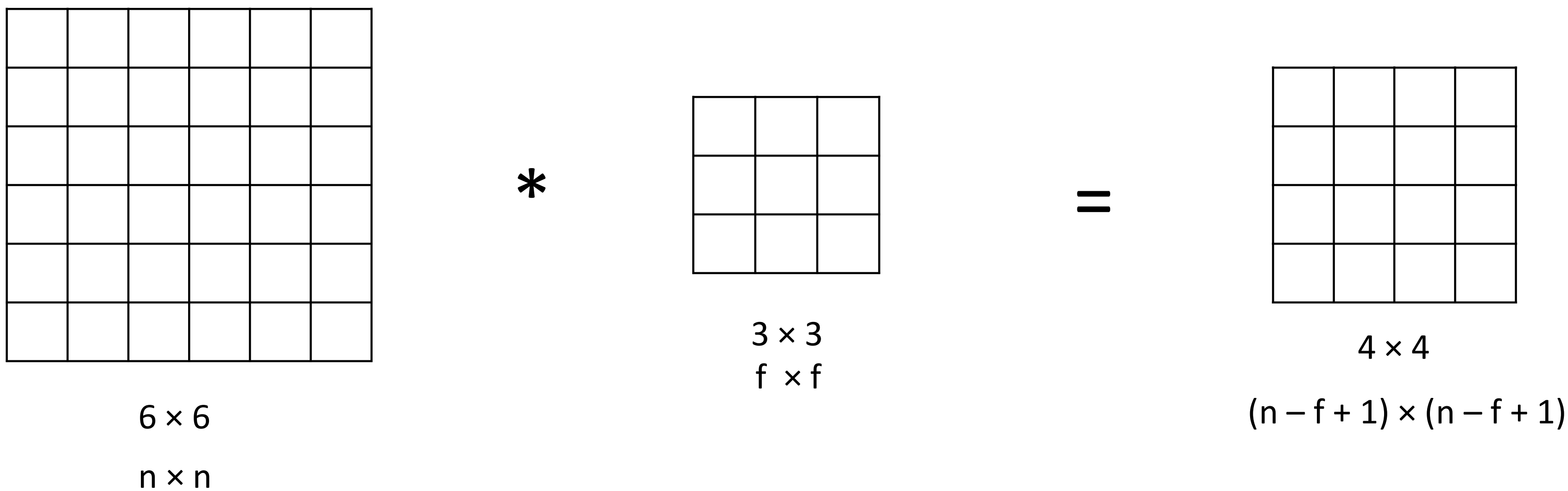
Object Detection

Neural Style transfer

Why not NN?

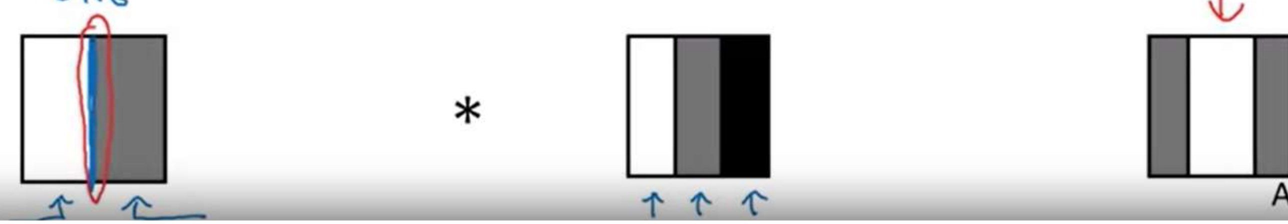


Convolution



Vertical edge detection

$$\underset{6 \times 6}{\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix}} * \underset{3 \times 3}{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}} = \underset{4 \times 4}{\begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}}$$

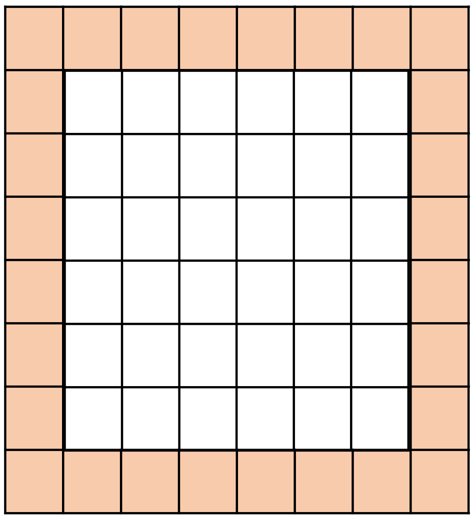


Padding

Valid Conv. = no padding

Same Conv. = pad so that output size is the same as input size.

$p=(f-1)/2$

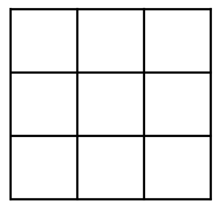


$n \times n$

8×8

Padding (p) = 1

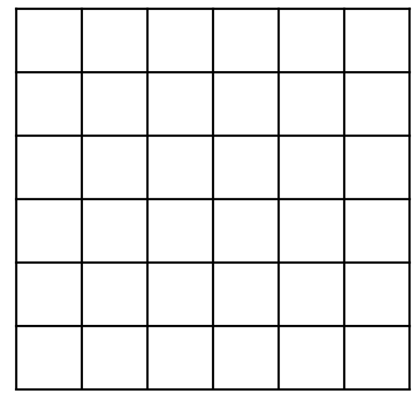
$*$



$f \times f$

3×3

$=$



6×6

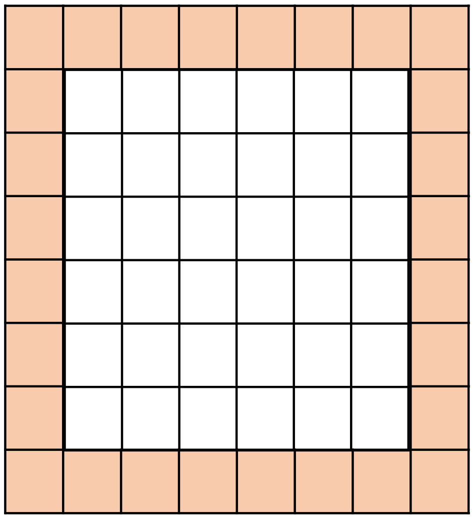
$(n + 2p - f + 1) \times (n + 2p - f + 1)$

Why padding?

- Throwing away information from edges.

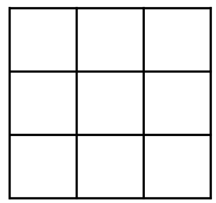
- Shrink the image every time after convolution.

Strided Convolution



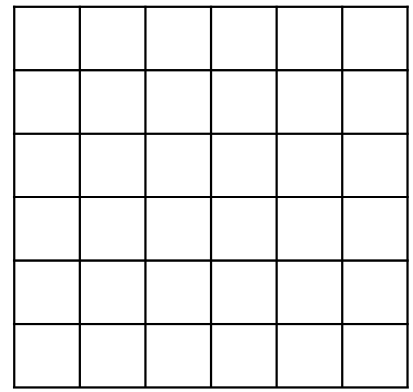
n × n
8 × 8
Padding (p) = 1
Stride (S) = 2

$*$



f × f
3 × 3

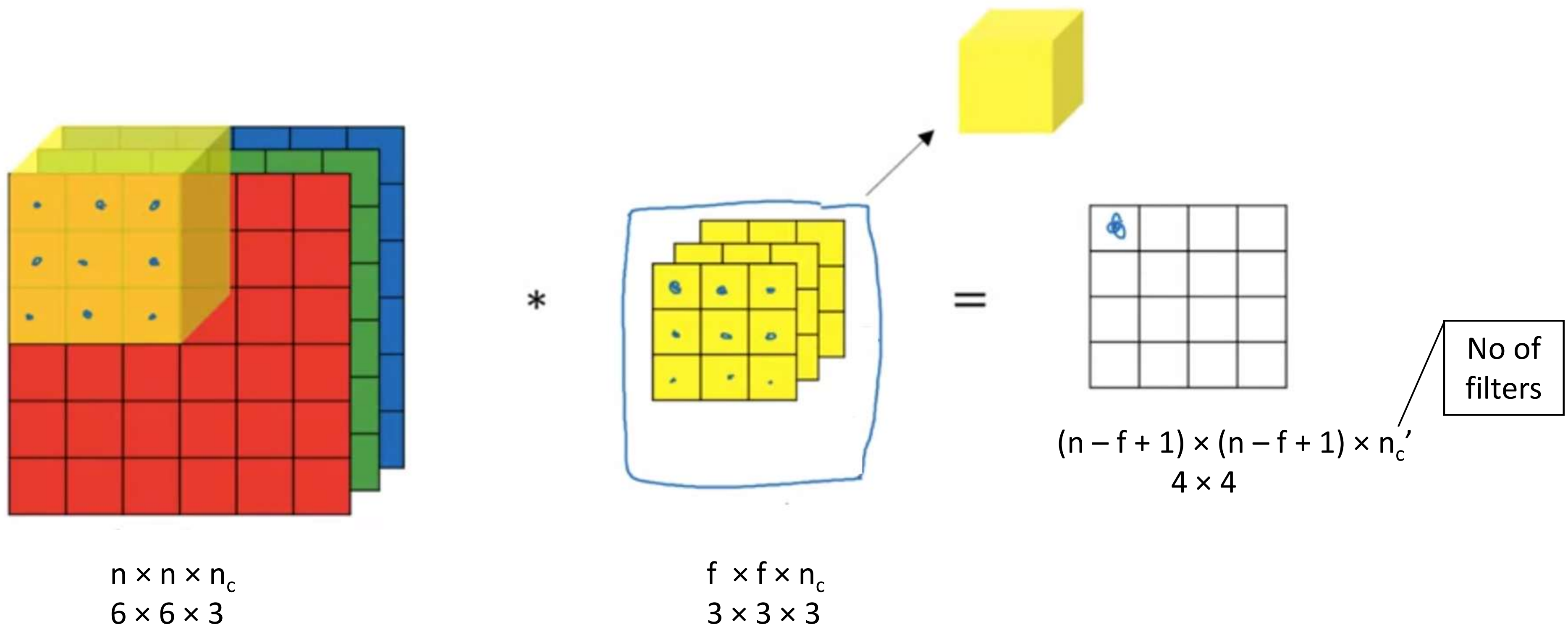
$=$

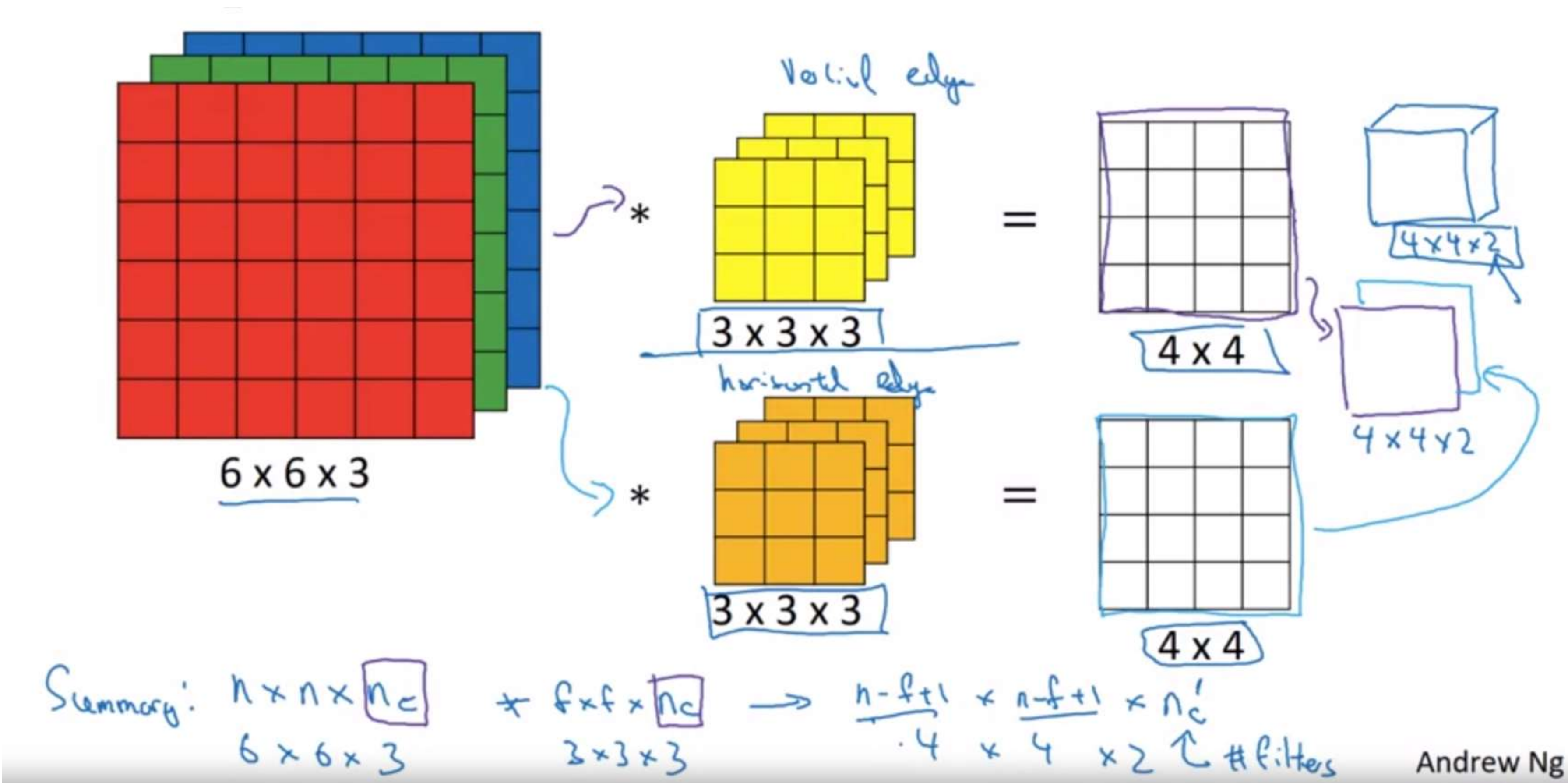


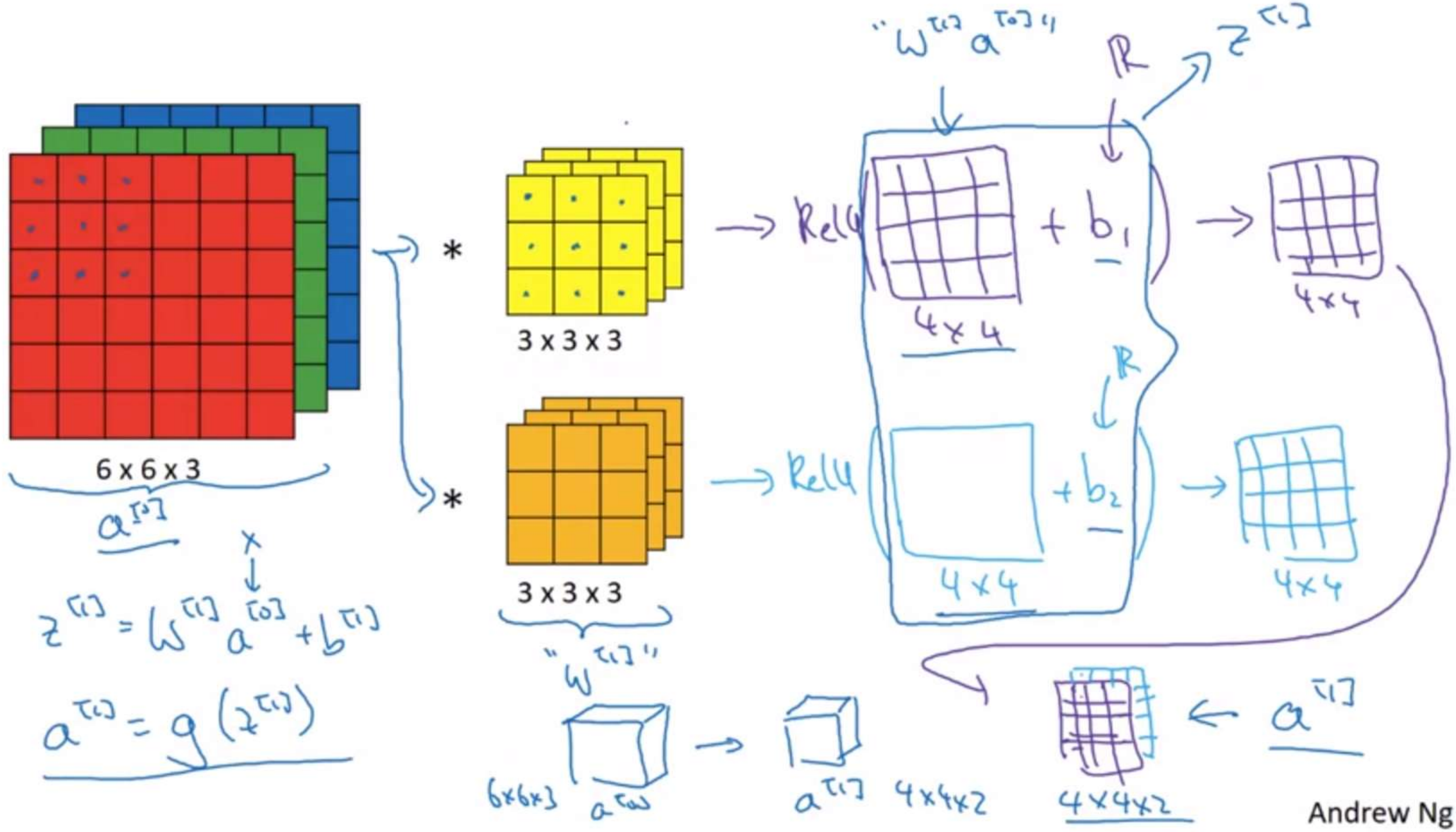
6 × 6

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor$$

Convolution on RGB image




Vertical edges
horizontal edges
6 x 6 x 3
3 x 3 x 3
3 x 3 x 3
4 x 4
4 x 4
4 x 4 x 2
4 x 4 x 2
Summary: n x n x nc * f x f x nc → n-f+1 x n-f+1 x nc'
6 x 6 x 3
3 x 3 x 3
4 x 4 x 2
filters
Andrew Ng



6 x 6 x 3
3 x 3 x 3
3 x 3 x 3
4x4
4x4
4x4x2
Andrew Ng

Summary of Notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$\boxed{n_c^{[l]}}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times \boxed{n_c^{[l-1]}}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]}$ — $(1,1,1,n_c^{[l]})$ ← #filters in layer l.

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times \boxed{n_c^{[l-1]}}$ ←

Output: $n_H^{[l]} \times n_W^{[l]} \times \boxed{n_c^{[l]}}$ ←

$$n_{HW}^{[l]} = \left\lfloor \frac{n_{HW}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

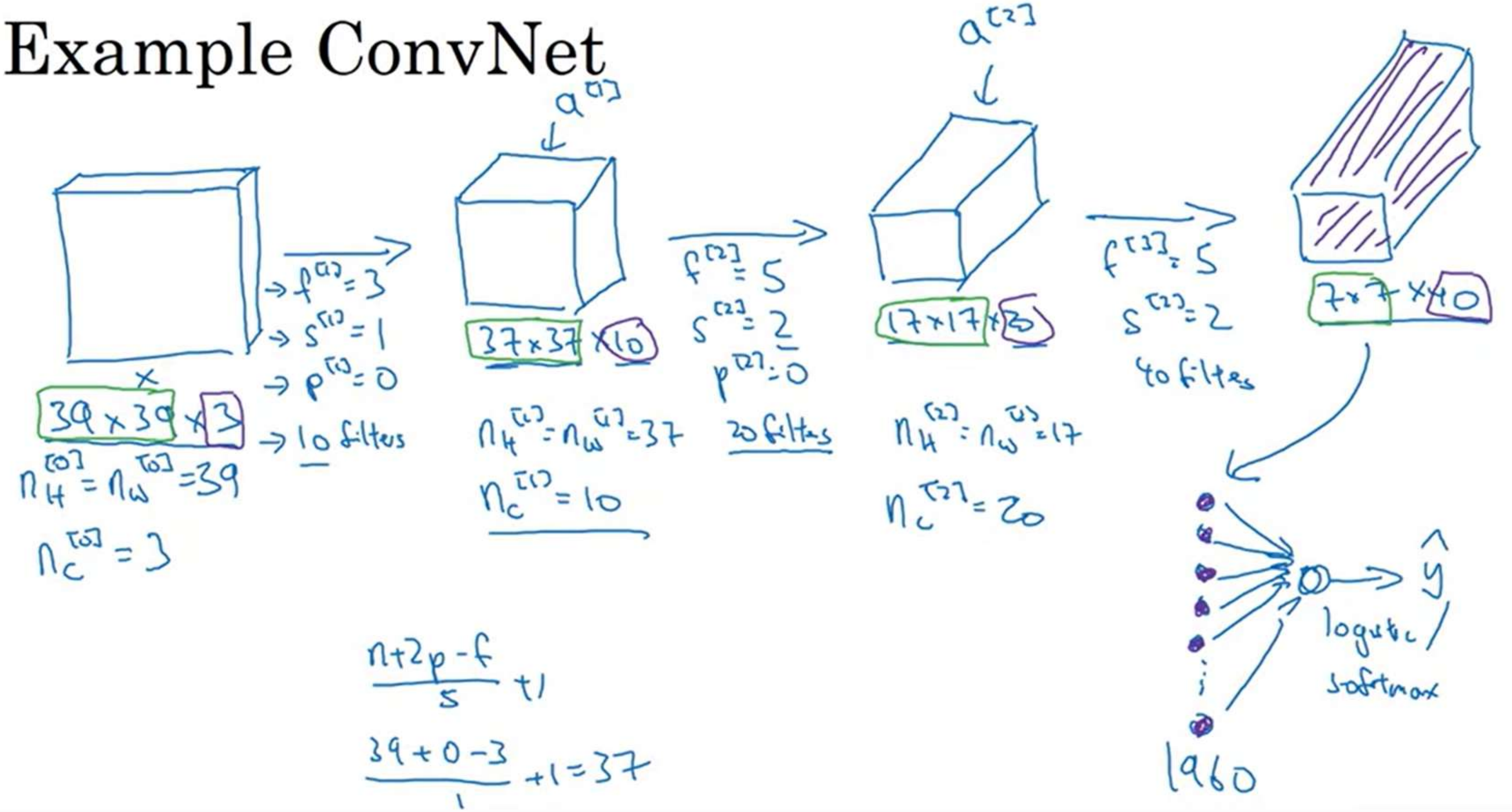
$A^{[l]} \rightarrow m \times \underbrace{n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}}$

$n_c \times n_H \times n_W$

ConvNet

	Shape
Input	(39,39,3)
Conv1 (f=3, s=1, p=0), 10-filters	
Conv2 (f=5, s=2, p=0) 20-filters	
Conv3 (f=5, s=2, p=0) 40-filters	
Flatten	

Example ConvNet



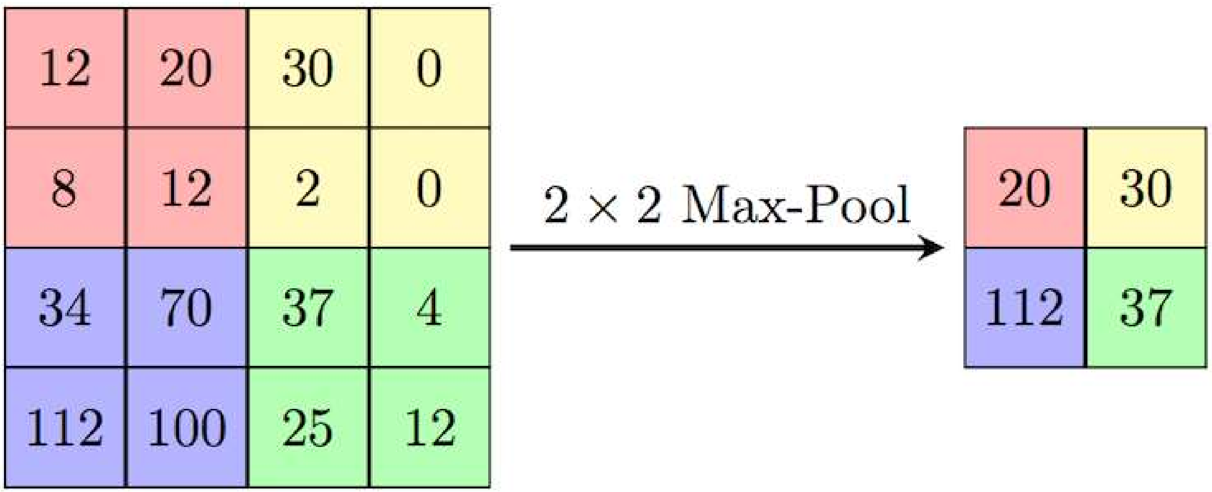
Andrew Ng

Types of Layer in ConvNets

- Convolution (Conv)
- Pooling (pool)
- Fully Connected (FC)

Pooling

Max pooling



1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

$5 \times 5 \times \cancel{1}\ n_c$

9	9	5
9	9	5
8	6	9

$3 \times 3 \times \cancel{1}\ n_c$

$\boxed{\frac{f=3}{s=1}}$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

No parameters to learn in pooling

Summary of pooling

Hyperparameters:

f : filter size

s : stride

Max or average pooling

$f=2, s=2$

$f=3, s=2$

→ ~~p: padding.~~

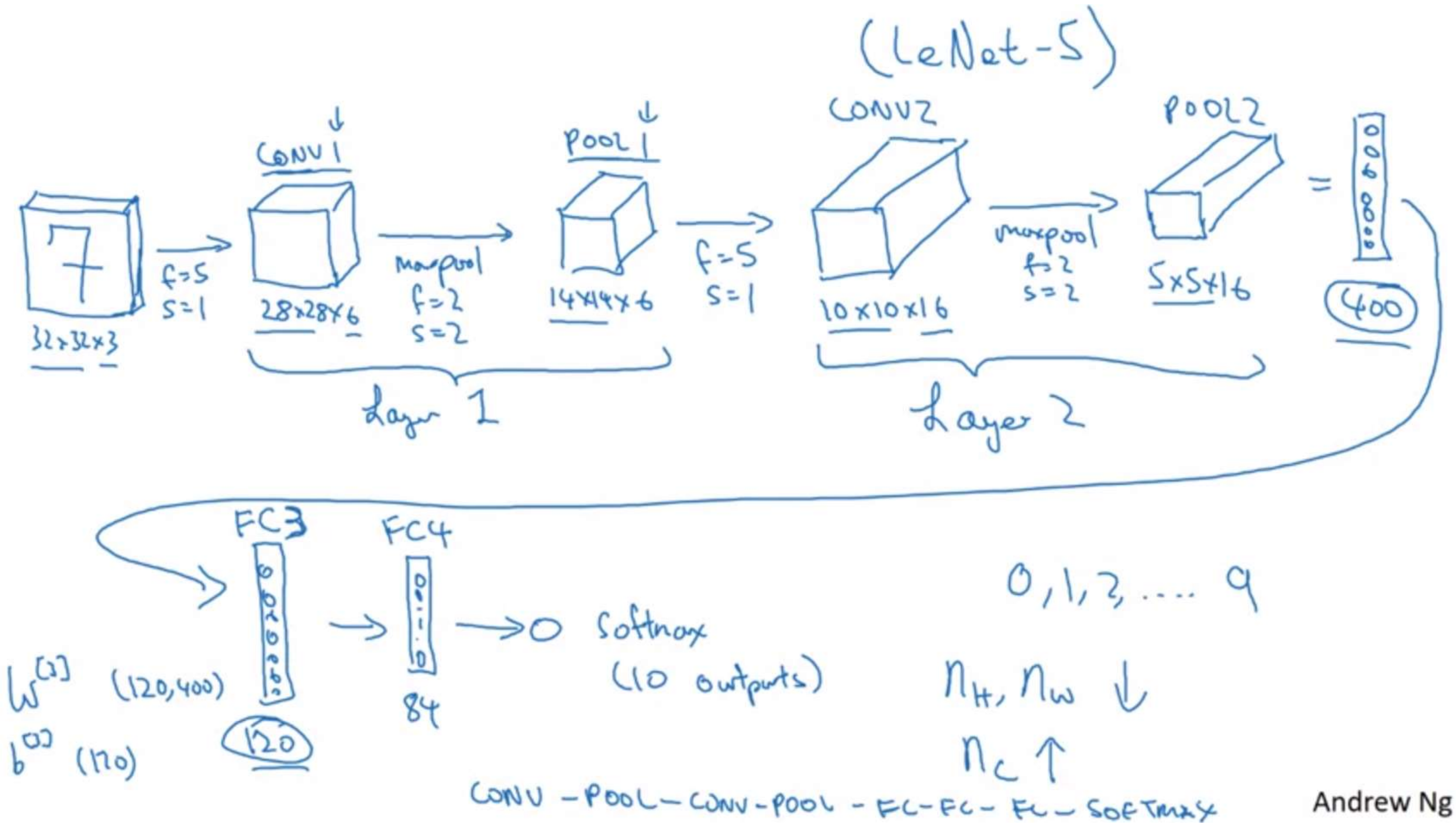
No parameters to learn!

$n_H \times n_W \times n_C$

$\downarrow$

$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_C$

LeNet-5


(LeNet-5)
CONV1
POOL1
CONV2
POOL2
7
32×32×3
f=5
s=1
28×28×6
maxpool
f=2
s=2
14×14×6
f=5
s=1
10×10×16
maxpool
f=2
s=2
5×5×16
400
Layer 1
Layer 2
FC3
FC4
120
84
Softmax
(10 outputs)
W[3] (120,400)
b[3] (120)
0, 1, 2, ... 9
nH, nW ↓
nC ↑
CONV - POOL - CONV - POOL - FC - FC - FC - SOFTMAX
Andrew Ng

	Shape	Parameters
Input	(32,32,3)	-
Conv1 (f=5, s=1), 8-filters (3D-Filters)	(28,28,8)	608
Pool1	(14,14,8)	-
Conv2 (f=5, s=1) 16-filters	(10,10,16)	3216
Pool2	(5,5,16)	-
FC3	120	48120
FC4	84	10164
Output-SoftMax	10	850

List of Parameters and Hyperparameters

Hyperparameters: Which needs to be tuned, Usually with validation set

1. Kernel size - 3 x 3, 5 x 5, 7 x 7
2. Stride - 1, 2 or 3
3. Padding size - 0, 1, 2
4. Number of filters - 32, 64, 128, etc..
5. No of hidden layers - Trial and error
6. No of neurons in each hidden layer - Trial and error
7. Learning rate– 0.1, 0.01, 0.001
8. Activation functions
9. Number of iteration – Trial and error
10. No. of Epoch – Trial and error
11. Batch size – 4, 8, 16, 32, 64, 128 (Power of 2)
12. Regularization – Dropout, L1, L2
13. Weights and Bias Initialization – Zero, Random, He
14. Optimization algorithm – SGD, ADAM, RMSProp
15. Learning Rate decay
16. Momentum – 0.9
17. Number of iterations, epoch and batch size

Parameters: Which are learned during training, Weights and bias.

Transfer Learning

- Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.

- Transfer learning with convolutional neural networks (CNNs) has revolutionized the field of computer vision by enabling the reuse of pre-trained models on new, related tasks. This powerful technique leverages the knowledge learned from large-scale datasets, allowing for faster and more accurate model training, even with limited labeled data.

- transfer learning significantly reduces the need for extensive training time and computational resources.

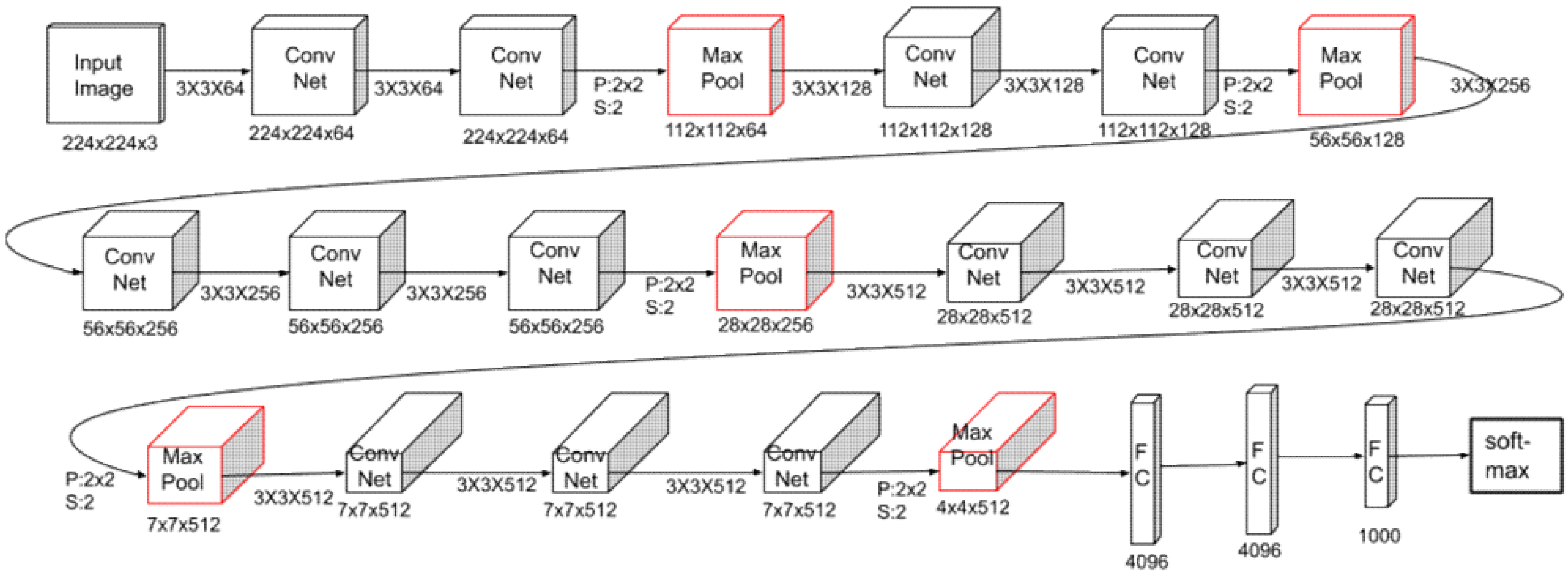
- Transfer learning takes advantage of the fact that CNNs trained on large datasets, such as ImageNet, have learned general features that are relevant to many visual tasks. Instead of training a CNN from scratch on a new dataset, transfer learning involves using a pre-trained CNN as a starting point and fine-tuning it on the new dataset.

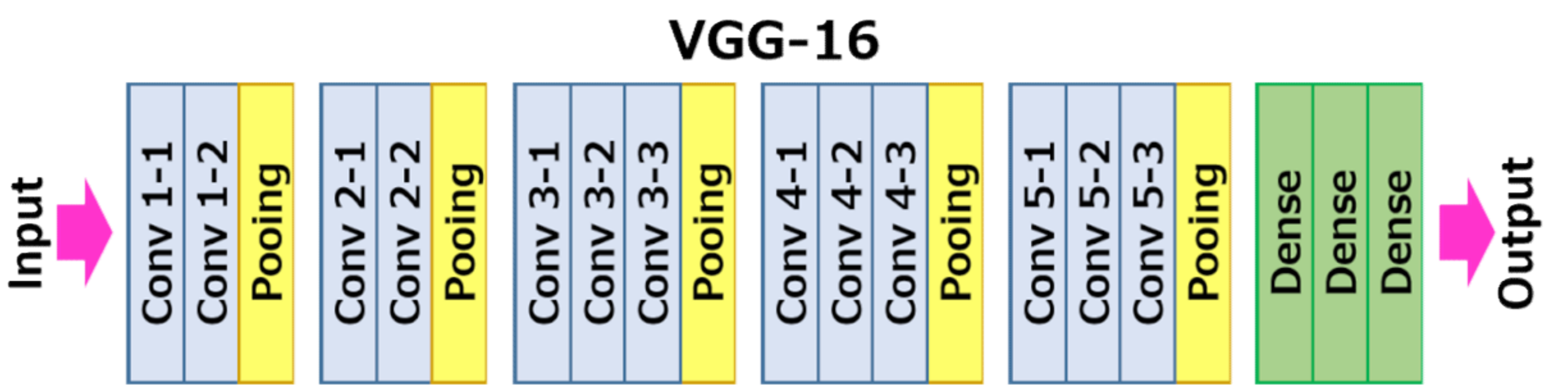
- The pre-trained CNN acts as a feature extractor, capturing high-level visual representations. These features are then passed to new layers designed for the specific task. The pre-trained layers are frozen during fine-tuning, while the new layers are adjusted.

Steps to Implement Transfer Learning

- Select a Pre-trained Model
- Load Pre-trained Model
- Customize the Model
- Freeze Pre-trained Layers
- Prepare Data
- Train the Model
- Fine-tuning (Optional)
- Evaluate and Test

VGG-16


Input Image
224x224x3
3X3X64
Conv Net
224x224x64
3X3X64
Conv Net
224x224x64
P:2x2 S:2
Max Pool
112x112x64
3X3X128
Conv Net
112x112x128
3X3X128
Conv Net
112x112x128
P:2x2 S:2
Max Pool
56x56x128
3X3X256
Conv Net
56x56x256
3X3X256
Conv Net
56x56x256
3X3X256
Conv Net
56x56x256
P:2x2 S:2
Max Pool
28x28x256
3X3X512
Conv Net
28x28x512
3X3X512
Conv Net
28x28x512
3X3X512
Conv Net
28x28x512
P:2x2 S:2
Max Pool
7x7x512
3X3X512
Conv Net
7x7x512
3X3X512
Conv Net
7x7x512
3X3X512
Conv Net
7x7x512
P:2x2 S:2
Max Pool
4x4x512
FC
4096
FC
4096
FC
1000
soft-max


VGG-16
Input
Conv 1-1
Conv 1-2
Pooing
Conv 2-1
Conv 2-2
Pooing
Conv 3-1
Conv 3-2
Conv 3-3
Pooing
Conv 4-1
Conv 4-2
Conv 4-3
Pooing
Conv 5-1
Conv 5-2
Conv 5-3
Pooing
Dense
Dense
Dense
Output

Example for transfer learning

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load CIFAR-10 dataset
(itrain, ltrain), (itest, ltest) = cifar10.load_data()

# Preprocess the data
itrain = itrain / 255.0
itest = itest / 255.0
ltrain = to_categorical(ltrain)
ltest = to_categorical(ltest)
```

```python
# Load pre-trained VGG16 model (excluding the top fully-connected layers)
basem = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the pre-trained layers
for layer in basem.layers:
   layer.trainable = False

# Create a new model on top
semodel = Sequential()
semodel.add(basem)
semodel.add(Flatten())
semodel.add(Dense(256, activation='relu'))
semodel.add(Dense(10, activation='softmax'))  # CIFAR-10 has 10 classes

# Compile the model
semodel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

# Train the model
semodel.fit(itrain, ltrain, epochs=10, batch_size=32, validation_data=(itest, ltest))

# Evaluate the model on test data
ltest, atest = semodel.evaluate(itest, ltest)
print("Test accuracy:", atest)
```

Any Questions

?