Fundamentals of Neural Network

DEEP LEARNING

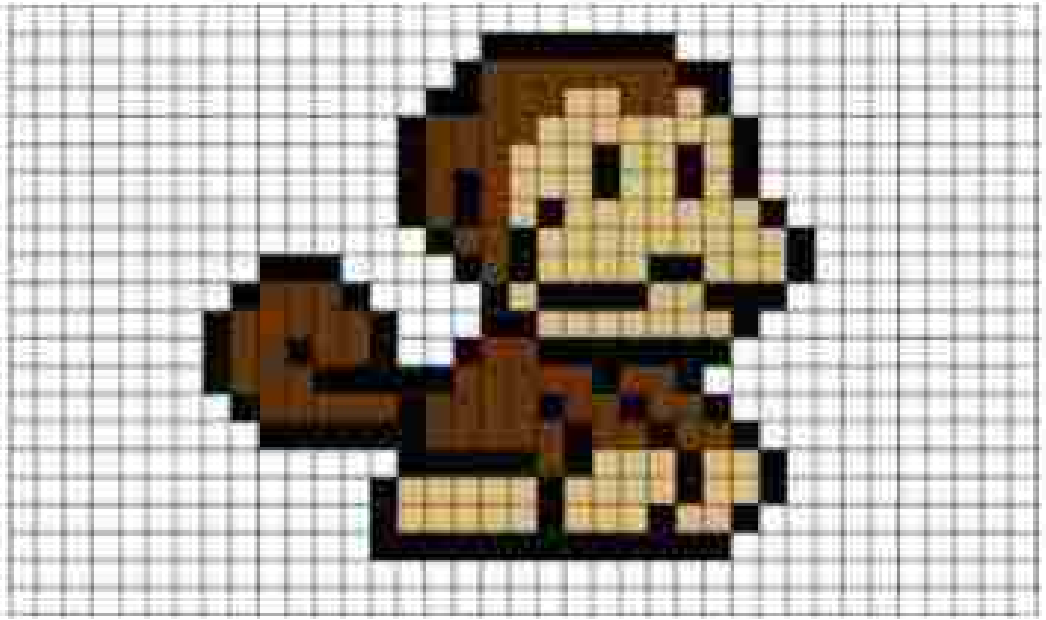
Module No: 04

Convolutional Neural Networks (CNN): Supervised Learning

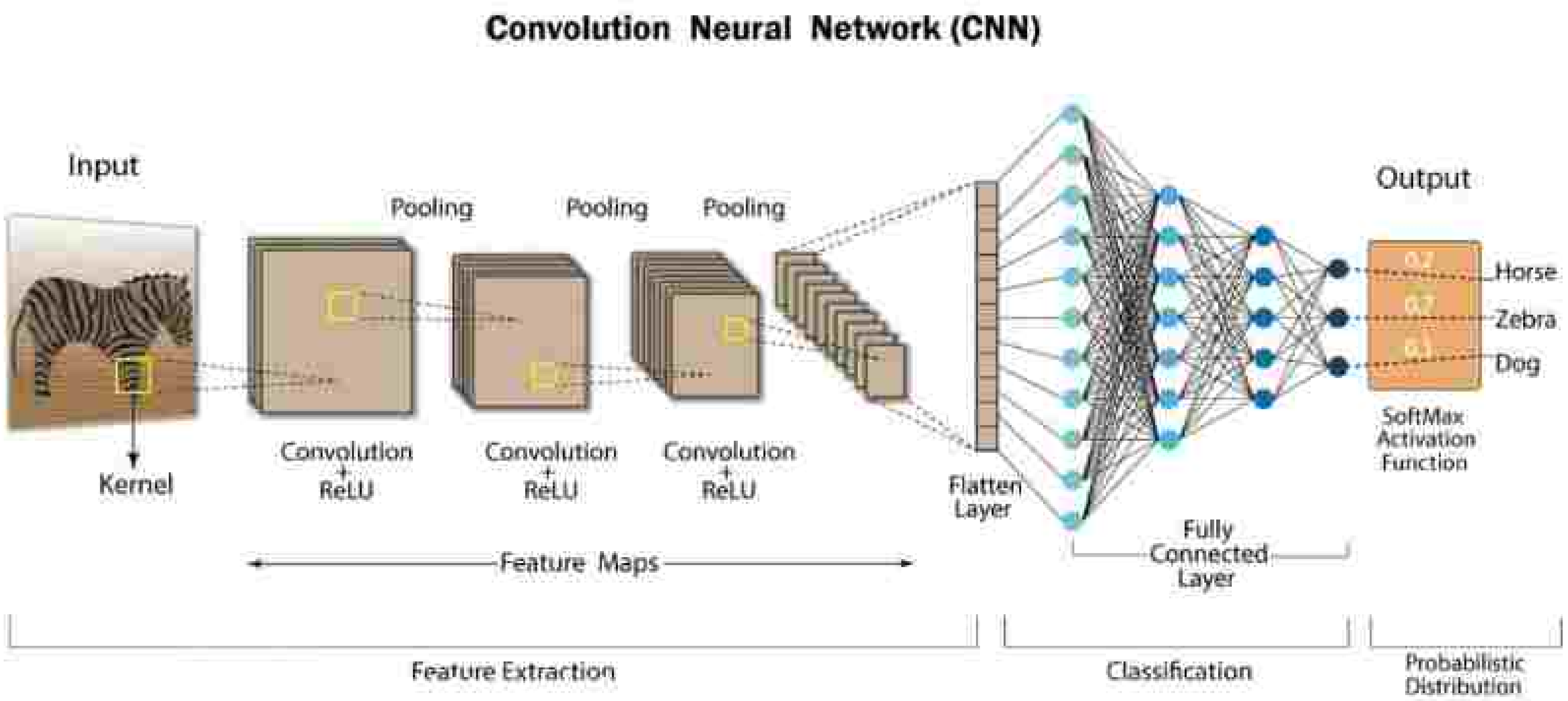
What is CNN

Definition:

- A Convolutional Neural Network (CNN) is a type of neural network that is primarily used for image classification and recognition tasks.
- It is specifically designed to process structured grid data, such as images, *by using a hierarchical pattern recognition method inspired by the organization of the animal visual cortex*.



Deep Learning



What is CNN

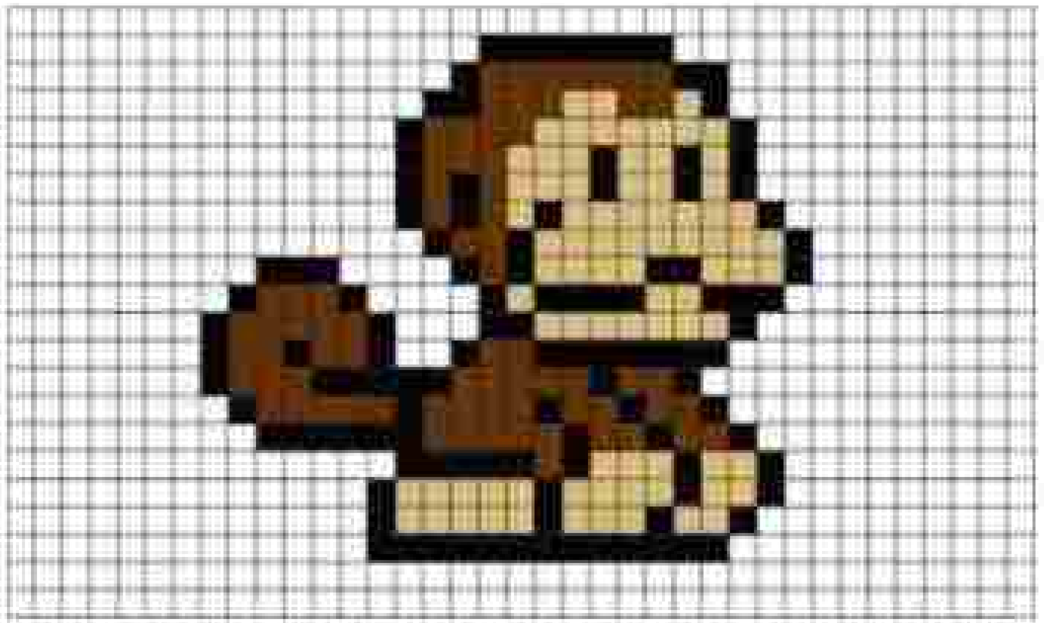
- CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.
- The convolutional layers apply convolution operations to the input image, using filters or kernels to extract various features.
- The pooling layers then downsample the feature maps obtained from the convolutional layers to reduce dimensionality and computational complexity.
- Finally, the fully connected layers perform classification based on the extracted features.

What is CNN

- CNNs are particularly effective for image-related tasks because they can automatically learn and extract hierarchical features from raw pixel data.
- CNN's have achieved remarkable success in various computer vision tasks, such as image classification, object detection, and image segmentation etc.

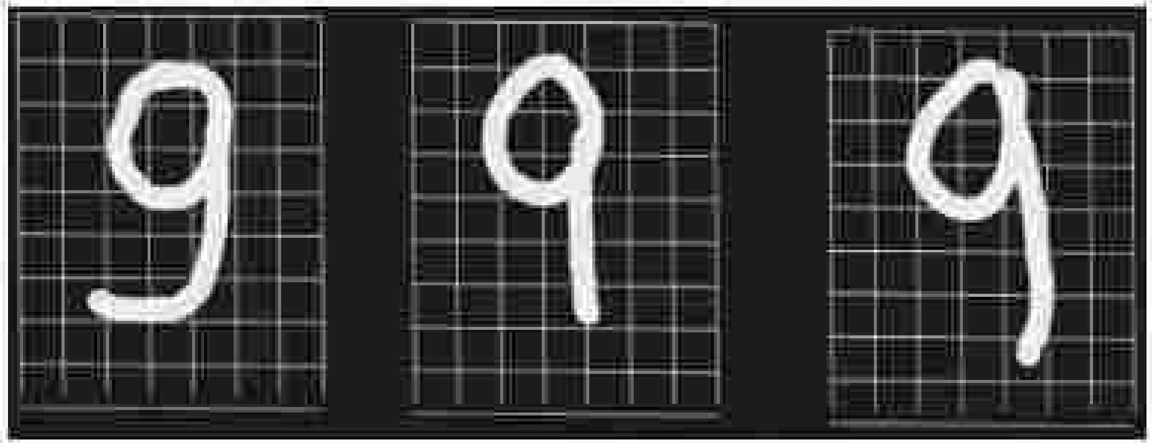
Why not use ANN?

1. High Computational Cost
2. Overfitting
3. Loss of important information like spatial arrangement of pixels



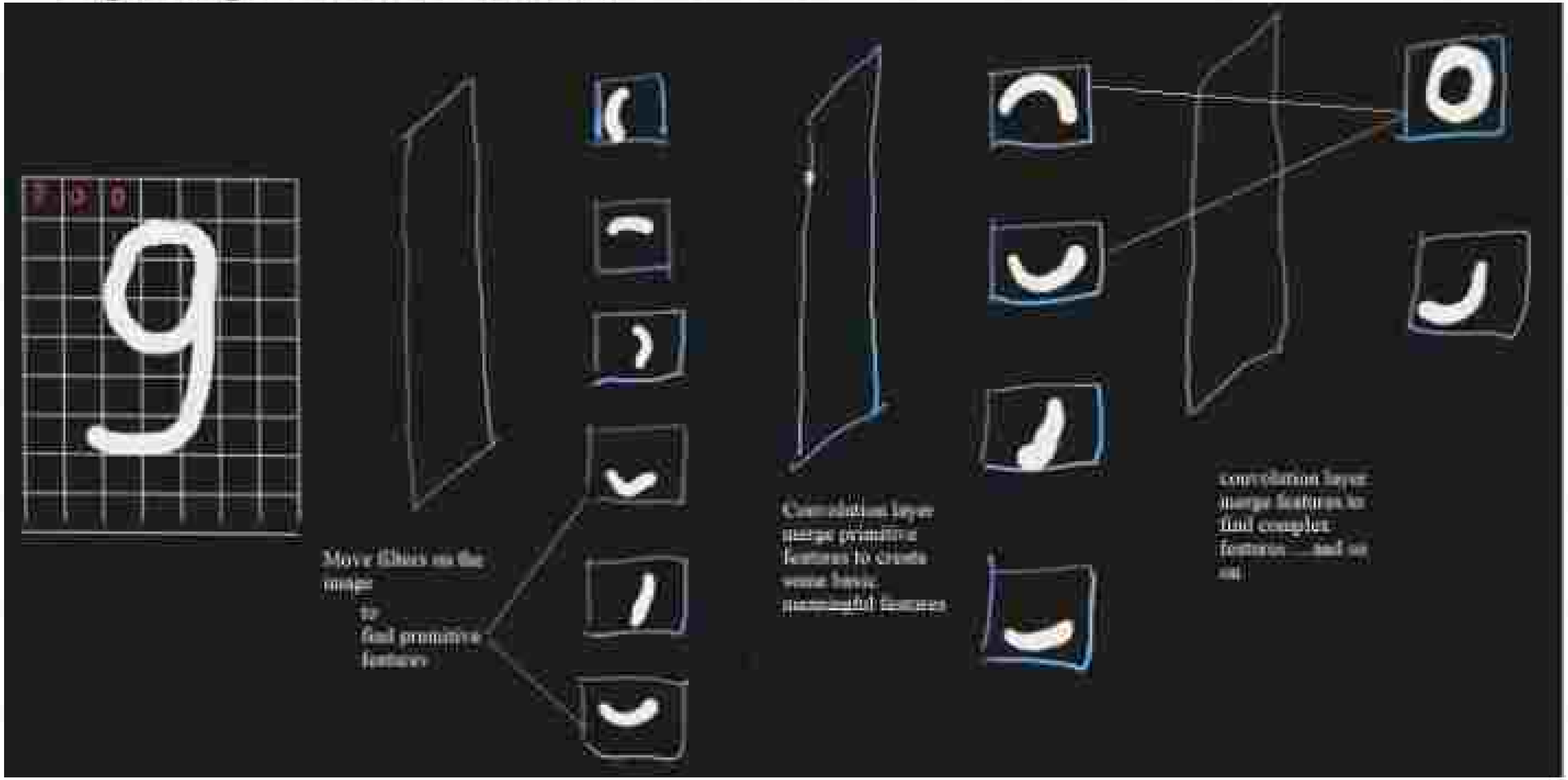
CNN Intuition

e.g. Image classification task



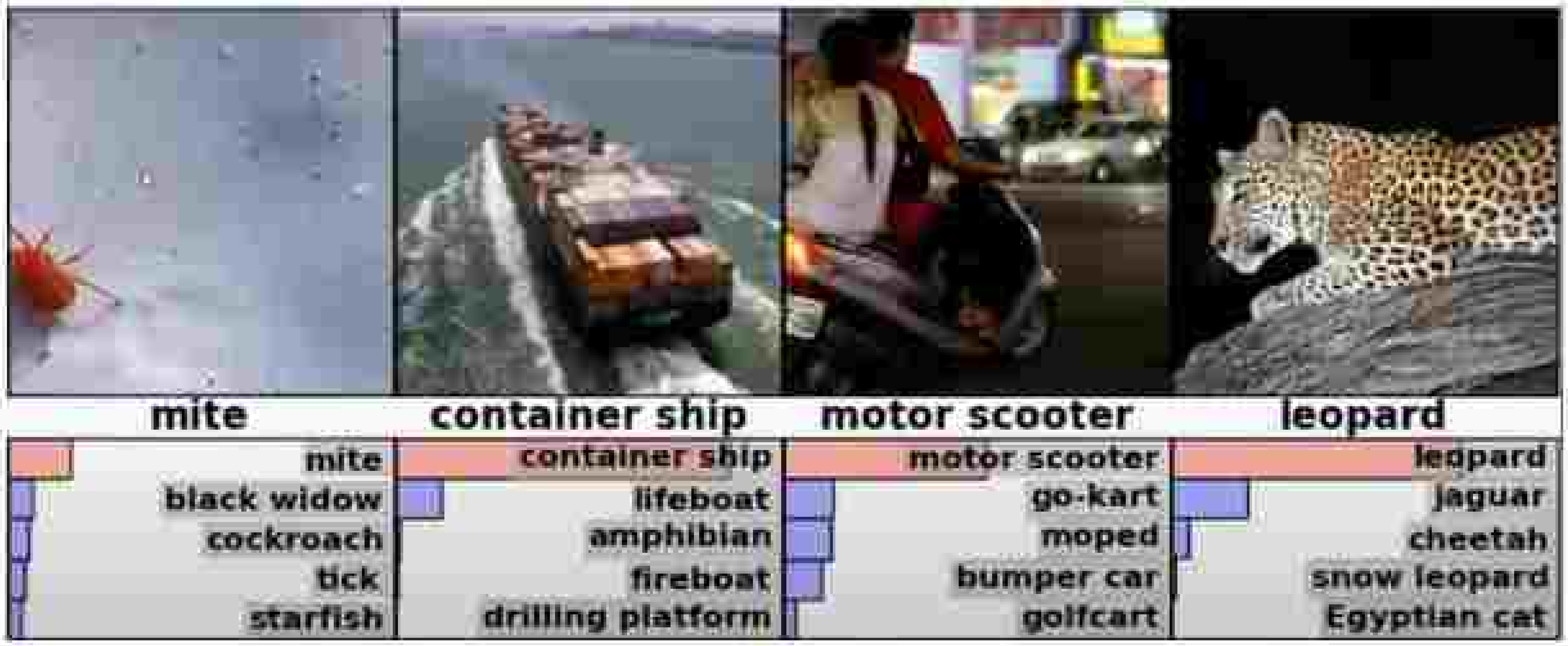
CNN Intuition

e.g. Image classification task


Move filters on the image
to
find primitive features
Convolution layer merge primitive features to create some basic meaningful features
convolution layer merge features to find complex features ....and so on

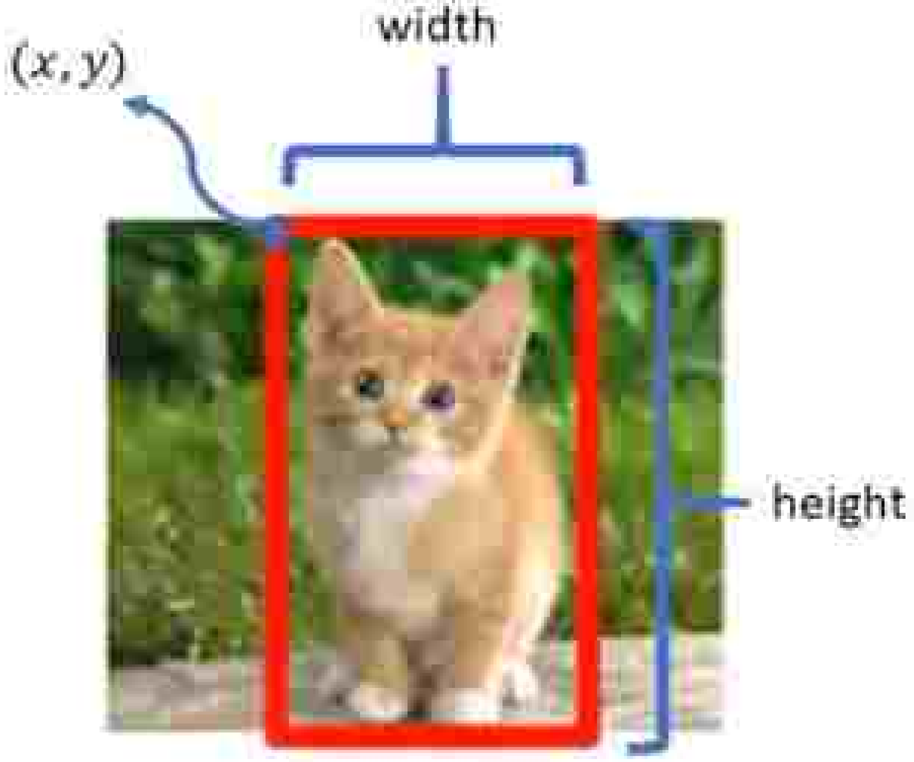
Applications of CNN

Image classification



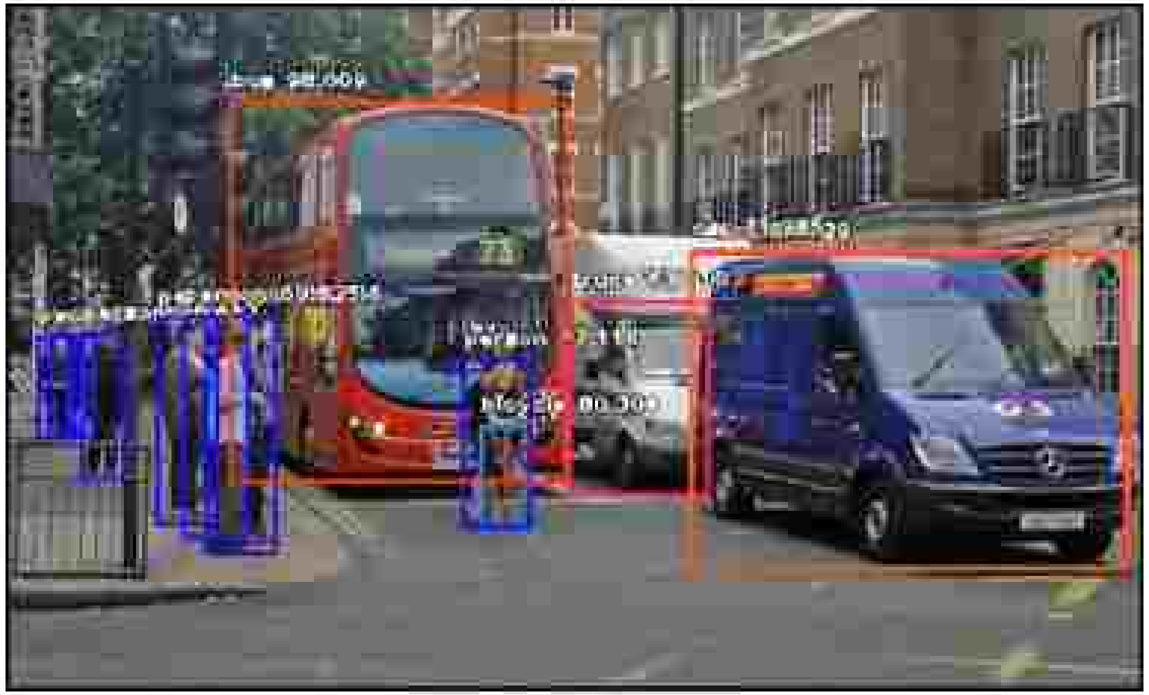
Applications of CNN

Object Localization



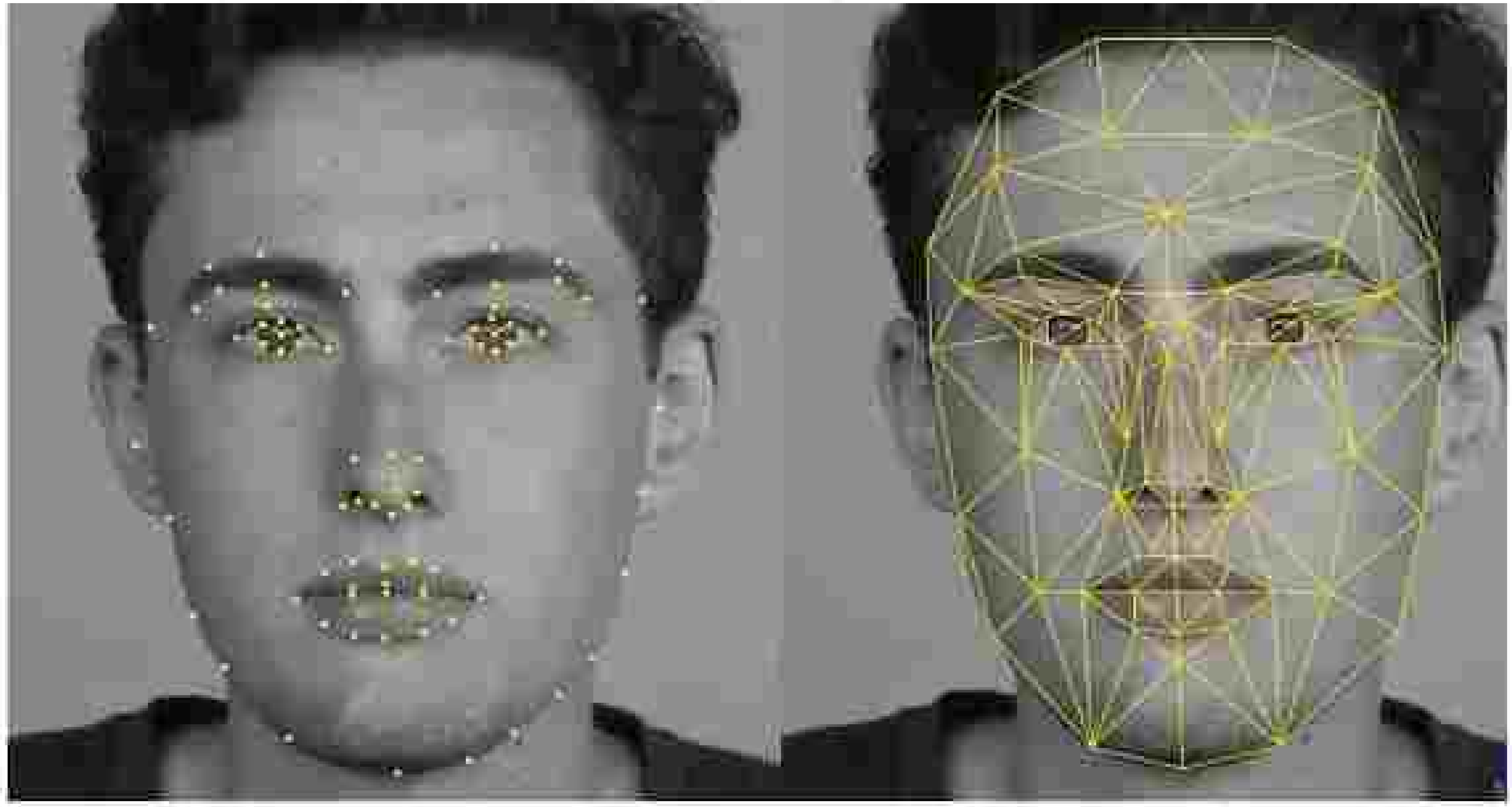
Applications of CNN

Object Detection



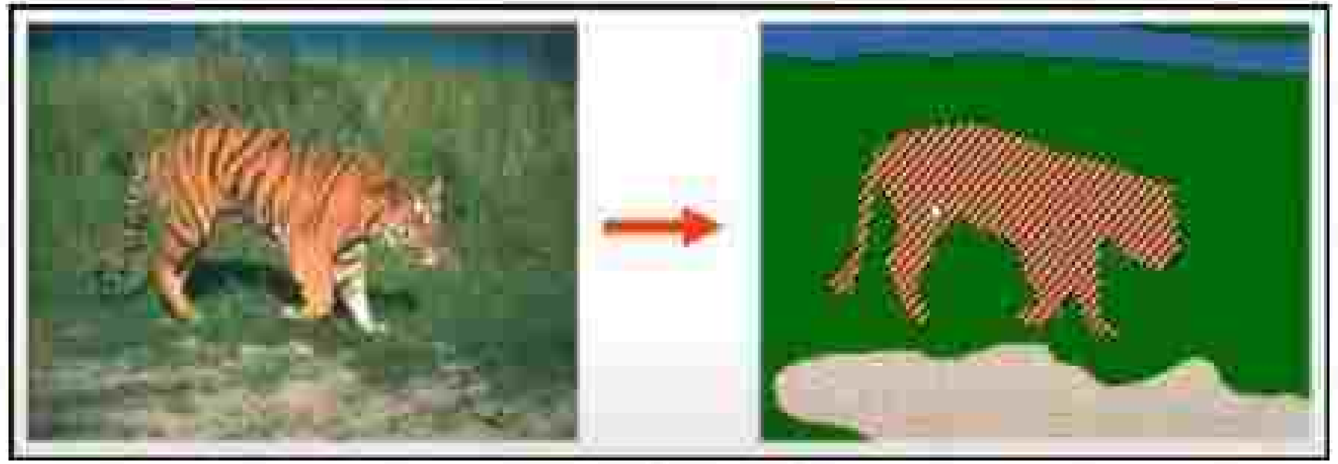
Applications of CNN

Face detection



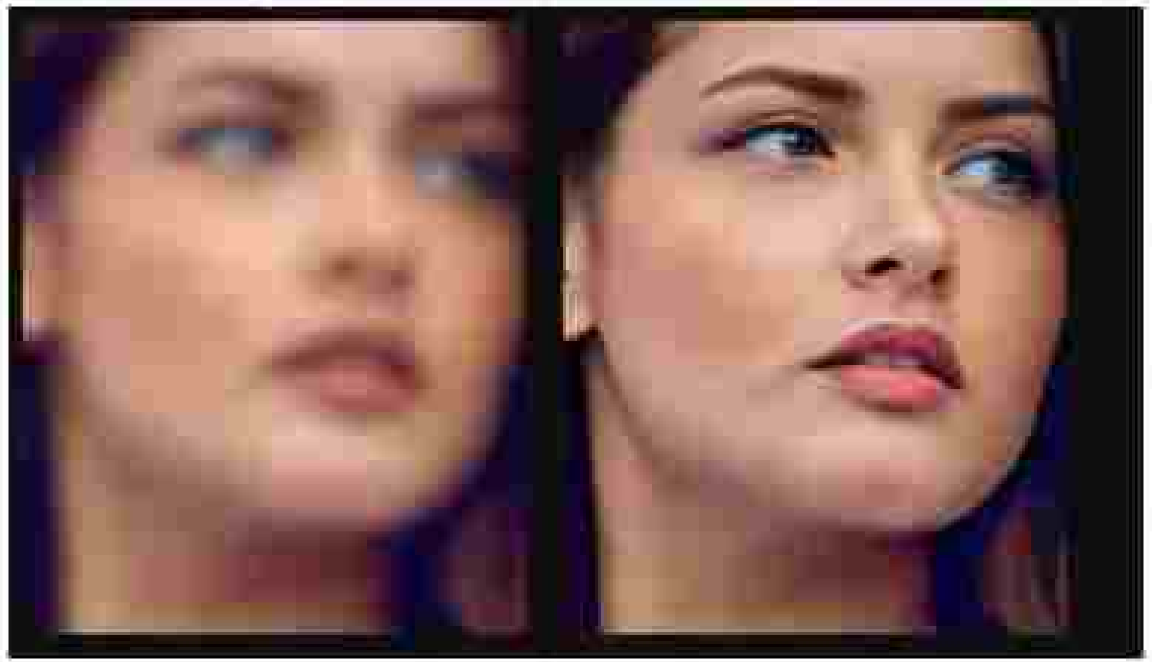
Applications of CNN

Image segmentation



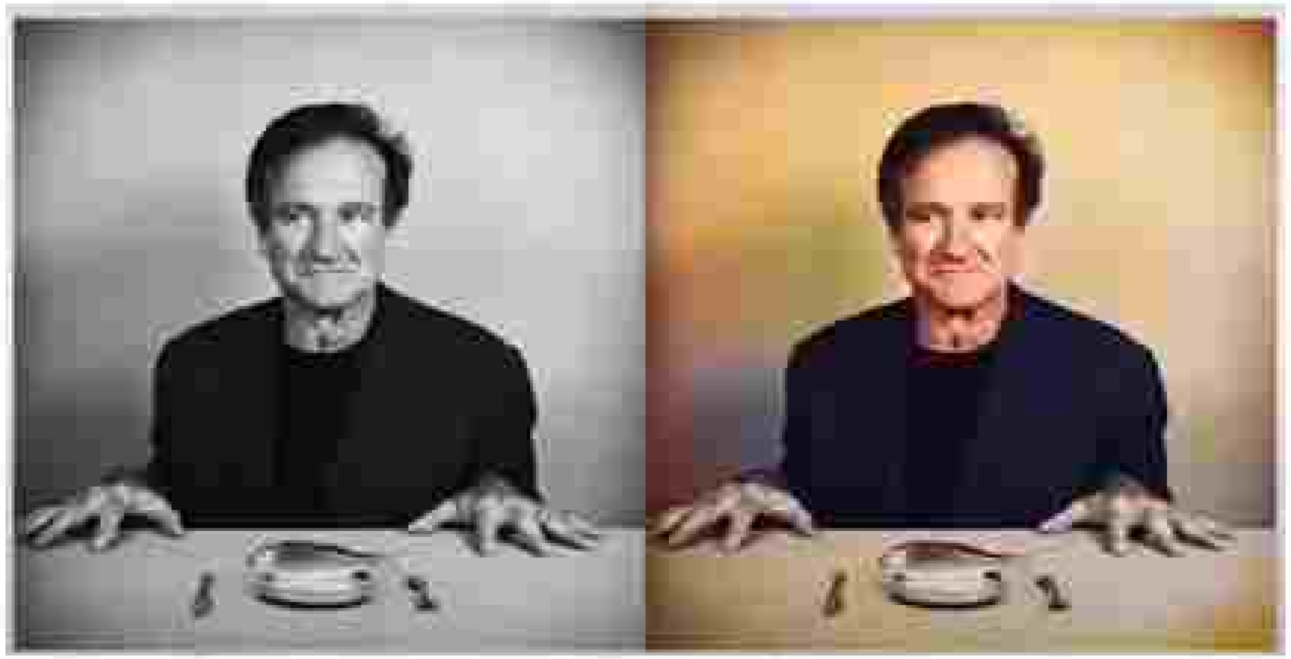
Applications of CNN

Super resolution



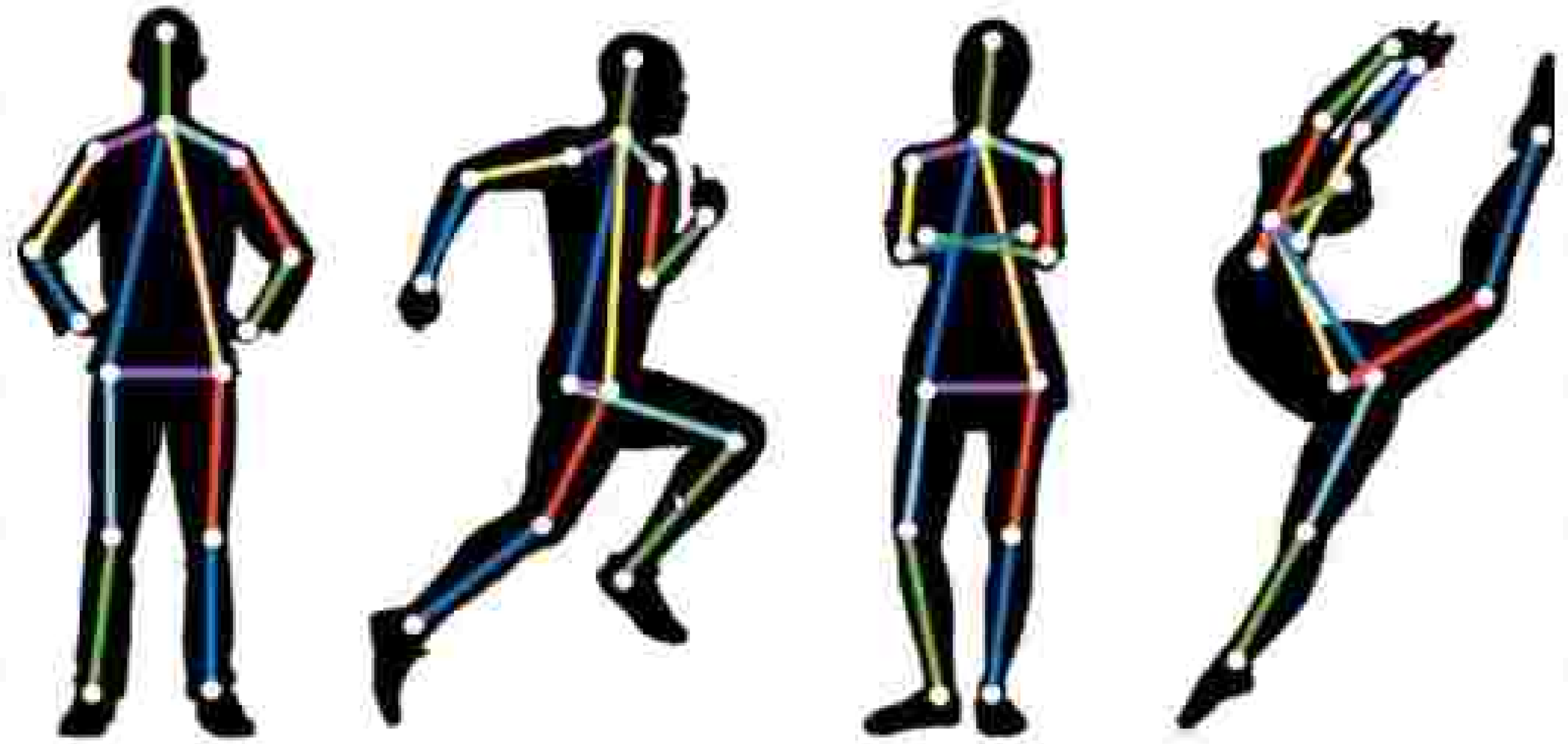
Applications of CNN

Convert grayscale to RGB(color)



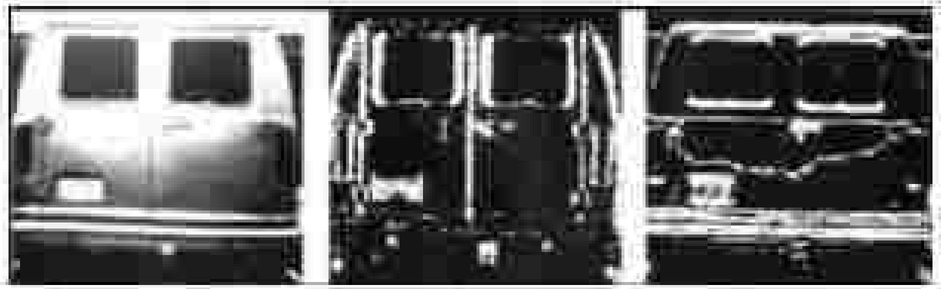
Applications of CNN

Posture detection



Convolution operation

Edge detection(detection of change in intensity)



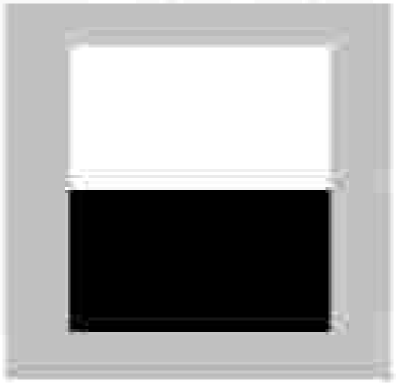
Convolution operation

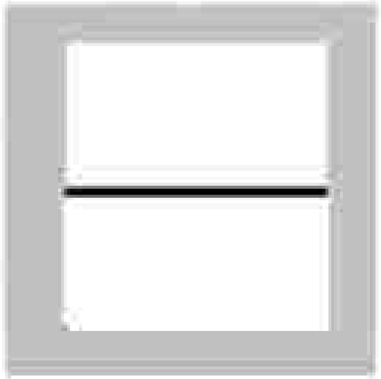
How algorithms detect edges mathematically

$$\begin{array}{|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 255 & 255 & 255 & 255 & 255 & 255 \\ \hline 255 & 255 & 255 & 255 & 255 & 255 \\ \hline 255 & 255 & 255 & 255 & 255 & 255 \\ \hline \end{array}_{n*n} * \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline \end{array}_{f*f} = \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

Filter/kernel

Feature map





Feature map size = (n – f + 1) by (n – f + 1)

Feature map size = (6 – 3 + 1) by (6 – 3 + 1)=4*4

Convolution operation

How algorithms detect edges mathematically

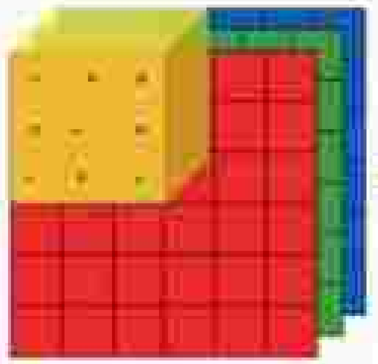
$$\begin{bmatrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 6 & & \\ & & \\ & & \end{bmatrix}$$

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

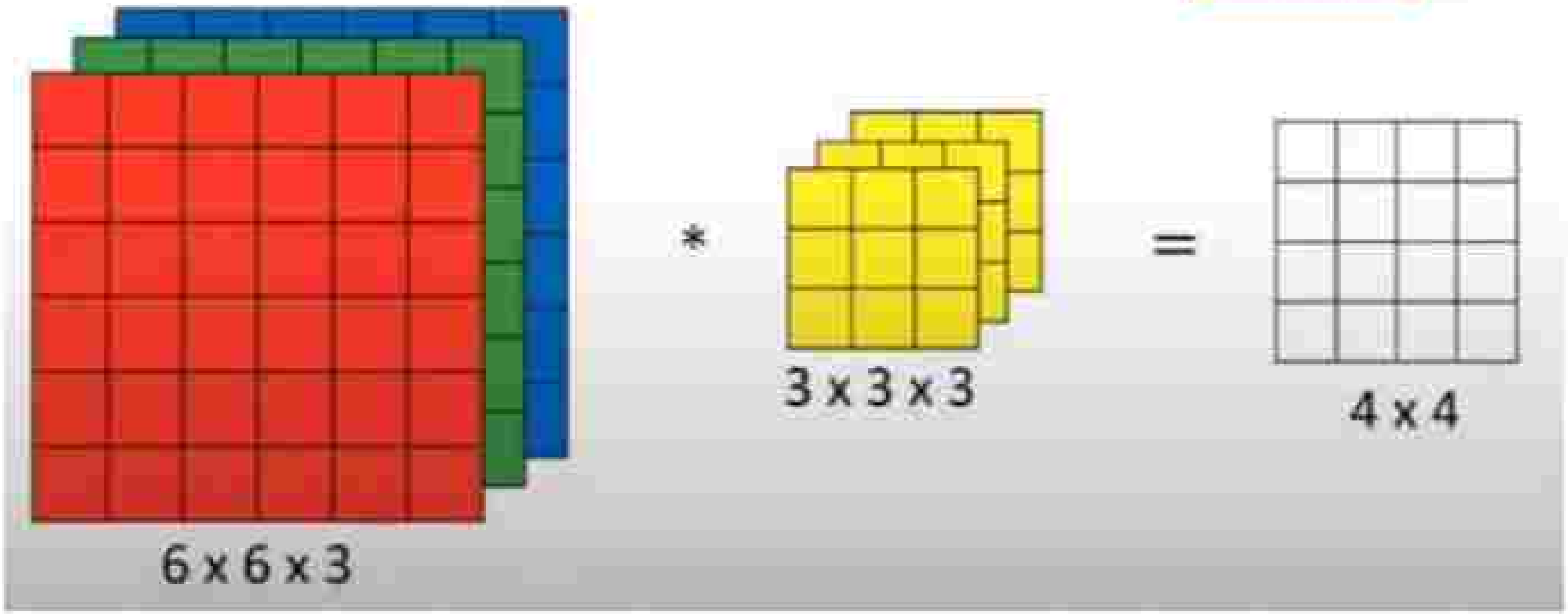
Filter/kernel

Feature map

Convolution operation



How algorithms detect edges mathematically

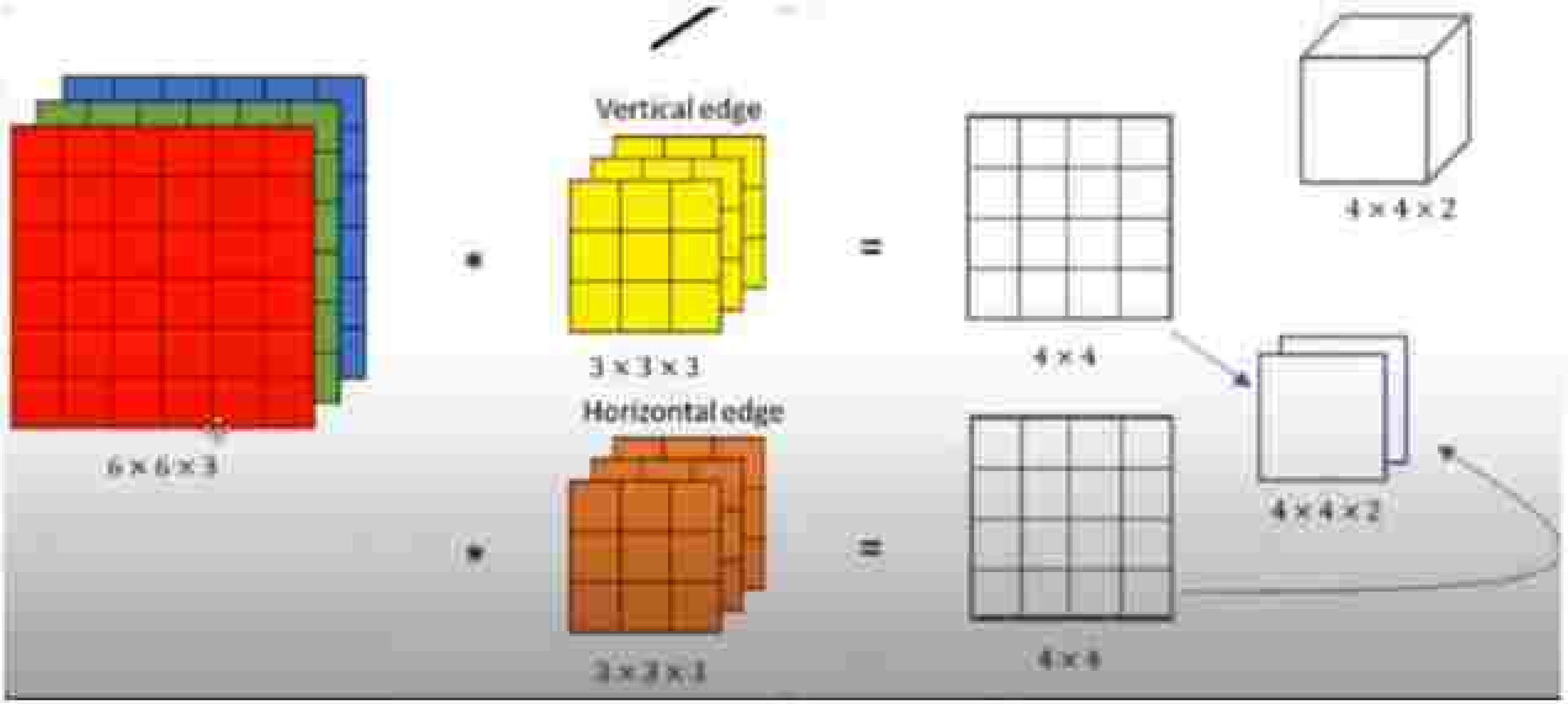


Filter/kernel

Feature map

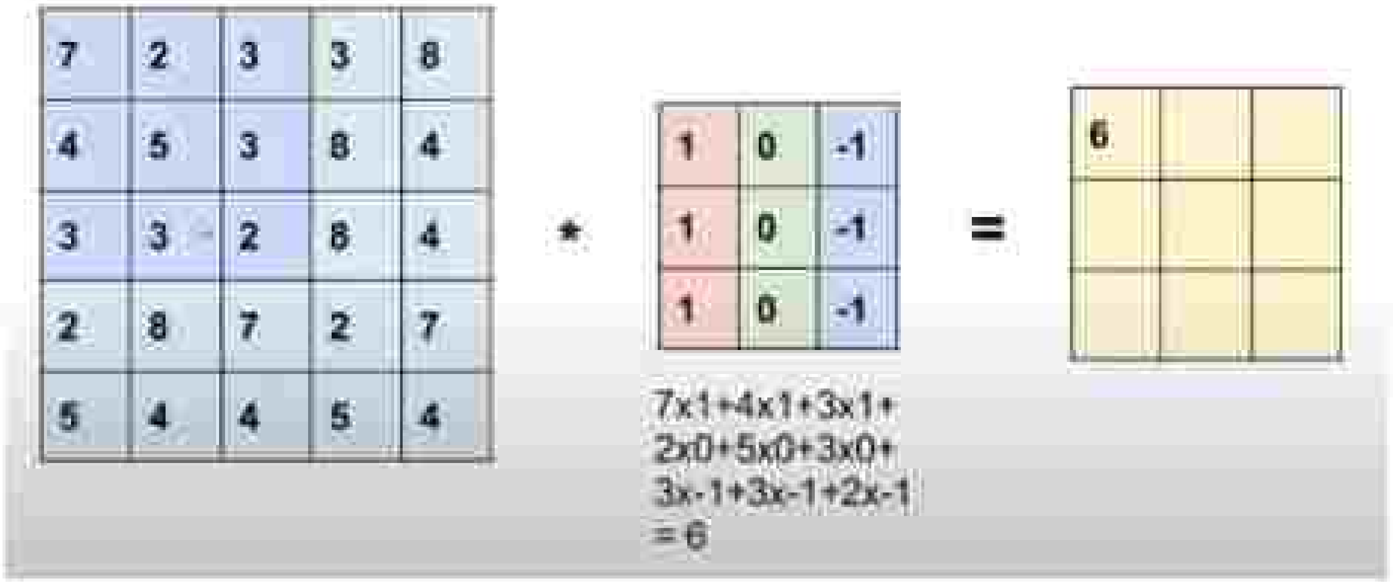
Convolution operation

Multiple Filters

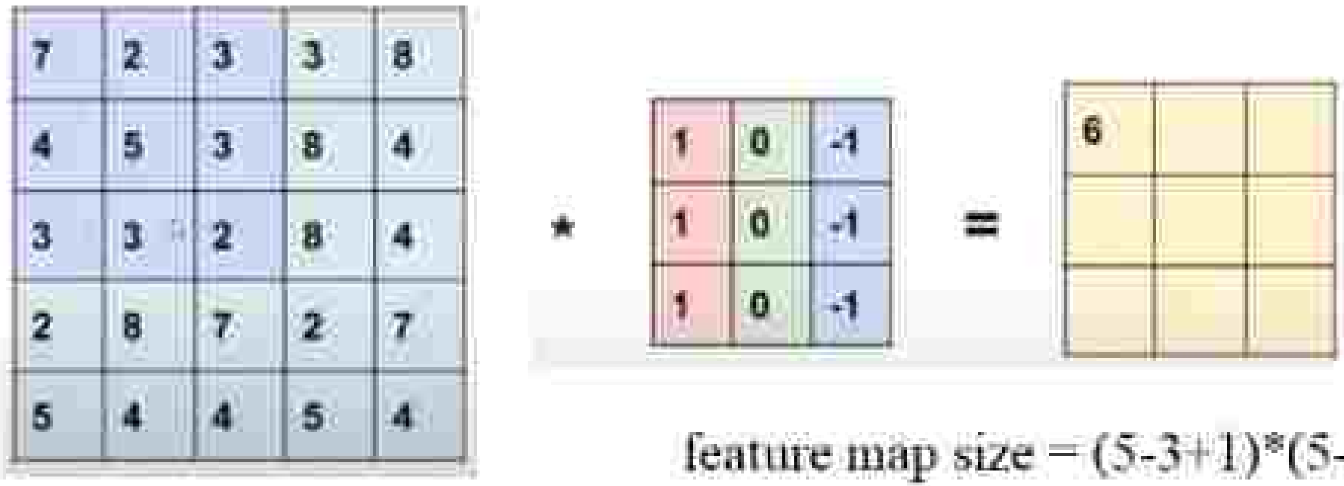


Padding

- Padding is a technique used in convolutional neural networks (CNNs) to manage the size of feature maps as they pass through convolutional layers.
- When a filter is applied to an input image, the output feature map is typically smaller than the input image due to the border pixels being ignored.
- Padding involves adding additional pixels (usually zeros) around the border of the input image before applying the filter.



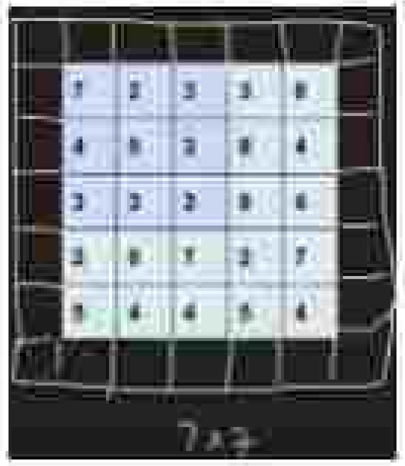
Padding



$$\begin{bmatrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 6 & & \\ & & \\ & & \end{bmatrix}$$

feature map size = (5-3+1)*(5-3+1)= 3*3

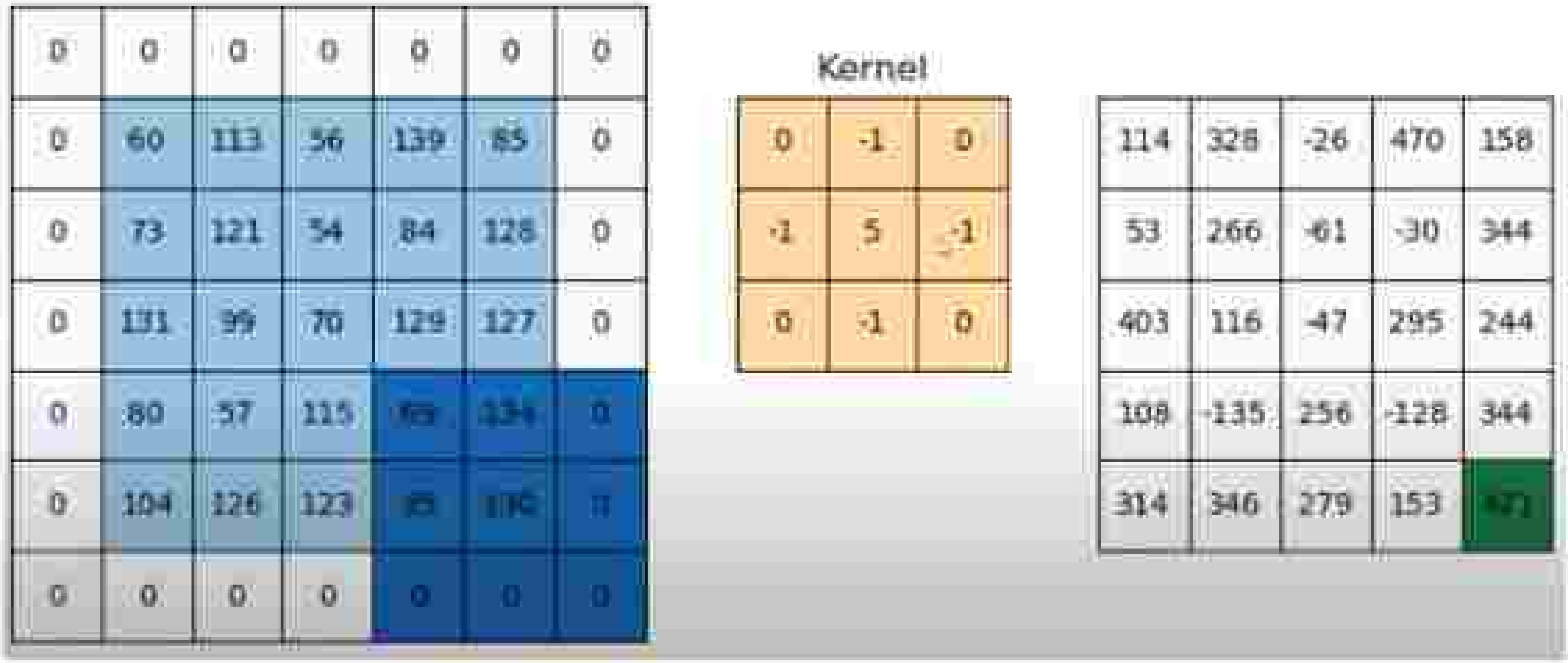
Padding required = (n − 3 + 1) = 5
So n = 7



Deep Leaning

Padding

- The main reason for using padding is to preserve spatial information and ensure that the output feature map has the same spatial dimensions as the input image.
- This is important because it allows the network to capture information from the entire input image, including pixels near the border.



0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

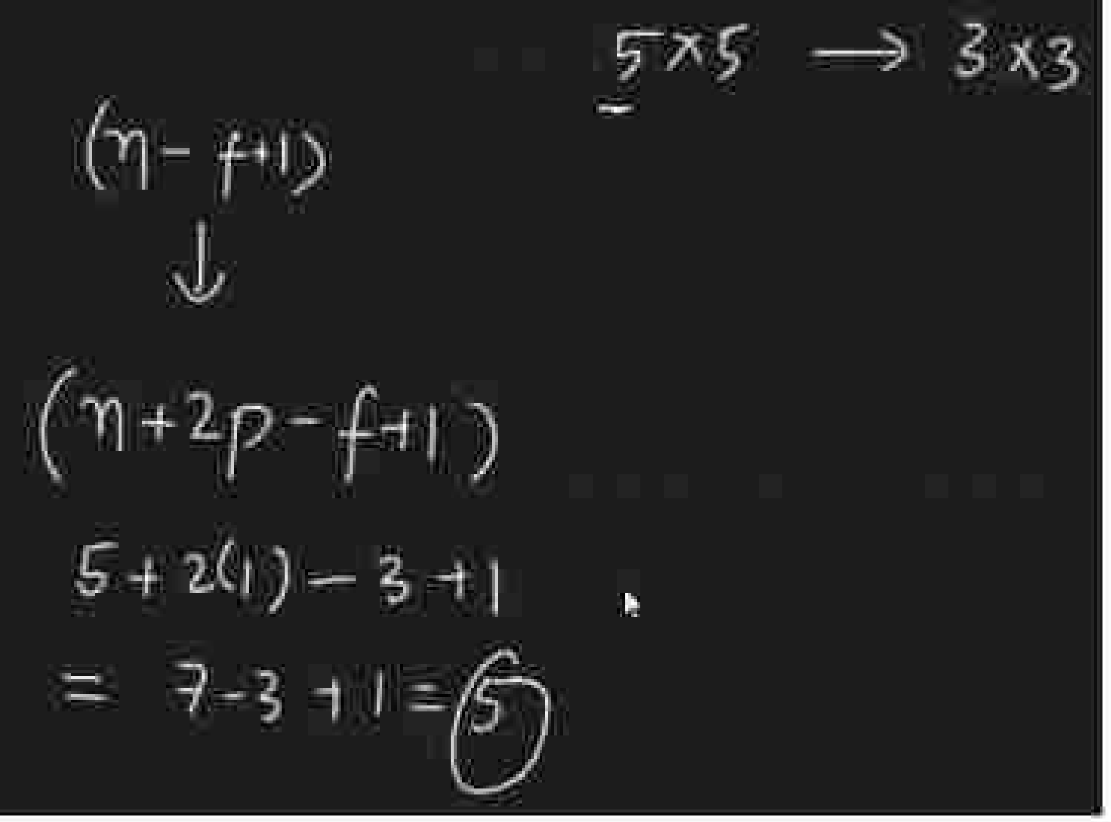
0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	[illegible]

Padding

- Without padding, the size of the feature maps would shrink with each convolutional layer, leading to a loss of spatial information.
- This reduction in size can also cause issues at the boundaries of the image, where important features may be located.
- Padding helps mitigate these issues by ensuring that the convolutional operation is applied uniformly across the entire input image, thereby preserving spatial information and improving the performance of the network.

Feature map size after padding $= (n + 2p - f + 1)$ by $(n + 2p - f + 1)$



$\underline{5} \times 5 \longrightarrow 3 \times 3$

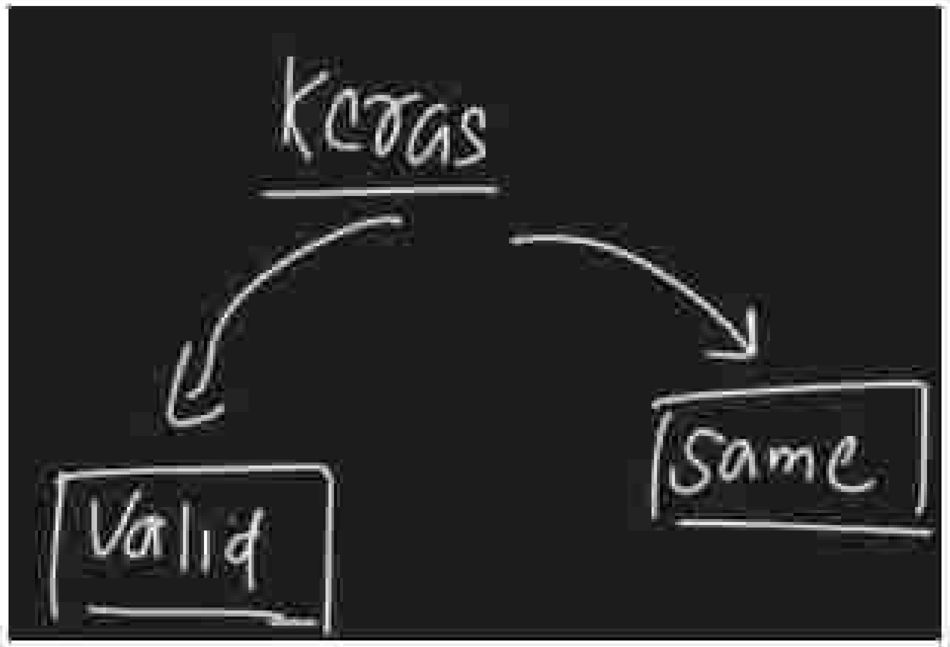
$(n - f + 1)$

$\downarrow$

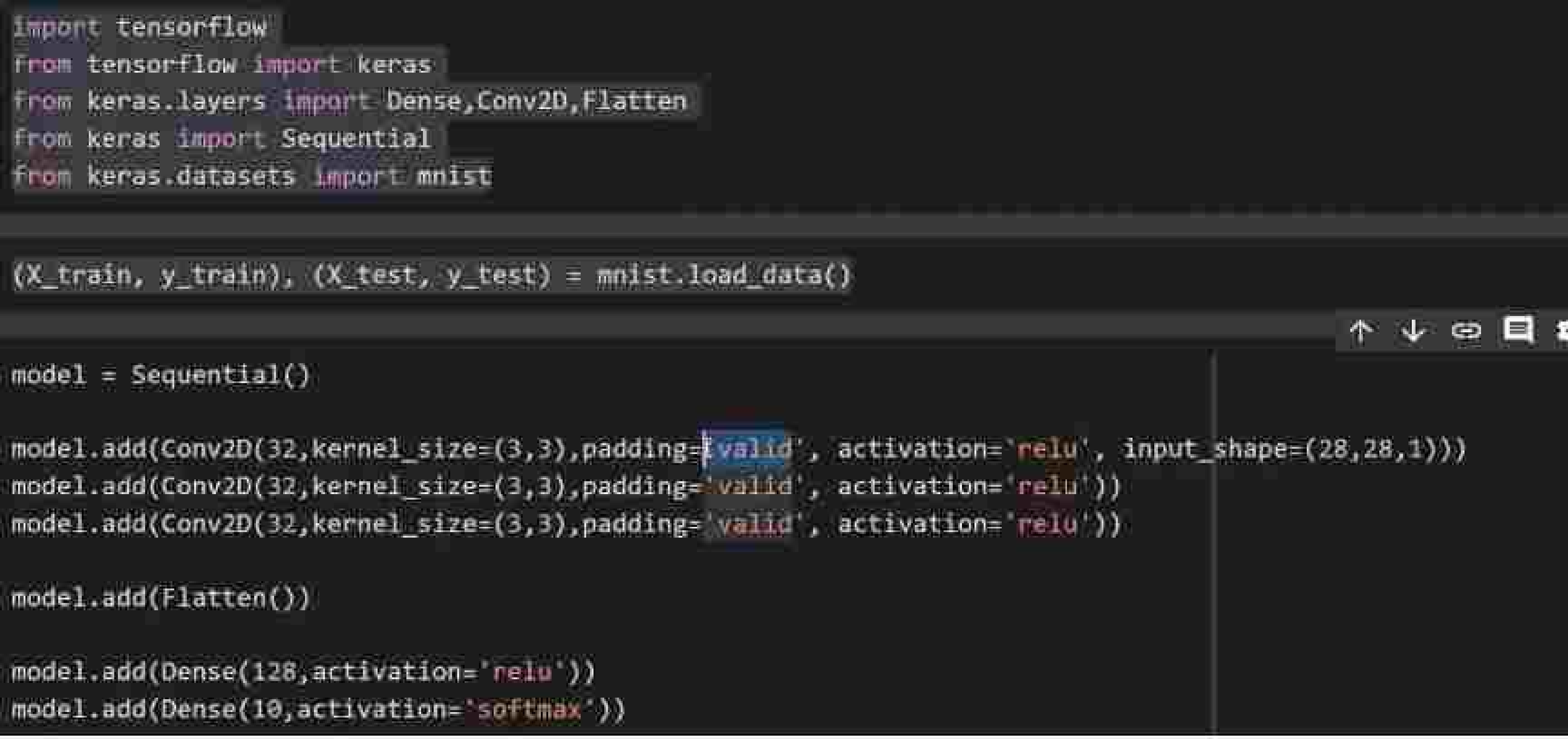
$(n + 2p - f + 1)$

$5 + 2(1) - 3 + 1$

$= 7 - 3 + 1 = \boxed{5}$



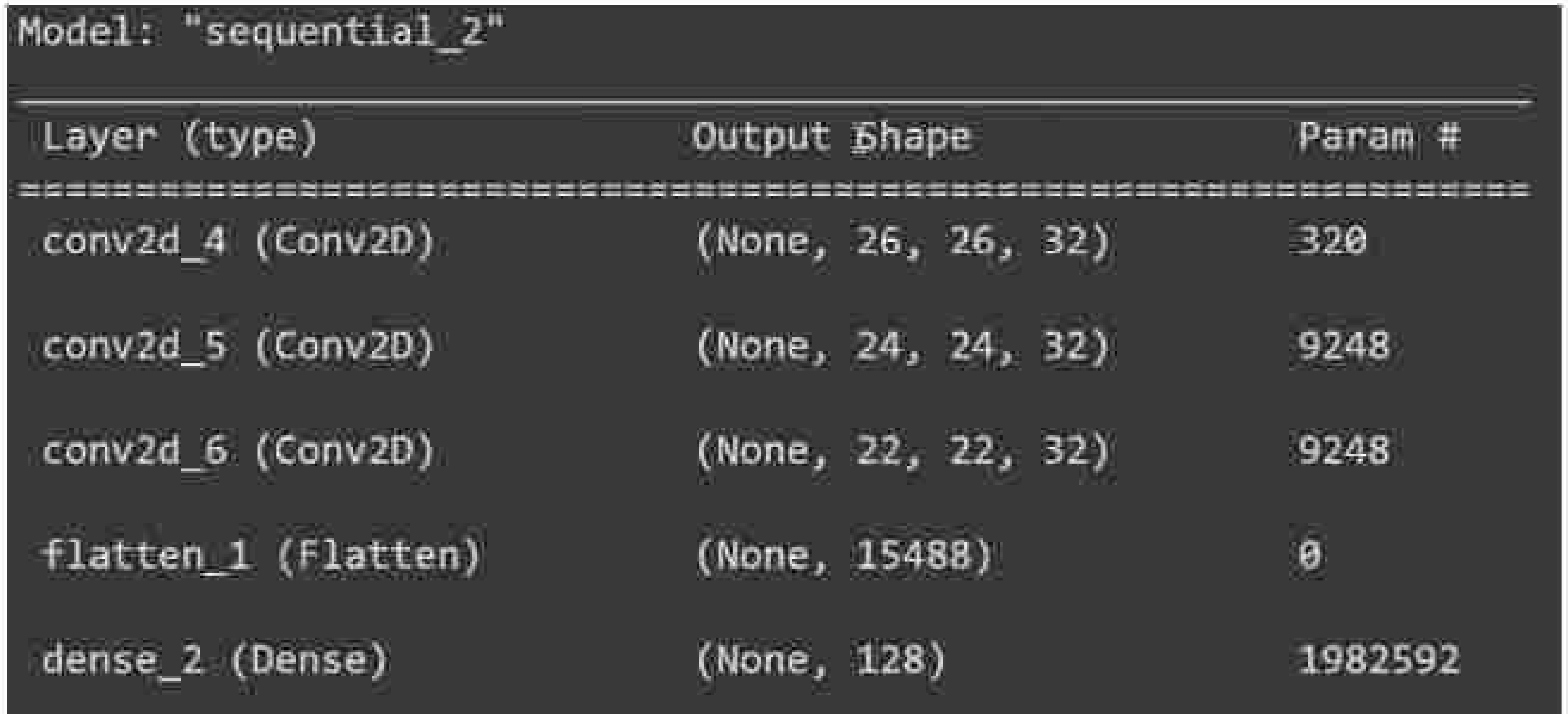
$$\begin{tikzcd}
 & \underline{\text{Keras}} \arrow[dl, bend right] \arrow[dr, bend left] & \\
\boxed{\underline{\text{Valid}}} & & \boxed{\underline{\text{same}}}
\end{tikzcd}$$



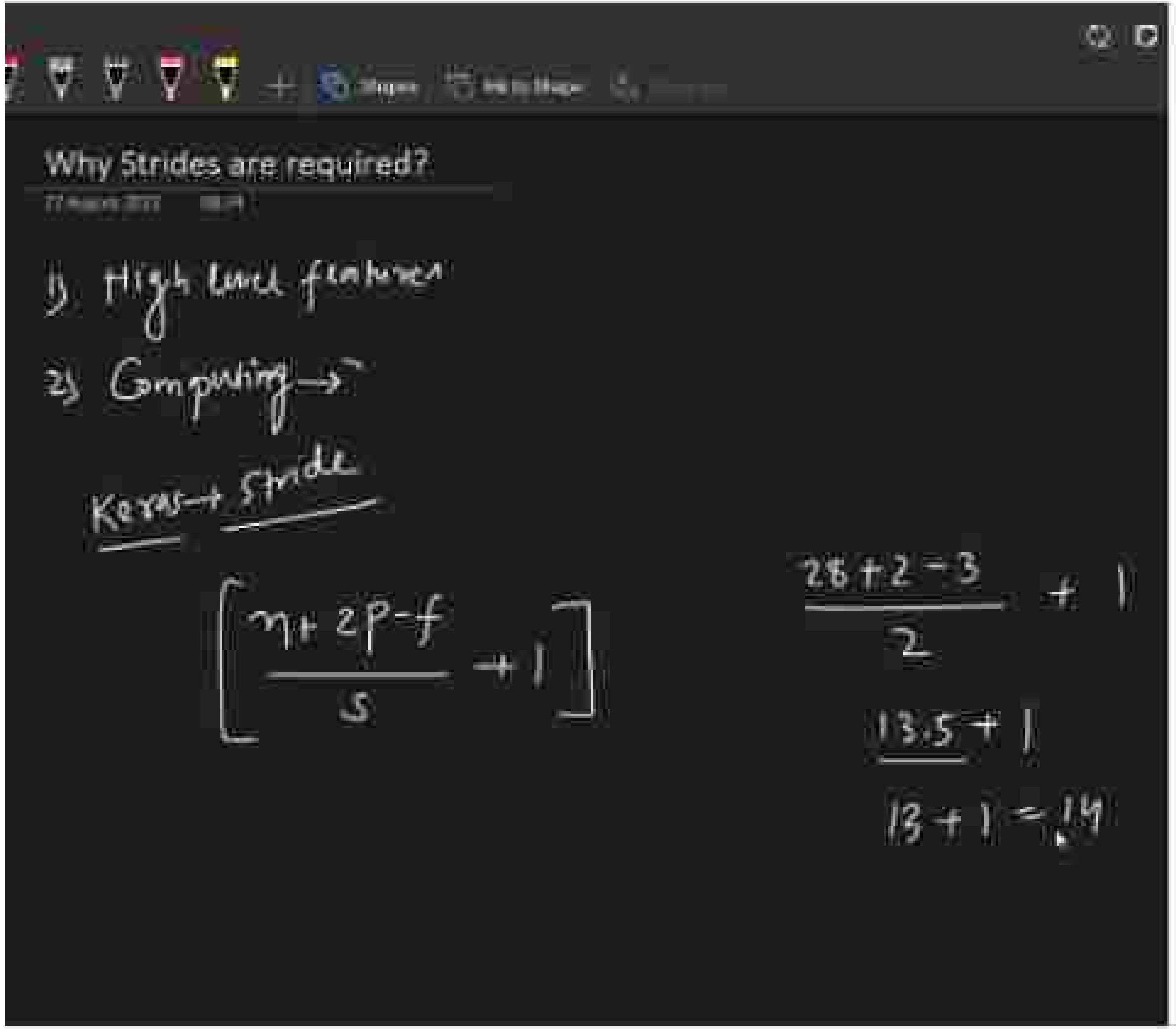
```
import tensorflow
from tensorflow import keras
from keras.layers import Dense,Conv2D,Flatten
from keras import Sequential
from keras.datasets import mnist
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),padding='valid', activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32,kernel_size=(3,3),padding='valid', activation='relu'))
model.add(Conv2D(32,kernel_size=(3,3),padding='valid', activation='relu'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
```



```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 32)        9248

 conv2d_6 (Conv2D)           (None, 22, 22, 32)        9248

 flatten_1 (Flatten)         (None, 15488)             0

 dense_2 (Dense)             (None, 128)               1982592
```



Why Strides are required?

[illegible]

1) High level features

2) Computing →

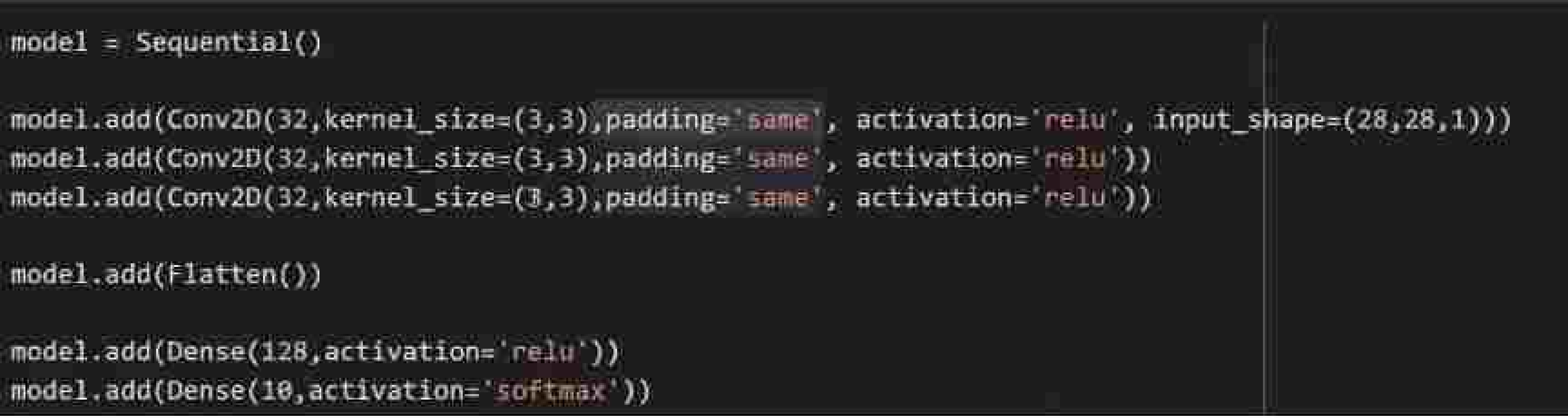
Keras → Stride

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

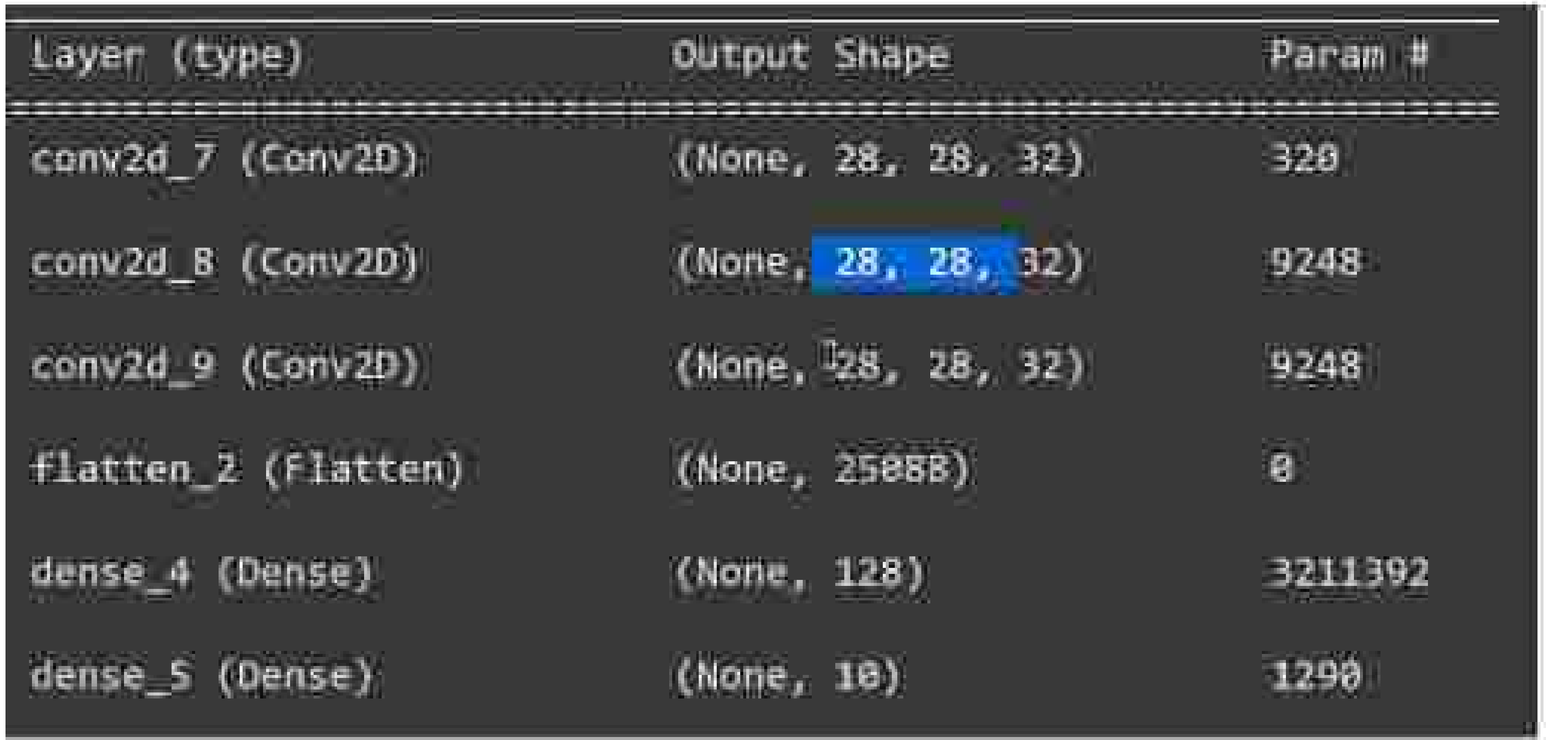
$$\frac{28 + 2 - 3}{2} + 1$$

$$\underline{13.5} + 1$$

$$13 + 1 = 14$$



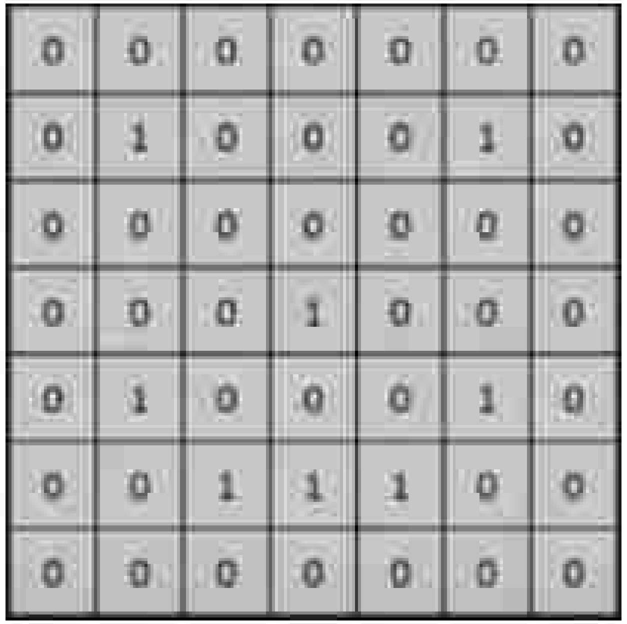
```
model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),padding='same', activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32,kernel_size=(3,3),padding='same', activation='relu'))
model.add(Conv2D(32,kernel_size=(3,3),padding='same', activation='relu'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
```



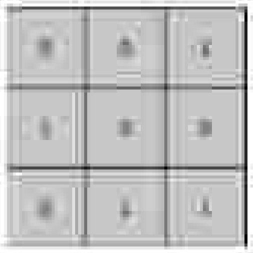
```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)           (None, 28, 28, 32)        320

conv2d_8 (Conv2D)           (None, 28, 28, 32)        9248

conv2d_9 (Conv2D)           (None, 28, 28, 32)        9248

flatten_2 (Flatten)         (None, 25088)             0

dense_4 (Dense)             (None, 128)               3211392

dense_5 (Dense)             (None, 10)                1290
```

Strides

- strides refer to the step size with which the convolutional filter slides (moves) across the input image or feature map during the convolution operation.
- The stride determines the amount by which the filter shifts over the input image at each step. For example, if the stride is set to 1, the filter moves one pixel at a time. If the stride is set to 2, the filter moves two pixels at a time, and so on.



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0



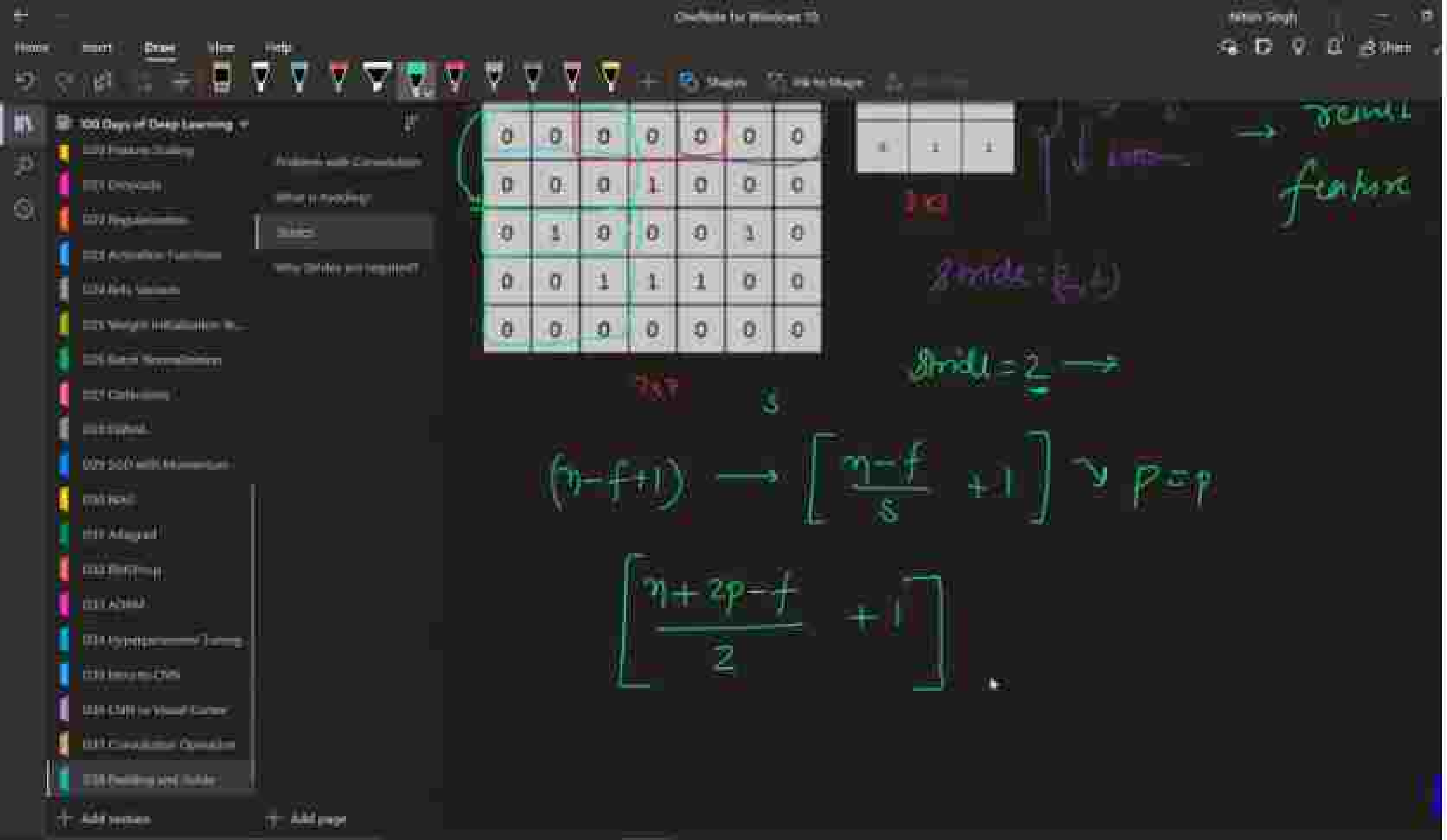
0	0	1
1	0	0
0	1	1

Strides

- The main purpose of using strides is to control the spatial dimensions of the output feature map produced by the convolutional layer. By adjusting the stride size, we can control the amount of overlap between adjacent receptive fields (regions of the input image that are covered by the filter).

- Using larger stride values reduces the spatial dimensions of the output feature map, leading to a decrease in computational complexity and memory usage. Conversely, smaller stride values result in a larger output feature map with more spatial information.

- strides allow us to adjust the spatial resolution of feature maps in CNNs, providing flexibility in balancing computational efficiency with spatial information preservation.


OneNote for Windows 10

Nitish Singh

Home Insert Draw View Help

Shapes Ink to Shape

100 Days of Deep Learning

- 019 Feature Scaling
- 021 Dropouts
- 022 Regularization
- 023 Activation Functions
- 024 ReLU Variants
- 025 Weight Initialization Te...
- 026 Batch Normalization
- 027 Optimizers
- 028 EWMA
- 029 SGD with Momentum
- 030 NAG
- 031 Adagrad
- 032 RMSProp
- 033 ADAM
- 034 Hyperparameter Tuning
- 035 Intro to CNN
- 036 CNN vs Visual Cortex
- 037 Convolution Operation
- 038 Padding and Stride

- Problem with Convolution
- What is Padding?
- Strides
- Why Strides are required?

Add section

Add page

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

0	1	1

7×7

3

3×3

stride $= \frac{1}{2}(1)$

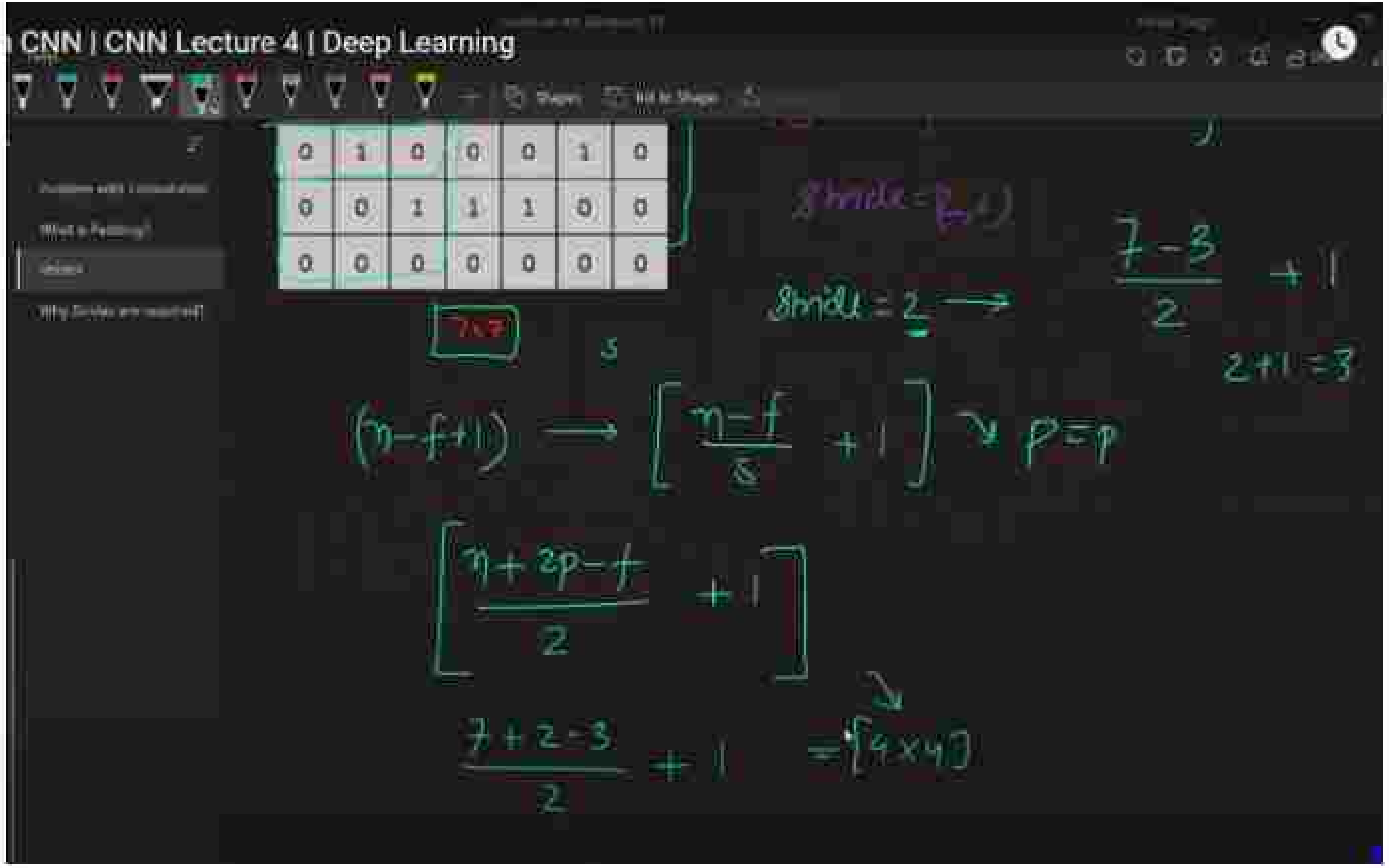
$\text{Stride} = 2 \longrightarrow$

left

feature

$$(n-f+1) \longrightarrow \left[\frac{n-f}{s} + 1\right] \rightarrow p=p$$

$$\left[\frac{n+2p-f}{2} + 1\right]$$



n CNN | CNN Lecture 4 | Deep Learning

0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

$\boxed{7\times7}$

Stride = [illegible]

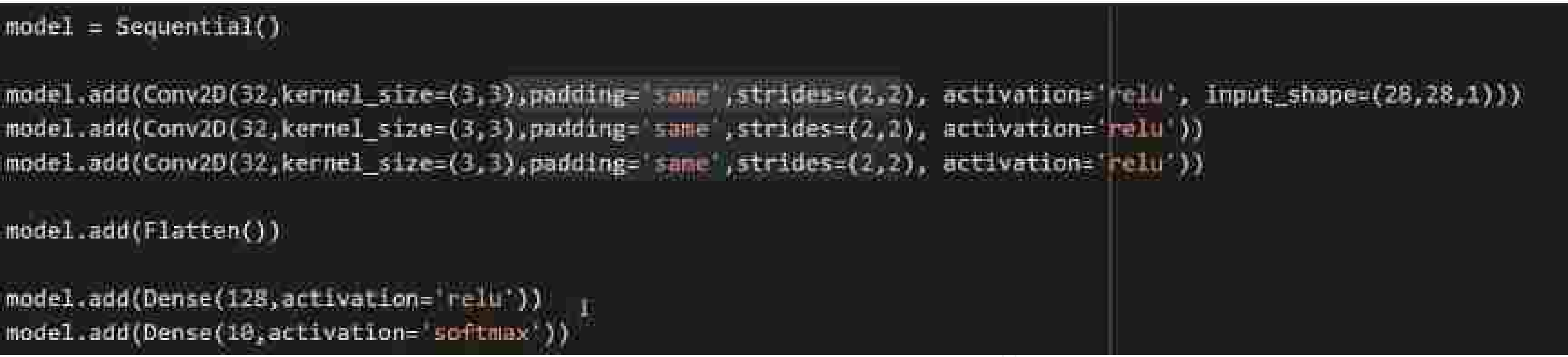
Stride = 2 $\longrightarrow$ $\frac{7-3}{2} + 1$

$2+1=3$

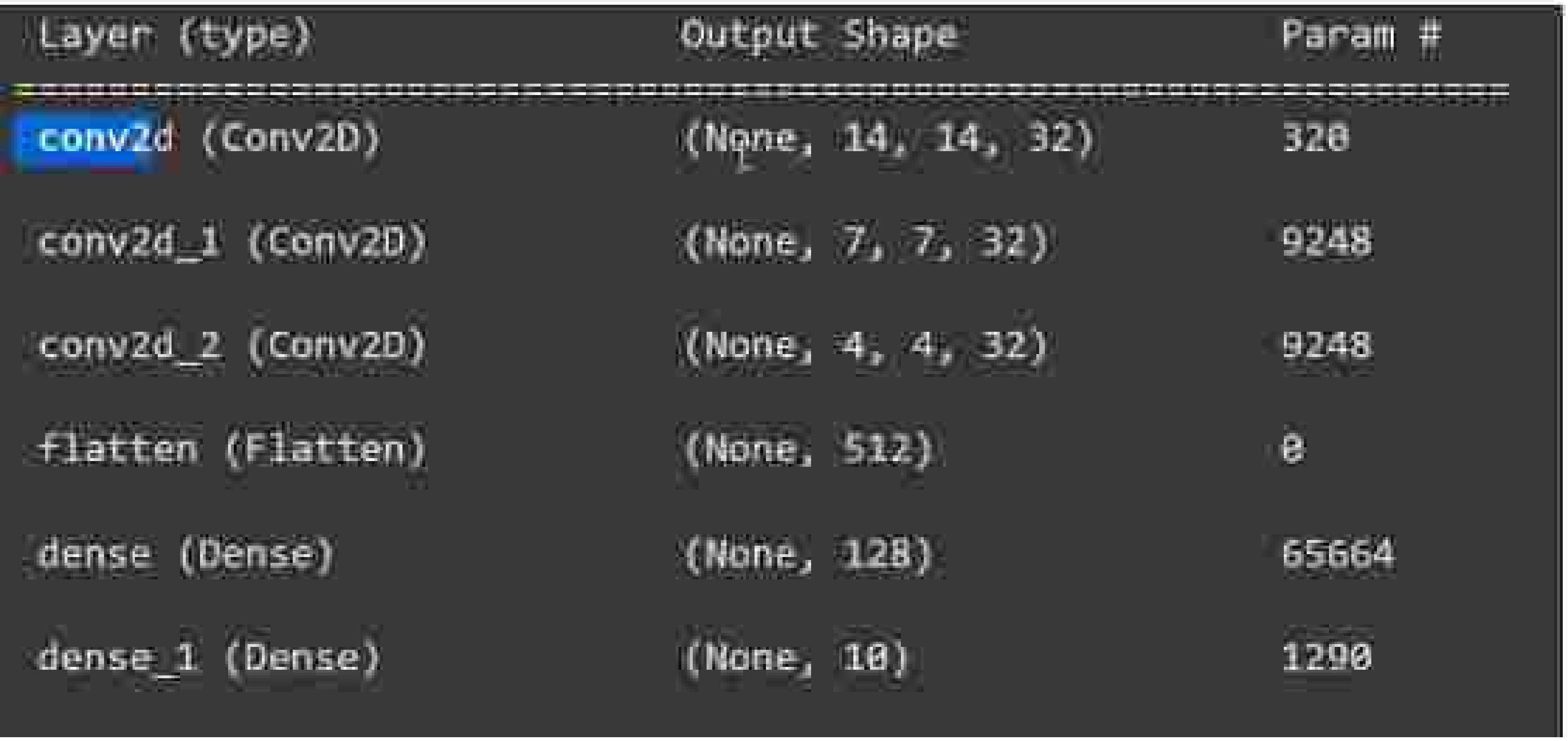
$$(n-f+1) \longrightarrow \left[\frac{n-f}{s} + 1\right] \searrow p=p$$

$$\left[\frac{n+2p-f}{2} + 1\right]$$

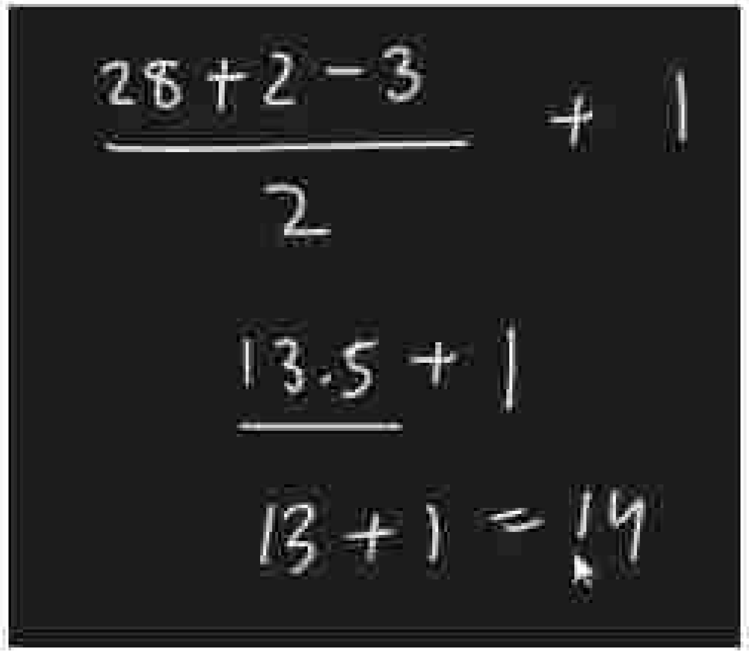
$$\frac{7+2-3}{2} + 1 = [4\times4]$$



```python
model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),padding='same',strides=(2,2), activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32,kernel_size=(3,3),padding='same',strides=(2,2), activation='relu'))
model.add(Conv2D(32,kernel_size=(3,3),padding='same',strides=(2,2), activation='relu'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
```



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 32)        320

 conv2d_1 (Conv2D)           (None, 7, 7, 32)          9248

 conv2d_2 (Conv2D)           (None, 4, 4, 32)          9248

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 128)               65664

 dense_1 (Dense)             (None, 10)                1290
```



$$\frac{28+2-3}{2} + 1$$

$$\underline{13.5} + 1$$

$$13 + 1 = 14$$

Explain input shape, output shape, filter, padding, stride and tensor ….. 10M MU-Last Exam

Input Shape:

- The input shape refers to the dimensions of the input data that is fed into a neural network layer.
- In CNNs, the input shape represents the dimensions of the input image or feature map, typically expressed as (height, width, channels).
- For example, an input shape of (228, 228, 3) indicates an input image with a height and width of 228 pixels and three color channels (e.g., RGB).

Output Shape:

- The output shape refers to the dimensions of the output data produced by a neural network layer after applying transformations.
- In CNNs, the output shape depends on factors such as the filter size, padding, stride, and the architecture of the layer.
- For example, the output shape of a convolutional layer depends on the number of filters used and the size of the input feature map.

Deep Leaning

Explain input shape, output shape, filter, padding, stride and tensor ….. 10M MU-Last Exam

Filter:

- A filter, also known as a kernel, is a small matrix of weights that is applied to the input data during convolutional operations.
- Filters are used to extract features from input data by performing element-wise multiplications and summations.
- In CNNs, filters slide across the input data to compute convolutions and generate feature maps.

Padding:

- Padding is a technique used to preserve the spatial dimensions of feature maps during convolutional operations.
- It involves adding extra rows and columns of zeros around the input data before applying convolutions.
- Padding helps to prevent information loss at the edges of feature maps and ensures that the output size matches the input size.

Explain input shape, output shape, filter, padding, stride and tensor ….. 10M MU-Last Exam

Stride:

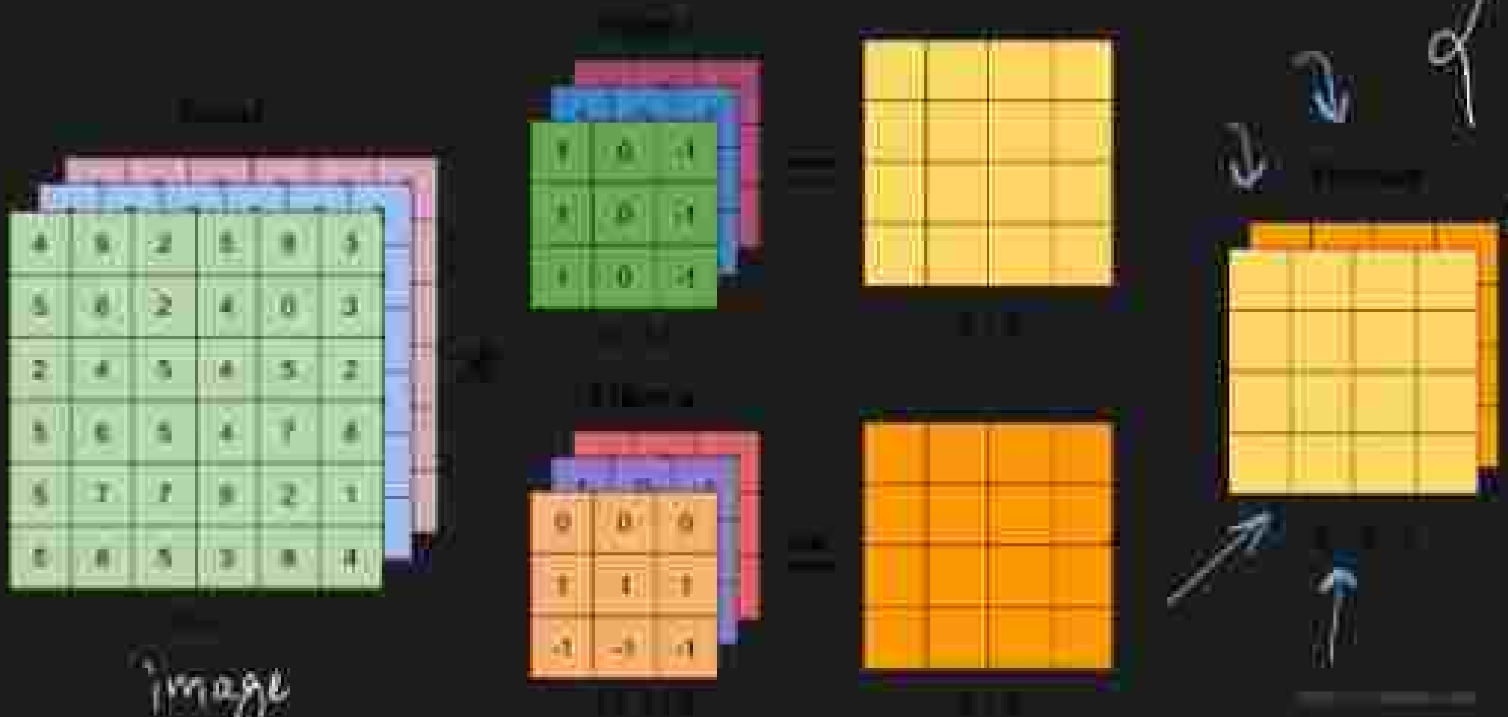
- Stride refers to the number of steps the filter moves across the input data during convolutional operations.
- A stride of 1 indicates that the filter moves one pixel at a time, while a larger stride skips pixels.
- Increasing the stride reduces the spatial dimensions of the output feature maps.

Tensor:

- A tensor is a multi-dimensional array used to represent data in neural networks.
- In CNNs, tensors are used to store and manipulate input data, weights, biases, and intermediate feature maps.
- Tensors have a rank, shape, and data type, and they are fundamental to the operations performed in deep learning models.

The Problem with Convolution

01 September 2022 09:35



stride=2

1) $\boxed{\text{Memory issue}}$

2) Translation variance

reduce size of feature map

pooling

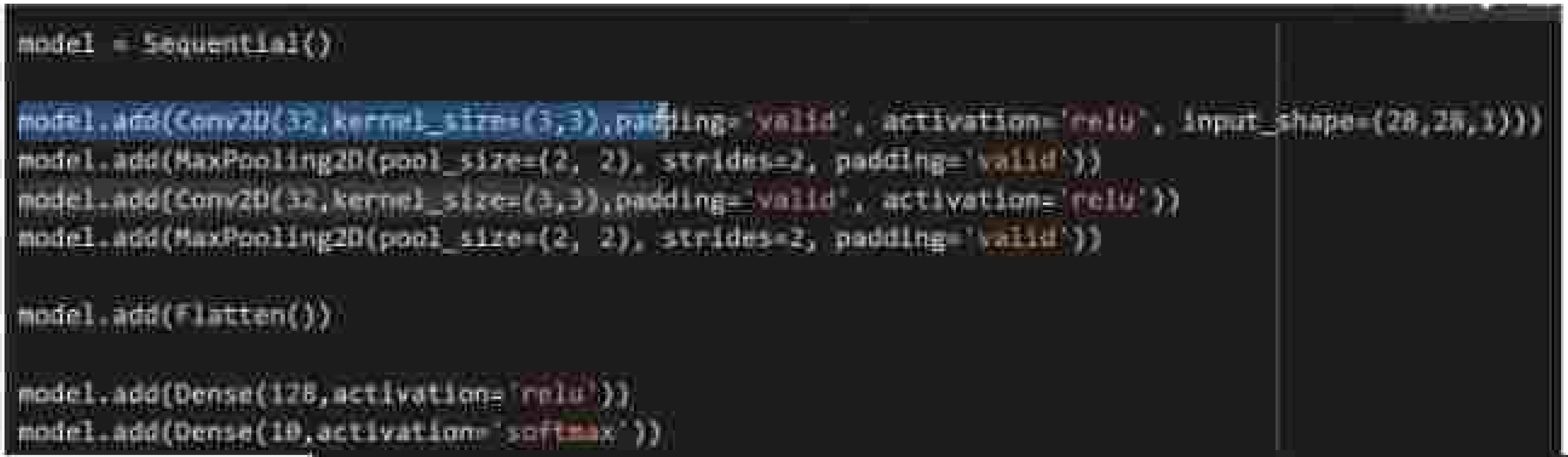
training → 100

image

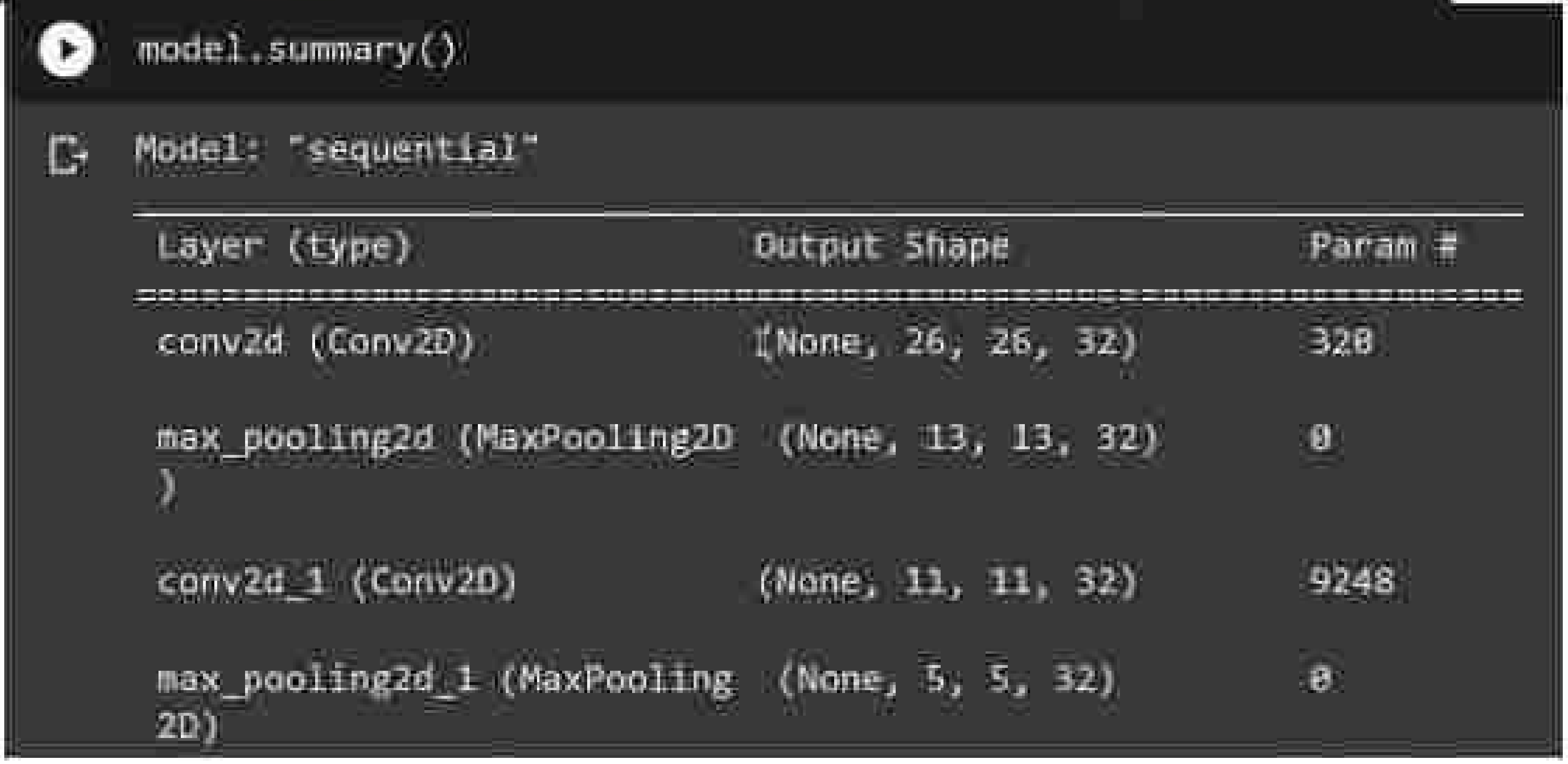
228×228×3

filters (100) → feature map → $\underbrace{(226 \times 226) \times 100}$ × 32 → 19 MB

1.5 GB



```
model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),padding='valid', activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'))
model.add(Conv2D(32,kernel_size=(3,3),padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
```



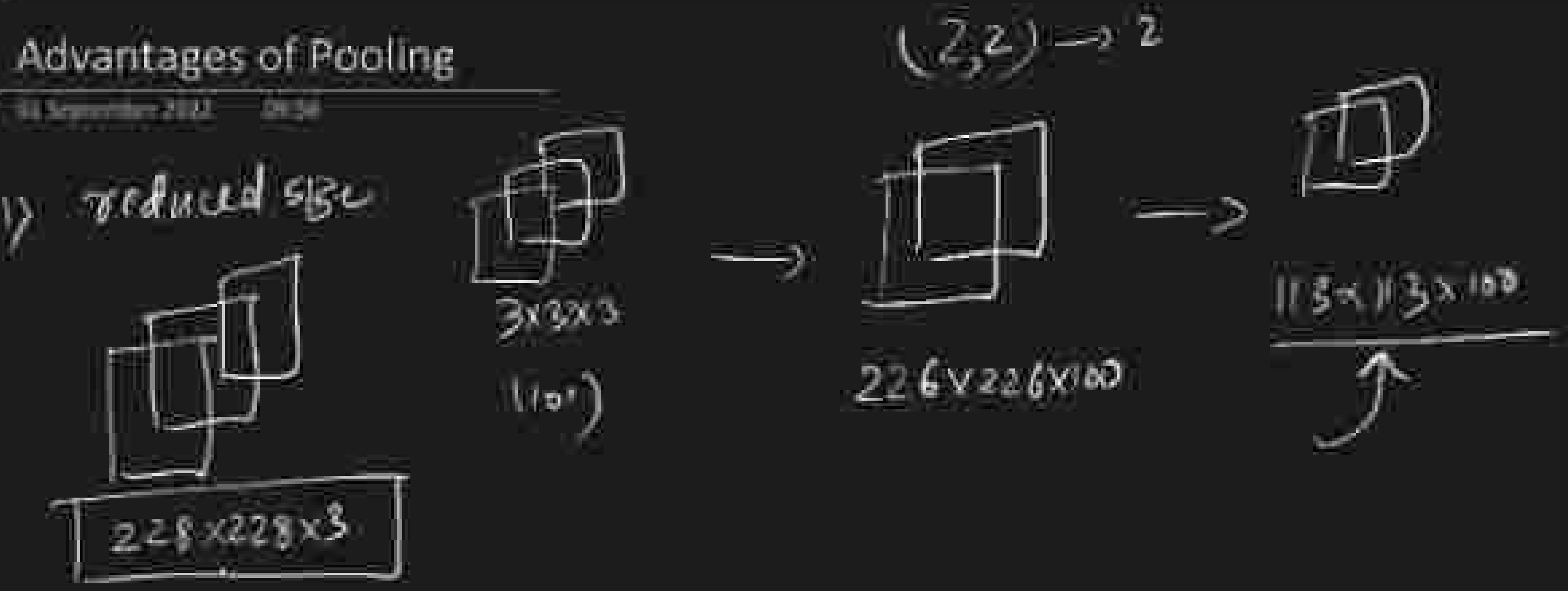
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)
```

Advantages of Pooling

01 September 2022 09:58

1) reduced size

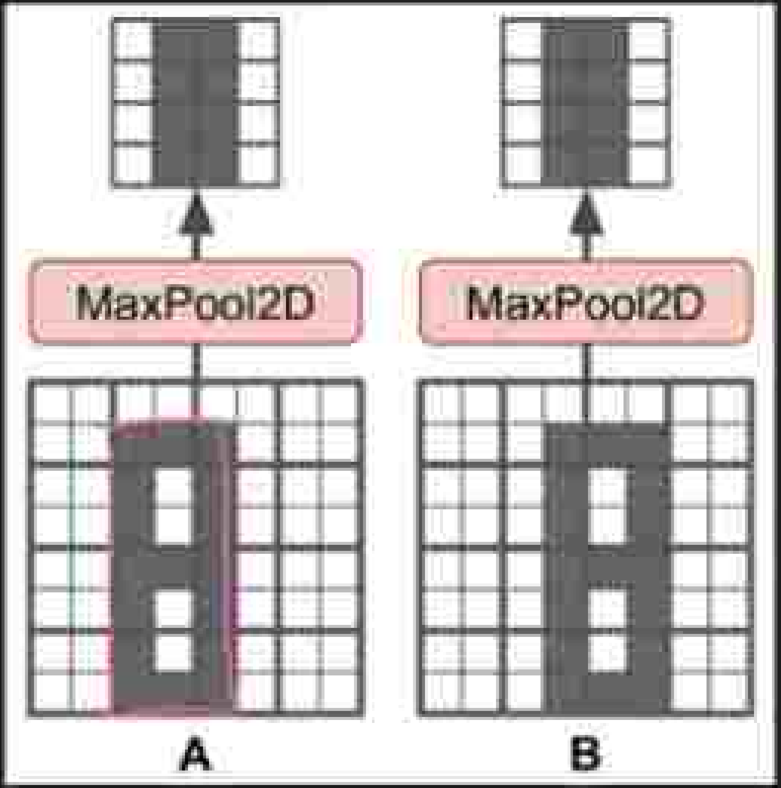


4:29

039 Pooling HOME INSERT DRAW VIEW

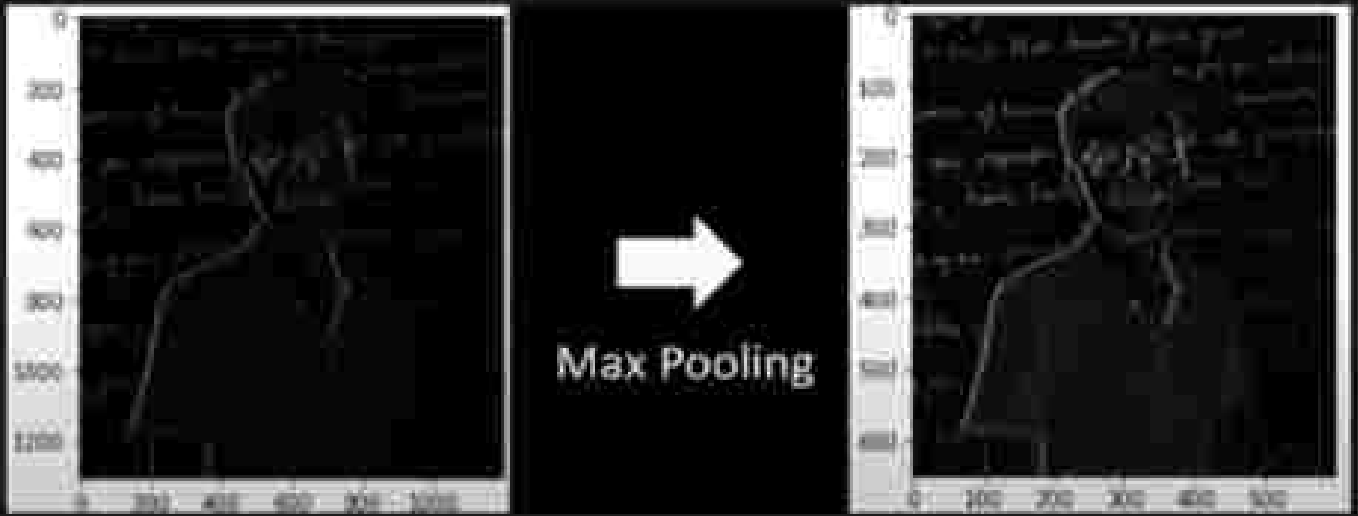
Stop

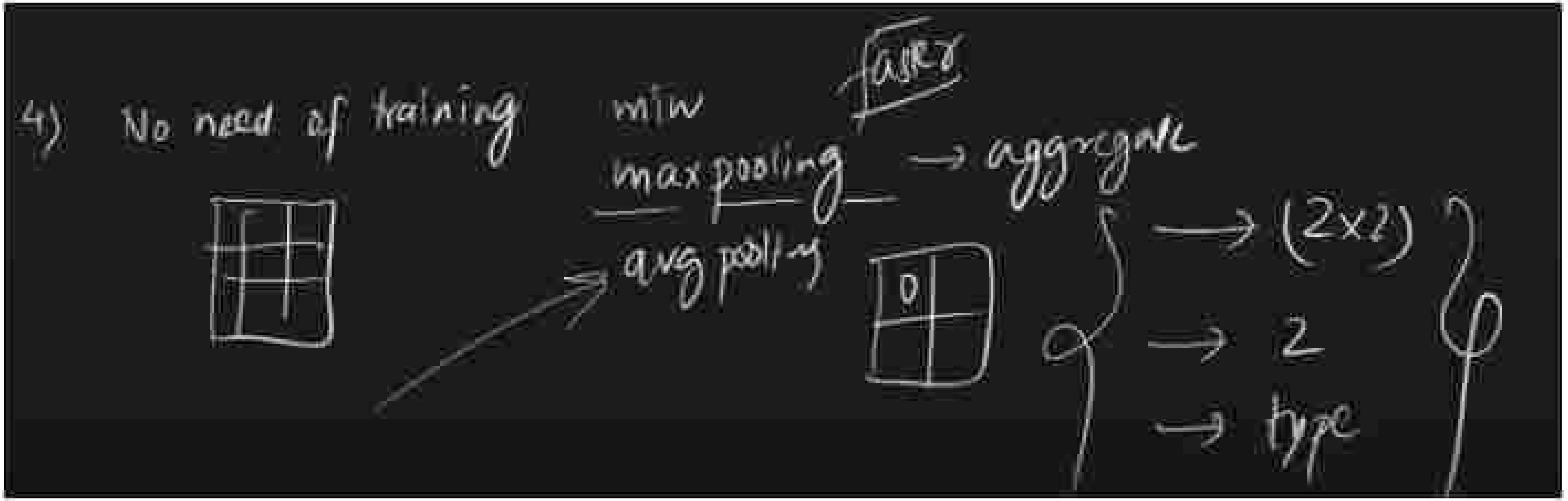
2> Translation invariance



←

3) Enhanced Features
(only in case of Max pooling)





4) No need of training

mtw

$\boxed{\text{faster}}$

max pooling → aggregate

avg pooling

→ (2x2)

→ 2

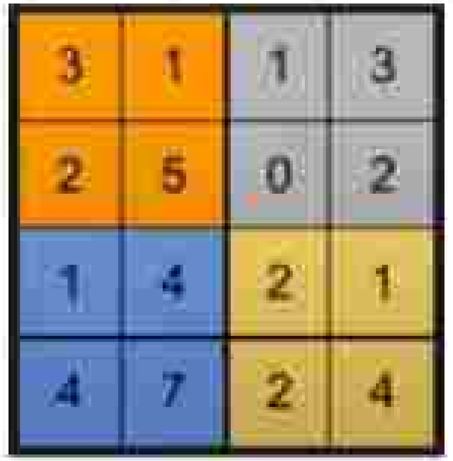
→ type

Pooling

- Pooling overcomes Memory issue and Translation variable issue

- Pooling is a technique used in convolutional neural networks (CNNs) to downsample the feature maps produced by convolutional layers.
- It involves partitioning the input feature map into non-overlapping regions and summarizing each region with a single value, typically by taking the maximum value (max pooling) or the average value (average pooling).
- The pooled output effectively reduces the spatial dimensions of the feature map while preserving important spatial information.

Pooling



3	1	1	3
2	5	0	2
1	4	2	1
4	7	2	4

Size = (2*2)

Stride =2

Pooling type=max

Pooling

Pooling serves several purposes in CNNs:

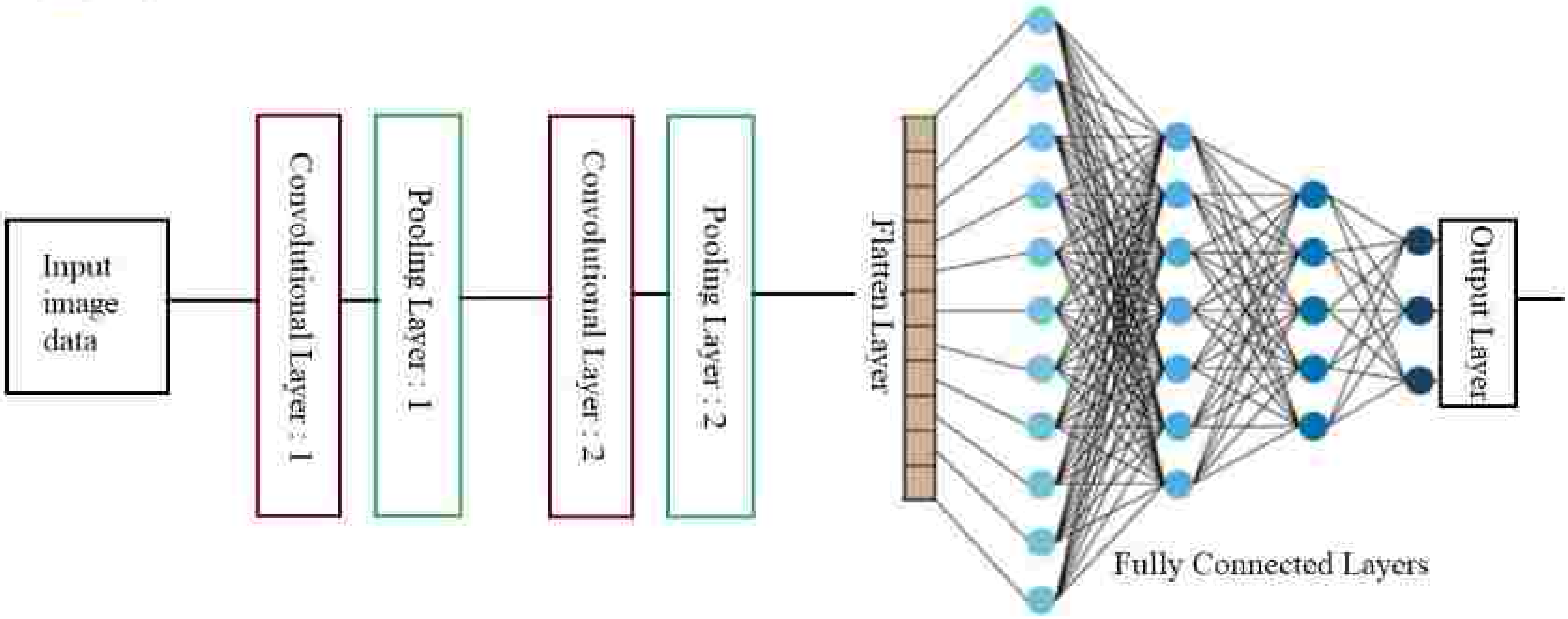
- Dimensionality Reduction
- Translation Invariance
- Feature Generalization

Common types of pooling operations include:

1. Max Pooling: Computes the maximum value within each pooling region, preserving the most prominent features in the input.
2. Average Pooling: Computes the average value within each pooling region, providing a smoother downsampled representation of the input.

Pooling layers are typically inserted between convolutional layers in CNN architectures, allowing the network to learn hierarchical representations of the input data while gradually reducing the spatial dimensions of the feature maps.

CNN Architecture



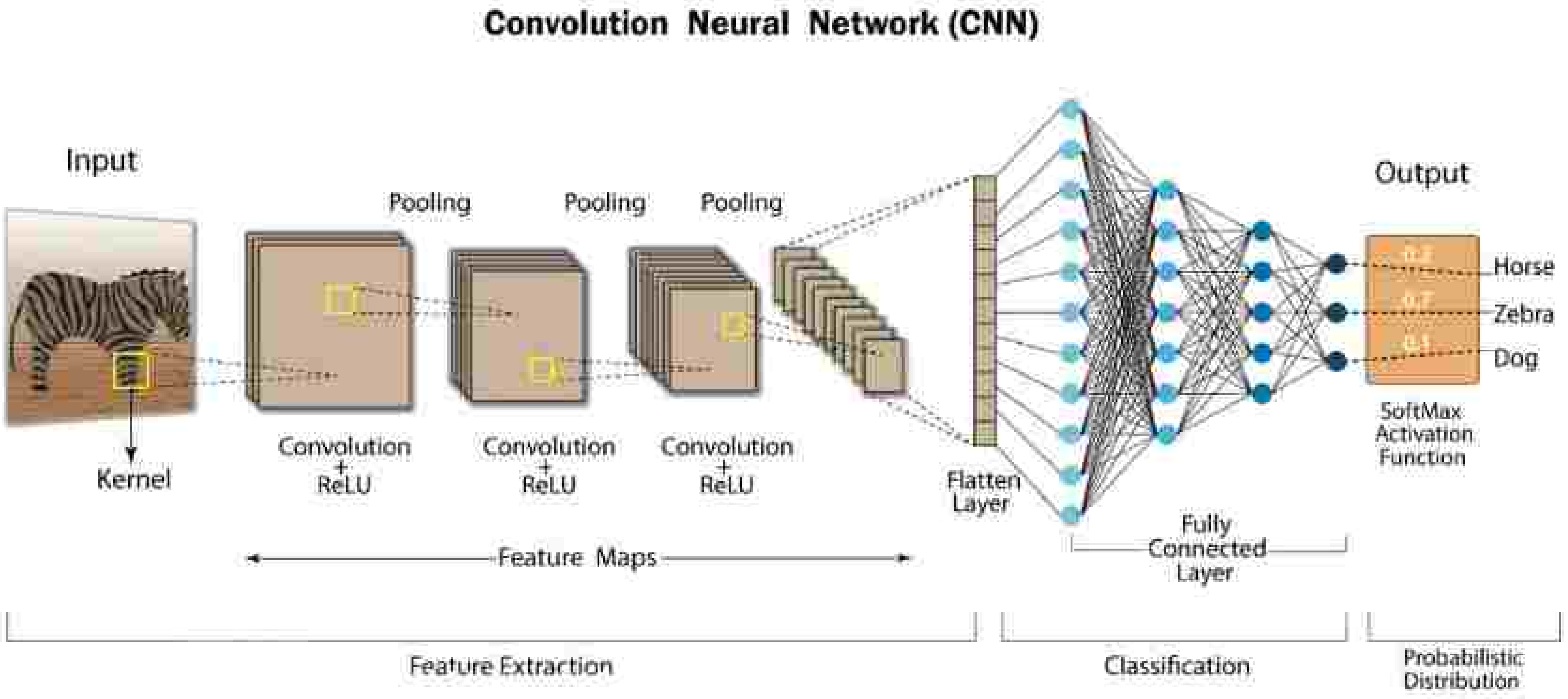
Block diagram illustrating the architecture of a Convolutional Neural Network (CNN)

CNN Architecture

CNN Architecture:

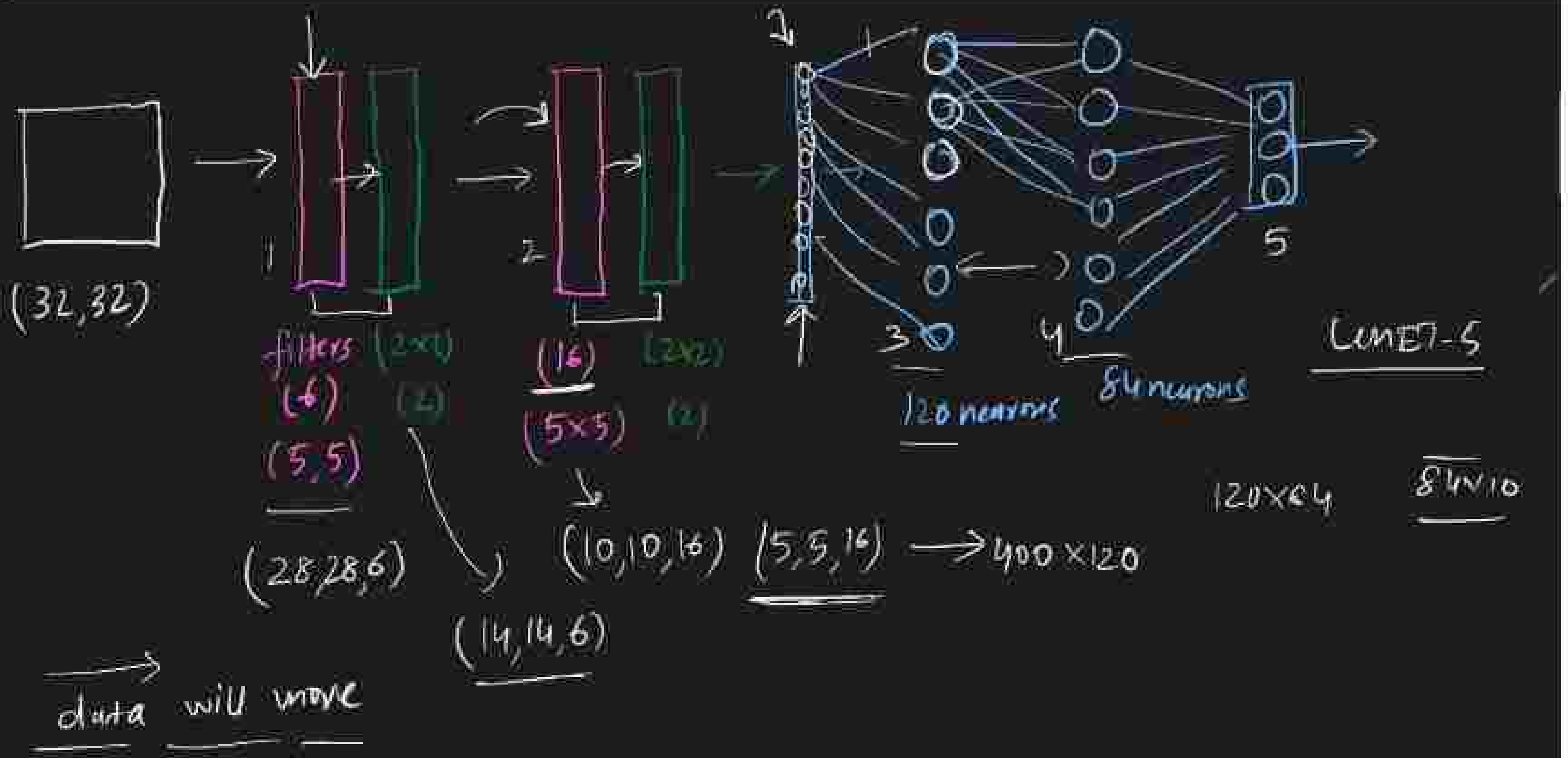
- The "Input" represents the raw image data.
- Convolutional Layers apply filters to the input data, extracting features through convolutions.
- Pooling Layers reduce the spatial dimensions of the convolved features, helping to reduce computation and control overfitting.
- The "Flatten" layer converts the pooled feature map into a one-dimensional vector.
- Fully Connected Layers perform classification based on the learned features.
- The "Output Layer" produces the final predictions or classifications.

CNN Architecture



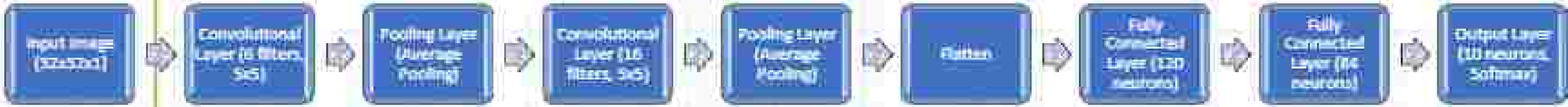
LeNet architecture

- LeNet is a pioneering convolutional neural network (CNN) architecture developed by Yann LeCun in 1998.
- It was primarily designed for handwritten digit recognition tasks.
- The architecture consists of seven layers, including three convolutional layers, two subsampling layers (pooling layers), and two fully connected layers.
- LeNet achieved remarkable success in character recognition and laid the foundation for modern CNN architectures.


(32,32)
1
filters
(6)
(5,5)
(2×1)
(2)
(28,28,6)
(14,14,6)
2
(16)
(5×5)
(2×2)
(2)
(10,10,16)
(5,5,16) → 400×120
2
3
120 neurons
4
84 neurons
5
120×84
84×10
LeNET-5
data will move

Deep Leaning

LeNet architecture



Input Image (32x32x1)

Convolutional Layer (6 filters, 5x5)

Pooling Layer (Average Pooling)

Convolutional Layer (16 filters, 5x5)

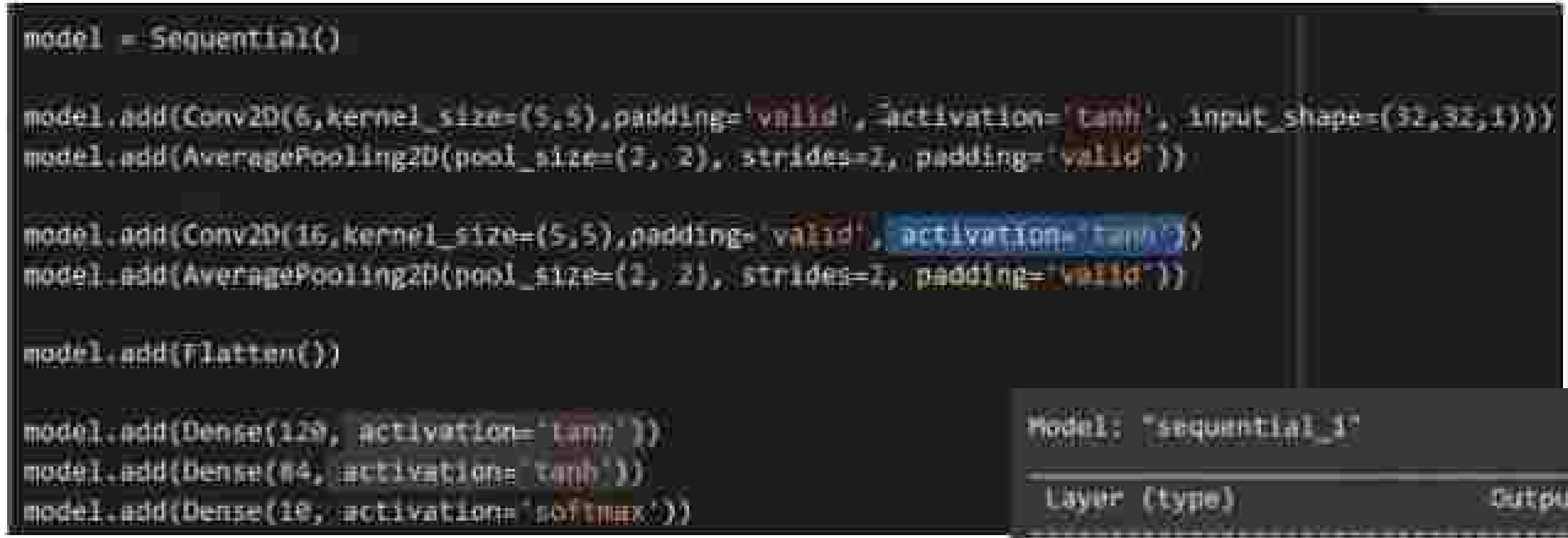
Pooling Layer (Average Pooling)

Flatten

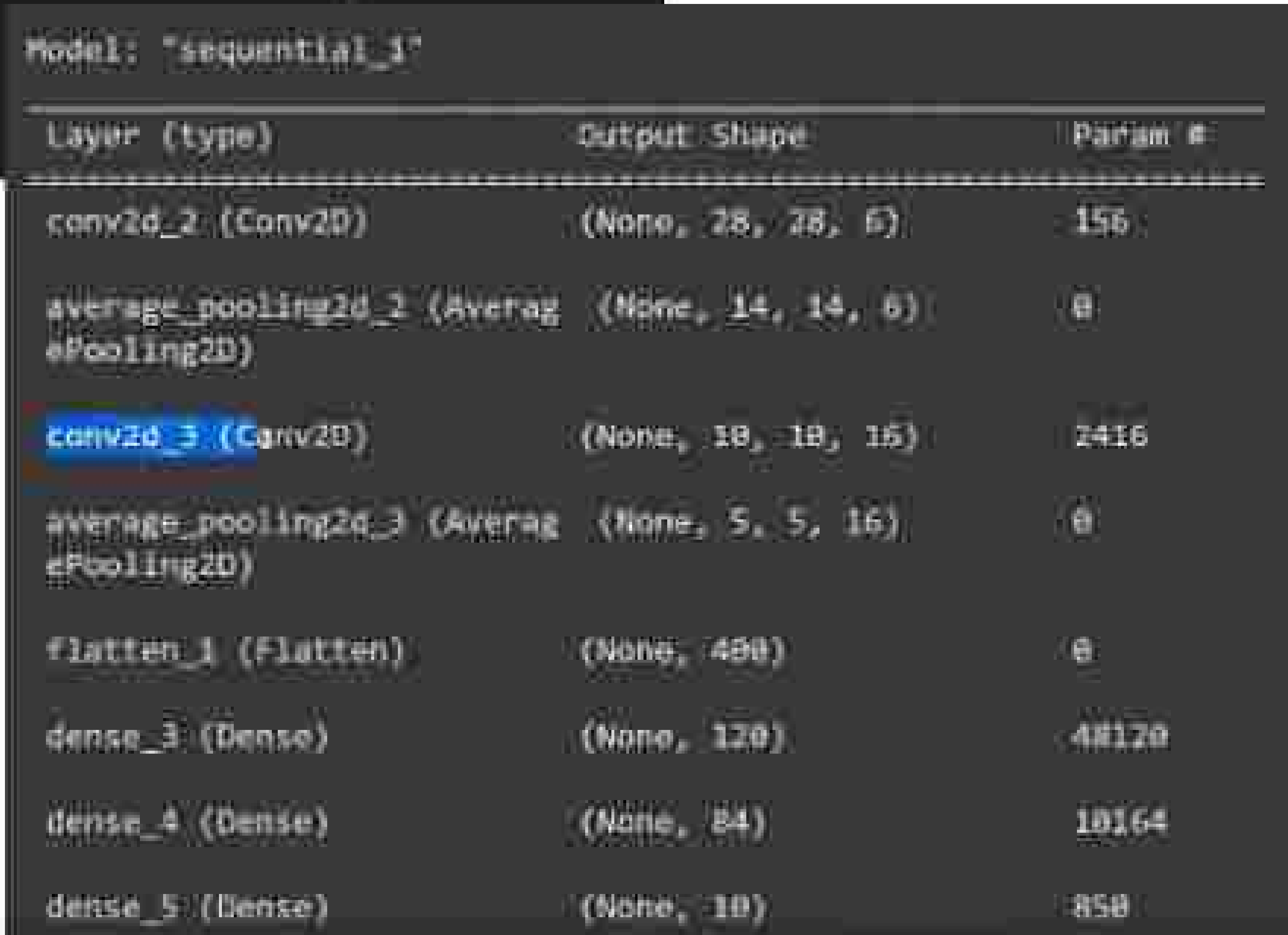
Fully Connected Layer (120 neurons)

Fully Connected Layer (84 neurons)

Output Layer (10 neurons, Softmax)



```
model = Sequential()

model.add(Conv2D(6,kernel_size=(5,5),padding='valid',activation='tanh', input_shape=(32,32,1)))
model.add(AveragePooling2D(pool_size=(2, 2), strides=2, padding='valid'))

model.add(Conv2D(16,kernel_size=(5,5),padding='valid', activation='tanh'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=2, padding='valid'))

model.add(Flatten())

model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(10, activation='softmax'))
```



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 6)         156

 average_pooling2d_2 (Averag  (None, 14, 14, 6)        0
 ePooling2D)

 conv2d_3 (Conv2D)           (None, 10, 10, 16)        2416

 average_pooling2d_3 (Averag  (None, 5, 5, 16)         0
 ePooling2D)

 flatten_1 (Flatten)         (None, 400)               0

 dense_3 (Dense)             (None, 120)               48120

 dense_4 (Dense)             (None, 84)                10164

 dense_5 (Dense)             (None, 10)                850
```

LeNet architecture

- **Input Image (32x32x1):** The architecture starts with the input image, which is typically a grayscale image of size 32x32 pixels.

- **Convolutional Layer (6 filters, 5x5):** The first convolutional layer applies six 5x5 filters to the input image. Each filter extracts different features from the input image, creating six feature maps.

- **Pooling Layer (Average Pooling):** After each convolutional layer, a pooling layer is applied to reduce the spatial dimensions of the feature maps. In LeNet, average pooling with a kernel size of 2x2 is commonly used.

- **Convolutional Layer (16 filters, 5x5):** Another convolutional layer follows the pooling layer, applying 16 filters of size 5x5 to the feature maps generated from the previous layer.

- **Pooling Layer (Average Pooling):** Similar to before, average pooling is applied to reduce the spatial dimensions of the feature maps.

LeNet architecture

- **Flatten:** After the convolutional and pooling layers, the feature maps are flattened into a one-dimensional vector. This prepares the data for feeding into the fully connected layers.

- **Fully Connected Layer (120 neurons):** The flattened features are then passed through a fully connected layer with 120 neurons. Each neuron in this layer is connected to every element in the flattened feature vector.

- **Fully Connected Layer (84 neurons):** Another fully connected layer with 84 neurons follows the previous layer.

- **Output Layer (10 neurons, Softmax):** The final layer of the LeNet architecture is the output layer, consisting of 10 neurons corresponding to the possible classes in the classification task. The softmax activation function is commonly used here to convert the raw scores into class probabilities.

AlexNet Architecture

Input Image (227x227x3)

↓

[Convolutional Layer]
Output: 55x55x96
Kernel Size: 11x11, Stride: 4, Padding: Valid
Activation: ReLU

↓

[Max Pooling Layer]
Output: 27x27x96
Pool Size: 3x3, Stride: 2

↓

[Convolutional Layer]
Output: 27x27x256
Kernel Size: 5x5, Stride: 1, Padding: Same
Activation: ReLU

↓

[Max Pooling Layer]
Output: 13x13x256
Pool Size: 3x3, Stride: 2

↓

[Convolutional Layer]
Output: 13x13x384
Kernel Size: 3x3, Stride: 1, Padding: Same
Activation: ReLU

↓

[Convolutional Layer]
Output: 13x13x384
Kernel Size: 3x3, Stride: 1, Padding: Same
Activation: ReLU

↓

[Convolutional Layer]
Output: 13x13x256
Kernel Size: 3x3, Stride: 1, Padding: Same
Activation: ReLU

↓

[Max Pooling Layer]
Output: 6x6x256
Pool Size: 3x3, Stride: 2

↓

[Flattening Layer]
Output: 9216 (6x6x256)

↓

[Fully Connected Layer]
Output: 4096
Activation: ReLU

↓

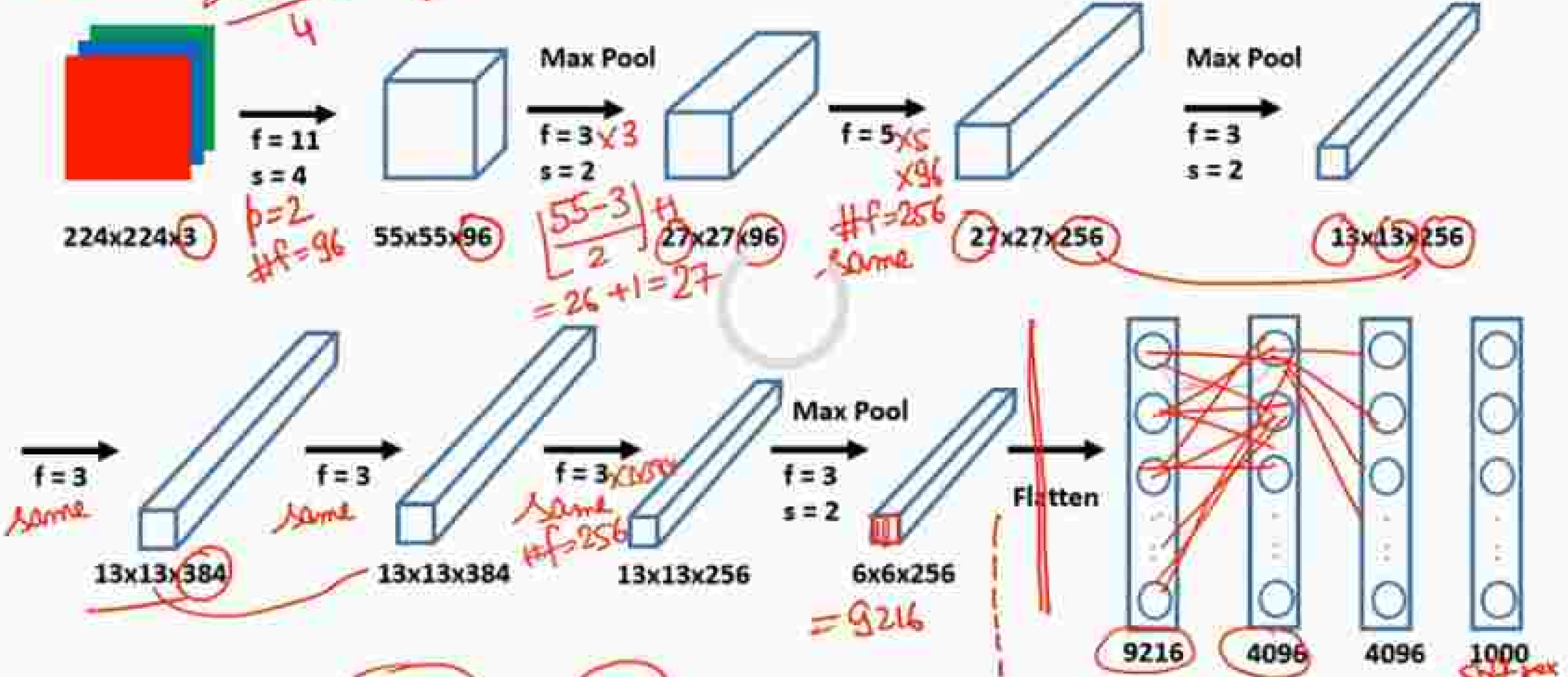
[Fully Connected Layer]
Output: 4096
Activation: ReLU

↓

[Output Layer]
Output: 1000 (Number of classes in ImageNet)
Activation: Softmax

AlexNet

$\frac{224+4-11}{4} = \left\lfloor \frac{217}{4} \right\rfloor + 1 = 54+1 = 55$



Krizhevsky et al., 2012. ImageNet Classification with Deep Convolutional Neural Netwo

AlexNet Architecture

- AlexNet is a pioneering convolutional neural network (CNN) architecture that played a significant role in advancing the field of deep learning, particularly in computer vision tasks.
- It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012.

- **Input Layer:** Accepts input images of size 227x227x3 (RGB images).

- **Convolutional Layers:** AlexNet consists of five convolutional layers. Each convolutional layer is followed by a ReLU activation function. These layers learn hierarchical features from the input images.

- **Max Pooling Layers:** Four max-pooling layers are interspersed between the convolutional layers. They downsample the feature maps to reduce spatial dimensions while retaining important information.

AlexNet Architecture

- **Fully Connected Layers:** Following the convolutional and pooling layers are three fully connected layers. These layers integrate the learned features and perform classification based on the extracted features.
- **Output Layer:** The final layer is a softmax activation layer that produces class probabilities. In the original AlexNet, it outputs probabilities for 1000 classes from the ImageNet dataset.
- **Dropout:** Dropout regularization is applied before the fully connected layers to prevent overfitting by randomly dropping neurons during training.
- AlexNet demonstrated the effectiveness of deep learning in image classification tasks and paved the way for subsequent advancements in CNN architectures.