University of Lancashire

Lecture 14

ConvNets-II

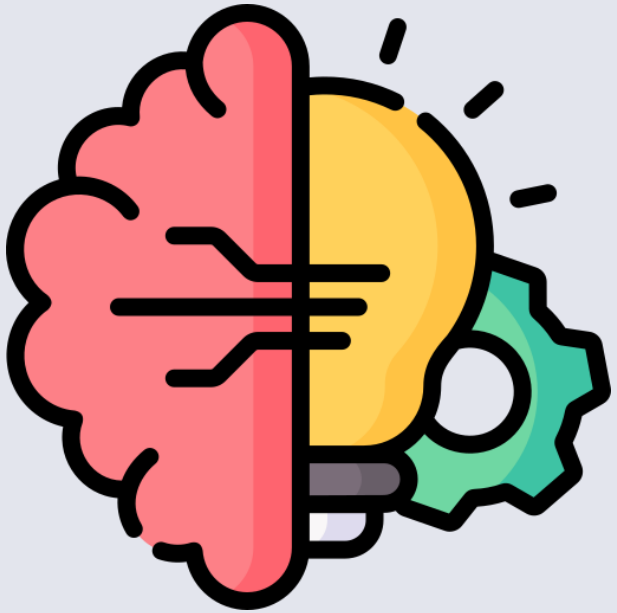
Course overview

- **Learning Objectives**
  - Introduce and familiarise you with the additional knowledge about ConvNet
  - Learning how complete CNN blocks works and the tackling the main issues
  - You will be able to build an understanding of the deep learning frameworks
  - Familiarise you about the metrics related to the recognition task
- **Important Directions**

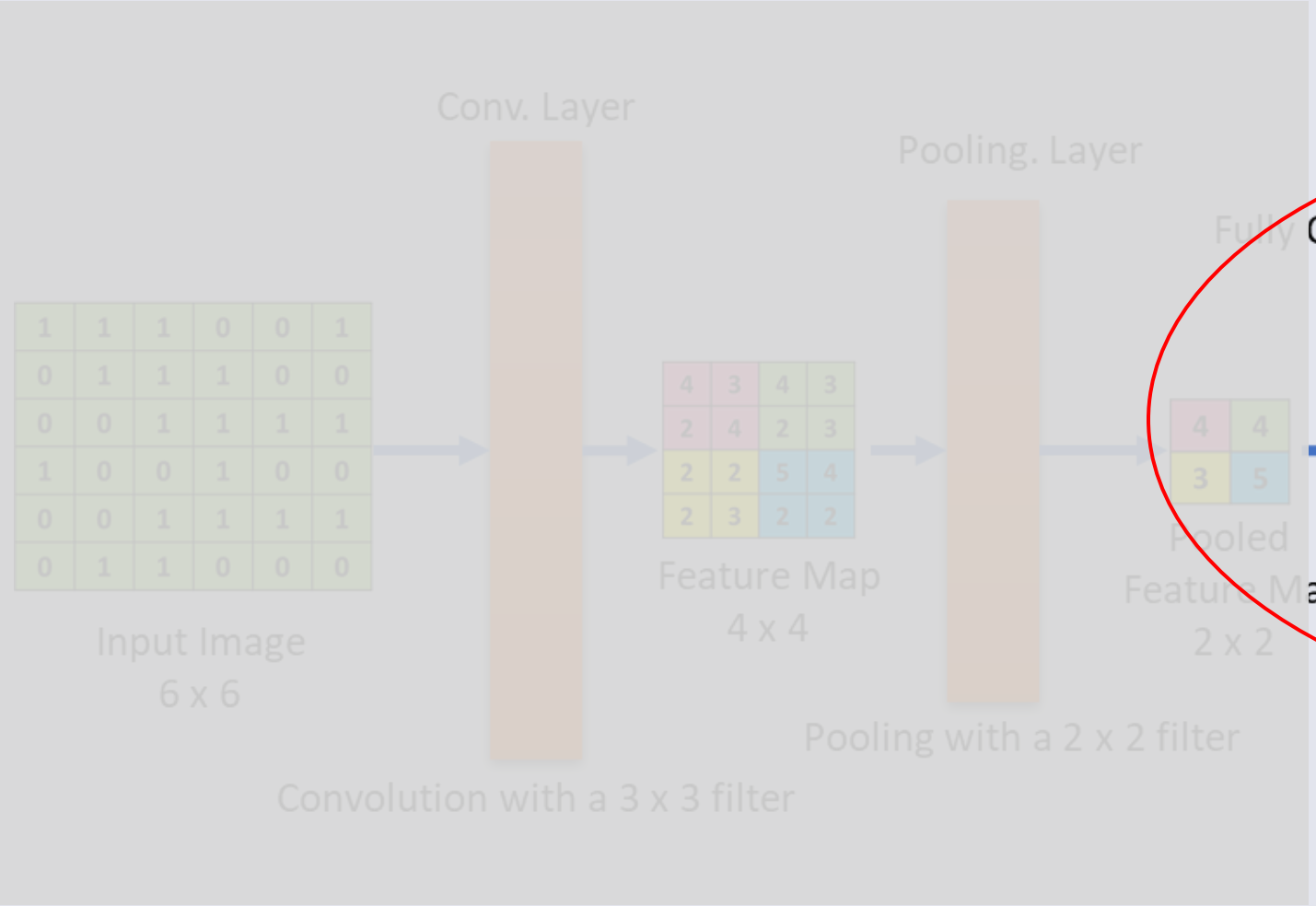


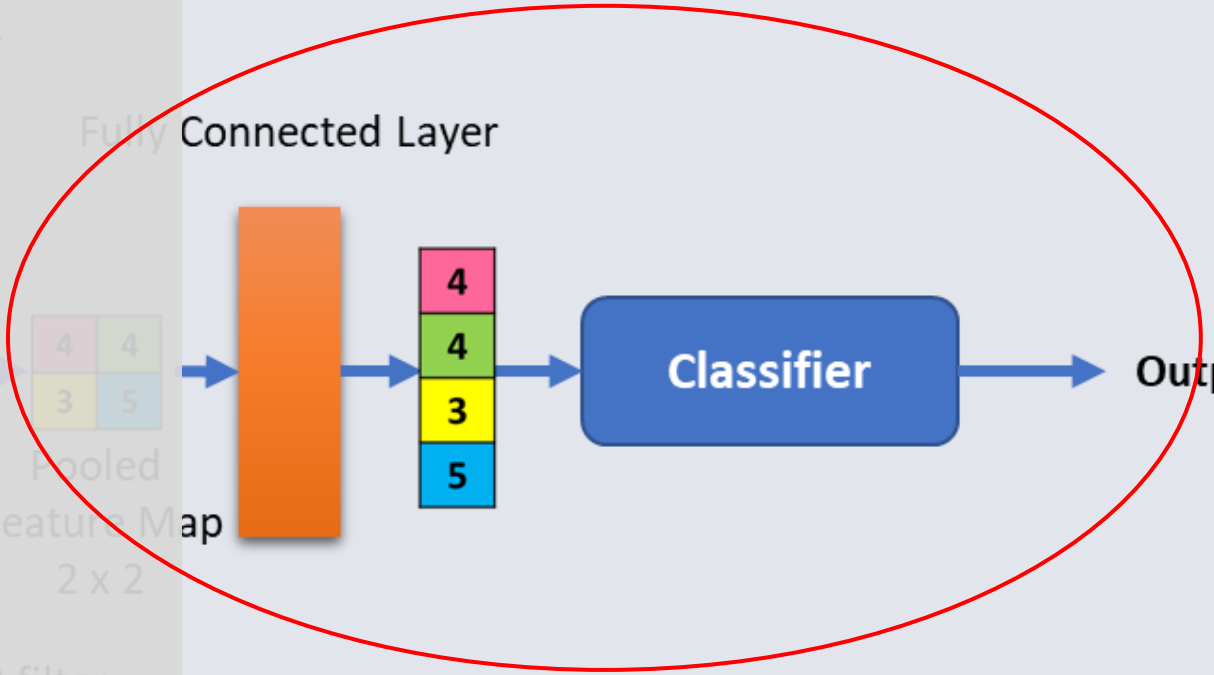
Today's Contents

- Deep Learning Frameworks
- Impact of change in the size of strides
- CNN final Components
- Model Development
- Model Evaluation
- 1D CNN



Recap the previous lecture





Deep Learning Frameworks

Deep learning frameworks are essential tools that simplify the process of designing, training, and deploying neural networks. Instead of implementing complex mathematical operations from scratch, these frameworks provide pre-built functions, optimized computation, and GPU acceleration to speed up development

1. **TensorFlow**: A highly scalable framework from Google.
2. **Keras** :A user-friendly API for deep learning.
3. **PyTorch** : A flexible and research-oriented framework from Facebook.
4. **Caffe:** A lightweight, fast framework for image processing.

Other Frameworks (MXNet, Theano, CNTK): Specialized frameworks with unique advantages

Deep Learning Frameworks

TensorFlow: An open-source deep learning framework developed by Google Brain.

Widely used for:

- Image and speech recognition
- Natural Language Processing (NLP)
- Reinforcement Learning

```
[1]  1 import tensorflow as tf
     2 from tensorflow import keras
     3 from tensorflow.keras import layers
     4 import numpy as np
     5 import matplotlib.pyplot as plt
```

What does it provide?

- Efficient computation using dataflow graphs.
- Multi-GPU and TPU support for large-scale AI projects.
- TensorBoard for real-time visualization of model performance.
- Pre-trained models in TensorFlow Hub.

```
Import tensorflow as tf

tf.__version__
```

```
# For GPU users
pip install tensorflow[and-cuda]
# For CPU users
pip install tensorflow
```

Deep Learning Frameworks

Keras: High-level deep learning API that runs on top of TensorFlow, making it easier to build and train deep learning models.

Widely used for

```
[1]  1 import tensorflow as tf
     2 from tensorflow import keras
     3 from tensorflow.keras import layers
     4 import numpy as np
     5 import matplotlib.pyplot as plt
```

What does it provide?

- Simple, intuitive syntax for building neural networks.
- Support for multiple backends like TensorFlow, Theano, and CNTK.
- Built-in pre-trained models like ResNet, VGG, and Inception.
- Rapid prototyping for beginners and professionals alike

```
Import keras

keras.__version__
```

```
Pip install keras
```

Deep Learning Frameworks

Pytorch: Open-source deep learning framework developed by Facebook AI Research (FAIR).

Used in research and production due to its dynamic computation graphs and intuitive debugging

```
Import torch
torch.__version__
```

What does it provide?

- Dynamic computation graphs for flexible model building.
- Strong GPU acceleration with simple commands.
- Better debugging compared to static graph-based frameworks.
- TorchScript for converting models to production-ready versions.

```
Import keras

keras.__version__
```

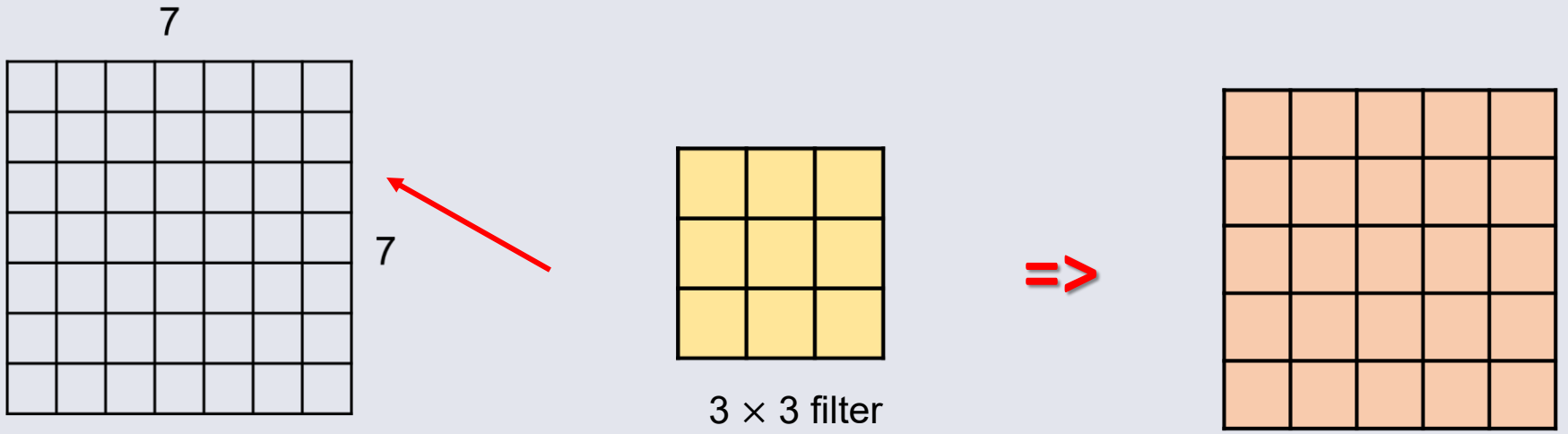
Deep Learning Frameworks

Year	Framework	Description
2002	**Torch**	Open-source deep learning library based on Lua, precursor to PyTorch.
2007	**Theano**	One of the earliest deep learning libraries, developed by the Montreal Institute for Learning Algorithms (MILA).
2013	**Caffe**	Deep learning framework developed by the Berkeley Vision and Learning Center (BVLC).
2014	**DL4J (Deeplearning4j)**	Deep learning framework for Java, used in production environments.
2015	**Chainer**	Deep learning framework developed by Preferred Networks, focusing on flexibility and performance.
2015	**MXNet**	Open-source deep learning framework supported by Apache Software Foundation.
2015	**Keras**	High-level neural networks API, originally developed as a wrapper for TensorFlow.
2015	**TensorFlow**	Open-source library developed by Google for deep learning and machine learning.
2016	**PyTorch**	Open-source machine learning library developed by Facebook's AI Research lab.
2016	**CNTK (Microsoft Cognitive Toolkit)**	Deep learning framework developed by Microsoft, optimized for performance.
2017	**ONNX**	Open Neural Network Exchange; a cross-platform framework for model interchange.
2018	**Fastai**	High-level deep learning library built on top of PyTorch, designed to make training deep neural networks easy and fast.

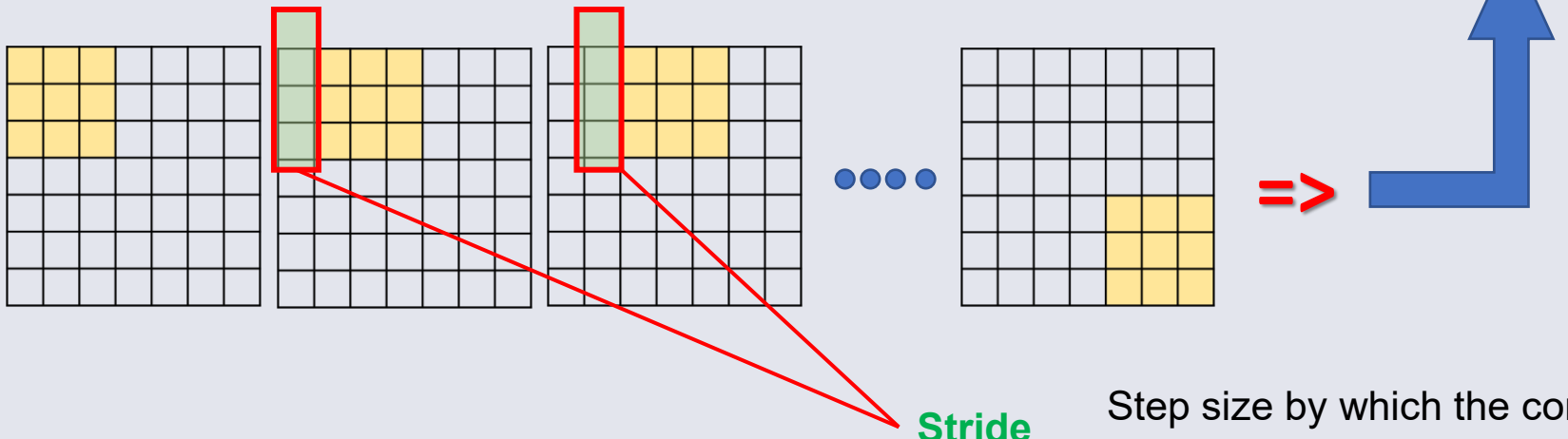
Spatial Dimensions: Closer Look

Now let see about the stride in convolution

Spatial Dimensions: Closer Look

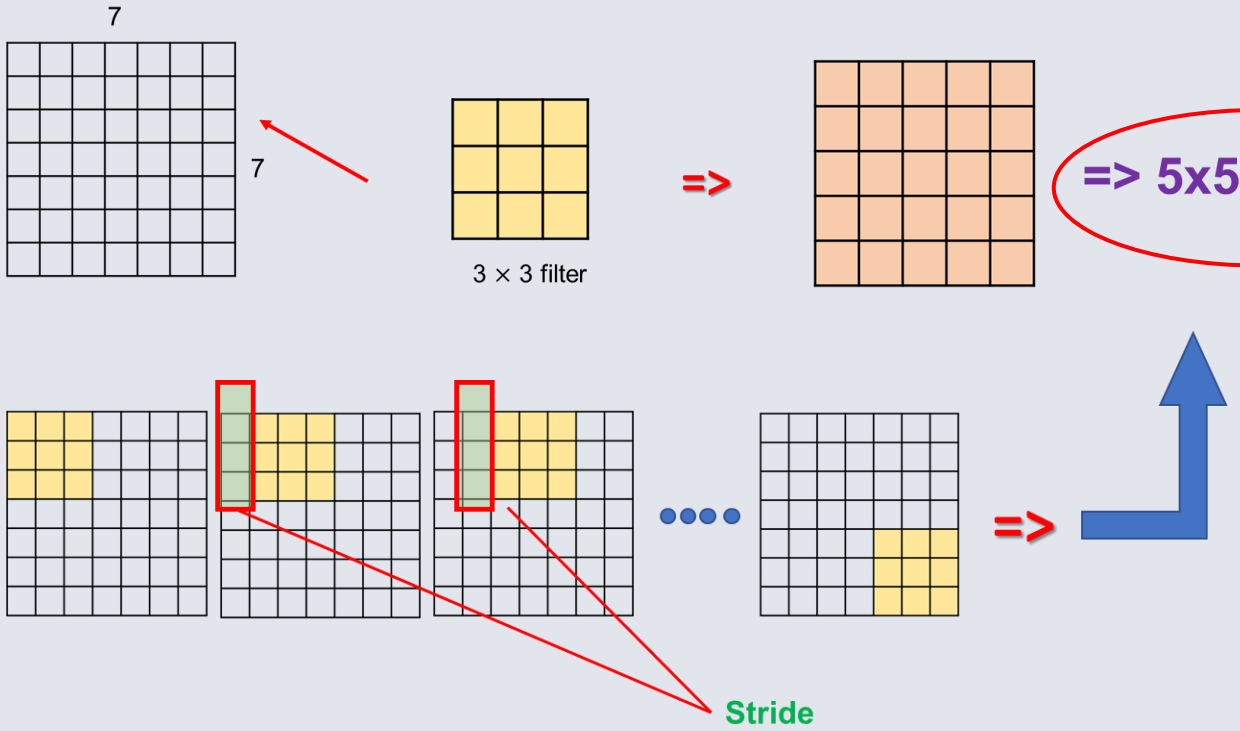


=> 5x5 output



Step size by which the convolutional filter (or kernel) moves across the input image. Here, it is one stride.

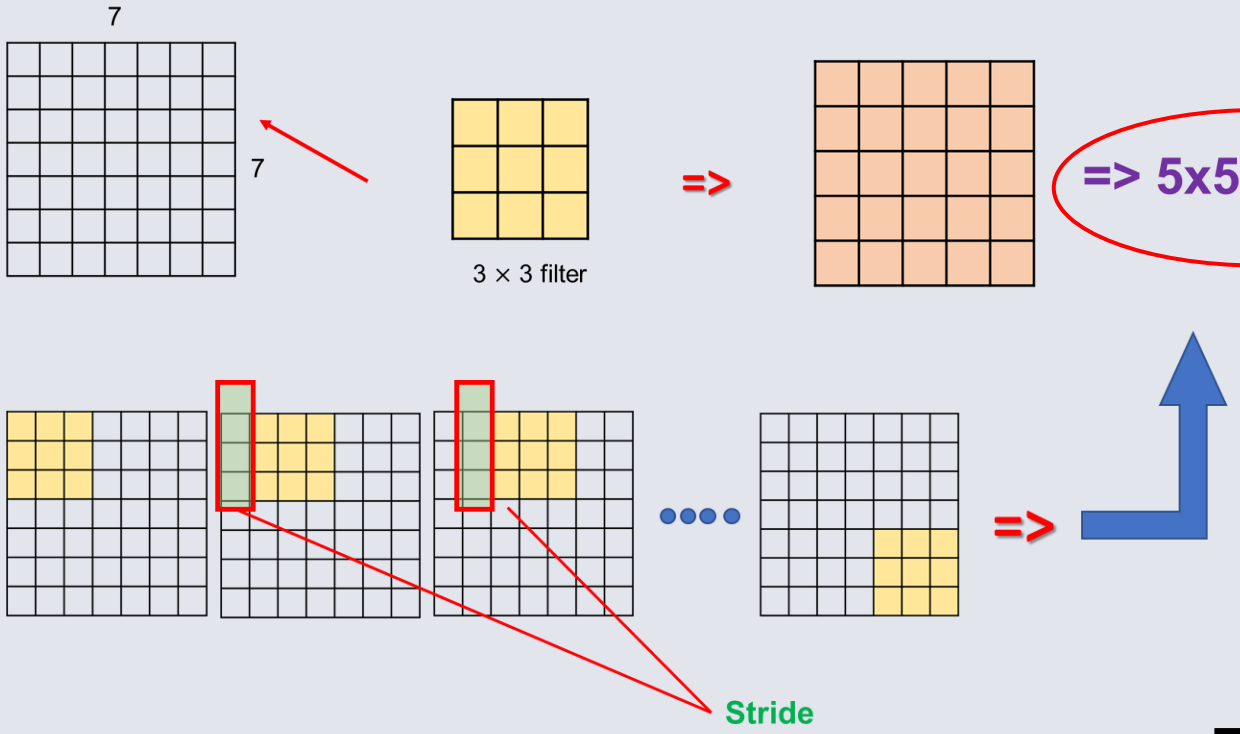
Spatial Dimensions: Closer Look



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

For an input of size **N×N**, a filter of size **F×F,** and stride S, the output size O (assuming no padding) is calculated as:

Spatial Dimensions: Closer Look



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

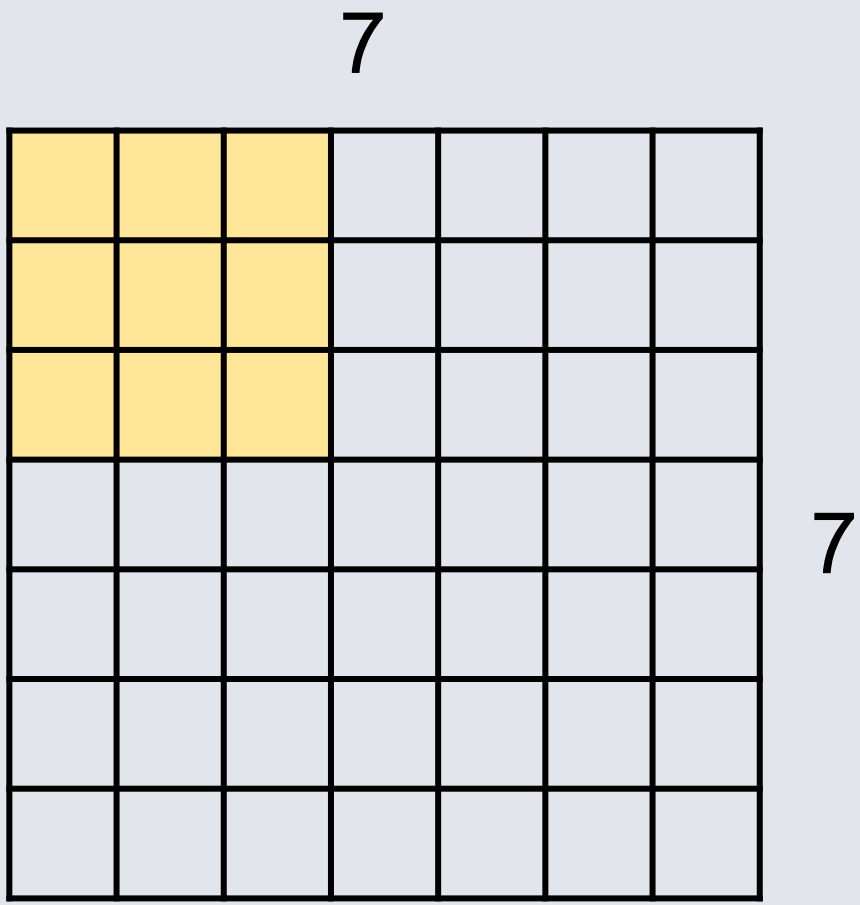
E.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

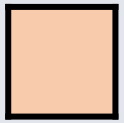
stride 2 => (7 - 3)/2 + 1 = 3

We saw example in previous lecture

Closer Look at the Spatial Dimensions

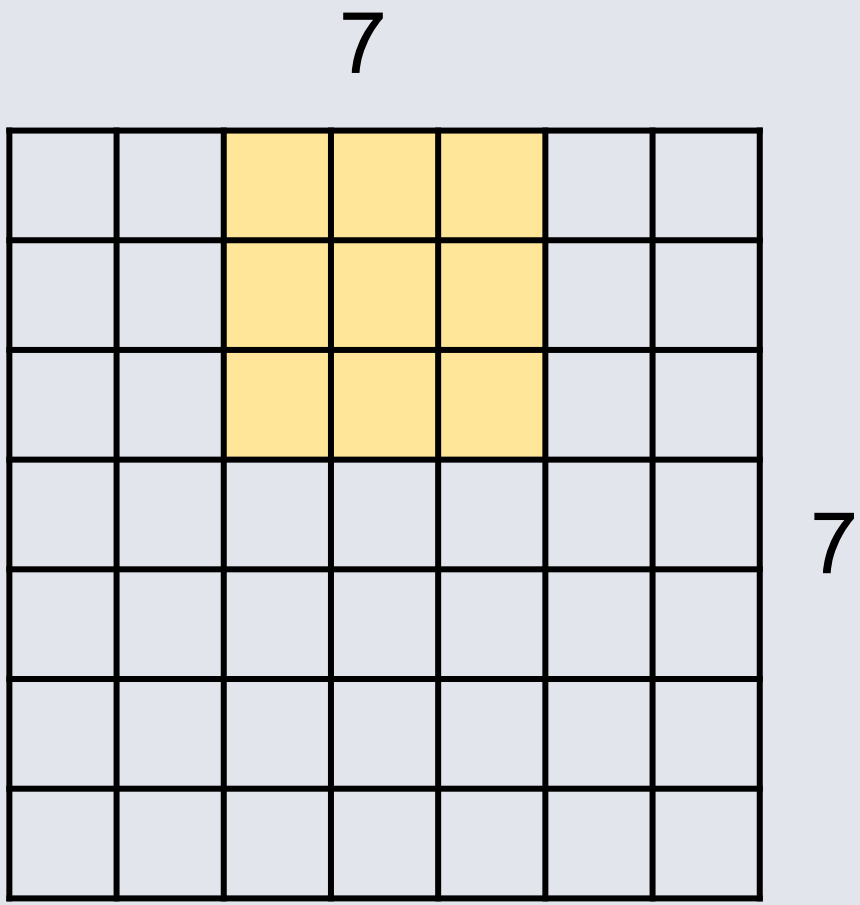


7×7 input (Spatially)

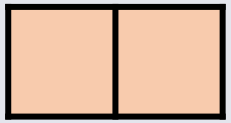


Stride = 2

Closer Look at the Spatial Dimensions

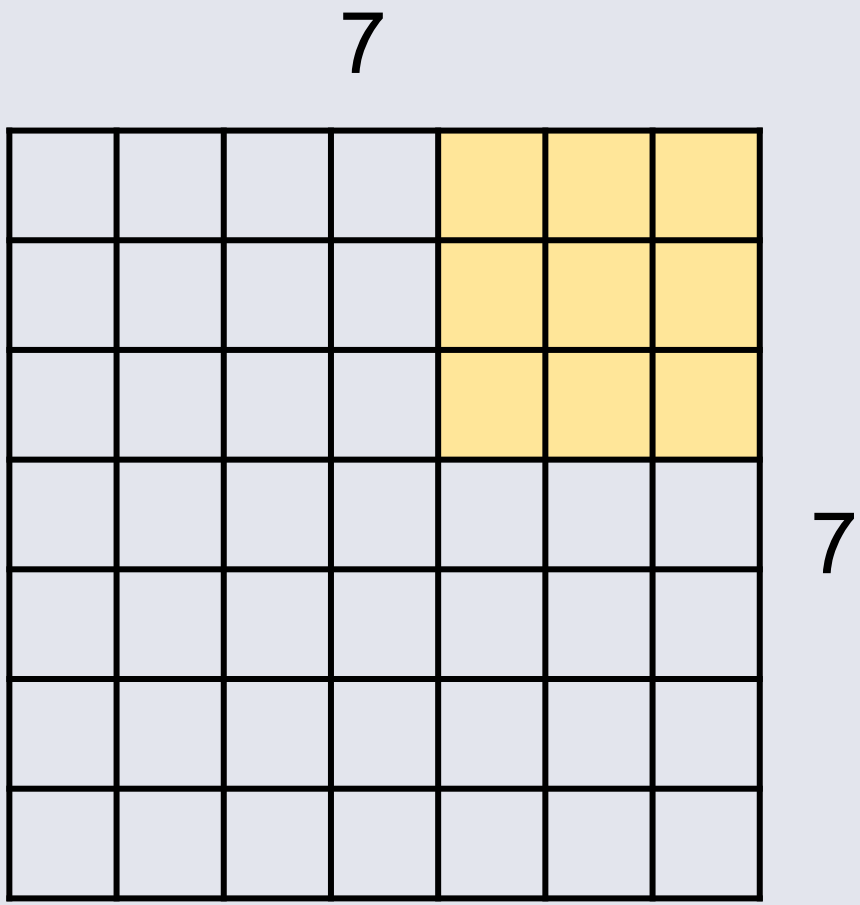


7×7 input (Spatially)

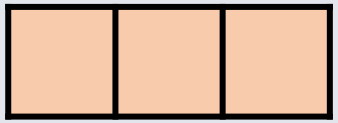


Stride = 2

Closer Look at the Spatial Dimensions

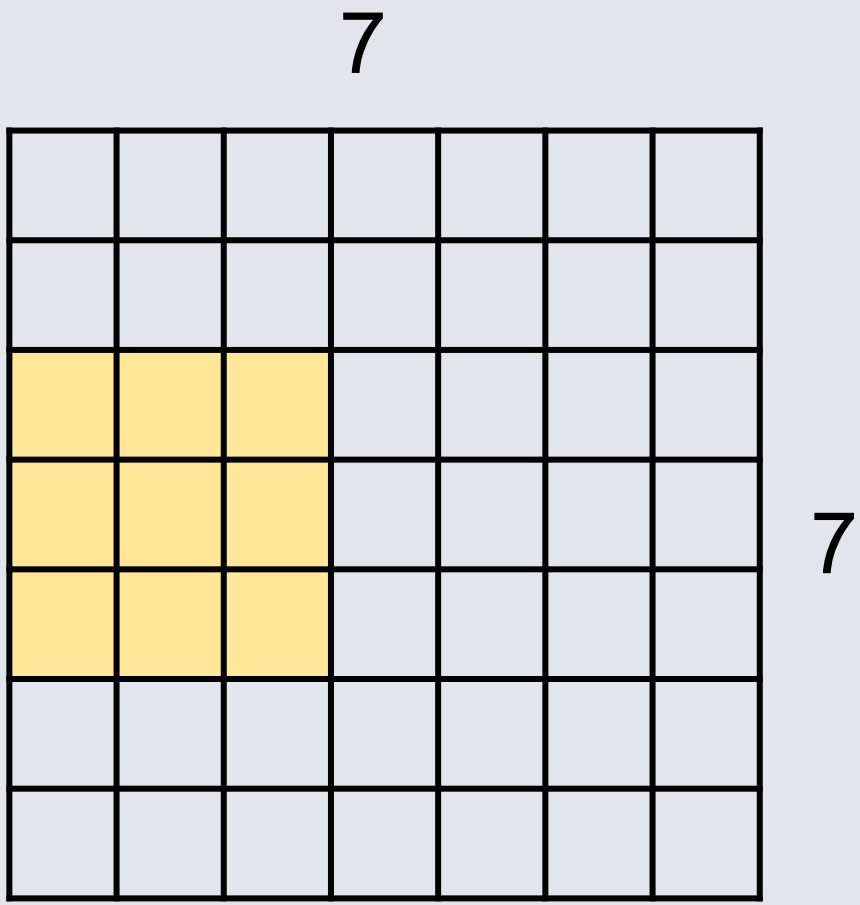


7×7 input (Spatially)

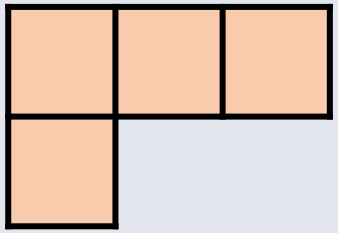


Stride = 2

Closer Look at the Spatial Dimensions

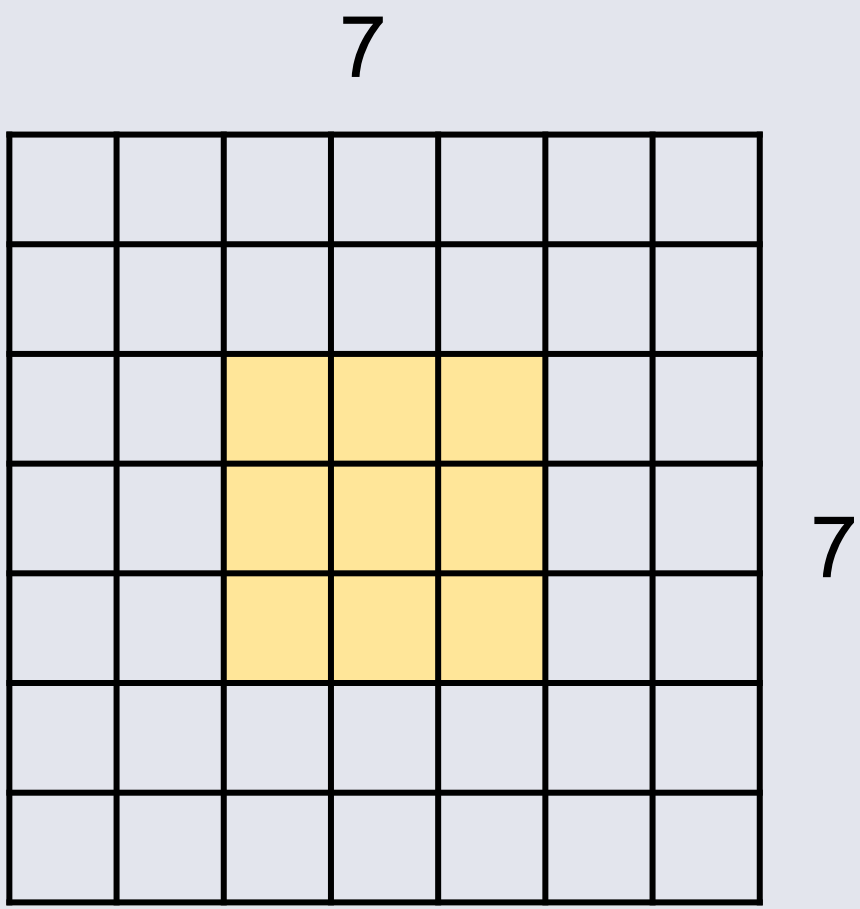


7×7 input (Spatially)

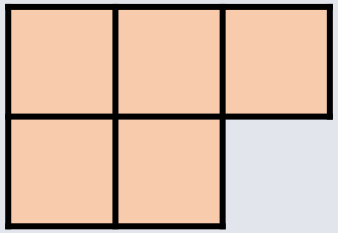


Stride = 2

Closer Look at the Spatial Dimensions

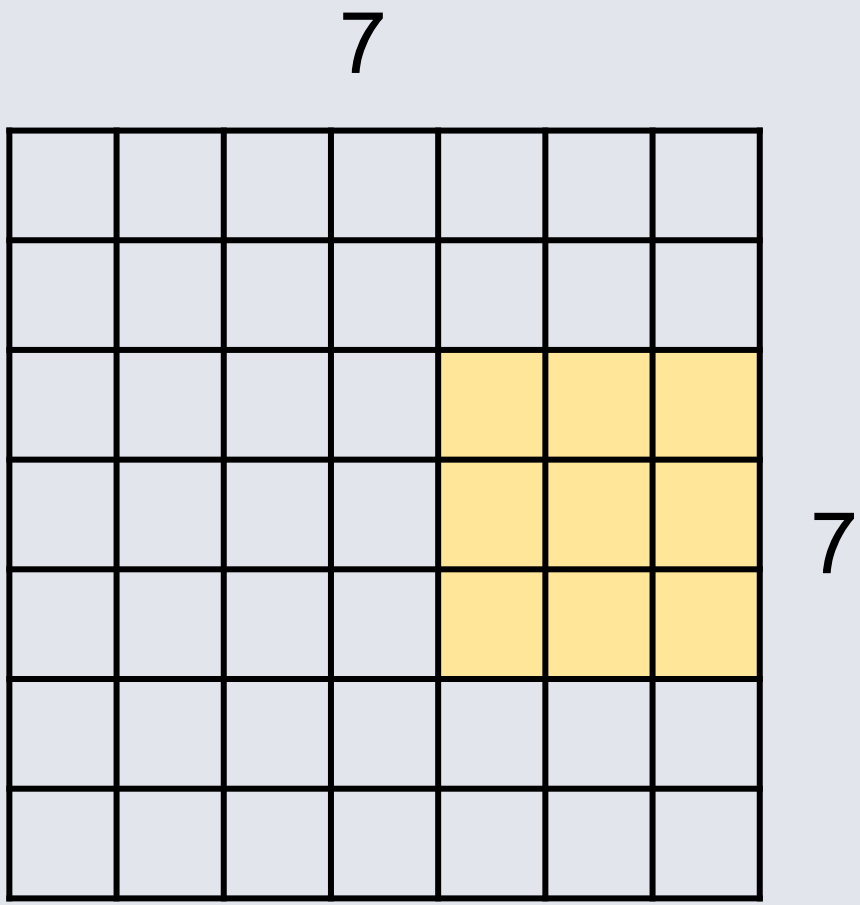


7×7 input (Spatially)

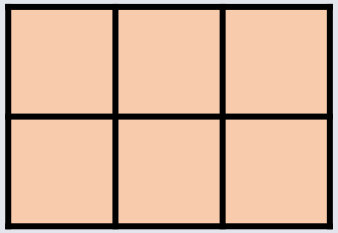


Stride = 2

Closer Look at the Spatial Dimensions

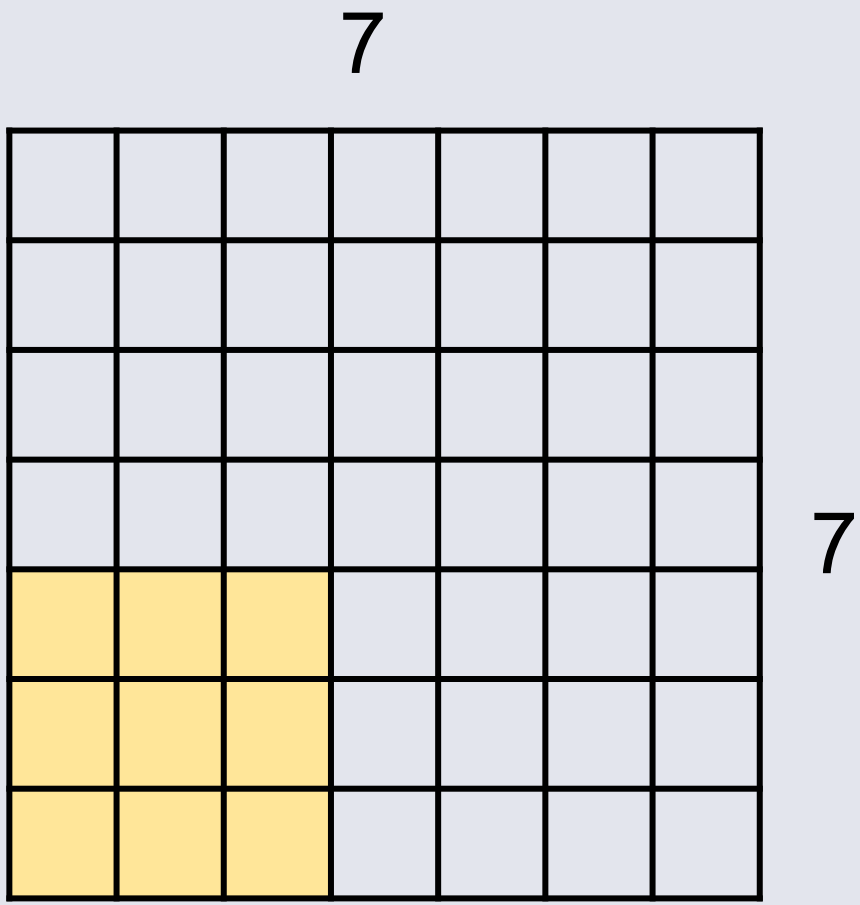


7×7 input (Spatially)

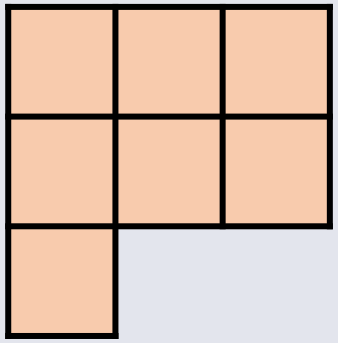


Stride = 2

Closer Look at the Spatial Dimensions

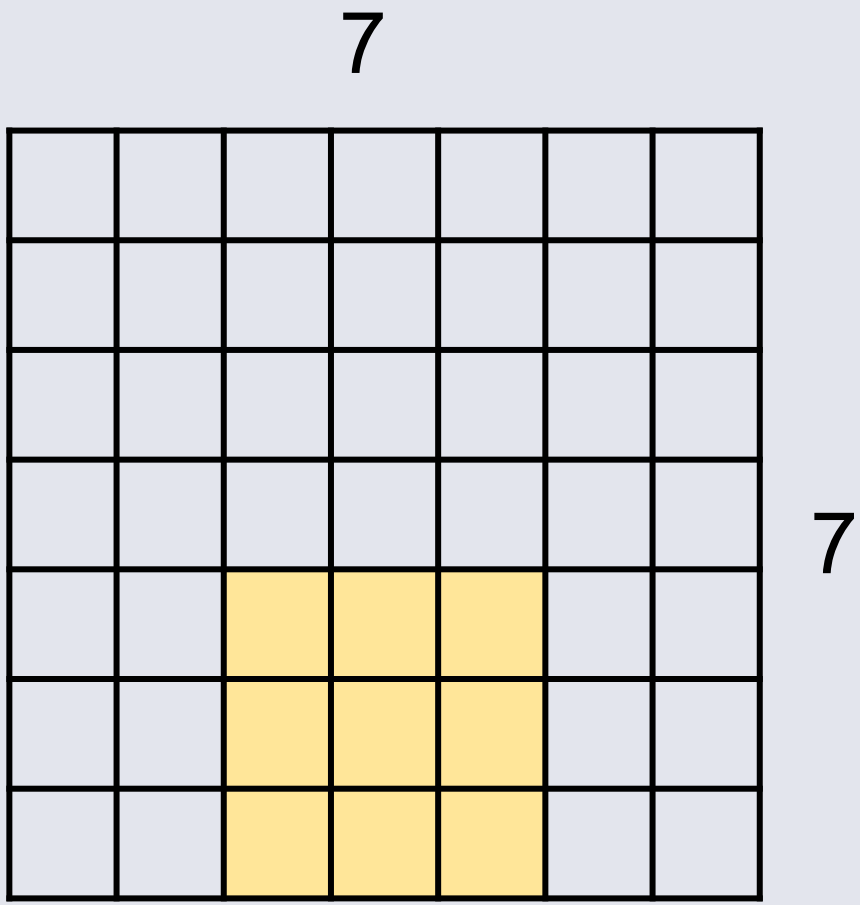


7×7 input (Spatially)

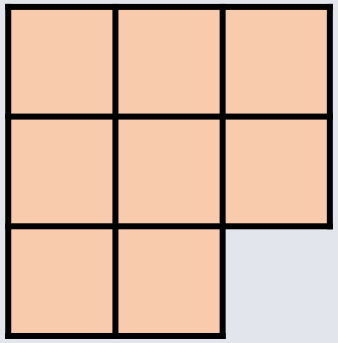


Stride = 2

Closer Look at the Spatial Dimensions

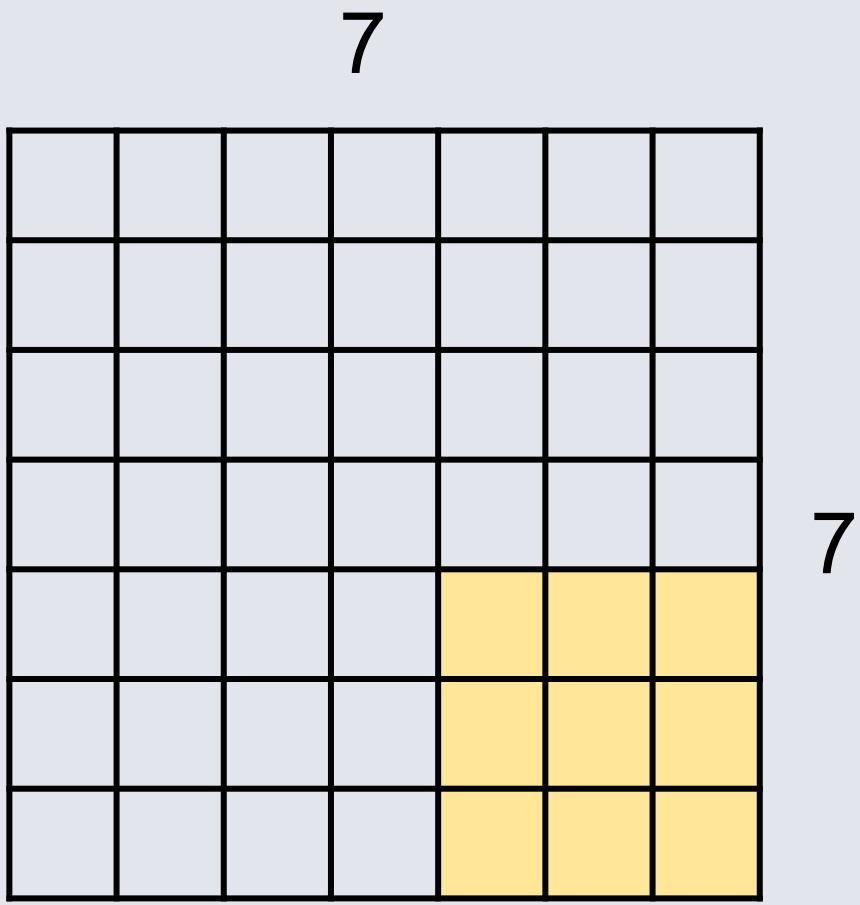


7×7 input (Spatially)

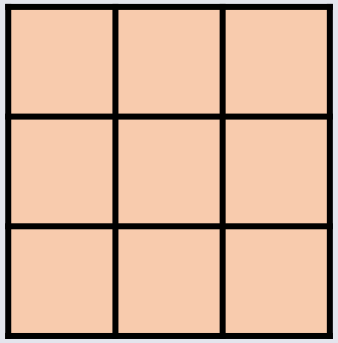


Stride = 2

Closer Look at the Spatial Dimensions

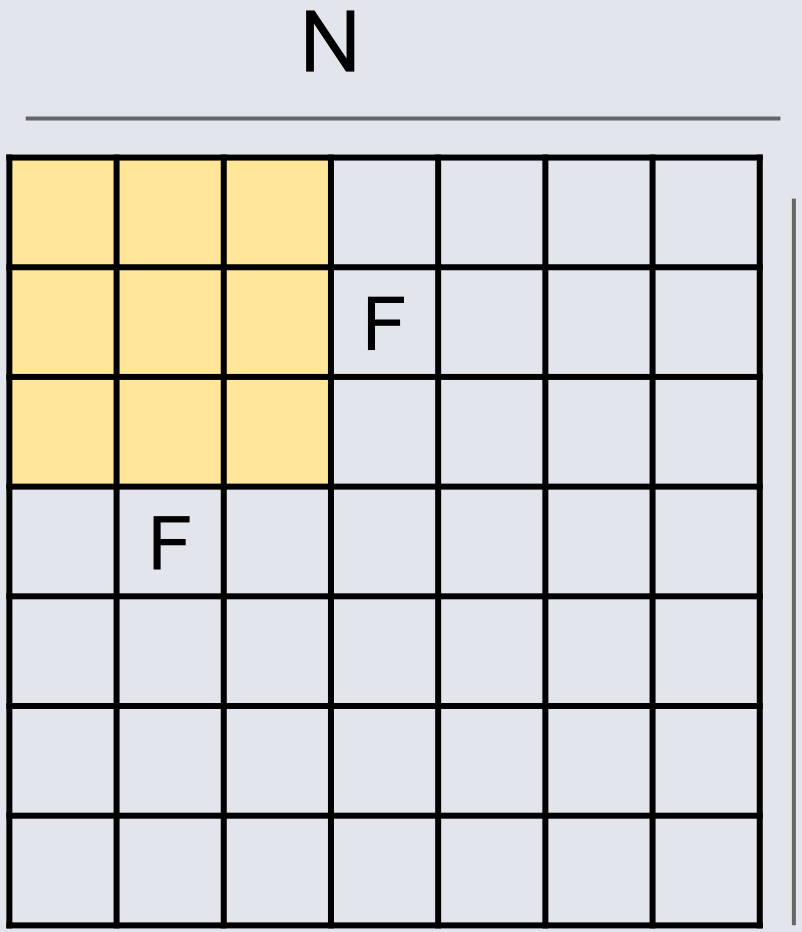


7×7 input (Spatially)



Stride = 2

Spatial Dimensions: Closer Look



What if stride is 3??

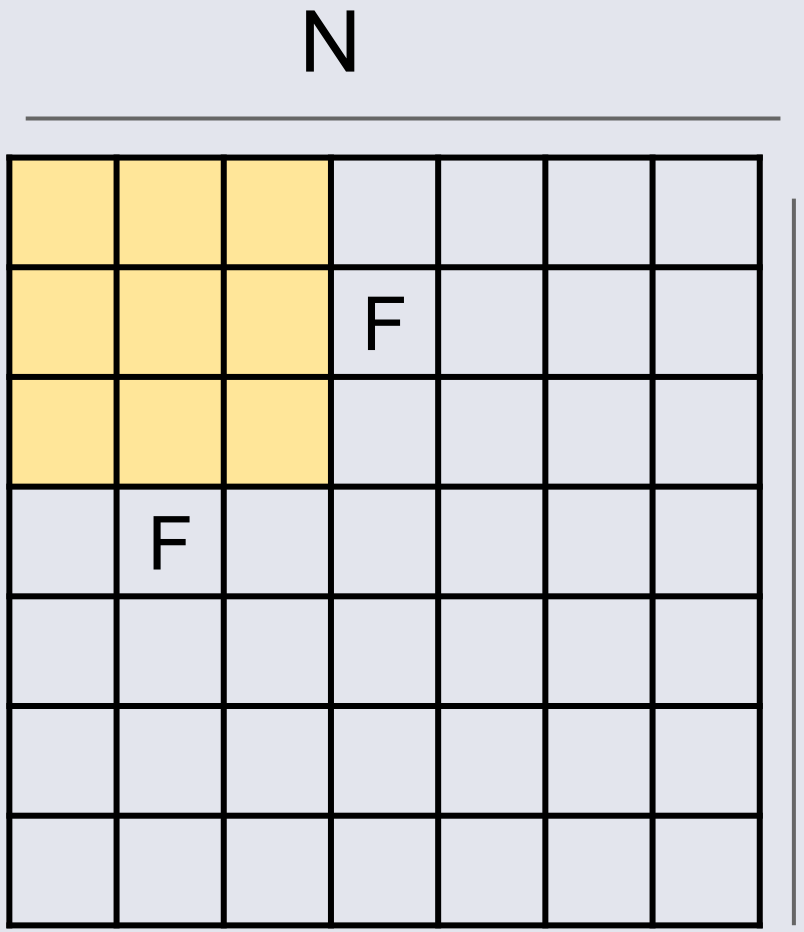
Output size: **(N - F) / stride + 1**

Stride 1 => (7 - 3)/1 + 1 = 5

Stride 2 => (7 - 3)/2 + 1 = 3

Stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Spatial Dimensions: Closer Look



What if stride is 3??

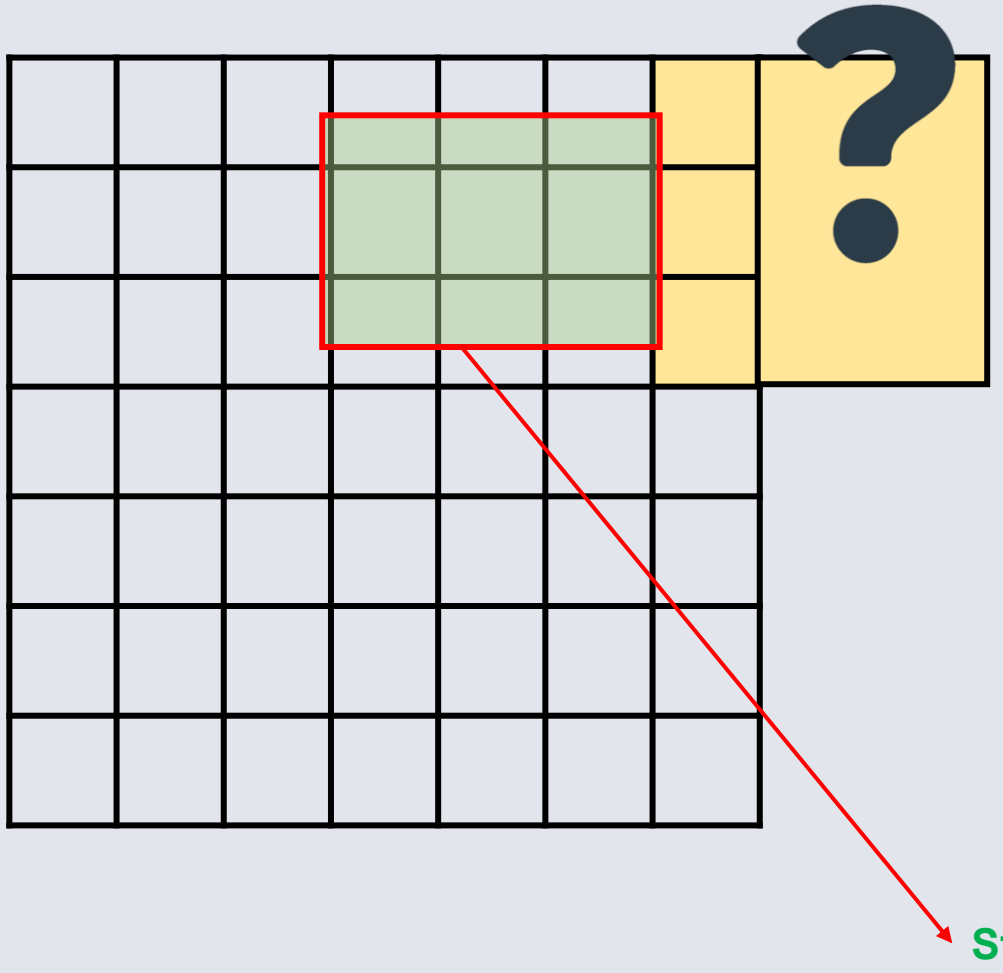
Output size: **(N - F) / stride + 1**

Stride 1 => (7 - 3)/1 + 1 = 5

Stride 2 => (7 - 3)/2 + 1 = 3

Stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Spatial Dimensions: Closer Look



What if stride is 3??

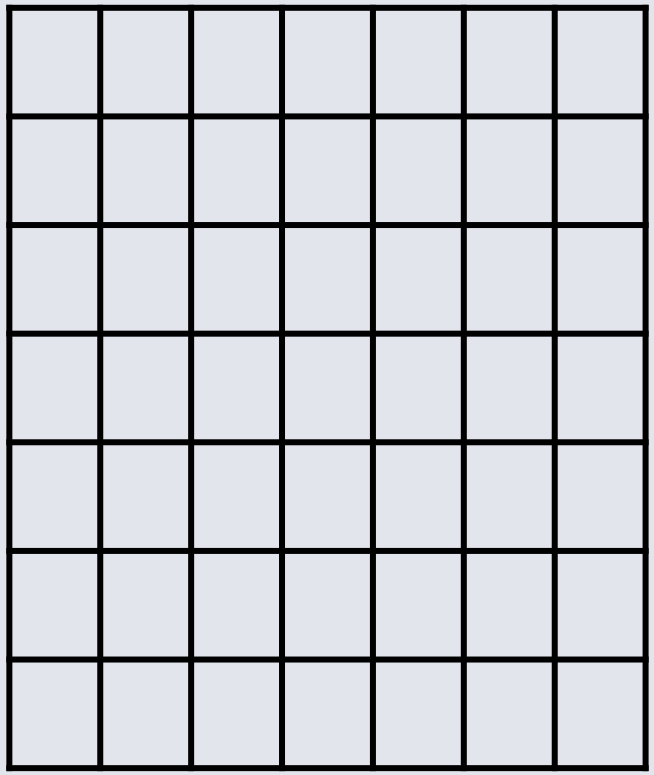
Output size: **(N - F) / stride + 1**

Stride 1 => (7 - 3)/1 + 1 = 5

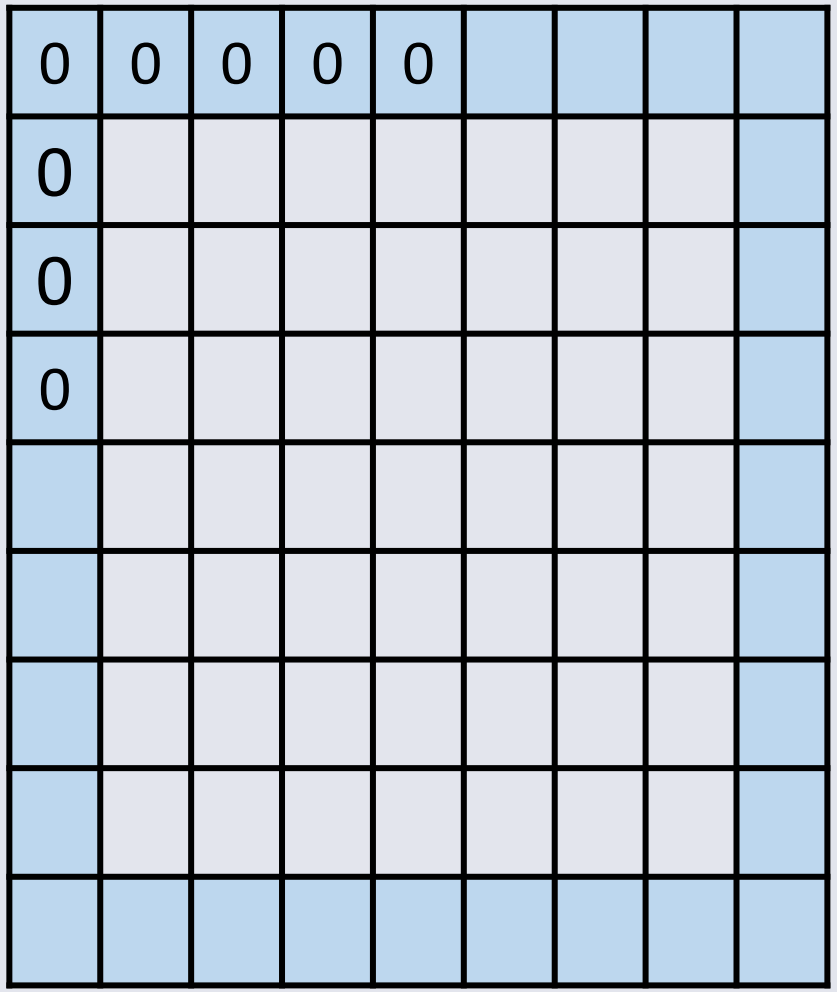
Stride 2 => (7 - 3)/2 + 1 = 3

Stride 3 => (7 - 3)/3 + 1 = 2.33 :\

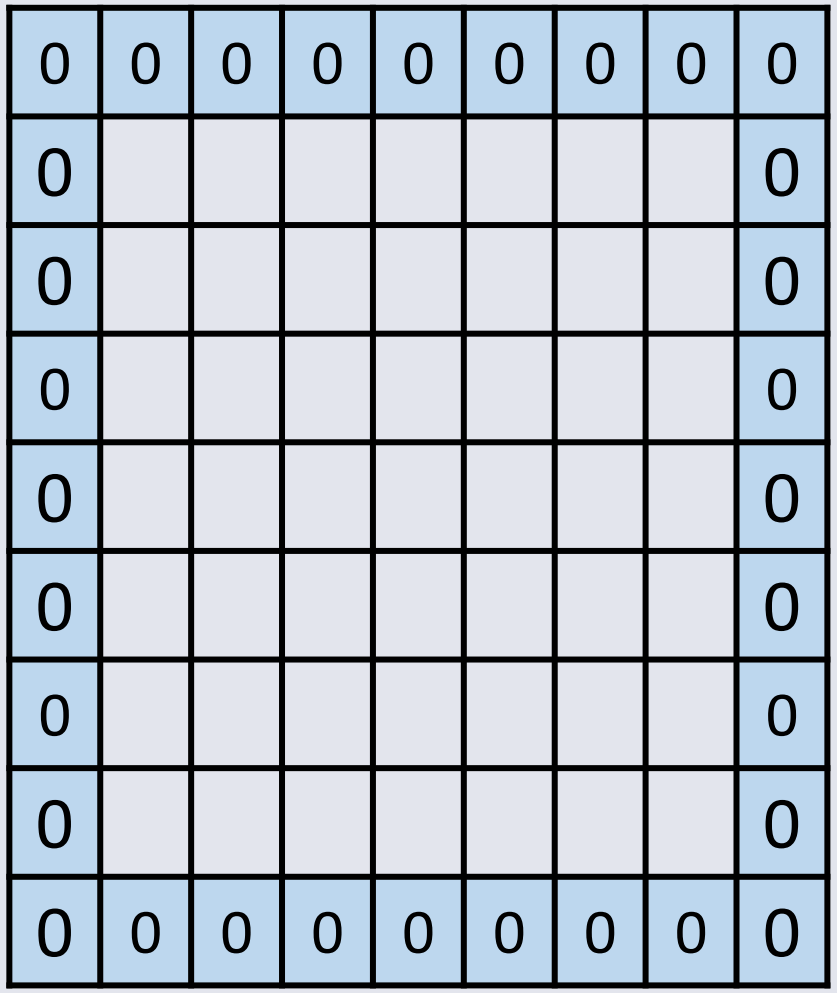
In practice: Common to zero pad the border



In practice: Common to zero pad the border



In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output

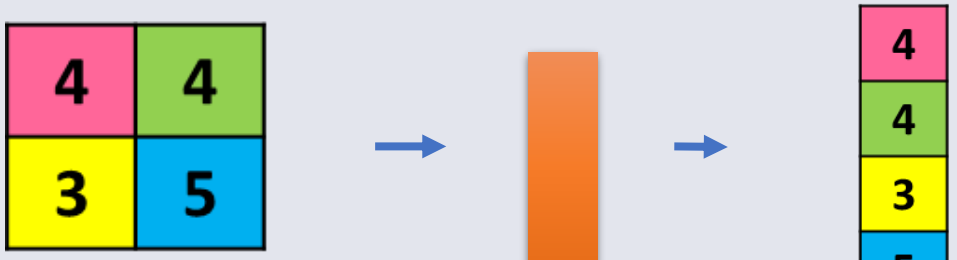
In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g.

F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Fully Connected FC Layer

- A layer where every neuron (node) is connected to every neuron in the previous layer.
- Takes input from all neurons in the previous layer and applies a transformation using weights and biases.
- The primary purpose of this layer is to take the high-level features extracted by the earlier layers (convolutional and pooling layers) and use them to make predictions,

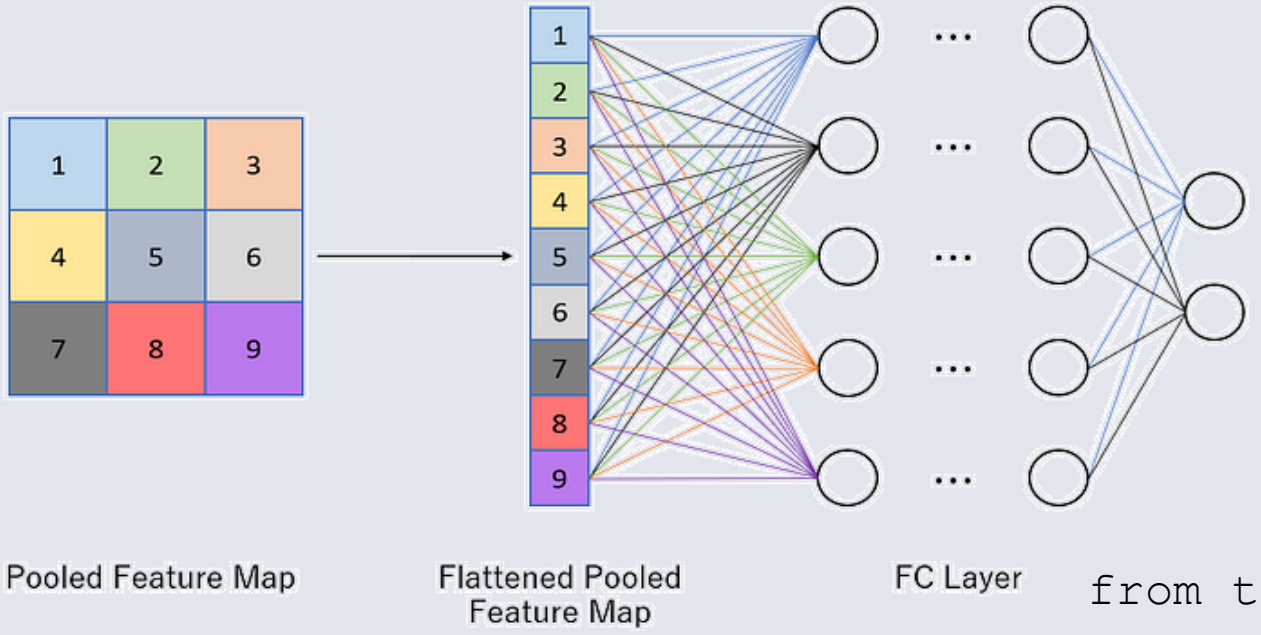


Pooled
Feature Map
2 x 2

Fully Connected Layer

Fully Connected FC Layer

- Imagine you have a convolutional output with dimensions (7, 7, 128), where 7x7 is the spatial dimension (height and width) and 128 is the depth (number of feature maps).
- Flattening this would result in a vector of length 7 * 7 * 128 = 6272.



High-Level Decision Making

Dimensionality Transformation

```
from tensorflow.keras.layers import Dense
fc_layer = Dense(units=128, activation='relu')
```

Dropout Layer

The Dropout Layer is a regularization technique used to prevent overfitting by randomly setting a fraction of input units to zero during training.

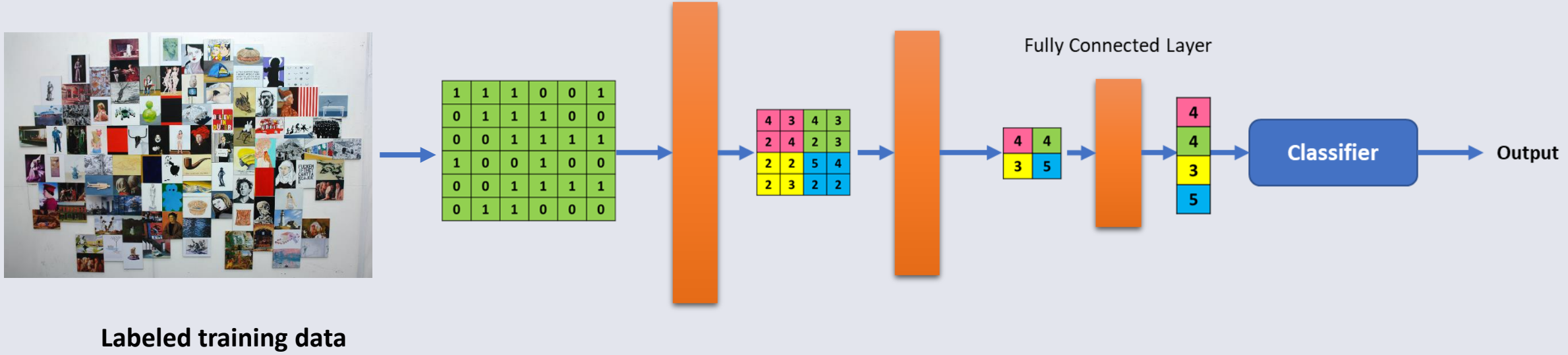
```
from tensorflow.keras.layers import Dropout
dropout_layer = Dropout(rate=0.5)  # Drops 50% of neurons randomly
```

It is not a standard part of the network or fully connected layer, but it is added to improve generalization by making the model more robust.

Comparison

Layer	Function	Key Use Case
Fully Connected (FC) Layer	Connects every neuron in the previous layer to every neuron in the next layer	Used in deep networks
Dense Layer	Same as FC Layer in Keras	Defines a fully connected layer in TensorFlow/Keras
Dropout Layer	Randomly deactivates neurons during training	Prevents overfitting
Softmax Layer	Converts logits into probabilities	Used in classification tasks

Model Development



Labeled training data

Model Development

```
[1]  1 import tensorflow as tf
     2 from tensorflow import keras
     3 from tensorflow.keras import layers
     4 import numpy as np
     5 import matplotlib.pyplot as plt
```

```
[3]  1 model = keras.Sequential([
     2     layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
     3     layers.MaxPooling2D((2,2)),
     4     layers.Conv2D(64, (3,3), activation='relu'),
     5     layers.MaxPooling2D((2,2)),
     6     layers.Flatten(),
     7     layers.Dense(64, activation='relu'),
     8     layers.Dense(10, activation='softmax')
     9 ])
```

Conv Layer

Filter/W

Our input image

Anyone know?

Model Development

```
10
11 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
[4]   1 history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

Model Evaluation

Image Classification/recognition

- **Accuracy**
- **Precision**
- **Recall**
- **F1-Score**

Model Evaluation

Image Classification/recognition

Accuracy

Precision

Recall

F1-Score

Confusion Matrix

		Positive	Negative
	Positive	TP	FP
	Negative	FN	TN

A matrix showing the actual vs. predicted classifications, helping in understanding the types of classification errors.

Useful for multi-class classification tasks.

Model Evaluation

Image Classification/recognition

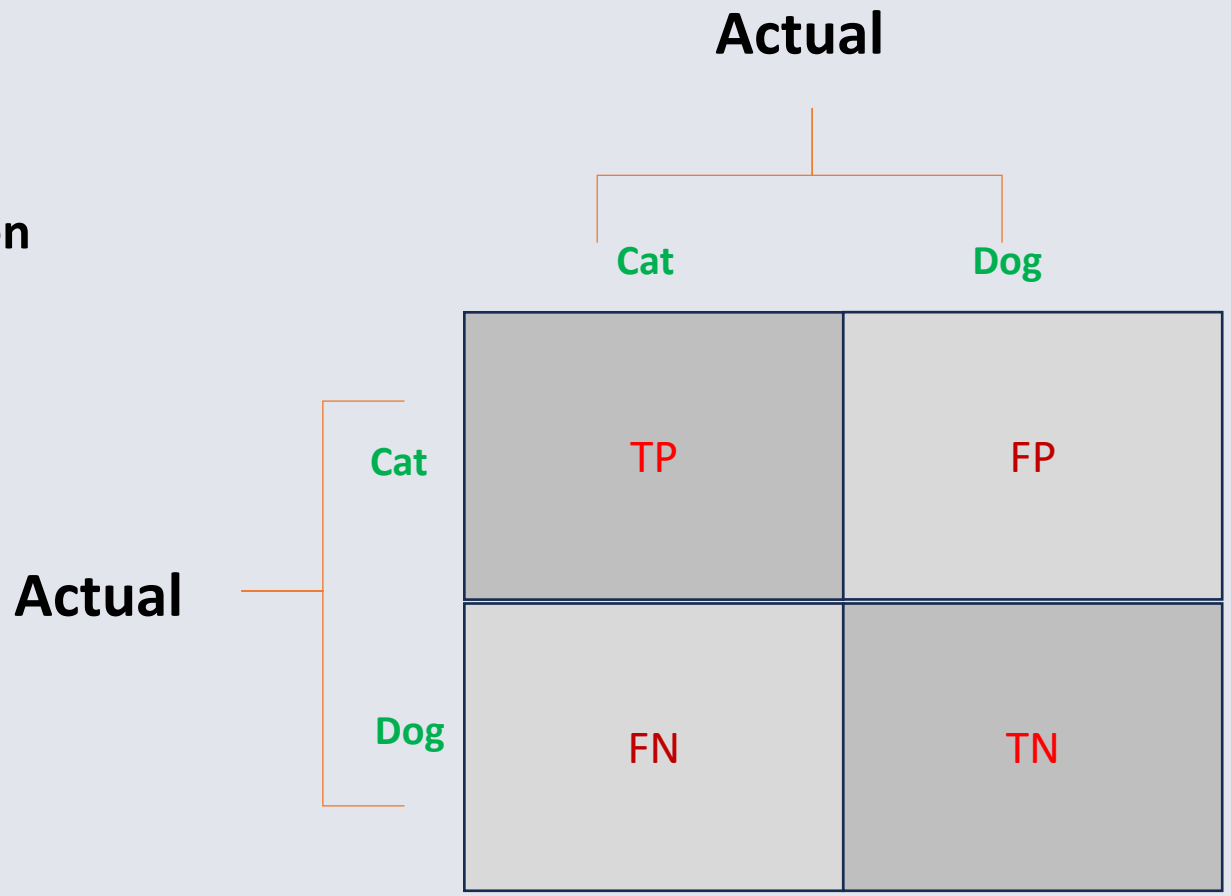
Accuracy

Precision

Recall

F1-Score

Confusion Matrix



A matrix showing the actual vs. predicted classifications, helping in understanding the types of classification errors.

Useful for multi-class classification tasks.

Model Evaluation

Image Classification/recognition

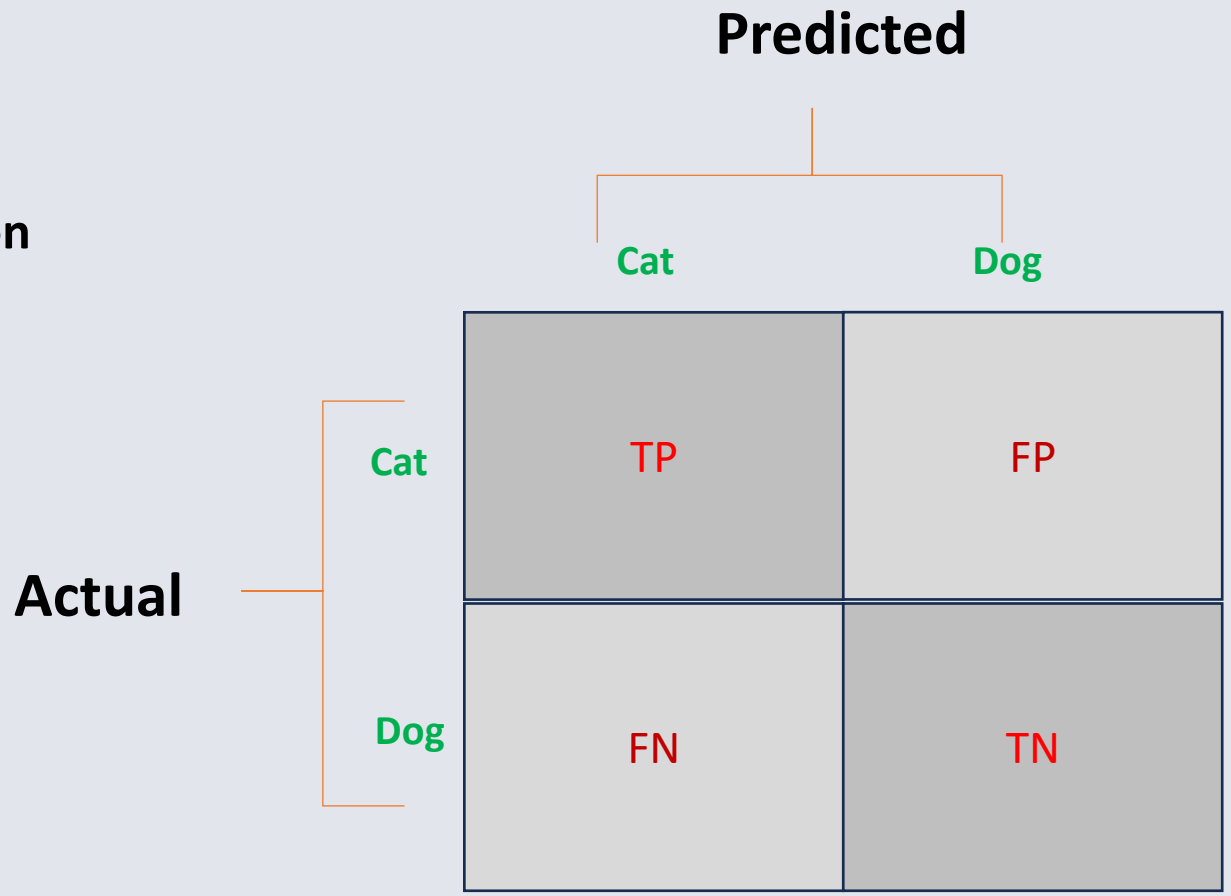
Accuracy

Precision

Recall

F1-Score

Confusion Matrix



A matrix showing the actual vs. predicted classifications, helping in understanding the types of classification errors.

Useful for multi-class classification tasks.

Model Evaluation

Image Classification/recognition

	Cat	Dog
Cat	45	5
Dog	5	45

Testing set: 100 images

(**Cat**: 50, **Dog**: 50)

For instance: Model 1

True Positives (TP) = 45 (The model correctly predicted the positive class)

True Negatives (TN) = 45 (The model correctly predicted the negative class)

False Positives (FP) = 5 (The model incorrectly predicted the positive class)

False Negatives (FN) = 5 (The model incorrectly predicted the negative class)

$$\boldsymbol{Accuracy} = Number\ of\ correct\ predictions/total\ number\ of\ predictions$$

Accuracy = 90/100=0.90 or 90%

Model Evaluation

Image Classification/recognition

	Cat	Dog
Cat	32	18
Dog	38	12

Testing set: 100 images

(**Cat**: 50, **Dog**: 50)

For instance: Model 2

True Positives (TP) = 32 (correctly predicted positive class)

False Positives (FP) = 18 (incorrectly predicted positive class)

False Negatives (FN) = 38 (incorrectly predicted negative class)

True Negatives (TN) = 12 (correctly predicted negative class)

$$\boldsymbol{Accuracy} = Number\ of\ correct\ predictions/total\ number\ of\ predictions$$

Accuracy = 44/100=0.44 or 44%

Model Evaluation

Image Classification/recognition

	Happy	Fear	Neutral	Sad	Angry	Surprise
Happy	30	0	5	0	3	2
Fear	1	7	0	3	2	1
Neutral	2	0	25	3	0	0
Sad	0	3	0	27	0	0
Angry	0	4	0	5	39	2
Surprise	0	0	2	0	0	18

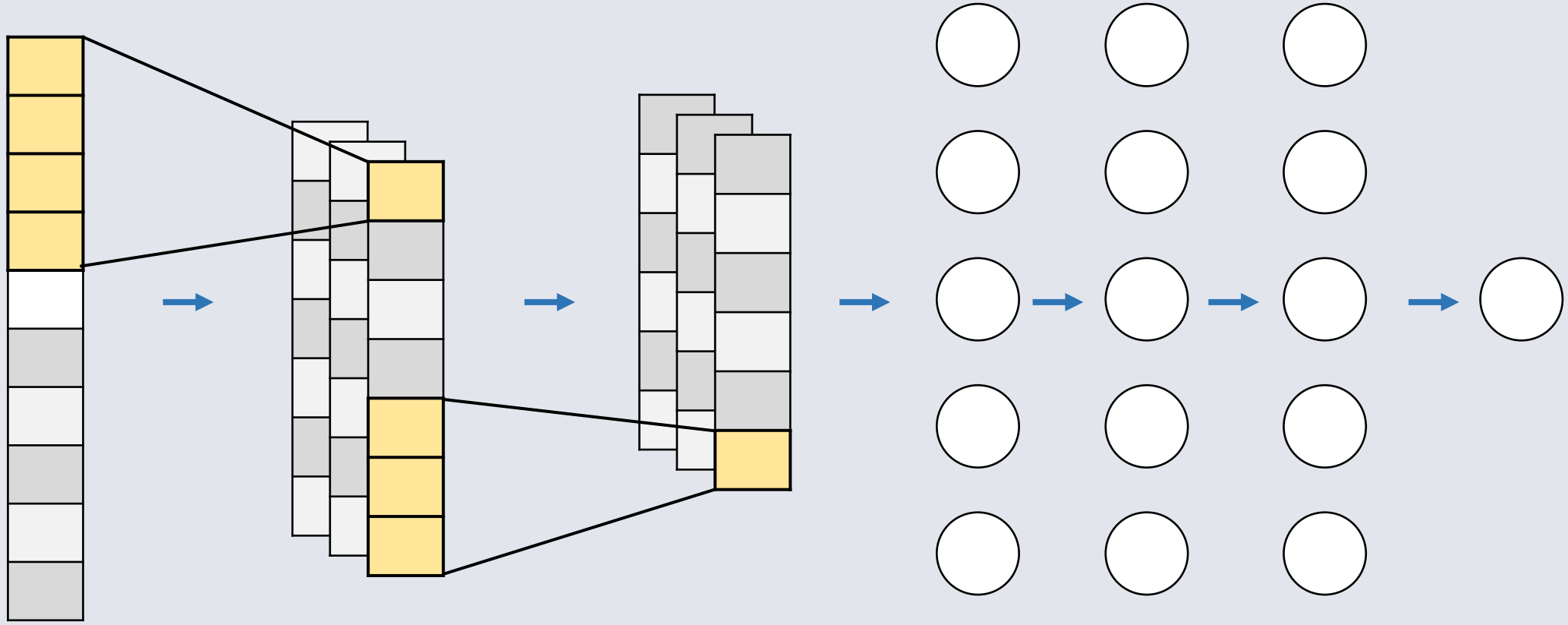
Testing set: 184 images

(Happy: 40, Fear: 14, Neutral: 30, Sad: 30, Angry: 50, Surprise: 20)

1D CNN

Type of CNN specifically designed for sequential data such as time series, text, and audio signals.

It applies **1D convolutions** along a single spatial axis (e.g., time or sequence length), capturing **local patterns and dependencies** in the data.

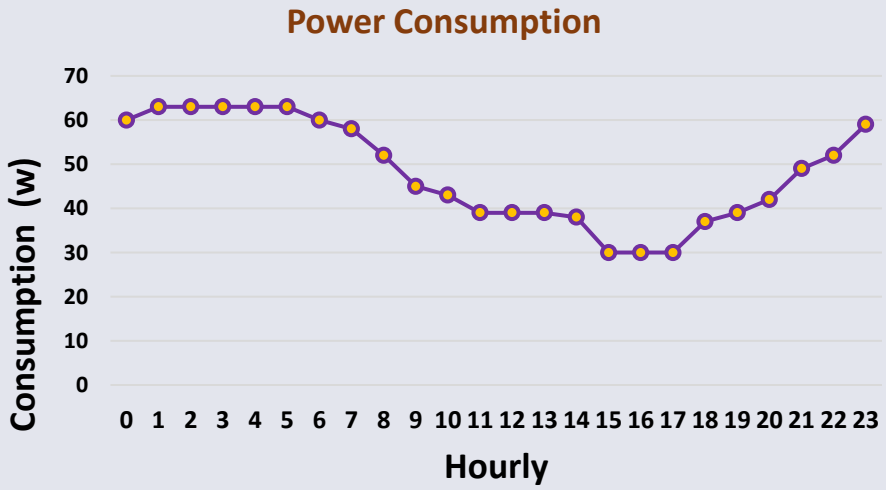


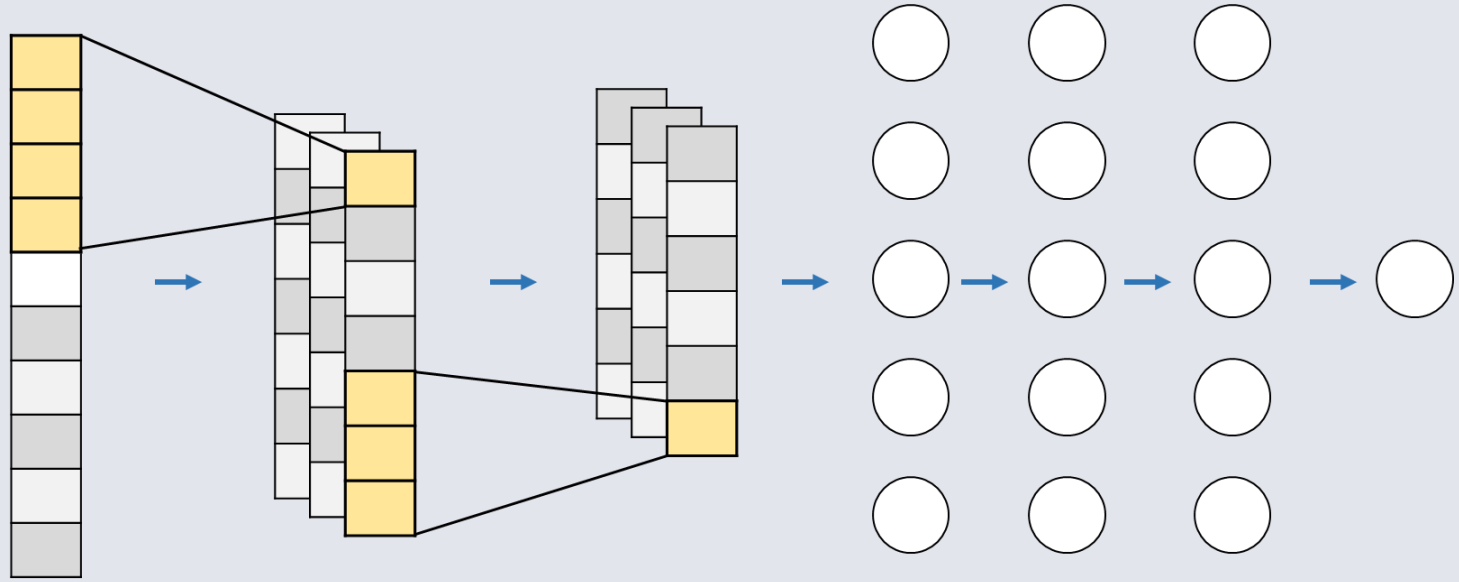
1D CNN

The input data has a single spatial dimension (e.g., time-series data, ECG signals).

Accepts **1D data** in the shape **(sequence_length, channels)**.

Example: A **24-timesstep** time-series signal with **1 feature per timestep** → Input shape = **(24, 1)**.



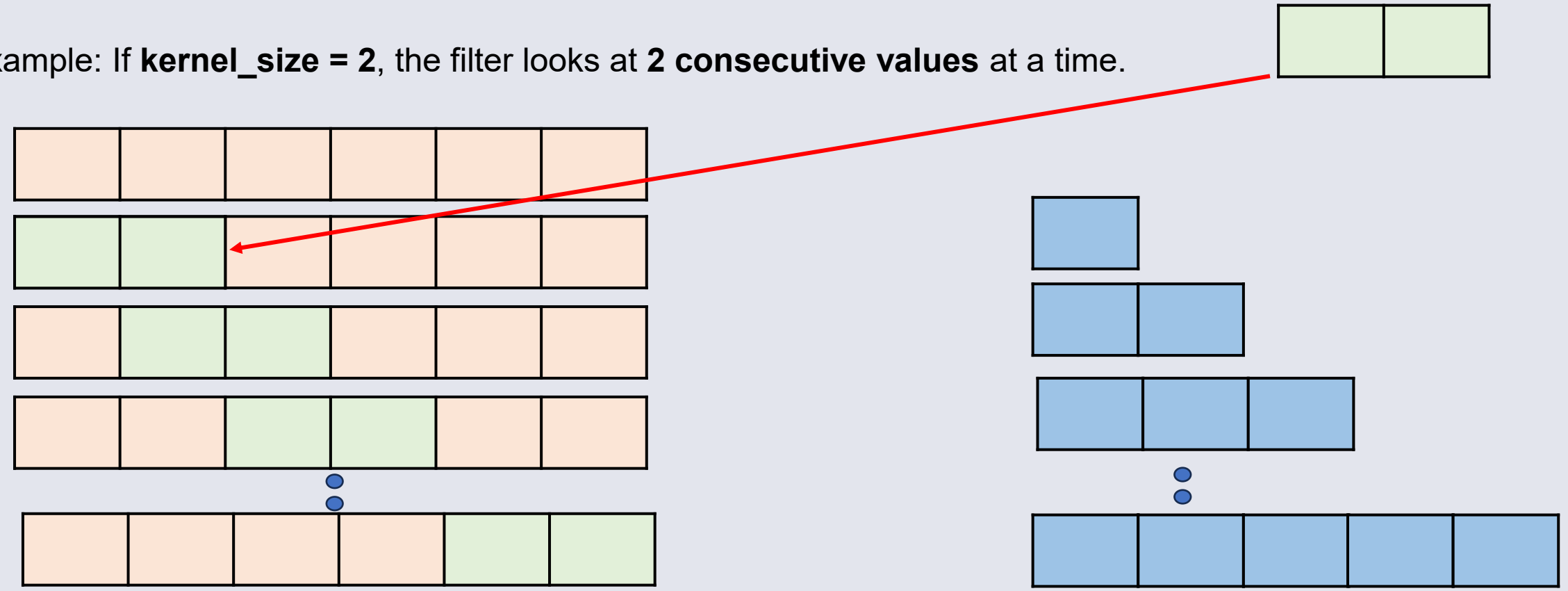


1D CNN

Applies a **1D filter (kernel)** that moves along the sequence.

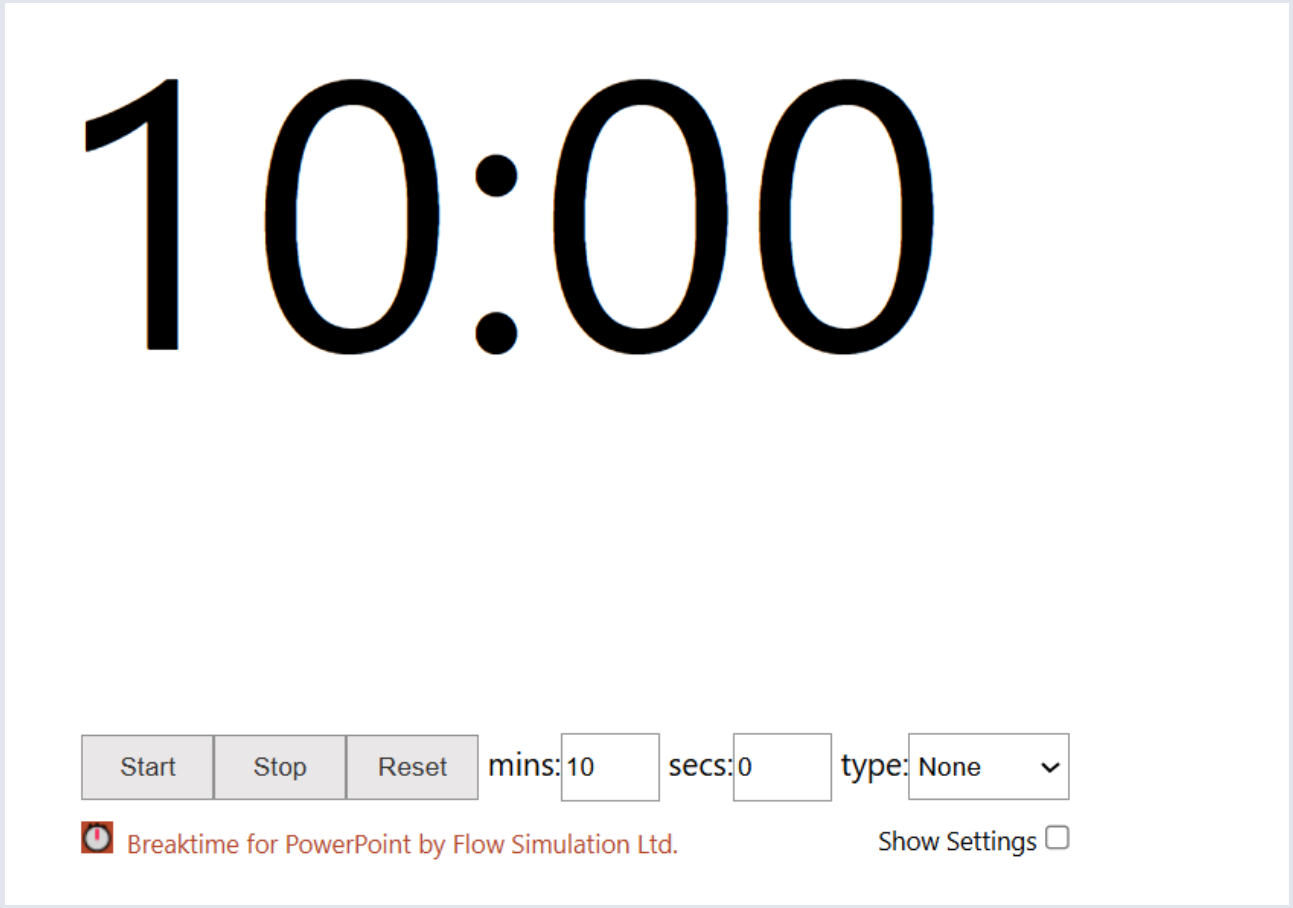
Detects **local features** (e.g., frequency patterns in audio, word embeddings in NLP).

Example: If **kernel_size = 2**, the filter looks at **2 consecutive values** at a time.



University of Lancashire

Warm-up Quizzes Activity



Next week

CNN Architectures

AlexNet, ResNet, ….

Any Question?