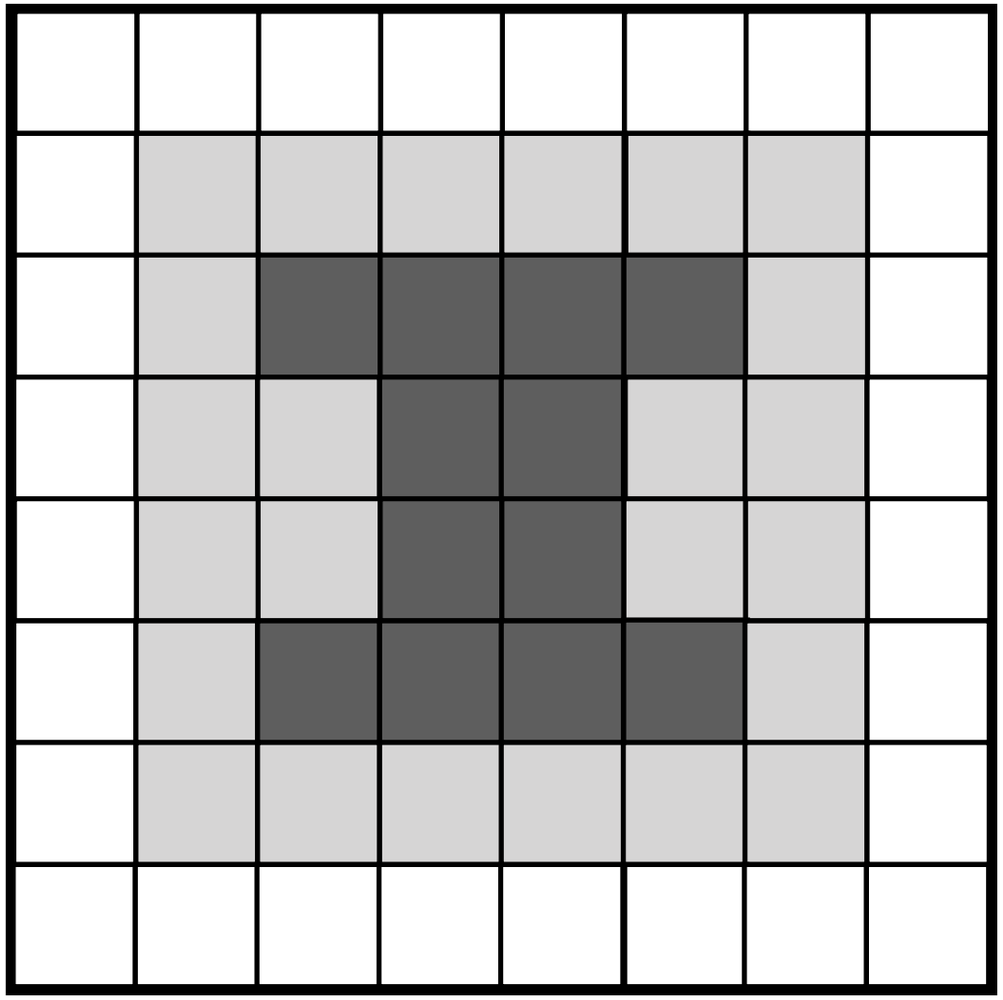
# PRACTICAL MACHINE LEARNING: CLASSIFICATION & CONVOLUTIONAL NEURAL NETWORKS

Dr. Alexander Booth (he / him)
Universidad Católica del Norte, Chile
October 10th, 2024

Queen Mary
University of London

# Fundamentals of Images

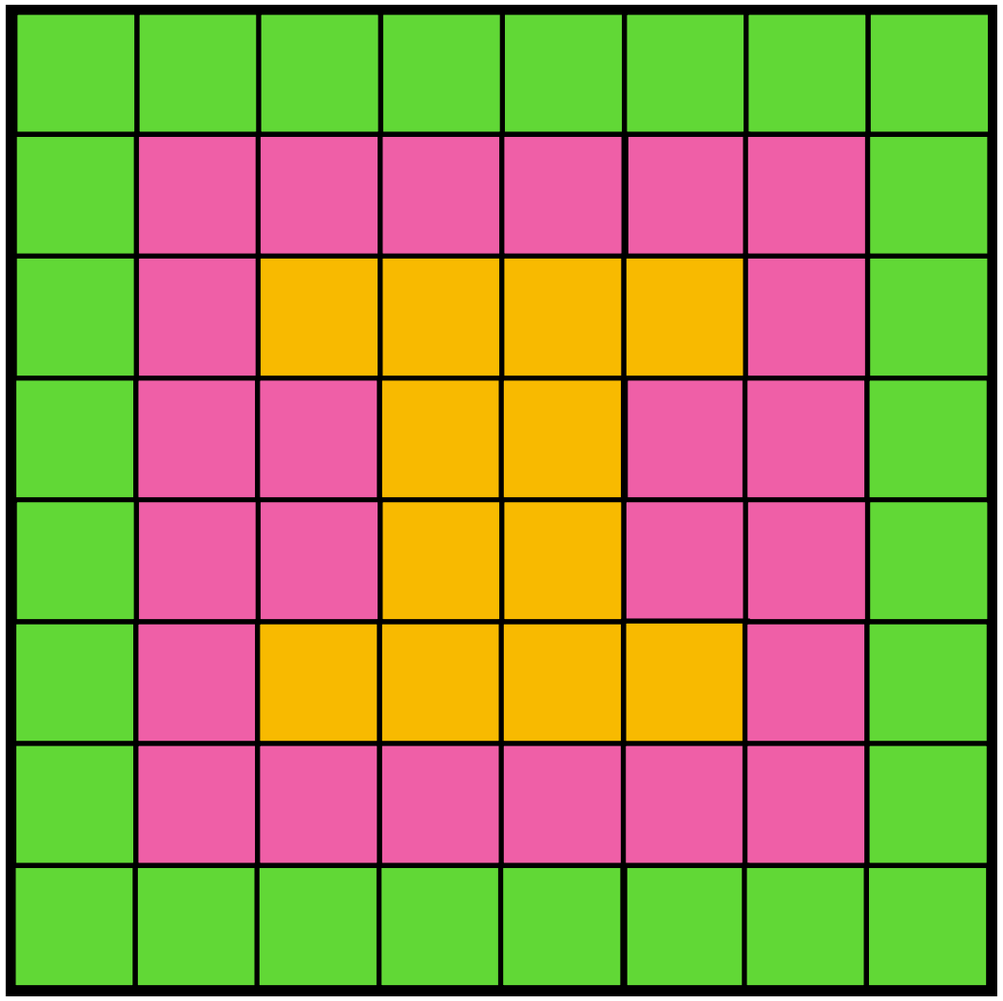


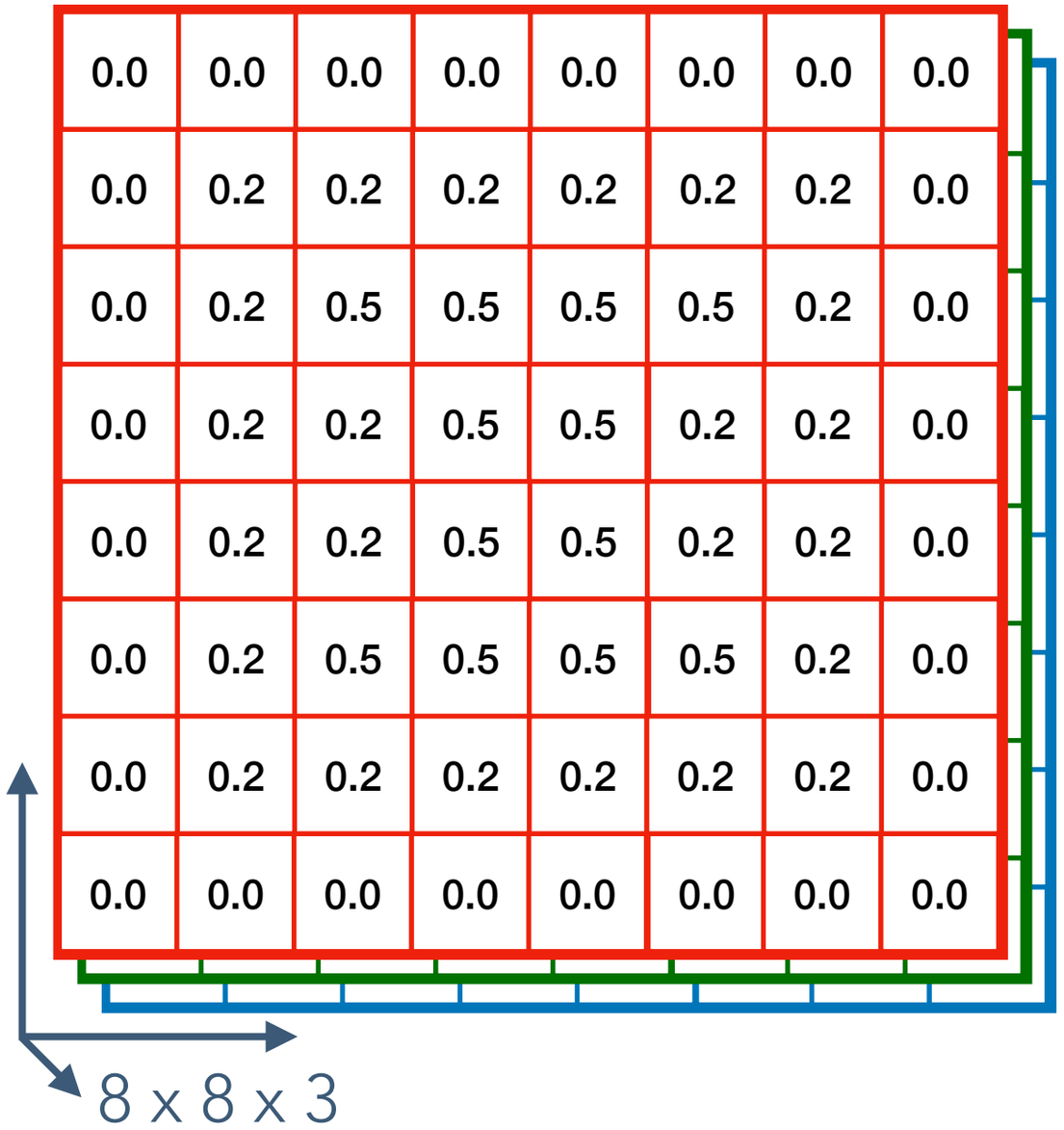
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.5 | 0.5 | 0.5 | 0.5 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.5 | 0.5 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.5 | 0.5 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.5 | 0.5 | 0.5 | 0.5 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

8 x 8

- Images are **grids of data**.
- Each cell in the grid is a **pixel**.
- The intensity of each pixel in a **greyscale** image can be quantified by a **single number** (8 x 8 image -> quantified by 64 numbers).

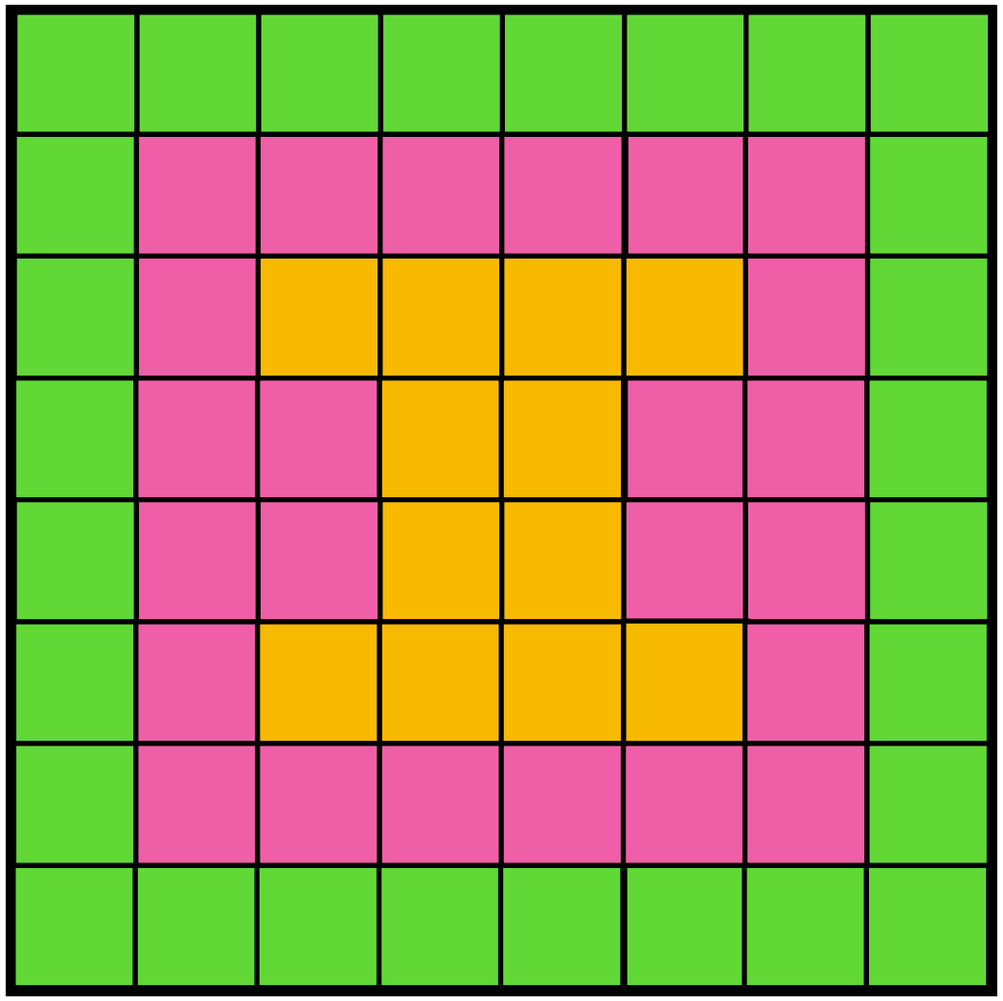
# Fundamentals of Images

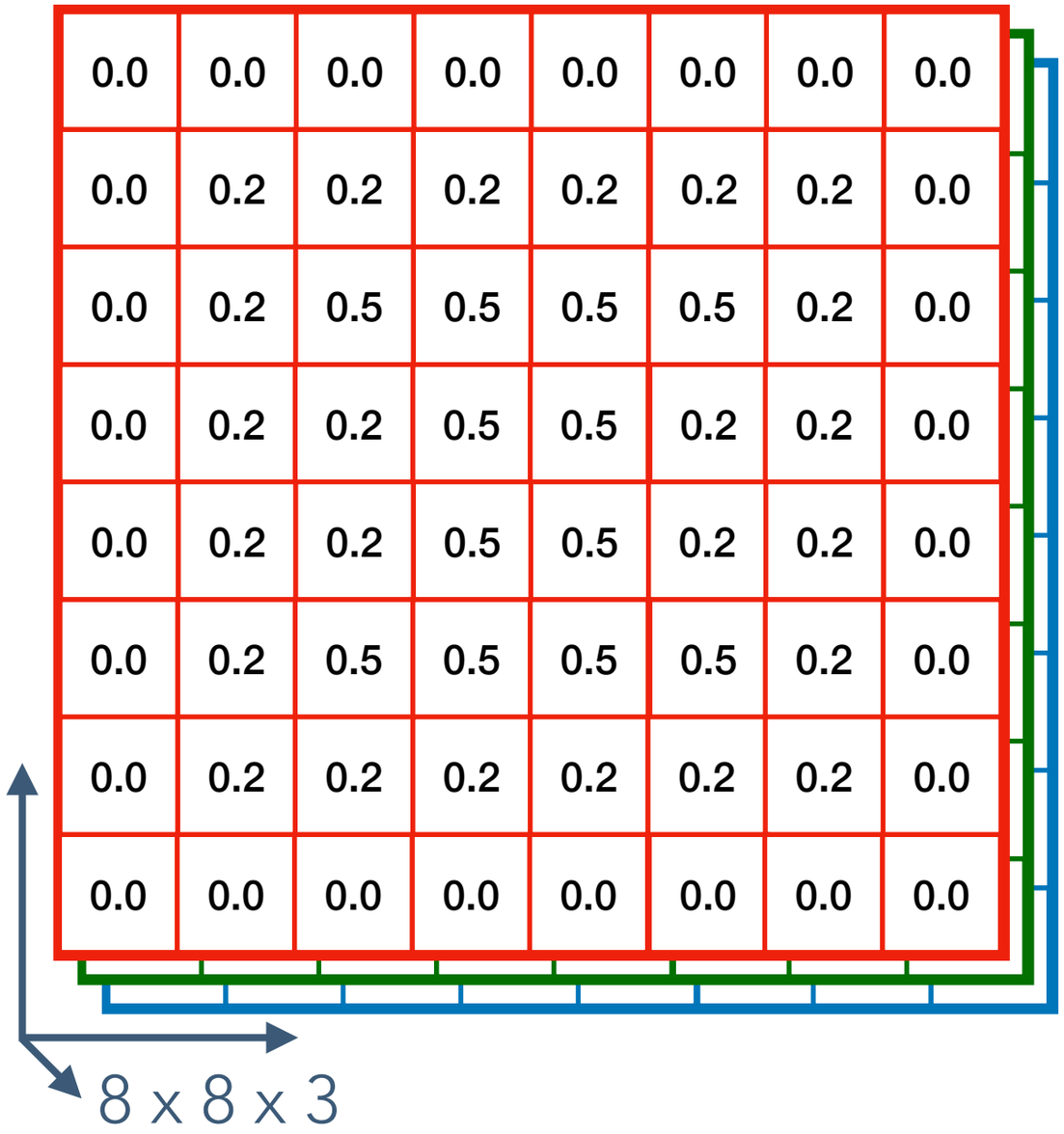





- Images are **grids of data**.
- Each cell in the grid is a **pixel**.
- The intensity of each pixel in a **colour** image can be quantified by **three numbers, RGB** (8 x 8 x 3 image -> quantified by 192 numbers).

# Computer Vision






- Neural networks can be used to tackle many image related tasks -> "computer vision":
  - Classification.
  - Object detection.
  - Semantic segmentation.
  - Instance segmentation.

# Computer Vision

Examples & images taken from [1]


Categorising **entire** images into user-defined classes.

- Neural networks can be used to tackle many image related tasks -> "computer vision":
  - **Classification**.
  - Object detection.
  - Semantic segmentation.
  - Instance segmentation.

# Computer Vision

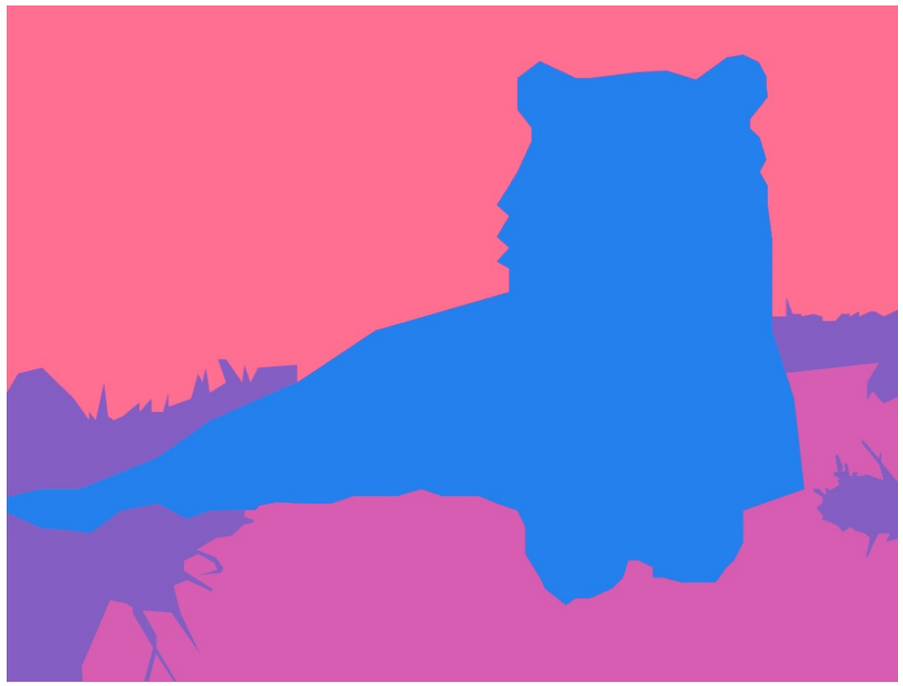
Examples & images taken from [1]


Identifying specific objects within an image, can be multiple objects (usually with bounding boxes).

- Neural networks can be used to tackle many image related tasks -> "computer vision":
  - Classification.
  - **Object detection**.
  - Semantic segmentation.
  - Instance segmentation.

# Computer Vision

Examples & images taken from [1]



Identifying each pixel in an image for more detailed classification.

- Neural networks can be used to tackle many image related tasks -> "computer vision":
  - Classification.
  - Object detection.
  - **Semantic segmentation**.
  - Instance segmentation.

# Computer Vision

Examples & images taken from [1]


Recognise different instances of the same class.

- Neural networks can be used to tackle many image related tasks -> "computer vision":
  - Classification.
  - Object detection.
  - Semantic segmentation.
  - **Instance segmentation**.

# Recap: Basic Structure of a Neural Network

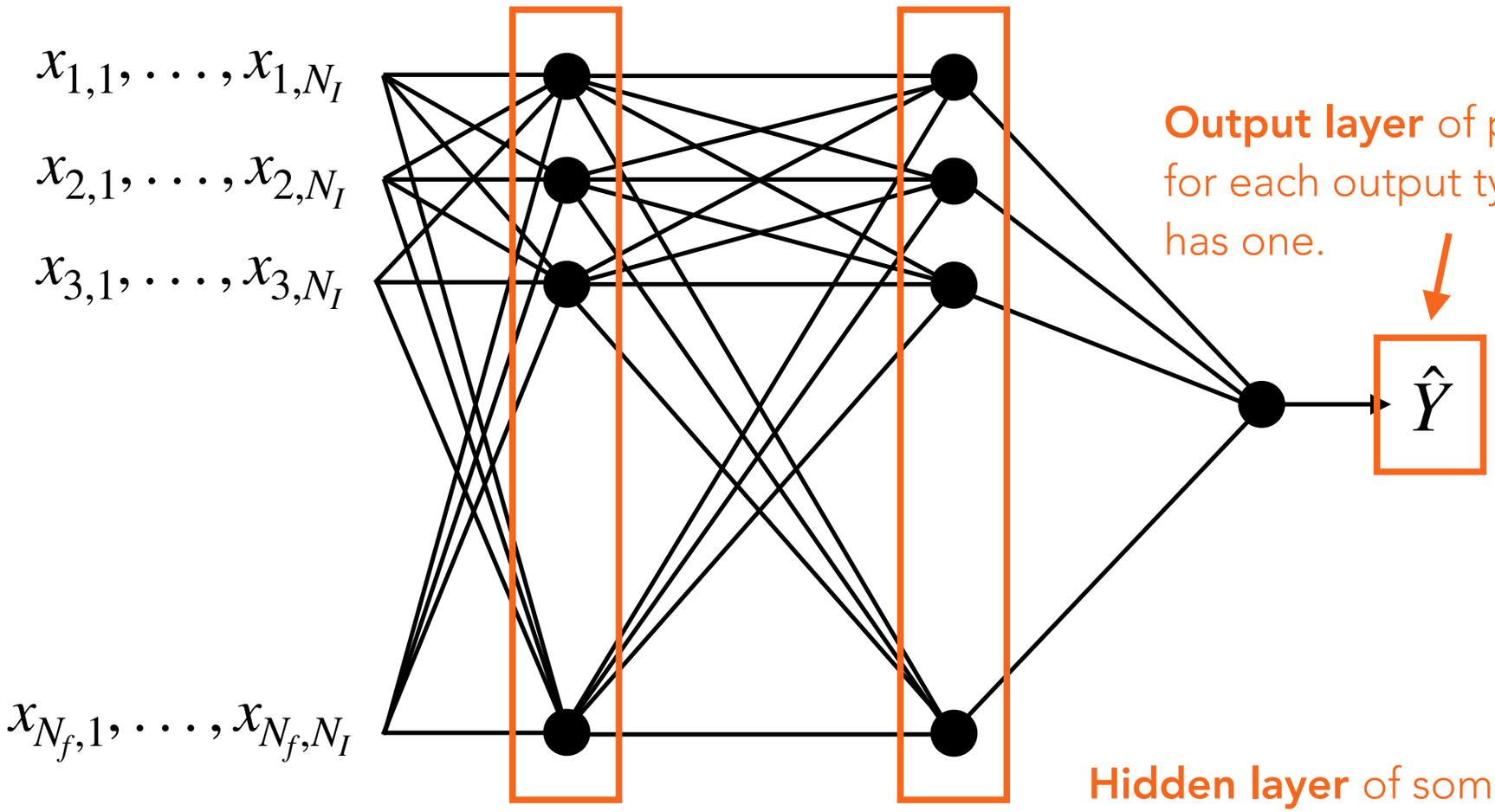
Initial features →

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,N_I} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,N_I} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N_f,1} & x_{N_f,2} & x_{N_f,3} & \cdots & x_{N_f,N_I} \end{bmatrix}$$

↓ Instances

Target feature (to predict)

$$Y = [y_{I_1}, y_{I_2}, \ldots, y_{I_{N_I}}]$$




**Output layer** of perceptrons, one for each output type. This network has one.

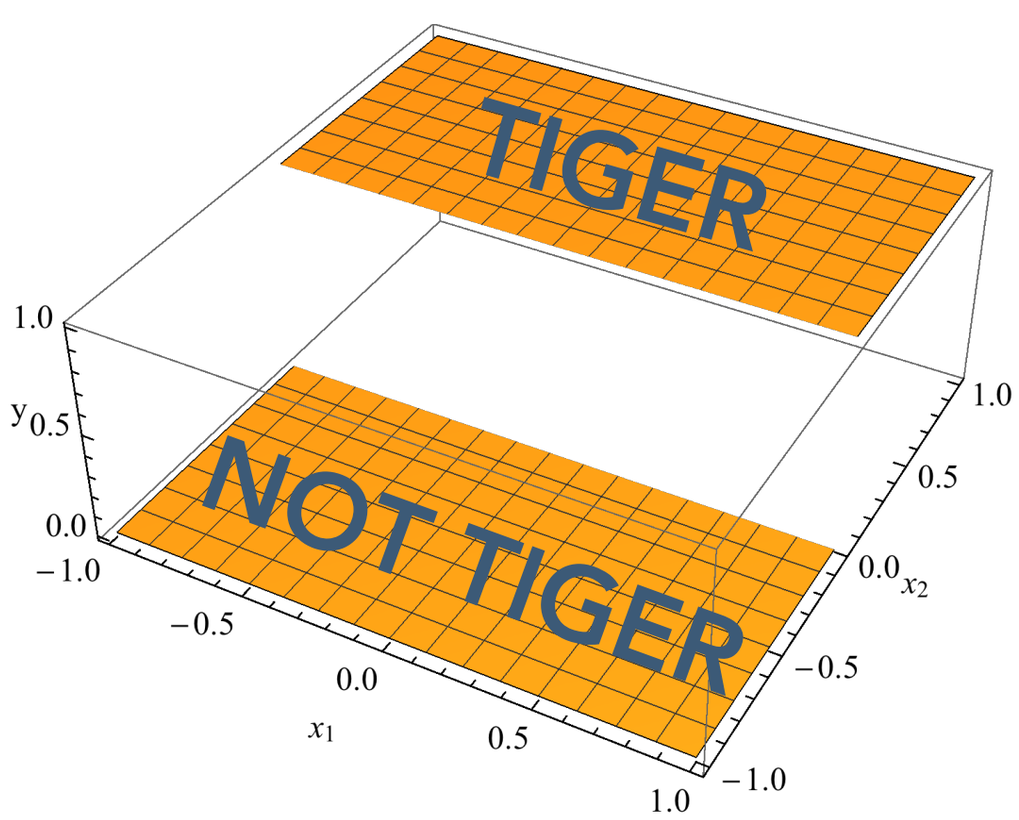
**Input layer** of $N_f$ perceptrons (one for each dimension of input *feature* space).

**Hidden layer** of some number of perceptrons, $N_{H_1}$; at least one for each dimension of the input feature space.

# Recap: Basic Structure of a Neural Network

- The final perceptron can be used to assign a type to data instance -> "classify" it.

- For example, consider a final perceptron with a binary activation function** that gives an all or nothing response with data that contains **two classes** of event (tiger and not tiger).
  - Let "all" response correspond to one type: tiger.
  - Let "nothing" response correspond to the other: not tiger




**Slightly more subtle -> usually use sigmoid.

# Multi-class Output

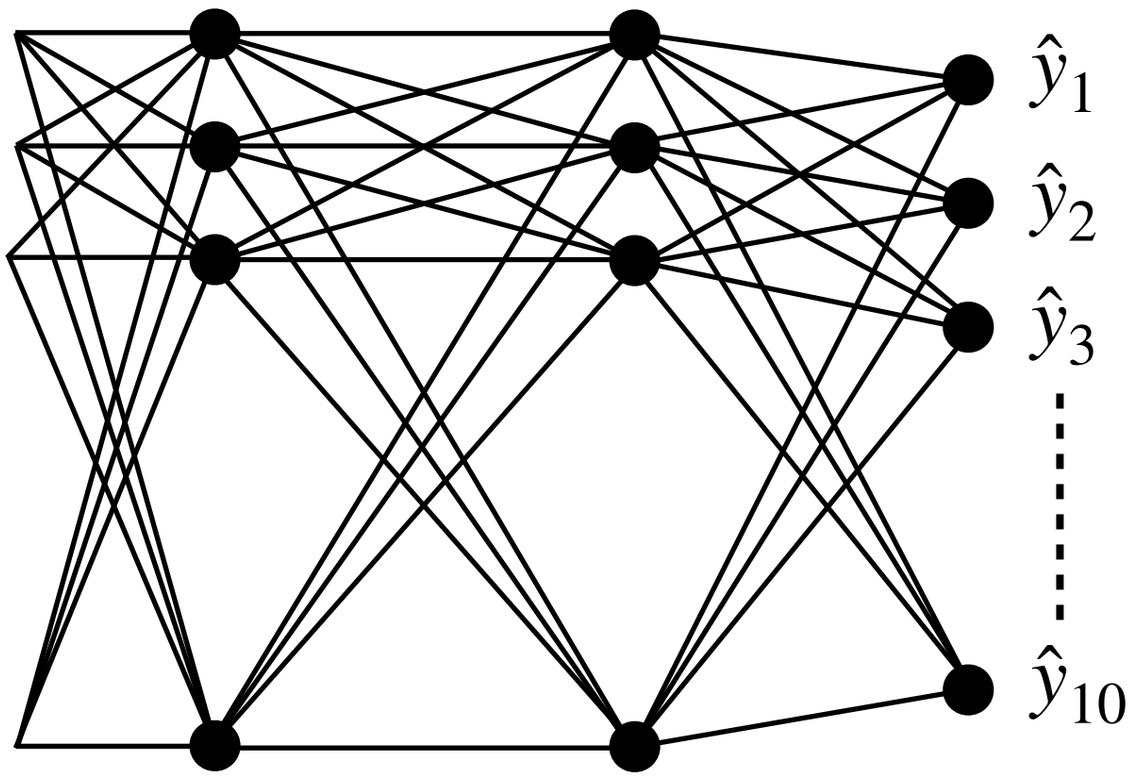
- Often have more than two possible classes - require a network with **an output layer** with **number of nodes equal to the number of possible classes**.
- For example: classifying the **MNIST dataset** [2]:
  - Collection of labelled images each containing a single handwritten digit between 0 and 9.
  - 70000 instances.
  - 28 x 28 greyscale pixels.
  - Pixel intensity in range [0, 255].

[3]

# Multi-class Output

- Often have more than two possible classes - require a network with **an output layer** with **number of nodes equal to the number of possible classes**.
- For example: classifying the **MNIST dataset** [2]:
  - Collection of labelled images each containing a single handwritten digit between 0 and 9.
  - 70000 instances.
  - 28 x 28 greyscale pixels.
  - Pixel intensity in range [0, 255].

10 classes, 10 nodes in output layer

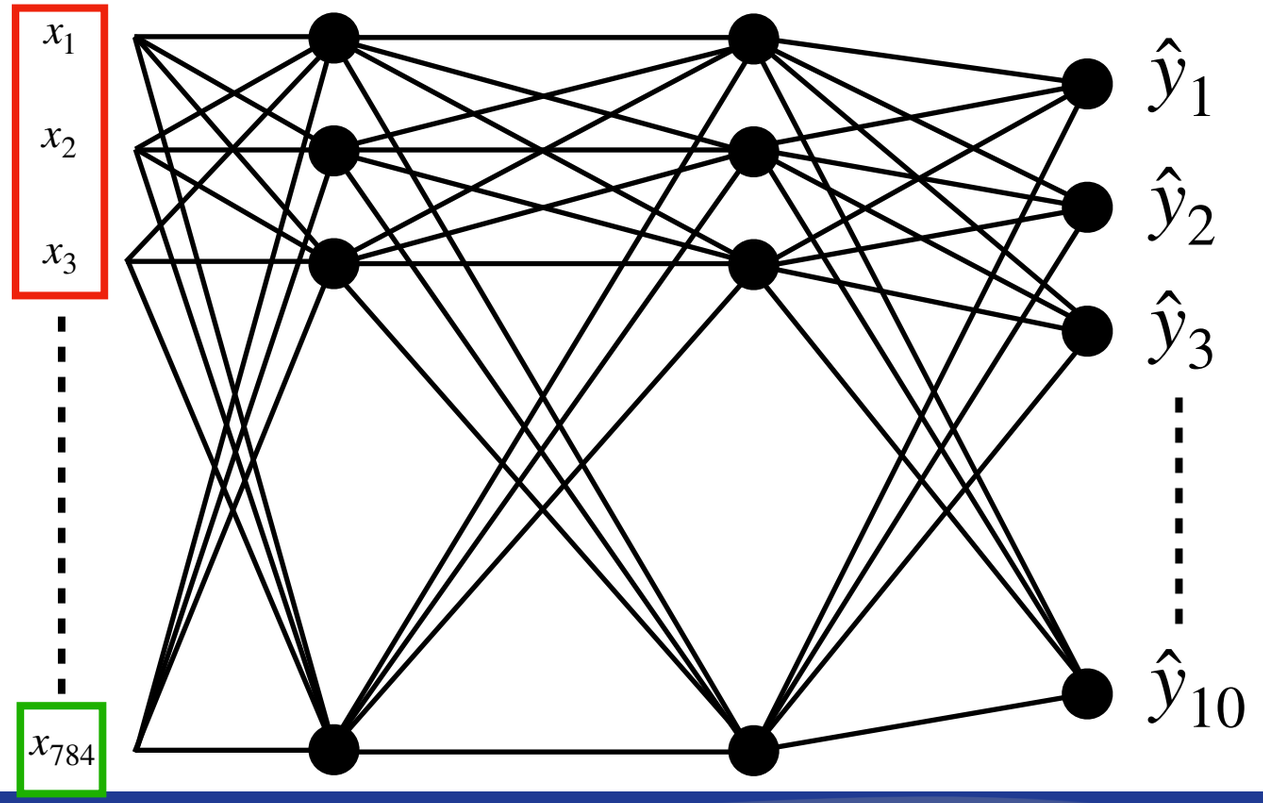


# Multi-class Output

- Often have more than two possible classes - require a network with **an output layer** with **number of nodes equal to the number of possible classes**.
- For example: classifying the **MNIST dataset** [2]:
  - Collection of labelled images each containing a single handwritten digit between 0 and 9.
  - 70000 instances.
  - 28 x 28 greyscale pixels.
  - Pixel intensity in range [0, 255].

784 input features, 784 nodes in input layer

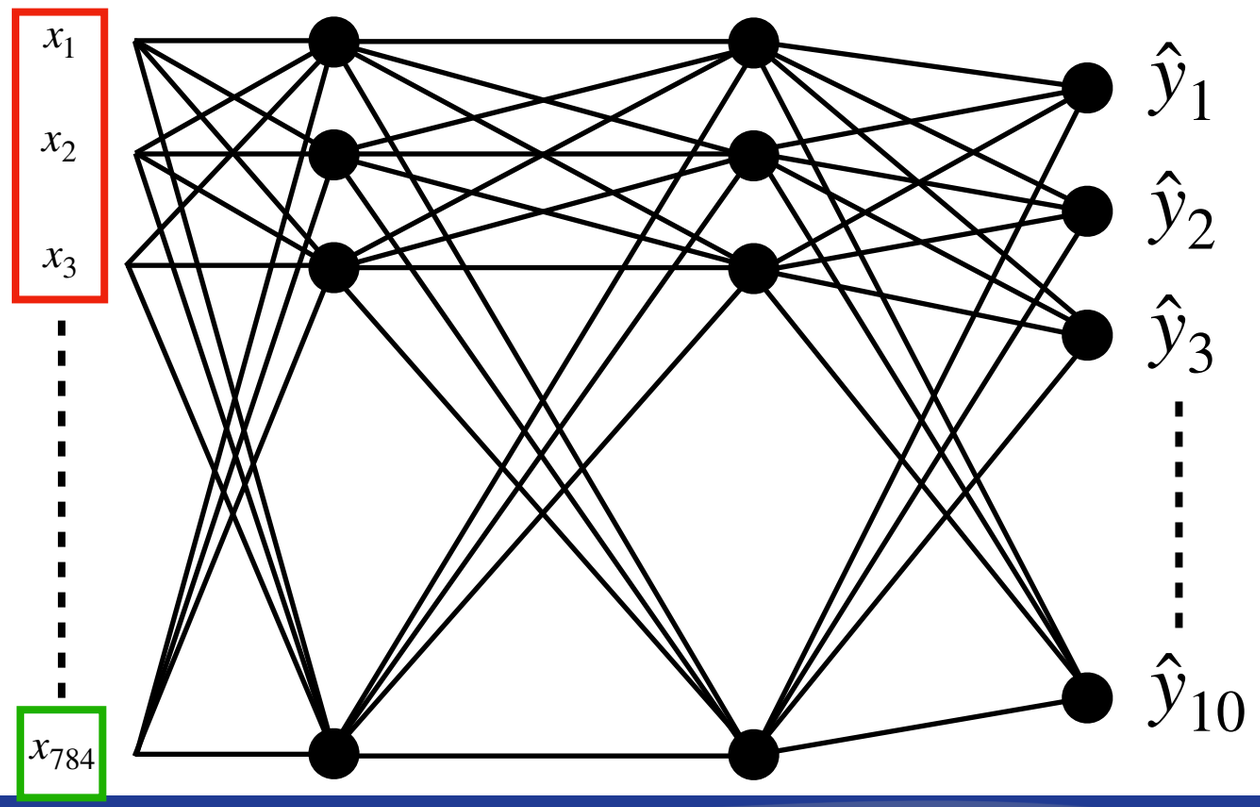
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | … | … | … | … | $x_{27}$ | $x_{28}$ |
| $x_{29}$ | … | … | … | … | … | … | $x_{56}$ |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| $x_{729}$ | … | … | … | … | … | … | $x_{756}$ |
| $x_{757}$ | $x_{758}$ | … | … | … | … | $x_{783}$ | $x_{784}$ |



# Multi-class Output

- From each output node:
  - $\hat{y}_1$ makes the decision if an instance is the number 1 vs is not the number 1.
  - $\hat{y}_2$ makes the decision if an instance is the number 2 vs is not the number 2.
  - etc…
- Called **one vs. all** model.
  - In this approach each binary set of outcomes is determined independently of all other sets.
  - E.g. determining outcome of 1 vs. not 1 without considering 2 vs. not 2.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | … | … | … | … | $x_{27}$ | $x_{28}$ |
| $x_{29}$ | … | … | … | … | … | … | $x_{56}$ |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … |
| $x_{729}$ | … | … | … | … | … | … | $x_{756}$ |
| $x_{757}$ | $x_{758}$ | … | … | … | … | $x_{783}$ | $x_{784}$ |

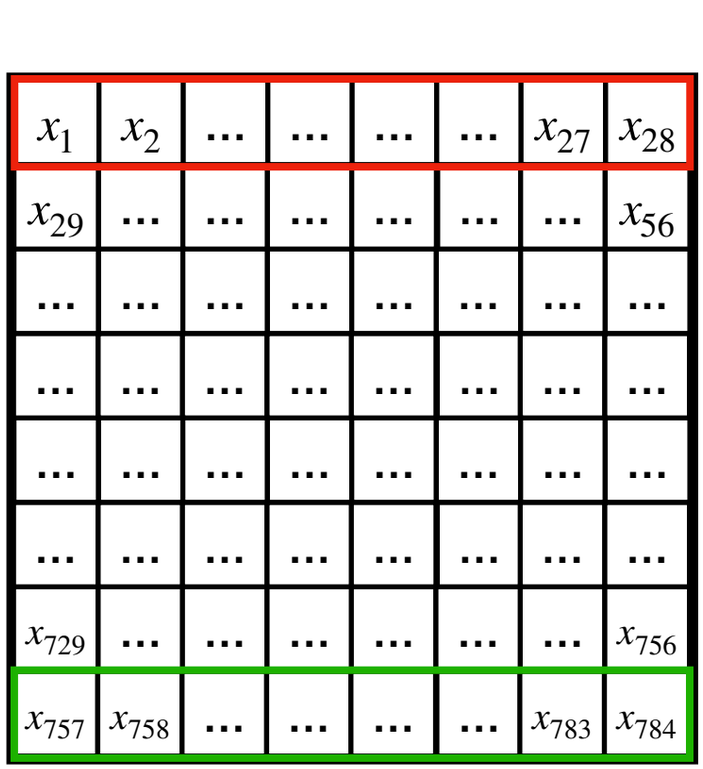


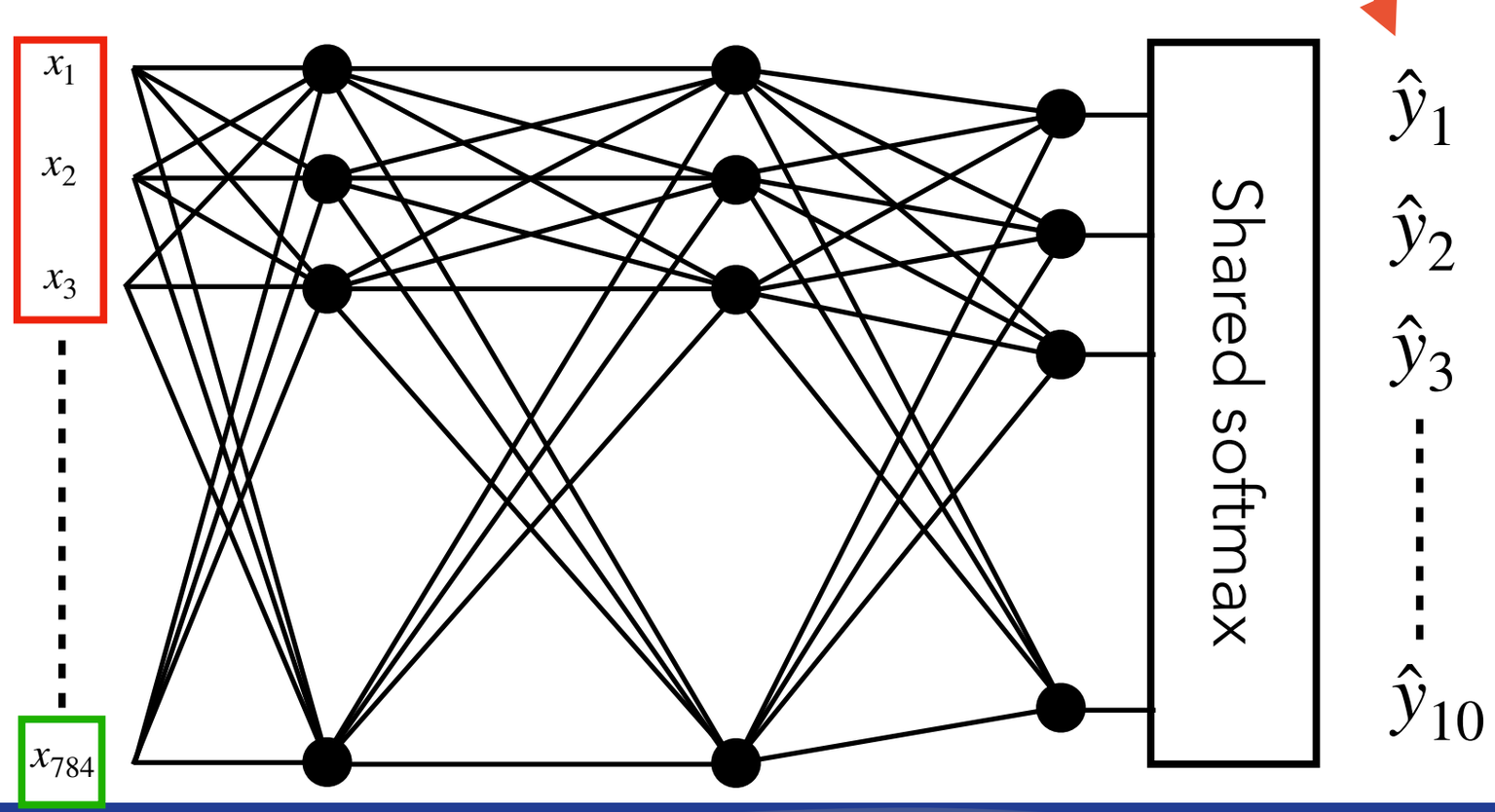
# Multi-class Output

- Usually want to predict the probabilities of each outcome relative to each other -> **one vs. one** model.
  - In this approach after the output layer we can apply the **softmax function** over all outputs with respect to each individual output.
  - Transforms / renormalises the output
  - If $j$ labels the class and there are $N_C$ classes:

$$\hat{y}_j(x) = \frac{e^{W_j^T x}}{\sum_k^{N_C} e^{w_k^T x}}$$

$$\sum_j^{N_C} \hat{y}_j = 1$$



# Performance on MNIST

- The MNIST website [2] lists a number of complicated ways to train neural networks to solve this classification problem.

- MLPs have produced good results! Best error rate achieved of 0.35% (Ciresan et al [4]).

| ID | architecture (number of neurons in each layer) | test error for best validation [%] | best test error [%] | simulation time [h] | weights [milions] |
|---|---|---|---|---|---|
| 1 | 1000, 500, 10 | **0.49** | 0.44 | 23.4 | 1.34 |
| 2 | 1500, 1000, 500, 10 | **0.46** | 0.40 | 44.2 | 3.26 |
| 3 | 2000, 1500, 1000, 500, 10 | **0.41** | 0.39 | 66.7 | 6.69 |
| 4 | 2500, 2000, 1500, 1000, 500, 10 | **0.35** | 0.32 | 114.5 | 12.11 |

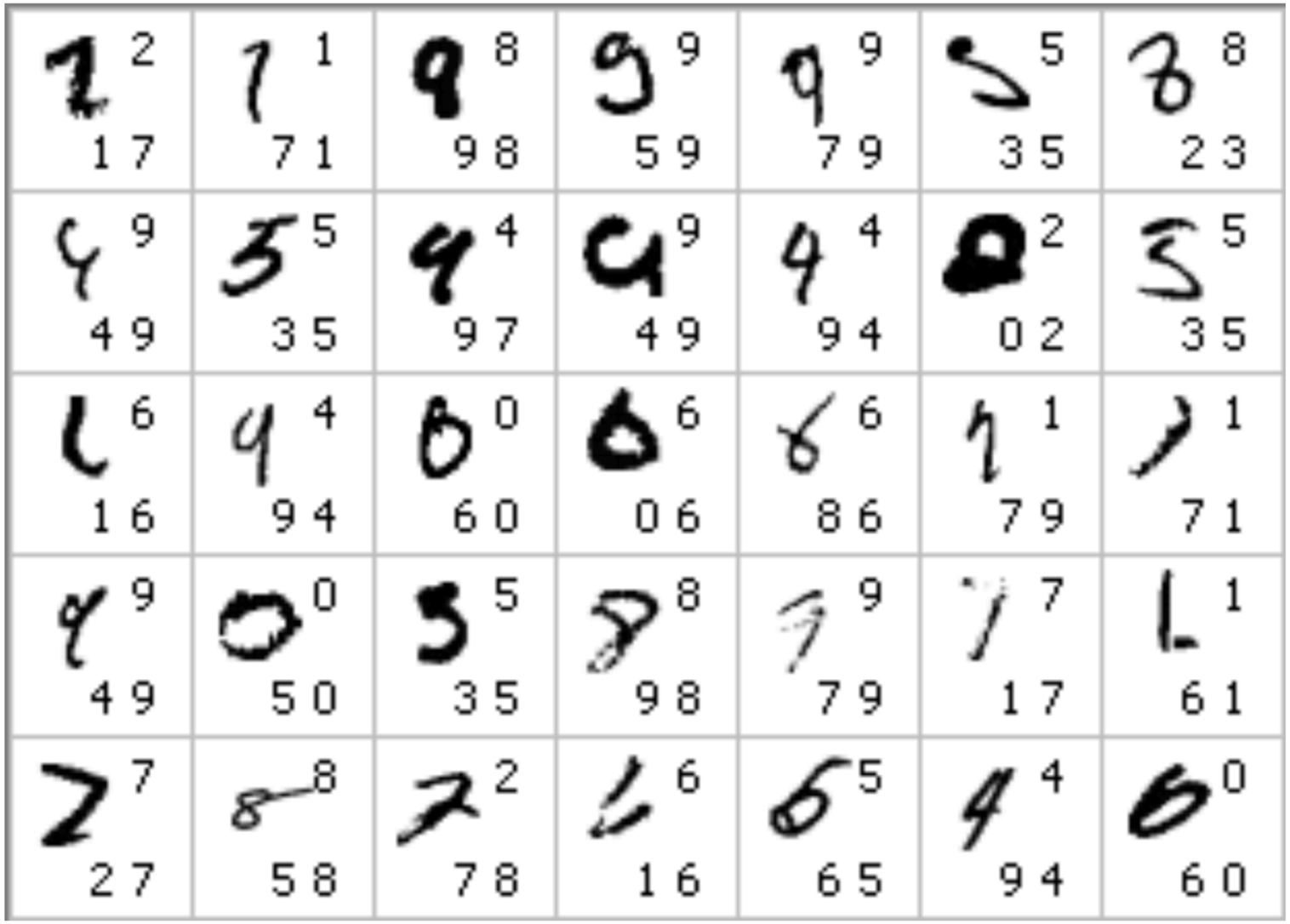
# Performance on MNIST

**35 instances out of 10000 in the test set!**

Correct label

**NUMBER**

Two most likely predictions (L, R)



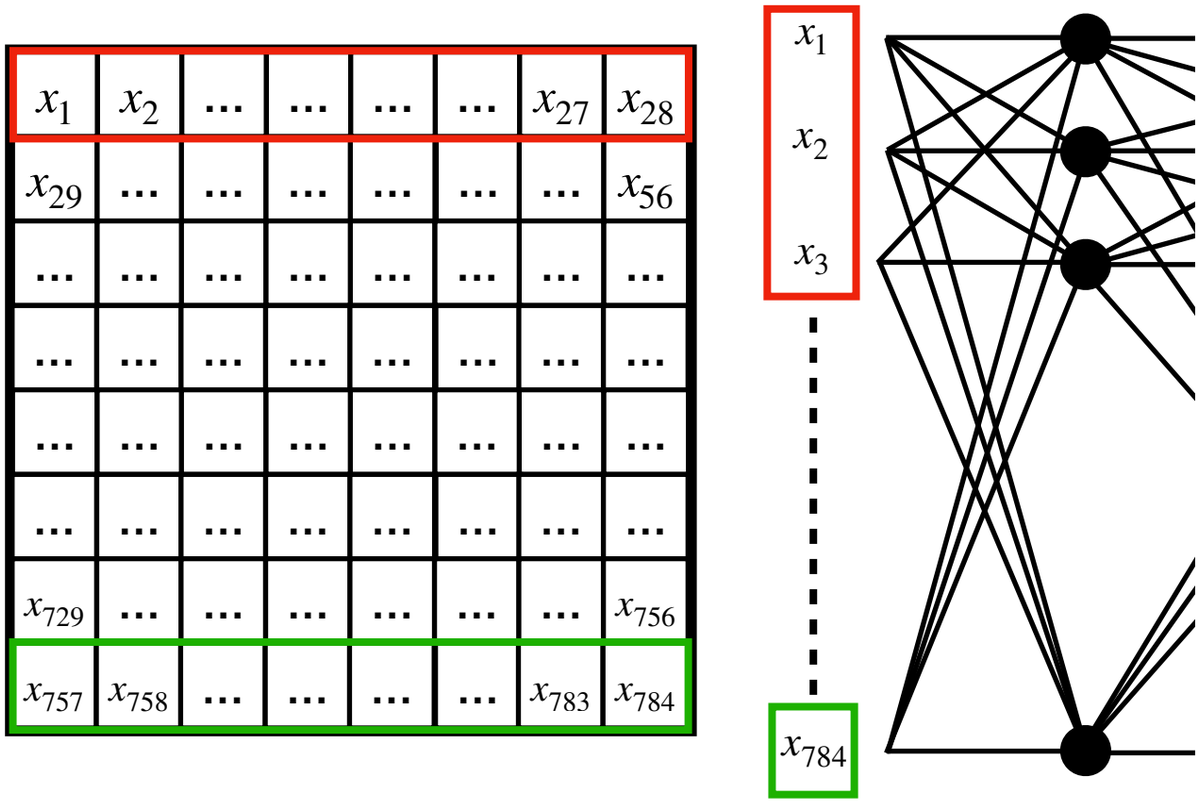
# Convolutional Neural Networks (CNNs)

# CNNs vs. MLPs

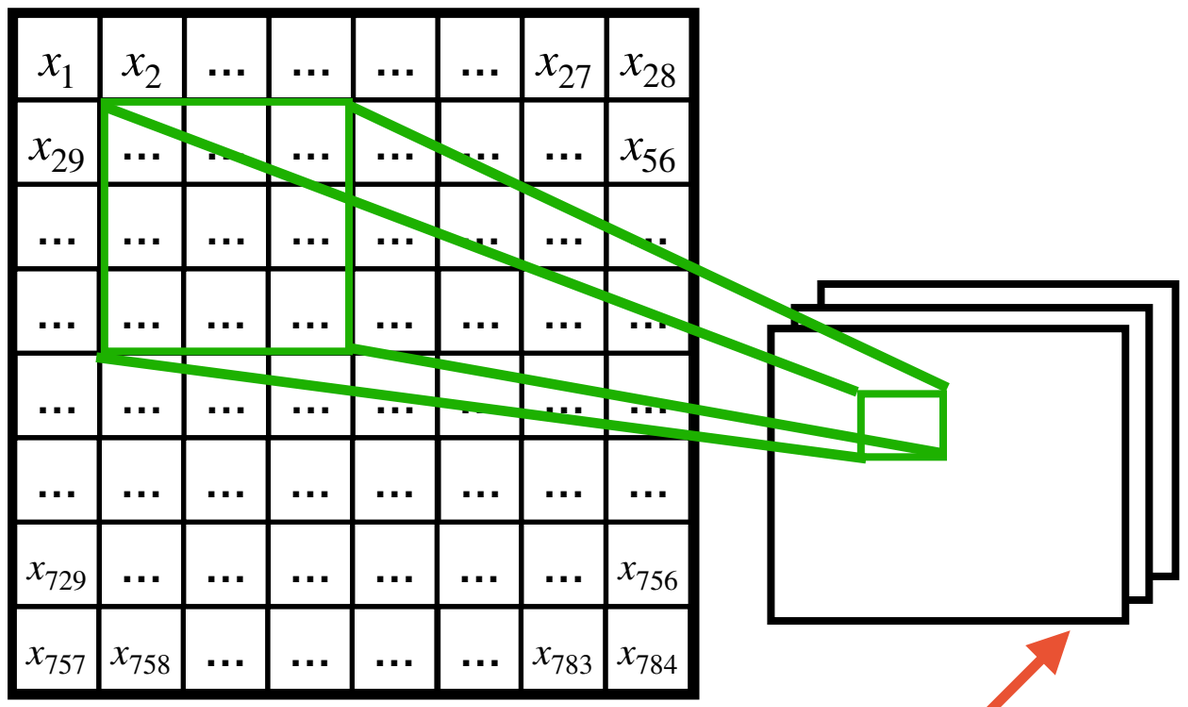
- Like MLPs, CNNs can be applied to problems in **computer vision**.
  - Particularly good at dealing with **grid-like data** (images).

- Unlike MLPs they can take advantage of **spacial correlations** in the input feature space.

MLP




CNN



# Convolution Layers

- In the convolutional layers we **perform convolutions** of the whole input image **piece by piece** by sliding **a filters** over the image.

- Elements of the **filters** are the **weights** in this network -> the thing that you optimise.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.5 | 0.5 | 0.5 | 0.5 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.5 | 0.5 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.5 | 0.5 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.5 | 0.5 | 0.5 | 0.5 | 0.2 | 0.0 |
| 0.0 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

8 x 8 Can be $M \times M$

| | | |
|---|---|---|
| $w_{11}$ | $w_{21}$ | $w_{31}$ |
| $w_{12}$ | $w_{22}$ | $w_{32}$ |
| $w_{13}$ | $w_{23}$ | $w_{33}$ |

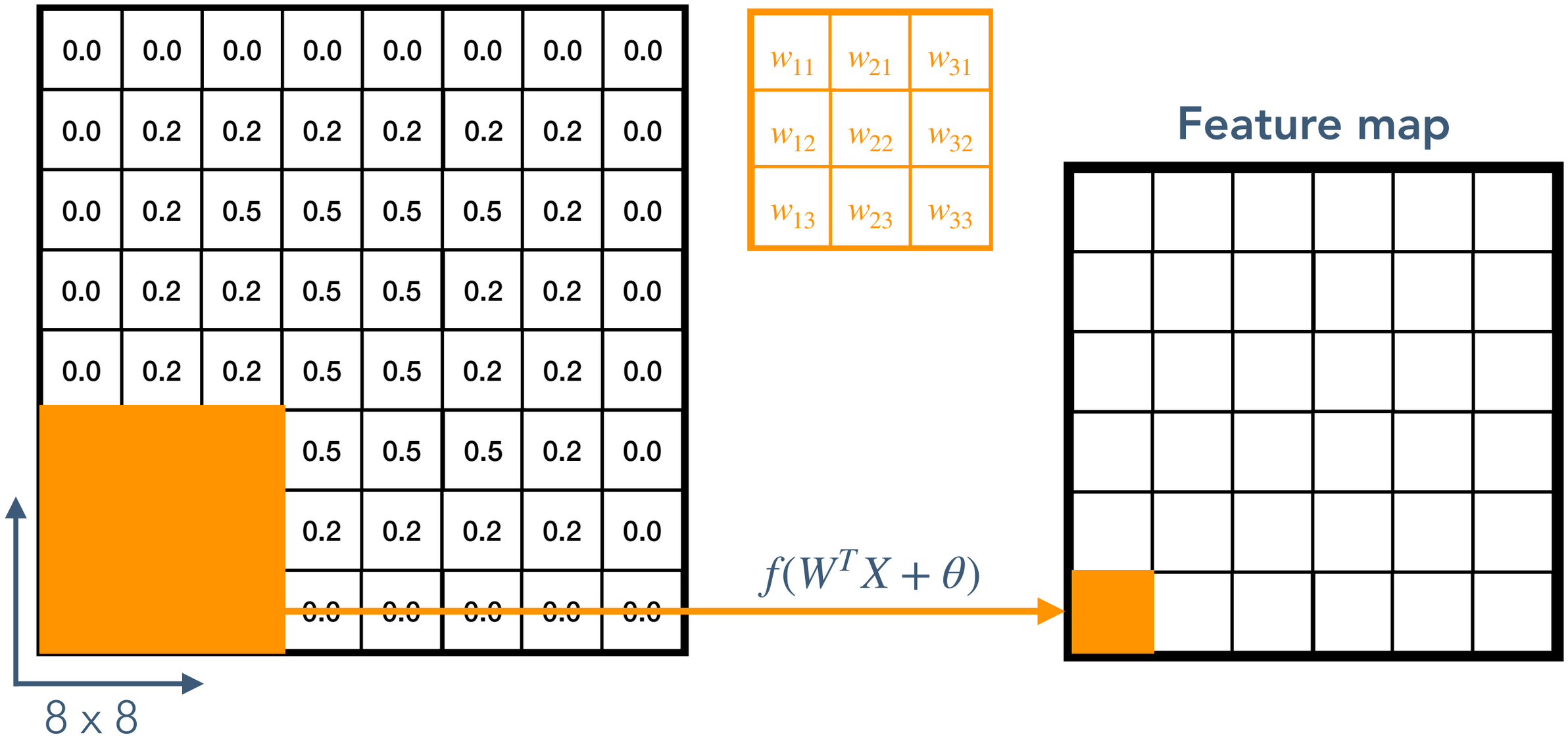
**3 x 3 filter**

Can be $N \times N$ for $N \leq M$

# Convolution Layers

- In the convolutional layers we **perform convolutions** of the whole input image **piece by piece** by sliding **a filters** over the image.

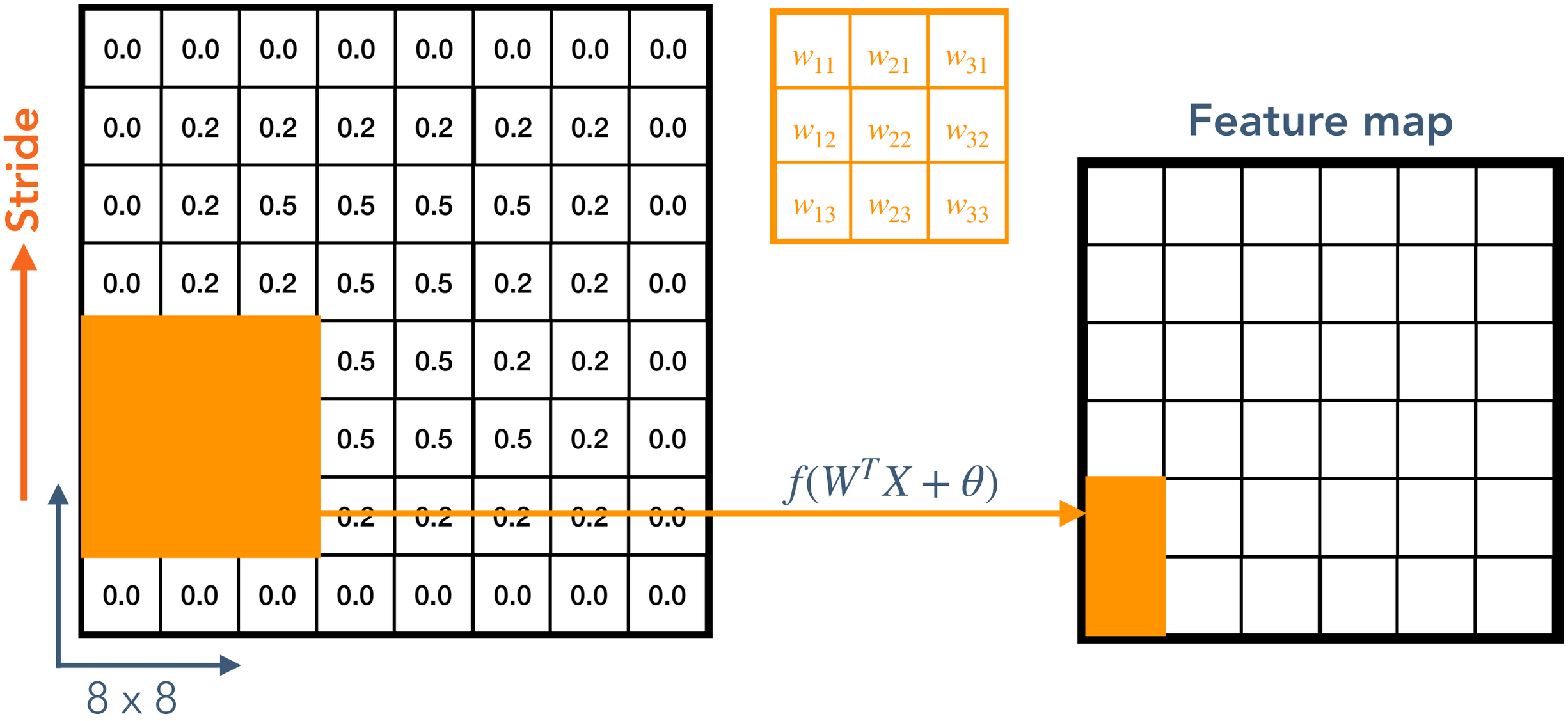
- Apply the filter in "position 1" -> taking a weighted sum of pixels in the filter's **receptive field**.



# Convolution Layers

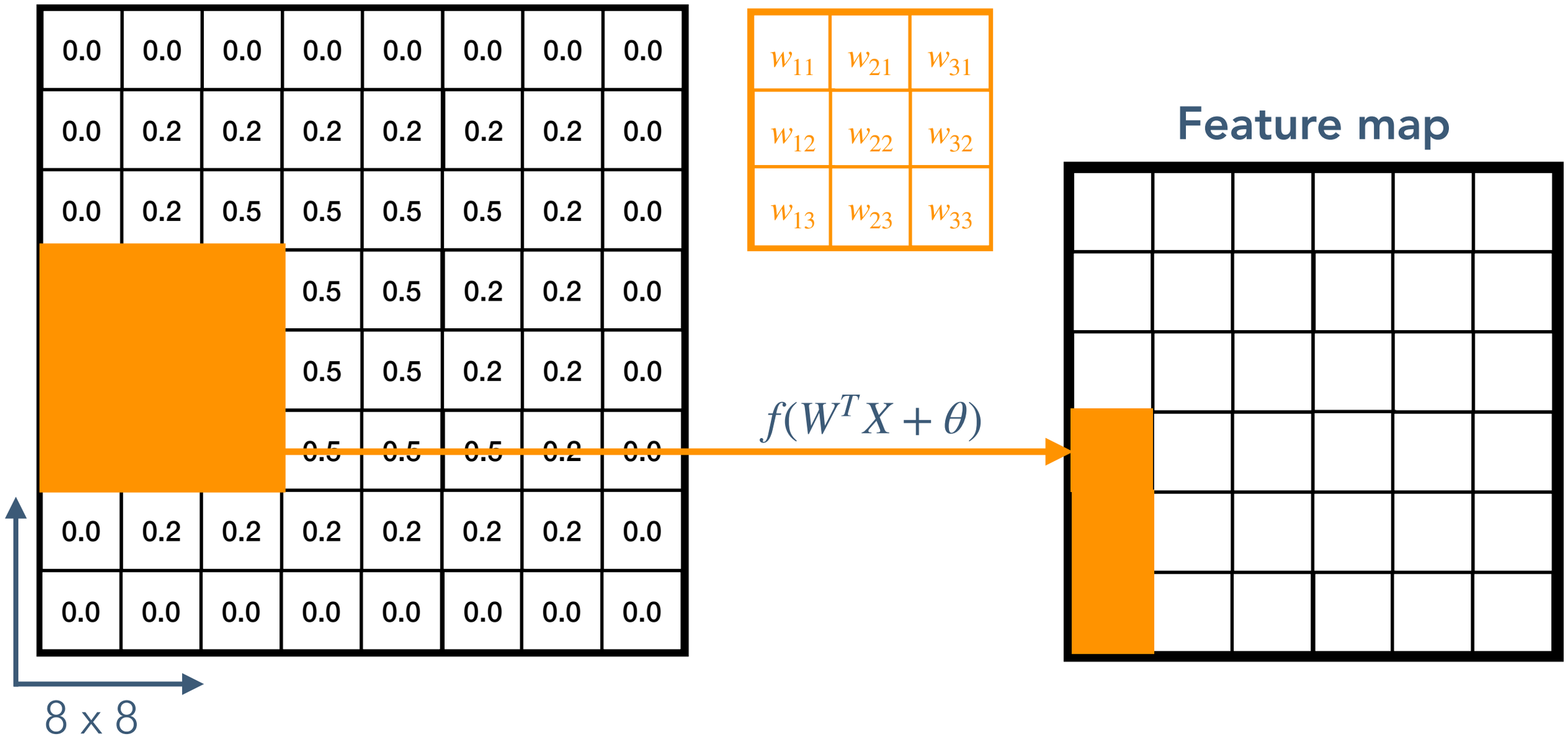
- In the convolutional layers we **perform convolutions** of the whole input image **piece by piece** by sliding **a filters** over the image.

- Apply the filter in position 2. Distance between position 1 and position 2 is governed by the **stride**.



# Convolution Layers

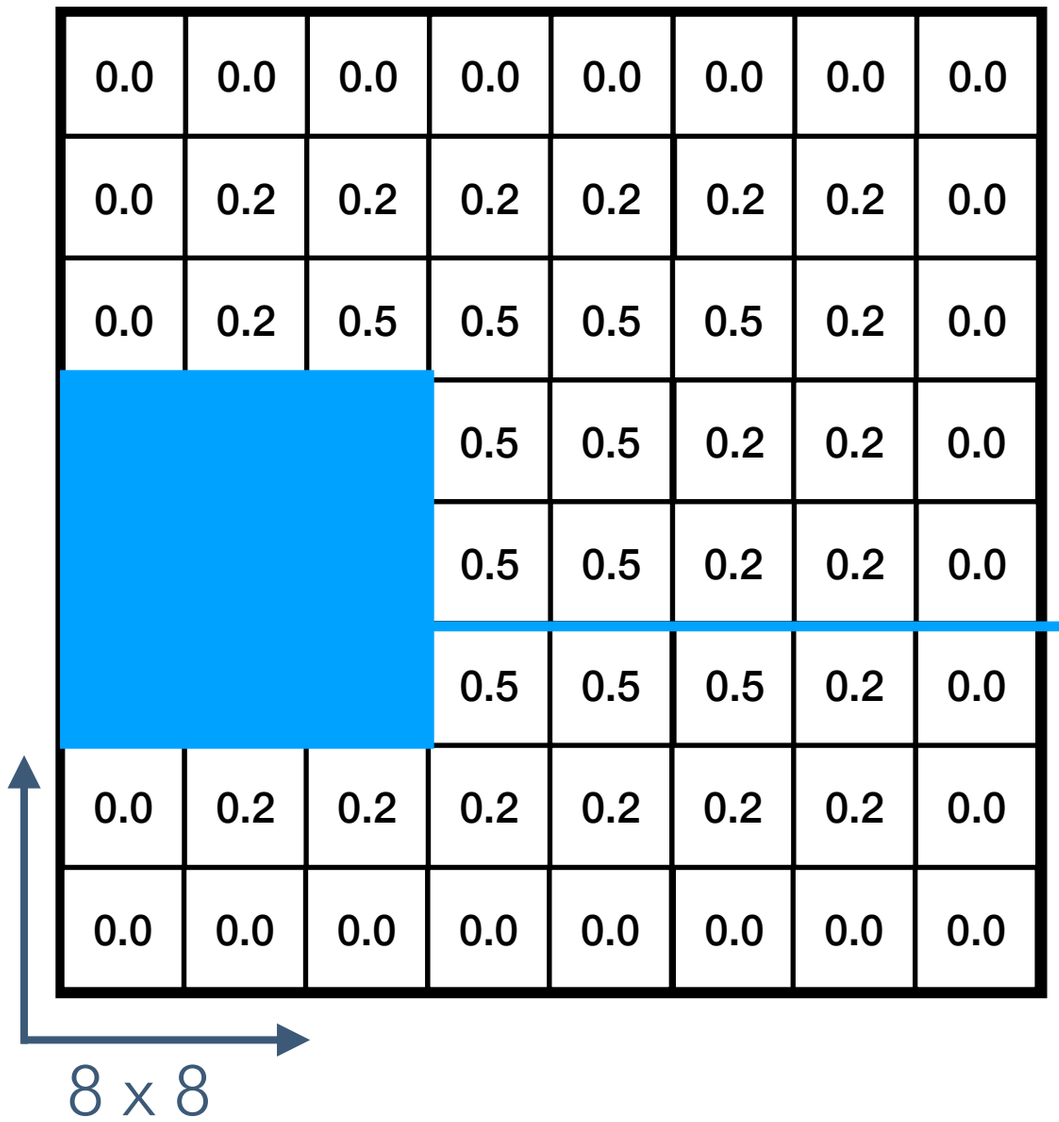
- Each "neuron" in a given feature map has the **same weights**.
  - Reduced numbers of parameters in the model compared to an MLP.
  - Once a CNN has learned to recognise a pattern in one location in the image, it can recognise it **anywhere** in the image - not true for an MLP.

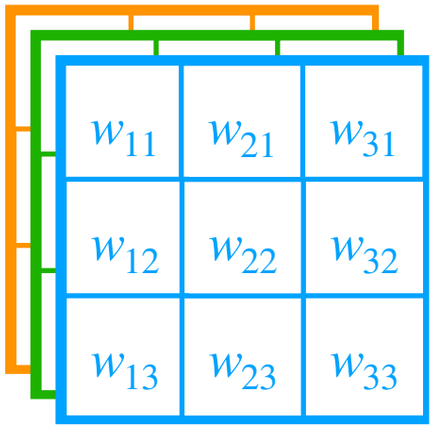


# Convolution Layers

- A network architecture design choice is how many **feature maps** are produced per convolutional layer -> a different filter (and different set of weights) each.

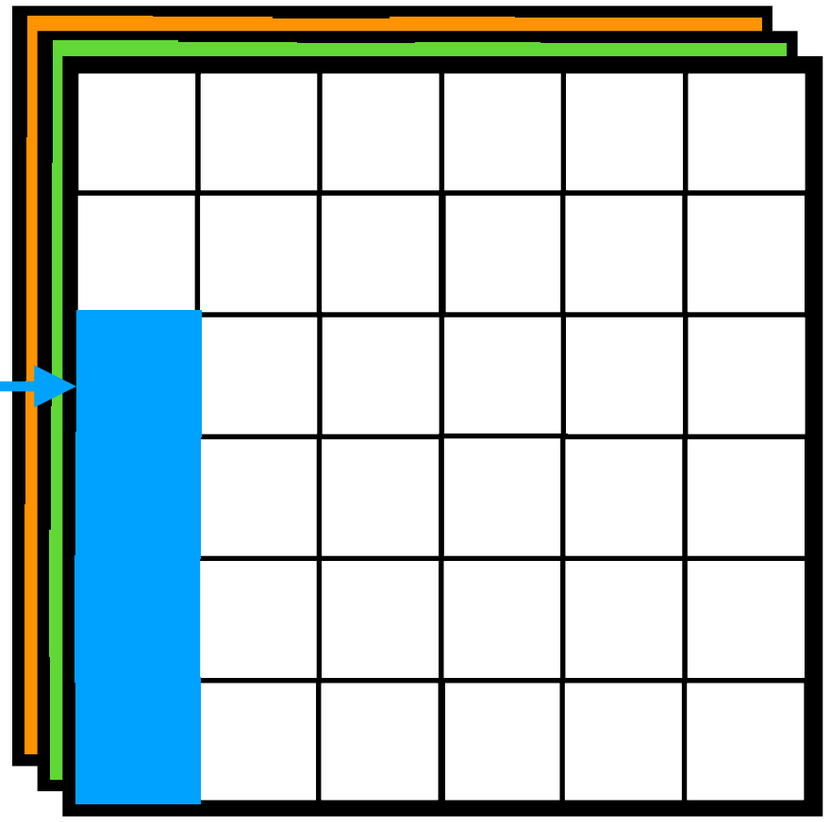
- Each filter enhances a different feature in the image.




$f(W^T X + \theta)$

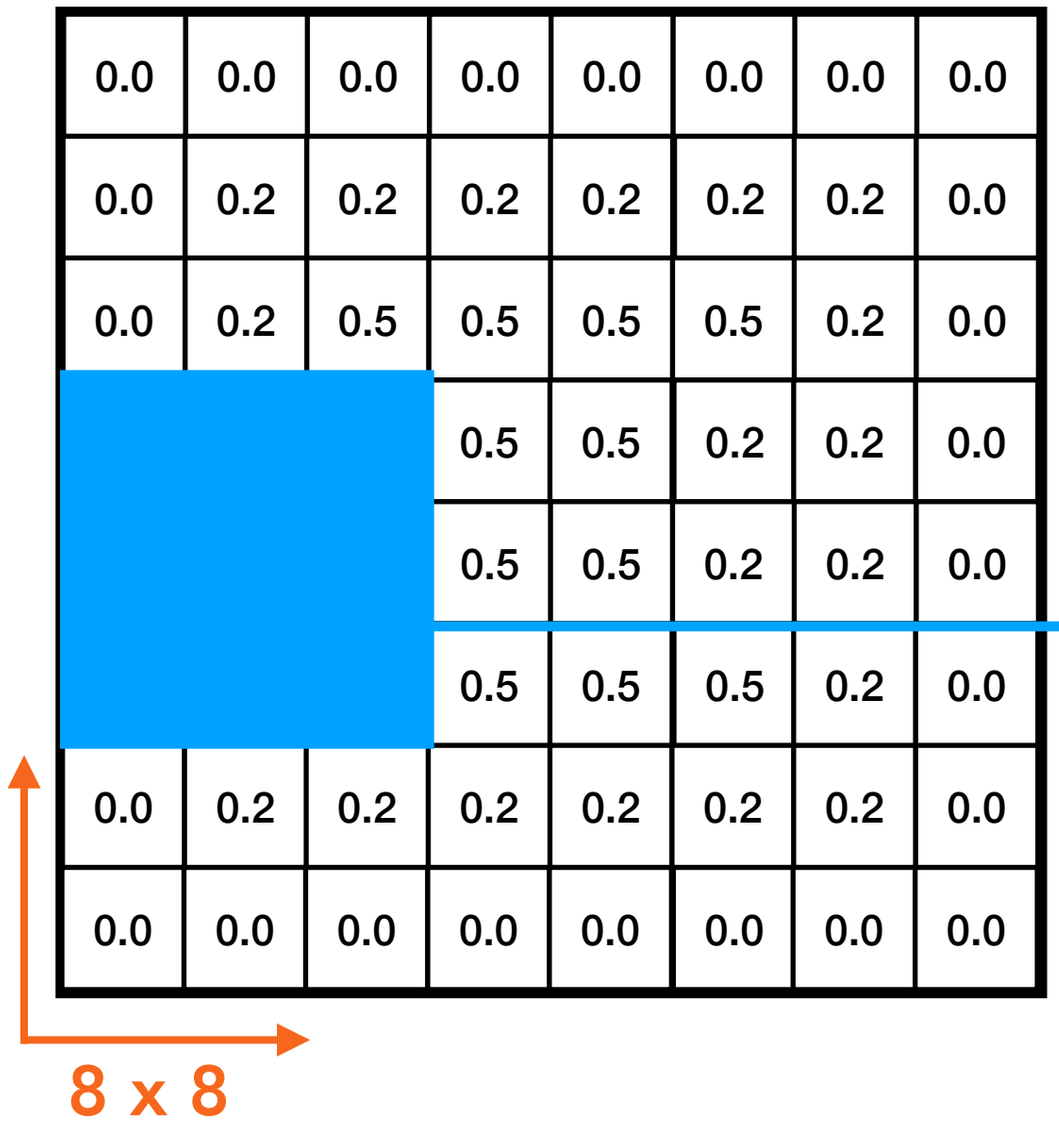
**Multiple feature maps in each convolutional layer**

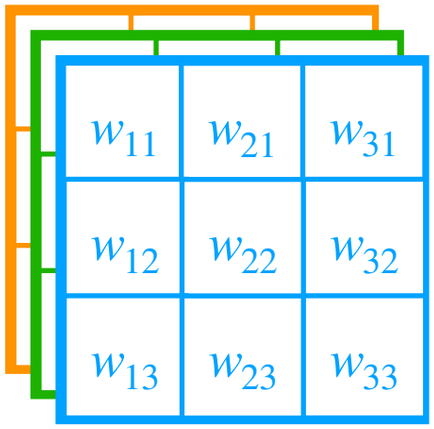


# Dimension Reduction

- The feature map has reduced dimensionality compared to the input image -> loss of information.

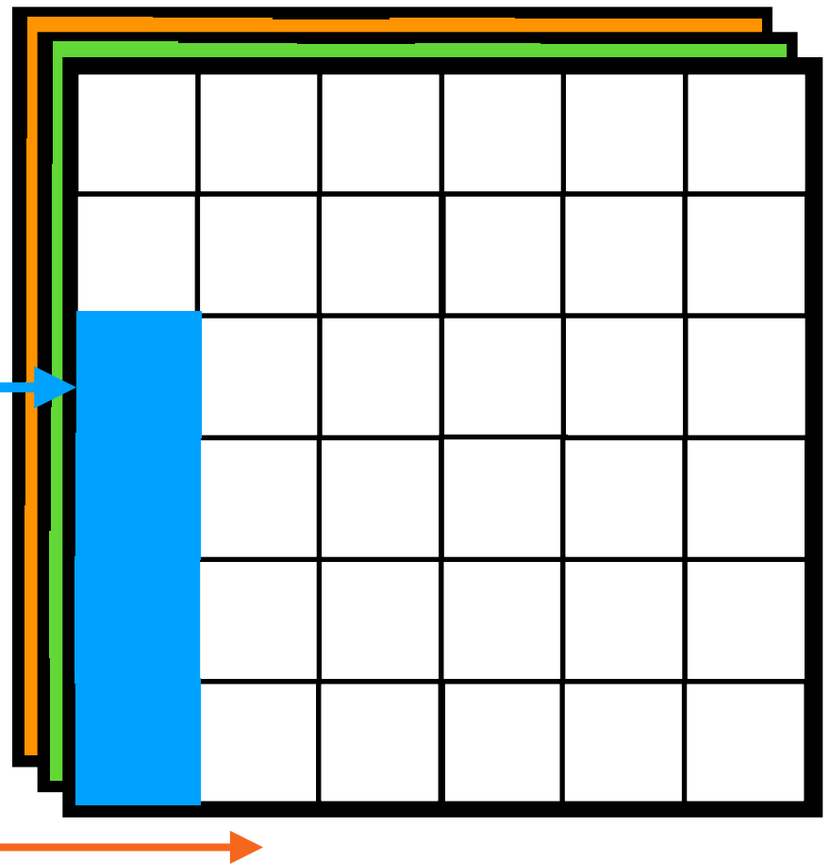
- Comes from size of the filter and the stride used compared to the size of the input image.




$f(W^T X + \theta)$

**Multiple feature maps in each convolutional layer**



# Dimension Reduction

- Loss of information is undesirable, use "zero padding" to recover original input size.

- No none trivial information lost.

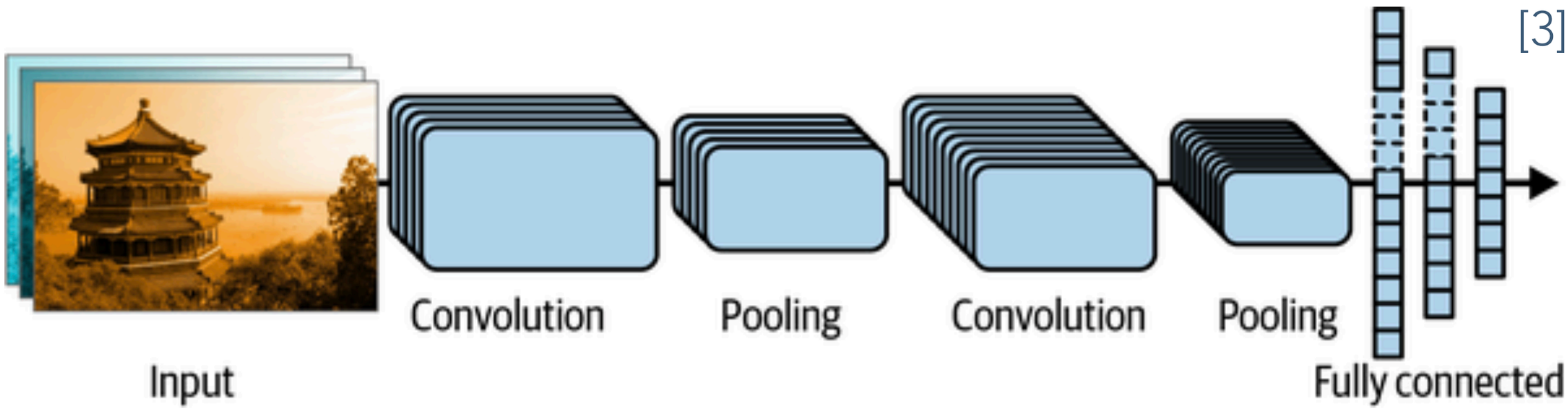
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.2 | 0.5 | 0.5 | 0.5 | 0.5 | 0.2 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.2 | 0.2 | 0.5 | 0.5 | 0.2 | 0.2 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.2 | 0.2 | 0.5 | 0.5 | 0.2 | 0.2 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.2 | 0.5 | 0.5 | 0.5 | 0.5 | 0.2 | 0.0 | 0.0 |
| | | | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 | 0.0 |
| | | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| | | |
|---|---|---|
| $w_{11}$ | $w_{21}$ | $w_{31}$ |
| $w_{12}$ | $w_{22}$ | $w_{32}$ |
| $w_{13}$ | $w_{23}$ | $w_{33}$ |

$f(W^T X + \theta)$

8 x 8

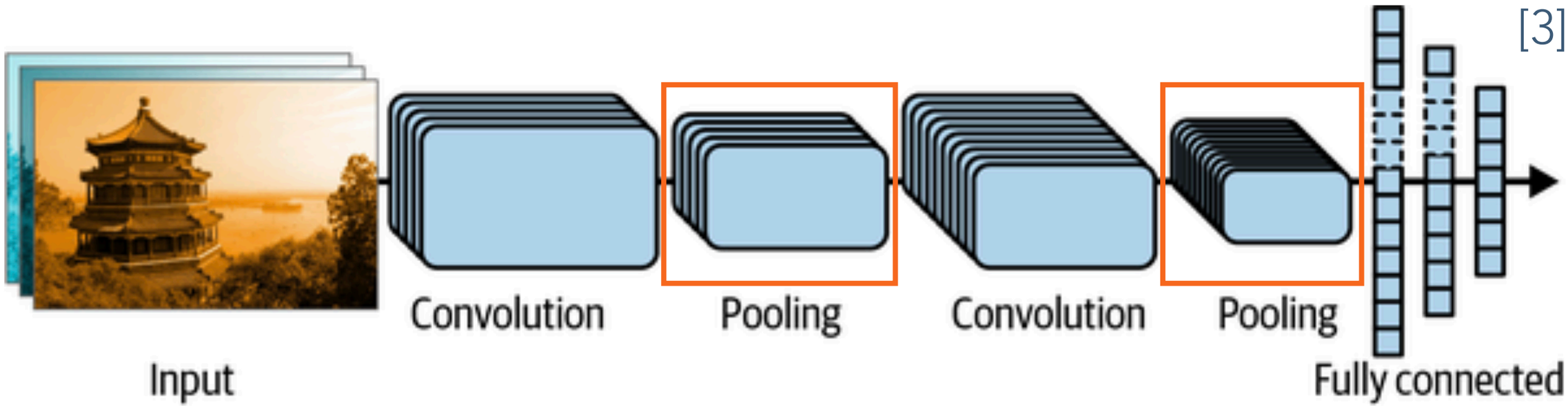
# Typical CNN Architecture




[3]

- One or two convolution layers immediately after the input.
- Followed by a series of a pooling layer followed by a few convolutional layers.
- Finally standard, fully connected MLP to collate the convolution outputs as features and format the output for classification (via softmax).

# Pooling




[3]

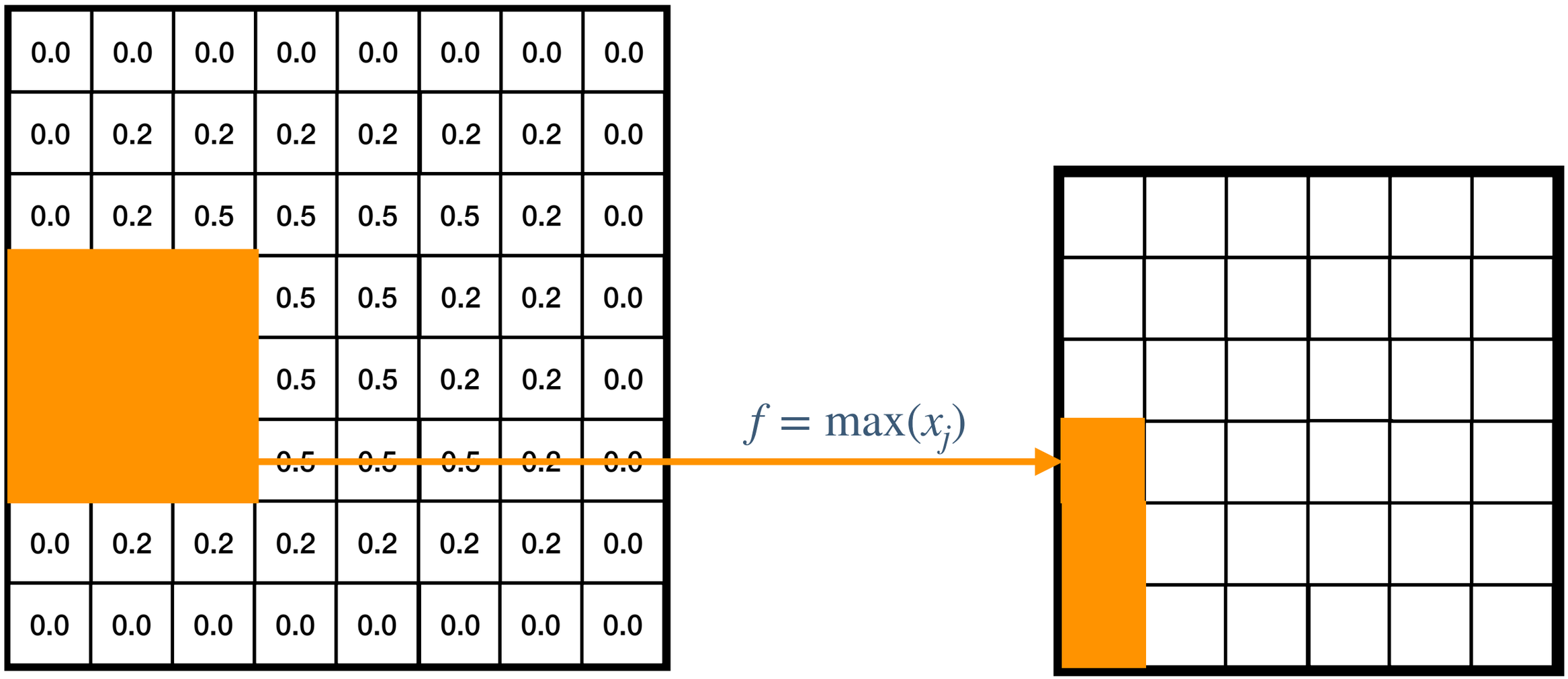
- Sometimes desirable to reduce the dimensionality of input image (or feature map):
  - Reduce the computational load.
  - Reduce memory usage.
  - Reduce the number of parameters (limiting the risk of overfitting).

- Often don't need the highest resolution possible to identify the shapes of objects -> use lower resolution representation and reach the same conclusion.

- This **subsampling** is the goal of the pooling layers.

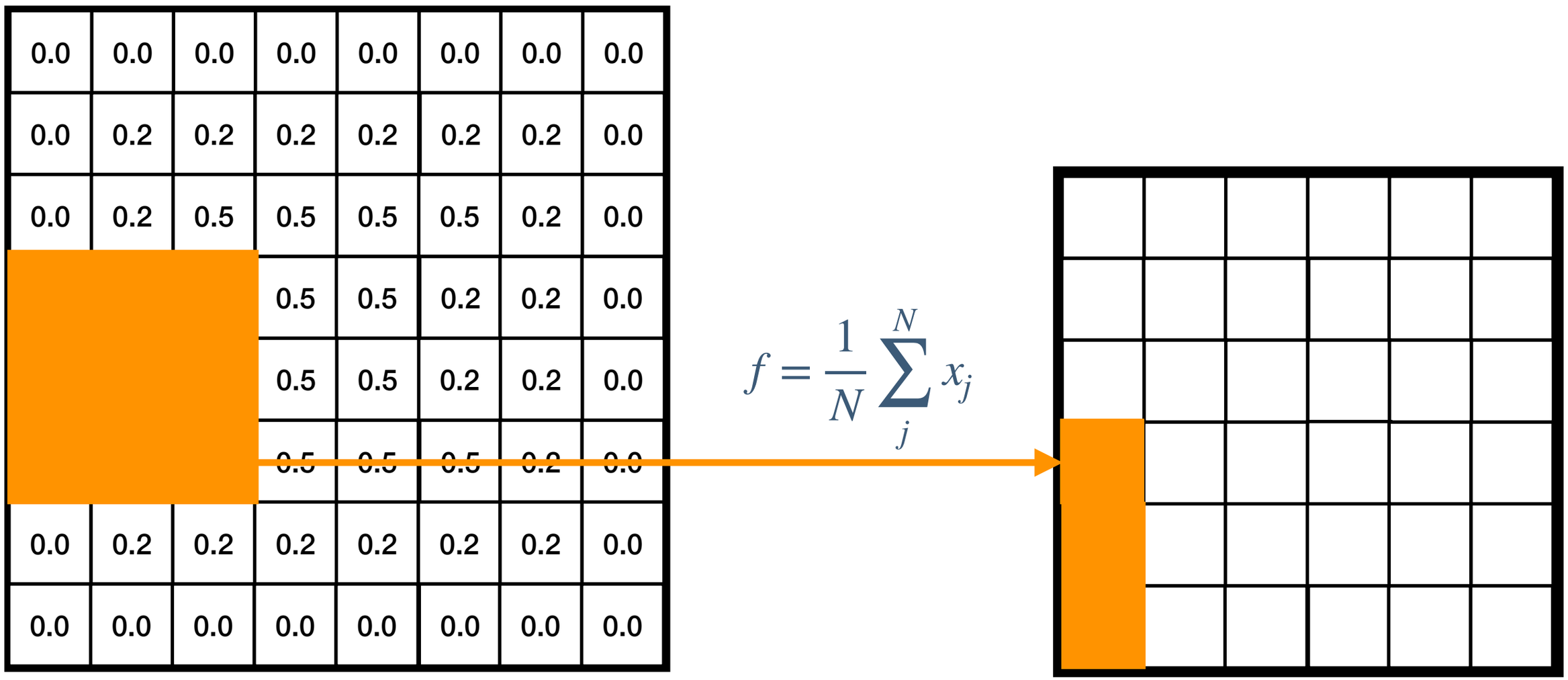
# Pooling Layers

- Define some filter (**kernel**) size, a stride and padding type then perform an operation on the pixels in the kernel's receptive field at each neuron to calculate:
  - their maximum value (**max pooling**).
    - Invariant under small changes - result will be the same post-pooling if input image is rotated or translated slightly -> **don't sweat the small things**.
    - Increase receptive field - summarising information in larger regions enables the network to capture more significant spacial patterns -> **see the bigger picture**.



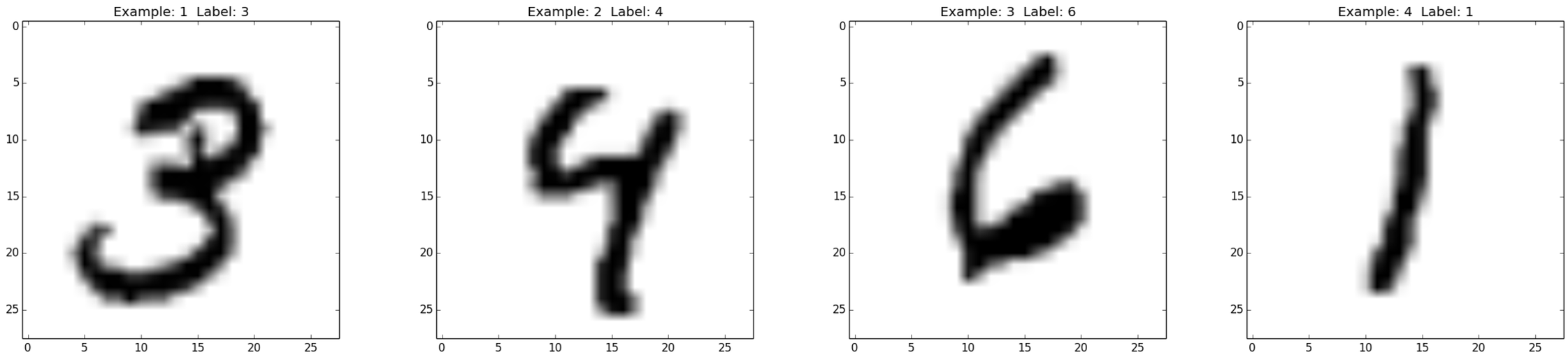
# Pooling Layers

- Define some filter (**kernel**) size, a stride and padding type then perform an operation on the pixels in the kernel's receptive field at each neuron to calculate:
  - their mean value (**average pooling**).
    - Smoother feature representation - less variance in the pixels post-pooling.
    - Reduces sensitive to outliers -> regularisation effect - reducing overfitting by not letting any single pixel dominate.



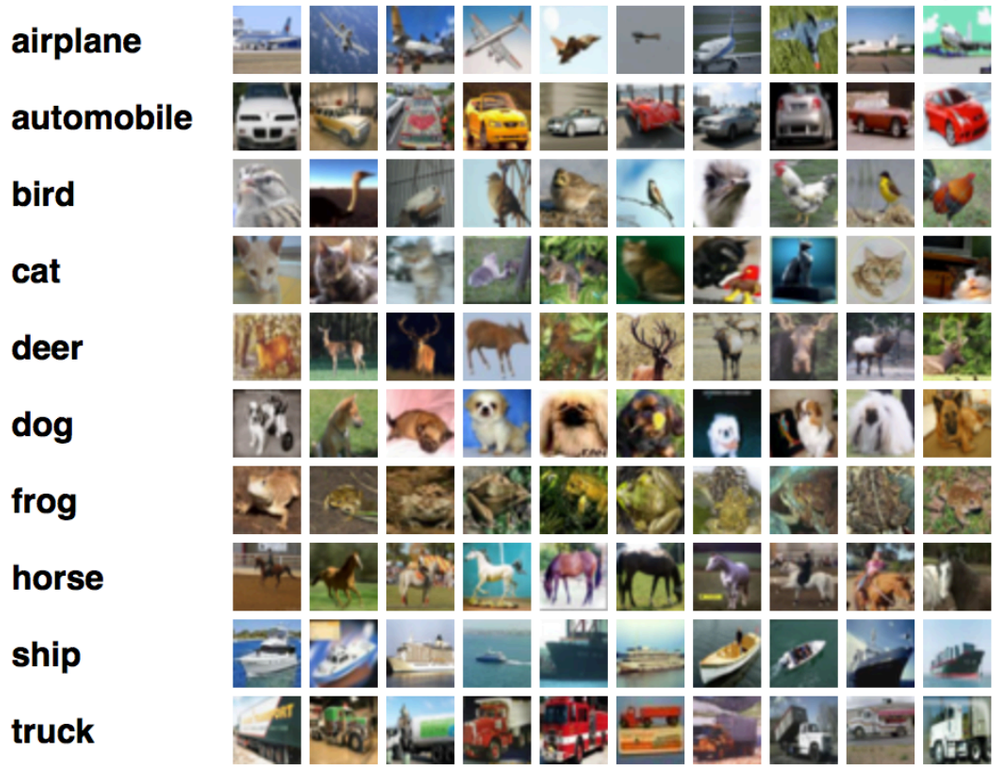
# Today's Notebook

- Learned about all of the basic building blocks of a CNN.
- Discussed the process of convolution of images, why we do it and the need for padding.
- Introduced the concept of pooling, why it's useful and how to apply it.
- Put all of this together to build a CNN to classify images in two datasets.
  - **MNIST**: library of labelled handwritten greyscale numbers [2].



# Today's Notebook

- Learned about all of the basic building blocks of a CNN.
- Discussed the process of convolution of images, why we do it and the need for padding.
- Introduced the concept of pooling, why it's useful and how to apply it.
- Put all of this together to build a CNN to classify images in two datasets.
  - **CFAR10**: library 60000 of labelled 32 x 32 pixel **colour** images in 10 classes [5].




**Trivial to extend CNN to images of arbitrary depth (colour = 3).

-> 3 fold increase in the number of weights in the network.

# Accessing the Notebooks

Follow this link:

**https://github.com/alexbooth92/workshop-UCDN_PracticalML.git**

alexbooth92 / workshop-UCDN_PracticalML

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

**workshop-UCDN_PracticalML** Public

Pin | Unwatch 1 | Fork 0 | Star 0

main | 1 Branch | 0 Tags | Go to file t | Add file | Code

**alexbooth92** Remove commit hash. fc5f23b · 24 minutes ago 12 Commits

| | | |
|---|---|---|
| lecture_1 | Update train test to be better ratio. | 26 minutes ago |
| lecture_2 | Initial commit of notebooks. | 2 weeks ago |
| lecture_3 | Initial commit of notebooks. | 2 weeks ago |
| lecture_4 | Initial commit of notebooks. | 2 weeks ago |
| optional_python_revision | Binder test commit. | 2 weeks ago |
| README.md | Update README.md | 2 weeks ago |
| requirements.txt | Remove commit hash. | 24 minutes ago |

README

## workshop-UCDN_PracticalML

A set of lectures and hands-on exercises introducing machine learning for a workshop At Universidad Católica del Norte, Chile. All material is based on a similar course put together by Abbey Waldron.

You can run the code using TensorFlow 2.11 locally or online. To run, click on the following: launch binder

**About**

A set of lectures and hands-on exercises introducing machine learning for a workshop At Universidad Católica del Norte, Chile.

Readme
Activity
0 stars
1 watching
0 forks

**Releases**

No releases published
Create a new release

**Packages**

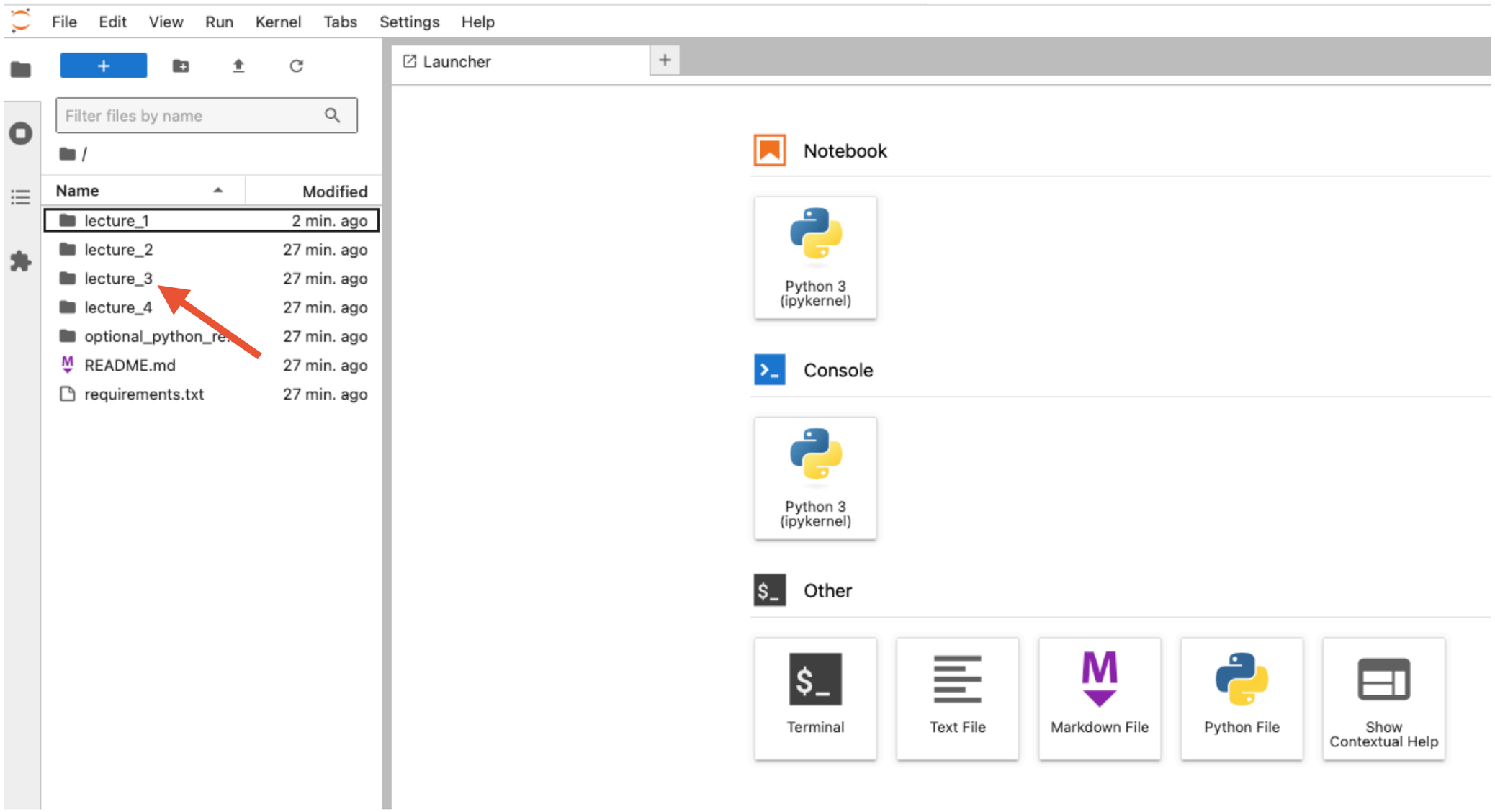
No packages published
Publish your first package

**Languages**

Jupyter Notebook 100.0%

Click here and be patient!

# Accessing the Notebooks




Remember that dropout is your friend!

# Have Fun!

Queen Mary
University of London

# References


[1] https://viso.ai/deep-learning/computer-vision-tasks/

[2] http://yann.lecun.com/exdb/mnist/

[3] A. Géron. Hands on Machine Learning with Scikit-Learn & TensorFlow, 2017.

[4] Neural Competition, Volume 22, Number 12, December 2010.

[5] https://www.cs.toronto.edu/~kriz/cifar.html