Ollscoil na Gaillimhe
University of Galway

Deep Learning

# Topic 05:
# Deep Neural Networks, Part 2


Prof. Michael Madden

Chair of Computer Science
Head of Machine Learning Group
University of Galway

# Learning Objectives – Same As Last Topic

After successfully completing Topics 4 and 5, you will be able to…

- Define key concepts related to Deep Learning, and discuss major advances relative to shallow neural networks
- Explain and implement approaches to handling unstructured data and multi-class classification
- Explain and implement regularization methods, and algorithms including Mini-Batch Gradient Descent, Momentum, RMSprop and Adam
- Explain the principles of operation of Convolutional Networks
- Correctly use these features within ML libraries, including selecting hyperparameters


# Previous Topic: DNNs Part 1

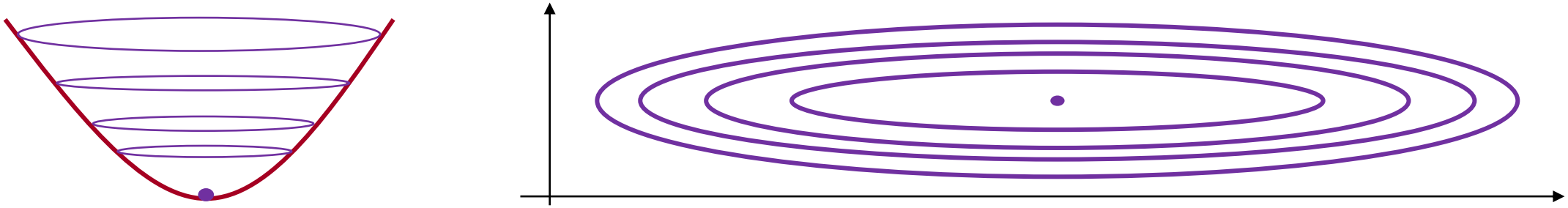
- Deep Neural Networks; Their Representation Power; Challenges Arising
- Handling Unstructured Data; Multi-Class Classification
- Methods to Avoid Overfitting
- Training Algorithm Challenges
- Mini-Batch Gradient Descent

# This Topic: DNNs Part 2

- More Training Algorithms:
  - Backprop with Momentum
  - RMSprop
  - Adam
- Locally Connected Networks
- Convolutional Neural Networks

# Training Algorithms: Backprop with Momentum Overview

- Basic ideas behind Backprop with Momentum:
  - Each parameter should be able to change at a rate appropriate to itself, rather than there being one fixed learning rate
  - The previous changes in parameter values should influence the current direction of updates
  - To achieve this, for each parameter, compute an exponentially weighted moving average of previous gradients, and use that to update the parameter



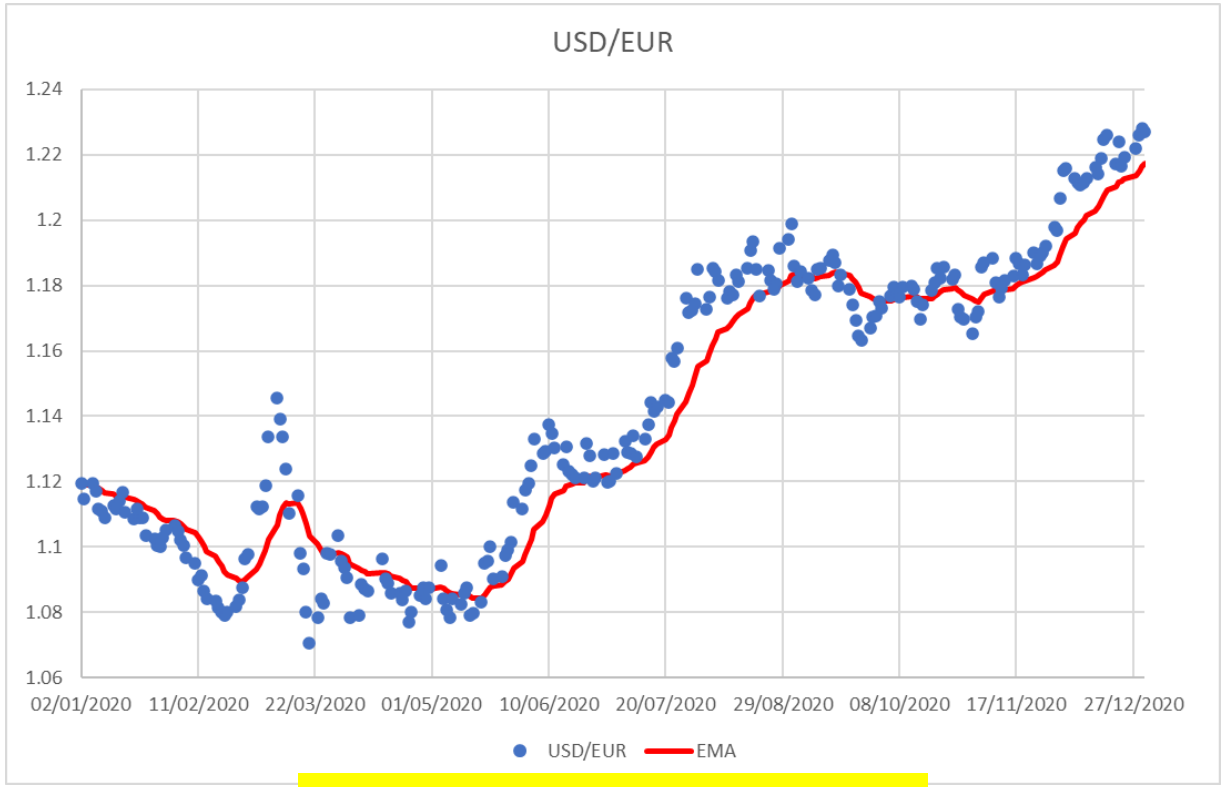
# Exponential Moving Average

- This is a low-cost way of computing a moving average for a stream of timeseries data points: $D_1, D_2, D_3$ …

  At Time $t$=1: $V_1 = D_1$

  At Time $t$>1: $V_t = (1 - \beta)\, D_t + \beta\, V_{t-1}$

- Current value of $D_t$ contributes most to current $V_t$, but all past values also contribute, with exponentially decreasing weight
- Roughly, the most recent $1/(1-\beta)$ steps contribute most to the current value
- Main attraction is its simplicity: only one extra value to store per series, and a simple calculation
- For Backprop with Momentum, a common value of $\beta$ is 0.9, though it is another hyperparameter that can be adjusted




USD-EUR-moving-average.xlsx

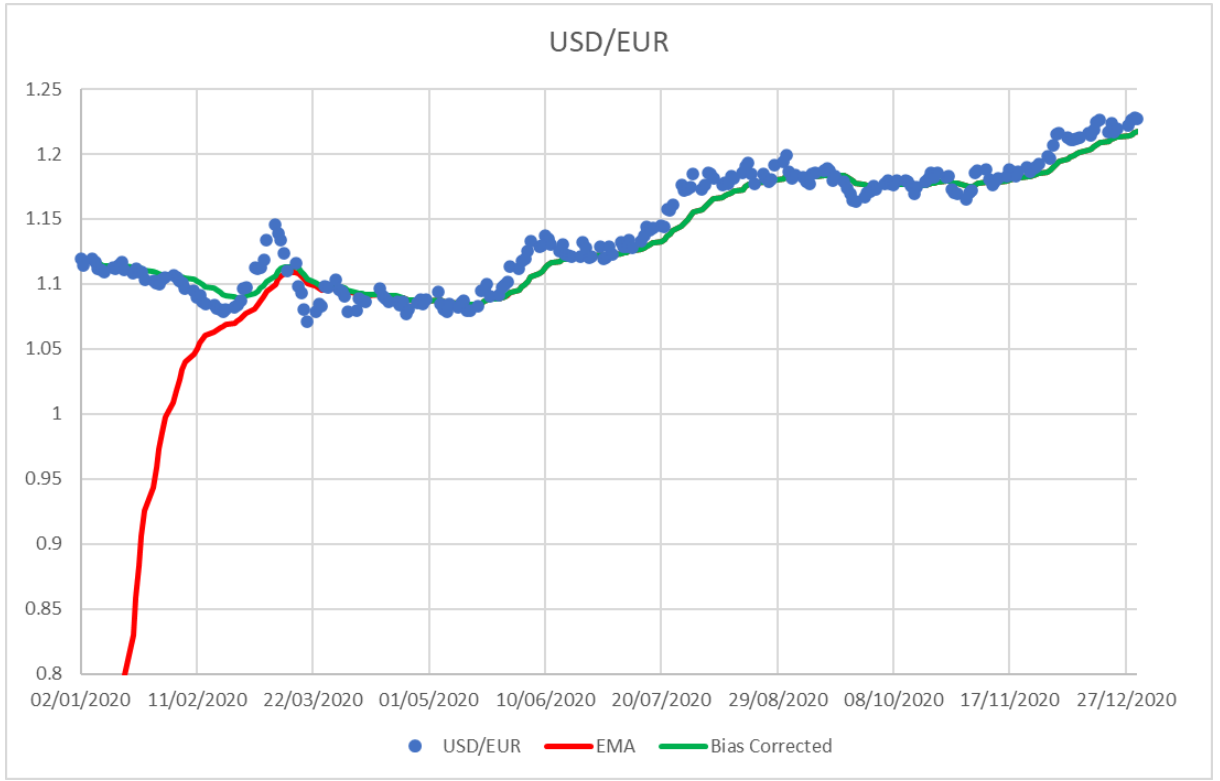
# Exponential Moving Average with Bias Correction

- Instead of initialising $V_1 = D_1$, we can initialise $V_0 = 0$ and then use bias correction:

  At Time $t = 0$: $V_0 = 0$

  At Time $t \geq 1$: $V_t = (1 - \beta)\, D_t + \beta\, V_{t\text{-}1}$

  $VC_t = V_t \,/\, (1 - \beta^t)$




- Red and green plotted lines correspond to red and green results
- If we initialise to 0 and don't use bias correction, the first values are very bad estimates
- When t=1, this calculation undoes the initialisation to 0, and at t gets large, $(1 - \beta^t)$ -> 1 so correction fades.

# Training Algorithms: Backprop with Momentum Algorithm

*Note:* omitting all subscripts and superscripts for weights $W$ and biases $b$, because you apply it to every one of them

**Modifying Backprop to add Momentum Algorithm:**

**At start:** for each weight $W$ and bias $b$, will store $V_{\Delta W}$ and $V_{\Delta b}$

**During each iteration:**

If it's first iteration, initialise the *V* values: $V_{\Delta W} = \Delta W, \;\; V_{\Delta b} = \Delta b$

After the Backprop step (when you have computed new values for all $\Delta W$ and $\Delta b$ terms), update the V values:

$$V_{\Delta W} = (1 - \beta)\, \Delta W + \beta\, V_{\Delta W}$$

$$V_{\Delta b} = (1 - \beta)\, \Delta b + \beta\, V_{\Delta b}$$

**Do Gradient Descent update step** using $V_{\Delta W}$ and $V_{\Delta b}$:

$$W \mathrel{-}= \alpha V_{\Delta W}, \;\; b \mathrel{-}= \alpha V_{\Delta b}$$

# Training Algorithms: RMSprop Overview

- Another adaptive learning rate algorithm, proposed by Geoff Hinton, and strongly related to an earlier one called AdaGrad
  - For each parameter, keep a moving average of its squared gradient
  - When updating, divide the current gradient by the square root of the average squared gradient
- In Backprop with Momentum, if the analogy is that we are calculating "velocity" terms, then in RMSprop we are calculating "acceleration" terms
  - In both cases, we are using these terms to adapt the learning rate individually for each parameter, allowing faster learning overall

# Training Algorithms: RMSprop Algorithm

*Note:* omitting all subscripts and superscripts for weights $W$ and biases $b$, because you apply it to every one of them

**Modifying Backprop to turn it into RMSprop:**

**At start:** for each weight $W$ and bias $b$, initialise the $S$ values to 0:

$S_{\Delta W} = 0, \;\; S_{\Delta b} = 0$

**During each iteration:**

After the Backprop step (when you have computed new values for all $\Delta W$ and $\Delta b$ terms), update the $S$ values:

$$S_{\Delta W} = (1 - \beta)\, \Delta W^2 + \beta\, S_{\Delta W}$$

$$S_{\Delta b} = (1 - \beta)\, \Delta b^2 + \beta\, S_{\Delta b}$$

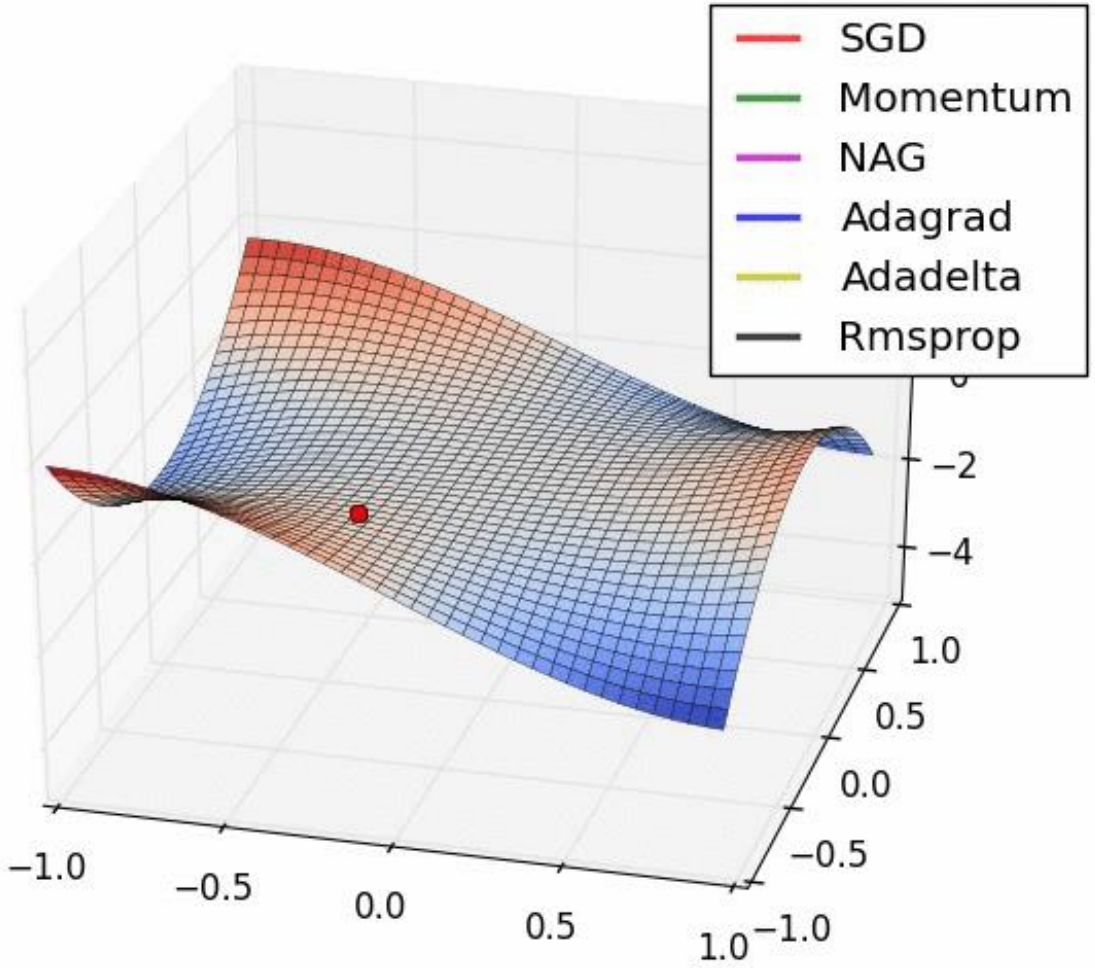
**Do Gradient Descent update step** using square roots of $S_{\Delta W}$ and $S_{\Delta b}$:

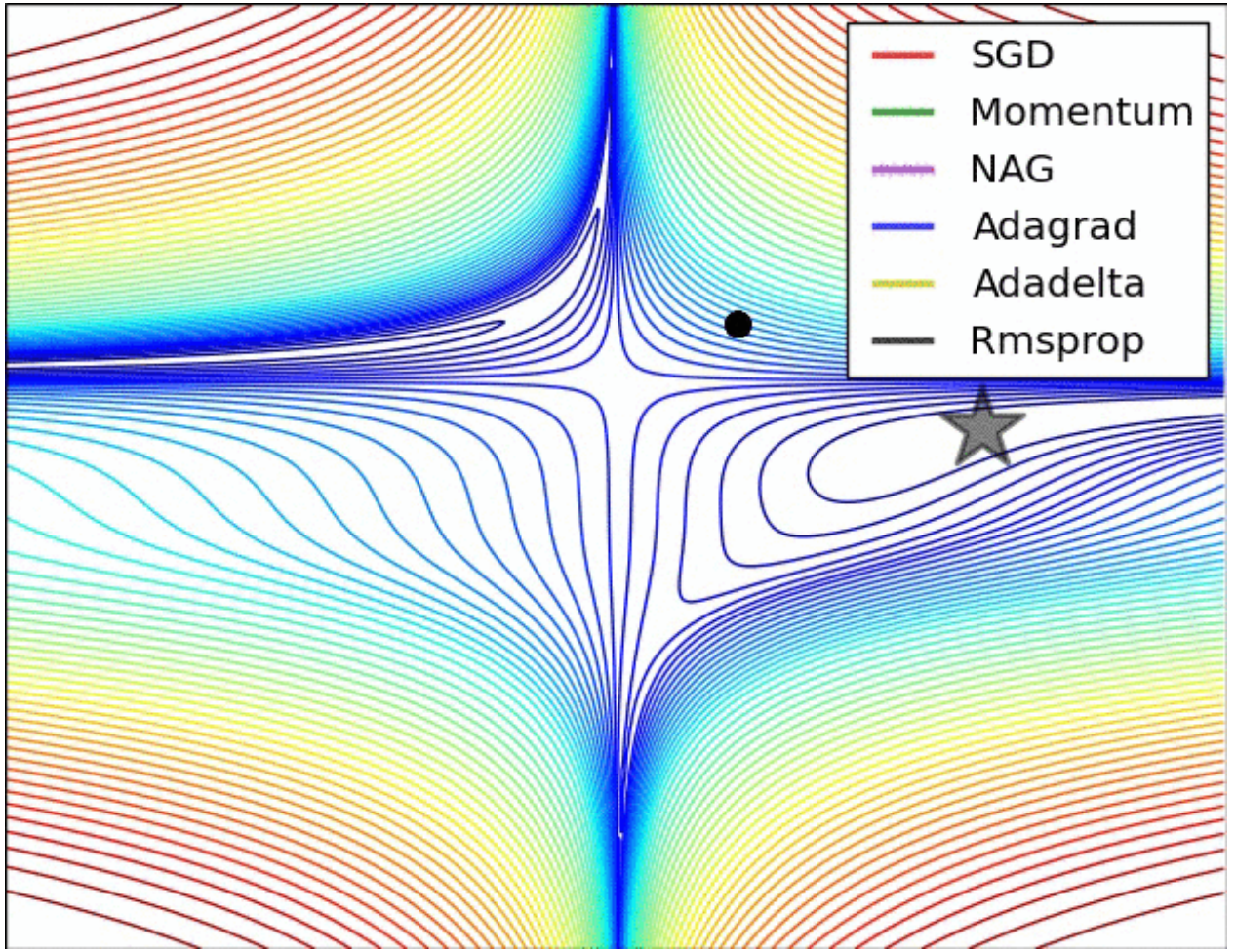
$$W \;-= \; \alpha \frac{\Delta W}{\sqrt{S_{\Delta W}} + \epsilon} \qquad b \;-= \; \alpha \frac{\Delta b}{\sqrt{S_{\Delta b}} + \epsilon}$$

where $\epsilon$ is a very small value, e.g. $\epsilon$ = 1e-8, for numerical stability to avoid dividing by 0 or close to 0

# Visualising Some Training Algorithms

Source: https://imgur.com/a/Hqolp#NKsFHJb

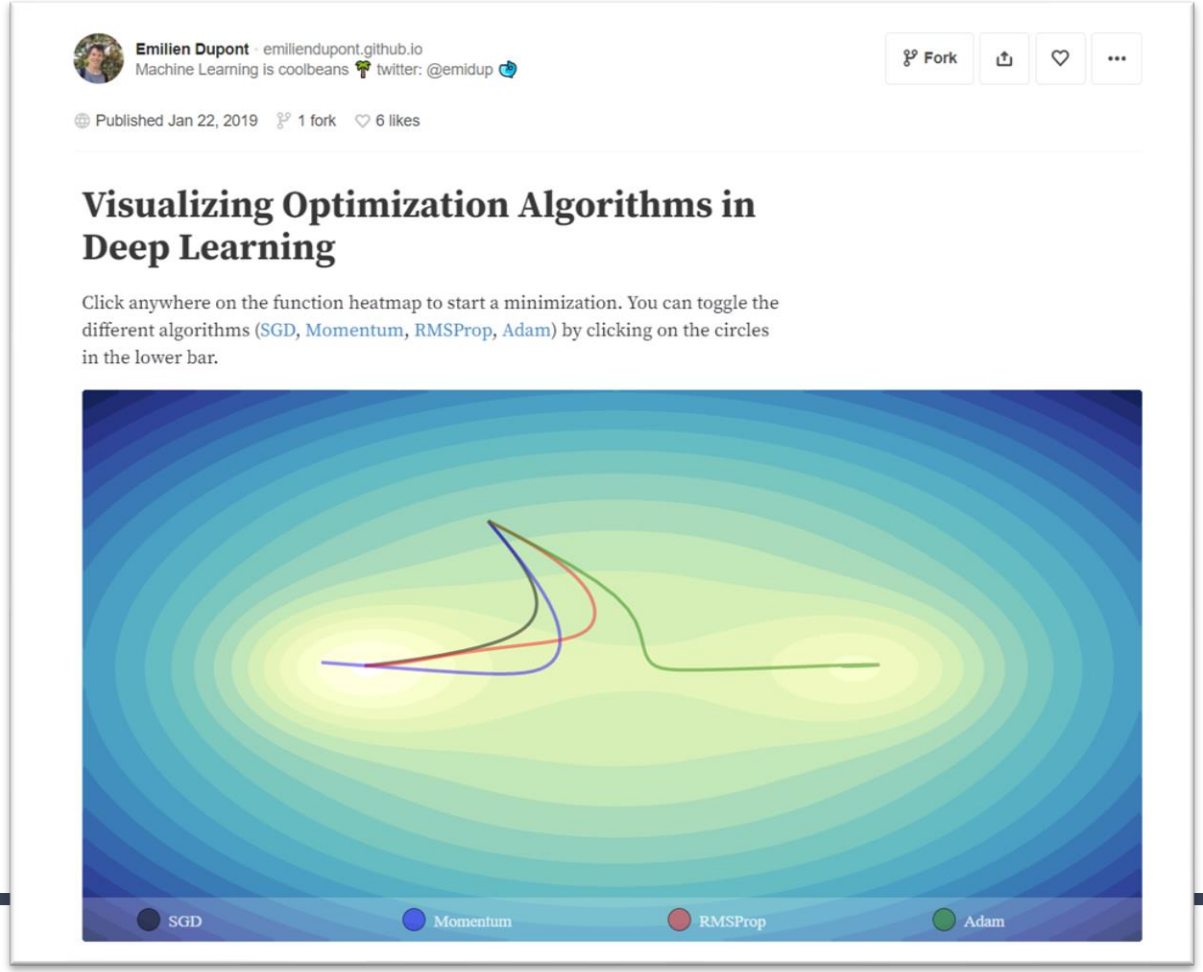


# Visualising Some Training Algorithms

Another nice visualisation – this one is interactive:

https://observablehq.com/@emiliendupont/visualizing-optimization-algorithms-in-deep-learning



# Training Algorithms: Adam Optimisation Algorithm Overview

- Essentially, combine the ideas from Momentum and RMSprop: consider both velocity and acceleration terms
  - Name comes from "Adaptive Moment Estimation", as 'velocity' and 'acceleration' terms are the 1st and 2nd moments of gradients
  - For the moving average, Adam initialises all terms to 0 and performs the bias correction calculation
  - There are two separate $\beta$ parameters:
    $\beta_1$ for "velocity" moving average
    $\beta_2$ for "acceleration" moving average
  - In theory, this means there are 2 more hyperparameters to set, but in practice, Adam works well with default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$
- Adam Optimizer was published as a poster at ICLR in 2015
  - since then has become very widely used as people have found it to be robust and effective.

# Training Algorithms: Adam Optimisation Algorithm Details

Here, omitting subscripts and superscripts on $W$ and $b$ terms, except at very end.

## Modifying Backprop to turn it into Adam Optimizer:

**At start:** for each weight $W$ and bias $b$, initialise all $V$ and $S$ terms to 0:

$V_{\Delta W} = 0,\ V_{\Delta b} = 0,\ S_{\Delta W} = 0,\ S_{\Delta b} = 0$

**At each iteration, $t$:**

After the Backprop step, update all $V$ and $S$ terms and calculate corrections:

$$V_{\Delta W} = (1 - \beta_1)\,\Delta W + \beta_1\, V_{\Delta W}\ , \qquad V_{\Delta b} = (1 - \beta_1)\,\Delta b + \beta_1\, V_{\Delta b}$$

$$S_{\Delta W} = (1 - \beta_2)\,\Delta W^2 + \beta_2\, S_{\Delta W}\ , \qquad S_{\Delta b} = (1 - \beta_2)\,\Delta b^2 + \beta_2\, S_{\Delta b}$$

$$VC_{\Delta W} = V_{\Delta W}/(1 - \beta_1{}^t)\ , \qquad VC_{\Delta b} = V_{\Delta b}/(1 - \beta_1{}^t)$$

$$SC_{\Delta W} = S_{\Delta W}/(1 - \beta_2{}^t)\ , \qquad SC_{\Delta b} = S_{\Delta b}/(1 - \beta_2{}^t)$$

Do Gradient Descent update step using these new terms:

$$W_{j,i}^{[l]} \mathrel{-}= \alpha\, \frac{VC_{\Delta W}{}_{j,i}^{[l]}}{\sqrt{SC_{\Delta W}{}_{j,i}^{[l]}} + \epsilon} \qquad b_j^{[l]} \mathrel{-}= \alpha\, \frac{VC_{\Delta b}{}_{j}^{[l]}}{\sqrt{SC_{\Delta b}{}_{j}^{[l]}} + \epsilon}$$

where $\epsilon$ is a very small value, e.g. $\epsilon$ = 1e-8, for numerical stability

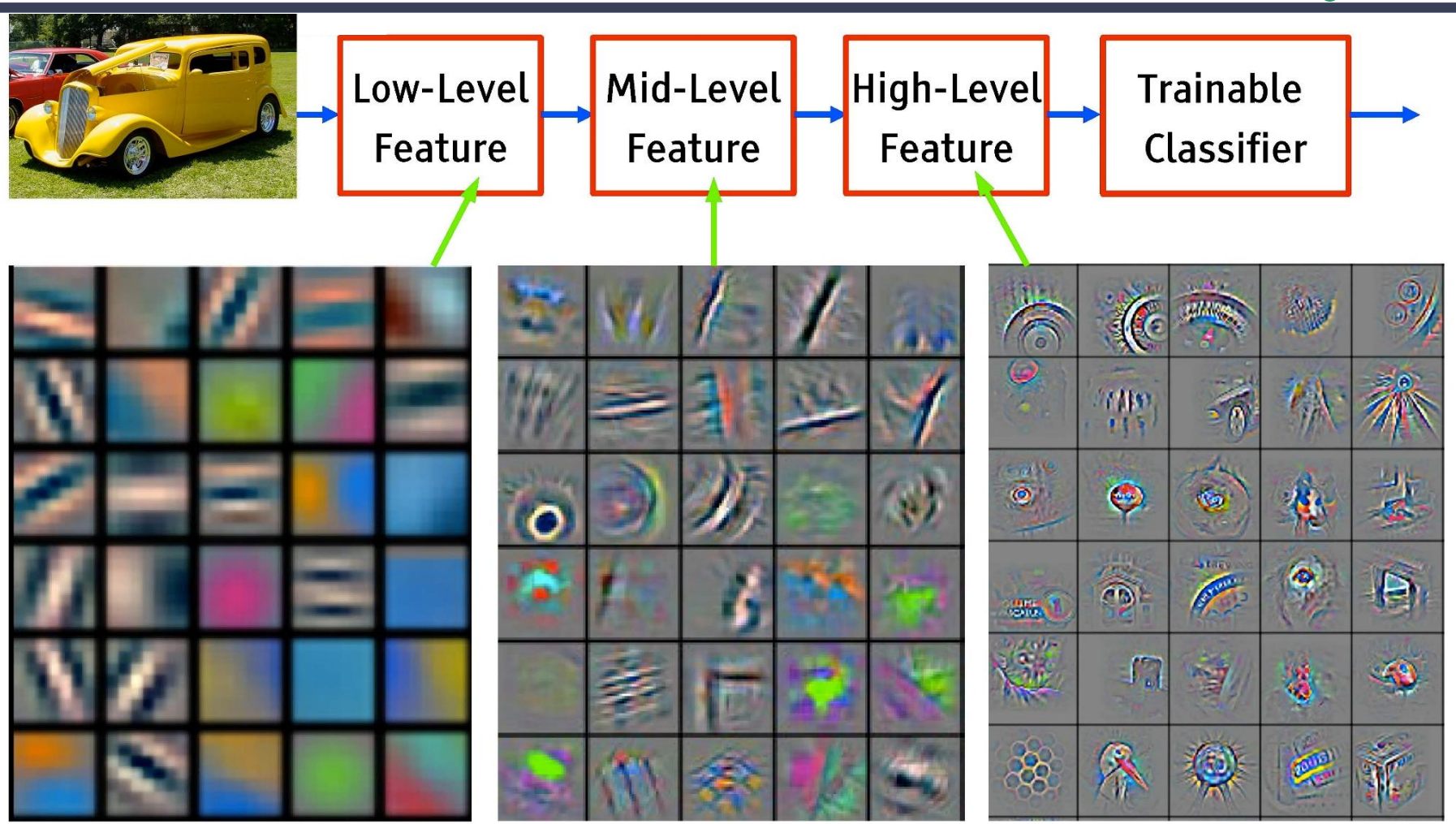
# Locally Connected NNs

- In tasks such as image analysis, each input node (pixel) has a local context
  - Makes more sense to consider pixels adjacent to each other rather than far away from each other
- Leads to idea of locally connected NNs:
  - Input nodes connected in "patches" (**fields**) to nodes in next layer, e.g. 20x20 image, 5x5 patches that all overlap, so hidden layer has 16x16 = 256 nodes
- Termed **Local Receptive Fields**
  - Inspired by how the eye works
  - Much fewer weights than the fully connected NNs we have been looking at
- Similar ideas apply to analysis of other forms of data, such as audio and text, where each input may have a local context

# Convolutional Neural Nets – Overview

- Building on previous idea, introduce a set of **shared weights and biases** for each field
  - Modify the backprop algorithm for these nodes so that they are always the same ("tied")
  - This means that they become a **feature detector** that detects the same feature everywhere
  - Helps achieve *translational invariance* in computer vision
  - Number of parameters becomes *much* smaller
- Usually have multiple convolutional layers in a CNN
  - Idea that earlier layers might detect low-level features,
    and later layers might aggregate these to detect more complex features

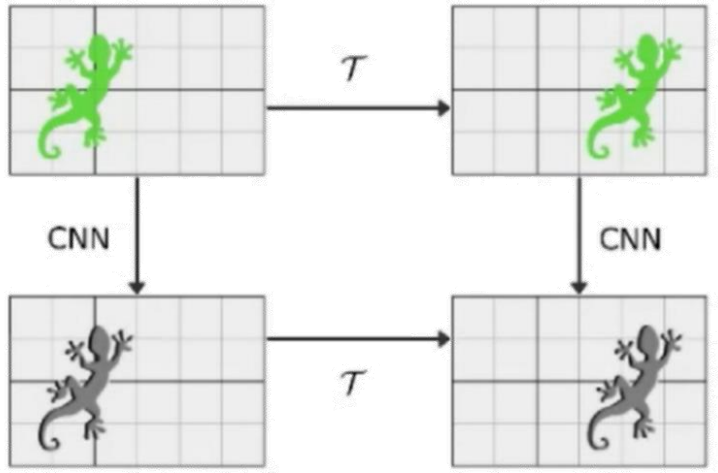
# Convolutional Neural Nets: Visualisations of Features in Different Layers




Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# CNNs Incorporate Important Ideas that can Help Learning

- Sparse Connectivity
  - Rather than having fully-connected layers, only some units in one layer are derived from values of some units in previous layer
  - Reduces number of weights, reducing overfitting risk, computational cost and training demands
- Parameter Sharing
  - Rather than learning a separate parameter for each connection, learn shared parameters that represent specific operations
  - Again, reduces number of weights
- Equivariant Representations
  - If you apply a transformation to input and then put it through conv layer, equivalent to putting it through conv layer and then applying the transformation
  - For CNNs, true of translation operations specifically, but not others such as rotations




[Image from Max Welling, ICPR 2021]

# Convolution Operators and Filters in Computer Vision

- A convolution operation in computer vision:
  - Step across each section of an **input**
  - Perform element-wise multiplication by a **filter** and sum results
  - Result is a **feature map**
- Example: **input** (e.g. greyscale image) is green, **filter** (or **kernel** or **convolution matrix**) is blue, result in pink is a **feature map**

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 4 | 3 | 2 | 0 |
| 1 | 1 | 2 | 4 | 5 | 4 | 1 |
| 3 | 0 | 0 | 1 | 2 | 3 | 4 |
| 4 | 5 | 2 | 1 | 1 | 2 | 3 |
| 4 | 3 | 2 | 0 | 2 | 1 | 1 |
| 4 | 5 | 4 | 1 | 2 | 4 | 5 |
| 3 | 1 | 4 | 2 | 1 | 3 | 5 |

*

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

=

| | | | | |
|---|---|---|---|---|
| 1 | -7 | -5 | 0 | 5 |
| 4 | 0 | -4 | -3 | 0 |
| 7 | 6 | -1 | -4 | -3 |
| 4 | 11 | 3 | -5 | -4 |
| 1 | 6 | 5 | -5 | -6 |

# Convolution Operators and Filters – What is the Point? (1)

- What is the point?
  - Depending on the filter you apply, you can detect different kinds of features
  - Examples: horizontal and vertical edges

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 |

*

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

=

| | | | | |
|---|---|---|---|---|
| 0 | 30 | 30 | 0 | 0 |
| 0 | 30 | 30 | 0 | 0 |
| 0 | 30 | 30 | 0 | 0 |
| 0 | 30 | 30 | 0 | 0 |
| 0 | 30 | 30 | 0 | 0 |

# Convolution Operators and Filters – What is the Point? (2)

- What is the point?
  - Depending on the filter you apply, you can detect different kinds of features
  - Examples: horizontal and vertical edges

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 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 |

*

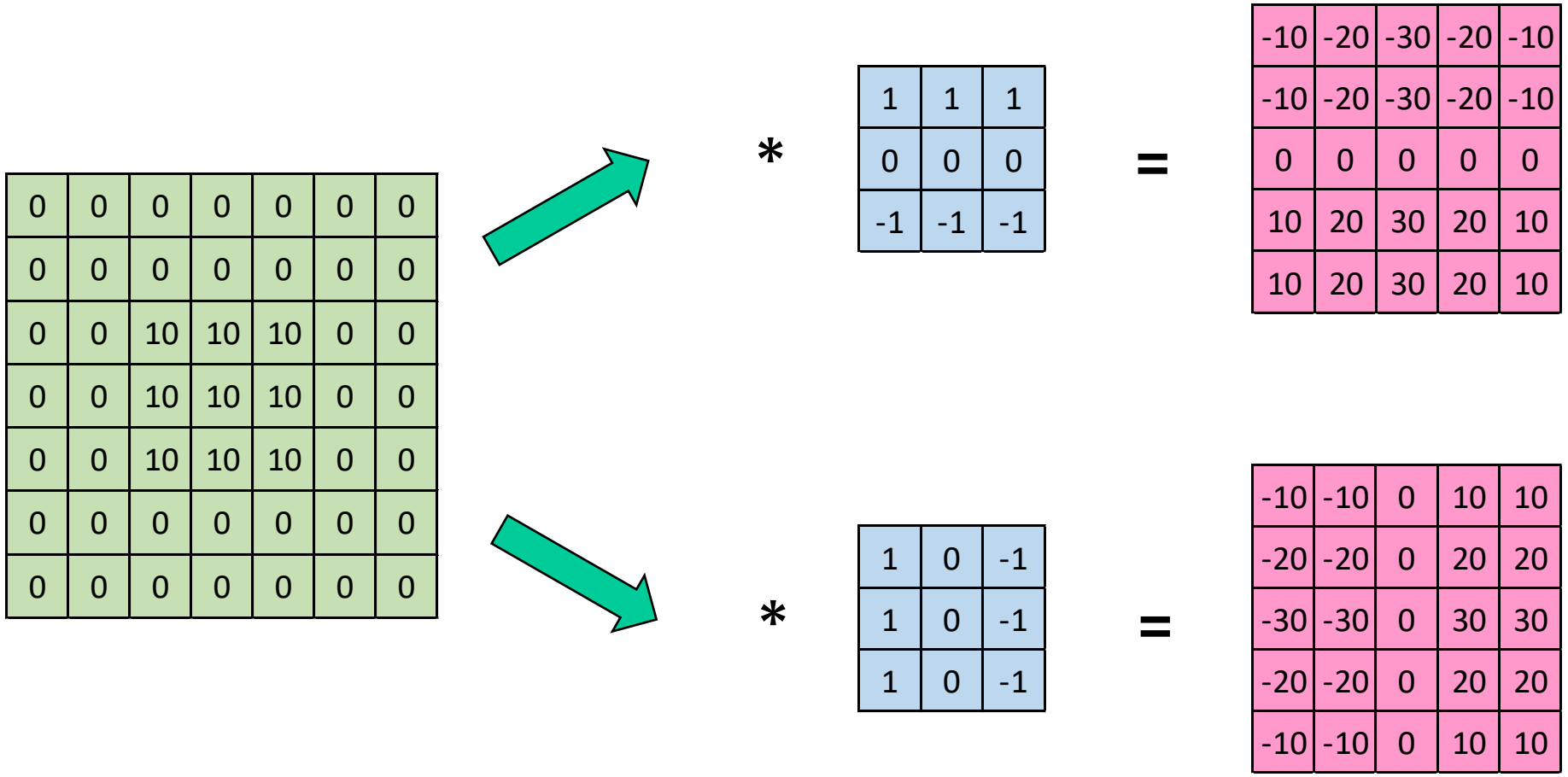
| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

=

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 30 | 30 | 30 | 30 | 30 |
| 30 | 30 | 30 | 30 | 30 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

# Convolution Operators and Filters – Multiple Filters

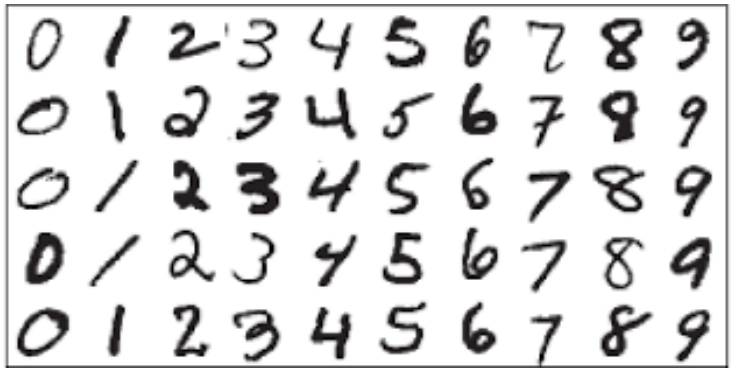
- Multiple filters can be applied to detect different features



# What is the Link to Deep Learning?

- For many years, researchers in Computer Vision created filters manually
  - Some variants of the basic edge detectors work better than others for different tasks

- Key insight: could replace hand-crafted filter values with parameters in a neural net that can be learned with Backprop
  - LeCun started developing these ideas in the 1990s, and named them Convolutional Networks
  - Exceeded previous state-of-the-art performance in tasks such as recognising handwritten digits, **without** needing manual feature engineering
  - See https://web.archive.org/web/20240129000543/http://yann.lecun.com/exdb/mnist/



# Convolutions – Filter Size and Padding

- Filter does not have to be 3×3
  - Can be other sizes such as 5×5, 7×7, 9×9
  - Filter size is denoted $f$ (almost always odd)
- Without padding, output ends up smaller than input
  - If input is $n_W \times n_H$, output will be $(n_W - f + 1) \times (n_H - f + 1)$
- To avoid this, can **pad** the input with 0s all round
  - To keep the input the same size as the output, have padding $p = (f-1)/2$
- Convolutions can be Valid or Same:
  - Valid: no padding, output width is $(n_W - f + 1)$
  - Same: pad by amount $p = (f-1)/2$
  - (Obscure terminology coming from Matlab)

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 4 | 3 | 2 | 0 |
| 1 | 1 | 2 | 4 | 5 | 4 | 1 |
| 3 | 0 | 0 | 1 | 2 | 3 | 4 |
| 4 | 5 | 2 | 1 | 1 | 2 | 3 |
| 4 | 3 | 2 | 0 | 2 | 1 | 1 |
| 4 | 5 | 4 | 1 | 2 | 4 | 5 |
| 3 | 1 | 4 | 2 | 1 | 3 | 5 |

# Convolutions – Padding Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2 | 1 | 3 | 4 | 3 | 2 | 0 | 0 |
| 0 | 1 | 1 | 2 | 4 | 5 | 4 | 1 | 0 |
| 0 | 3 | 0 | 0 | 1 | 2 | 3 | 4 | 0 |
| 0 | 4 | 5 | 2 | 1 | 1 | 2 | 3 | 0 |
| 0 | 4 | 3 | 2 | 0 | 2 | 1 | 1 | 0 |
| 0 | 4 | 5 | 4 | 1 | 2 | 4 | 5 | 0 |
| 0 | 3 | 1 | 4 | 2 | 1 | 3 | 5 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

=

| | | | | | | |
|---|---|---|---|---|---|---|
| -2 | -2 | -6 | -3 | 2 | 7 | 6 |
| -2 | 1 | -7 | -5 | 0 | 5 | 9 |
| -6 | 4 | 0 | -4 | -3 | 0 | 9 |
| -8 | 7 | 6 | -1 | -4 | -3 | 6 |
| -13 | 4 | 11 | 3 | -5 | -4 | 7 |
| -9 | 1 | 6 | 5 | -5 | -6 | 8 |
| -6 | -1 | 3 | 5 | -4 | -7 | 7 |

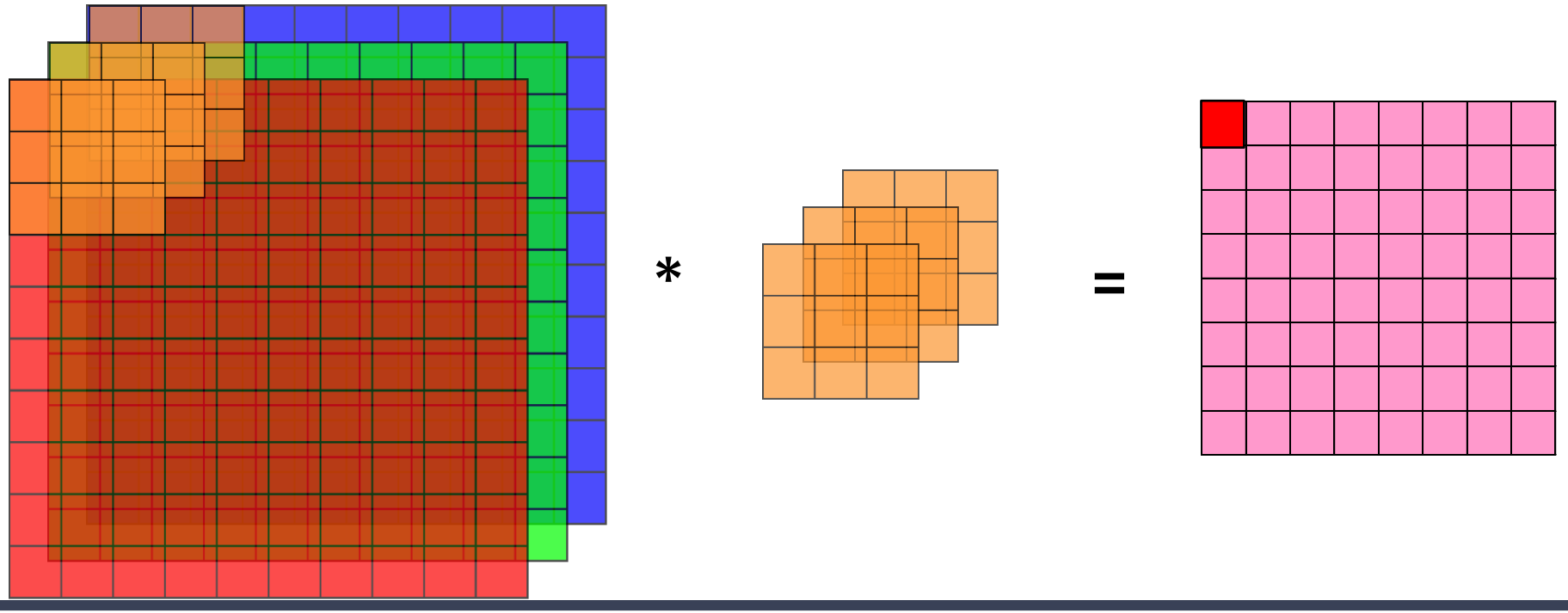
# Convolutions – Stride and Output Size

- Don't have to move filter in steps of 1
  - Can define a different stride
  - E.g. Stride $s$=2 means you move the filter 2 steps across or down each time, rather than 1 step
  - If $s$=2, output feature map will be about half the size it would be if $s$=1
- At edges, only apply convolution if the filter fits fully within input
- For an input is $n_W \times n_H$, with stride $s$ and padding $p$, output size is:

$$n_H^{[out]} = \left\lfloor \frac{n_H + 2p - f}{s} + 1 \right\rfloor \qquad n_W^{[out]} = \left\lfloor \frac{n_W + 2p - f}{s} + 1 \right\rfloor$$

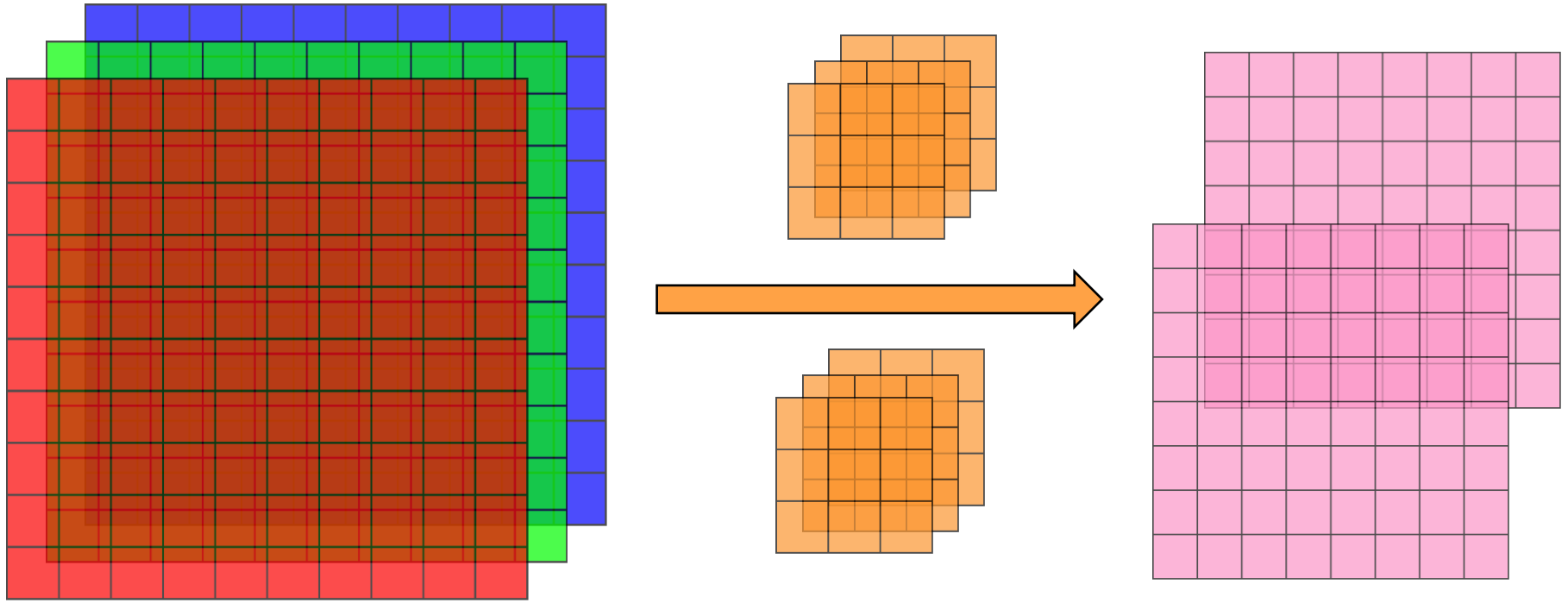
# Convolutions on Multi-Channel Inputs

- For multi-channel inputs, need a filter with the same number of channels, e.g. 3 if we have image with Red-Green-Blue channels
  - E.g. if input size is $n_W \times n_H \times n_C$, filter must be $f \times f \times n_C$
  - A single convolution calculation sums all multiplications from one filter, so we get a scalar output from it, and a 2D output feature map overall



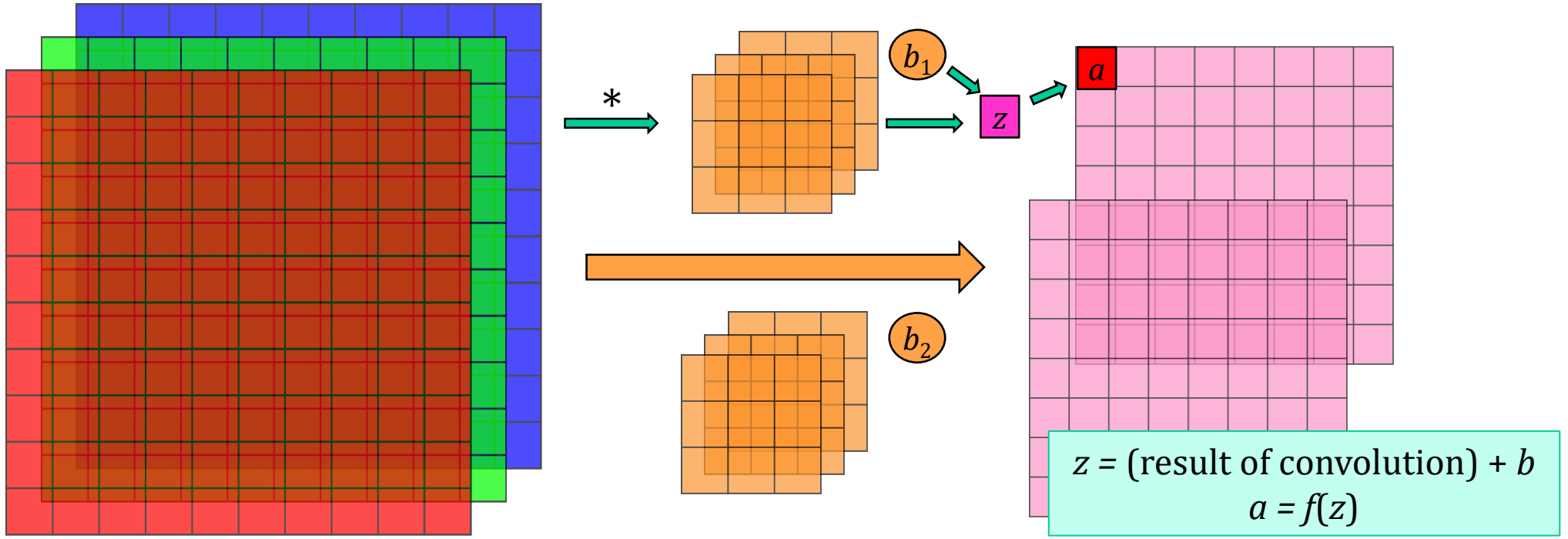
# Multiple Feature Detectors

- Need to detect more than one feature:
  - We always have multiple filters in 1 layer: 6 – 512 or even more
  - Each filter has independent weights and they are applied in parallel
  - Each filter creates one channel in the output feature map



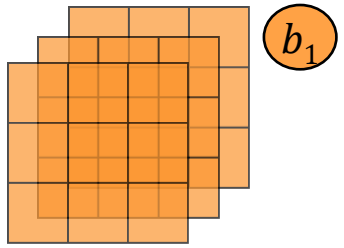
# One Layer of a CNN: Forward-Propagation

- The parameters of a filter act like weights in a standard NN:
  - Applying 1 filter to a 1 patch of an input is a sum of inputs*weights
  - Then add one bias for each filter
  - Put the result though an activation function, usually ReLU
  - Result is activation of for that value of the output feature map



# One Layer of a CNN: Backprop

- The algorithms for backpropagation with gradient descent work for convolutional layers
  - In a convolutional layer, learn values for each parameter of each filter, which are handled as weight values (W), and one bias (b) term for each filter
  - For classification, use cross-entropy loss function as usual; different loss functions for different tasks, e.g. bounding box regression
  - Backprop is used to calculate gradients for W and b terms, with modifications to account for the convolution calculations
  - W and b terms updated using gradient descent

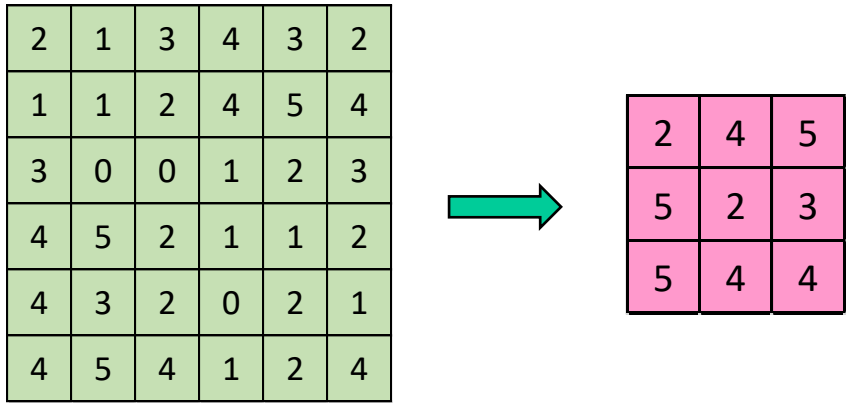


# Layers of a CNN: Pooling and Fully Connected

- The output of one convolutional layer can be used as the input into another one
- Conv layers can also be followed by **pooling**, which is an operation that down-samples its inputs (next slide)
  - A secondary operation that can be optionally performed after convolution
  - Pooling operation has no weights to be learned, so does not add parameters
- After all conv layers, usually have **fully connected layers**
  - The last feature map is reshaped into a single column of nodes
  - Can have multiple fully connected hidden layers
  - These feed into the final layer, often a softmax layer for multi-class classification

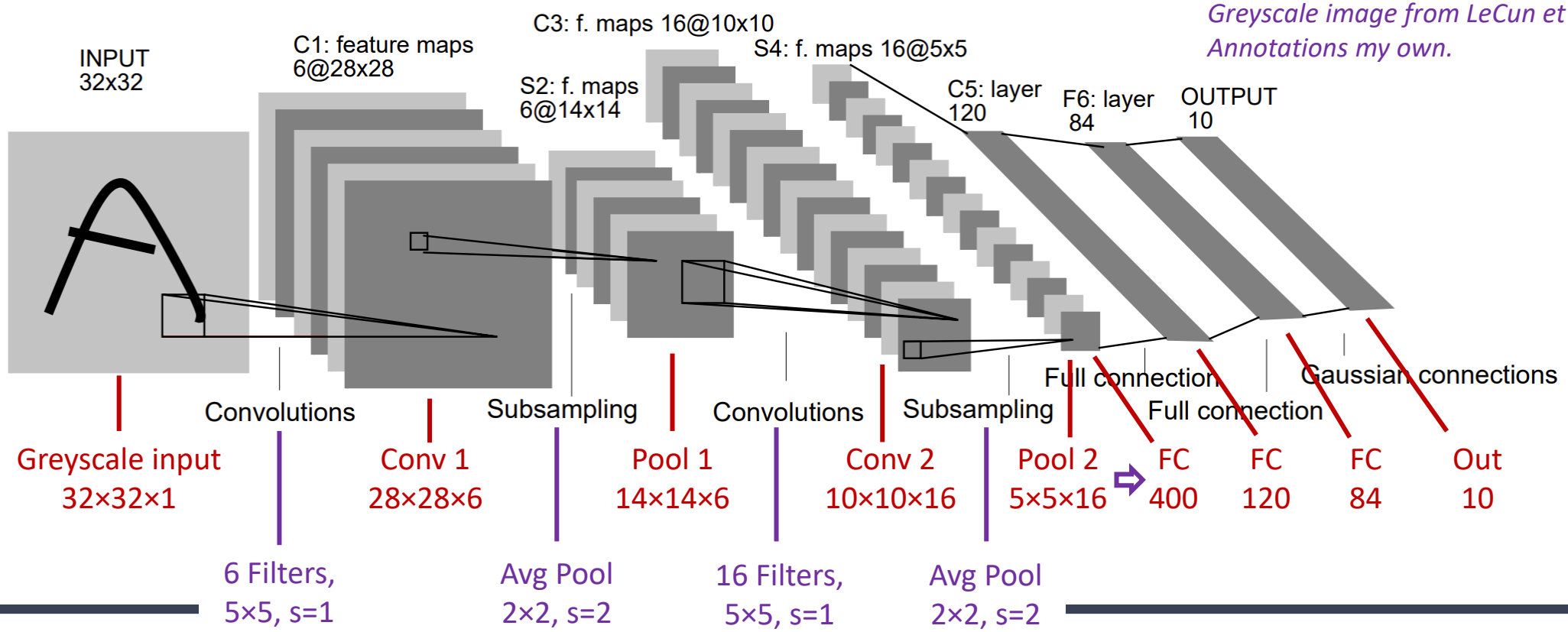
# Layers of a CNN: Pooling

- A convolutional stage is often followed by a **pooling stage** that summarises its inputs and reduces dimensionality
  - These typically get the **mean** (Avg-Pool) or **maximum** (Max-Pool) of a block of inputs, applied to each channel separately
  - Most common: Max-Pool of size 2x2, non-overlapping (stride=2)
- No learnable parameters, just overall settings for the pooling stage
  - However, need to be accounted for during backprop, as forward propagation is affected by the pooling operations



# Layers of a CNN: LeNet-5 Example

- In diagrams, often represent CNNs by size of feature maps in each layer
  - Annotations added to indicate filter size, number of filters, stride and padding
  - If a conv layer is followed by pooling, they are both given the same layer number



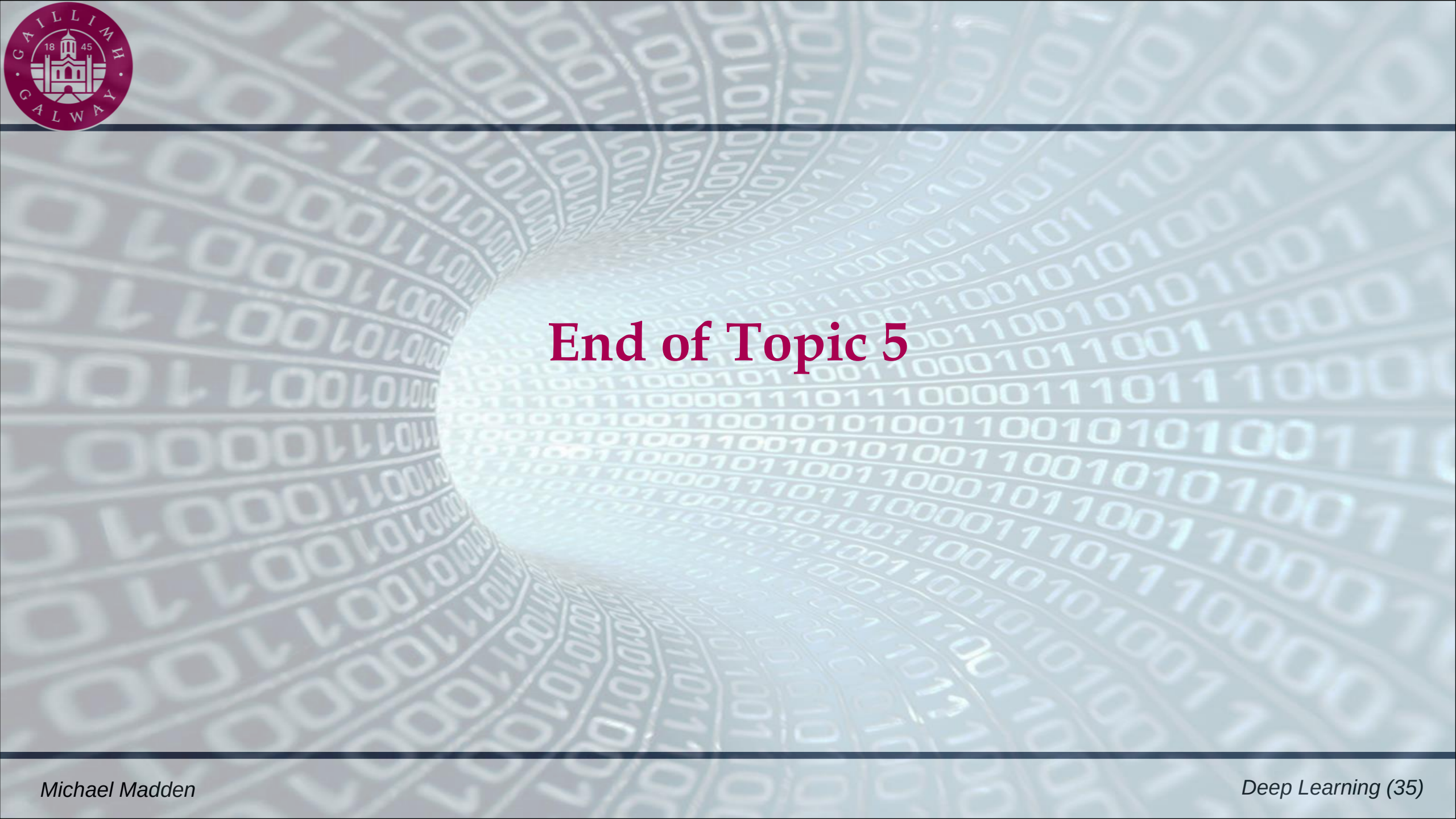
# Convolutional Networks – Some Notes

- Convnets covered here are specifically designed for image processing and related tasks
  - Not just image classification, but object detection within images, segmentation of images, tracking objects in videos, medical image analysis, etc.
  - Their structure is inspired by past work in computer vision, and encoding strong assumptions/biases, e.g. how groups of neighbouring pixels define features
  - Have led to major increases in performance in computer vision
- A conv layer has far fewer weights than a fully-connected layer to handle same number of inputs
  - Improves feasibility of learning
  - Since size of layer is determined by number and size of filters, it is independent of the size of the image itself
  - Makes it possible to work with higher resolution images


# Review of Learning Objectives

Now that you have completed Topics 4 and 5, you should be able to…

- Define key concepts related to Deep Learning, and discuss major advances relative to shallow neural networks
- Explain and implement approaches to handling unstructured data and multi-class classification
- Explain and implement regularization methods, and algorithms including Mini-Batch Gradient Descent, Momentum, RMSprop and Adam
- Explain the principles of operation of Convolutional Networks
- Correctly use these features within ML libraries, including selecting hyperparameters



# End of Topic 5