CS6482

Deep Reinforcement Learning

C: CNNs Basics.

J.J. Collins

Dept. of CSIS

University of Limerick

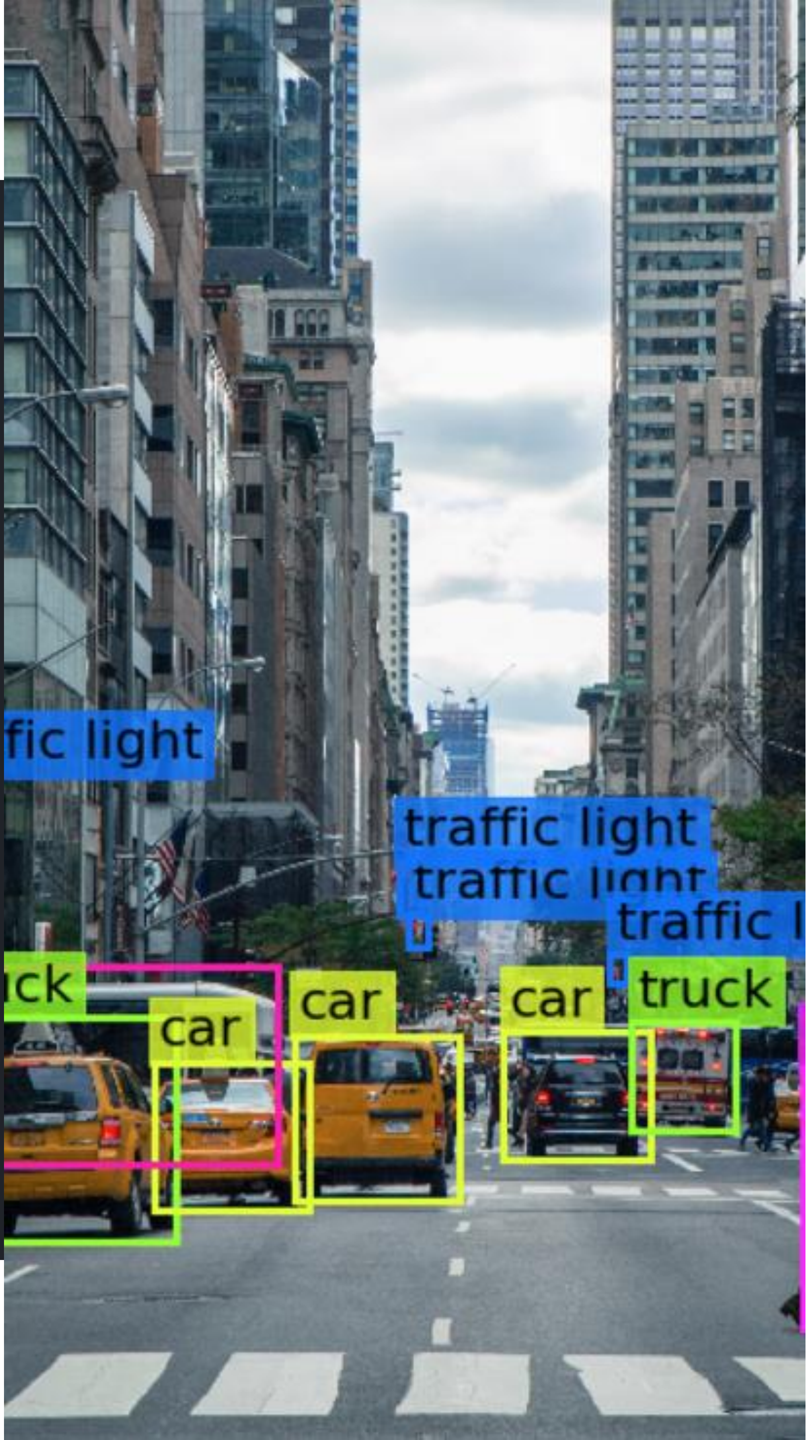
Intro to CNNs

Objectives

A gentle introduction to Convolutional Neural Networks

Some content based on:

- ❑ **Ming Lee**, University of Waterloo
https://cs.uwaterloo.ca/~mli/cs898-2017.htm
- ❑ **Yunzhe Xue**, University of Stanford
http://cs231n.github.io/convolutional-networks/

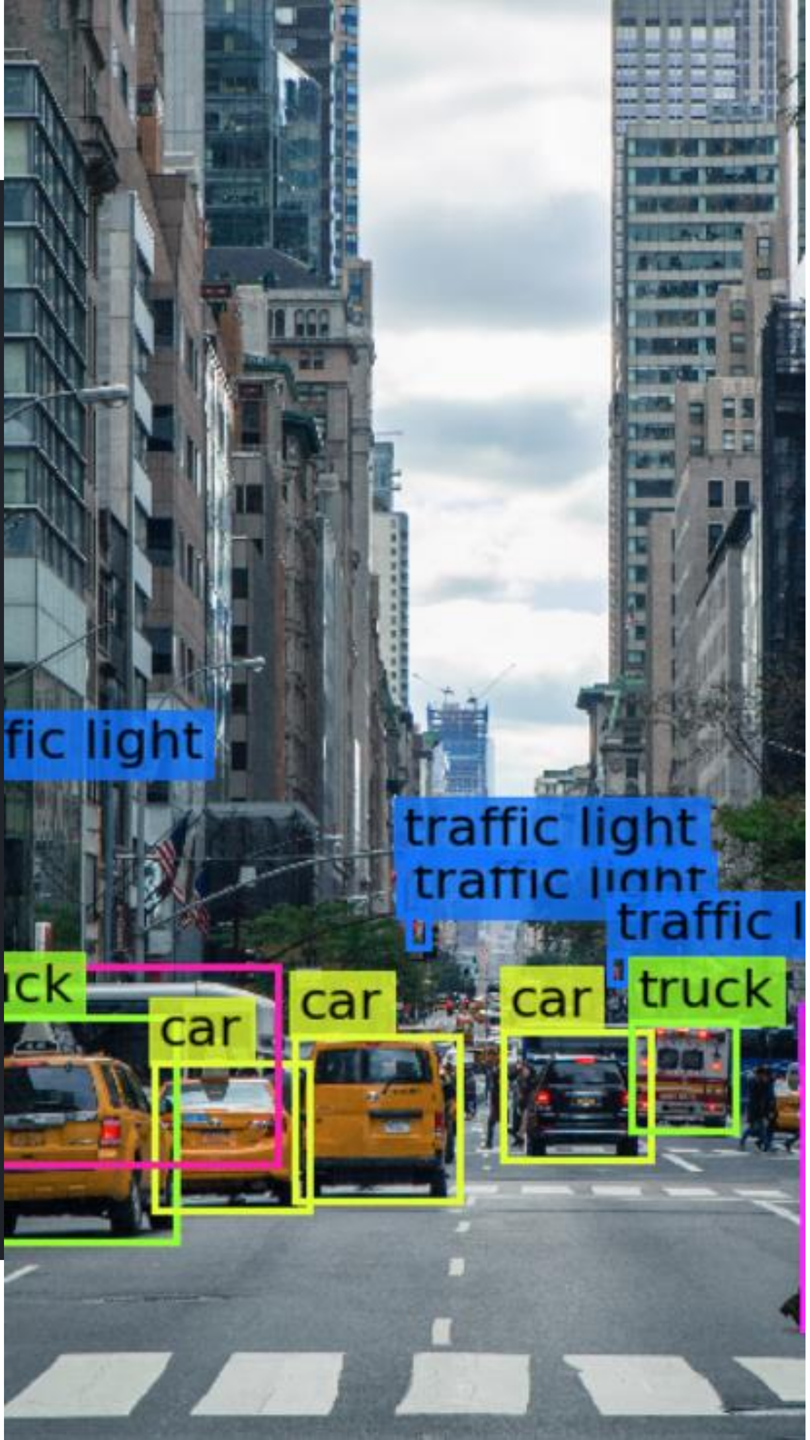


Objectives

- ❑ Part 1: Convolution
- ❑ Part 2: A little History
- ❑ Part 3: Implementing LeNet-5 using RELUs
- ❑ Part 4: Loss Functions and Optimisers
- ❑ Part 5: Dropout.

Part 1

Convolution

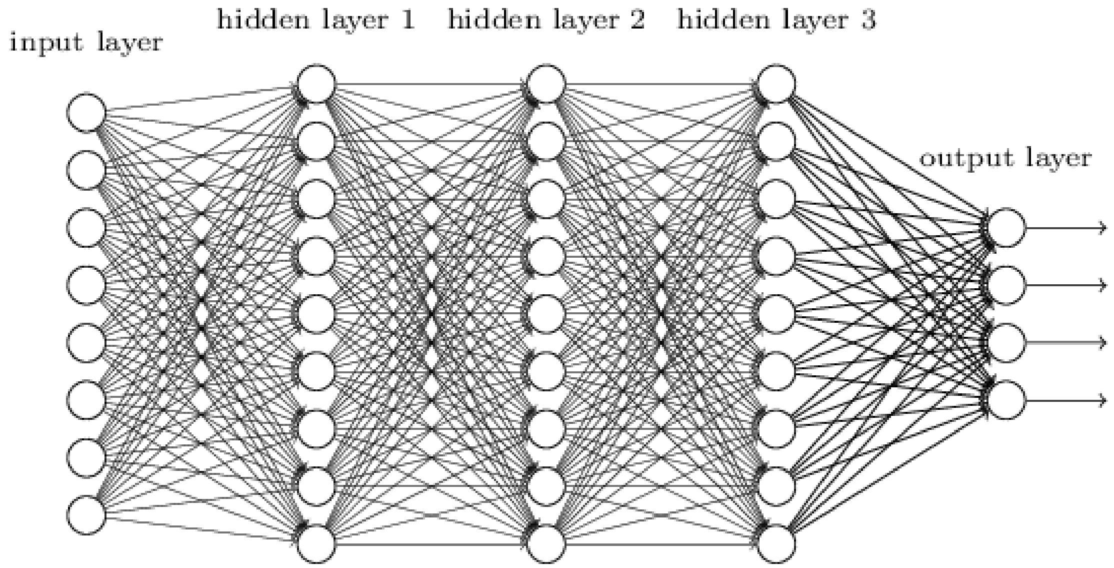


So Many FIlters

- ❑Smoothing
- ❑Sharpening
- ❑Edge Detectors
- ❑Sobel, Canny, LoG
- ❑Texture
- ❑Histogram of Orientated Gradients (HoGs)
- ❑Wavelets

Intro to CNNs

- From the fully connected model:
- Premise is that lower HLs learn kernels/filters to extract primitive features that are used to create higher order abstractions in subsequent HLS
  - Are the filters spatially agnostic?
  - Are the filters being Duplicated?



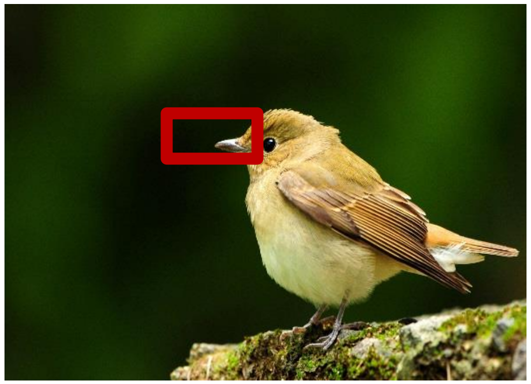
- It is easier to learn a small model.
  - As long as accuracy and generalization is not sacrificed

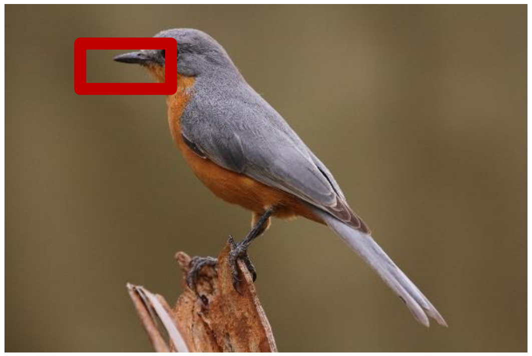
Intro to CNNs

- In the old days, the researcher would select the kernels
- CNNs → Learn convolution filters for Regions of Interest (RoIs)
- Such as a beak if identifying birds in images
- And filters for other primitives such as eye(s), leg(s), etc

Intro to CNNs

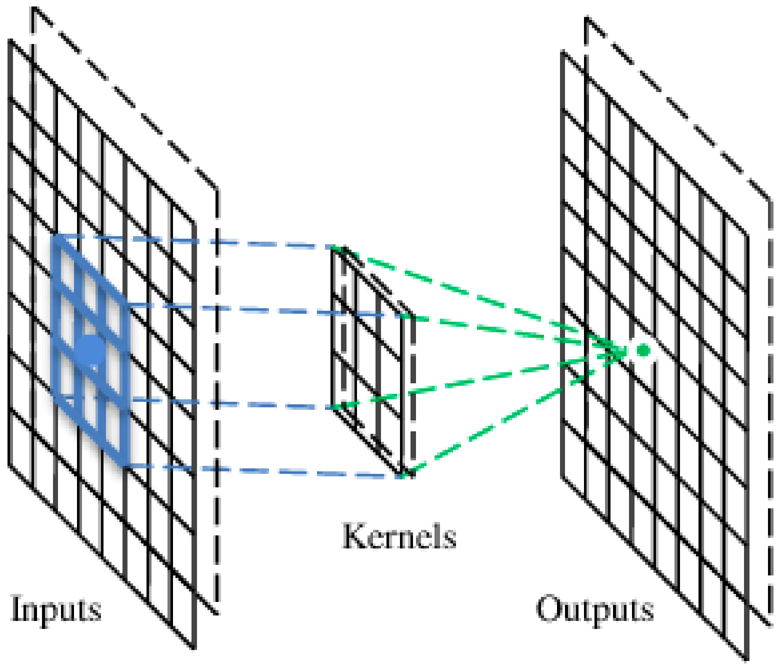
- RoIs can occur in any part of the image
- Learn a filter and run it over the entire image
- Where the filter is the weights of a neuron
- Weights are the same for the filters below





Intro to CNNs

- A CNN is a neural network with some convolutional layers
  - And others
- A convolutional layer has a number of filters that perform convolution



Intro to CNNs

- An example of Convolution
- Input is a 6x6 image
- Filter is 3x3

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

1	-1	-1
-1	1	-1
-1	-1	1

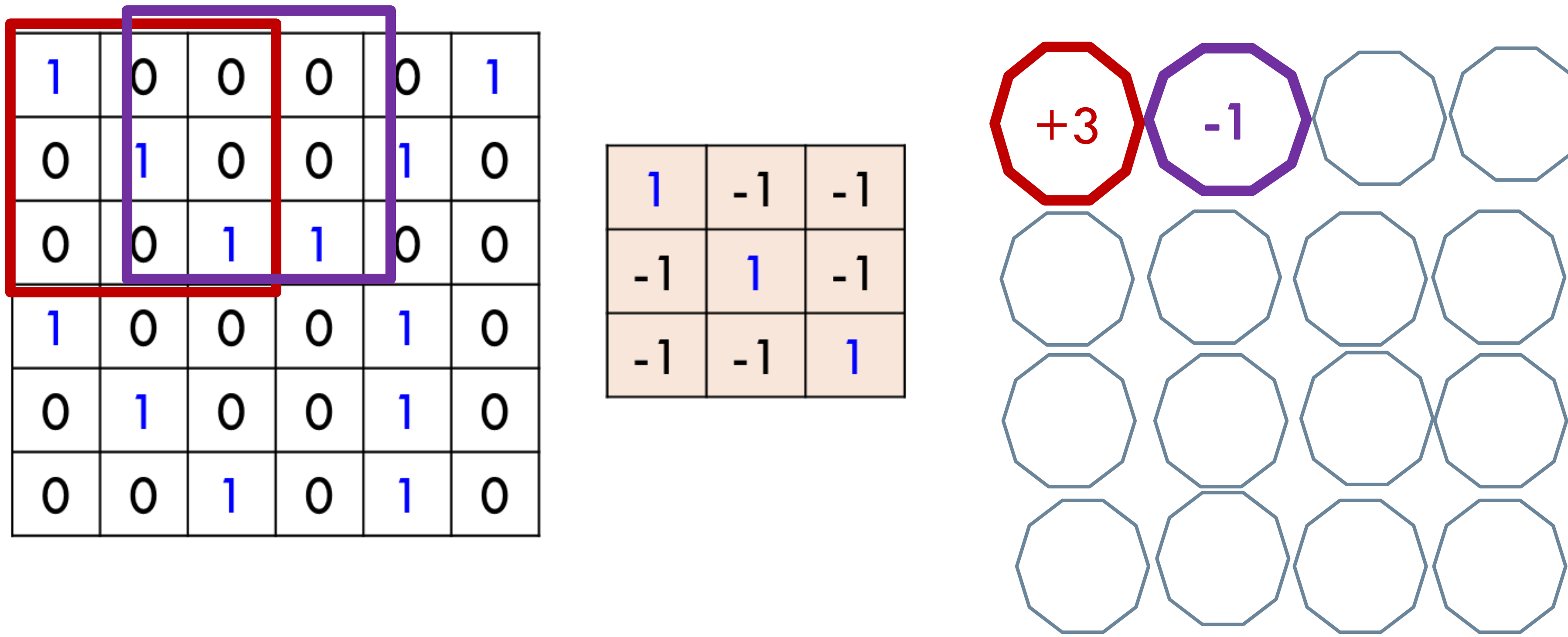
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Intro to CNNs

- Stride = 1



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

1	-1	-1
-1	1	-1
-1	-1	1

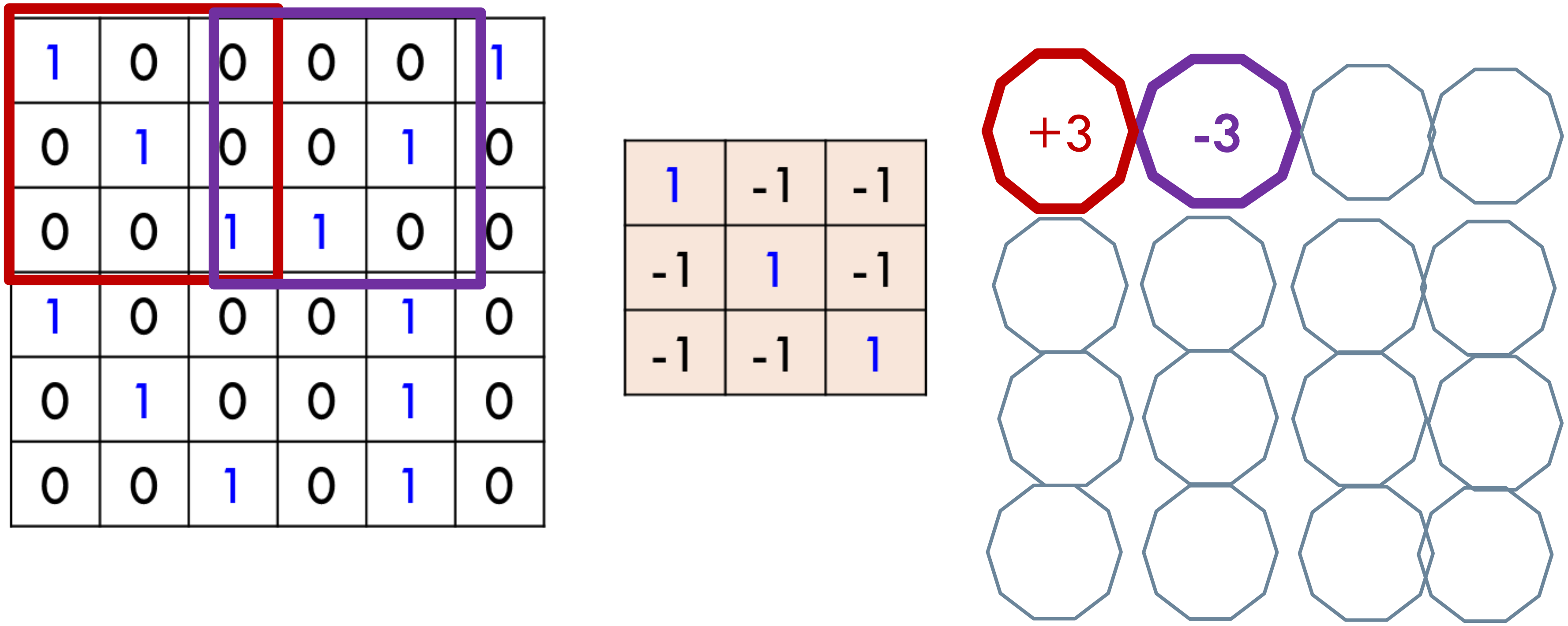
- Input

Filter

Dot Product

Intro to CNNs

- Stride = 2



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

1	-1	-1
-1	1	-1
-1	-1	1

- Input

Filter

Dot Product

Intro to CNNs

- Stride = 1

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

1	-1	-1
-1	1	-1
-1	-1	1

3	-1	-3	-1
-3	1	0	3
-3	-3	0	1
3	-2	2	1

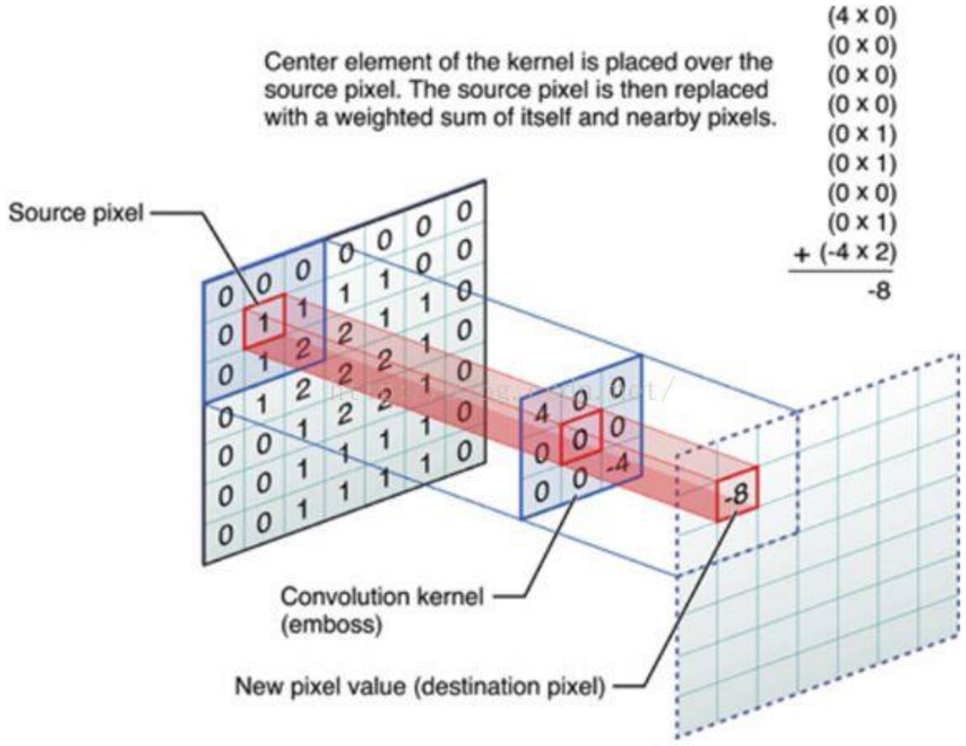
- 6x6 Input

3x3 Filter

4x4 Dot Product

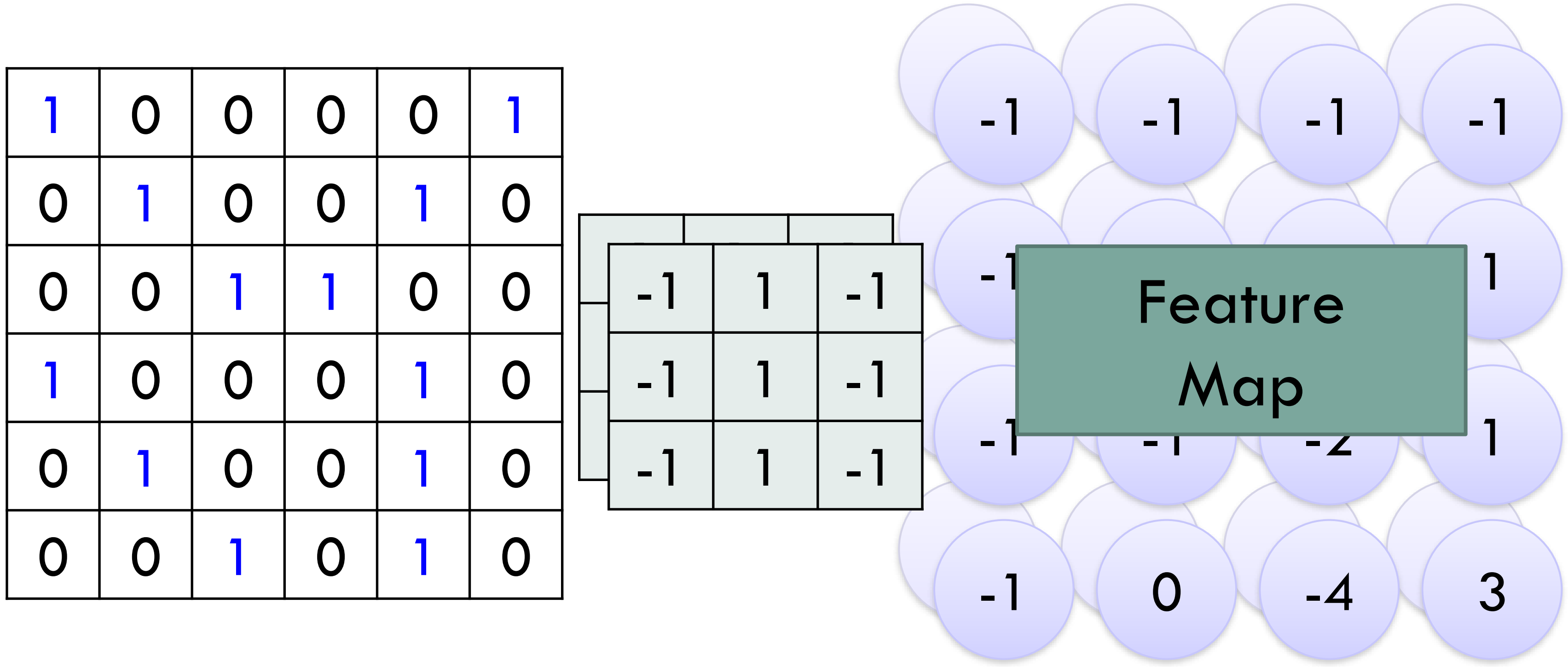
Intro to CNNs

- Padding on the input volume with zeros in such way that the convolutional layer does not alter the spatial dimensions of the input


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Convolution

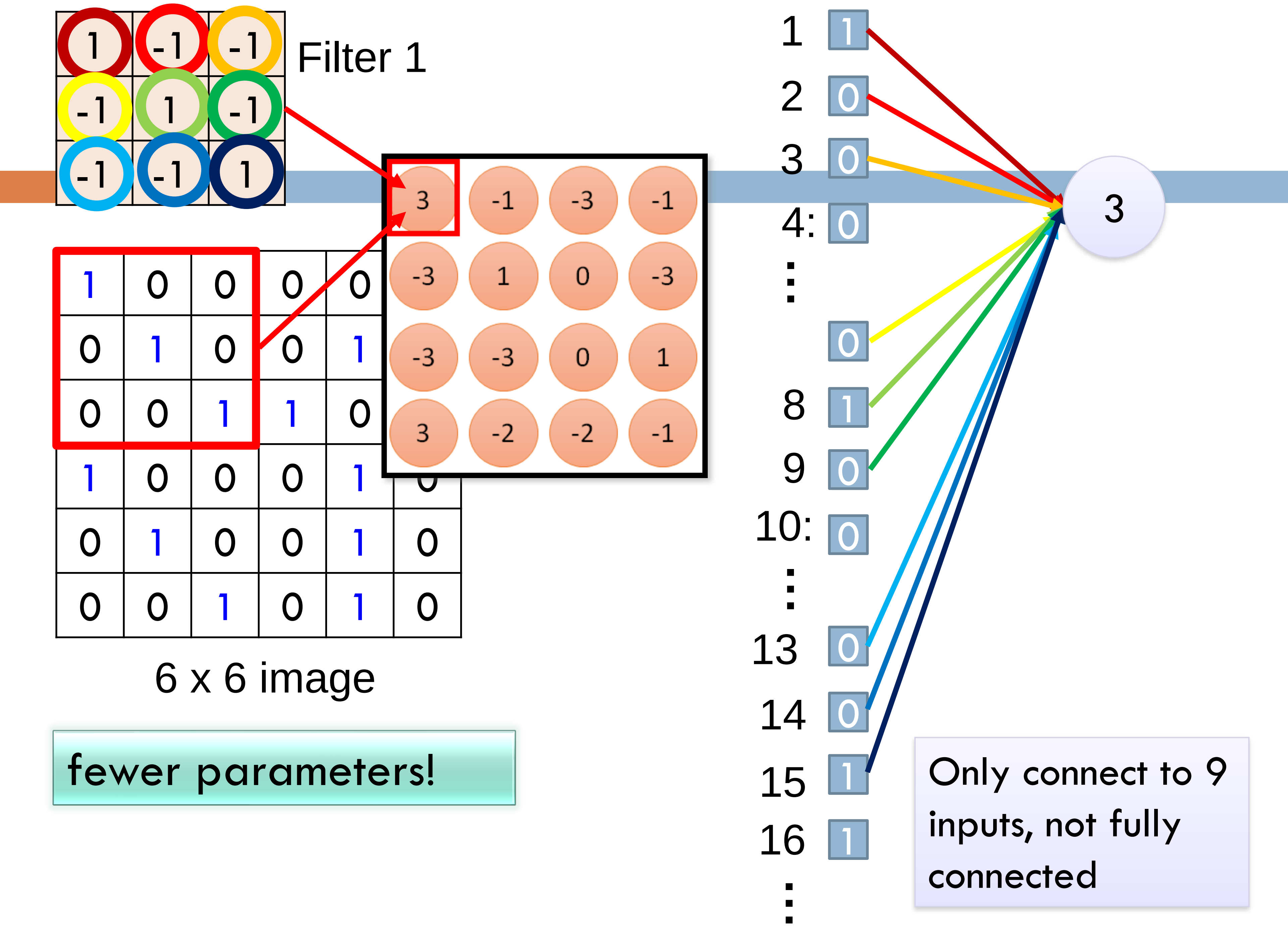
- Repeat for each filter
- Stride = 1



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

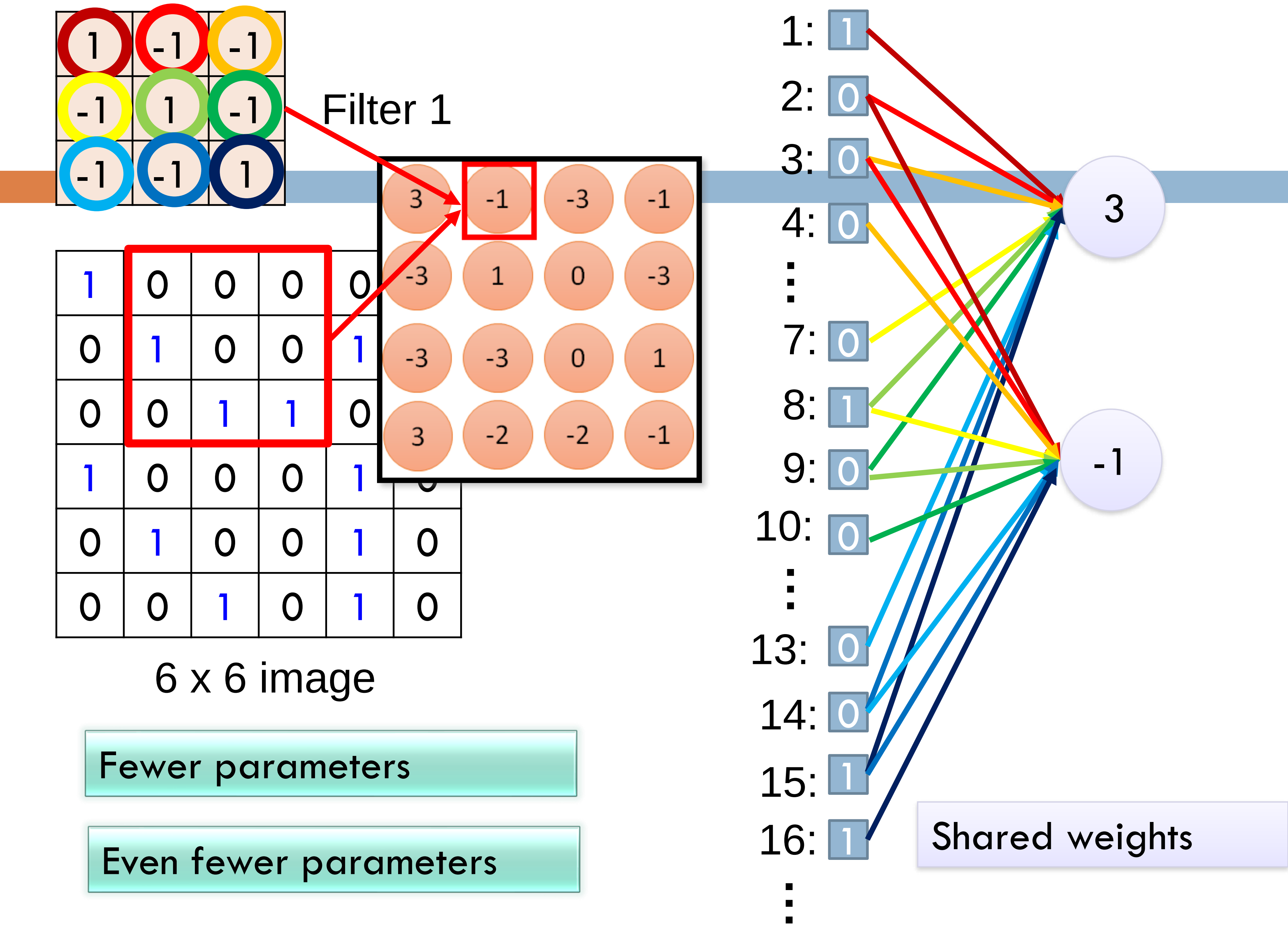
-1	1	-1
-1	1	-1
-1	1	-1

Two 4 x 4 FMs
Forming 2 x 4 x 4 matrix



fewer parameters!

Only connect to 9 inputs, not fully connected



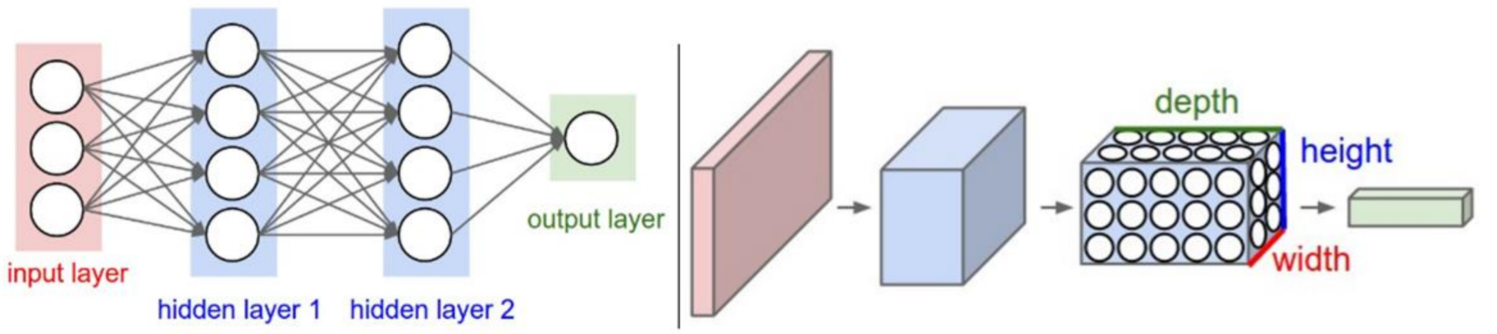
Fewer parameters

Even fewer parameters

Shared weights

Intro to CNNs

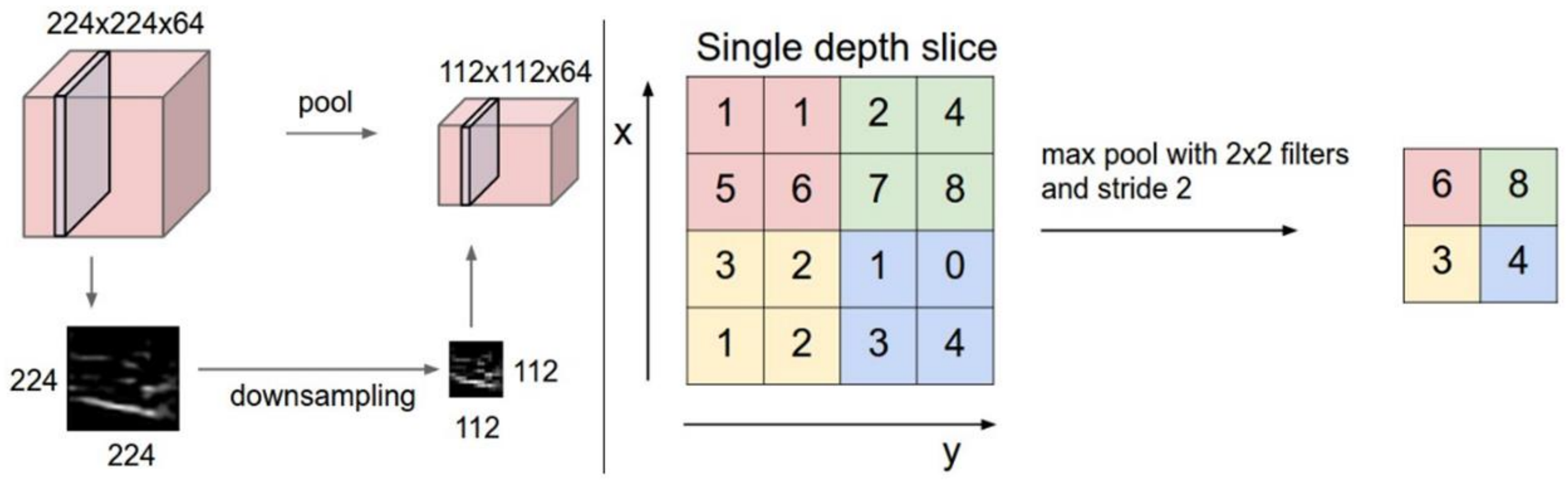
- Dense neural network and Convolutional neural network



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Intro to CNNs

- Pooling Layer → Down sampling



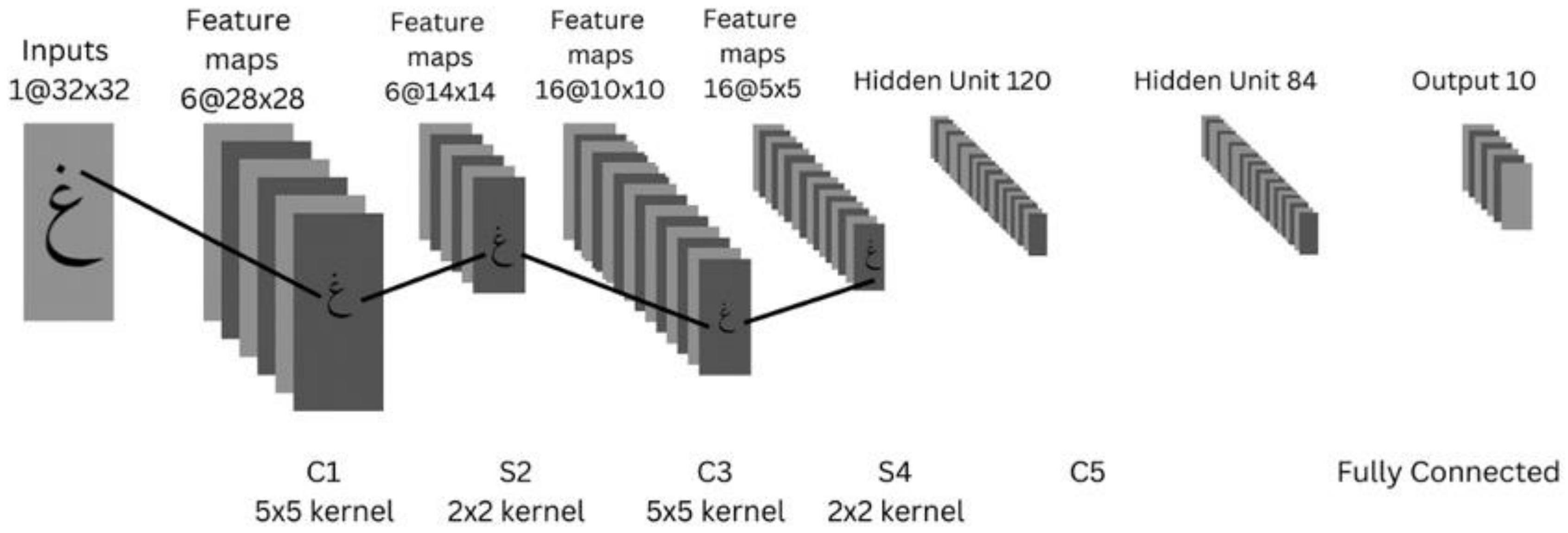
Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Part 2

A Little History

Intro to CNNs

- LeNet-5 for MNIST



https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=726791

https://www.researchgate.net/publication/371158423/figure/fig5/AS:11431281171418412@1688152250976/LeNet-5-Architecture-25.ppm

LeNet-5: How Many Parameters

- Convolutional Layers (C1, C3, C5):
  - C1: (5x5 kernel x 1 input channel x 6 output channels) + 6 biases = 156 parameters
  - C3: (5x5 kernel x 6 input channels x 16 output channels) + 16 biases = 2416 parameters
  - C5: (5x5 kernel x 16 input channels x 120 output channels) + 120 biases = 48120 parameters
- Subsampling Layers (S2, S4): These are pooling layers that reduce the spatial dimensions of the feature maps. They don't have any learnable parameters
- Fully Connected Layers (F6, Output): These layers connect every neuron in the previous layer to every neuron in the current layer.
  - F6: (120 input neurons x 84 output neurons) + 84 biases = 10164 parameters
  - Output: (84 input neurons x 10 output neurons) + 10 biases = 850 parameters
- Total Parameters
- 156 + 2416 + 48120 + 10164 + 850 = 60,000 params.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Intro to CNNs

LeNet-5.

- The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990’s.
- Reading zip codes, digits, etc.

AlexNet.

- The first work that popularized CNNs in Computer Vision was the AlexNet by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.
- Submitted to the 2012 ImageNet ILSVRC Challenge ImageNet ILSVRCin
  - top 5 error of 16% compared to runner-up with 26% error.
- The Network had a very similar architecture to LeNet,
- But was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).

Case studies

GoogLeNet.

- The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. (Google).
- Includes an *Inception Module* that dramatically reduced the number of parameters in the network
  - (4M, compared to AlexNet with 60M).
  - uses Average Pooling instead of Fully Connected layers at the top of the CNN
    - Eliminates a large amount of parameters.
    - There are also several followup versions to the GoogLeNet, most recently InceptionV4.

Case studies

- **VGGNet.**
- The runner-up in ILSVRC 2014 from Simonyan and Andrew Zisserman.
- Showed that the depth of the network is a critical component for good performance.
- Their final best network contains 16 CONV/FC layers based on an architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- Their pretrained model is available for plug and play use in Caffe.
- VGGNet is more expensive to evaluate and uses a lot more memory and parameters (140M).
- Most of these parameters are in the first fully connected layer, and it was shown that these FC layers can be removed with no reduction in accuracy.

Case studies

ResNet.

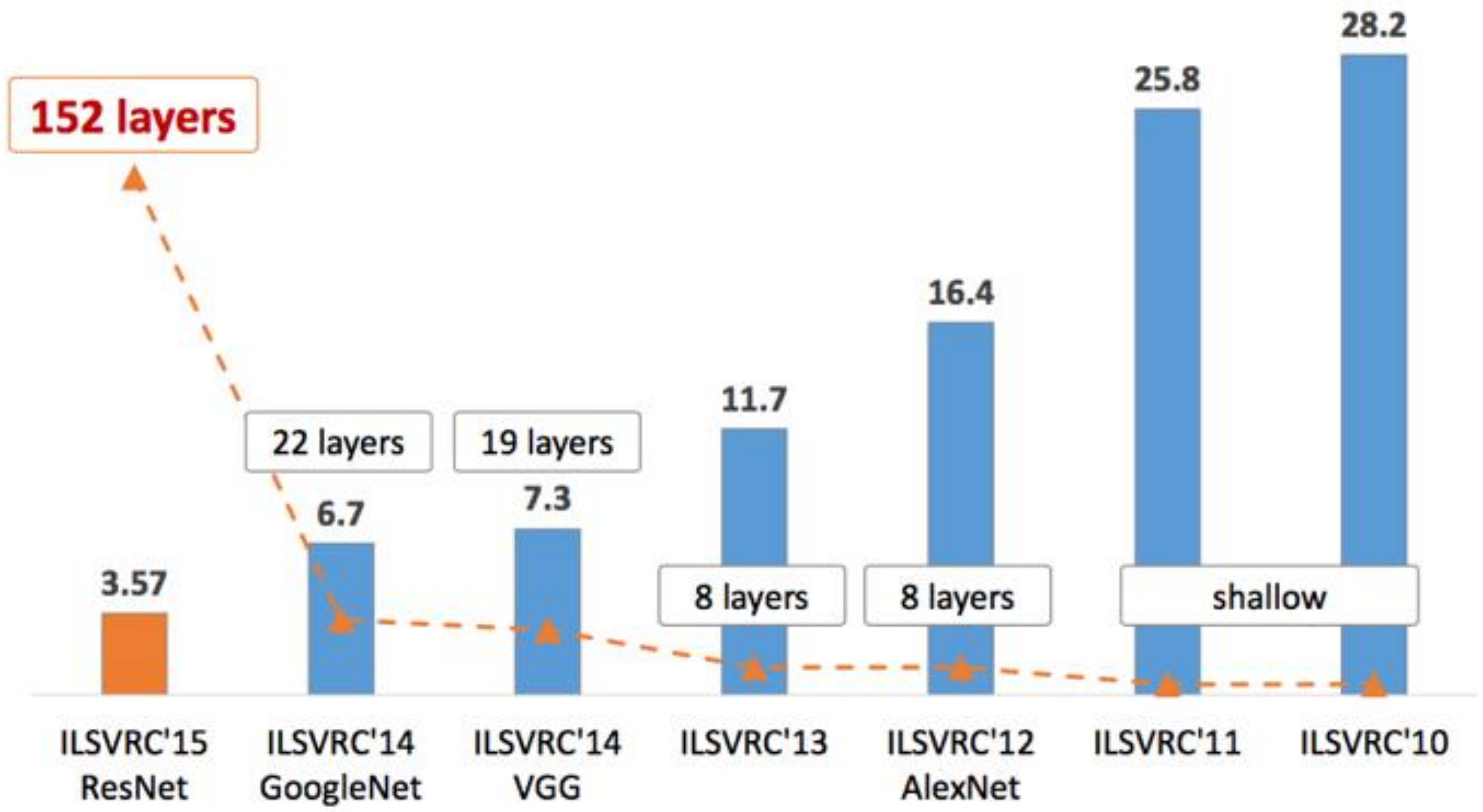
- Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015.
- Uses a *skip connections* and batch normalisation.
- The architecture is also missing fully connected layers at the end of the network.
- ResNets are state of the art in Convolutional Neural Networks.

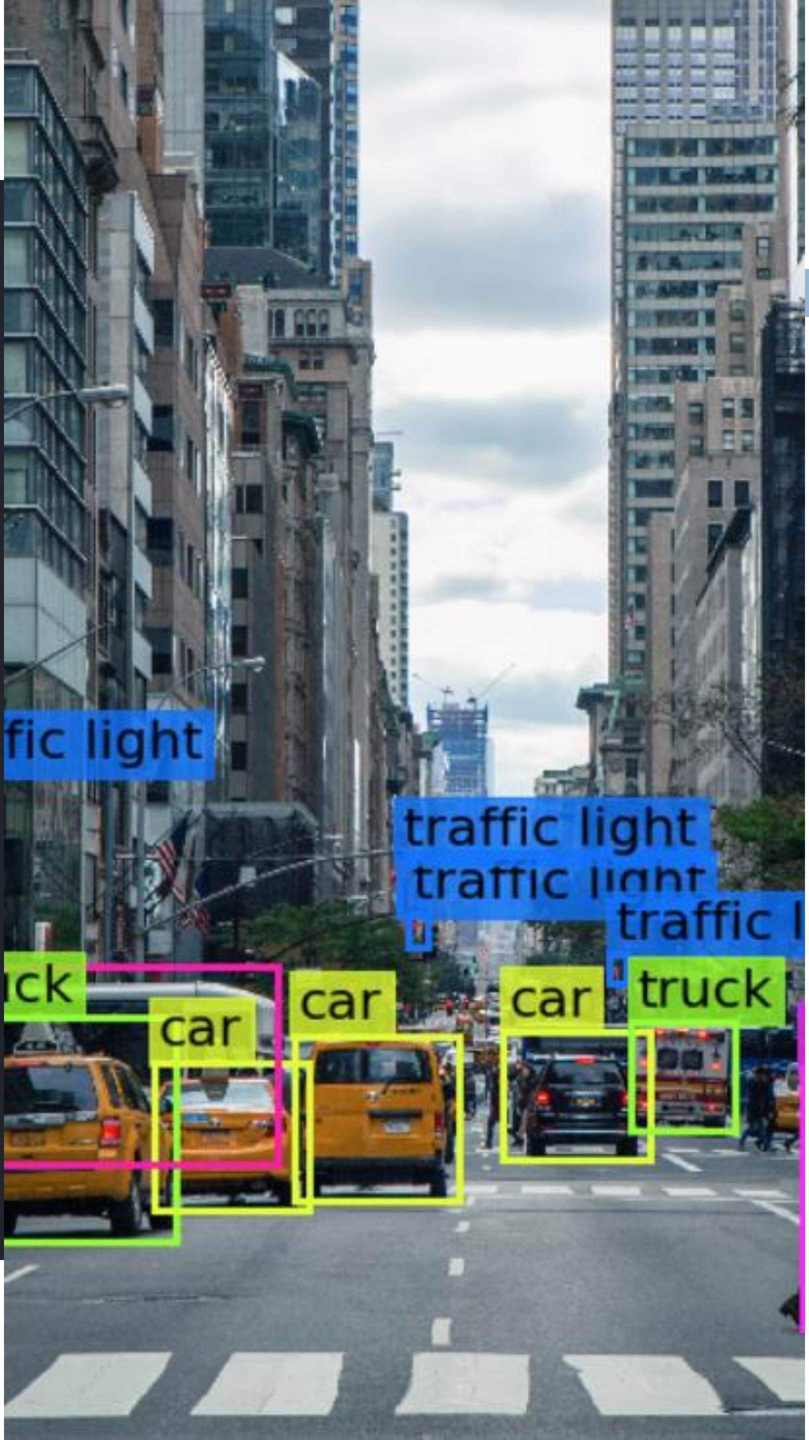
Case studies

ResNet.

- Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015.
- Uses a *skip connections* and batch normalisation.
- The architecture is also missing fully connected layers at the end of the network.
- ResNets are state of the art in Convolutional Neural Networks.

- S. Das 2017. [medium.com]()





SUGGESTED READING

- ❑ Any text on Deep Learning with a chapter on CNNs
- ❑ Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Ed. 2019. O'Reilly.
- ❑ Eugene Charniak. Introduction to Deep Learning. 2018. The MIT Press.

Part 3

Implementing LeNet-5 using RELUs

BASED ON:

- Gulli, Kapoor, Pal. Deep Learning with TensorFlow 2 and Keras. Packt Publishing. 2019
- Aurelien Geron. Hands On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Ed. O'Reilly. 2019

The LeNet CNN Architecture

- **Imports and hyperparameters**

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers

# network and training
EPOCHS = 50
BATCH_SIZE = 128
VERBOSE = 1
OPTIMIZER = tf.keras.optimizers.Adam()
VALIDATION_SPLIT=0.90

IMG_ROWS, IMG_COLS = 28, 28 # input image dimensions
INPUT_SHAPE = (IMG_ROWS, IMG_COLS, 1)
NB_CLASSES = 10  # number of outputs = number of digits
```

The LeNet CNN Architecture

- Then define the network

```
class LeNet:
    @staticmethod
    def build(input_shape, classes):
        model = models.Sequential()
```

- First stage is a convolutional filter with Rectified Linear Units (ReLUs)
  - Learns 20 filters each of size 5x5
  - Output dimension is the same as the input shape so it will be 28x28
- followed by MaxPooling

```
        # CONV => RELU => POOL
        model.add(layers.Convolution2D(20, (5, 5), activation='relu',
                        input_shape=input_shape))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

Saturation

- "More generally, deep neural networks suffer from unstable gradients; different layers may learn at widely different speeds … and one of the reasons DNNs were abandoned in the early 2000s".

  Geuron. Hands on Machine Learning with …., 2nd Ed. 2019. O'Reilly.

- It wasn't clear what caused the gradients to be unstable when training a DNN, but some light was shed in a 2010 paper by Xavier Glorot and Yoshua Bengio examining saturation and weight initialisation
- Saturation is an issue with both the sigmoid and tanh activation functions.
- Large values from $z = \mathbf{w} \cdot \mathbf{x} + b$ snap to 1.0 and small values snap to -1 or 0 for tanh and sigmoid.

Saturation

- These activation functions are only really sensitive to changes around their mid-point of their input, such as 0.5 for sigmoid.
- Once saturated, it becomes challenging for the learning algorithm – backpropagation, to adapt the weights to improve the model.
- "sigmoidal units saturate across most of their domain—they saturate to a high value when z is very positive, saturate to a low value when z is very negative, and are only strongly sensitive to their input when z is near 0"

Goodfellow, Bengio, and Corville. Deep Learning. The MIT Press. 2016.

Vanishing Gradient

- Layers deep in large networks using these nonlinear activation functions do not receive useful gradient information.
- Error is back propagated through the network and used to update the weights.
- The amount of error decreases with each additional layer through which it is propagated.
- This is called the vanishing gradient problem and prevents deep (multi-layered) networks from learning effectively.
- "Vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function"

Goodfellow, Bengio, and Corville. Deep Learning. The MIT Press. 2016.

ReLU

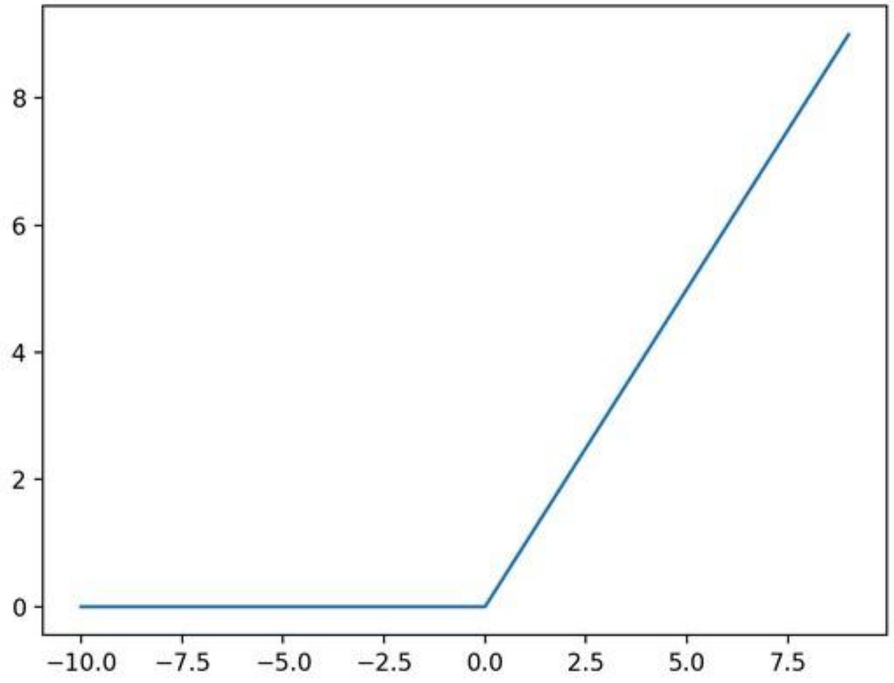
- In order to use stochastic gradient descent to train deep networks, an activation function is needed that
  - **Approximates a linear function**
    - For calculation of gradients,
  - **But is, in fact, a nonlinear function** allowing complex relationships in the data to be learned.
- "[another] major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units".

Goodfellow, Bengio, and Corville. Deep Learning. The MIT Press. 2016.

ReLU

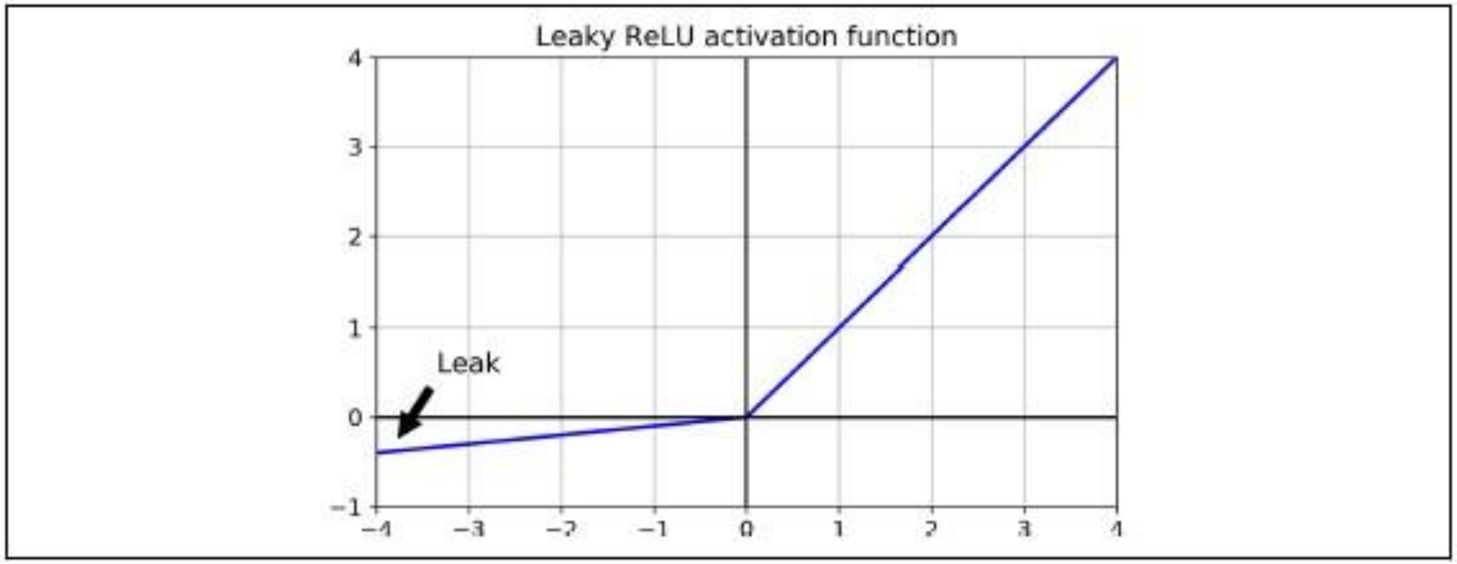
- Solution: ReLU
- "… the rectified linear function g(z) = max{0, z} is not differentiable at z = 0. This may seem like it invalidates g for use with a gradient-based learning algorithm. In practice, gradient descent still performs well enough for these models to be used for machine learning tasks".

Goodfellow, Bengio, and Corville. Deep Learning. The MIT Press. 2016.



ReLu

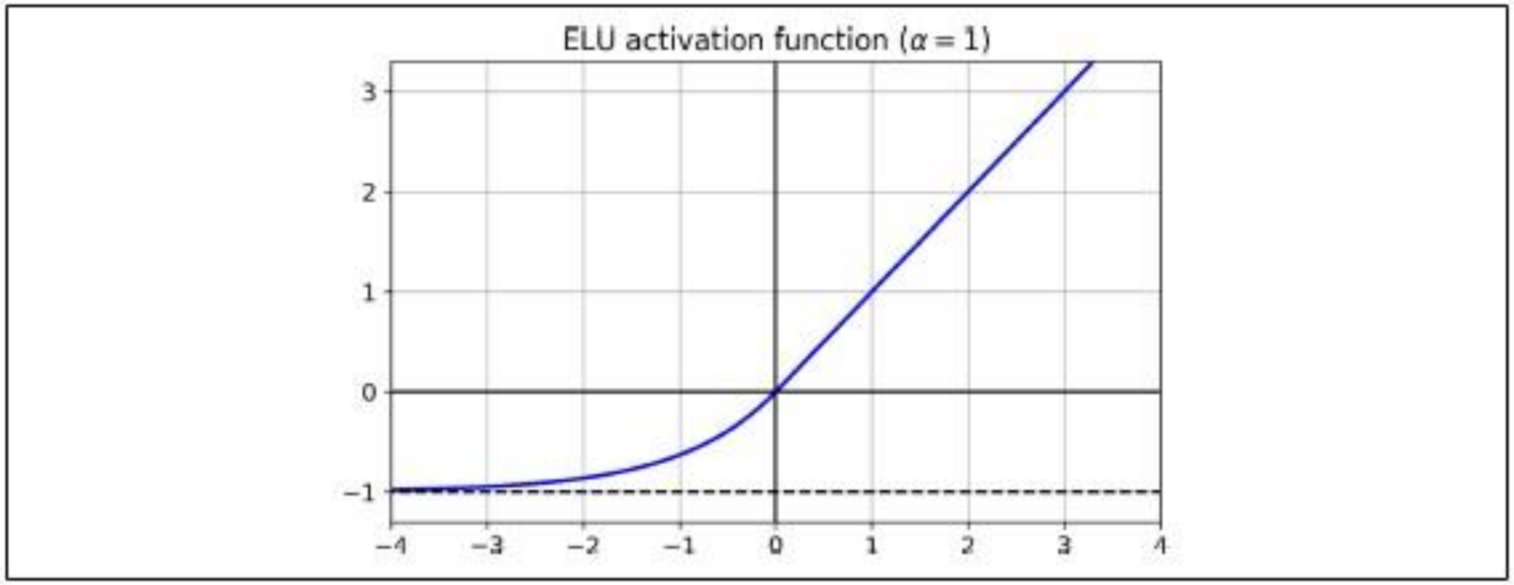
- Unfortunately, the ReLU activation function is not perfect. It suffers from a problem known as the *dying ReLUs*
  - during training, some neurons effectively "die," meaning they stop outputting anything other than 0.
- Use Leaky ReLU



ELU

- 2015 paper by Clevert et al. proposed a new activation function called the *Exponential Linear Unit* (ELU) that outperformed all the ReLU variants in the authors' experiments:

- $ELU_{\alpha}(z) = \begin{cases} \alpha(\exp(-z) - 1) & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$

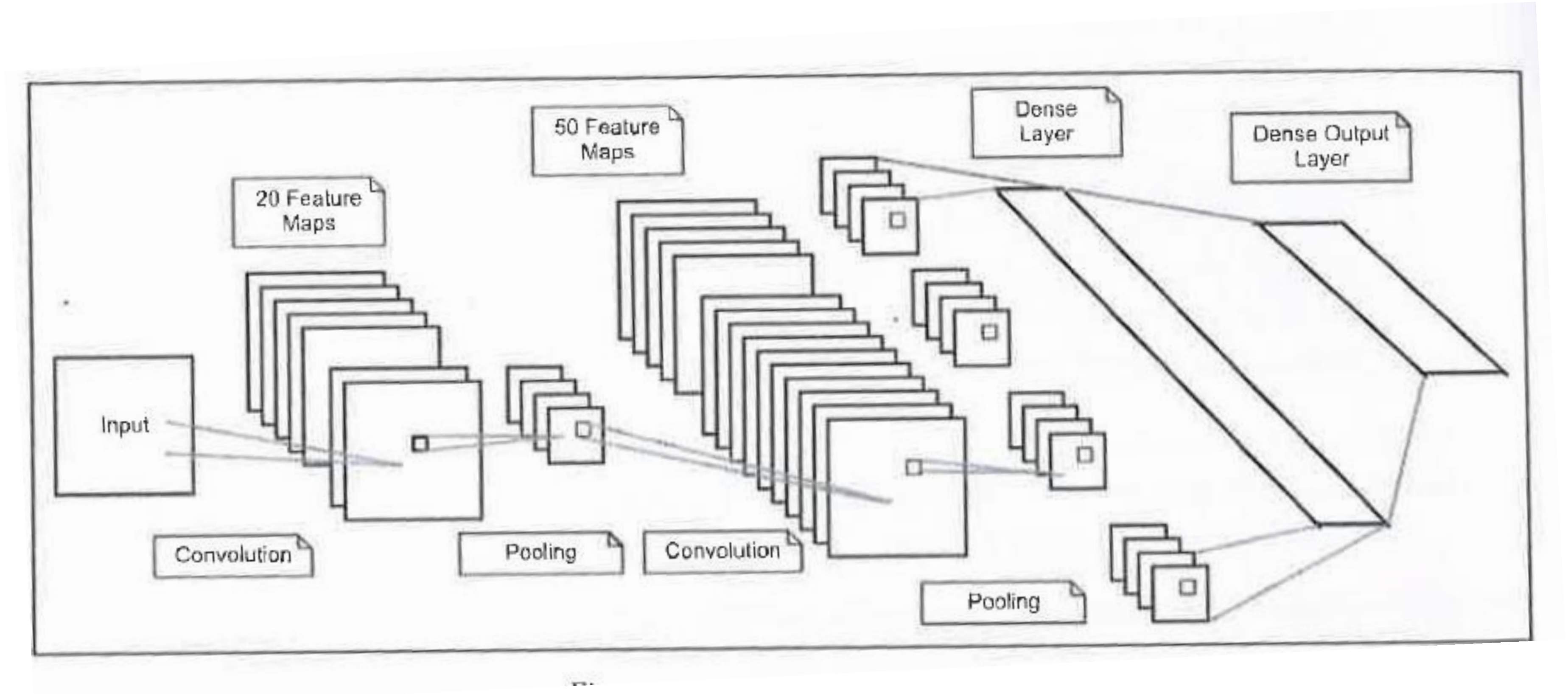


ELU

- It takes on negative values when $z < 0$, which allows the unit to have an average output closer to 0 and helps alleviate the vanishing gradients problem.
- The hyperparameter α defines the value that the ELU function approaches when z is a large negative number. It is usually set to 1, but you can tweak it like any other hyperparameter.
- It has a nonzero gradient for $z < 0$, which avoids the dead neurons problem.
- If α is equal to 1 then the function is smooth everywhere, including around $z = 0$, which helps speed up Gradient Descent since it does not bounce as much to the left and right of $z = 0$.
- 2017 paper by Klambauer et al introduces Scaled ELU (SELU)

Another LeNet CNN Architecture

- Another LeNet Architecture



- Adapted from Gulli et al (2019).

The LeNet CNN Architecture

- 2nd convolutional stage with ReLU activations
  - 50 filters in this stage
- Followed by a max pooliing layer

```
# CONV => RELU => POOL
model.add(layers.Convolution2D(50, (5, 5),
                    activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2),
                    strides=(2, 2)))
```

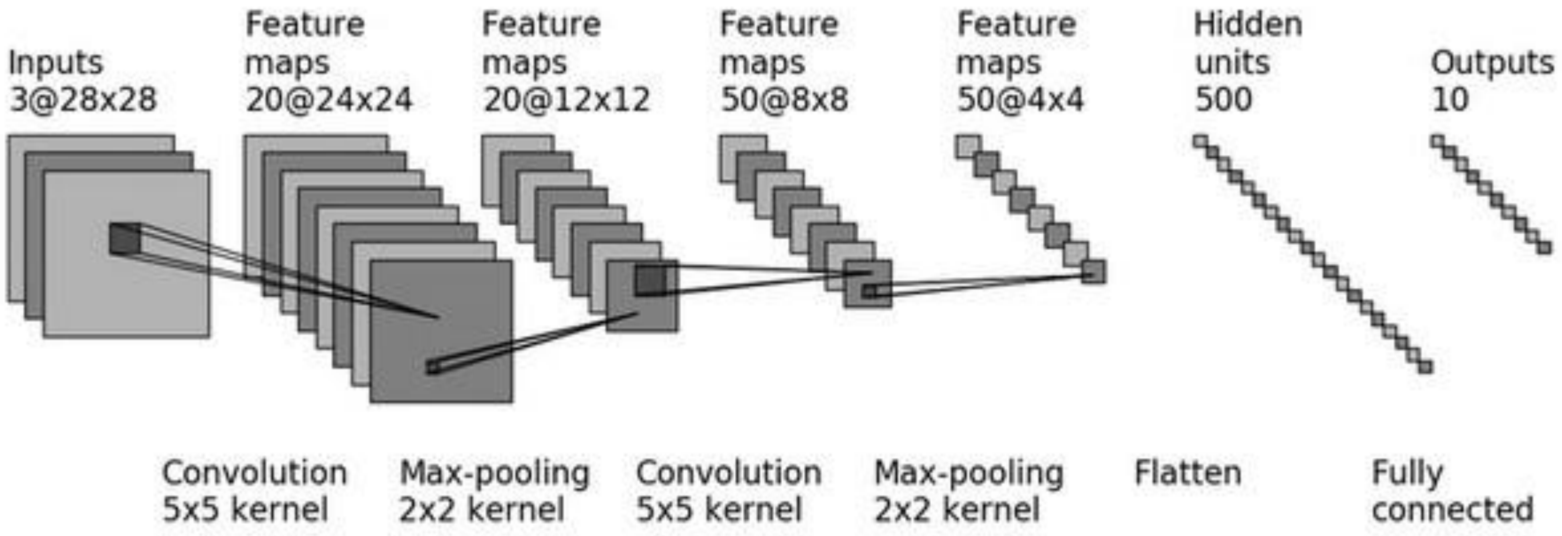
The LeNet CNN Architecture

- Final stage is flattening and a dense network of 500 neurons, followed by a softmax classifier with 10 classes

```
 # Flatten => RELU layers
model.add(layers.Flatten())
model.add(layers.Dense(500, activation='relu')
# a softmax classifier
model.add(layers.Dense(classes, activation="softmax"))
return model
```

The LeNet CNN Architecture

- The LeNet Architecture



- https://www.researchgate.net/publication/312170477_On_Classification_of_Distorted_Images_with_Deep_Convolutional_Neural_Networks.

The LeNet CNN Architecture

- The MNIST data set

```
# data: shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()

# reshape
X_train = X_train.reshape((60000, 28, 28, 1))
X_test = X_test.reshape((10000, 28, 28, 1))

# normalize
X_train, X_test = X_train / 255.0, X_test / 255.0

# cast
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

The LeNet CNN Architecture

- The MNIST data set

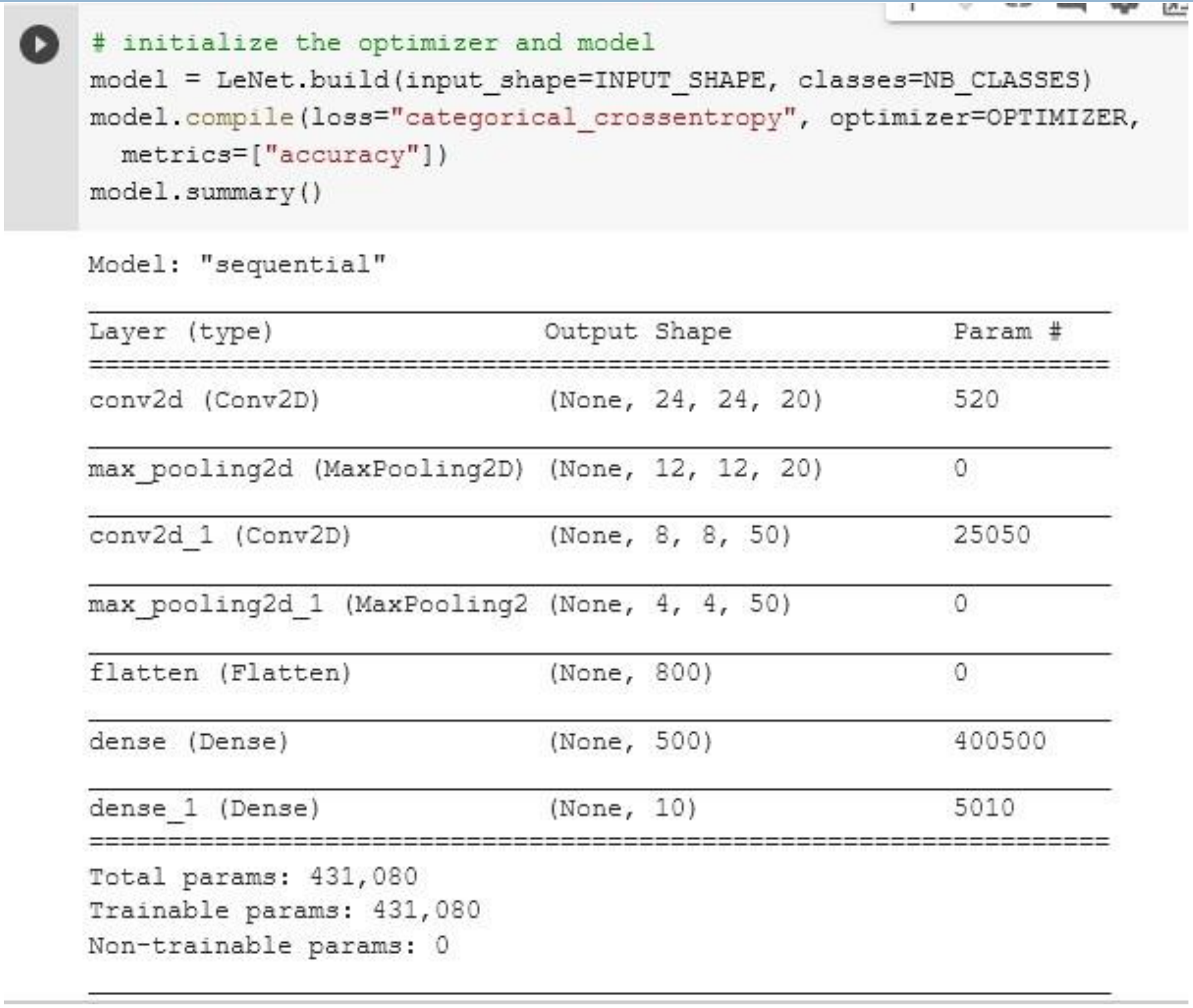
```
# convert class vectors to binary class matrices
y_train = tf.keras.utils.to_categorical(y_train, NB_CLASSES)
y_test = tf.keras.utils.to_categorical(y_test, NB_CLASSES)
```

The LeNet CNN Architecture

- Initialise the model and optimiser

```
model = LeNet.build(input_shape=INPUT_SHAPE,
classes=NB_CLASSES)
model.compile(loss="categorical_crossentropy",
optimizer=OPTIMIZER, metrics=["accuracy"])
model.summary()
```

The LeNet CNN Architecture



```
# initialize the optimizer and model
model = LeNet.build(input_shape=INPUT_SHAPE, classes=NB_CLASSES)
model.compile(loss="categorical_crossentropy", optimizer=OPTIMIZER,
  metrics=["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 20)        520
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 20)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 50)          25050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 50)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 500)               400500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 431,080
Trainable params: 431,080
Non-trainable params: 0
_________________________________________________________________
```

The LeNet CNN Architecture

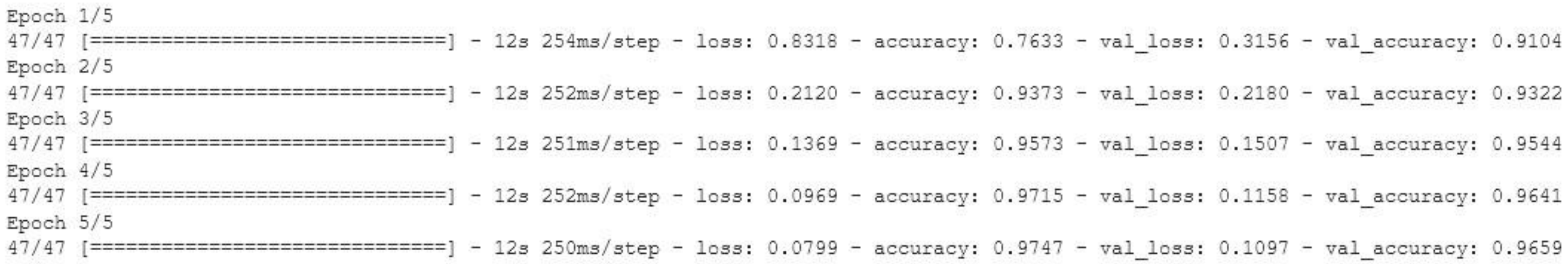
- Using TensorBoard

```
callbacks = [
    # Write TensorBoard logs to `./logs` directory
    tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
```

The LeNet CNN Architecture

- Train the network

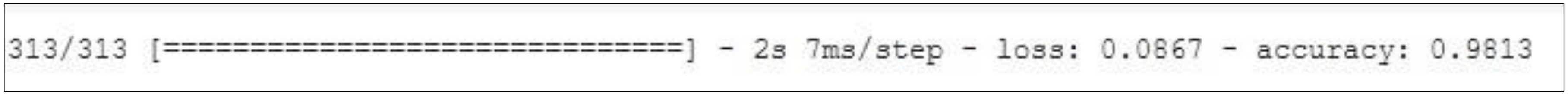
```
history = model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, verbose=VERBOSE, validation_split=VALIDATION_SPLIT, callbacks=callbacks)
```



```
Epoch 1/5
47/47 [==============================] - 12s 254ms/step - loss: 0.8318 - accuracy: 0.7633 - val_loss: 0.3156 - val_accuracy: 0.9104
Epoch 2/5
47/47 [==============================] - 12s 252ms/step - loss: 0.2120 - accuracy: 0.9373 - val_loss: 0.2180 - val_accuracy: 0.9322
Epoch 3/5
47/47 [==============================] - 12s 251ms/step - loss: 0.1369 - accuracy: 0.9573 - val_loss: 0.1507 - val_accuracy: 0.9544
Epoch 4/5
47/47 [==============================] - 12s 252ms/step - loss: 0.0969 - accuracy: 0.9715 - val_loss: 0.1158 - val_accuracy: 0.9641
Epoch 5/5
47/47 [==============================] - 12s 250ms/step - loss: 0.0799 - accuracy: 0.9747 - val_loss: 0.1097 - val_accuracy: 0.9659
```

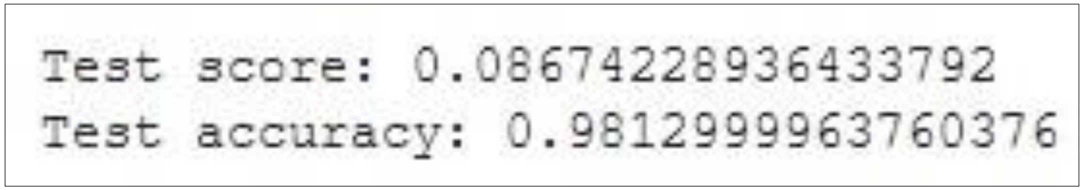
Train the network

```
score = model.evaluate(X_test, y_test, verbose=VERBOSE)
```



```
313/313 [==============================] - 2s 7ms/step - loss: 0.0867 - accuracy: 0.9813
```

```
print("\nTest score:", score[0])
print('Test accuracy:', score[1])
```

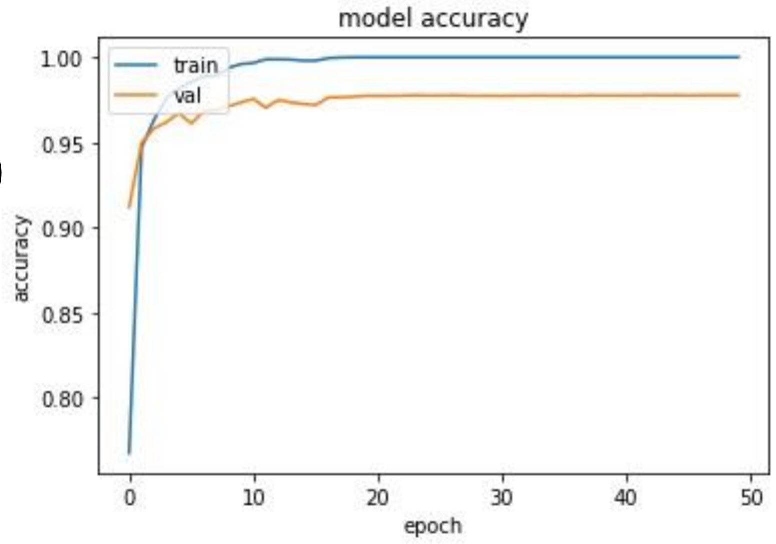


```
Test score: 0.08674228936433792
Test accuracy: 0.9812999963760376
```

The LeNet CNN Architecture

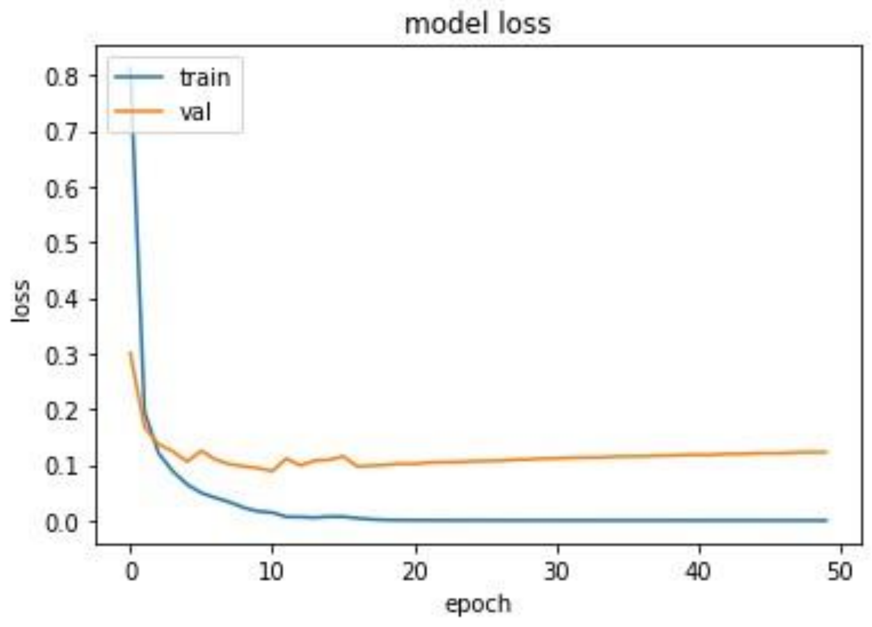
- Plotting

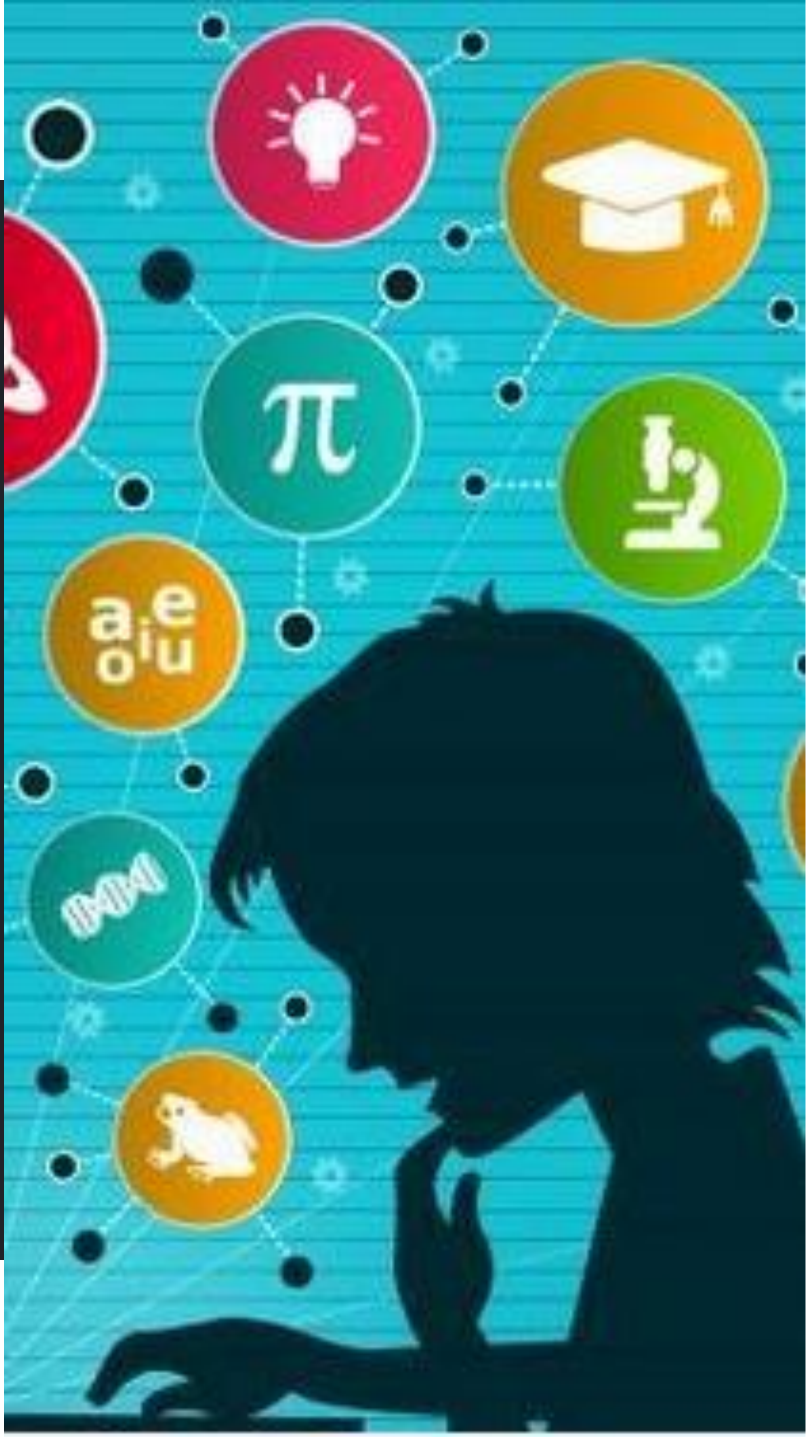
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper
left')
plt.show()
```



The LeNet CNN Architecture

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')plt.show()
```



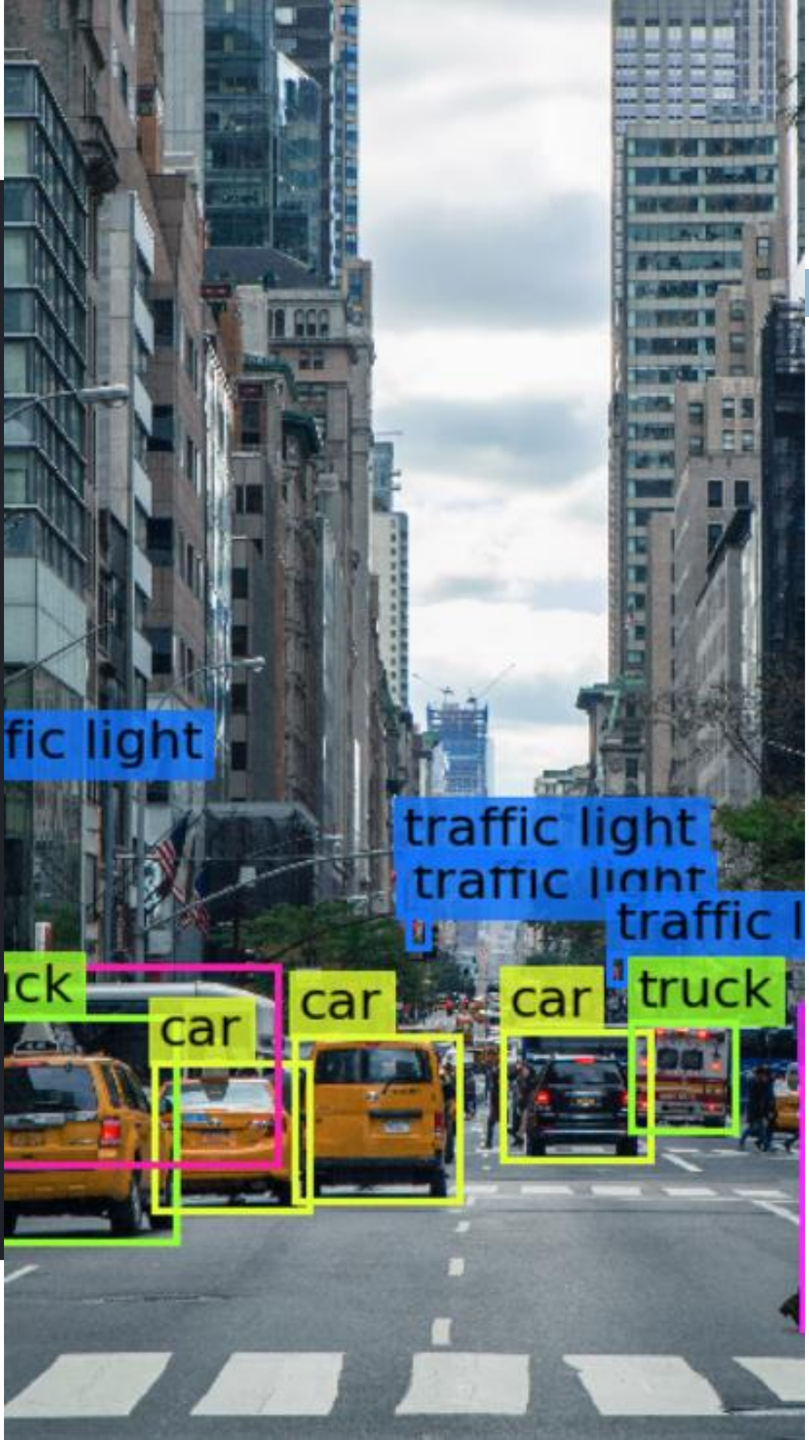


Exercise

- ❑ Implement an CNN for MNIST

Look up

- ❑ http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html



Recap

- ❑LeNet-5
- ❑Vanishing Gradients
- ❑Saturation
- ❑Replace Logistic with ReLU
- ❑Dying Units
- ❑ELU and other variants

Part 4

Data

Based on:

- K. Keoch. Cross Entropy Loss Function.
  https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e
- Aurelien Geron. Hands On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Ed. O’Reilly. 2019

1. Categorical Data

- **Nominal data** is classified without a natural order or rank
- A "*pet*" variable with the values: "*dog*" and "*cat*".
- A "*color*" variable with the values: "*red*", "*green*", and "*blue*".

- **Ordinal data**. Variable comprises a finite set of discrete values with a ranked ordering between values.
- A "*place*" variable with the values: "*first*", "*second*", and "*third*".
- The "*place*" variable above does have a natural ordering of values.

The other data type is **numeric** which is continuous or discrete, and can be further decomposed into ratios and intervals.

1. Conversion

- 3 common approaches for converting categorical variables to numeric format. They are:
- **Ordinal / Integer Encoding**
  - For example, "*red*" is 1, "*green*" is 2, and "*blue*" is 3
  - It imposes an ordinal relationship where no such relationship may exist.
  - Forcing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).
  - One-hot encoding may be used instead.
- **One-Hot Encoding**
- **Dummy Variable Encoding**

1. One Hot Encoding

- A vector of binary bits is used as the label
- tf.one_hot()

Categorical Nominal	[['red'] ['green'] ['blue’]]
One Hot Encoding	[[0. 0. 1.] [0. 1. 0.] [1. 0. 0.]]

Dummy Variable Encoding

- One-hot encoding creates one binary variable for each category.
- Inbuilt redundancy.
- For example, if we know that [1, 0, 0] represents “*blue*” and [0, 1, 0] represents “*green*” we don’t need another binary variable to represent “*red*“, instead we could use 0 values for both “*blue*” and “*green*” alone, e.g. [0, 0]. Categorical
- [['red’] ['green’] ['blue’]]
- [[0. 1] [1. 0] [0. 0]]
- This is called a dummy variable encoding, and always represents C categories with C-1 binary variables.

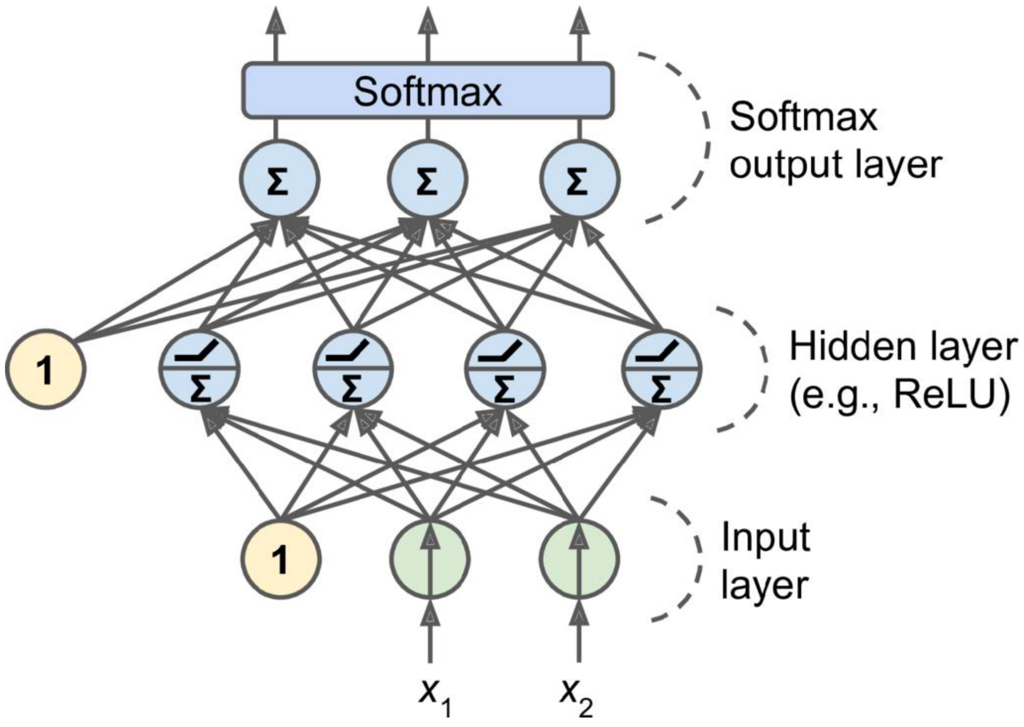
Other Encodings

FYI in passing

- Effect Encoding
- Binary Encoding
- BaseN Encoding
- Hash Encoding
- Target Encoding
- https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/

Softmax

- If each instance can belong only to a single class,
- And there are 3 or more classes
- Multiclass Classification
- MINST → classes 0 through 9 for digit image classification, one output neuron per class, and
- Recommended to use the softmax activation function for the whole output layer.



Softmax

- Ensures that all the estimated probabilities for each possible class are between 0 and 1 and that they add up to 1 (which is required if the classes are exclusive).
- Use MSE for Linear Regression
  - More generally, a linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias term* (also called the *intercept term*),

Linear Regression

- $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots. + \theta_n x_n$
- $\hat{y} =$ predicted value
- n is the number of features
- x_i is the ith feature value
- θ_j is the jth model parameter including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \theta_3, \theta_4, \ \ldots\theta_n$

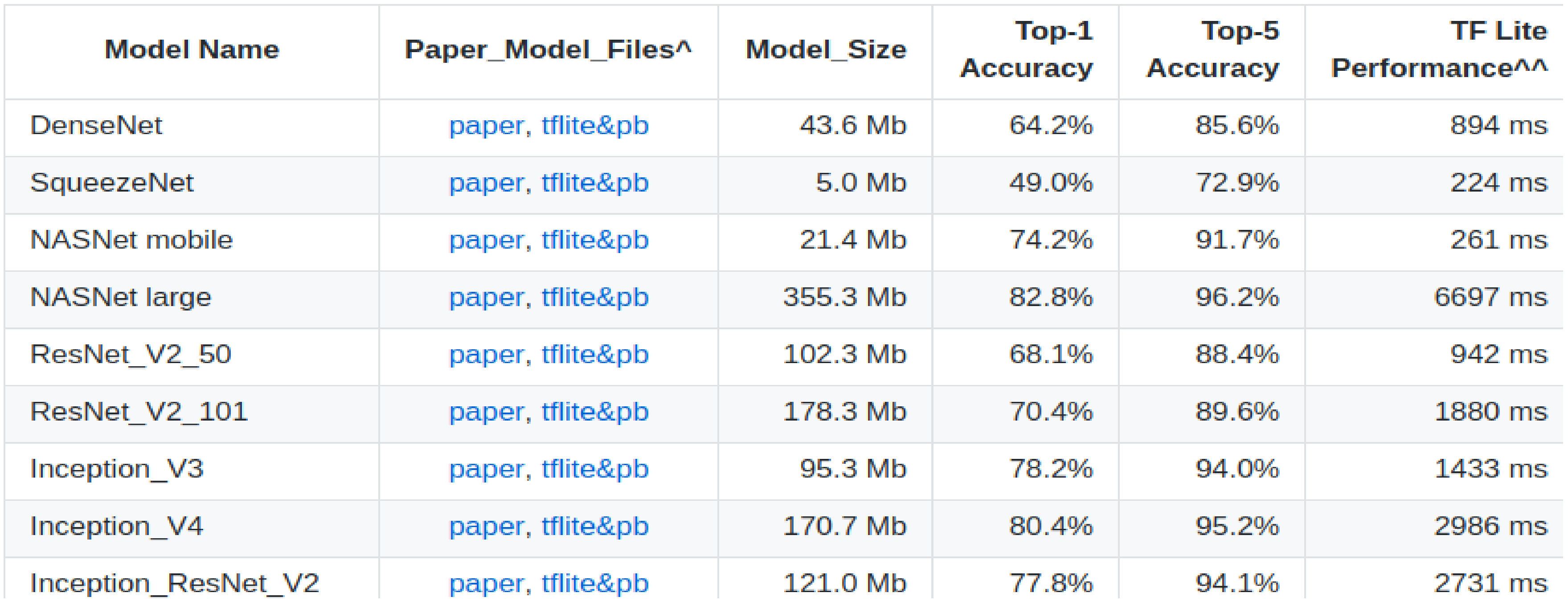
Part 4

Loss Functions And Optimisers

Measuring Performance

1. **Top-1 and Top-5**

- Most often seen method across the research papers.



Model Name	Paper_Model_Files^	Model_Size	Top-1 Accuracy	Top-5 Accuracy	TF Lite Performance^^
DenseNet	paper, tflite&pb	43.6 Mb	64.2%	85.6%	894 ms
SqueezeNet	paper, tflite&pb	5.0 Mb	49.0%	72.9%	224 ms
NASNet mobile	paper, tflite&pb	21.4 Mb	74.2%	91.7%	261 ms
NASNet large	paper, tflite&pb	355.3 Mb	82.8%	96.2%	6697 ms
ResNet_V2_50	paper, tflite&pb	102.3 Mb	68.1%	88.4%	942 ms
ResNet_V2_101	paper, tflite&pb	178.3 Mb	70.4%	89.6%	1880 ms
Inception_V3	paper, tflite&pb	95.3 Mb	78.2%	94.0%	1433 ms
Inception_V4	paper, tflite&pb	170.7 Mb	80.4%	95.2%	2986 ms
Inception_ResNet_V2	paper, tflite&pb	121.0 Mb	77.8%	94.1%	2731 ms

Error Metrics

For Regression Tasks:

Mean Absolute Error (MAE):
Measures the average magnitude of errors, without considering their direction.
Easily interpretable.

Mean Squared Error (MSE):
Calculates the average of the squared differences between predicted and actual values.
Penalizes larger errors more heavily

Root Mean Squared Error (RMSE): The square root of MSE.
Provides an error metric in the same units as the target variable, making it more interpretable.

Root Mean Squared Logarithmic Error (RMSLE):
Useful when the target variable has a wide range of values.
Less sensitive to outliers than RMSE

R-Squared (R2):
Measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s).
Indicates how well the model fits the data

Error Metrics

<u>**For Classification:**</u>
Accuracy
Precision
Recall
F1-Score
AUC-ROC
Confusion Matrix
Cross Entropy (Log Loss)

Cross Entropy

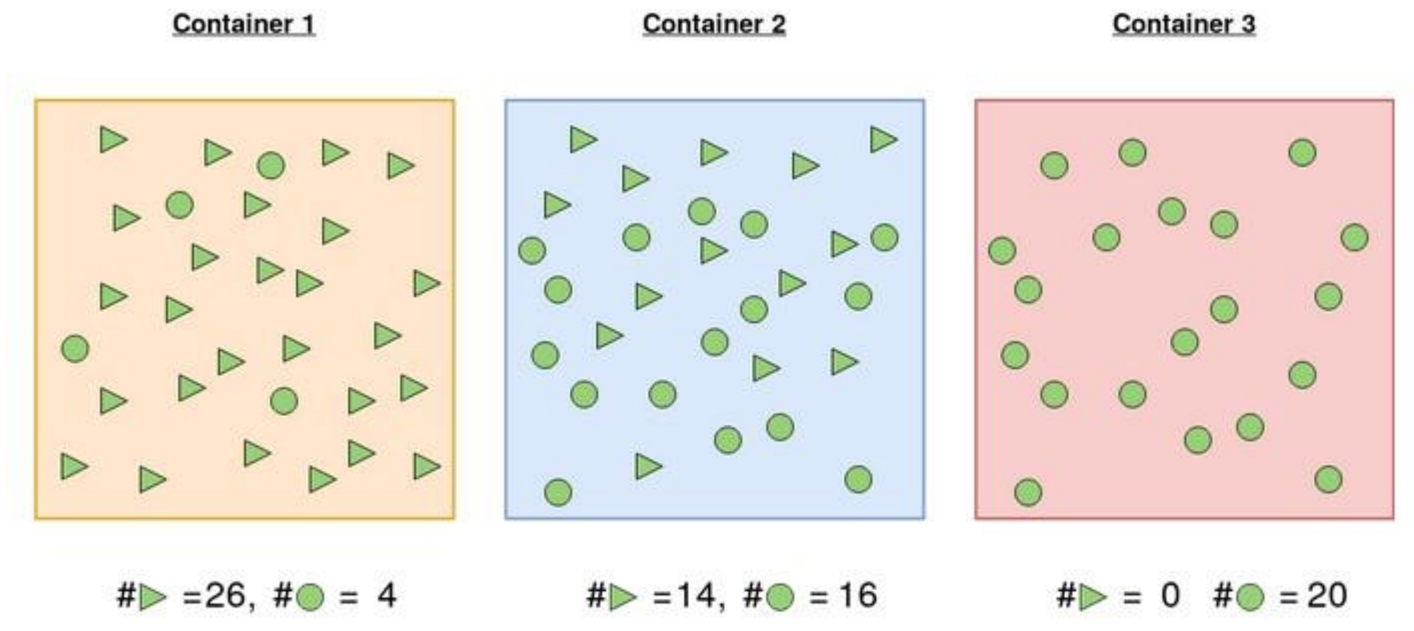
- Cross entropy originated from information theory.
- Suppose you want to efficiently transmit information about the weather.
- If there are eight options (sunny, rainy, etc.), you could encode each option using three binary bits
  - $2^3 = 8$.
- However, if it will be sunny almost every day, it would be much more efficient to code “sunny” on just one bit (0) and the other seven options on four bits (starting with a 1).
- Cross entropy measures the average number of bits you actually send per option.
- If your assumption about the weather is perfect, cross entropy will be equal to the entropy of the weather itself (i.e., its intrinsic unpredictability).
- But if your assumptions are wrong (e.g., if it rains often), cross entropy will be greater by an amount called the *Kullback–Leibler (KL) divergence.*

Cross Entropy

- Low probability event → unpredictable / surprising → more bits
- High probability event → predictable / not surprising → less bits
- Entropy for a random variable X is
- H$(X) = \begin{cases} -\int_x p(x) \log p(x) \\ -\sum_x p(x) \log p(x) \end{cases}$
- $\log p(x) <$ 0 for all p(x), hence minus sign

Cross Entropy

- https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e



Cross Entropy

- https://towardsdatascience.com/cr
  entropy-loss-function-f38c4ec8643

Entropy for container 1

$$\begin{aligned} H(X) &= -\sum_x p(x)\log(p(x)) \\ &= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\ &= -\left[\frac{26}{30}\log_2\left(\frac{26}{30}\right) + \frac{4}{30}\log_2\left(\frac{4}{30}\right)\right] \\ &= 0.5665 \end{aligned}$$

Entropy for container 2

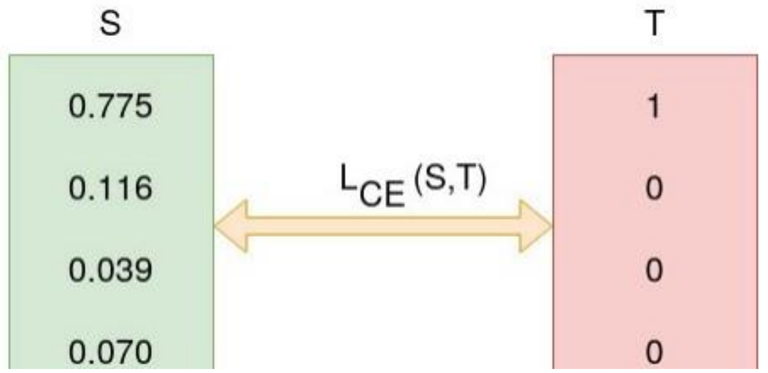
$$\begin{aligned} H(X) &= -\sum_x p(x)\log(p(x)) \\ &= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\ &= -\left[\frac{14}{30}\log_2\left(\frac{14}{30}\right) + \frac{16}{30}\log_2\left(\frac{16}{30}\right)\right] \\ &= 0.9968 \end{aligned}$$

Entropy for container 3

$$\begin{aligned} H(X) &= -\sum_x p(x)\log(p(x)) \\ &= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\ &= -\left[\frac{20}{20}\log_2\left(\frac{20}{20}\right) + \frac{20}{20}\log_2\left(\frac{20}{20}\right)\right] \\ &= 0 \end{aligned}$$

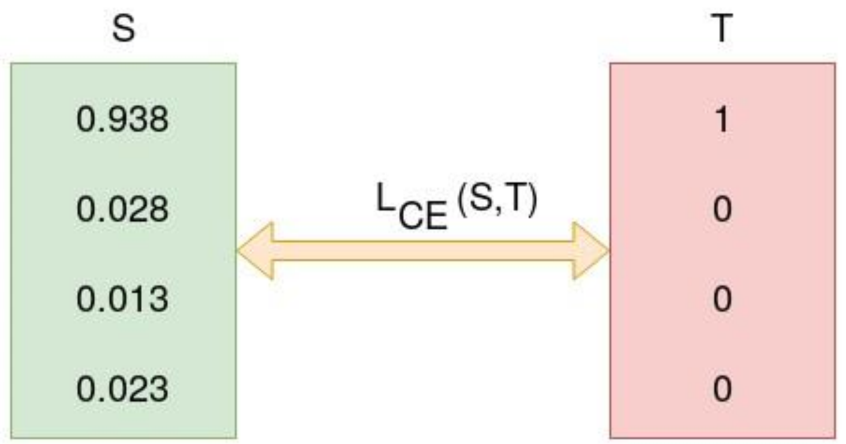
Cross Entropy

- https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e
- For N data points where t_i is the truth value 0 or 1 for the ith data point, and p_i is the Softmax probability for the ith data point.
- $L = \frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1-t_j)\log(1-p_j)\right]\right]$



$$\begin{aligned} L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\ &= -\left[1\log_2(0.775) + 0\log_2(0.126) + 0\log_2(0.039) + 0\log_2(0.070)\right] \\ &= -\log_2(0.775) \\ &= 0.3677 \end{aligned}$$

Cross Entropy



$$L_{CE} = -1\log_2(0.936) + 0 + 0 + 0$$
$$= 0.095$$

- It minimizes the distance between two probability distributions - predicted and actual
- “using the cross-entropy error function instead of the sum-of-squares for a classification problem leads to faster training as well as improved generalization.”

C. M. Bishop. Pattern Recognition and Machine Learning. Springer. 2006.

More on Cross Entropy

- Aurélien Géron video.
  - https://www.youtube.com/watch?v=ErfnhcEV1O8&t=66s
- Mustafa Murat Arat blog
  - https://mmuratarat.github.io/2018-12-21/cross-entropy

Cross Entropy

Spare Categorical Cross Entropy
- Truth labels are integer encoded [1], [2], etc.

Categorical Cross Entropy
- Labels are one hot encoded.
- For example, if there are 3 classes [0,0,1], [0,1,0], [1,0,0]

Optimisers

Fundamental:

- ❑ GD

  GD provides a more accurate estimate of the gradient but is computationally expensive.

- ❑ SGD: updates the model parameters using the gradient of the loss function on a single randomly selected data point or a small batch of data points

  SGD provides a noisy but efficient estimate, enabling faster convergence and better scalability

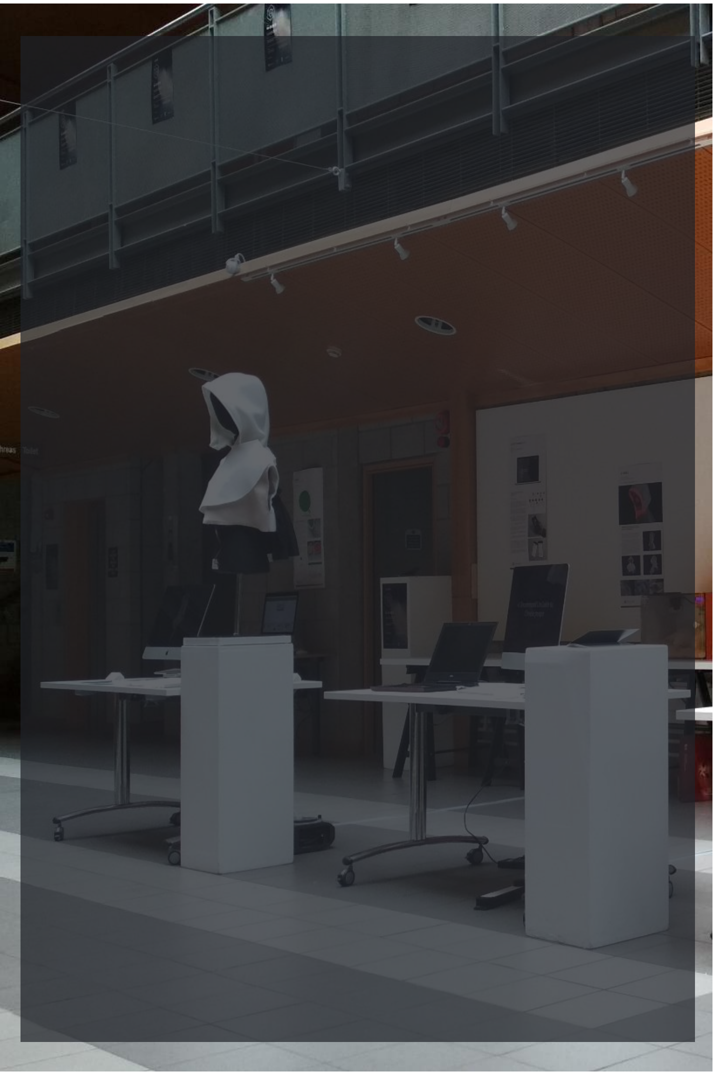
  The "noise" allows the algorithm to escape shallow local minima in the loss function, potentially finding better, more generalized solutions.

Optimisers

Adaptive Learning Rate Optimisers	Adagrad (Adaptive Gradient)
	Adadelta (Adaptive Delta)
	RMSProp
	Adam (Adaptive Moment Estimation)
	AdamW
	Nesterov-accelerated Adaptive Moment Estimation (NADAM)
Others	Nesterov Accelerated Gradient (NAG): A variant of momentum that often leads to faster convergence
	Lion: A newer optimizer, designed to be more efficient than Adam

Part 5

Dropout



Dropout

Drop Out Is one of 3 approaches to improve generalisation (reduce overfitting)

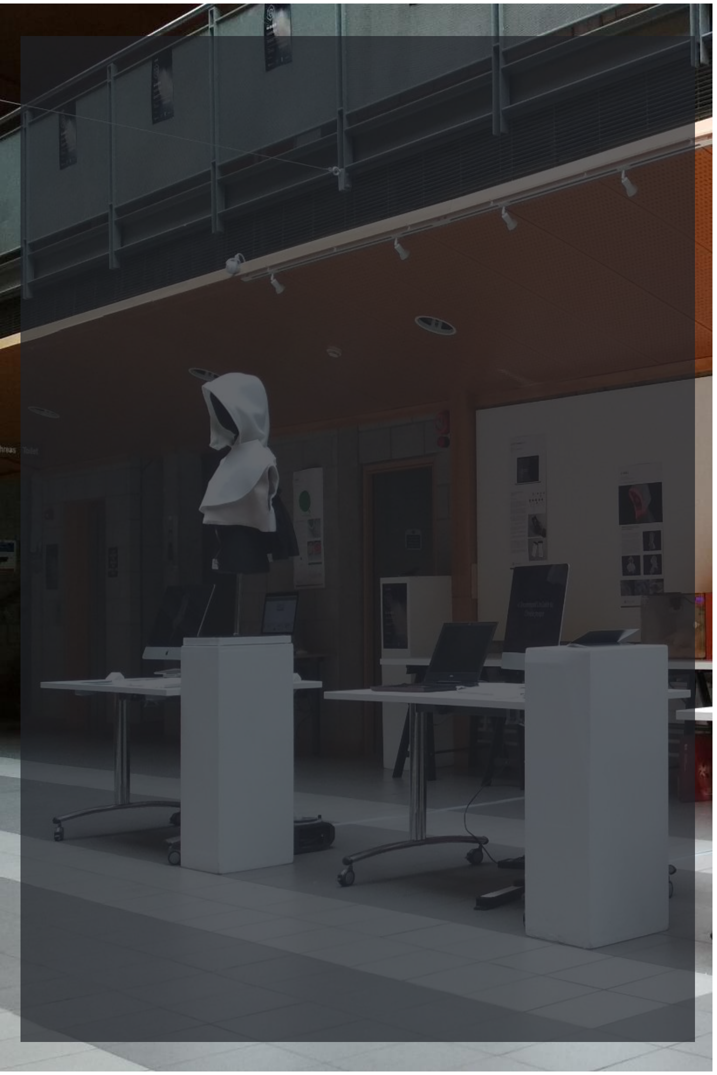
- L1 and L2 regularisation
- Batch Normalisation: where primary intent was to reduce vanishing gradients

Dropout was proposed by Geoffrey Hinton in 2012 and further detailed in a 2014 paper Nitish Srivastava et al.,

Even the state-of-the-art neural networks get a 1–2% accuracy boost simply by adding dropout.

A model that has 95% accuracy will reduce error rate by a 40% with a gain an additional 2% in accuracy going from 5% error to roughly 3%.

Dropout



Rationale:

- Network is an ensemble of thousands of networks
- Robustness and adaptability
- Less likely to overfit (memorise training data)

Every neuron including the input neurons but always excluding the output neurons has a probability p of being temporarily “dropped out,”

- it will be entirely ignored during this training step,
- It outputs a zero
- The hyperparameter p is called the dropout rate, and it is typically set between 10% and 50%:
    - closer to 20–30% in recurrent neural nets
    - closer to 40–50% in convolutional neural networks

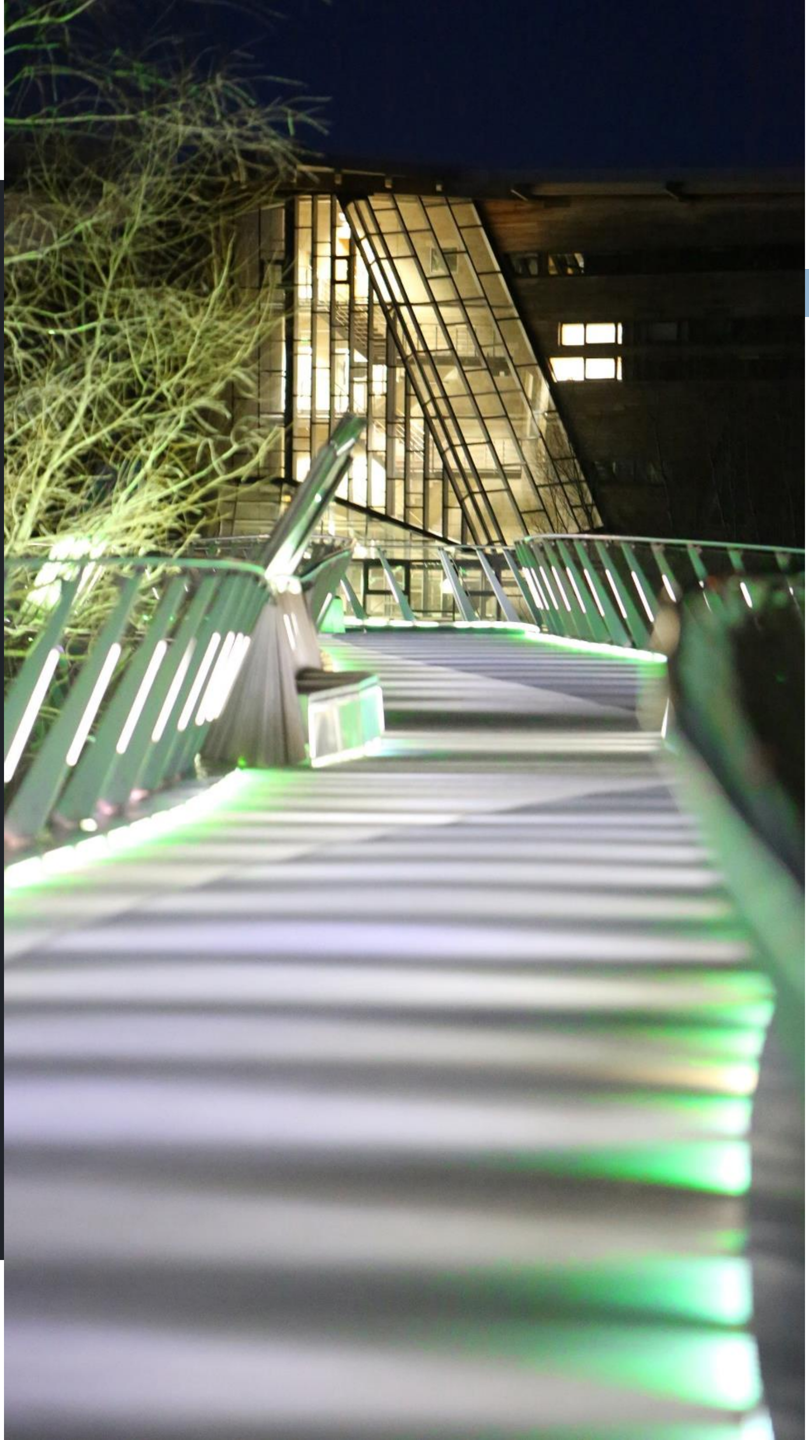
Drop Out

- Suppose p = 50%, in which case
- during testing a neuron would be connected to twice as many input neurons as it would be (on average) during training.
- To compensate for this fact, multiply each neuron's input connection weights by 0.5 after training.
- If we don't, each neuron will get a total input signal roughly twice as large as what the network was trained on and will be unlikely to perform well.
- More generally, multiply each input connection weight by the **keep probability** (1 – p) after training.
- Alternatively, divide each neuron's output by the keep probability during training (these alternatives are not perfectly equivalent, but they work equally well).

Drop Out

- Since dropout is only active during training, comparing the training loss and the validation loss can be misleading.
- In particular, a model may be overfitting the training set and yet have similar training and validation losses.
- **So make sure to evaluate the training loss without dropout (e.g., after training).**

CS6482



Next Lecture

- He / LeCunn Initialisation
- Batch Normalisation
- Measuring Performance
- Google LeNet and ResNet