Computer Vision

Computer Vision Basics

Visual Perception

The act of observing patterns and objects through sight or visual input. We need to build systems that understand the environment through visual input

Vision Systems

Human Vision Systems

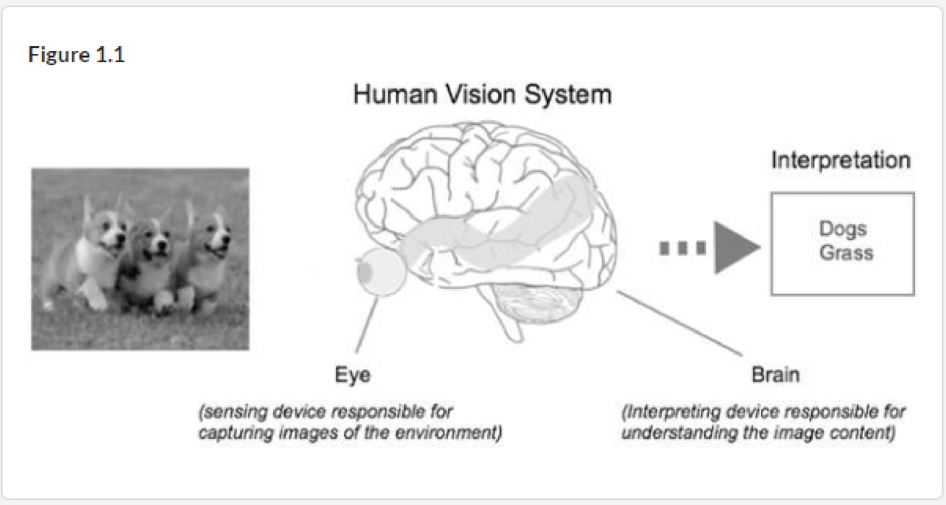


Figure 1.1

Human Vision System
Interpretation
Dogs
Grass
Eye
(sensing device responsible for capturing images of the environment)
Brain
(Interpreting device responsible for understanding the image content)

- Input = (**Sensing device**) eyes
- brain = (**Interpreting device**) neurons firing in order to detect the object
- output = The object seen, detected and described

AI Vision Systems

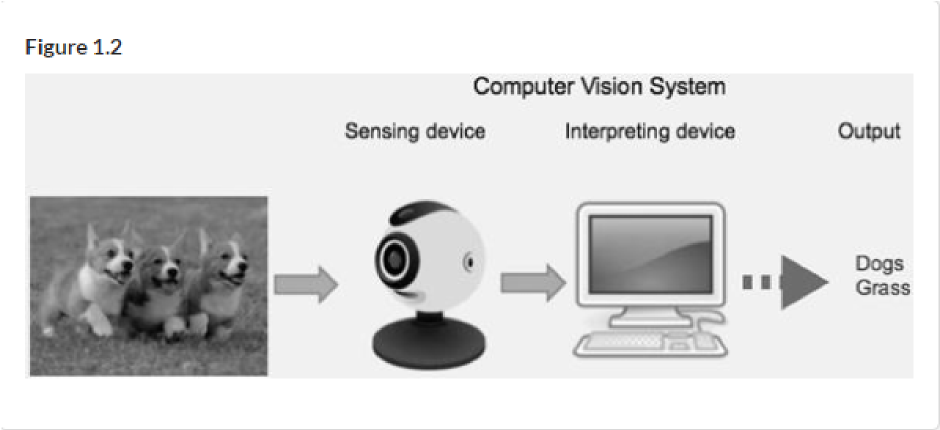


Figure 1.2

- Input = (**Sensing device**) camera

- Computer = (**Interpreting device**) Processing + algorithm for detecting the object
- Output = The object detected and described in the form of probability

Sensing Devices

Any device that could be used for an input for vision systems,

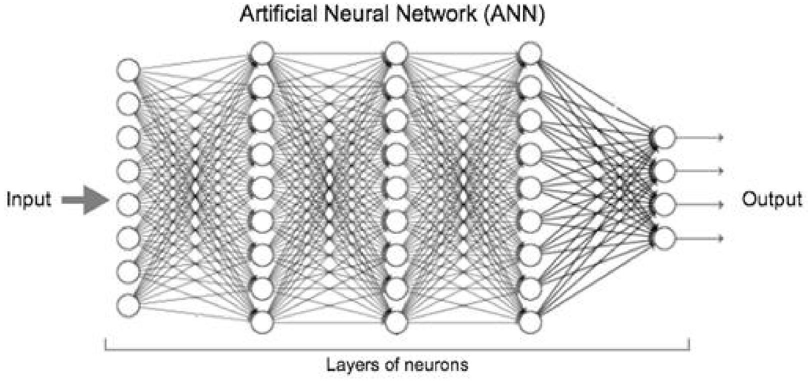
Types of devices include:

- Lidar - technique used to fire invisable pulses of light to map the area in 3D
- Cameras - Can be used in various ways in the electromagnetic radiation spectrum for different use cases
- Radars - Can measure distance and velocity, cannot see in detail

Sensing devices are chosen depending on the use case. For medical use cases X-ray immagery is more likely to be used. For self driving cameras or Lidar are the best choices

Interpreting Device

Figure 1.5



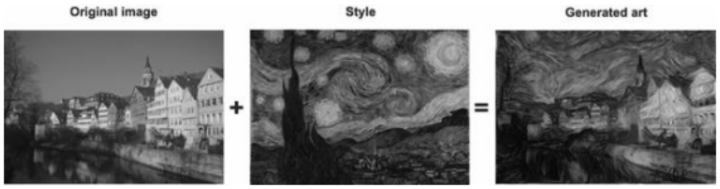
Its the algorithm used for identifying images in the object.

- The algorithm is based on the human model of the brain
  - Many neurons linked together im the brain are modeled using ANNs (Artificial Neural Networks)
  - Artifical Neurons are built in layers. Building multilayer neural networks is known as deep learning
  - There are many deep learning architectures such as:
    - ANN (Artificial Neural Networks)
    - CNN (Convolutional Neural Networks)
    - RNN (Recurrent Neural Networks)
  - Deep learning is required for computer vision

Applications of Computer Vision

1. Image classification
    - CNNs usually the best choice
    - Mainly used to classify and recognise certian objects in images
2. Object detection and localisation
    - Combination of CNNs and RNN
    - Common architectures used are:
        - YOLO (You Only Look Once)
        - SSD (Single Shot Detection)
    - The task is to label the image
3. Automatic image captioning
    - CNNs and RNNs are used in image captioning
    - The task is to provide a caption of the image description
4. Generative art (Style transfer)

Figure 1.10



    - Take an image and apply a style of art to that image
5. Creating images
    - GANs (Generative Adversarial Models)
        - Its an evolved CNN architecture
    - Can generate entirely new images that are realistic from previous images

Computer Vision Pipeline

The pipeline (set of steps) used to understand and interpret images

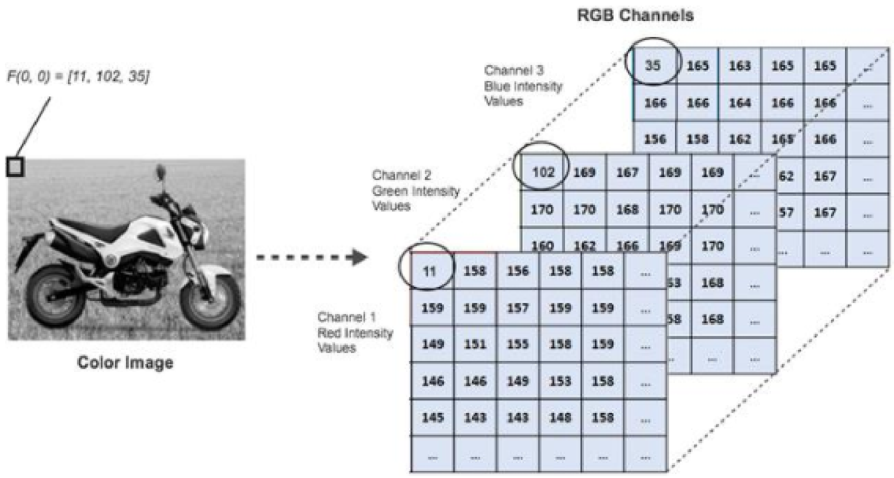
1. Input data
    - Images, Videos (Image Frames)
2. Preprocessing
    - Geting the data ready

3. Feature extraction
    - Finding and distinguishing information about the image
4. ML model
    - Learn from the extracted features to predict and classify objects

1. Input Data

The input data is an image pixel value.

Figure 1.18



Channel 3

35	165	163	165	165	...
166	166	164	166	166	...
156	158	162	165	166	...
			62	167	...
			57	167	...
			...	...	...

Channel 2

102	169	167	169	169	...
170	170	168	170	170	...
160	162	166	169	170	...
			53	168	...
			58	168	...
			...	...	...

Channel 1

11	158	156	158	158	...
159	159	157	159	159	...
149	151	155	158	159	...
146	146	149	153	158	...
145	143	143	148	158	...
...	...	...	...	...	...

- The pixel value is stored in a matrix
- Pixel values could be in the form of grayscale which is 0 → 255 in a single tuple OR in RGB form.
    - RGB has 3 channels Red, Green and Blue in the form (0→255, 0→255, 0→255)
    - In matrx form they are represented as 3 seperate matrices and when combined produce a colored image.
    - A tuple of (700, 700, 3) would mean 3 channes (matrices) of 700*700 images. That means a total of 490000 pixels in a single channel
- Think of Images as a function of inputs
    - F(x, y) gives the intensity at position (x, y) → this is for a grayscale image
    - F(x, y) = [red(x,y), green(x,y), blue(x,y)] → this is for a colored image

2. Image Processing

The cleaning an preperation of data

- Data could be messy, confusing or inefficient to process, they need to be consistent
- Data processing techiques include

1. **Convert color images to grayscale** if color is not needed to detect features. This could reduce the amount of data that needs processing and overall make the computation faster. If the use case would need color imagery then use colors othewise the black and white is the better option.
2. **Resise the images** in the data set to fit the inputs of the CNN as different sized images would not satisfy the algorithm requirements
3. **Augment the existing dataset** with different versions of itself. This is done to enlarge the dataset and expose the neural network to different types of senarios
4. **Removing background color** to reduce noise
5. **Brighen or darken** the image
6. **Any other adjustments** needed could be nessesary to help the neural network perform better

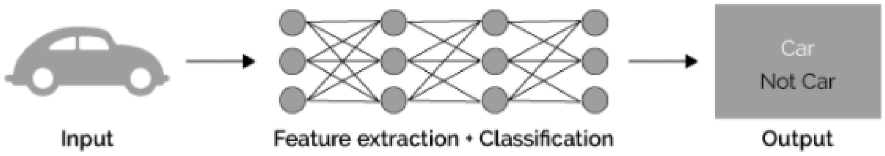
3. Feature extraction

Core component in the computer vision pipeline. Extract useful features

You want to transform raw data into a **features vector.**

- This uses a feature extraction algorithm
- Before using the algorithm we need to decide on a good feature
  - Good features are:
    - Identifiable
    - Easily tracked and compared
    - Consistent across different scales, lighting conditions, and viewing angles
    - Still visable in noisy images or when only part of an object is visable
- Extracting features
  - Traditional machine learning manually extracts features

Figure 1.30

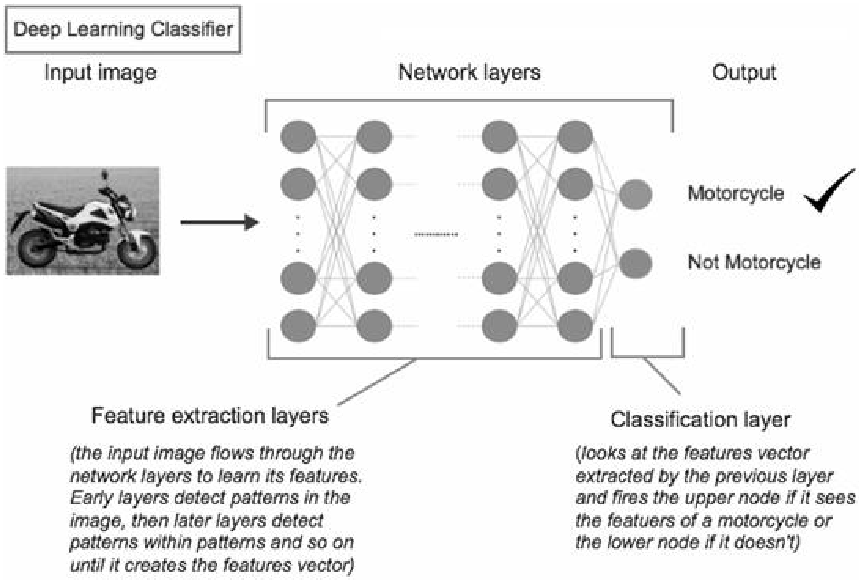


  - In deep learning we do not need to manually extract features from the image. The network automatically extracts features while learning the importance of these features.

4. ML Model

Feeding the extracted features into a classifier to output a class label for the images (Car or not)

Figure 1.33



(the input image flows through the network layers to learn its features. Early layers detect patterns in the image, then later layers detect patterns within patterns and so on until it creates the features vector)

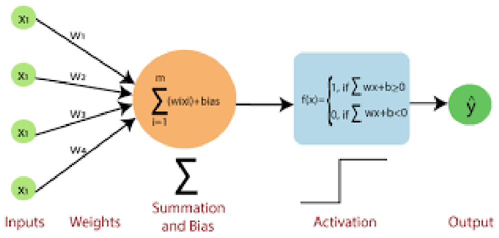
(looks at the features vector extracted by the previous layer and fires the upper node if it sees the featuers of a motorcycle or the lower node if it doesn't)

- Automatically extract useful features from a dataset and act as a classifier to output the class labels of images. Input images are passed through the network to learn their features layer by layer.

Deep Learning and Neural Networks

The perceptron

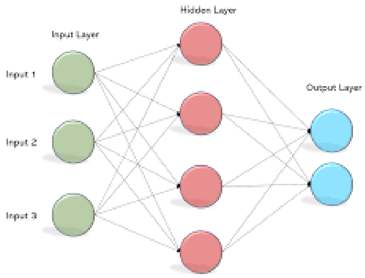
- Most simple neuron.
- Takes an input and outputs a value between a 0 or a 1.
    - 1 = neuron is activated, 0 = neuron is deactivated.
- To calculate the output of the neuron
    - f(o) = x
        - o = w1.x1 + w2.x2 + ... + wn.xn (weighted sum)
        - f(o) = activation function for the weighted sum.
            - Perceptron's activation is the step function.


x_1
x_2
x_3
x_4
w_1
w_2
w_3
w_4
$\sum_{i=1}^{m}(w_ix_i)+\text{bias}$
$f(x)=\begin{cases}1, \text{ if } \sum wx+b\geq0\\0, \text{ if } \sum wx+b<0\end{cases}$
$\hat{y}$
$\sum$
Summation and Bias
Inputs
Weights
Activation
Output

            - according to the step function:
                - threshold = 0.5.
                    - f(weighted sum + (bias)) > 0.5 then neuron outputs a 1.
                    - f(weighted sum + (bias)) < 0.5 then neuron outputs a 0.

MLP (Multi Layer Perceptron)

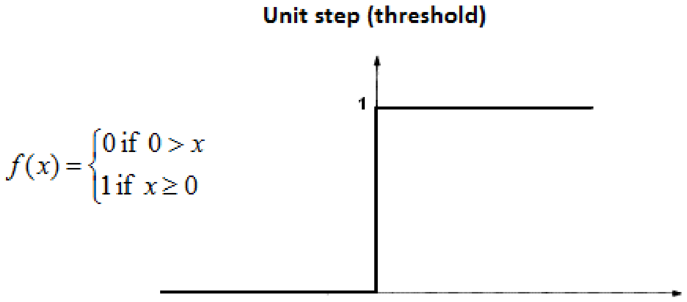
- A single neuron is not enough to train a neural network to learn its inputs.
- Need multiple neurons organised in multiple layers; known as MLP
    - We have the input layer → hidden layers → output layer
        - input layer: data to input
        - hidden layer: hidden from sight, learns patterns in the data
        - output layer: layer that outputs the results.
    - Why do we need multiple layers instead of one neuron?
        - To abstract finer details of the data inputted
        - We don't necessarily need multiple layers for this, 3 is the standard.
            - More layers could mean overfitting, less layers could mean underfitting, 3 layers hits the sweet spot.
            - 1 input layer, 1 hidden layer and 1 output layer



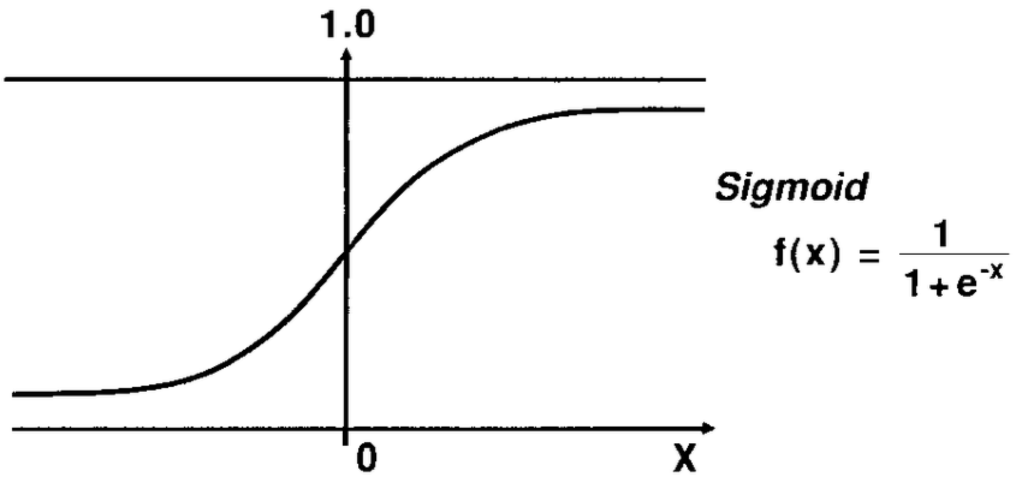
-
    - Every neuron in a MLP is connected to every other neuron with weights.
    - Bias term added to each layer
        - Why do we need a bias?
            - Imagine the weighted sum being a graph.
                - Without the bias all the graphs stem from the origin.
                - Bias prevents this and allows more flexibility and opportunity for the neuron to access a wide range of values.
                - This can prevent overfitting and underfitting.

How does an MLP learn?

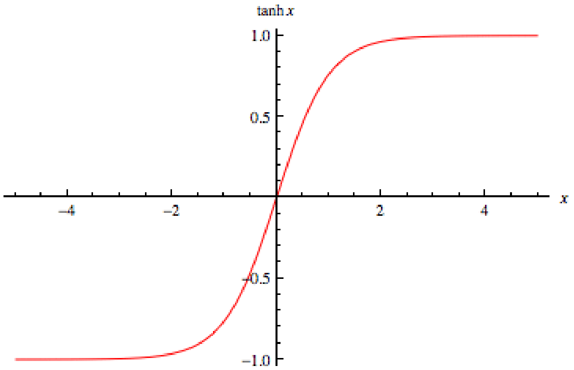
- Three main steps of learning:
  - **Feed Forward**
    - **Step 1: Weighted sum**
      - Node in the next layer = node in the previous layer . weight connecting to the node in the next layer (or the next layer weight)
      - $n + 1 = \Sigma_k^{i=1} n.n_{1i}$ (in other words)
    - **Step 2: Activation function**
      - Apply an activation function to the weighted sum to get the strength of the neuron
      - $O = activation\_function(\Sigma_k^{i=1} n.n_{1i})$
      - There are various activation functions to use in different use cases which include:
        - **Step function**
          - If the input is < 0 return 0; else return 1.
          - Used in the perceptron and binary classification of True of False scenarios



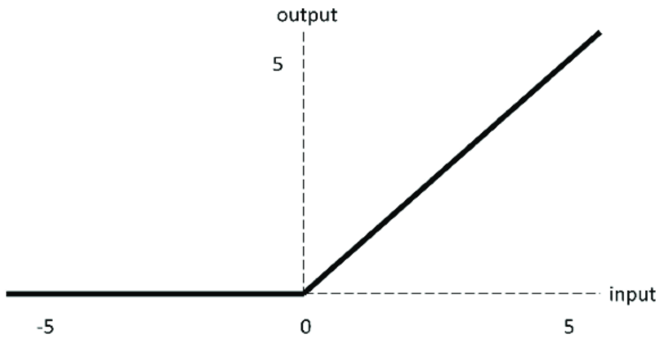
        - **Sigmoid**
          - $x \to +\infty, x \to -\infty$
          - $y > 0, y < 1$
          - S shaped curve, gives a more fluid range of the activations than a step function.



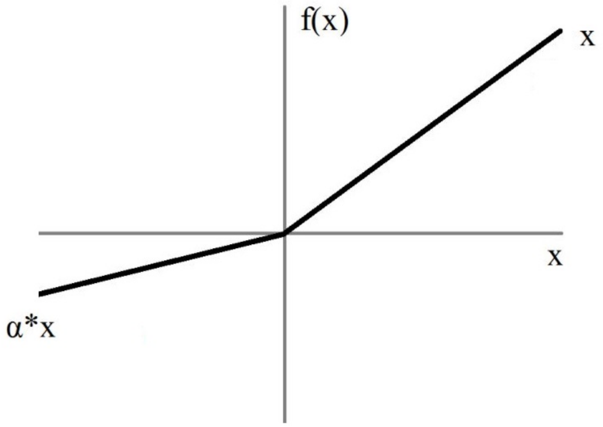
- **Softmax**
  - Similar to sigmoid but can accept more than 2 classes
  - $O(x_i) = e^{xi} / \sum_i e^{xi}$
  - Usually added on the output layer. xi is the expected x output for that current node and the sum is all the outputs for all the output nodes
- **Hyperbolic tanh function (tanh)**
  - Shifted sigmoid function
  - Slows down gradient descent if Z is too large or too small



- **Rectified Linear Unit (ReLU)**
  - Activates the node only if above 0
  - Trains better in hidden layers

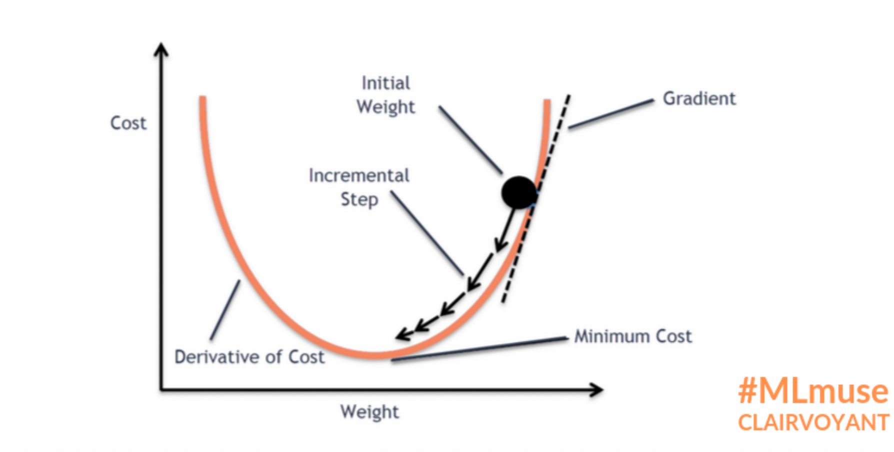


- **Leaky ReLU**
    - Disadvantage of ReLU is that $\frac{dy}{dx} = 0 \ if \ x < 0$ hence neuron is no longer active.
    - To fix this Leaky ReLU introduces a small slope at $x < 0$



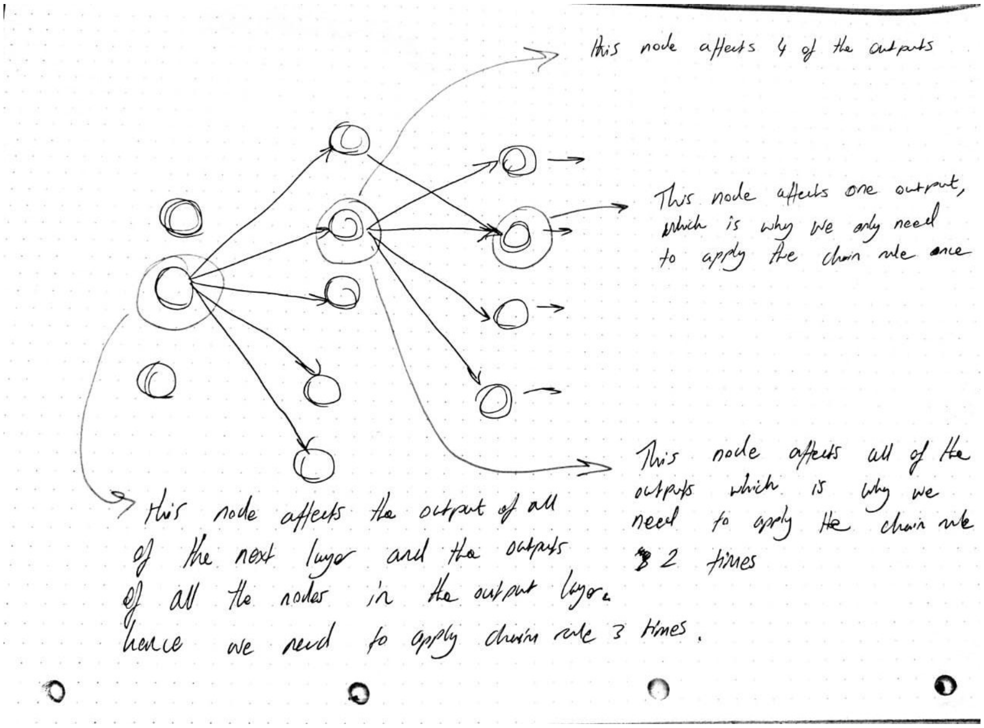
- **Calculate Loss**
    - Once the signal strengths of the nodes have been calculated we calculate the loss or the error of the network.
    - We use a loss function for this.
    - There are various loss functions to use in different use cases which include:
        - **MSE (Mean Squared Error)**
            - Ensures that the error is always +ve
            - Sensitive to outliers since the error can grow quadratically
            - MSE is used if the outliers don't matter as much
            - Equation:
            - $E(W, b) = \frac{1}{N}\Sigma_{i=1}^{N}(\hat{y}_i - y_i)^2$

        - $\hat{y}$ is the prediction
        - y is the output
    - **MAE (Mean Absolute Error)**
        - $E(W,b) = \frac{1}{N}\Sigma_{i=1}^{N}|\hat{y}_i - y_i|$
        - Similar to MSE but no need to square. Instead take the absolute.
        - This is used in situations where the outliers matter
    - **Cross Entropy**
        - Used for classification problems with multiple classes
        - Difference between 2 or more classification problems
        - It determines how close the predicted distribution is to the true distribution
        - More close to 0 the more true the prediction is
        - $E = -\Sigma_{i=1}^{n}\Sigma_{i=1}^{m} y_{ij}\log(P_{ij})$
- **Backpropagate**
    - Once a loss has been calculated we backpropagate the loss function to minimise the error.
    - Lets map the loss value against the weight.
    - **Step 1: Gradient descent**



    - Gradient descent is an optimisation algorithm that finds the minimum value for the loss to be close to 0 and the optimum weight.
    - $\frac{\partial loss}{\partial weight}$ → multiple factors determine this value such as
        - The loss with respect to the activation function

    - The activation function with respect to the input
    - The input with respect to the weights
- This forms a chain rule to calculate the loss with respect to the weight
    - We use the chain rule to find the derivative of composite functions
    - Example node in the output layer:
        - $\frac{\partial loss}{\partial weight_{12}^{(L)}} = (\frac{\partial loss}{\partial activation_1^{(L)}})(\frac{\partial activation}{\partial input_1^{(L)}})(\frac{\partial input}{\partial weight_{12}^{(L)}})$ where 1→2 is the weight from node 2 in the output layer to node 1 in the hidden layer at L-1
        - where 1 is the node in the hidden layer L-1 represented as a function
        - This means that the update in the weight on layer L depends on the activation output, the input and the weight on L-1
    - Example node in the hidden layer
        - we need to calculate derivatives later in the network for all the outputs instead of 1 like in the node for the output layer.
        - $\frac{\partial loss}{\partial weight_{22}^{(L-1)}} = (\frac{\partial loss}{\partial activation_2^{(L-1)}})(\frac{\partial activation_2^{(L-1)}}{\partial input_2^{(L-1)}})(\frac{\partial input_2^{(L-1)}}{\partial weight_{22}^{(L-1)}})$
            - The outputs of this network all depend on activation in (L-1) hence we need another chain rule to calculate this
            - $\frac{\partial loss}{\partial activation_2^{(L-1)}} = \Sigma_{j=0}^{nL-1}(\frac{\partial loss}{\partial activation_j^{(L)}})(\frac{\partial activation}{\partial input_j^{(L)}})(\frac{\partial input_j^{(L)}}{\partial activation_2^{(L-1)}})$ hence:
            - $\frac{\partial loss}{\partial weight_{22}^{(L-1)}} = [\Sigma_{j=0}^{nL-1}(\frac{\partial loss}{\partial activation_j^{(L)}})(\frac{\partial activation}{\partial input_j^{(L)}})(\frac{\partial input_j^{(L)}}{\partial activation_2^{(L-1)}})](\frac{\partial activation_2^{(L-1)}}{\partial input_2^{(L-1)}})(\frac{\partial input_2^{(L-1)}}{\partial weight_{22}^{(L-1)}})$
            - We do this until we reach the input layer



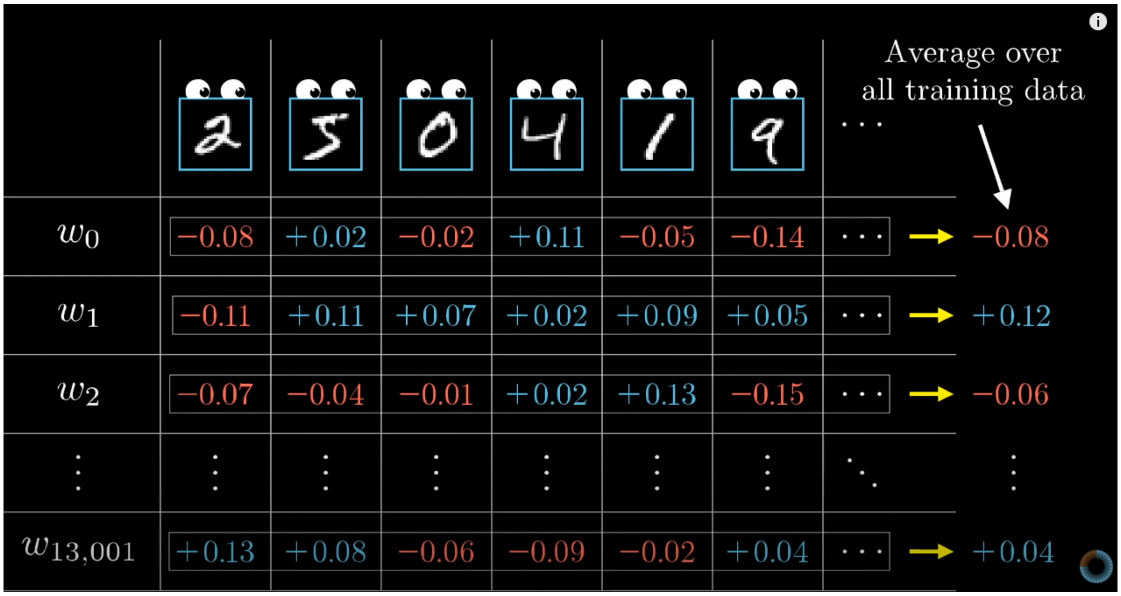
this node affects 4 of the outputs

This node affects one output, which is why we only need to apply the chain rule once

This node affects all of the outputs which is why we need to apply the chain rule ~~3~~ 2 times

this node affects the output of all of the next layer and the outputs of all the nodes in the output layer. hence we need to apply chain rule 3 times.

- To allow for multiple data, we calculate the gradient descent for each weight, add that to a table, average the values for each weight and update accordingly
- Every step gradient descent takes, the weights get updated accordingly



	2	5	0	4	1	9	. . .	Average over all training data
w_0	-0.08	$+0.02$	-0.02	$+0.11$	-0.05	-0.14	. . . →	-0.08
w_1	-0.11	$+0.11$	$+0.07$	$+0.02$	$+0.09$	$+0.05$	. . . →	$+0.12$
w_2	-0.07	-0.04	-0.01	$+0.02$	$+0.13$	-0.15	. . . →	-0.06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
$w_{13,001}$	$+0.13$	$+0.08$	-0.06	-0.09	-0.02	$+0.04$	. . . →	$+0.04$

- Gradient descent has different algorithms that look to speed up the training process. These algorithms include:

  - **Batch Gradient Descent (BGD)**
    1. Take all the data
    2. Compute the gradient
    3. Update the weights, take a step down the slope
    4. Repeat for n number of epochs
  - **Stochastic Gradient Descent (SGD)**
    - Stochastic means random
    - Randomly picks an instance of a training set for each one step and calculates the gradient based on a single instance
    1. Randomly shuffles samples in training set
    2. Pick one data instance
    3. Compute the gradient
    4. Update the weights, take a step down the slope
    5. Pick another data instance
    6. Repeat for n number of epochs
  - **Mini Batch Gradient Descent (MB-GD)**
    - Similar to SGD but takes mini batches and computes the gradient
    - Updates the weights with fewer iterations
- Gradient descent functions that improve the speed of the algorithms above include:
  - **Nestor accelerated gradient**
  - **RMS prop**
  - **Adam**
  - **Adagrad**
- **Step 2: Update the weights**
  - The weights get updated after every gradient descent step for a specific weight.
  - $w\_new_{jk}^{(L)} = w\_old_{jk} - \frac{\partial loss}{\partial weight}$

Core of Learning

FOR EACH EPOCH:

1. Feed forward
   - Weighted sum (Linear combination)
2. Calculate the error
   - Use a loss function to calculate the difference between the expected and predicted

3. Backpropagate
    - Use gradient descent to calculate the change in weight
        - Change in weight is calculated depending on which node and layer the weight is connected to
        - May require multiple applications of the chain rule to do so
    - Then update the weights accordingly

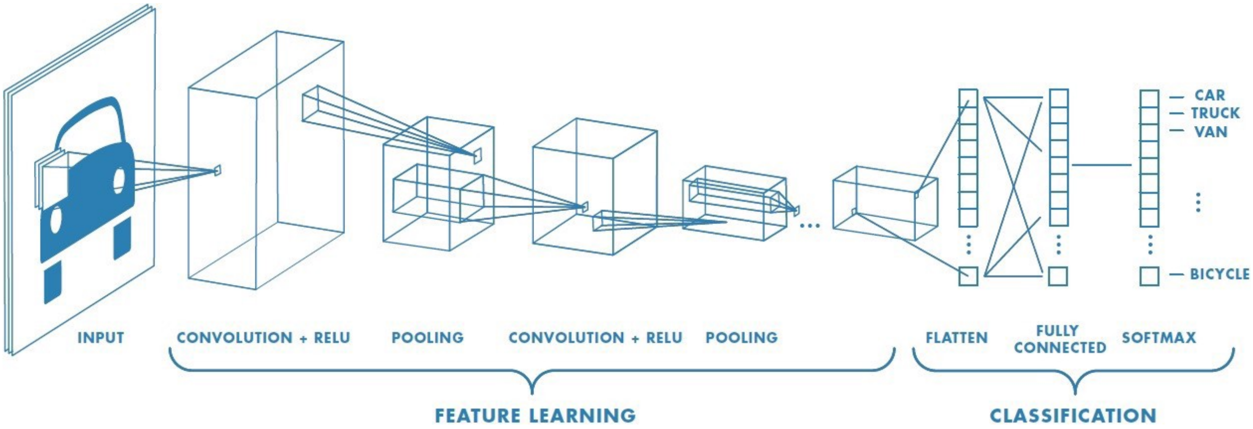
Questions

What is the difference between an epoch and a batch?

Why is Mini-Batch Gradient Descent preferred?

How are the weights stored and updated during an epoch?

CNN Architecture



- CNN = Convolutional Neural Network
- Convolution = operation of two functions to produce a 3rd modified function
- Computer Vision Pipeline
    - Input → Image Preprocessing → Feature Extraction (Using convolutional layers) → ML Model (Using fully connected layers)
    - Other methods previously extracted the features manually, CNN is an architecture that extracts the features automatically
    - Deep learning models process information that is similar to the human brain.
        - It allows the features to be extracted manually and learned after
        - This is done over many layers, usually more than 3.
- CNN architecture is mainly used in computer vision applications

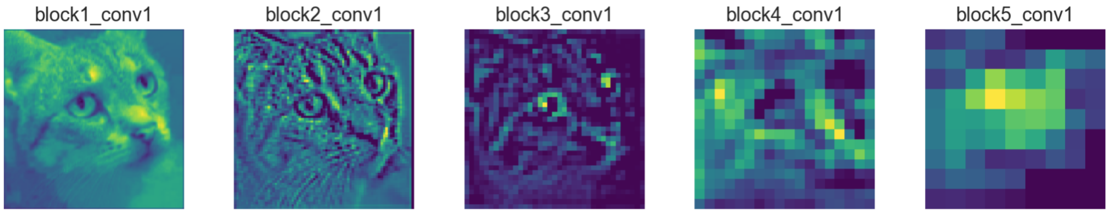
    - There is a feature extractor part and a classification part
    - The feature extractor extracts certain features from an image from high level features to low level features
    - Then the features are fed into a neural network to classify the low level feature

Feature Extractor (Convolutional Layers)

- Represented in convolutional layers
    - Convolutional layers can be stacked like a neural network
    - They are different to neural networks
        - They are not fully connected
        - They look to extract lower level features rather than learn
    - Convolutional layers look to extract features like the human eye
        - Each layer of a convolutional neural network has filters (also known as kernels)
            - Filters are small matrices that detect patterns in an image.
            - Filters scan the image until the image has been fully scanned
            - Filters are like weights in the neural network model as they are learnable
            - Filters like weights are initialised randomly
            - The receptive field is the area the filter is currently scanning.
            - Filter dimensions are represented can be
                - 2×2
                - 3×3
                - 5×5
            - The receptive field has the same dimensions as the filter
            - Example of a 3×3 filter initialised randomly

$$\begin{bmatrix} -1 & 2 & 1 \\ 3 & 1 & 4 \\ 5 & 2 & 1 \end{bmatrix}$$

            - The dot product of a filter with the input image produces a feature map for the next layer
                - Each scan applies the dot product of the receptive field of the feature map / image the filter is scanning, this is then added to a feature map in the next layer over time until the image is fully scanned
            - Once the image is fully scanned the operations move to the next layer and so on ...
            - The dimensions of the feature map reduces layer by layer
            - This has the effect of extracting lower and lower level features



block1_conv1 block2_conv1 block3_conv1 block4_conv1 block5_conv1

    - Filters have certain hyperparameters such as
      - Dimensions: size of the filter which means the size of the receptive field
      - Stride: How many pixels should the scanner move after each scan
        - Usually set to 1 or 2
      - Padding: add padding around the image by using 0's so the filter can focus on the features that matter
    - Convolutional layer in code:

```
# in_channels = input feature maps
# out_channels = output feature maps
# kernel_size = dimensions of the filter
# stride = how many pixels to scan by
# padding = add padding around the image
# padding_mode = what value to use for the padding
Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=0, padding_mode="zeros")
Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=0, padding_mode="zeros")
```

    - 1st convolutional layer has 1 input and 16 outputs. This means 16 filters are scanning the image to produce a feature map of 16 outputs
    - This is fed to the 2nd convolutional layer, the 16 outputs get scanned again but this time using 32 filters. this outputs a feature map of size 32
- How to improve the feature extractor?
  - **Pooling**
    - Reduces the dimensions of the feature maps to improve the speed of training without loosing much detail
      - Also helps with overfitting
    - This is known as down-sampling → reducing parameters and complexity
    - Pooling uses the same concept as filters where we have a 2×2, 3×3, or 5×5 filter scanning the input.
      - Bigger → deeper network, better learning, may overfit
      - Smaller → Captures finer details, preferred over bigger for this reason
    - Just like filters in convolutional layers pooling has parameters such as dimensions, stride and padding
    - Two types of pooling

        - **Max Pooling**
          - The filter scanning the input will take the maximum value in the current receptive field
          - After doing so the value gets added onto the output and takes a stride
          - This is done until the image has been scanned
          - This will reduce the feature map dimensions
          - Objects of interest have the largest pixel values which is why max pooling is preferred to other pooling methods
        - **Average Pooling**
          - Same concept as max pooling but rather than taking the max of each region you take the average
          - Average pooling is used when the position of the objects matter
      - **ReLU**
        - Using ReLU activation function after each layer in a convolutional neural network will introduce non-linearity
        - Non-Linearity
          - Output cannot be reproduced from a linear combination of the inputs
        - This means results are less likely to be replicated and is entirely based on the consistency of the network
        - Most activation functions introduce non-linearity as one of their main use cases
    - **Feature Extraction General Architecture**
      - $conv \rightarrow relu \rightarrow pool$

```
x = pool(relu(conv))
```

    - **Feature Extraction Example Architecture**
      - $conv_1 \rightarrow relu_2 \rightarrow pool_3 \rightarrow conv_4 \rightarrow relu_5 \rightarrow pool_6$

Classification Part (Fully Connected Layers)

- After the feature extraction, the pixel values get flattened from a matrix to a vector and passed into a neural network.
- This is the classification part and the layers are known as FC layers or Fully Connected layers
- FC layers generally have 3 layers
  - Input layer → convolutional output as a vector
  - Hidden layer → extracting meaningful features

    - Activation function → ReLU because it speeds up gradient descent for a highly dense layer such as the hidden layer
  - Output layer → outputs the results
    - Activation function → Softmax because this activation function is built for classification problems to compare multiple classes.
    - Loss function → Cross Entropy Loss because this loss function works well with classification problems
- Why don't we use fully connected layers to feature extract?
  - Fully connected layers may miss some important information about the image such as
    - Spatial features
      - Depth
      - Positions of certain features
    - This is because when flattening a full image into a single vector the image is transformed from 2D to 1D
    - This also happens between convolutional layers and fully connected layers but the effect negated as the pixels are closely grouped together due to convolutions and down-sampling
  - In addition there would be more parameters for the network to work with
    - 500×500 image is 250,000 inputs which can result in a lot of parameters and long training times
    - Convolutional layers extract important features while also down-sampling the images to reduce the number of inputs
- **Fully Connected Layer General Architecture**
  - $fc \rightarrow relu \rightarrow softmax_3$
- **Fully Connected Layer Architecture**
  - $fc_1 \rightarrow fc_2 \rightarrow relu_3 - > softmax_4$

Putting It All Together

- This is an example of a simple CNN architecture involving and feature extraction part and a classification part
- $conv_1 \rightarrow relu_2 \rightarrow pool_3 \rightarrow conv_4 \rightarrow relu_5 \rightarrow pool_6 \rightarrow fc_7 \rightarrow fc_8 \rightarrow softmax_9$
- The above is known as a deep neural network as there are many layers and the network automatically extracts features and classifies them

Structuring Deep Learning Projects and Hyperparameter Tuning

- Terminology

- Parameters → values that are tuned automatically
    - Weights for example
- Hyperparameters → values that are tuned manually
    - Learning rate for example

- Decisions that need to be made when structuring deep learning projects and tuning hyperparameters
    - How accuracy is calculated
    - How data is prepared and processed
    - Baseline model architecture
    - When to tune hyperparameters
    - How to tune hyperparameters

Baseline Model Architecture

- Decide whether the problem best fits an MLP, CNN or RNN
- Is the problem classification, object detection or both
- How deep should the network be
- What activation functions to use
- What loss functions to use
- The type of gradient descent algorithm and function
- Any techniques to tune hyperparameters
- Performance metrics such as accuracy and loss
- How to prepare the data
    - Split
        - Training set used to train the model with
        - Test / Validation set used to validate the model while training
        - Training set, Test set / Validation set
            - 80, 20 split
            - 70, 30 split
    - Data preprocessing
        - Image augmentation
            - Increases data set size, more variance in the data
        - Normalising pixel values from [0→255] → [0→1]
            - Speeds up gradient descent
        - Image grey-scaling

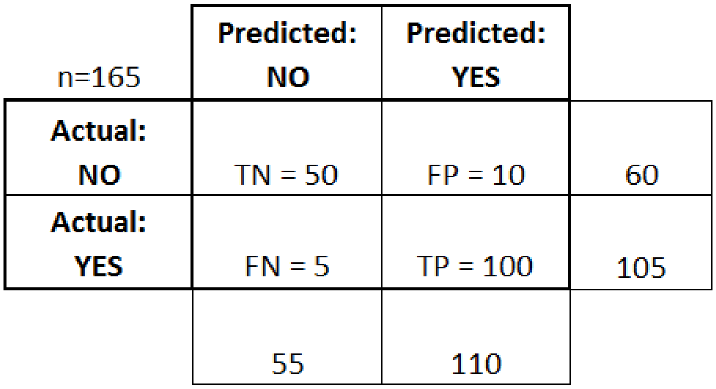
- Reduces the number of channels
- Speeds up training

After baseline model has been decided, implement it and evaluate the models accuracy and performance

Evaluating Model, Determining Performance

Defining Model Evaluation Metric

- Accuracy can be calculated in many ways
  - $accuracy = \frac{\#correct\ predictions}{\#examples}$ simplest way to calculate accuracy and would work for some problems
  - The formula above is not suitable for most classification problems as the model is determined from one angle
  - If we want to know more on how the model is performing we need to know when the model is wrong, when the model is right and anything in-between
  - **Confusion Matrix**
    - Describes the model from different angles
    - There are 4 outcomes in a confusion matrix
      - True Positive (TP) = Correctly predicted yes
        - "Predicted a woman is pregnant and she actually is"
      - True Negative (TN) = Correctly predicted no
        - "Predicted a man is not pregnant and he is actually not"
      - False Positive (FP) = Predicted yes but actually no
        - "Predicted a man is pregnant but he is actually not"
      - False Negative (FN) = Predicted no but actually yes
        - "Predicted a woman is pregnant but she actually is"

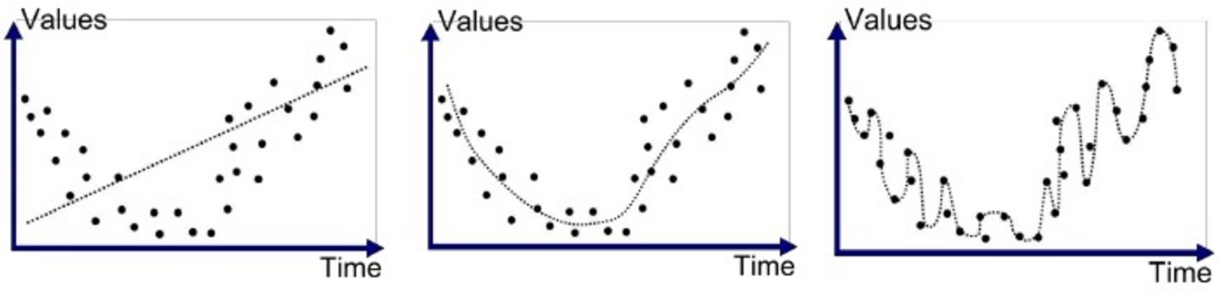


n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

- **Evaluating a confusion matrix**
  - **Recall**
    - Out of all the positive classes how many did we predict correctly
    - $Recall = \frac{TP}{TP+FN}$
    - Useful metric in cases where FN is a higher concern than FP
  - **Precision**
    - How many of the correctly predicted cases actually turned out to be positive
    - $Precision = \frac{TP}{TP+FP}$
    - Useful metric in cases where FP is a higher concern than FN
  - **Accuracy**
    - $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$
  - **F-Score**
    - Way to combine precision and recall.
    - If you want the best of both worlds you would use F-Score as your performance metric
    - $F - score = \frac{2pr}{p+r}$
- Deciding the best accuracy model
  - Decide whether your model needs a confusion matrix or just accuracy
  - Then decide if your model needs precision, recall or f-score if you decide to use a confusion matrix

Interpreting Performance

- The data is usually split between training and testing / validation datasets.
  - This means we can test our model on unseen data
- At the end of each epoch we will have a training error and a validation error
  - Training error → Tells us how well our model is performing while training
  - Validation error → Tells us how well our trained model is performing on unseen data
- We can tell if the model is underfitting or overfitting by comparing the training data and the validation error



Underfitted

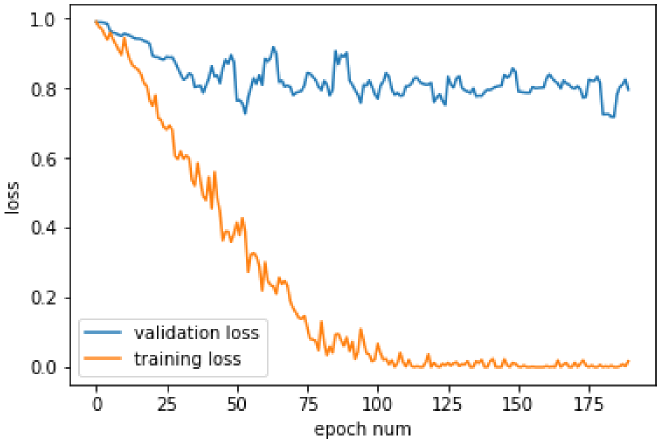
Good Fit/Robust

Overfitted

- How do we tell?
  - $training\ error < validation\ error$ model failed to generalise and it is overfitting
  - $training\ error > validiation\ error$ model failed to generalise and it is underfitting
- How do we generalise?
  1. Determine if the performance is acceptable
  2. Visualise the training accuracy and the validation accuracy
  3. Is the model underfitting? Tune hyperparameters where possible and clean up data
  4. Is the model overfitting? More data could be effective
  5. Repeat until the model has succeeded in generalising

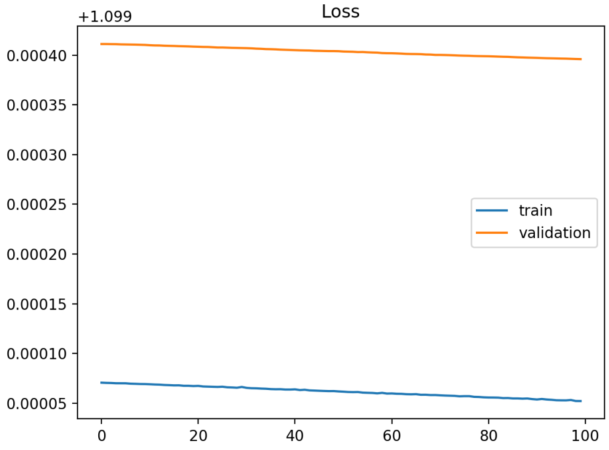
Hyperparameter Tuning Techniques

Plotting Learning Curves

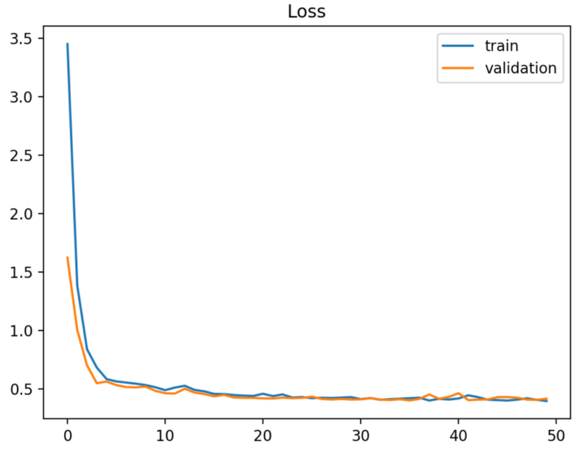


- Example of overfitting

- Training set learns but validation set failed to generalise



- Example of underfitting
    - Training and validation failed to generalise



- Example of generalization
    - Training and validation decreasing in loss

- From here we can decide which hyperparameters to tune depending on the graphs shown above

What to do when the model is underfitting

- Get more training data
    - Data augmentation

- Increase the size or the number of parameters in the model
- Increase the complexity of the model
    - Goal is to increase the number of layers so the network can generalise
- Increase the training time, until the loss function is minimised
    - Increase the number of epochs

What to do when the model is overfitting

- **Regularisation**
    - Penalises the loss function by adding an extra term to decrease the weight value
    - Reduces variance of the model without substantially increasing the bias
    - By penalising the loss function how much the weights get tuned are affected and could make some weights less relevant
    - $error\_func_{new} = error\_func_{old} + regularisation\_term$
    - There are two regularisation terms L1 and L2 that penalise the loss function in different ways
        - **L2 Regularisation**
            - Also known as weight decay as it forces the weights to go towards zero but not exactly zero
            - The regularisation term
                - $L2 = \frac{\lambda}{2m} * \sum ||w||^2$
                - λ = regularisation parameter → hyperparameter that can be tuned to improve overfitting if overfitting still occurs
                - m number of instances
                - w is the weight
            - The full equation
                - $error\_func_{new} = error\_func_{old} + \frac{\lambda}{2m} * \sum ||w||^2$
        - **L1 Regularisation**
            - We penalise the absolute value of the weights instead of the squared value
            - The weights may be reduced to zero here unlike L2
                - This is very useful when trying to compress our model
            - The regularisation term
                - $L1 = \frac{\lambda}{2m} * \sum ||w||$
                    - λ = regularisation parameter → hyperparameter that can be tuned to improve overfitting if overfitting still occurs
                    - m number of instances

        - w is the weight
    - The full equation
        - $error\_func_{new} = error\_func_{old} + \frac{\lambda}{2m} * \sum ||w||$
    - For both L1 and L2 the regularisation term increases the loss function which leads to larger derivatives. This will mean smaller weights when the weights get updated by:
        - $W_{new} = W_{old} - \alpha(\frac{\partial error}{\partial W_x})$ → larger derivative of the loss function, smaller W_{new}
- **Dropout**
    - Dropout a certain percentage of weights in the fully connected layer
    - This gives a chance for the weaker weights to strengthen and learn
    - This also reduces the number of parameters and increases the speed of training
    - Dropout introduces randomness in the network and is preferred when we have large neural networks to introduce more randomness
- **Early Stopping**
    - Monitors validation error and stops when the error starts to increase
- **Batch Normalisation**
    - Normalising the inputs of the hidden layer
    - If we can normalise the input layer why not normalise the inputs to the hidden layers
    - Why does normalising the input increase the speed of training and reduce overfitting?
        - If we train a model of a white cat and feed it a different colored cat that is brown for example the data is in a different distribution with different values.
        - This can result in covariance shift
            - Covariance shift occurs if the model is trained on another dataset and the x and y values shift
        - Batch normalisation can fix covariance shift by distributing the data in the same range of values like how an image is normalised between 0→1 pixel values
    - Batch normalisation occurs before the activation function is applied in the convolutional layers
        - This is because we can normalise the values in the same range before applying the activation function
        - This can reduce the chance of the **vanishing gradient problem**
            - This is where the gradient almost vanishes making the neuron pretty much dead
    - The goal of batch normalisation is to achieve a stable distribution of activation values throughout training
    - Note: It may not be a good idea to combine batch normalisation and dropout
        - The data may become noisy due to the statistics used in batch normalisation that could affect nodes later in the network

    - Therefore using either batch normalisation or dropout should be enough, be careful using both
- Reduce the complexity of the model
  - Goal is to decrease the number of layers so the network can generalise

What you can do for both scenarios

- Get more training data
  - Data augmentation
    - Image augmentation involves many transformation and different image manipulation techniques to increase the size and range of the dataset
- Remove noise from the data
  - Image preprocessing
- **Optimise learning rate**
  - Learning rate is the most important hyperparameter
    - Too low model could undershoot the minimum
    - Too high model could overshoot the minimum
    - Good learning rate and the loss decreases consistently

What to do to reduce parameters and/or increase training speed

- **Activation type**
  - Make sure that the activation functions are non-linear
  - More neurons in the network the more parameters the network needs to optimise therefore the more computational complexity
    - Hence choose your activation functions wisely for each layer
- **Pooling**
  - Add pooling to convolutional layers
  - Tweak the stride to reduce dimensions
  - Add padding if the borders are not needed to reduce dimensions
  - The techniques above reduces parameters
- Optimise learning rate
  - **Learning rate decay and adaptive learning**
    - Learning rate changes linearly during learning
    - The learning rate is set to a high value and decreases over time
    - This can increase the training time and decreases the chance of the overshooting the minimum

- **Use a different gradient descent optimisers**
    - **Adam**
        - Adaptive moment estimation
        - Behaves like a heavy ball with friction
        - Outperforms other optimisers
- **Use a different gradient descent algorithm**
    - MB-GD (Mini Batch Gradient Descent) is used in most cases as we can train multiple training instances at a time
        - Batches can be modified like a hyperparameter
            - Typical starting value would be 64 or 128 batch size however could range between 32→1024 depending on the problem
    - **SGD vs MB-GD vs BGD**
        - SGD
            - More epochs required for a good model
            - Less computer recourses used per epoch however
        - MB-GD
            - Hits the sweet spot when it comes to epochs and computer resources
        - BGD
            - Less epochs required for a good model
            - More computer resources used per epoch

Advanced CNN Architectures

Transfer Learning

Object Detection with R-CNN, SSD and YOLO

Object detection vs Image classification

- Image classification
    - Models sole purpose is to identify the target category
- Object detection
    - Not only classify the object but where the object is in the image

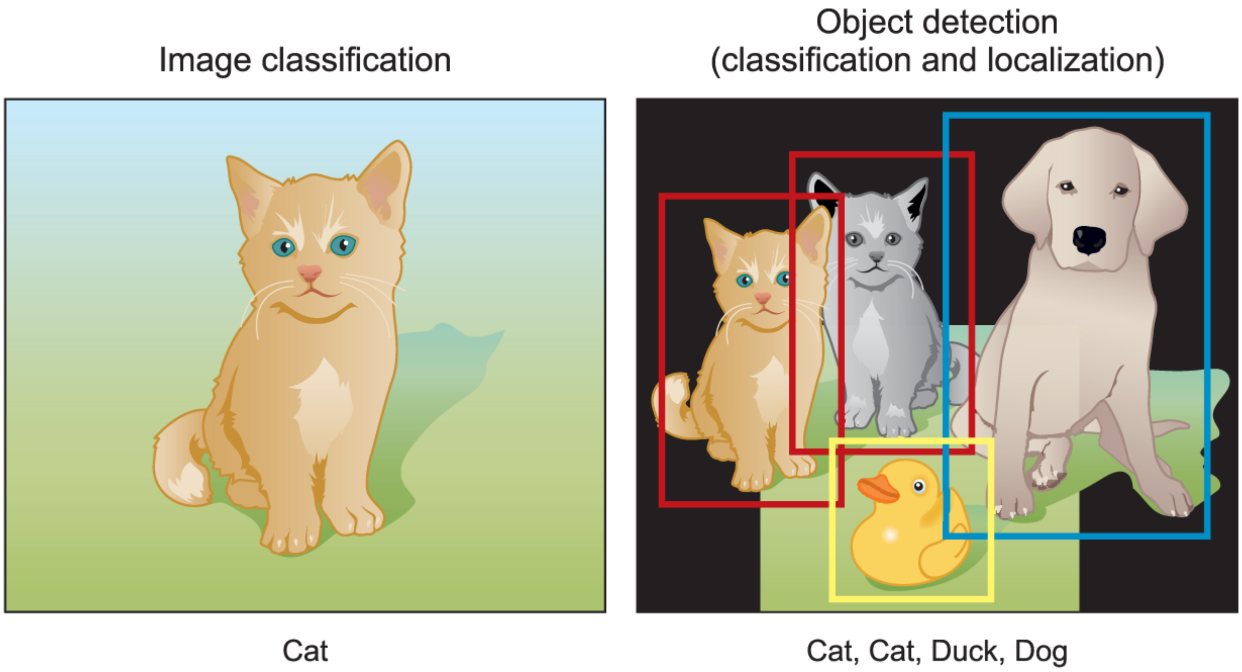


Image classification

Object detection (classification and localization)

Cat

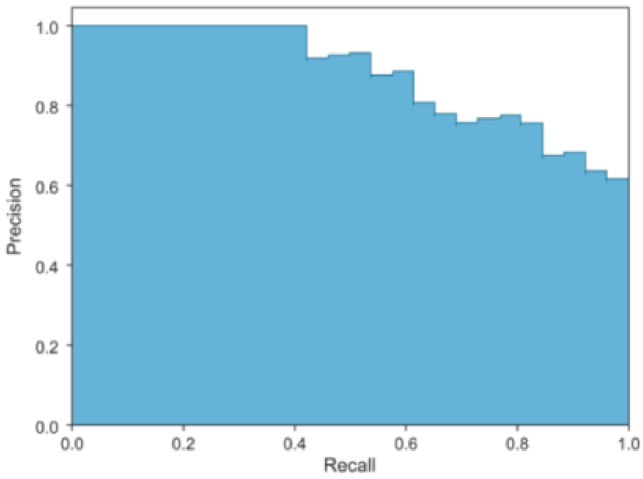
Cat, Cat, Duck, Dog

General object detection framework

- **Region proposal**
    - Algorithm or deep learning model needs to generate regions of interest (RoIs)
    - RoI (Regions of Interest)
        - Regions that the network believes might contain an object
            - Objectness score → The likelihood the system believes that there is an object or not
                - High score → above a certain threshold
                    - Considered a foreground object
                    - RoI moved to the next stage
                - Low score → below a certain threshold
                    - Considered a background object
                    - RoI discarded
                - Note: threshold is configurable
    - Output - passed in later layers for further processing
        - Large number of bounding boxes
            - Each have an objectness score
- **Feature extraction and network predictions**
    - Use a pre-trained CNN network that is used for feature extraction

        - Pre-trained classification models are used because they are trained under large datasets like image-net
        - This makes them good at extracting generic features
    - Network analyses all the regions of interest (RoI) that have been identified
        - The RoI identified has to have a high likelihood of containing an object (high objectness score)
            - Makes two predictions for each region
                - Bounding box prediction
                    - The coordinates that locate the box surrounding the object
                    - Represented as a tuple (x, y, w, h)
            - Class prediction
                - Softmax function that predicts the class probability for each object
- **Non-maximum suppression (NMS)**
    - Model has likely found multiple bounding boxes for the same object
    - NMS avoids repeated detection
        - Looks at all the boxes surrounding an object
        - Find the box that has the maximum prediction probability
        - Then suppress and eliminate the other boxes
    - NMS algorithm
        1. Discard all bounding boxes that have predictions that are less than a certain threshold → confidence threshold → its tunable
        2. Look at the remaining bounding boxes, select the bounding box with the highest probability
        3. Calculate the overlap of the remaining boxes that have the same class prediction
            - Bounding boxes that have a high overlap and predict the same classes are averaged together
            - Overlap metric is called IoU (Intersection over Union)
        4. Suppress any box that has an IoU value smaller than a certain threshold → NMS threshold
            - NMS threshold is equal to 0.5 but tunable aswell
- **Evaluation metrics**
    - Object detection has own metrics to evaluate performance
    - **Frames Per Second (FPS)**
        - Measure detection speed
        - How fast your object detection model process the frame and generate the desired output
        - 7 FPS → 7 Frames processed by the model per second

- **Intersection over Union (IoU)**
    - Measures overlap between two bounding boxes
        - The ground truth bounding box (Bground truth)
            - The ground truth is the labelled bounding box
        - The predicted bounding box (Bpredicted)
            - The predicted is the network output prediction of the bounding box
        - Applying IoU can tell whether prediction is
            - Valid (True Positive)
            - Invalid (False Negative)
        - IoU value can range between 0 (no overlap) to 1 (100% overlap)
        - $IoU = \frac{B_{ground\,truth} \cap B_{predicted}}{B_{ground\,truth} \cup B_{predicted}}$
        - IoU above a certain threshold then the prediction is considered a true positive (TP)
        - IoU below a certain threshold then the prediction is considered a false negative (FN)
- **Precision-recall curve (PR curve)**
    - Calculate precision and recall for all classes
    - Plot the PR curve



    - Detector is considered good if
        - Its precision stays high as the recall increases
- **Mean average precision (mAP)**
    - Measures network precision
    - Most common evaluation metric used in object recognition
    - % value, higher the better

- How to calculate?
  - Need to use the PR curve
    1. Get each bounding box's objectness score
    2. Calculate precision and recall
    3. Compute the PR curve for each class by varying the score threshold
    4. Calculate the AP (Average Precision): area under the PR curve. AP is computed for each class
    5. Calculate the mAP: the average AP over all the different classes

Faster R-CNN

Questions

- How does faster R-CNN extract its features?
  - Using a pre-trained CNN
    - Either ResNet or VGG architecture to extract features
      - ResNet more preferred due to being a more deeper network that is more efficient and gives better results
- How does faster R-CNN find regions of interest?
  - Anchors
    - Anchors are a collection of points that are evenly placed on the original input image
    - These points are used as references to draw reference boxes or anchor boxes
      - These anchor boxes stem from each anchor point, many can be drawn at different aspect ratios from an anchor point. These anchor boxes are used to compare to the ground truth bounding boxes.
    - Anchor points are calculated on the original input, however anchor points are used in the RPN to find background and foreground features. Before the image gets passed into the RPN the images are downscaled into feature maps due to the feature extractor.
      - This means we need to also downscale the anchor points for each feature map before the images get passed into the RPN
      - This is calculated using ratios. $\frac{img\_width}{feature\_map\_width} : \frac{feature\_map\_width}{feature\_map\_width}$. This is the same with the height. For example if the image was 16×16 and there were 16×16 anchor points per input image, then the ratio is 1:1. When the image is down sampled to 4×4 after feature extraction the ratio of anchor points is 16:4. This means that the final ratio will be 4:1. This translates to 4×4 anchor points in the final feature mapped image.
    - After the anchor points are defined the image gets passed to the RPN for region proposals.

- After the features have been extracted, the feature maps are passed the RPN (Region Proposal Network).
    - RPN is a FCN (Fully Convolutional Network) that consists of 2 layers. The first layer acts as an input, the second layer is two convolutional neural networks working in parallel
        - The first conv net in the second layer acts as a classification network.
            - Used to predict the object-ness score. Whether the feature map has a foreground object or a background object
        - The second conv net in the second layer acts as a region regressor.
            - Used to predict the (x,y,w,h) of the bounding box
    - When training the RPN the anchors are randomly sampled per image to get an even distribution of positive (foreground) and negative (background) anchors
        - For each anchor there are 9 possible anchor boxes as each anchor will have 3 aspect ratios and 3 scales. A sliding window will slide across the feature map and calculate the objectness score and the regressor
            - The objectness score is calculated by doing the IoU of the ground truth and the anchor box
            - If the anchor box is > 0.7 then the region is classified as a foreground object and labelled positive (1)
            - If the anchor box is < 0.3 then the region is classified as a background object and labelled negative (0)
            - After the output the loss is calculated using the multitask loss function.
                - The multi task loss function calculates the loss of both of the outputs of the classification part of the RPN and the bounding box regressor part of the RPN

- How does faster R-CNN discard overlapping boxes?
    - The overlapping boxes are discarded through NMS (Non Maximum Suppression). NMS avoids repeated detection by combining overlapping bounding boxes into a single box
- How does faster R-CNN evaluate scores?
    - Using FPS, Precision over recall curve, Intersection over union, mean average precision (mAP)
- What is the overall architecture of a faster R-CNN?
    - Input → ResNet / VGG (Feature extraction) → RPN (Region Proposal Network) → RoI Pooling (Region of Interest Pooling) → Fully connected network → 2 Outputs (Classification of the object and predicted bounding box)
- What does faster R-CNN achieve compared to previous models?
    - Less parameters
    - More FPS

  - Increased mAP
- What is the output of a faster R-CNN?
  - Classification of the object and the bounding box prediction outputting a tuple of (x, y, w, h
- What are the disadvantages of a faster R-CNN?
  - Traning the data may still take a while
  - Training happens in multiple phases
  - There are better methods for real time object detection
- What are the use cases for faster R-CNN
  - Detection of smaller objects possible
  - Useful for image detection