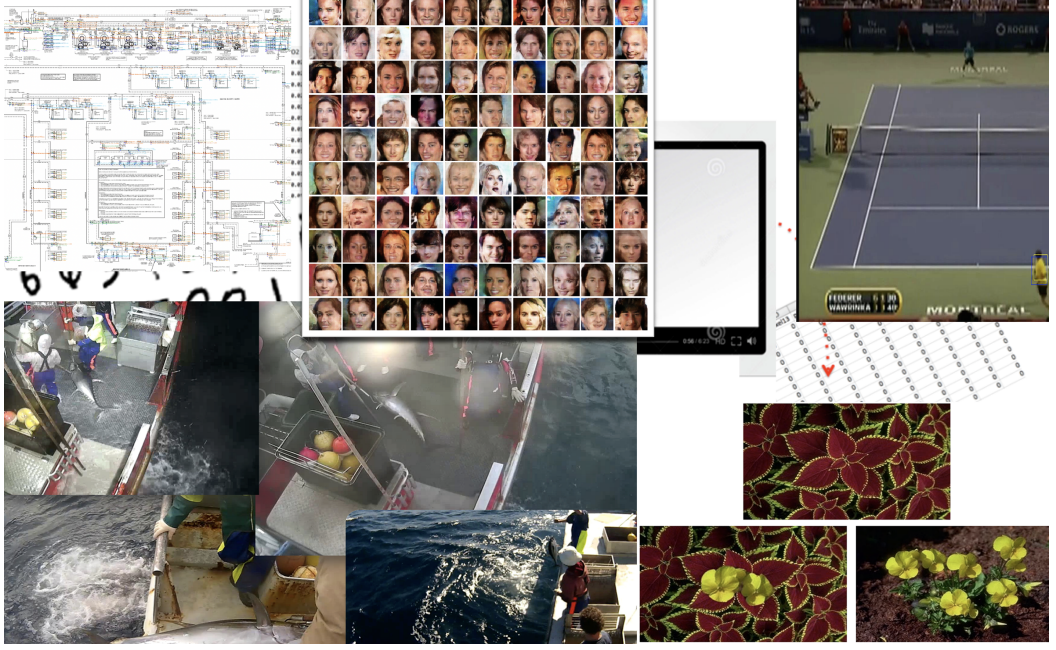
From Neural Nets to Deep Nets and Deep Convolutionial Neural Networks, Applied Computer Vision- Day 1 - 1

Professor Eyad Elyan

July 26, 2022


FEDERER
WAWRINKA
ROGERS
MONTREAL

Generic Framework

Machine Learning and Deep Learning Projects:

- Well-defined problem
- Data
- Algorithm/s

Problem

Clear and well defined problem

Data

Numerical data, text, images, videos, and other forms of structured and unstructured data

Prepare

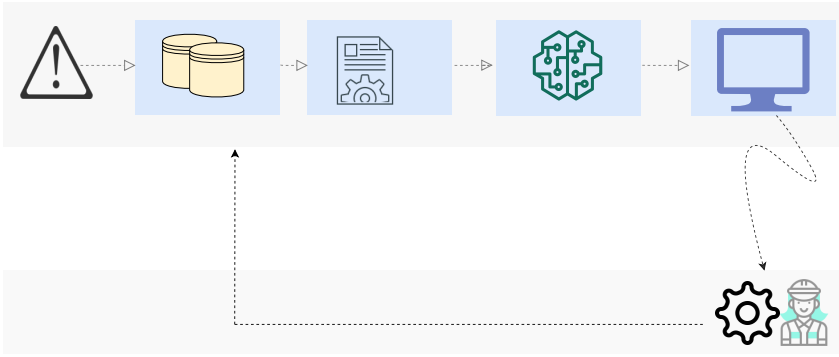
Missing values, noise, imbalance, labelling, dimension reduction, ...

Models

Train AI models, compare, validate, fine-tune, results interpretation ...

Deploy

Deploy as service (cluster, cloud), ...with all the necessary functional features and front-end



Monitoring

monitor, evaluate and re-train where necessary

Problem

Clear and well defined problem

Data

Numerical data, text, images, videos, and other forms of structured and unstructured data

Prepare

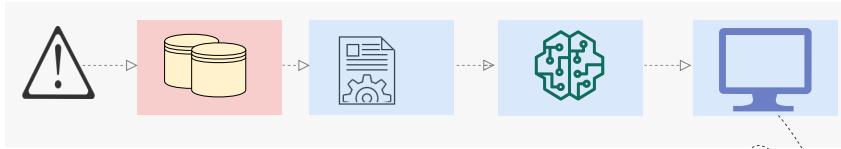
Missing values, noise, imbalance, labelling, dimension reduction, ...

Models

Train AI models, compare, validate, fine-tune, results interpretation ...

Deploy

Deploy as service (cluster, cloud), ...with all the necessary functional features and front-end



Monitoring

monitor, evaluate and re-train where necessary

Problem

Clear and well defined problem

Data

Numerical data, text, images, videos, and other forms of structured and unstructured data

Prepare

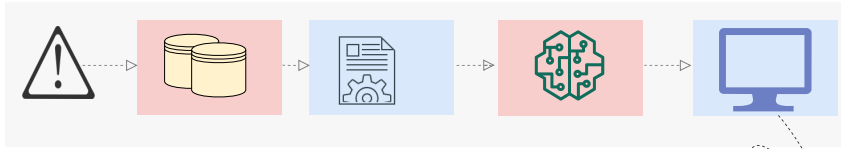
Missing values, noise, imbalance, labelling, dimension reduction, ...

Models

Train AI models, compare, validate, fine-tune, results interpretation ...

Deploy

Deploy as service (cluster, cloud), ...with all the necessary functional features and front-end



$$h(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

$$Pr(Y = 1|X) = \frac{exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n}{1 + exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X}$$

Monitoring

monitor, evaluate and re-train where necessary

Problem

Clear and well defined problem

Data

Numerical data, text, images, videos, and other forms of structured and unstructured data

Prepare

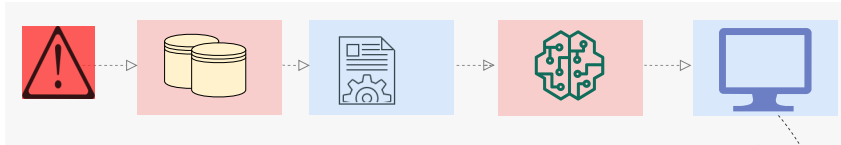
Missing values, noise, imbalance, labelling, dimension reduction, ...

Models

Train AI models, compare, validate, fine-tune, results interpretation ...

Deploy

Deploy as service (cluster, cloud), ...with all the necessary functional features and front-end



Timely

Challenging

Impactful

Multidisciplinary

$$h(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

$$Pr(Y = 1|X) = \frac{exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n}{1 + exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X}$$

Monitoring

monitor, evaluate and re-train where necessary

Plan

1. Review
2. Generic Framework
3. Computer Vision
4. Neural Networks and CNNs
   - NN Overview
   - CNNs
5. Applied Computer Vision
   - Engineering Diagrams
   - Online User's Authentication
   - Remote Inspection
6. Conclusion

Image Representation

- *MNIST*: Can computers recognise handwritten digits?

Image Representation

- *MNIST*: Can computers recognise handwritten digits?

Image Representation

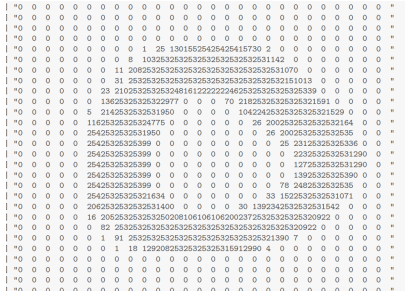


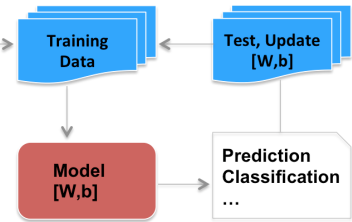
Image Representation

px0	px1	px2	...	pxn	Label
0	0	0	...	35	1
0	0	0	...	255	3
0	0	0	...	0	5
0	5	0	...	4	5
0	3	0	...	100	9
0	7	0	...	10	0
0	0	0	...	0	7
0	0	0	...	0	7
0	10	0	...	0	1
0	0	0	...	0	5
...	...	...	...	...	...

- A 28×28 image would be represented as a 748 row vector in the dataset
- A supervised learning problem (an image is the input data, the label is the outcome)

Image Representation for ML

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & \vdots & \cdots & x_{np} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ .. \\ .. \\ y_m \end{bmatrix}$$



Keras python

Image Representation for ML

Image Representation for ML

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & \vdots & \dots & x_{mn} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ .. \\ .. \\ .. \\ y_m \end{bmatrix}$$

Each image is represented in a row vector and we want to learn a function h(x) that maps an image $\mathbf{x}_i \in A$ to a class $\mathbf{y}_j \in Y$
e.g. *patient has pneumonia*

Computer Vision

Giving the computer the ability to process, analyse and **synthesise** visual content such:

- 2D Images
- Videos
- 3D Images
- Multi-modal data (2D &3D, etc..)

Computer Vision

Giving the computer the ability to process, analyse and **synthesise** visual content such:

- 2D Images
- Videos
- 3D Images
- Multi-modal data (2D &3D, etc..)

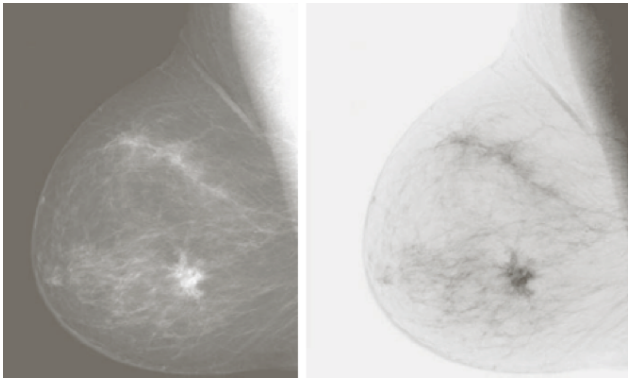
Motivation

- To achieve faster, more accurate, safer practises, ...

Basic Tasks

Basic CV Methods

- Methods include thresholding, contrast stretching, morphological operations, negation of images, ...

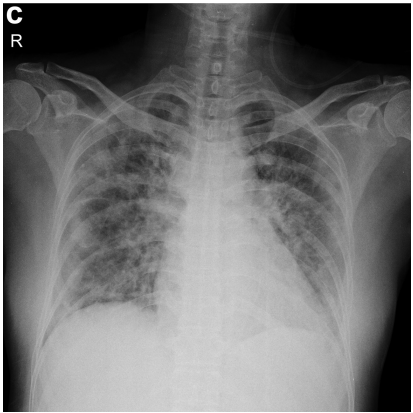


[source: Gonzalez and Woods]

More Complex Tasks

Classification

Covid-19 positive

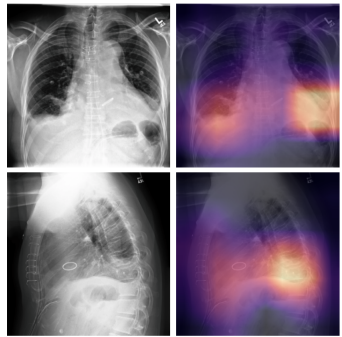


Covid-19 negative

[Source: https://github.com/ieee8023/covid-chestxray-dataset]

Object Detection/ Recognition

- Classify and localise the area of interest



(a) Frontal and lateral radiographs of the chest in a patient with bilateral pleural effusions; the model localizes the effusions on both the frontal (top) and lateral (bottom) views, with predicted probabilities $p = 0.936$ and $p = 0.939$ on the frontal and lateral views respectively.

[Source: Irvin, J., Rajpurkar, P., Ko, M., Yu, Y., Ciurea-Ilcus, S., Chute, C., Marklund, H., Haghgoo, B., Ball, R., Shpanskaya, K., Seekins, J., Mong, D. A., Halabi, S. S., Sandberg, J. K., Jones, R., Larson, D. B., Langlotz, C. P., Patel, B. N., Lungren, M. P., & Ng, A. Y. (2019). CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 590-597. https://doi.org/10.1609/aaai.v33i01.3301590]

Segmentation

- Cardiographic image segmentation[1]

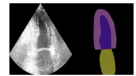


Fig. 1. An image (left) and ground truth (right) of the CAMUS competition.

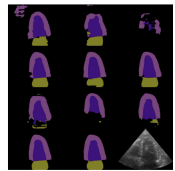


Fig. 2. Example result for datasets of the CAMUS competition. From left to right, top to bottom: UNet-VGG16, LinkNet-VGG16, FPN-VGG16, UNet-ResNet34, LinkNet-ResNet34, FPN-ResNet34, UNet-ResNet101, LinkNet-ResNet101, FPN-ResNet101, Proposed ensemble (6), Proposed ensemble (9), test image (ground truth not available)

[1] T. Dang, T. T. Nguyen, C. Francisco Moreno-García, E. Elyan and J. McCall, "Weighted Ensemble of Deep Learning Models based on Comprehensive Learning Particle Swarm Optimization for Medical Image Segmentation," 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 744-751, https://doi.org/10.1109/CEC45853.2021.9504929

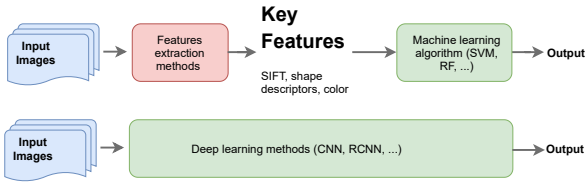
The Real Breakthrough in CV

The Real Breakthrough in CV

Traditional Versus DL-based Methods

Good Old-Fashioned Artificial Intelligence (GOFAI)[2]

- Hand-craft features (color, shapes, etc...) and heuristics (edge, line detectors, filters, etc...)
- Feed into ML algorithms



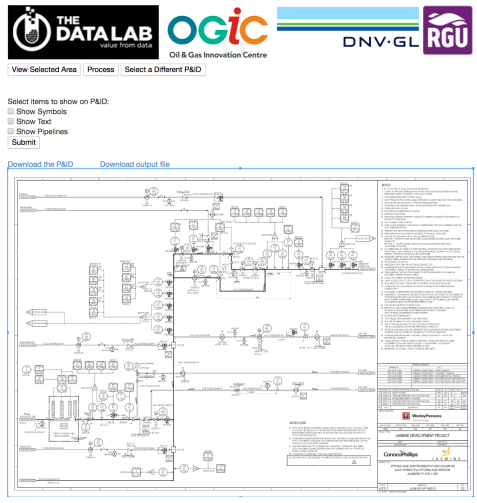
End-to-end methods for solving CV tasks [3]

[2]Haugeland, John. Artificial intelligence: the very idea. Cambridge, Mass: MIT Press, 1985.

[3]Gumbs, A.A.; Frigerio, I.; Spolverato, G.; Croner, R.; Illanes, A.; Chouillard, E.; Elyan, E. Artificial

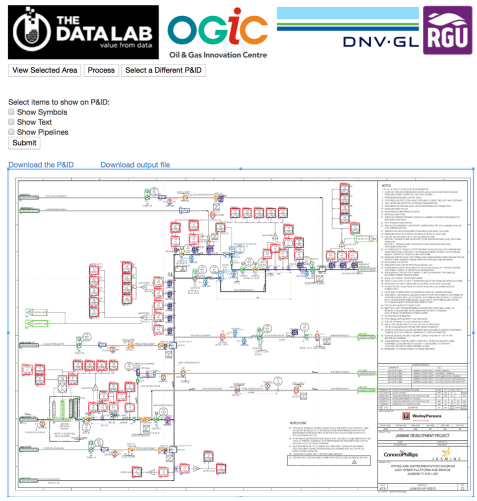
Object Detection/ Recognition

- Classify and localise the area of interest



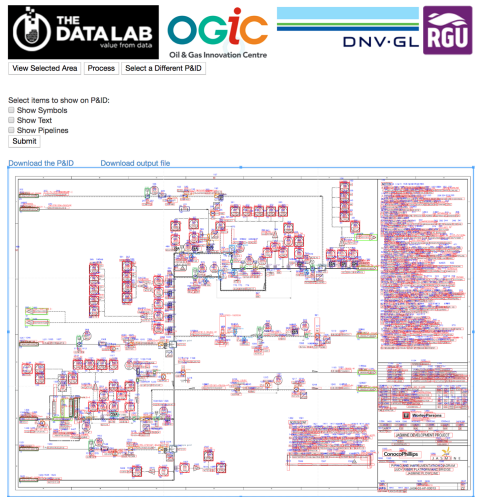
Object Detection/ Recognition

- Classify and localise the area of interest



Object Detection/ Recognition

- Classify and localise the area of interest



Object Detection/ Recognition

Neural Networks 129 (2020) 91–102

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2020 Special Issue

Deep learning for symbols detection and classification in engineering drawings

Eyad Elyan *, Laura Jamieson, Adamu Ali-Gombe

School of Computing Science and Digital Media, Robert Gordon University, UK

ARTICLE INFO

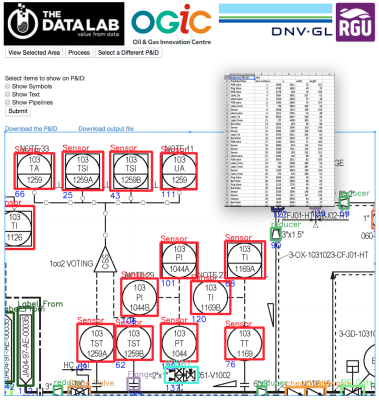
Article history:
Available online 1 June 2020

Keywords:
Deep learning
YOLO
P&ID
Engineering drawings
Symbols recognition
GANs

ABSTRACT

Engineering drawings are commonly used in different industries such as Oil and Gas, construction, and other types of engineering. Digitising these drawings is becoming increasingly important. This is mainly due to the need to improve business practices such as inventory, assets management, risk analysis, and other types of applications. However, processing and analysing these drawings is a challenging task. A typical diagram often contains a large number of different types of symbols belonging to various classes and with very little variation among them. Another key challenge is the class-imbalance problem, where some types of symbols largely dominate the data while others are hardly represented in the dataset. In this paper, we propose methods to handle these two challenges. First, we propose an advanced bounding-box detection method for localising and recognising symbols in engineering diagrams. Our method is end-to-end with no user interaction. Thorough experiments on a large collection of diagrams from an industrial partner proved that our methods accurately recognise more than 94% of the symbols. Secondly, we present a method based on Deep Generative Adversarial Neural Network for handling class-imbalance. The proposed GAN model proved to be capable of learning from a small number of training examples. Experiment results showed that the proposed method greatly improved the classification of symbols in engineering drawings.

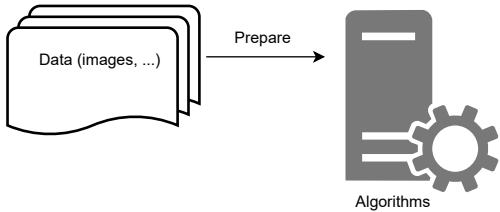
© 2020 Elsevier Ltd. All rights reserved.



Data & Algorithms

Automating complex CV tasks (classification, localisation and segmentation) requires:

- Lots of GOOD quality data
- Algorithms
- Computing power



Plan

1. Review
2. Generic Framework
3. Computer Vision
4. Neural Networks and CNNs
   - NN Overview
   - CNNs
5. Applied Computer Vision
   - Engineering Diagrams
   - Online User's Authentication
   - Remote Inspection
6. Conclusion

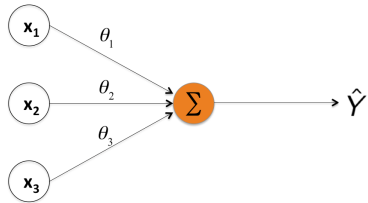
Perceptron

- Perceptron is the simplest Neural Network, proposed in 1957, and often referred to as threshold logic unit (TLU)
- Takes numbers as input, and outputs numbers
- Each input is associated with a weight as can be seen in the next slide

Perceptron

- NN building Block (consider the simple 3-input perceptron below)

$$\hat{y} = \sum_{i=1}^{m} x_i \theta_i \tag{1}$$



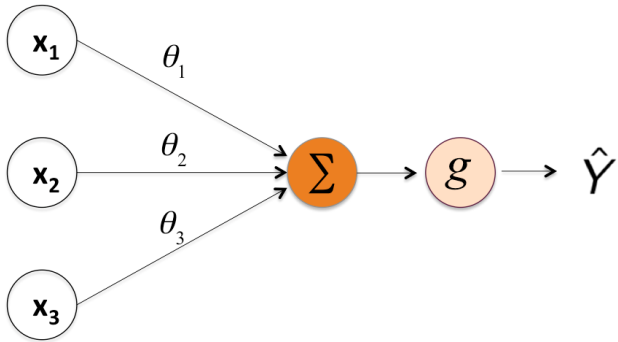
- x_i is the inputs, θ are the weights (we want to learn), the $\sum$ is the sum of the products of weights by the corresponding inputs, and $\hat{Y}$ is the output
- θ_i's are the weights to be learnt

Perceptron

- Add non-linearity

$$\hat{y} = g\Big(\sum_{i=1}^{m} x_i \theta_i\Big) \tag{2}$$

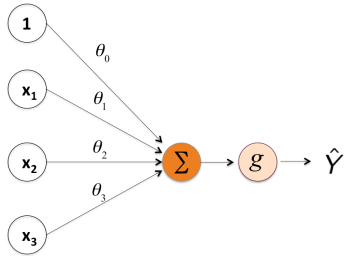
- Often referred to as activation function or step function (g is a non-linear function)



Perceptron

- θ_0 is a bias
- Add non-linearity

$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \theta_i\right) \tag{3}$$



- Equation 3 is often written as $g\left(\theta_0 + X^T\theta\right)$
- θ is sometimes denoted by w

Perceptron

- Consider the perceptron above and suppose

$$x = \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix}, \theta = \begin{bmatrix} 0.0 \\ 0.1 \\ 0.2 \end{bmatrix}$$

- Following Equation3), we multiply x^T by θ

$$\begin{bmatrix} 3 & 4 & 2 \end{bmatrix} \times \begin{bmatrix} 0.0 \\ 0.1 \\ 0.2 \end{bmatrix}$$

$$= (3 \times 0.0) + (4 \times 0.1) + (2 \times 0.2) = 0.8$$

Perceptron

- More generally

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \theta = \begin{bmatrix} \theta_1 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

- Again, based on Equation3), we multiply X^T by θ

$$\begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} \times \begin{bmatrix} \theta_1 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

$$= x_1\theta_1 + x_2\theta_2 + \cdots + x_m\theta_m$$

- Notice that x could be of any dimension

Perceptron

- Notice that a single *Perceptron* can be used for linear binary classification (by computing linear combination of the inputs) and if the results exceed a threshold, it outputs a positive class

Perceptron

- Notice that a single *Perceptron* can be used for linear binary classification (by computing linear combination of the inputs) and if the results exceed a threshold, it outputs a positive class
- The final values for the weights (θs, or ws) are found during the training

Perceptron

- Notice that a single *Perceptron* can be used for linear binary classification (by computing linear combination of the inputs) and if the results exceed a threshold, it outputs a positive class
- The final values for the weights (θs, or ws) are found during the training
- A *Perceptron* is made of a single layer (hardly used this way in today's AI applications)

Perceptron

- Notice that a single *Perceptron* can be used for linear binary classification (by computing linear combination of the inputs) and if the results exceed a threshold, it outputs a positive class
- The final values for the weights (θs, or ws) are found during the training
- A *Perceptron* is made of a single layer (hardly used this way in today's AI applications)
- When all neurons in the layer are connected to every neuron in the previous layer (in perceptron case, to the input layer), then the layer is called *fully connected layer* or Dense Layer

Perceptron

- Notice that a single *Perceptron* can be used for linear binary classification (by computing linear combination of the inputs) and if the results exceed a threshold, it outputs a positive class
- The final values for the weights (θs, or ws) are found during the training
- A *Perceptron* is made of a single layer (hardly used this way in today's AI applications)
- When all neurons in the layer are connected to every neuron in the previous layer (in perceptron case, to the input layer), then the layer is called *fully connected layer* or Dense Layer
- Input neurons are used to input the data (inputs) and with an extra bias feature called *bias neuron* is added

Perceptron

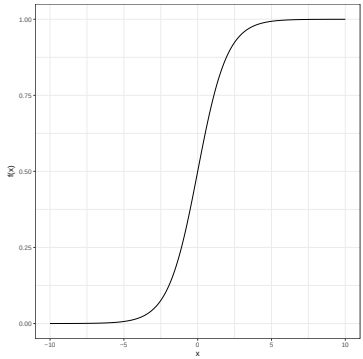
- Notice that a single *Perceptron* can be used for linear binary classification (by computing linear combination of the inputs) and if the results exceed a threshold, it outputs a positive class
- The final values for the weights (θs, or ws) are found during the training
- A *Perceptron* is made of a single layer (hardly used this way in today's AI applications)
- When all neurons in the layer are connected to every neuron in the previous layer (in perceptron case, to the input layer), then the layer is called *fully connected layer* or Dense Layer
- Input neurons are used to input the data (inputs) and with an extra bias feature called *bias neuron* is added

Activation Functions

- g, Used to process information and pass it through the network
- Transforms the information in a unit (neuron) into a signal
- Different kinds of activation functions
- Most common in Deep Learning is *Relu* activation function

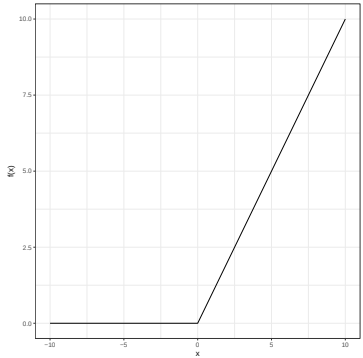
Sigmoid Activation Function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$



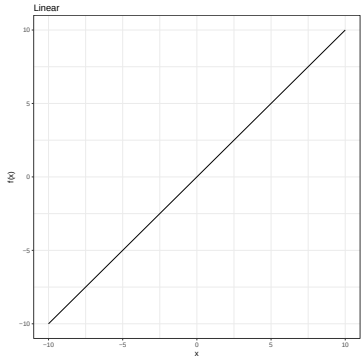
Relu Activation Function

$$f(x) = max(0, x) \tag{5}$$



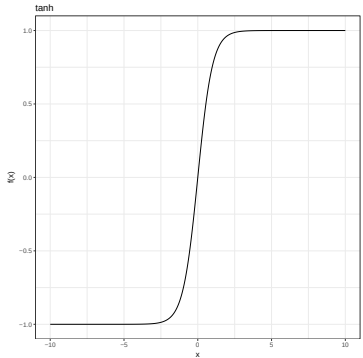
Other activation function **Linear**

$$f(x) = x \tag{6}$$



Other activation function **Hyperbolic Tangent**

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{7}$$



Cost Function

- A cost function is the measure of the error of the model (the difference between the model's prediction $f_\theta(x_i)$, and the actual value y_i
- A cost function can be the L_2 Loss, defined as

$$\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

- n is the number of instances in the dataset (e.g. images)
- θ the weights learnt by the model
- y_i is the actual value, and $f_\theta(x_i))^2$ is the model output

Cost Function

- Loss functions, cost and objects are all sused to describe the way evaluate the performance of the model, common loss functions/ evaluation methods include:

- Mean Squared Error (MSE) $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Absolute Error Loss $L = |y - \hat{y}|$
- Binary Classification Loss

$$\frac{FP + FN}{TP + TN + FP + FN}$$

(FP and FN are the incorrectly classified examples)

Cross Entropy Loss

- Cross-entropy loss or log loss is used to evaluate the performance of a classification model whose output between 0 and 1 (indicating a probability). Small loss values indicate that the model's output is close to the actual output and is defined as follows

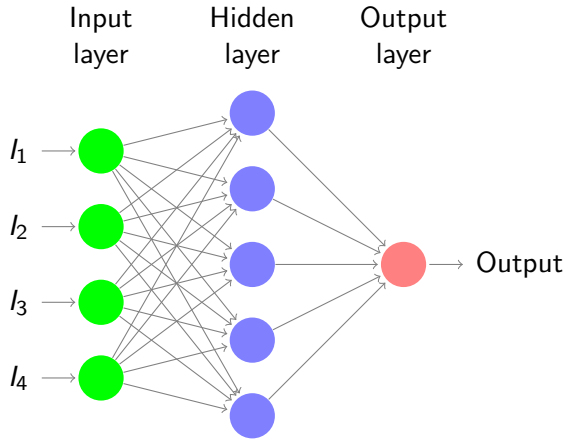
$Loss = \sum_{i=1}^{n} -y^{(i)} log(p) - (1 - y^{(i)}) log(1 - p)$, where p is the predicted probability

MLP

- MLP is made of one input layer, one or more hidden layer, and an output layer
- All layers except the output layer have a bias feature added to them and fully connected to the next layer
- A *deep neural network* is a NN with many hidden layers while a shallow network is often made of one or two hidden layers

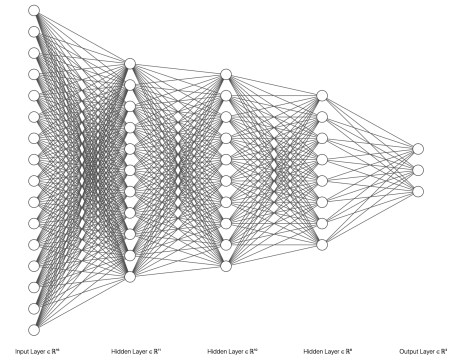
Neural Network

- Activation function,
- A network topology
- Training algorithm



Training Neural Networks

- Training NN continues to be challenging task till 1986 when David Rumehlhart, Geoffrey Hinton and Ronald Williams developed the *backpropagation* algorithm



Training Neural Networks

- The *backpropagation algorithm* continues to be used in training neural networks

Training Neural Networks

- The *backpropagation algorithm* continues to be used in training neural networks
- It is based on Gradient Descent and aims at computing the gradients of the network. The data is passed through the network in two passes (one forward, and one backward), and the algorithm is then being able to calculate the gradients of the network's error

Training Neural Networks

- The *backpropagation algorithm* continues to be used in training neural networks
- It is based on Gradient Descent and aims at computing the gradients of the network. The data is passed through the network in two passes (one forward, and one backward), and the algorithm is then being able to calculate the gradients of the network's error
- The method finds how each connection weight (θ) and each bias should be adjusted in order to reduce the error

Training Neural Networks

- The *backpropagation algorithm* continues to be used in training neural networks
- It is based on Gradient Descent and aims at computing the gradients of the network. The data is passed through the network in two passes (one forward, and one backward), and the algorithm is then being able to calculate the gradients of the network's error
- The method finds how each connection weight (θ) and each bias should be adjusted in order to reduce the error
- Once the gradients are obtained, the regular Gradient Descent step is performed, and the process is repeated until the network converges to a solution

Gradient Descent

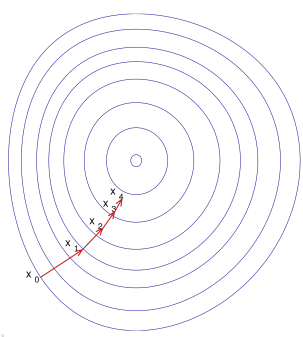
- The main idea of Gradient Descent is to iteratively modify/ tweak parameters in order to minimise the *cost function*

Gradient Descent

- The main idea of Gradient Descent is to iteratively modify/ tweak parameters in order to minimise the *cost function*
- Recall, that the aim of training is to find the right values for θs which are initialised randomly, the Gradient Descent taking repeated steps to desrease the cost function (e.g. MSE or thers) until the algorithm converges to minimum

Gradient Descent

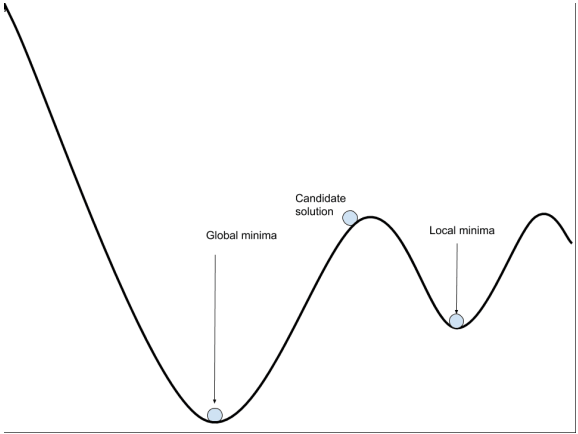
- The main idea of Gradient Descent is to iteratively modify/ tweak parameters in order to minimise the *cost function*
- Recall, that the aim of training is to find the right values for θs which are initialised randomly, the Gradient Descent taking repeated steps to desrease the cost function (e.g. MSE or thers) until the algorithm converges to minimum
- A key parameter for the algorithm is the size of the step (learning rate))



[https://en.wikipedia.org/wiki/Gradient_descent]

Gradient Descent

- Algorithm convergence depends on where the search starts
- Fortunately, Gradient Descent is guaranteed to approach a point close to global minimum, providing enough training time is given, and the learning rate is not too high



[https://machinelearningmastery.com/why-training-a-neural-network-is-hard/]

As mentioned earlier Gradient Descent plays the main role in finding the right weights for the neural networks (θs), so imagine that we have a dataset of 600 instances (rows):

As mentioned earlier Gradient Descent plays the main role in finding the right weights for the neural networks (θs), so imagine that we have a dataset of 600 instances (rows):

- It handles one batch of instances at a time (for example 32 rows each batch), and it goes through the full training set multiple times. Each pass (e.g. all the training set is passed forward and backward through the network) is called *epoch*

How NN Works

As mentioned earlier Gradient Descent plays the main role in finding the right weights for the neural networks (θs), so imagine that we have a dataset of 600 instances (rows):

- It handles one batch of instances at a time (for example 32 rows each batch), and it goes through the full training set multiple times. Each pass (e.g. all the training set is passed forward and backward through the network) is called *epoch*

- Forward Pass: Each batch is passed to the network input layer which passes it to the first hidden layer. The algorithm then computes the output of all neurons in this layer, and the result is passed to the next layer and so on, until we get the output of the last layer

How NN Works

As mentioned earlier Gradient Descent plays the main role in finding the right weights for the neural networks (θs), so imagine that we have a dataset of 600 instances (rows):

- It handles one batch of instances at a time (for example 32 rows each batch), and it goes through the full training set multiple times. Each pass (e.g. all the training set is passed forward and backward through the network) is called *epoch*
- Forward Pass: Each batch is passed to the network input layer which passes it to the first hidden layer. The algorithm then computes the output of all neurons in this layer, and the result is passed to the next layer and so on, until we get the output of the last layer
- All results in the forward pass are preserved, because they are needed for further computation

Once the forward pass is completed, the algorithm calculates the output's error:

- This is achieved by computing the loss function that compares the desired output and the actual output of the network

Once the forward pass is completed, the algorithm calculates the output's error:

- This is achieved by computing the loss function that compares the desired output and the actual output of the network
- The algorithm then computes how much each output connection contributed to the error. This is done based on *chain rule*

Once the forward pass is completed, the algorithm calculates the output's error:

- This is achieved by computing the loss function that compares the desired output and the actual output of the network
- The algorithm then computes how much each output connection contributed to the error. This is done based on *chain rule*
- Backward Pass: The algorithm then measures how much of these error contributions came from each connection in the previous layer (using chain rule, calculus), and working backward to reach the input layer

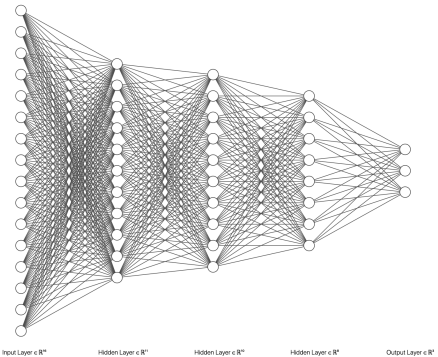
Once the forward pass is completed, the algorithm calculates the output's error:

- This is achieved by computing the loss function that compares the desired output and the actual output of the network
- The algorithm then computes how much each output connection contributed to the error. This is done based on *chain rule*
- Backward Pass: The algorithm then measures how much of these error contributions came from each connection in the previous layer (using chain rule, calculus), and working backward to reach the input layer
- The final step is to tweak all connection weights (θs) in the network using the computation above (gradients error)

Network Hyperparameters

Many parameters needs to be tweaked to ensure achieving the best possible results:

- Number of hidden layers
- Number of neurons per layer
- Activation function
- Number of epochs (training for many epochs can lead to overfitting)
- And certainly other parameters



Network Hyperparameters

One of the most common and perhaps intuitive approaches to try various combination of parameters, for example our network with the following combination of parameters:

Network Hyperparameters

One of the most common and perhaps intuitive approaches to try various combination of parameters, for example our network with the following combination of parameters:

- Number of hidden layers options (1,2,3, 4)
- Number of neurons to range between (5 to 100)
- Learning rate to range between (0.0001 to 0.01)
- and so on

Network Hyperparameters

One of the most common and perhaps intuitive approaches to try various combination of parameters, for example our network with the following combination of parameters:

- Number of hidden layers options (1,2,3, 4)
- Number of neurons to range between (5 to 100)
- Learning rate to range between (0.0001 to 0.01)
- and so on

Fortunately, *python* provides various mechanisms to simplify this using GridSearchCV and RAndomizedSearchCV. Note, that using such method can cause the training to last for hours, subject to the size of the dataset among other factors

Number of Hidden Layers in NN

The number of hidden layers is critical, however there is a simple rule of thumb that is commonly used:

Number of Hidden Layers in NN

The number of hidden layers is critical, however there is a simple rule of thumb that is commonly used:

- For various types of problems, you can start with a single hidden layer and this should give you reasonable results, networks of this type are often referred to as shallow networks

Number of Hidden Layers in NN

The number of hidden layers is critical, however there is a simple rule of thumb that is commonly used:

- For various types of problems, you can start with a single hidden layer and this should give you reasonable results, networks of this type are often referred to as shallow networks
- Shallow networks with one single hidden layer and enough neurons can model complex functions

Number of Hidden Layers in NN

The number of hidden layers is critical, however there is a simple rule of thumb that is commonly used:

- For various types of problems, you can start with a single hidden layer and this should give you reasonable results, networks of this type are often referred to as shallow networks
- Shallow networks with one single hidden layer and enough neurons can model complex functions
- Consider the *mnist* digits dataset, you can achieve up to 97% with a single hidden layer

Number of Hidden Layers in NN

The number of hidden layers is critical, however there is a simple rule of thumb that is commonly used:

- For various types of problems, you can start with a single hidden layer and this should give you reasonable results, networks of this type are often referred to as shallow networks
- Shallow networks with one single hidden layer and enough neurons can model complex functions
- Consider the *mnist* digits dataset, you can achieve up to 97% with a single hidden layer
- For more complex problem, deeper networks provides a better option (e.g. adding more hidden layers)

Number of Neurons per Hidden Layers

Again, there are few rules to follow when considering the number of neurons in your network various layers

- For hidden layers, it is common to determine the number of neurons to form a pyramid, with fewer and fewer neurons at each layer (check the network's number of neurons for the fashion mnist dataset)
- You can always tweak the number of neurons at the different layers to improve performance, however, it is always better to start with enough number of neurons, and reduce this number accordingly

Learning Rate

- Learning rate is very important parameter, usually it is recommended to start with a very low learning rate e.g. 10^{-5}, and gradually increasing it up to a larger values e.g. 1 (after few hundreds of training iterations)

Optimizer

- Optimizer, we have already learnt about gradient descent, but other options such as *adam* optimizer is much more common (just a parameter settings in your code)

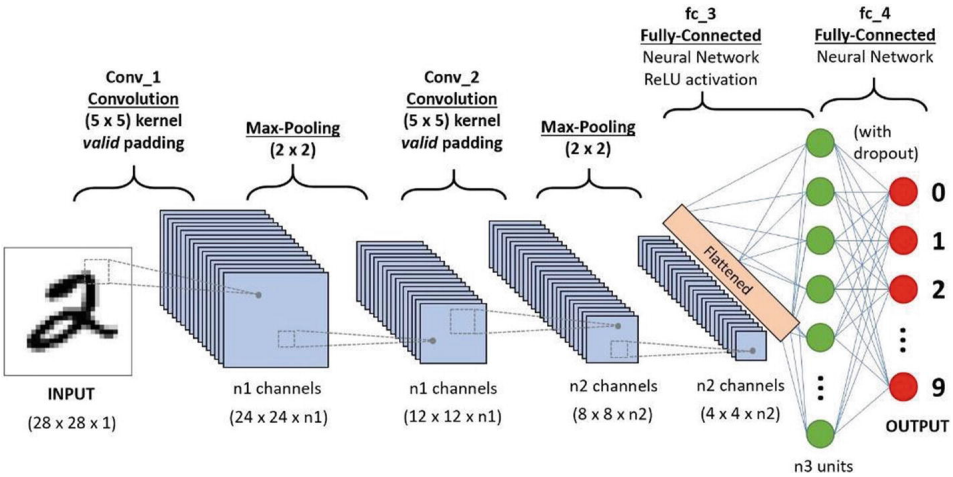
Batch Size

- Batch size can also have significant impact on the network model's performance and the training time. If you have a powerful machine with GPU, then larger batch size is recommended (processing more instances per second). However, prominent AI researchers recommend a batch size no larger than 32 (LeCun)

Convolutional Neural Network

- CNNs have been used in image recognition since the 1980s. However, and due to the increase in computational power, advances in research in ANN, and the availability of large volumes of data in the past few years CNNs have outperformed humans in many tasks

Application Convolutional Neural Network



[Image Source: El-Amir H., Hamdy M. (2020) Convolutional Neural Network. In: Deep Learning Pipeline. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-5349-6_11]

Application Convolutional Neural Network

CNNs have been successfully applied across wide range of domains:

- Image recognition (CNNs are very powerful across various machine vision problems e.g. image classification, objects detection from images, etc...)
- Applied successfully to voice recognition
- Text and natural language processing (NLP)
- and others

Why CNN

The question is why don't we just use deeper neural networks instead of convolutional neural networks (e.g. just add more layers to the NN we learnt in the previous session?):

Why CNN

The question is why don't we just use deeper neural networks instead of convolutional neural networks (e.g. just add more layers to the NN we learnt in the previous session?):

- Consider an image recognition task, say our image dataset has images of size 100 $\times$ 100
- Input layer will have 10,000 neurons
- If we set the first layer to have 1000 neuron, then this means a total of 10 million connection (Why?)
- Imagine adding more layers, the number of parameters will grow hugely
- CNN instead, solve this problem using partially connected layers and weight sharing (as we will see later)

What is an Image?

- Recall: An image is a two dimensional function $f(x, y)$ where x and y are the spatial coordinates and the value of f at any location is called the intensity value or the gray level value (0 to 255)

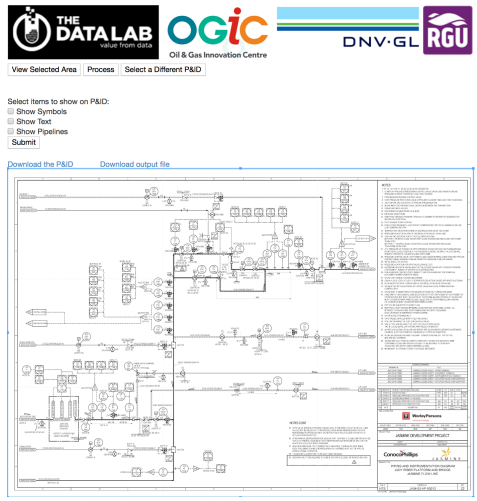
[Digital picture produced in 1921, source Gonzalez and Wood]

Plan

1. Review
2. Generic Framework
3. Computer Vision
4. Neural Networks and CNNs
   - NN Overview
   - CNNs
5. Applied Computer Vision
   - Engineering Diagrams
   - Online User's Authentication
   - Remote Inspection
6. Conclusion

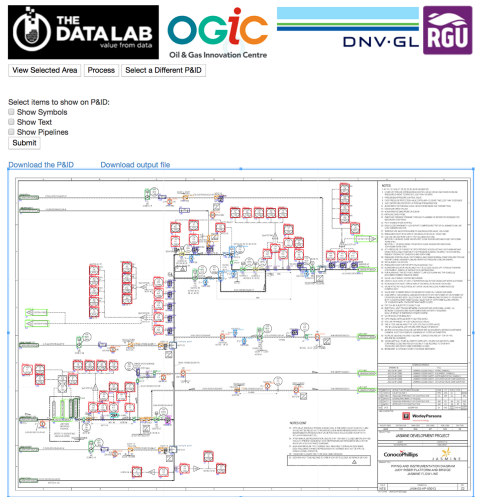
Engineering Diagrams

- Classify and localise the area of interest



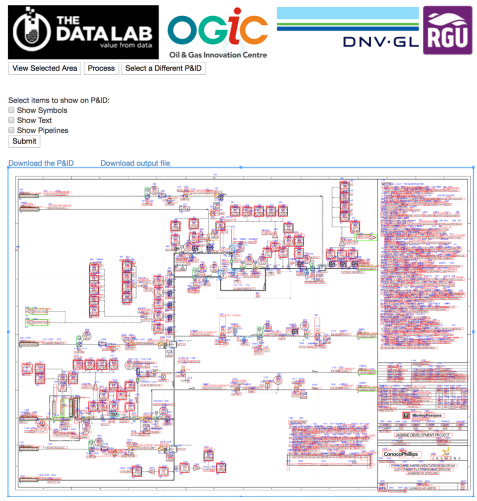
Engineering Diagrams

- Classify and localise the area of interest



Engineering Diagrams

- Classify and localise the area of interest



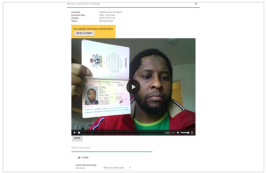
On-line User's Authentication

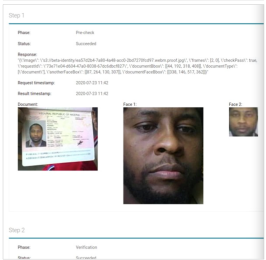
Research

- Deep Convolutional Neural Networks
- Object Detection
- Ensemble Learning

Application

- Remote user's authentication
- Value $\approx$ *£180K*
- *Aug-2019* to *Jul-2021*





Partners- In collaboration with Mintra Group - Aberdeen, UK

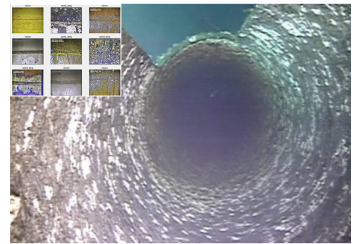
Remote Inspection

Research

- Ultrasonic image analysis using Deep Learning-based methods

Application

- Remote inspection and structural integrity of offshore assets
- KTP with AISUS
- Value $\approx$*£175K*



- *Start date June-2021*

Partners- In collaboration with AISUS LTD - Aberdeen, UK

Plan

1. Review
2. Generic Framework
3. Computer Vision
4. Neural Networks and CNNs
   - NN Overview
   - CNNs
5. Applied Computer Vision
   - Engineering Diagrams
   - Online User's Authentication
   - Remote Inspection
6. Conclusion

Conclusion

- Underlying knowledge
- Tools and technologies

- Learn as you go, start simple with structured data and move to more complex data and tasks

Thank You

@ElyanEyad

https://www3.rgu.ac.uk/dmstaff/elyan-eyad