**Why do we prefer CNN over feed-forward for image classification?**

**What is a Convolutional Neural Network and how does it work?**

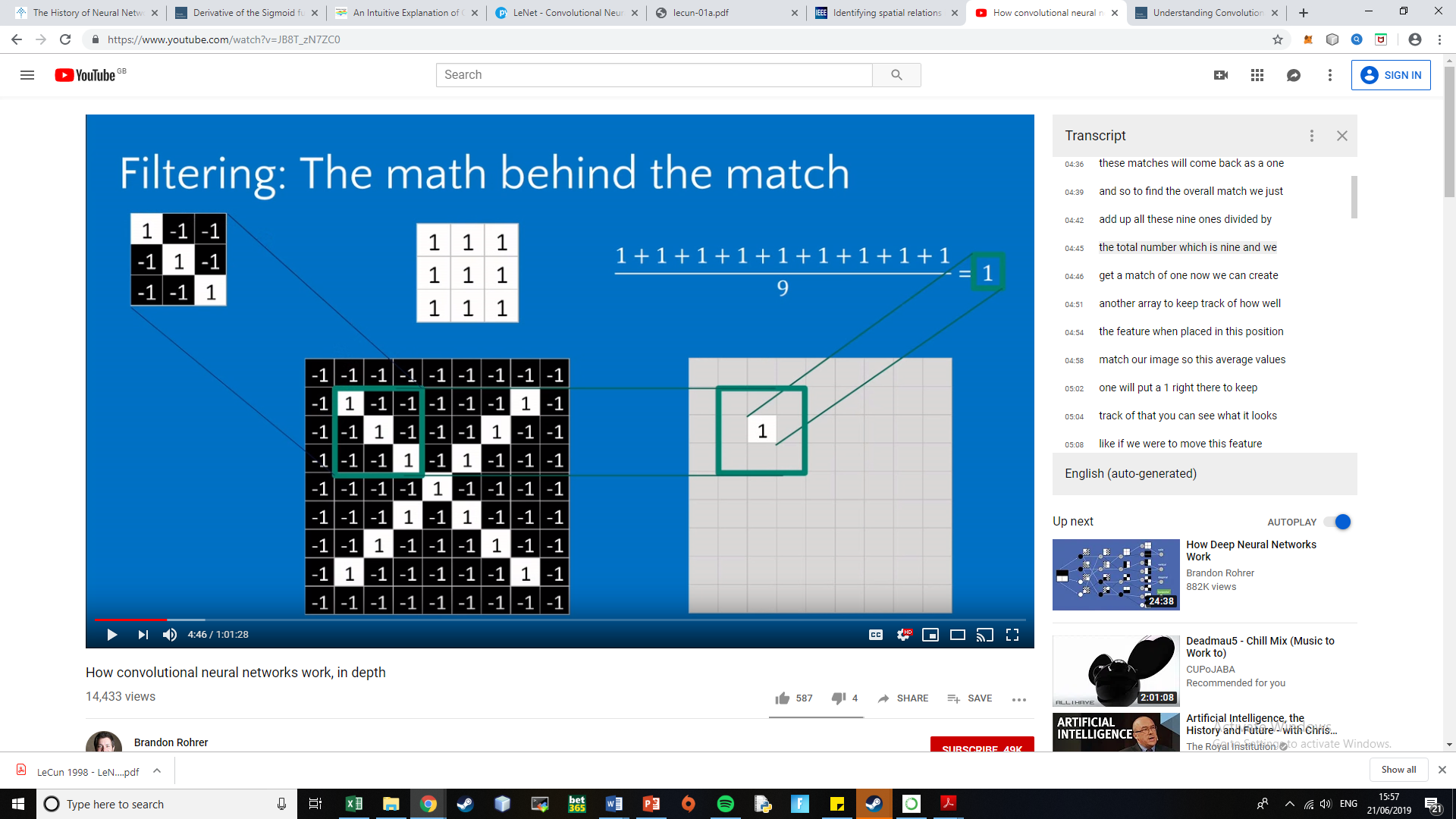
Convolutional Neural Networks (ConvNet/CNN) are a category of Neural Networks whom have proven to perform very efficiently in several tasks relating to image recognition and classification, such tasks include; but are not limited to, human face identification, powering vision in robots, and object recognition and classification for self-driving cars.

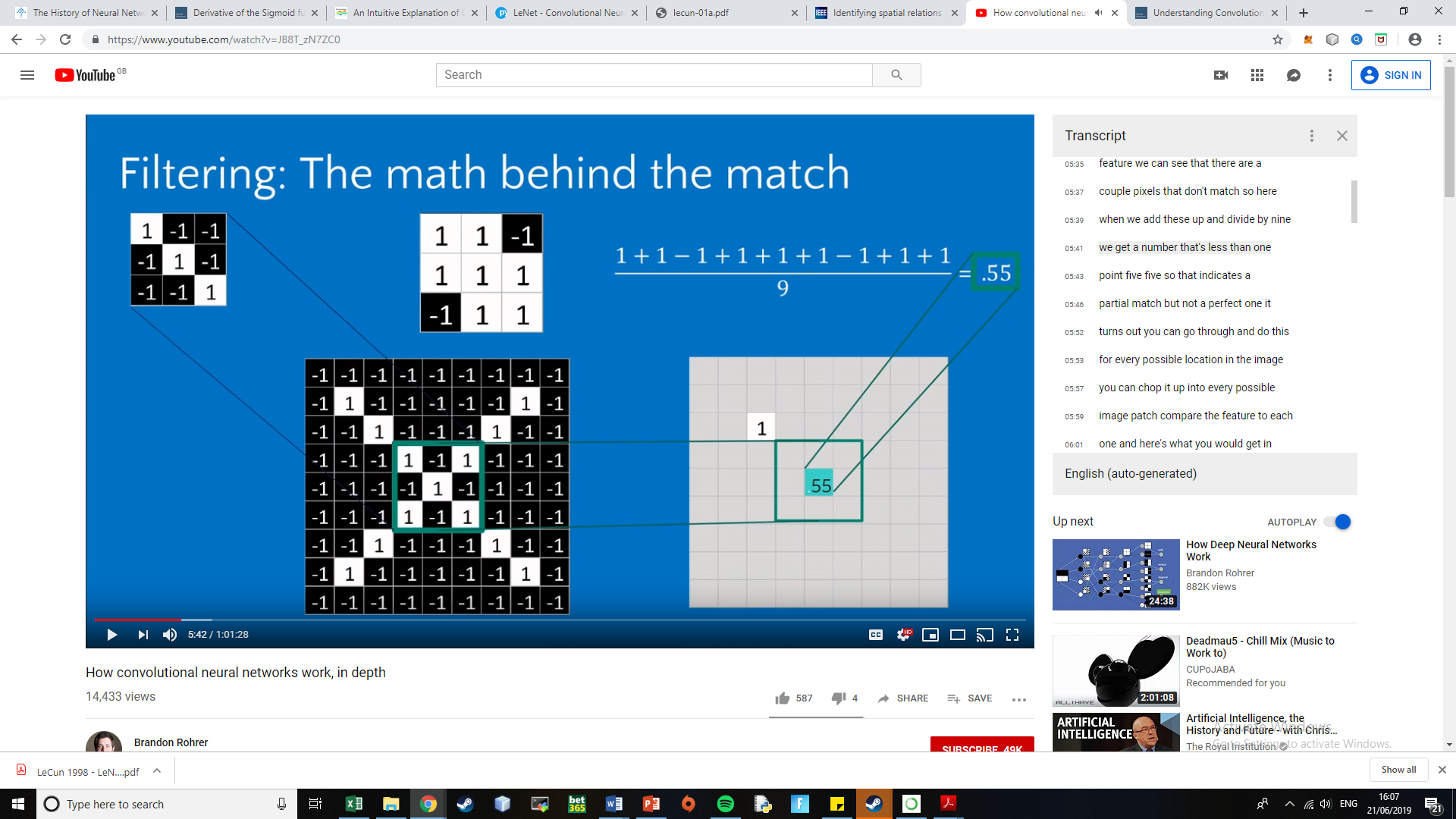
The aim of a CNN is to learn the patterns and building blocks that make up our input data (the images of the railway tracks). For instance, the first level of the network may learn low-level features, things like line segments that are at different angles and then at subsequent layers these will get built into higher-level features, things like faces, elements of cars, or in our case, elements of a railway track

We want the network to be able to classify our images into those that require track maintenance and those that do not – however this is not entirely straightforward due to variations between the content of the input images – different levels of zoom, rotations of the camera angle, different levels of lighting (if some of the images were taken at different times of the day, for example). A human would have no problem in classifying these images regardless of these variations, and we would like our CNN to be able to do it too.

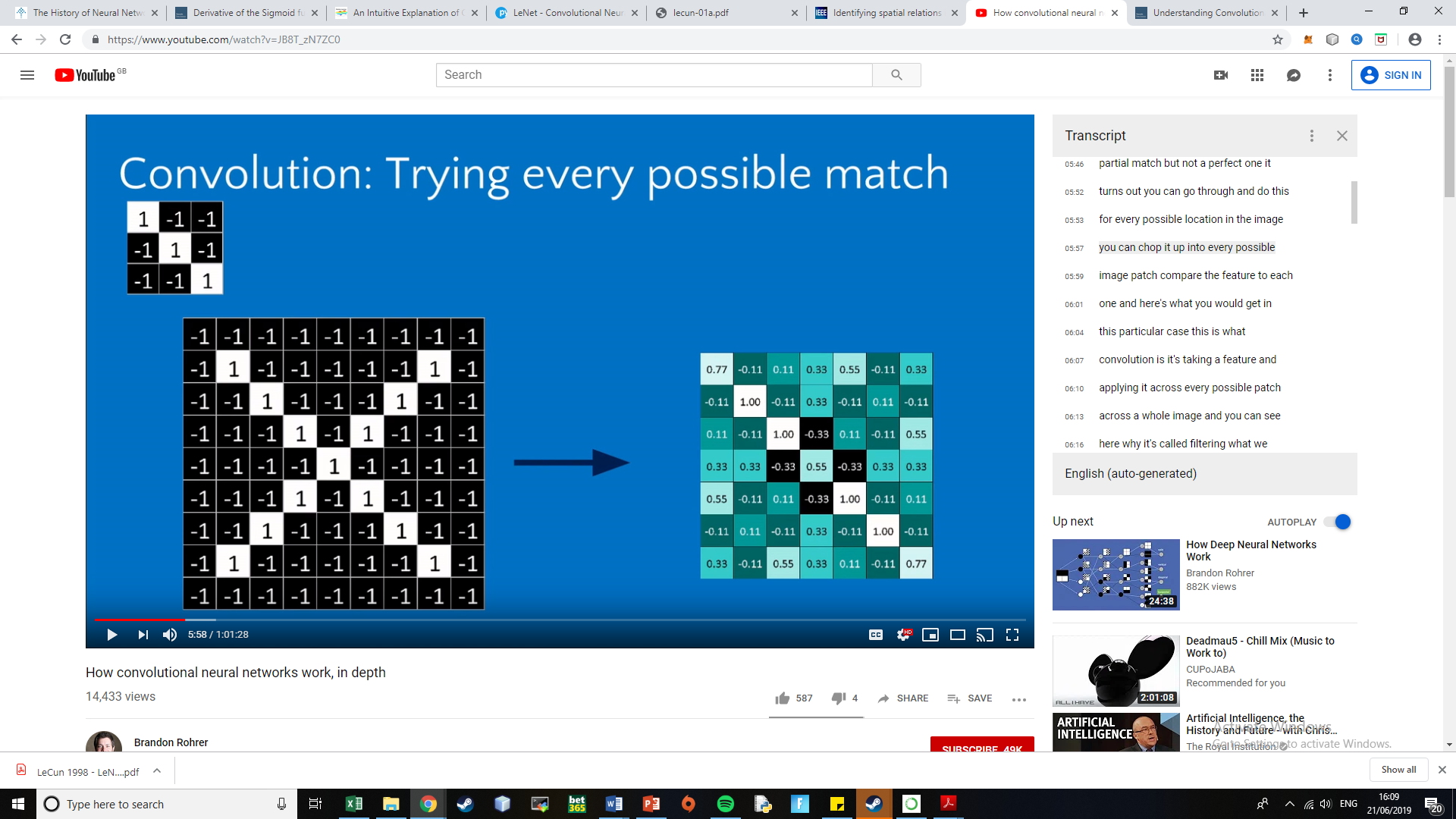
CNN aims to match pieces of the image – so you can move these pieces around the image a little bit but as long as they match then the overall image is still a pretty good match – we’ll call these pieces ‘features’. Filtering is the method behind finding these matches.

We can align the feature up on the image segment you’re concerned with, multiply pixel by pixel (if theres a perfect match we’ll get a 1, if not we’ll get -1), add up the values, and then divide by the total number of pixels to obtain a value that indicates how strong the match is. For example, a 3x3 filter where each and every pixel matches perfectly, when added together we’d get 1+1+1+1+1+1+1+1+1 = 9, then divide by the number of pixels 9/9 = 1, and so we’d get a perfect match between the feature and that segment of the image



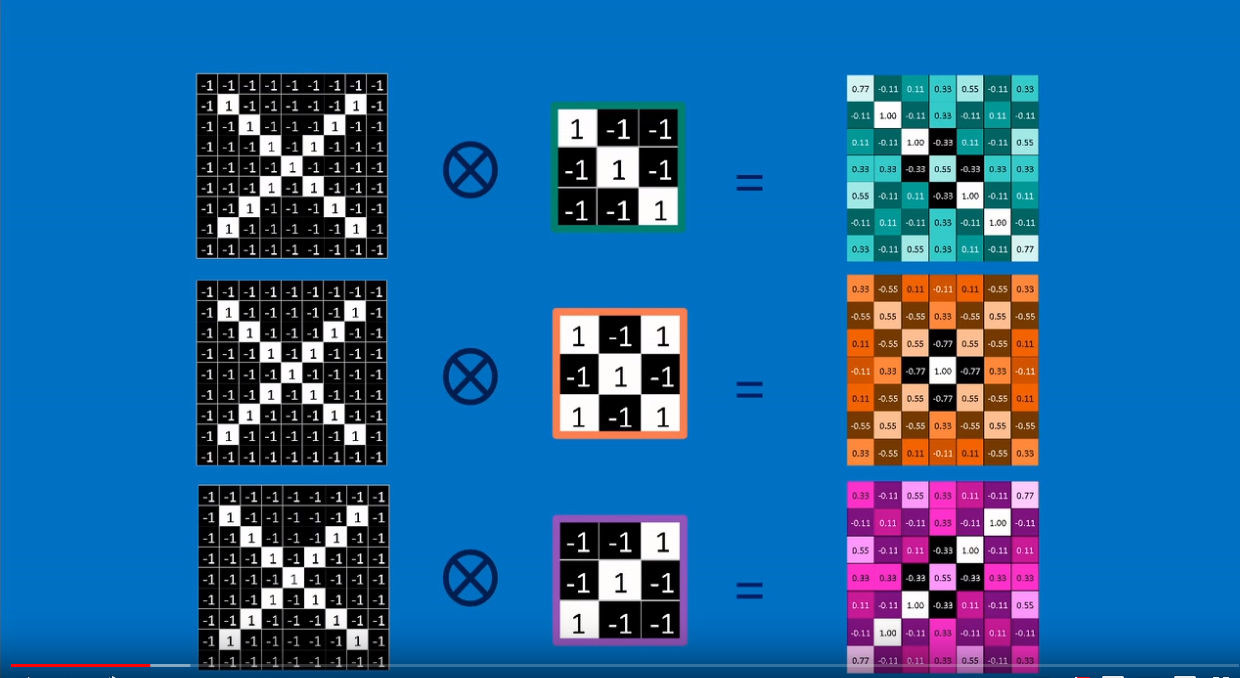


We can do this for every possible location on the image – we can slide the filter over every possible 3x3 segment of the image and calculate the scores to get the convolved feature below



We can see that we get strong positive values where the feature is present (along the left diagonal) and lesser values where the feature is not present – so it’s a filtered version of the original image that shows where the feature matches

We can repeat this with other features, for example below, we have three filtered versions of the original image.



‘Convolved feature’ or ‘Feature map’

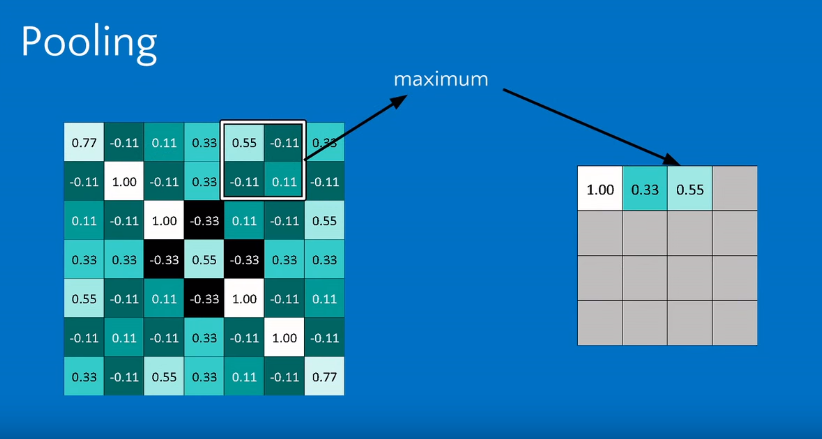
This is what a Convolution layer in a CNN does, it transforms an image into a stack of filtered images

**Why do we need a convolutional layer?**

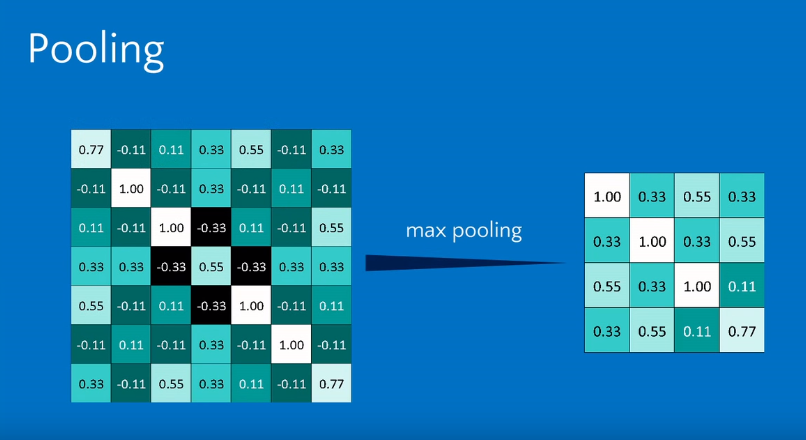
**What size filters should we use?**

Max Pooling is done to shrink the size of the image stack:

* Pick a window size (usually 2 or 3)
* Pick a stride (usually 2)
* Walk your window across your filtered images
* From each window, take the maximum value



We repeat this for all segments of the image, and when we are finished we obtain a ‘shrunken’ version of the filtered image – a smaller image that is still similar to the original



Again, we can do this with each of the filtered images so that our stack of images becomes a stack of smaller images

**Why do we need pooling?**

**What size window should we use?**

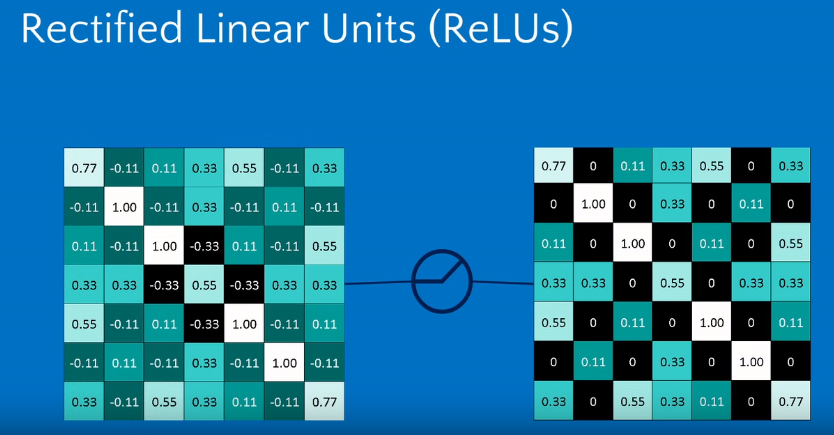
**What size stride?**

**Alternative to max pooling – average pooling??**

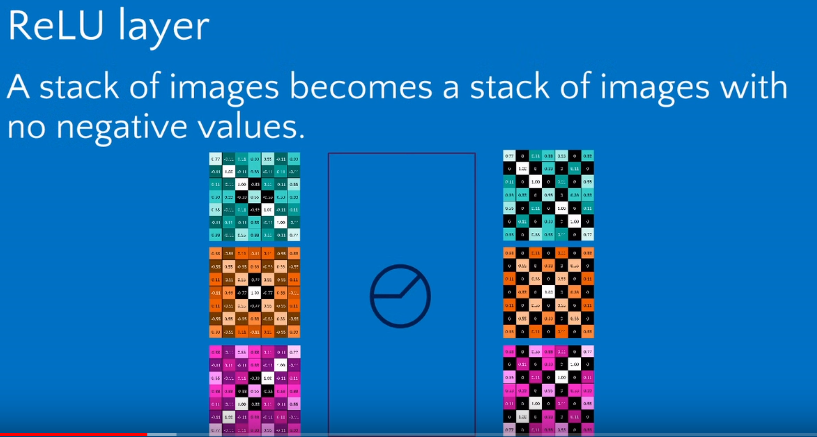
Normalization – ReLU (Rectified Linear Unit)

Keeps the math from breaking by tweaking each of the values just a bit

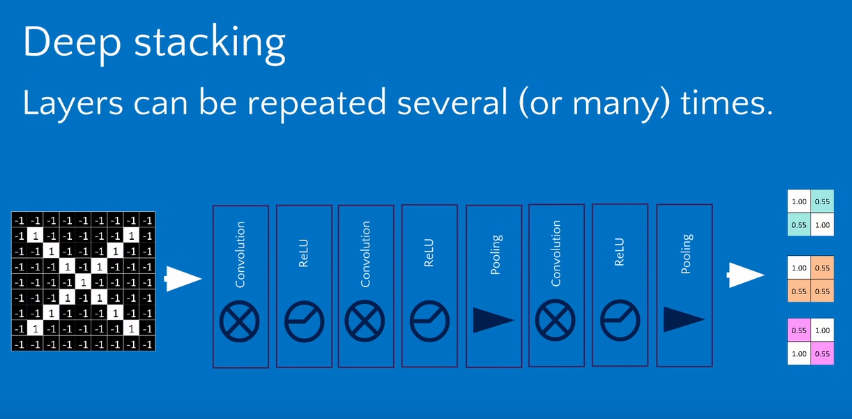
* Change everything negative to zero



A stack of images becomes a stack of images WITH NO NEGATIVE VALUES



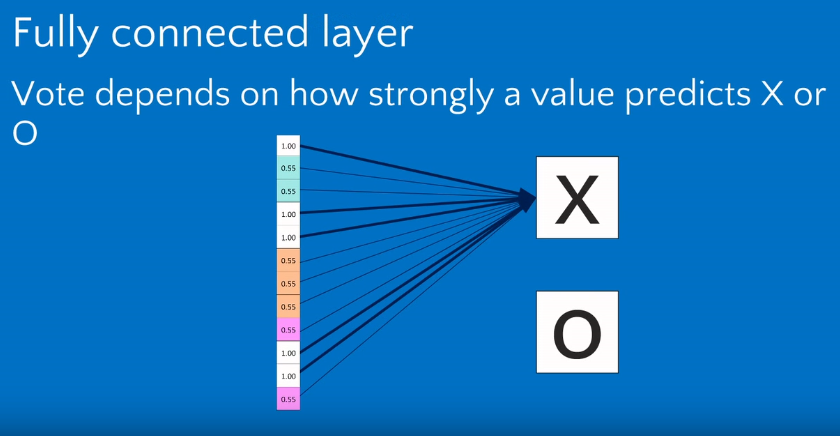
These types of layers in a CNN (convolution, ReLU, Pooling) get stacked – the output of one becomes the input of the next – these layers can be repeated several (or many) times to get a ‘deep stacking’



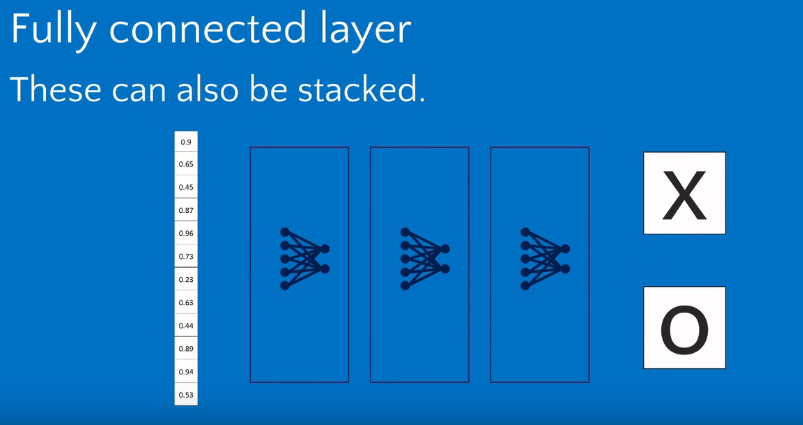
**Why do we use ReLU in CNN?**

**Which type of layers shall we use, how many, and in what order?**

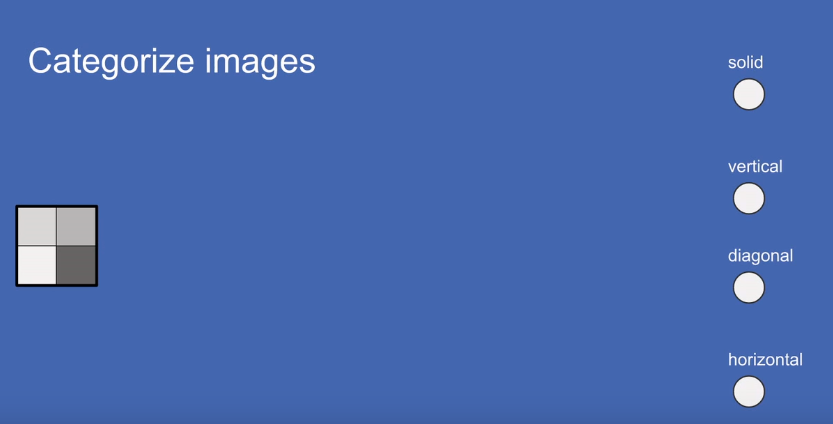
By the time we’ve gone through several iterations of the layers mentioned above, we take this and run it through a fully connected layer (this is more of a standard NN where every input gets connected to everything in the next layer with a weight)



These standard NN’s can also be stacked – we can have an arbitrary number of hidden layers, but these NN’s always stack onto the end

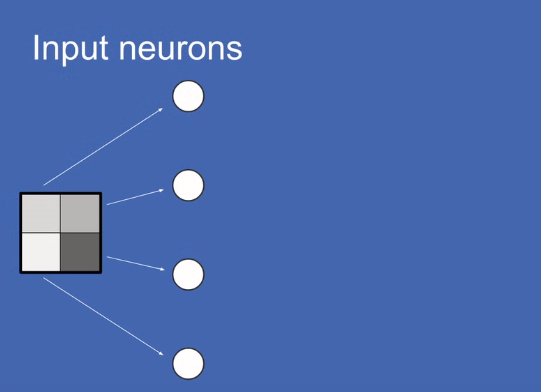


A CNN EXAMPLE

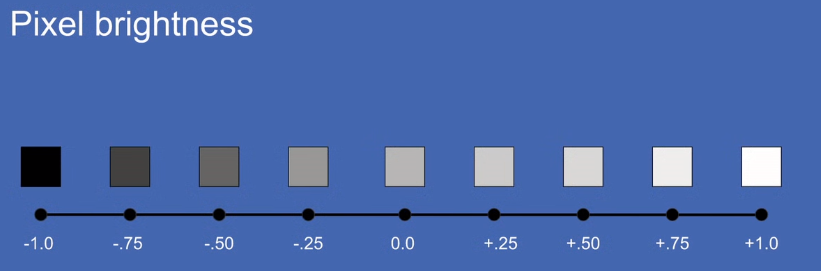


We have a 4 pixel image (2x2) and our task is to categorize this image into one of four classes (solid, vertical, diagonal, horizontal) based on what the image depicts.

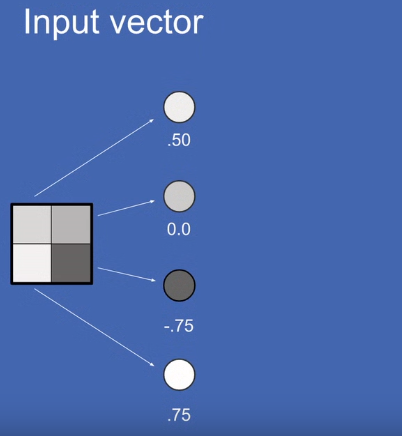
We start of by representing each pixel in the image as an input neuron. For our 2x2 image above, this means that we will have 4 input neurons



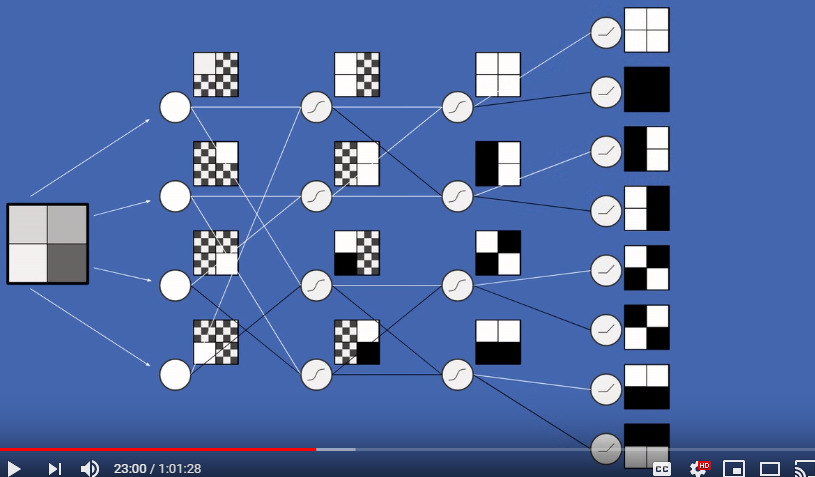
The pixel brightness is normalized to a range, and each pixel takes a value based on how “bright” it is… (see data prep for normalization)

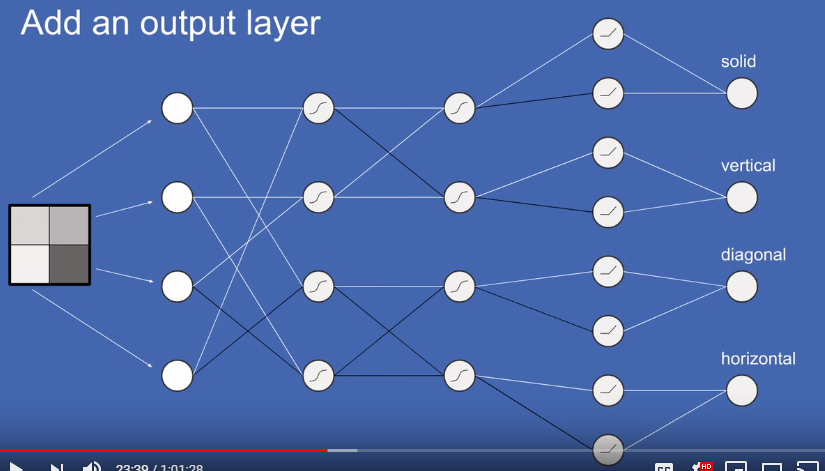


Based on the previous details, our input vector becomes the following



Each pixel has its own receptive field – the part of the image it represents/cares about



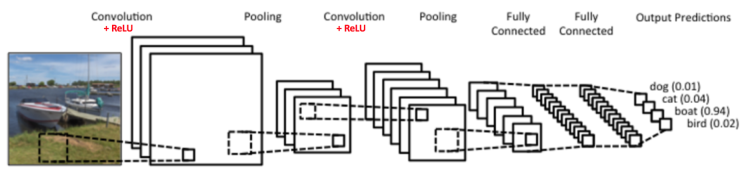


In CNN, things that are closer together are more closely related than things far away. If we were to randomly jumble the array of pixels for an image, we would lose the information.

**CNN history**

The LeNet architecture was first introduced by LeCun et al. in their 1998 paper, [*Gradient-Based Learning Applied to Document Recognition*](http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf)*.* LeCun et al. proposed

It was one of the very first Convolutional Neural Networks and helped to advance the field of Deep Learning. At that time the LeNet architecture was used mainly for character recognition tasks such as reading addresses, digits, names, etc.



The CNN above is similar in architecture to the original LeNet architecture and performs the task of classifying an input image into one of four categories: dog, cat, boat, or bird. Each of the four possible classes that the image could belong to will have a probability score. This score represents the probability of the image belonging in that class – these probabilities should add up to 1. For the image of the boat above, the “boat” class has a probability of 0.94, whereas the other three classes have very low scores – therefore the CNN will classify this image as being a boat.

There are four operations (layers?) in the CNN shown above:

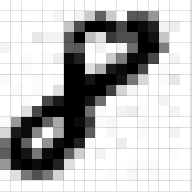
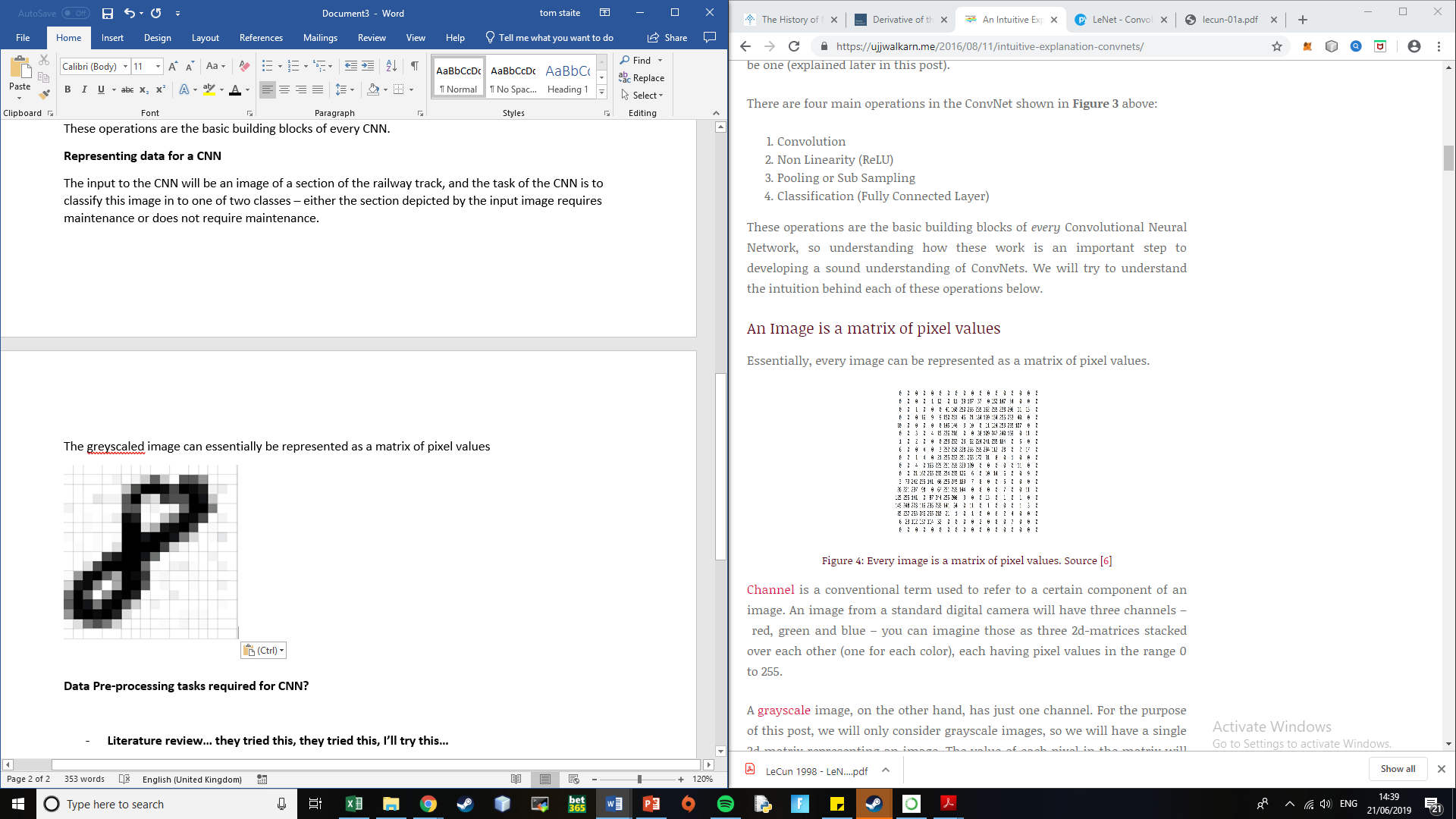
* Convolution
* Non Linearity (ReLU)
* Pooling or Sub-Sampling
* Classification (Full Connected Layer)

These operations are the basic building blocks of every CNN.

**Representing data for a CNN**

The input to the CNN will be an image of a section of the railway track, and the task of the CNN is to classify this image in to one of two classes – either the section depicted by the input image requires maintenance or does not require maintenance.

The greyscaled image can essentially be represented as a matrix of pixel values



Colour images have three channels – red, green, and blue (from these colours we can make any other colours through additive mixture) – we can imagine these as three 2d-matrices stacked over each other (one for each colour), each having a pixel value in the range 0 to 255

A greyscale image has only one channel, the range [0, 255] is chosen so that each pixel can be represented by 8 bits (one byte) – typically a black pixel takes the value 0, a white pixel takes the value 255, and pixel values in between this range represent the greyness of the pixel.

The dataset that I have been provided contains greyscale images and so only one channel (and a single 2D matrix) is required.

The Convolution Step

The primary purpose of Convolution in a CNN is to extract features from the input image. Convolution preserves the spatial relationship (how some object is located in space in relation to some reference object (for example: “the girl is standing behind the table”)) between pixels by learning image features using small squares of input data.

**Hyperparameters for CNN and what values to give them?**

**Data Pre-processing tasks required for CNN?**

* Each pixel in an image represents a neuron – the brightness is normalized between -1 and 1 instead of 0-255, why is this the case? Why is the range -1 and 1 and not some other range?
* Images are 6432 x 2048 – we can split these images (see point below)
* We only have 997 images in our dataset, we would like a dataset that contains a much higher number of images so that our model’s predictions are reliable – a large dataset also helps to prevent both overfitting and underfitting. Increasing dataset by splitting the image?
* Once data is split, look at class imbalance problem below
* We also have a class imbalance problem as it is expected that most of the track will not need maintenance, and so our majority class will be those images that represent track that doesn’t require maintenance – we need to look at how imbalanced the classes are and ways to mitigate this – artificial generation of new images/transformations??
* What about PCA?

Why unbalanced classes is bad

* There are more than 7000 ‘Happy ’images whereas the number of ‘Disgust’ images is less than 500. This would result in a biased classifier that predicts most of the images as majority class.
* if 9000 / 10000 images were in majority class, classifier could predict everything as majority class and obtain 90% accuracy – this doesn’t actually mean its performing well because it will incorrectly predict the class for 100% of the minority class – we should look at sensitivity, specifity, etc
* **Literature review… they tried this, they tried this, I’ll try this…**