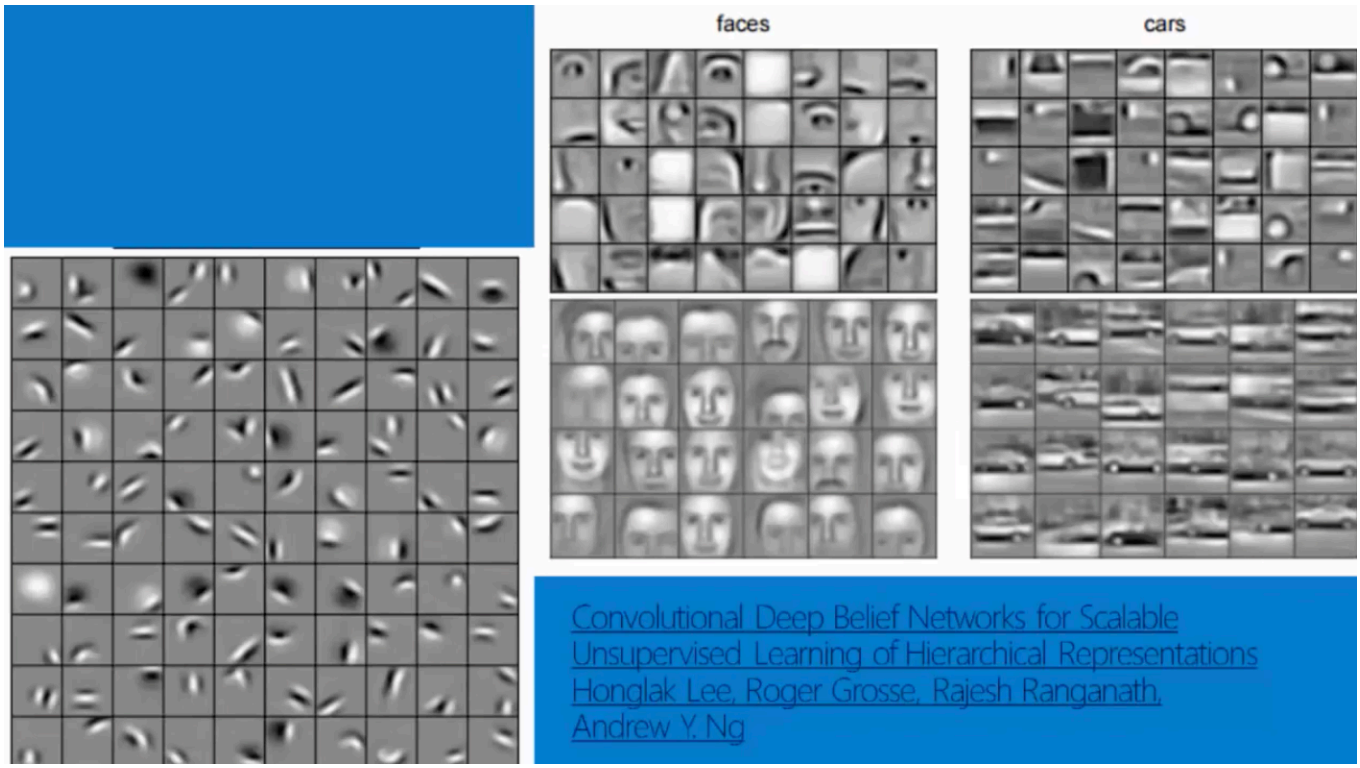
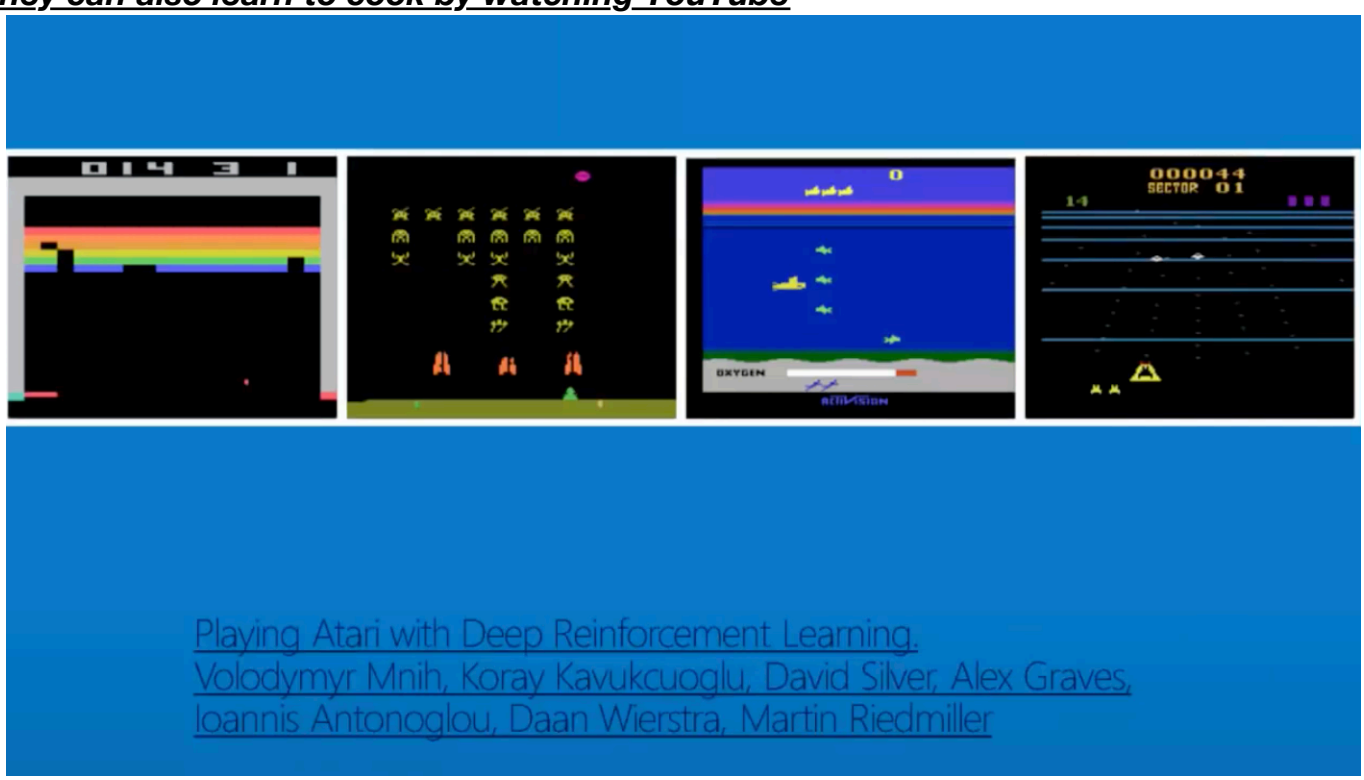
- ***-> convolutional neural networks***
  - -> cn's / con nets
  - ***-> you feed them pictures (for example) and they learn the different features***
    - -> layer by layer of the network these build up
    - -> so in the first layer you might have lines -> then in the next layer of the network they are forming features from the lines in the image

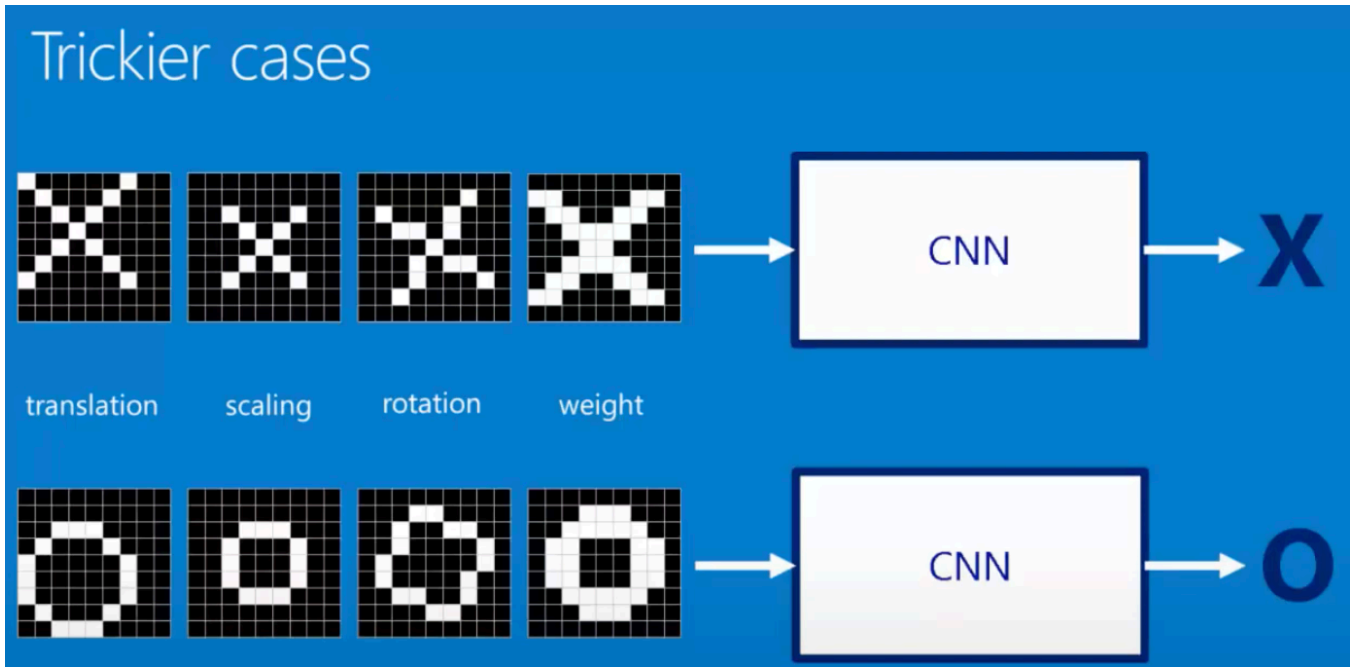


  - ***-> these can also play video games by learning the best action to take when it sees a certain patterns***
    - -> at a better rate than people can
    - -> they see the patterns at where they are in the game and then repeat the patterns which they learnt work in that context in the game
  - ***-> they can also learn to cook by watching YouTube***

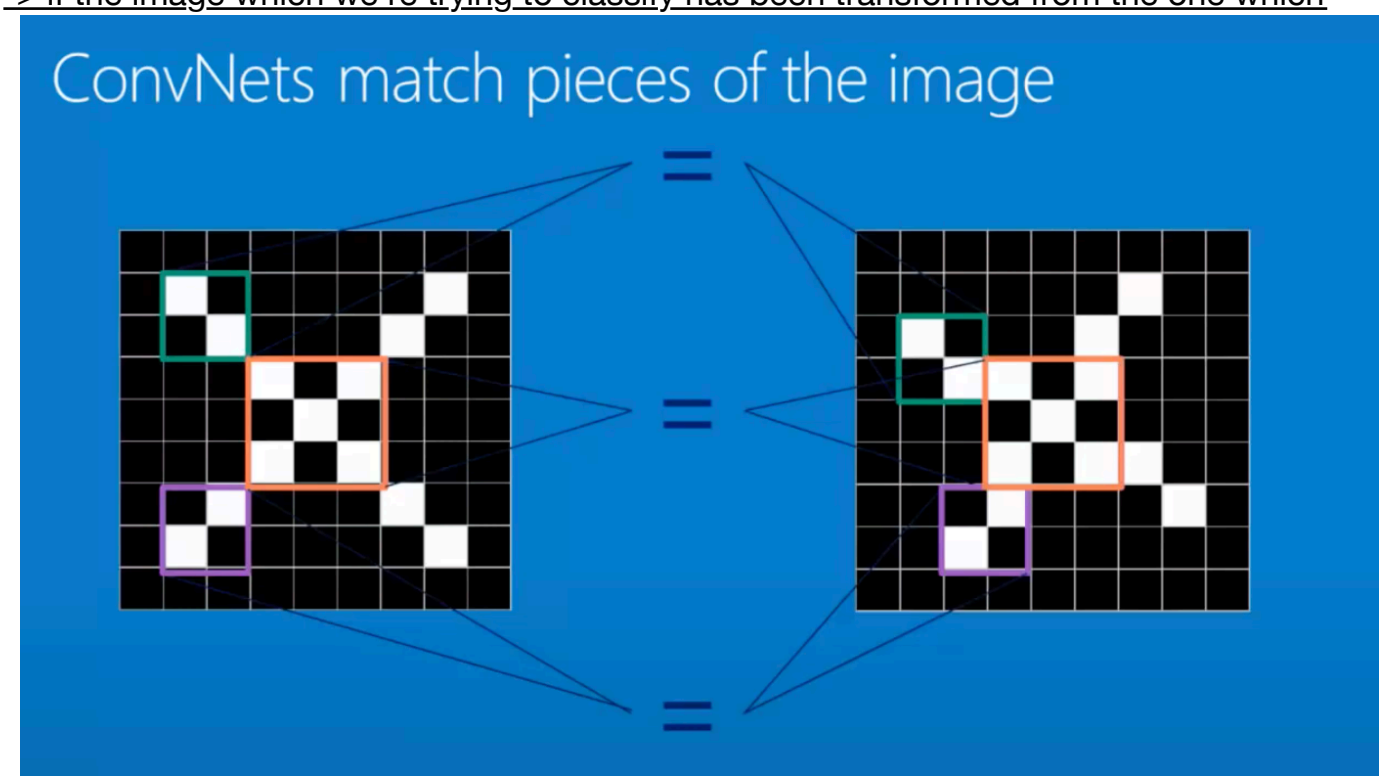


    - -> they can learn the different objects and actions

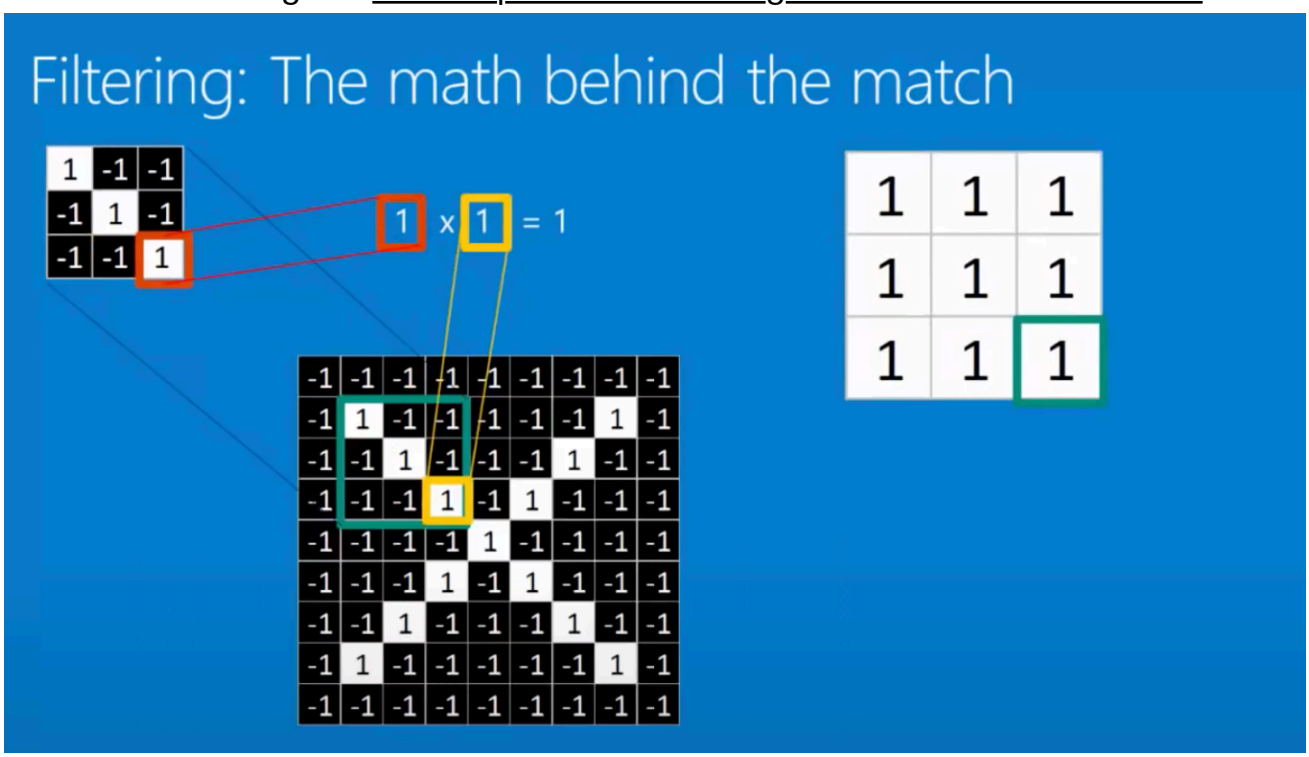
        - -> then repeat the patters
        - -> the different types of grasps
        - -> they see the object and then repeat the pattern which is associated with it
- ***-> checker board example***
    - -> the convolutional neural network is deciding whether the image is of an x or an o
    - -> the image is the input and then it's classification is the output
    - -> the x or o which it's classifying can be transformed and it still has to classify it-> for example made sideways / rotated, shrunk, some can be thicker
        - -> some of the cases are harder than the rest



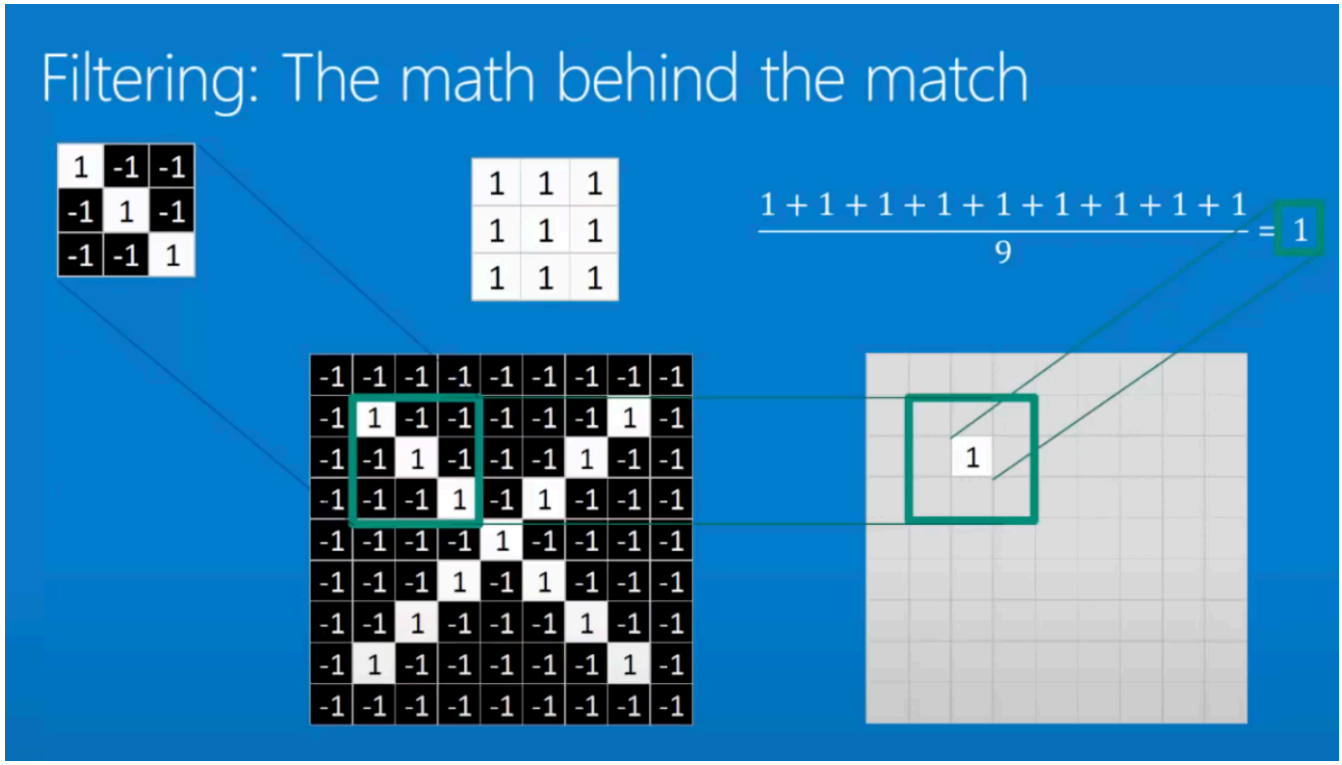
    - -> when you are deciding if the one on the left is of an x -> you compare it to the standard x it was trained on
        - -> but this is hard for it to do -> because it's been transformed
        - -> so it looks at parts of it to see if they match -> rather than the entire thing which doesn't match (because it's been transformed)
        - -> if the image which we're trying to classify has been transformed from the one which



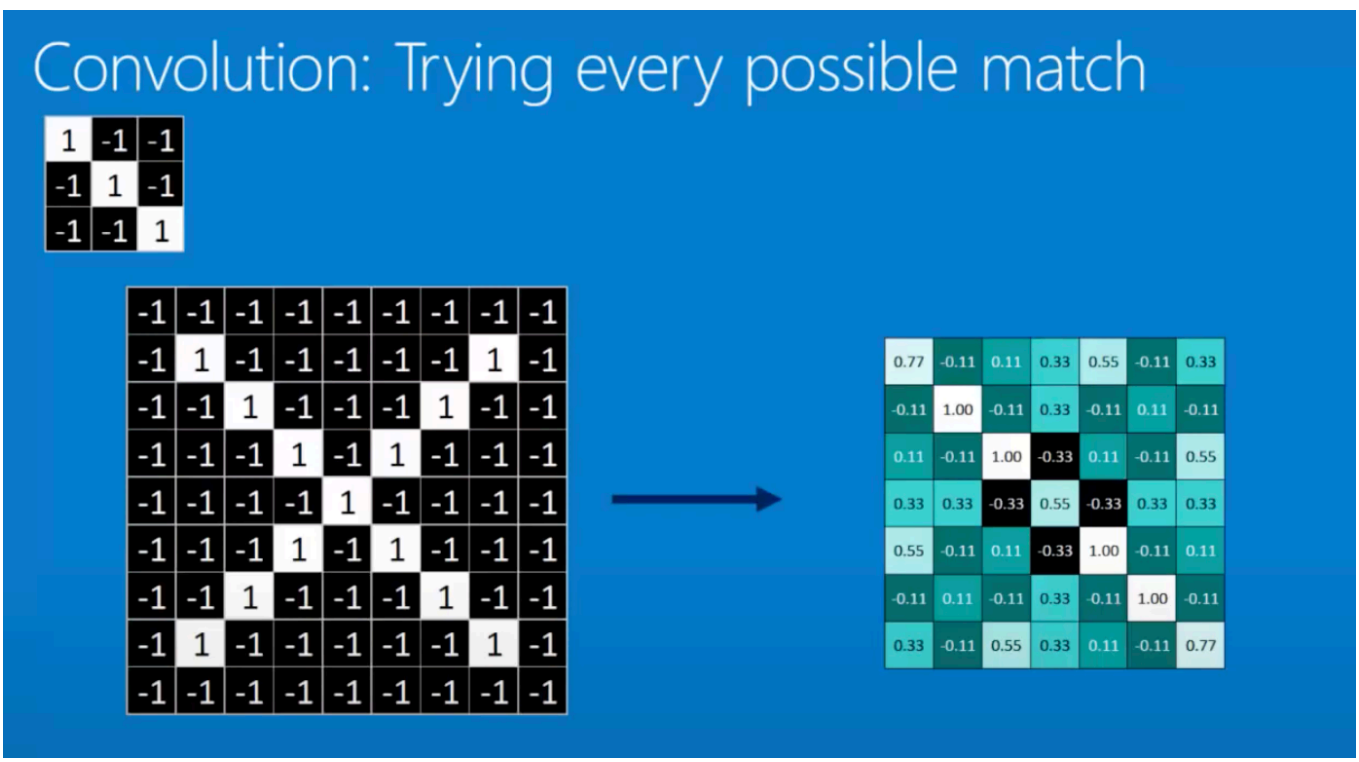
the model was trained on - then with convolutional neural networks the way around this is to look for local patterns which they have that match
    - -> and this way we're not attaching ourself to the global image
    - -> parts of the image we're trying to classify can be transformed and it can still be classified, because we're only looking for local features
    - -> breaking the images down into features
- ***-> filtering***
    - -> taking the entire image and scanning through it with a smaller box which is moved across it
    - -> there are different filters for different features which we're looking for
    - -> we scan through the entire image this way multiple times with the different filters (patterns) which the model picked up when it was training
    - -> and then we combine them to classify the image
    - -> each of the features is a mathematical object (for example a bunch of the elements which are forming a line from left to right in an x form a series of 1s and 0s
        - -> these filters are being compared to the section of the image which they are being scanned across
    - -> then the results from those filters on the images are combined using statistics
        - -> multiplying them by each other adding them up and dividing them by the total number of pixels
        - -> it looks like some kind of dot product or matrix multiplication of the filter with the section of the image <- the component of the image in that section of the filter



- -> once we have done the filtering for that section of image, we move the matrix to another section of the image and run the calculations for that filter again
- ***-> each time the filter is moved, we are getting a number - which indicates how similar that section of the image is to the filter***
    - -> then once we are done with the entire image -> we have many of these numbers which are again combined using statistics
    - -> we do this for multiple filters across the images
    - -> and the filters were derived when the convolutional neural network which uses them was trained
    - -> what we are doing is taking the dot product of the filter which each pixel -> calculating the component of the filter in the direction of the pixel for that pixel

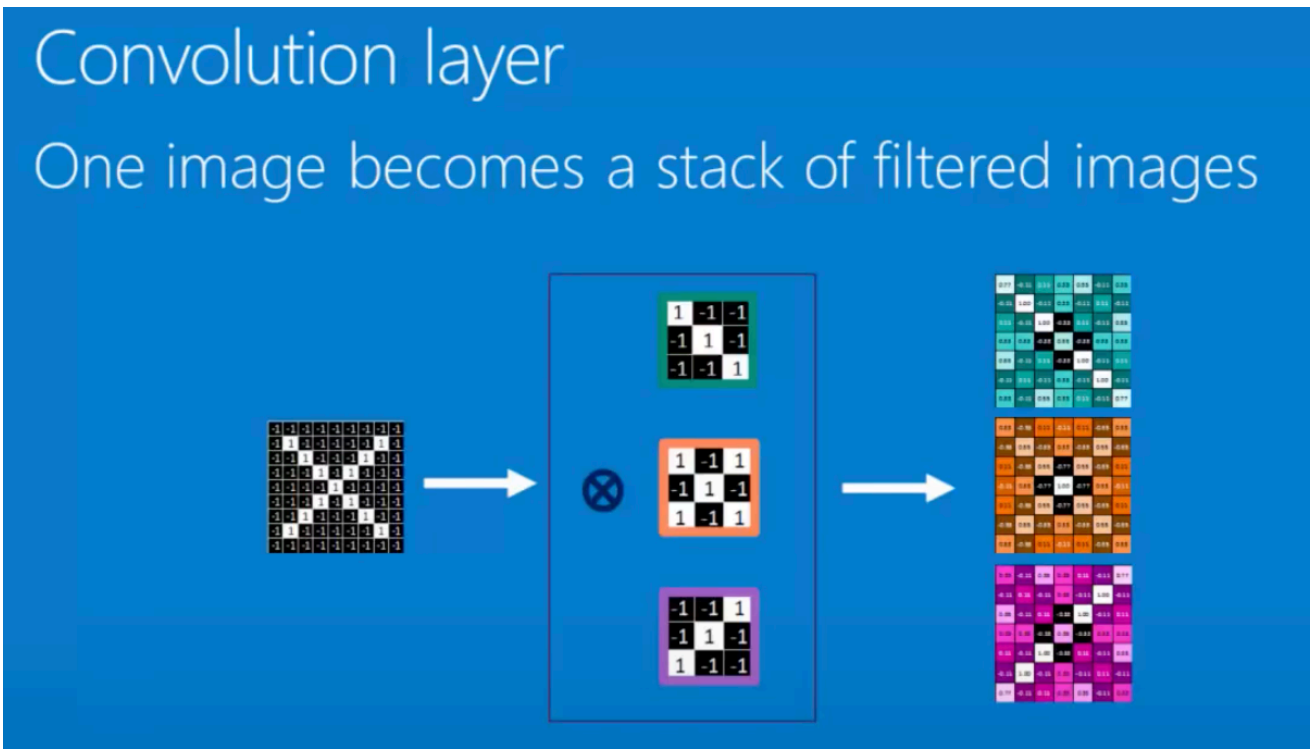


- ***-> and then we are adding all of those values together and normalising them for the entire filter***
  - -> so we have a number which literally indicates the component of that filter in the direction of the section of image which it's being moved over
  - -> in other words something which will indicate the presence of filters
- -> we are moving the filter across each section of the image and calculating the component of that filter in the direction of that section of image
- -> once the entire image has been scanned over, this is called a convolution



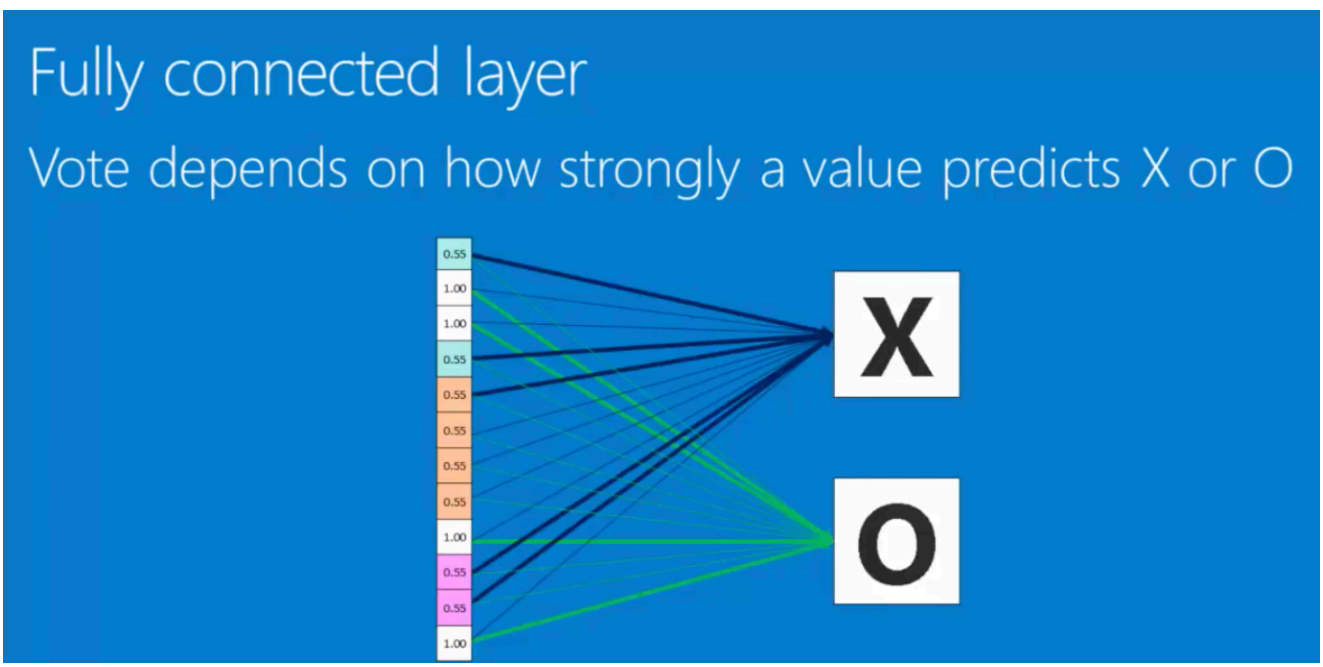
- ***-> on the RHS <- this is a map of those different values***
  - -> the similarities of that section of image with the filter
  - -> the convolution is what we get once the entire filter has been moved over the image (the blue matrix on the RHS of the slide <- how similar those sections of

image are to the filter)
    - -> then analysing if the convolution map makes sense of not -> this is the filtered image
    - -> this works because if the image is transformed then we can still pock up the features
- ***-> convolution layers***
    - -> we go through the same image but with different filters
    - -> each time we are going through the image, we are calculating the component of the filter (the feature which it represents, which was a pattern learnt from training the network) in the direction of the image
    - -> this is a convolution layer
    - -> so we change the filter and go through the entire image again
    - -> we do this for all of the different features / filters which we have
    - -> then we end up with multiple convolution layers <- and each of them is telling us how similar the image is with the feature which that filter represents
    - -> so we end up with multiple of these convolution layers -> each scanning the image for these filters



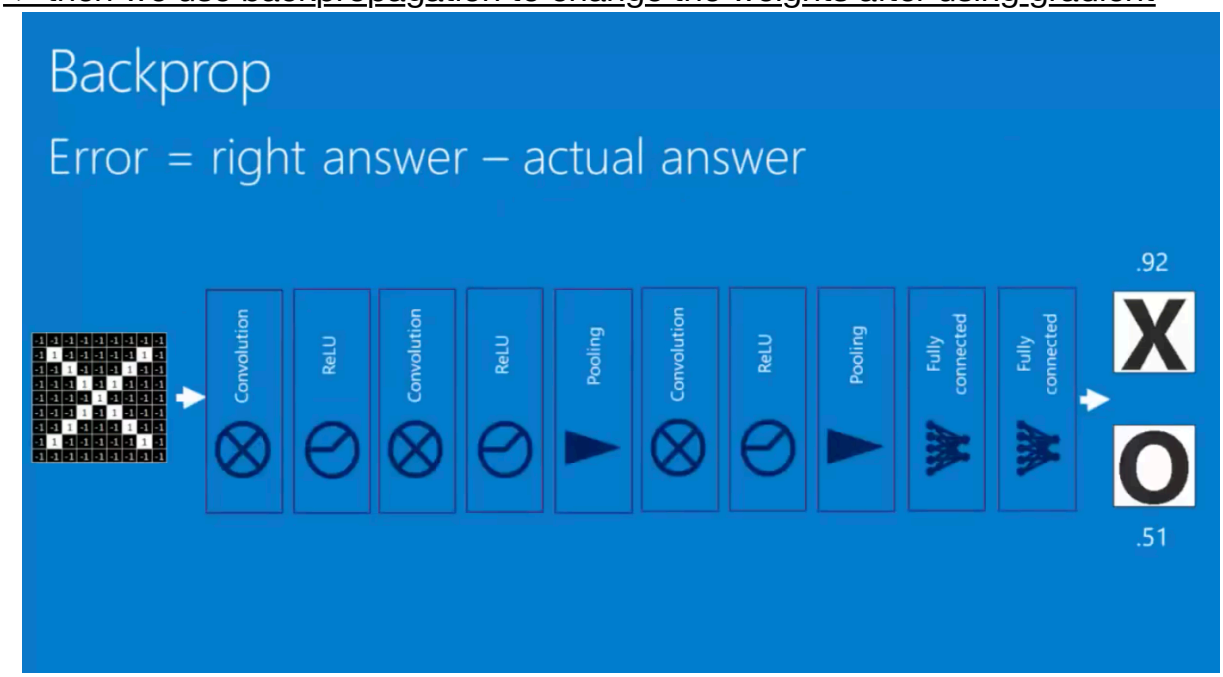
    - -> those layers can be stacked
    - -> the image is a stack of filters
- ***-> pooling***
    - -> so we have split the image into many filters in a stack
    - -> each of the convolution layers is the component of the image in the direction of a filter
        - -> and the filter is looking for a specific feature in the image
    - -> pooling is a technique which is used to reduce the size of the convolution layers (it doesn't combine them)
    - ***-> a convolution layer is an nxn matrix and it's full of dot products***
        - -> pooling scans through the nxn matrix with (in this example) a smaller 2x2 filter moved across the entire thing
        - -> it takes the highest dot product in each quadrant of the 2x2 matrix and neglects the rest of them
        - -> we take the highest dot product in the quadrant of that 2x2 matrix and have that as one element on the new matrix

        - -> so there are three dot products which are being neglected
        - -> this reduces the size of the convolution layer
            - -> in this example it's by a factor of 4
        - -> this is what pooling is
    - -> the dot products which survive this process are called ReLUs <- rectified linear units
- ***-> normasliation***
    - -> so we've scanned through the image for the different convolution layers
    - -> each of them with a different filter
    - -> then we've reduced the size of those convolution layers using pooling
    - -> now we're using normalisation
        - -> to stop the maths from blowing up
        - -> taking all of the negative numbers in the pooled matrix and making them 0 - so avoid the maths from blowing up
        - -> this is done with all of the convolution layers
        - -> this is called the ReLU layer <- a layer with no negative values -> the negatives have been replaced with zeros
- ***-> stacking the layers***
    - -> the convolution layers, rectified linear unit layers and pooling layers are all stacked
    - -> the output of one layer becomes the input of the next (stacking)
    - -> you can have as many of these layers as you want (e.g as many pooling layers)
    - -> this is called deep stacking
- ***-> fully connected layer***
    - ***-> about the fully connected layer***
        - -> each value gets a vote
        - -> we have a lot of different filters

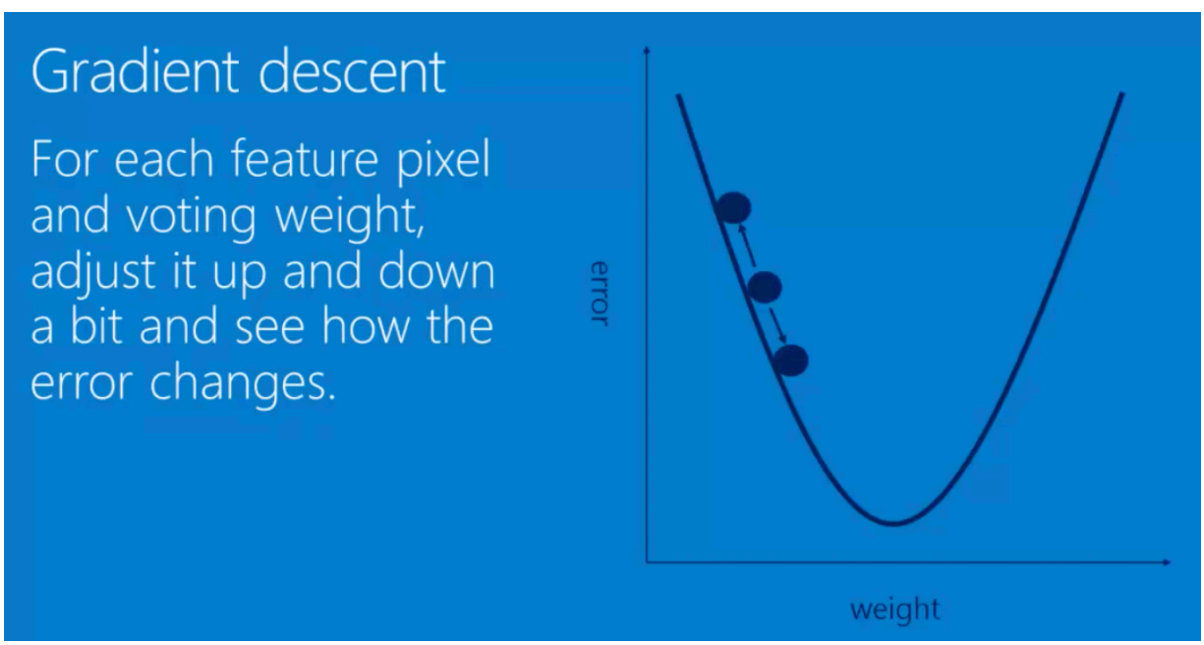


        - -> each of them votes for wither outcome -> the outcome being the prediction of the model
        - -> each filter 'votes' for what the outcome of the prediction will be
        - -> then we combine them and end up with an overall prediction
        - -> the fully connected layer of the network is the one which combines all of the different filters into a prediction
    - ***-> the output of one layer can be the input of the next***
        - -> each of those different cells on the left is a filter

    - -> and each of the filters is getting a vote as to what the overall prediction of the model will be
    - -> each layer has different weights <- these are literally dot products
        - -> the largest dot products in a little 2x2 matrix of pixels scanned across the image
        - ->there are hidden units in a neural networks -> ones which you don't see
- ***-> this layer is formed from back propagation***
    - -> back propagation is what is used to combine the matrices from all of the different steps in the process into the final layer (he fully connected layer)
    - -> the neural network learns the weights
    - -> you have the stack and train it to output the weights (initially starting at random)
    - -> we calculate the error of the prediction which this makes, in comparison to the training dataset
    - -> then we use backpropagation to change the weights after using gradient



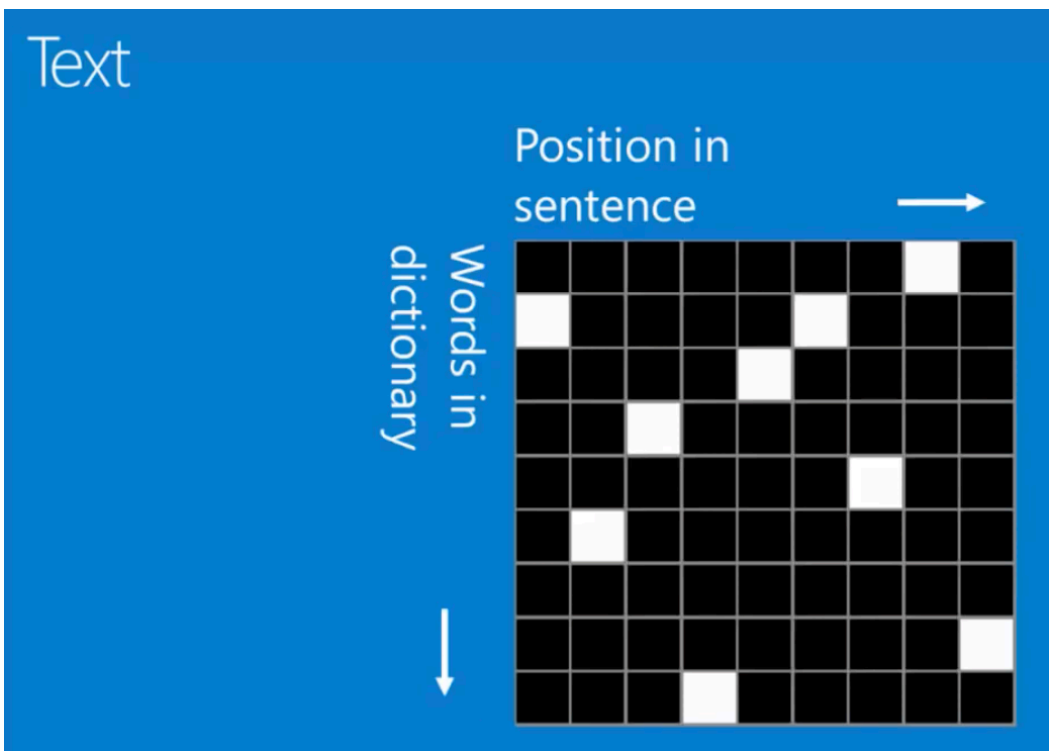
descent



- ***-> gr***

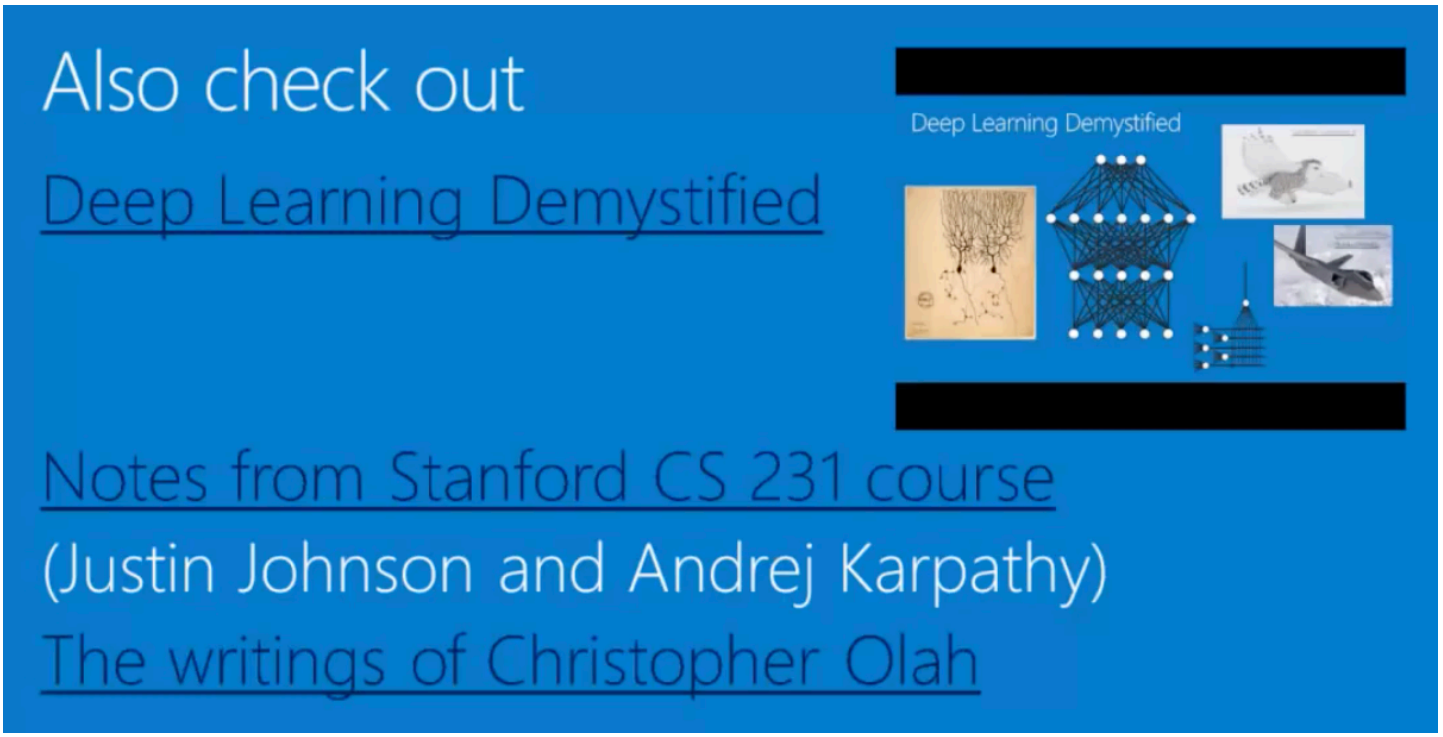
adient descent for deep neural networks

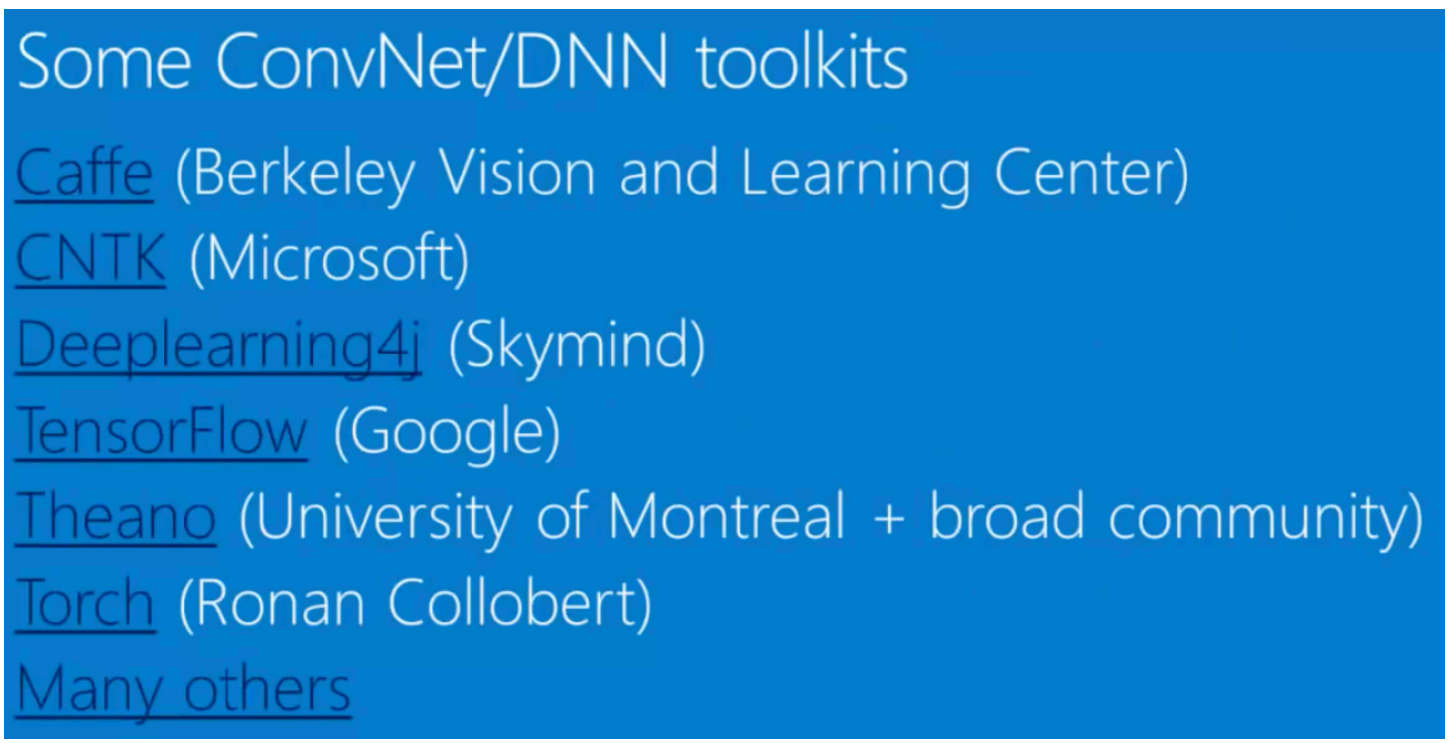
- -> it's similar to scanning an energy landscape -> so that the model gets more accurate and the loss function decreases into a minimum
  - -> we are comparing the predictions of the network to what the actual answers are
  - -> deep neural networks
  - -> changing the weights to see how the error changes
  - -> you go this many times across all of the features which are represented by the filters
  - -> the architecture is how you choose the number of hidden neurones and fine tune the weights aka the hyper-parameters using back propagation
    - -> convolutional neural networks <- these are the ones which are using gradient descent
    - -> the amount of those networks and in which order they go
- ***-> long term short term memory***
  - -> the model is more affected by the data which was just passed into it
  - -> predictions are more affected by the points which are close to them
  - -> in this example he's done it with a dictionary
    - -> weather the order of the words matter
    - -> in nlp the order of the words matters



- ***-> limitations***
  - -> only spatial patterns in the data are captured
  - -> data can't be made to look like an image
  - -> convolutional neural networks are only for local spatial patterns
  - -> they are for finding patterns, e.g to classify images
  - -> if you have a large dataset

- ***-> further resources for convolutional neural networks***





  - -> deep neural networks and convolutional neural networks
  - -> convolutional neural networks aren't useful if you want to make your data look like an image / rearrange elements of the data