Chapter 9 : Part 3

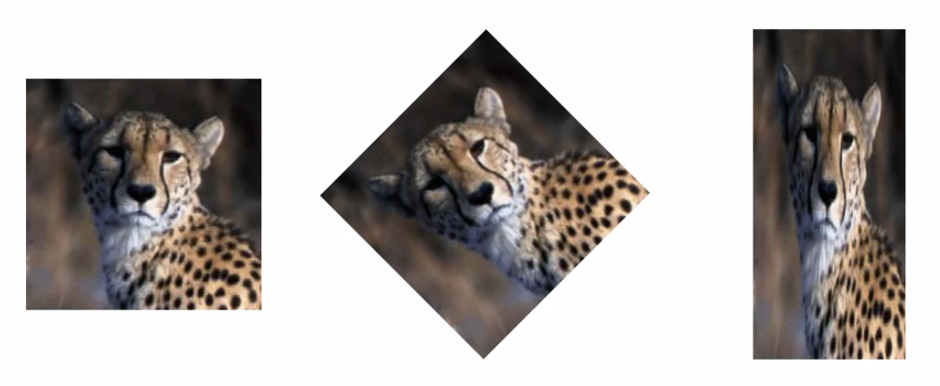
**Deep Learning**

**CNN: Pooling, flattening &**

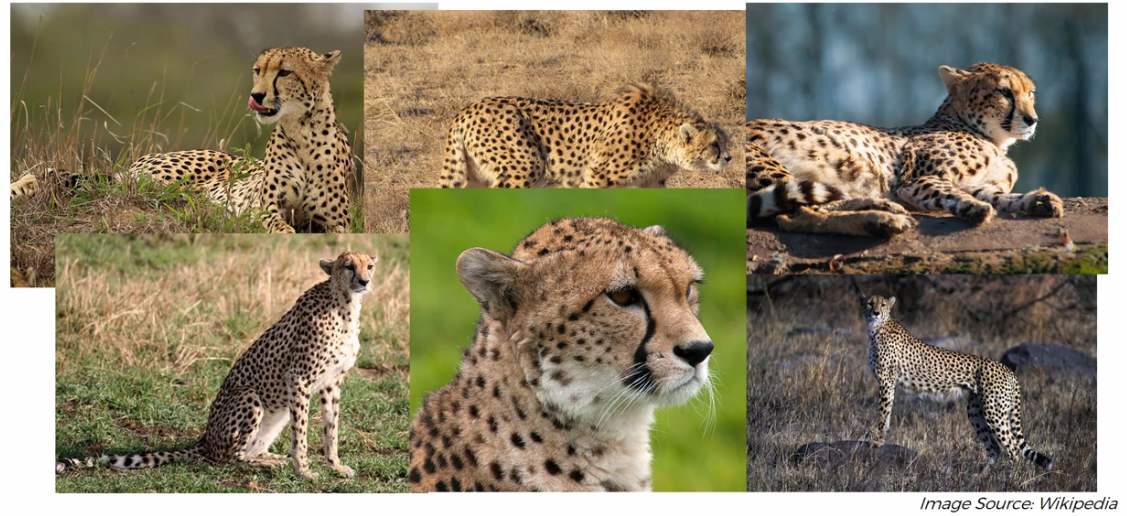
**Full Connection**

**9.3.1 Pooling**

Here we'll talk about Max pooling. And what is ***pooling*** and why do we need it. Well to answer that question let's have a look at these following.



* We've got a *cheetah*. Same *cheetah* but in 2nd image its *rotated* and in 3rd image its a bit *squashed*. We want the *neural* *network* to be able to recognize the cheetah in every single one of these images.
* However, this is just one cheetah. What happens if we have lots of different Images of different Cheetahs?
* We want the neural network to recognize all of these Cheetahs. So how can it do that if they're all looking at *different* *directions*, they're in *different* *parts* of the image, they're *faces are positioned differently* and many other differences like *light* and *texture* etc.



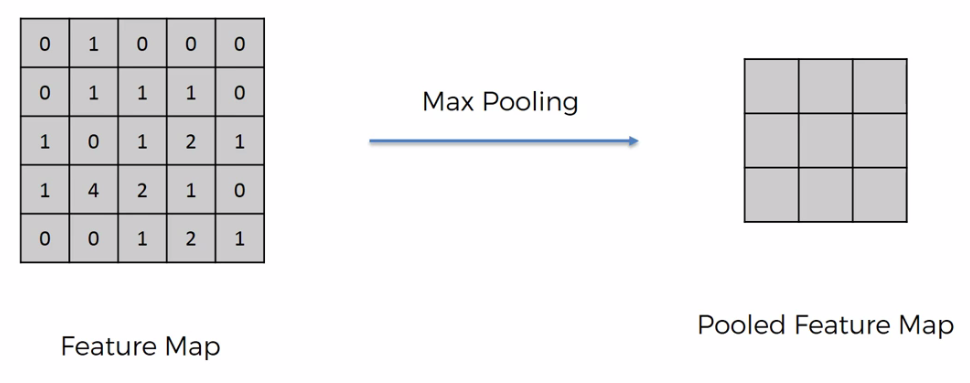
* There's lots of little differences and so if the neural network looks for a certain feature for instance: A distinctive feature of the Cheetah is the ***tear-mark*** that are on its ***face*** going from the ***eyes***.
* But if NN is looking for that feature which it learned from certain cheetahs in an *exact location* or an *exact shape* it will never find these for other Cheetahs.
* So we have to make sure that our NN has a property called ***"spatial invariance"*** meaning that it doesn't care where the features are. i.e. the NN doesn't have to *care* if the features are a *bit tilted*, bit *different* *in* *texture*, if the features are a *bit* *closer* or a *bit* *further* apart relative to each other.
* So if the feature itself is a bit distorted our neural network has to have some ***level of flexibility*** to be able to *still* *find* that *feature*. And that is what POOLING is all about.

**9.3.2 Max Pooling**

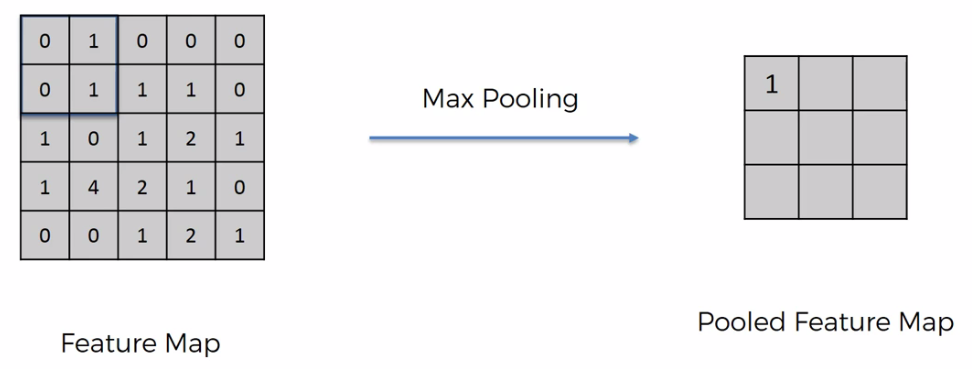
Consider following as our feature map (already done our convolution). Now we're going to apply pooling (also called Down-Sampling). There are different types of ***pooling***

* Max pooling: The maximum pixel value of the batch is selected.
* Min pooling: The minimum pixel value of the batch is selected.
* Average pooling: The average value of all the pixels in the batch is selected.
* Sum Pooling: This is a variation of the Max pooling. Here, instead of average or max value, the sum of all the pixels in the chosen region is calculated.

Here we'll apply ***Max-Pooling***.



* Max- pooling: We take a ***box of pixels***. And select ***Stride size*** of ***2*** (if you want overlapping box then select ***stride size*** of ***1***). However, you can choose any size of box and place it in the ***top left hand corner*** and you *find* the *maximum* *value* in that *box* and then you record only that max value.



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

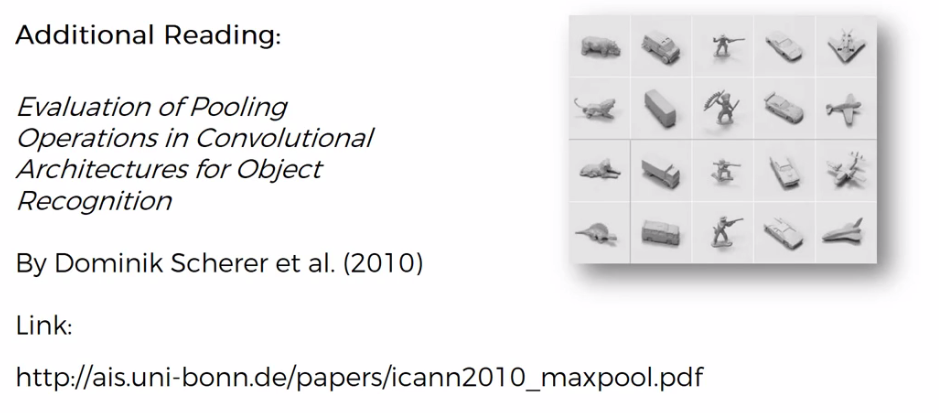
* Here we're selecting a ***stride of two*** and this size is *commonly* *used*. After repeating the process we record that maxim values shown in above pictures.
* Here following things happened:

1. Preserving the features: First of all we still were able to preserve the features, the maximum numbers they represents the features. These *large* *numbers* in your *feature-map* represent where you actually found the *closest* *similarity* to a *feature*.
2. We're introducing spatial invariants: And because we are taking the maximum of the pixels we are gaining ***"spatial invariance"***. For example: *Cheetah's Tear-mark* in *different* *version* of the image (*rotated* and *squashed*) or *other* *position* can *end-up* with same *Pooled-Feature-Map*.
3. Reducing the size of the data: We're reducing the ***size by 75%*** which is really going to help us in terms of ***processing***. By Pooling these features we are getting rid of 75% of the information that is not the feature (not related to the feature) by picking ***1px*** out of ***4px*** using ***box of pixels***.
4. Preventing Overfitting: Another benefit of pooling is we are *reducing* the *number* of *parameters*. We're reducing 75% of parameters that are going to go into our final Layers of the neural network and therefore we're preventing Overfitting (i.e. model doesn't get dependent on train images).

It is a very important benefit of pooling that we're *removing information* because that way our model won't be able to *Overfit onto* that *information*. Because it's important to see *exactly* the *features* rather than all other *noise* that is coming into our eyes. Same thing goes for neural networks by *eliminating* the *unnecessary* *information* we're helping with preventing of *overfitting*.

That's the point of pooling that we're still being able to preserve the features account for their possible spatial or textural or other kind of distortions.

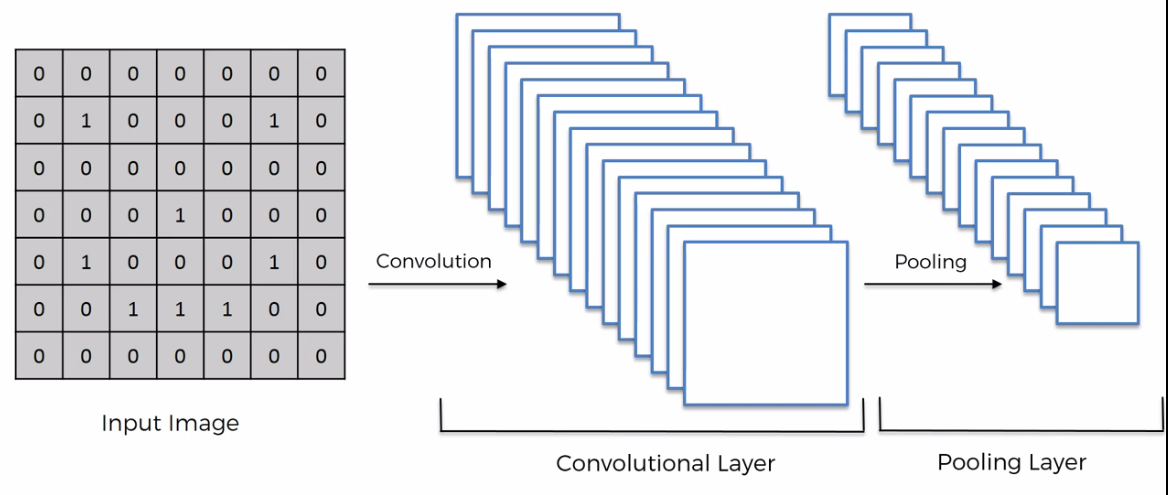
* Why Max pooling (Additional Reading): There's lots of different types of pooling and why stride of two and  ***pixels*** box size. On that note I'd like to introduce you to this lovely research paper called *"Evaluation Of Pooling Operations In Convolutional Architectures For Object Recognition"* by Dominic Scherer from University of Bonn.
* They talk about a concept called *Subsampling* which is basically *average* *pooling*.
* In average pooling, instead of taking the ***Max*** ***value*** we take the ***average***.



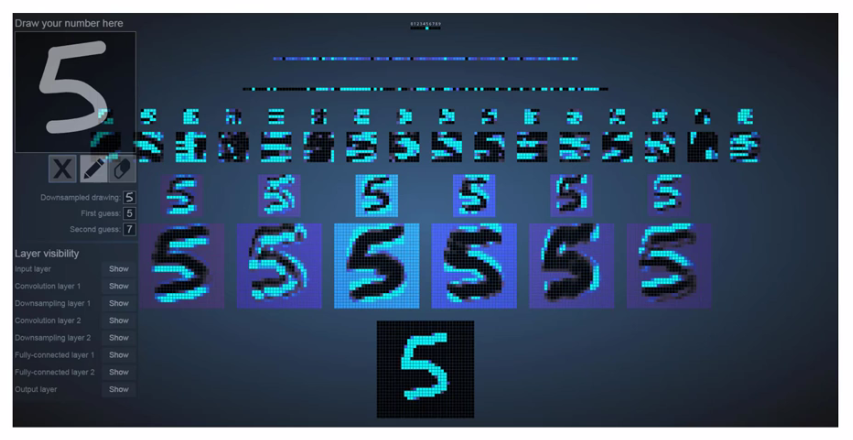
**Let's recap what we have done so far.**

* On our input image, we applied the *convolution* *operation* and we got the *convolution-layer* (collection of *Feature-Maps*).
* And then to each of those **feature-maps** we've applied the "**Max-Poolling**" and we get a **Pooling-Layer**.
* So we've done these two steps:

1. Convolution and
2. Pooling

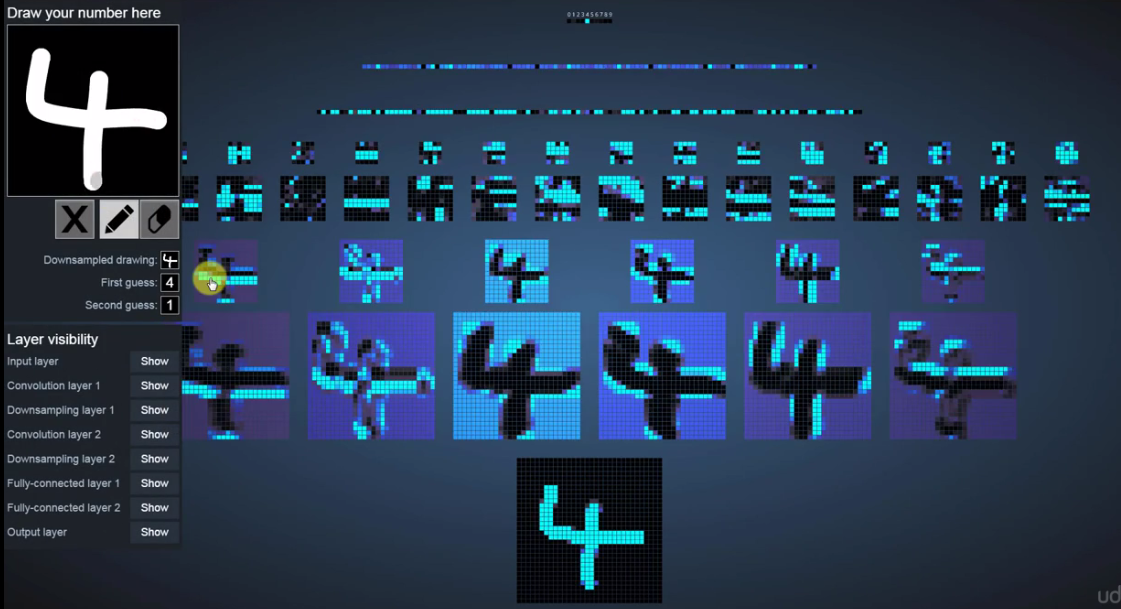


* Example: There is a fun example of CNN. Follow the link (may be deleted now).



|  |  |
| --- | --- |
| * And basically this is exactly what we're doing but we can visualize it here (kind of). So here we draw a number, say we draw number 4, and in this tool will put the number ***four*** at the bottom (that’s what we drawn). |  |

* After that, the ***1st*** row is the ***convolution*** ***step***, the ***2nd*** row is the ***Polling*** ***step*** (also called Down-Sampling). When it's applied *pooling* it's *reducing the size* and you can see here that *pooled* *image* has *same* *features* that having just *less* *information*.



* It also gives the 2nd guess.

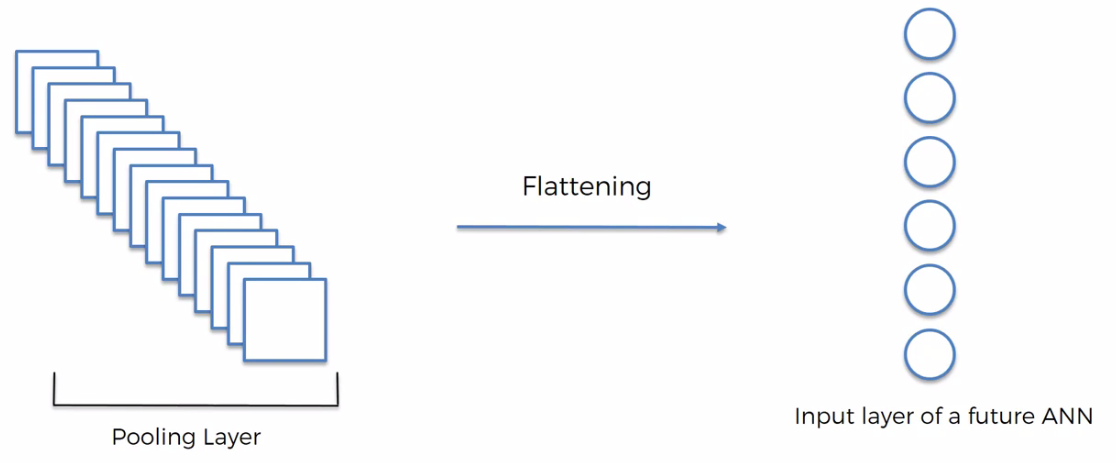
|  |  |
| --- | --- |
|  |  |

* We can also see the pixels of the image how it is reduced. And also we can see the ***Stride*** ***size*** (pointy thing). The 3rd row is also ***convolution*** ***layer*** of the previous ***Pooling*** ***layer***. 4th row is another ***Pooling***. Last two rows are ***flattened-layer***.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**9.3.3 Flattening**

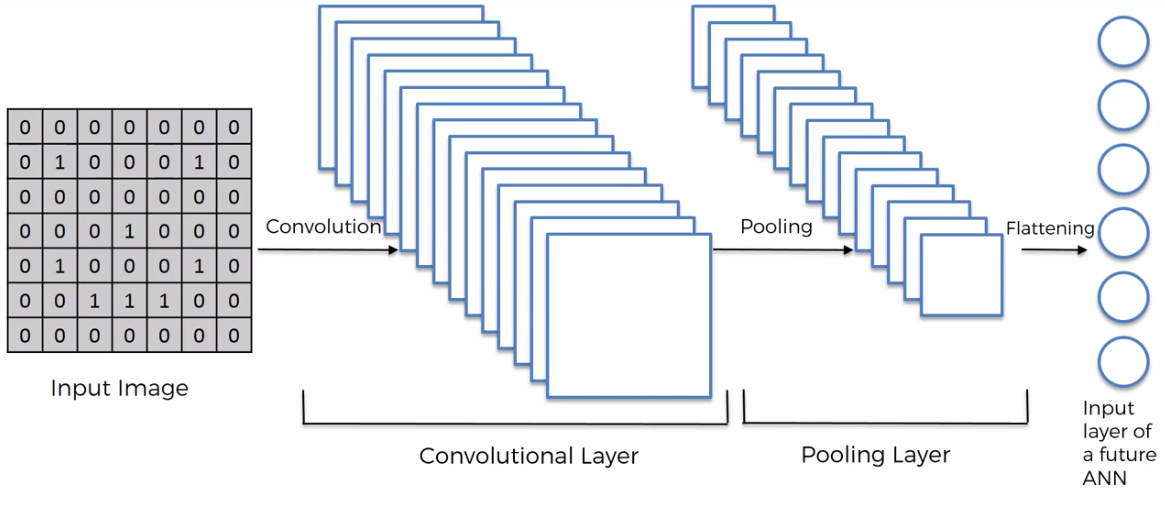
|  |  |
| --- | --- |
| After we get the ***Pooled Feature Map*** (in ***Pooled*** ***layer***) we're going to flatten it into a column. Basically just take the values of "row by row" and put them into one long column. i.e. we are putting ***2-D matrix*** into ***1-D vector***.   * The reason is we want to input this ***vector*** into an ANN for further processing. This ***vector*** will be ***Input Layer*** of the ***ANN***. * When flattened the pooling layers (with multiple Pooled Feature Map) into *one long column sequentially* then we end up like following: |  |



* So we put them one after the other and we get one huge vector of inputs for an Artificial Neural Network.

**So at this point we applied following steps:**

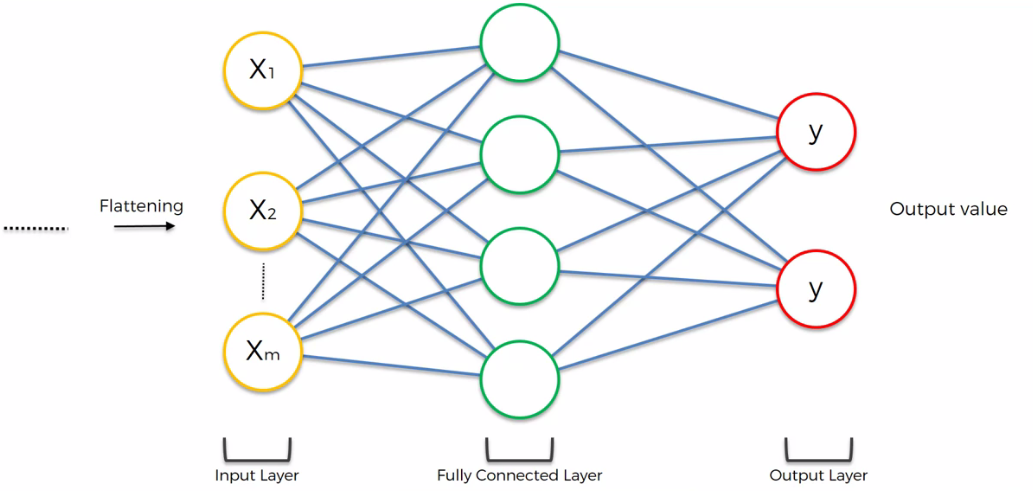
1. We've got an input image.
2. We apply a ***Convolutional-Layer*** and ***ReLU*** function
3. Then we apply pooling
4. After that we flatten everything into a long vector which will be our input layer for an ANN.



**9.3.4 Full Connection**

In this step we're adding a whole *Artificial Neural Network (ANN)* to our *Convolutional Neural Network (CNN)*.

* So here we've got the *input-layer*, *fully-connected-layer*, and *output-layer*. In ANN we used to call them as "Hidden-Layers" but in CNN we call them " fully-connected-layer "
* Because, indeed they are *hidden* *layers* but at the same time they're a more *specific type of hidden layers* that are fully connected.
* In ANN, *hidden letters* ***don't need to*** be *fully* *connected*. But in CNN those layers are fully connected, hence called fully-connected-layer.



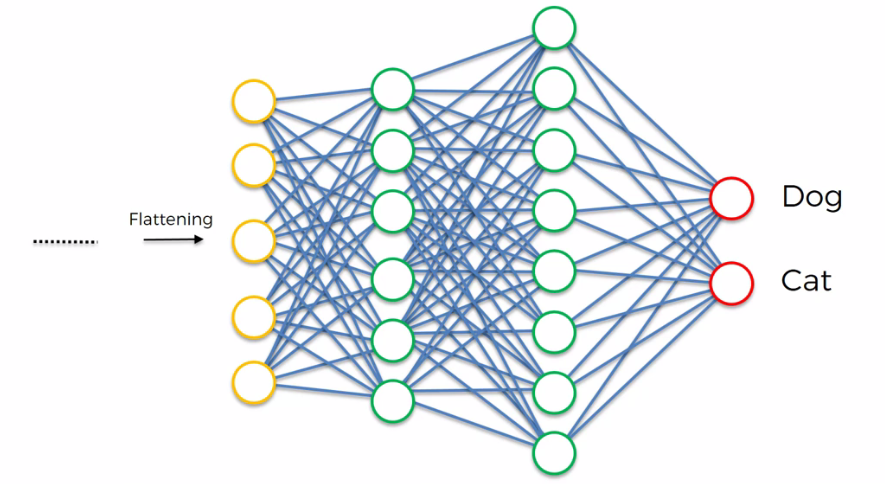
* Above is a very simplified example of ANN. The main purpose of this ANN is to *combine* our *features* into *more attributes* that predict the Classes even better. Here the column or ***vector*** of ***flattened-layer*** is passed into the Input Layer.
* In our vector of ***vector*** of ***flattened-layer*** we have some *features* encoded in the *values* in that *vector*. Those features can already predict what class we're looking at, for instance, whether it's a Dog or a Cat.
* But at the same time the ANN dealing with features and coming up with new attributes and combining attributes together to give even better prediction.

So we pass on those values (*Flattened-Layer Vector*) into an *ANN* and let it even further *optimize* everything that we're doing.

**9.3.5 How Fully-Connected-Layers Work**

Let's look at a more realistic example. Here we've got an ANN where we have

1. five attributes on the ***inputs*** that
2. we have in the first *hidden-layer (fully-connected-layer)* we have ***six*** ***neurons***,
3. in the *second fully connected layer* we have ***eight*** ***neurons*** and
4. then we have *two outputs*, one for Dog and one for Cat.



* Why do we have two outputs here: Before in ANN we used to have one output.
* Note that, ***one output*** is for kind of when you're ***predicting*** a ***numerical*** ***value*** and running a ***regression*** type of ***problem***.
* Here we are doing classification. In our case we have two classes : Cats and Dogs. We can do our job by using one output (binary output: ***1*** is a ***dog*** and ***0*** is a ***cat*** and that would have worked totally fine).
* But when you're doing classification, where you have ***more than two categories*** for instance ***dogs***, ***cats*** and ***birds*** then you have to have a ***neuron***for***each******category*** and that's why we're going to practice with ***two categories***.

|  |  |
| --- | --- |
| * So we've already done the *convolution*, *pooling* and *flattening* and now the information is gonna go through the ANN.  1. At the first step the *convolution*, *pooling* and *flattening* happens and the information going through the ANN and a prediction is made. 2. And for instance let's say, it predicted 80% that given image is a ***Dog's*** image. But it turns out that its actually a ***Cat***. |  |

1. Then an error is calculated. "*Cost* *function*" in ANN, where we used to calculate "*mean square error*". In CNN it's called a "Loss Function" and we use a "Cross Entropy Function" for that (we will talk about ***Cross Entropy Function*** later).

* This Loss Function tells us how well our network is performing and we're trying to optimize or minimize that Loss Function to *optimize* our *Network*.

|  |  |
| --- | --- |
| 1. After the error is calculated, it's Back Propagated through the network just like ANN and couple of things are adjusted:      1. The ***weights*** in the ANN Synapses (blue connected lines) and 2. The ***feature detectors (filters)*** are also adjusted (those little matrices that we had in convolution step). It updates the features if looking for the wrong features. So that next time the NN gets improved. Done with GD (or SGD) and Back-propagation. |  |

1. After Adjusting the ***weights*** and ***feature*** ***detectors*** the forward-propagation happens again. Then Error calculation and after that again Back-propagation.

* This processes are going on until our CNN is optimized and the network gets trained on the data.
* Important: Note that that the *data goes through* the whole *Network* from the *very* *start* (i.e *convolution*, *pooling* and *flattening* again) to the very end.
* So same thing as ANN but a bit longer because of that the first three steps *convolution*, *pooling* and *flattening*.
* How do these two classes/output neurons work: Let's see how does this image classification works.

|  |  |
| --- | --- |
| * Let's start with the *top output-neuron* (indicates *Dog*). * First of all let's *assume* (hypothetically) the weights of the Synapses that are connected to Dog. These numbers can be absolutely anything we are just doing this for learning purpose. Sot that, we can say which of the previous-layer neurons are actually important for the Dog.   So let's say we've got above numbers in our ***previous*** (final) ***fully connected layer***. Those can be any numbers but just for argument's sake we're assuming these numbers between 0 and 1 i.e. in [0, 1] interval. |  |
| * These ***neurons*** of final- fully connected layer are actually finding the ***features***. And ***0*** means that that neuron didn't find a feature is looking for. so because at the end of the day these neurons like anything else on this from on this left side * Even though These ***neurons are*** already processed, but still they're detecting a certain feature or a combination of features on the image.   For example following glowing neurons are for features of dogs ***nose***, ***floppy ears*** and ***eyebrows***. So these neurons do indeed fire up when the feature belongs to a dog. |  |
| * On the other hand the Cat neuron will know that it's not a Cat. Even though it got also ***floppy ears*** (***one*** ***neuron*** with ***1***) and Cat neuron will ignore this neuron. * Through lots and lots of iterations if this happens often. The Cat neuron will ignore this neuron more (Synapse with *Cat-Neuron* will be weak. i.e. weight gets smaller). And *Dog-neuron* will pick up those *3 neurons*. It will start attributing *higher importance/weight* to those *3 neuron*. * Hence we're going to say that this Dog neuron learned that these three neurons *(eyebrow neuron* and *big nose neuron* and *floppy ear neuron) are* contribute to the *classification* of *Dog* and *Cat* through this iterative process with many *samples* & *epochs*. * Important: One thing to note that the *signals* from those *3-neurons* are going actually to both *Cat-Neuron* and *Dog-neuron* simultaneously. * Remember, these *ears* and *nose* and *eyebrows* those are very approximate (because of going through *Convolution*, *Pooling*, *Flatenning*) or unrecognizable what they're looking for but at the same time it is ***something*** in the ***features*** of ***dogs***. | |

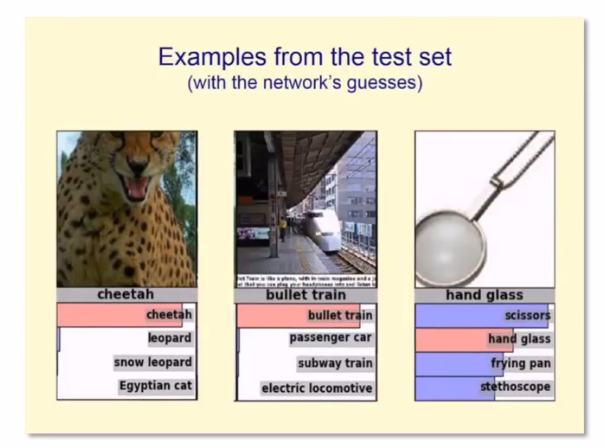
|  |  |
| --- | --- |
| * Now let's move on to Cat-neuron. But those 3-Synapses (how we've sorted out the dog using these weights) of *eyebrow neuron* and *big nose neuron* and *floppy ear neuron* are remembered. * Now how is the Cat-neuron works? Well whenever it is actually a cat, there will be *different values/numbers* for the *last fully-connected-layer-neurons* for different neurons. Here let's consider again *3 different neurons*. |  |
| * Notice these three neurons ***0.9***, ***0.9*** and ***1***. They're ***interacting*** to both the ***Dog*** and the ***Cat*** (remember that the ***Signals*** from those 3-neurons are going actually to both ***Cat-Neuron*** and ***Dog-Neuron*** simultaneously). * Important: This is again important to remember that those output signal goes both ways to ***Cat-Neuron*** and ***Dog-Neuron***. |  |
| * Remember: It's up to the ***Cat-Neuron*** and ***Dog-Neuron*** to decide whether to take into account that signal and learn from it or not (using Synapse weights). * And ***Cat-Neuron*** and ***Dog-Neuron*** analyze the given photo/image of Cat/Dog. * So basically the ***Dog-Neuron*** will recognize Cats *whiskers* and Cats *pointy triangle ears* and Cats *vertical eye pupil*. It will recognize because every time these neurons fired up the prediction is not Dog. * On the other hand the ***Cat-Neuron*** recognize those three neurons (*whiskers* , *pointy triangle ears* , *vertical eye pupil*) because most of the time these three lights up it matches ***Cat-Neuron's*** expectation for Cat. * ***Cat-Neuron*** is going to listen to those three neurons more and more. So basically it's listening to these three and it's ignoring the other five. | |

* And that is how these final neurons (***Cat-Neuron*** and ***Dog-Neuron***) learn: *"which neurons in the Final Fully Connected Layer to listen"*.
* That's how the features are propagated through the network and conveyed to the output.
* In reality these *features* don't have that *much meaning* to them like *floppy* *ears* or *whiskers* but they do have some *distinctive feature* of that specific class.
* That's how the network is trained because during the *back* *propagation*, we're not just adjust the *weights* but also the *feature-detectors*.
* If a feature is useless to the output, it's going to be disregarded and replaced with feature which is useful because this happens through thousands and thousands of iterations.

So at the end in this *Final Fully Connected Layer* of neurons have lots of *features* or *combinations of features* from the image that are indeed *representative* or *descriptive* of *dogs* and *cats*.

|  |  |
| --- | --- |
| * How prediction/recognition occurs : Once your network is trained up, let's say we pass on an image of a Dog. The values are propagated through a network we get certain values. * Dog and the Cat Neurons don't know that it's a dog or a cat. But they have learned to listen to what is being shown here. Dog-Neuron listens to these three neurons (Purple Synapse) a Cat-Neuron listens to other three neuron (Green Synapse) for features. * Purple weights are how the Dog neuron views their votes. How much importance is it assigns to these neurons and those votes. And Green weights are how much importance the Cat's neuron.  1. So when the Dog neuron ***founds*** its ***corresponding*** ***feature*** ***neurons*** pretty ***high*** then it give a ***high probability*** that *"given image is probably a dog"*. 2. And Cat neuron ***founds*** its ***corresponding*** ***feature*** ***neurons*** very ***low***. Then it give a ***low probability*** that *"given image is a Cat".*   And that's how we you get our prediction. So the answer is Dog. NN is voted *high* for a *Dog*. *Voting* is a term that is used for *neurons* in the *Final Fully Connected Layer*. The corresponding *weights* are the *importance* of their *vote*.   1. Same thing happens when you pass an Image Of A Cat. |  |
|  |
|  |

* So these neurons vote the *Dog* and the *Cat* based on their *learned weights* for *Final Fully Connected Layer*, then they make their *predictions*.
* And that's how you get images like this where you have a *Cheetah* (voted high). And we can see the *voting is different* for *different* *images*, because corresponding CNN recognize differently.



* So that's how the Full Connection works.