Contents

[Input Layer 2](#_Toc50848917)

[Hidden Layers (3 types: Convolution, ReLU, Pooling 2](#_Toc50848918)

[Convolution Layer 3](#_Toc50848919)

[ReLU Layer 4](#_Toc50848920)

[Pooling Layer 5](#_Toc50848921)

[Flattening 6](#_Toc50848922)

[Fully Connected Layer 6](#_Toc50848923)

[Output layer 7](#_Toc50848924)

[CH1 NOTES – Introducing Neural Networks 7](#_Toc50848925)

[CH2 NOTES – Coding Our First Neurons 7](#_Toc50848926)

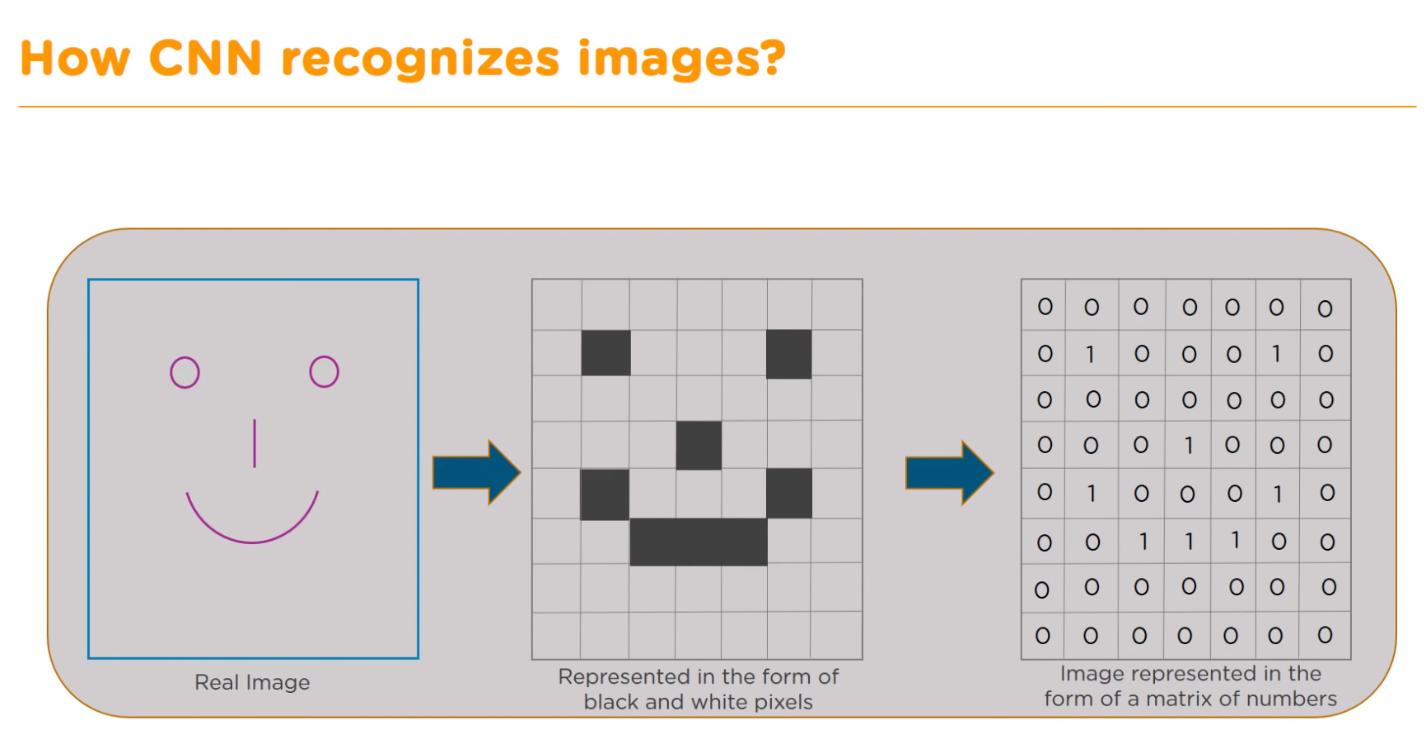
[Project Notes: 9](#_Toc50848927)

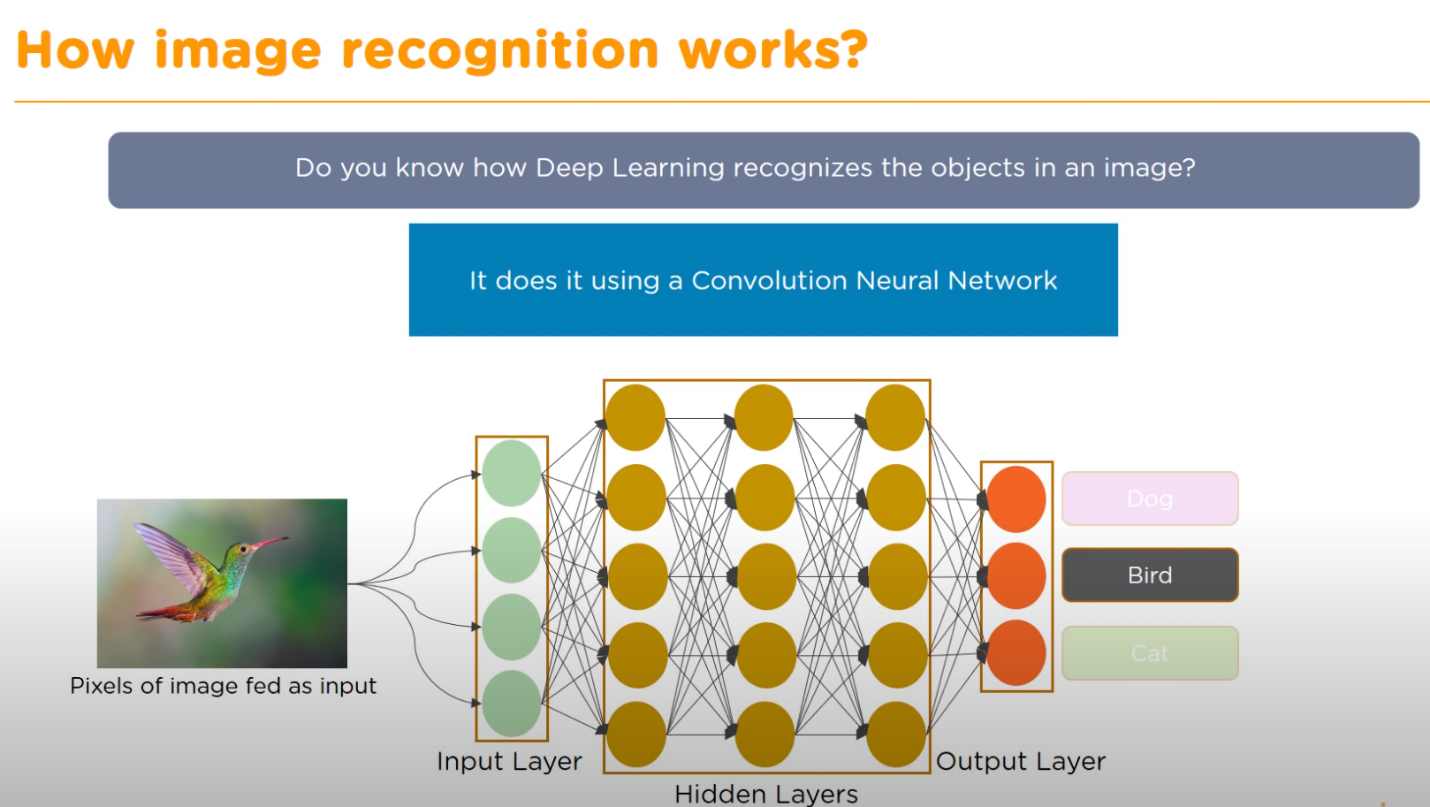
[Requirements for working with data in scikit-learn: 10](#_Toc50848928)

<https://www.youtube.com/watch?v=Jy9-aGMB_TE>

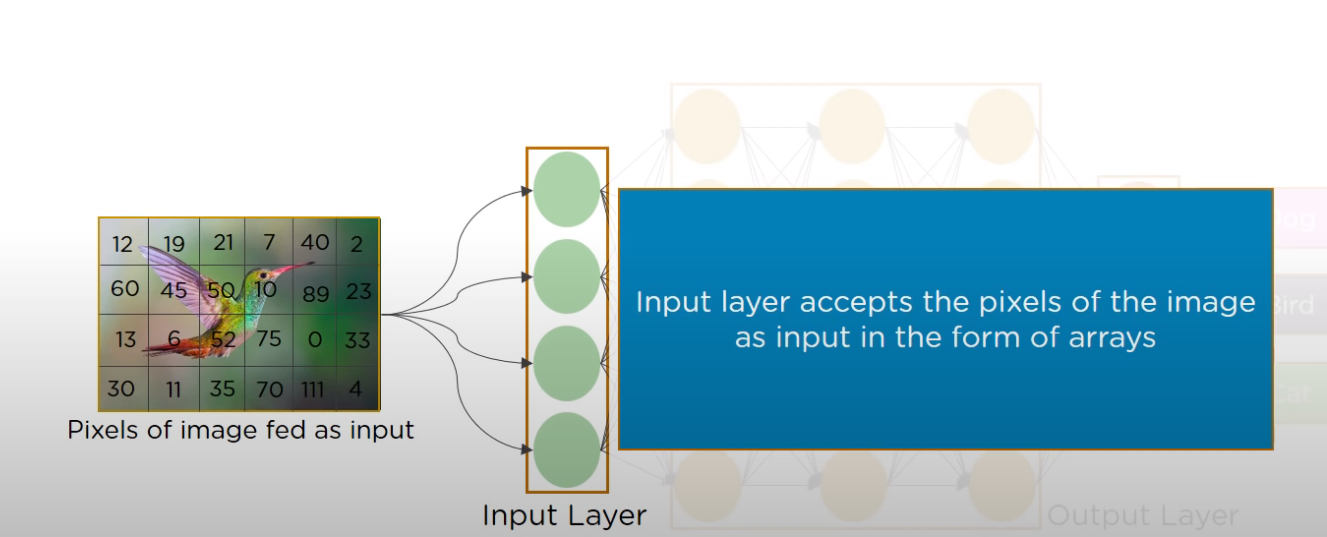
I used this to create the tensorflow/keras GPU environment: <https://www.youtube.com/watch?v=PnK1jO2kXOQ>

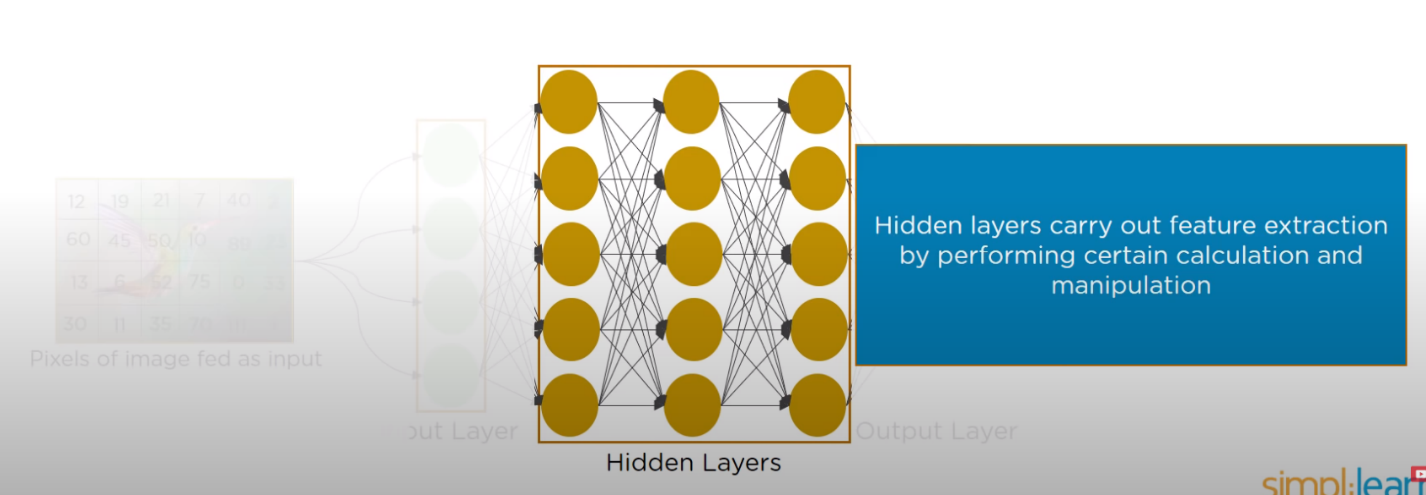
Convolutional Neural Network Tutorial:





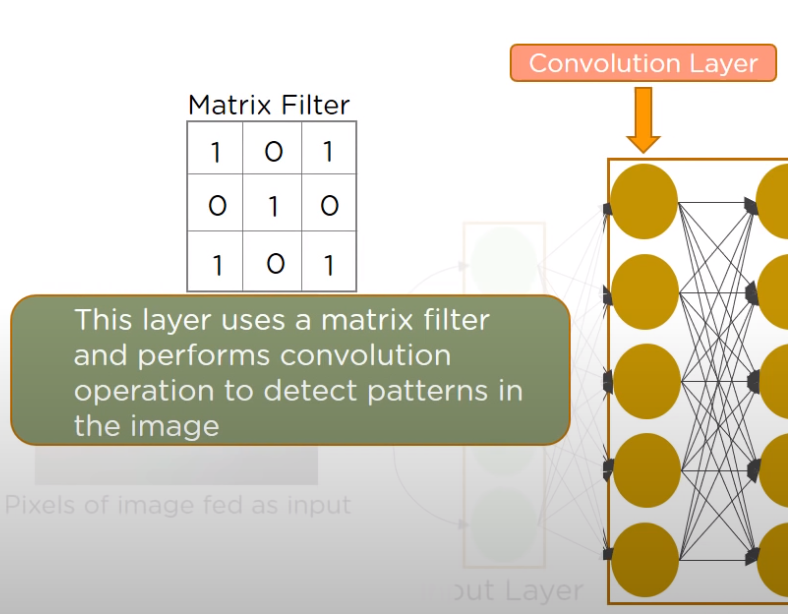
# Input Layer

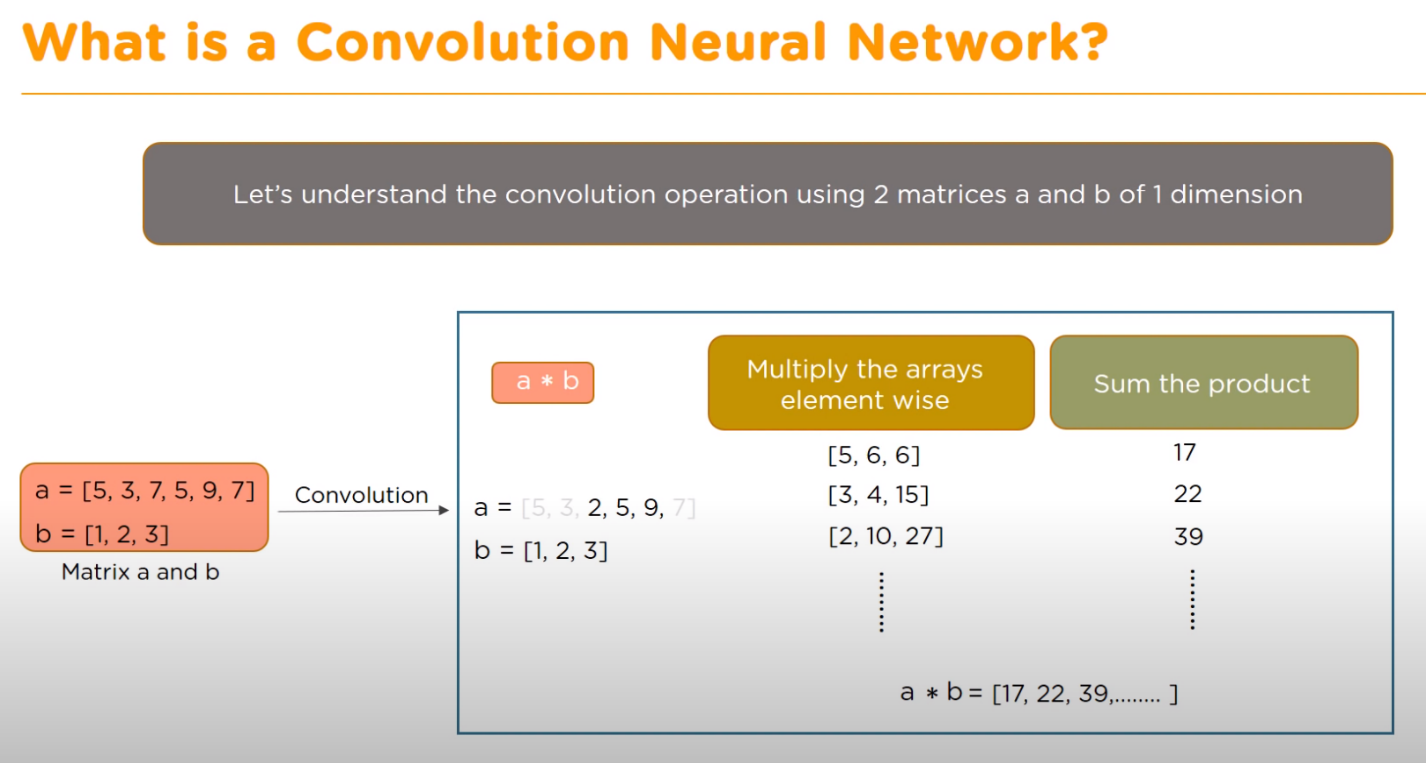


Hidden Layers (3 types: Convolution, ReLU, Pooling

This layer reorganizes the image multiple ways until we get some data that’s easy to read for the neural network.

# Convolution Layer

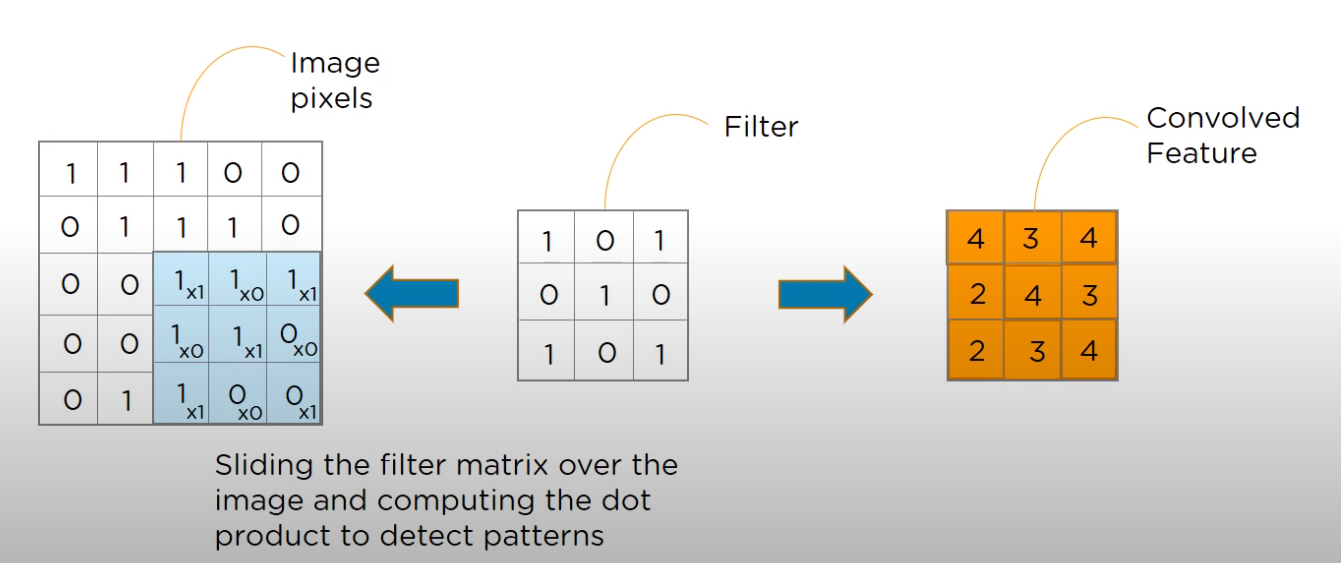
Convolution means to coil or to twist. So the data is twisted around alter it, and use that operation to detect a pattern. The central aspect of processing images



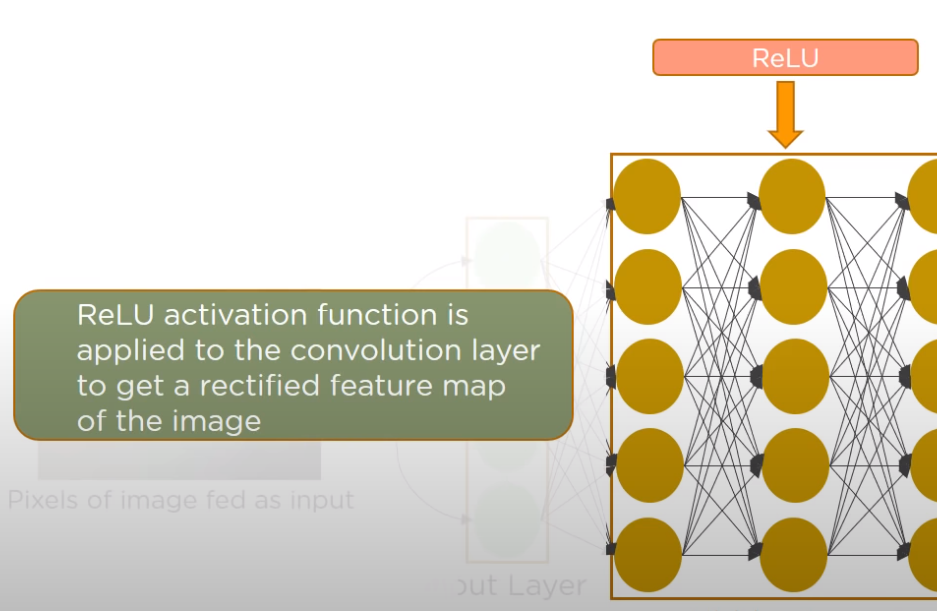
[5,6,6] is the result of: [5\*1],[3\*2], [2\*3] , [3,4,15] is the result of: [3\*1],[2\*2],[5\*3]

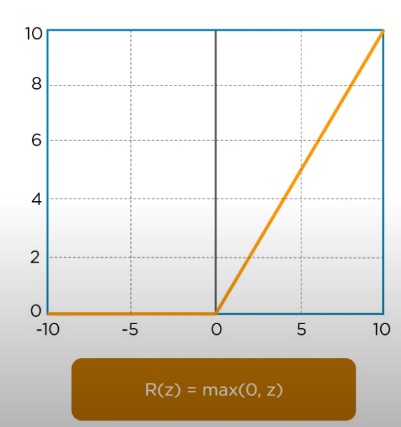
[2,10,27] is the result of: [2\*1],[5\*2],[9\*3]

A convolution layer has a number of filters that perform convolution operation. An image is a block of pixel values, say for example there is a 5x5 image, and a 3x3 filter (a filter is built when your program/train your model to detect things) – what the filter is doing is it is slid over the image and a new 3x3 Convolved Feature is created using the dot product (takes two equal length sequences of numbers and returns a single number).



# ReLU Layer

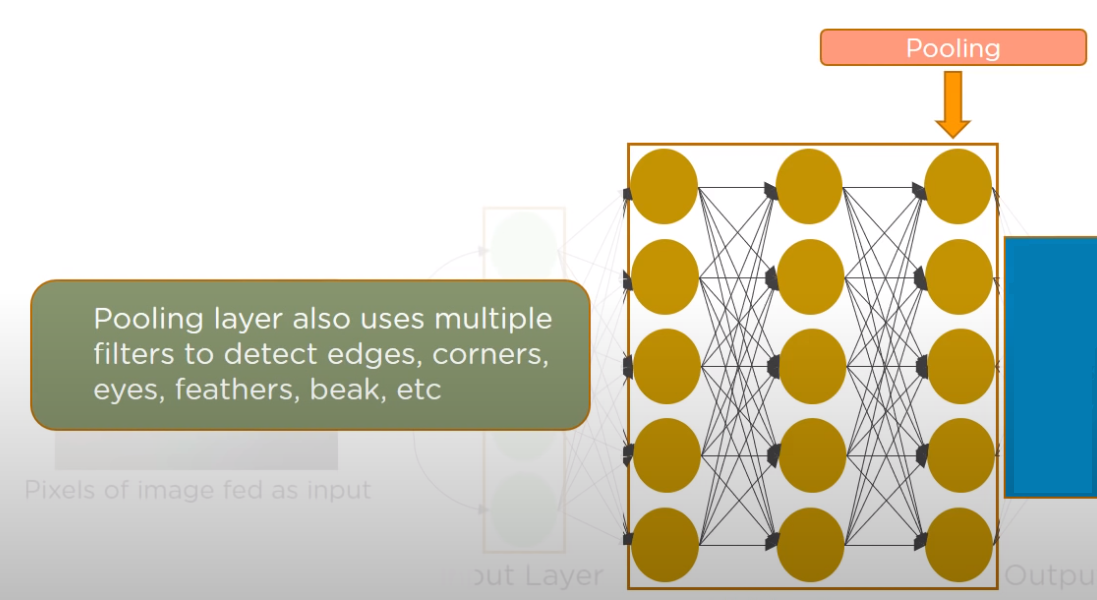
The Rectified Linear Unit. This layer has to do with the activation function that is used.

This is the math behind what makes the neurons fire. When you’re using it just by itself that’s for processing small amounts of data. Use the Atom Activation feature for large data coming in.

Once the feature maps are extracted from the convolution layer, the next step is to move them into the ReLU layer. First it’ll perform an **element wise operation** on each of the maps, then it **sets all negative pixels to 0**, this **introduces non-linearity to the network**. The word linearity is in reference to the fact that the Feature has a value, so it’s a linear feature. For example, if a Feature (like a beak, or a backslash) comes up, each feature a has a value so the closer it is to the value the more likely it is to be that feature. Using the image above as an example, if a feature has a value of one it might/might not be a beak. A -5 is definitely not. Finally, **the output is a rectified feature map**.

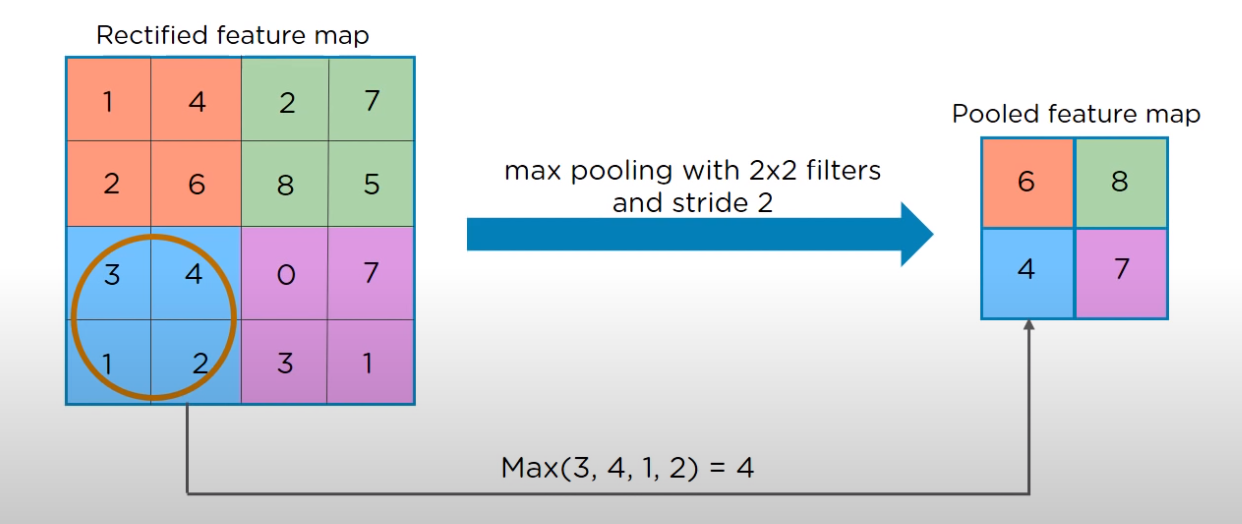
In review: a real image is scanned in multiple convolution and ReLU layers for locating features. You don’t have just one filter going across there, you have multiple filters. There isn’t just one ReLU layers, not only are there multiple features going through but also multiple ReLU layers are created for locating features.

# Pooling Layer



Pooling is neural network term for reduce. You’re pooling data together here.

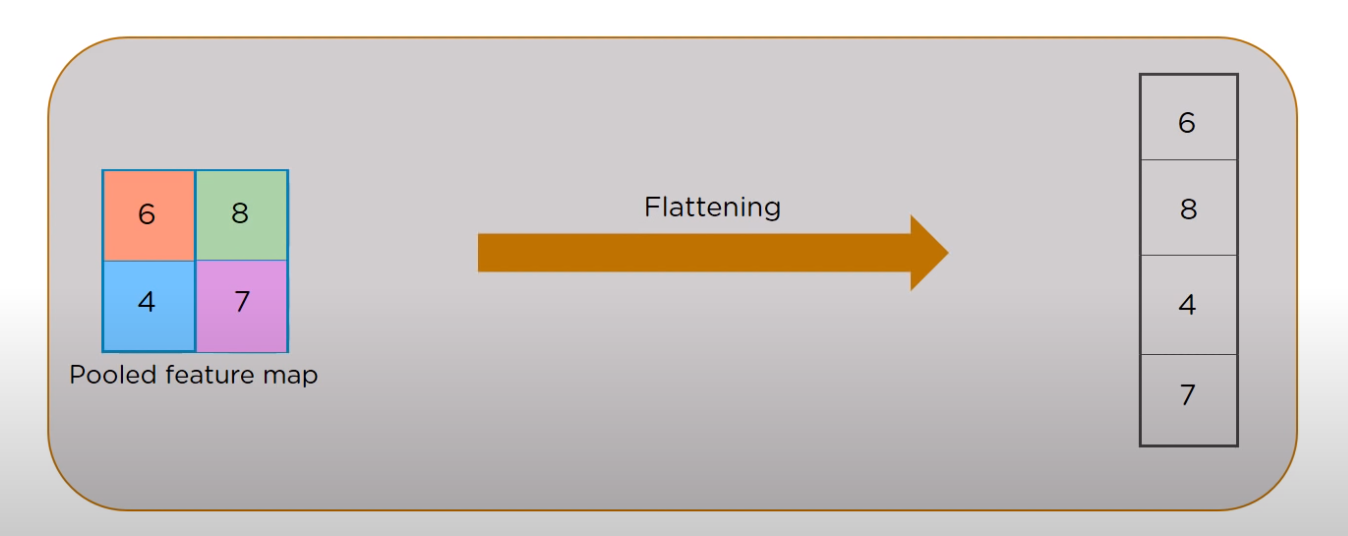
The Rectified Feature Map created in the ReLU layer goes through a pooling layer. Pooling is a down-sampling operation that reduces the dimensions of the feature map:



So we’re taking a huge amount of information and reducing it down to a single answer: this is a rose, a specific kind of bird, etc. So we have a rectified feature map, “max pooling with 2x2 filters” means you’re looking for the max value, and a “stride 2” means that instead of going one pixel at a time you’re moving 2.

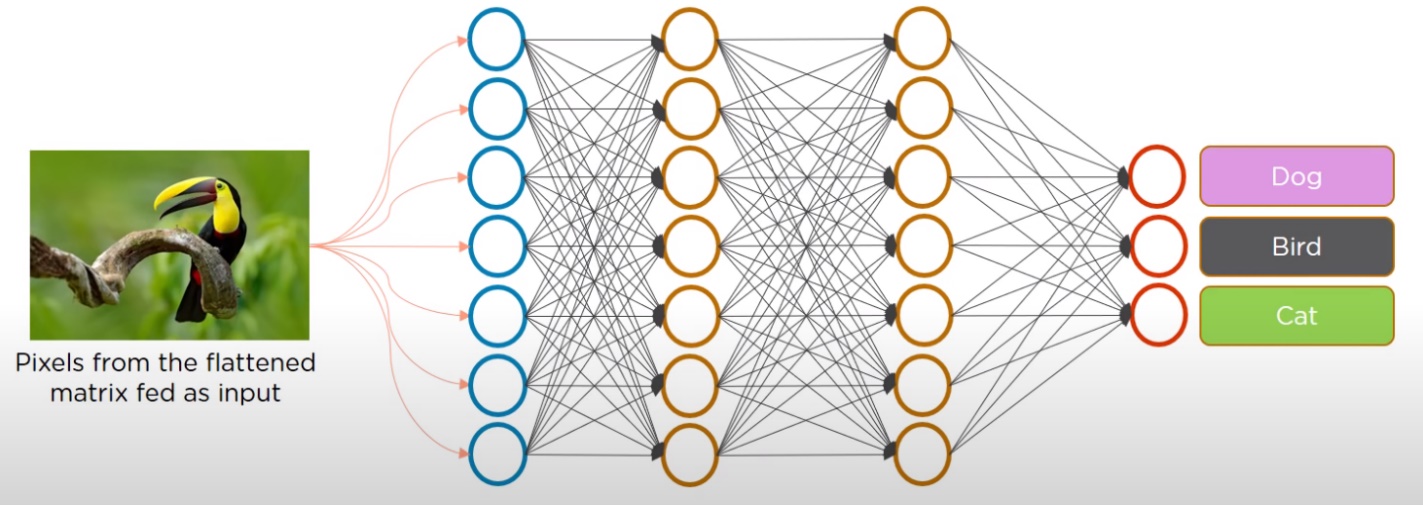
Summary so far: we have our input image come in and go through filters to twist and change that data that generations multiple convolution layers. We then take the convolution layers and run them through the ReLU setup. Once we go through the ReLU set up we have multiple ReLU layers. Then we take those layers and we pool them, there are multiple pools too. The next step is to reduce those dimensions:

## Flattening

Flattening is the process of converting all the resultant 2D arrays from the pooled feature map into a single long continuous linear vector. This combines the pooled feature map into a single input going into the final layer, the Fully Connected Layer, which will then classify the layer.

# Fully Connected Layer

The pooling layer is fed as an input to the Fully Connected Layer to classify the image. The fully connected layer forms the last few layers in the network. The input to the fully connected layer is the output from the final pooling or convolutional layer which is flattened and then fed into the fully connected layer.



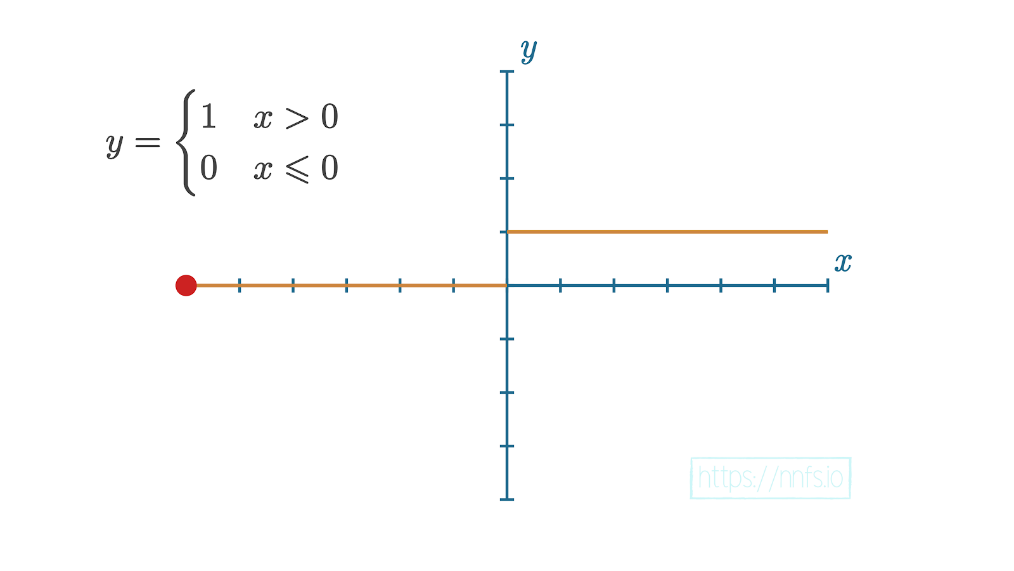
# Output layer

The output layer is what the neural network returns. I the goal is classification, and there are two classes such as “dog” and “cat” that means the output layer will have two neurons, one associated with dog and the other with cat.

# CH1 NOTES – Introducing Neural Networks

Each connection between neurons has a weight associated with it, and this weight gets multiplied by the input value. The formula for a single neuron looks like: ***Output = sum(weight\*input) + bias***(similar to the slope equation, y=mx+b). When you change the weight you’re changing the slope of the output, when you change the bias you’re moving the output up or down on the y-axis. Wights and bias’ are also known as **parameters**.

An **activation function** is like a step-function: this is mimicking if a neuron is firing or not. The activation function is usually applied to the output of the formula for a single neuron. So it would look something like ***Output =activation(output)***.

****

“To train these neural networks, we calculate how “wrong” they are and attempt to slowly adjust their parameters (weights and biases) so that, over many iterations, the network gradually becomes less wrong. The goal of all neural networks is to be able to generalize, meaning the network can see many examples of never-before-seen data, and accurately output the values we hope to achieve. Neural networks can be used for more than just classification. They can perform regression (predict a scalar, singular, value), clustering (assign unstructured data into groups), and many other tasks. Classification is just a common task for neural networks.”

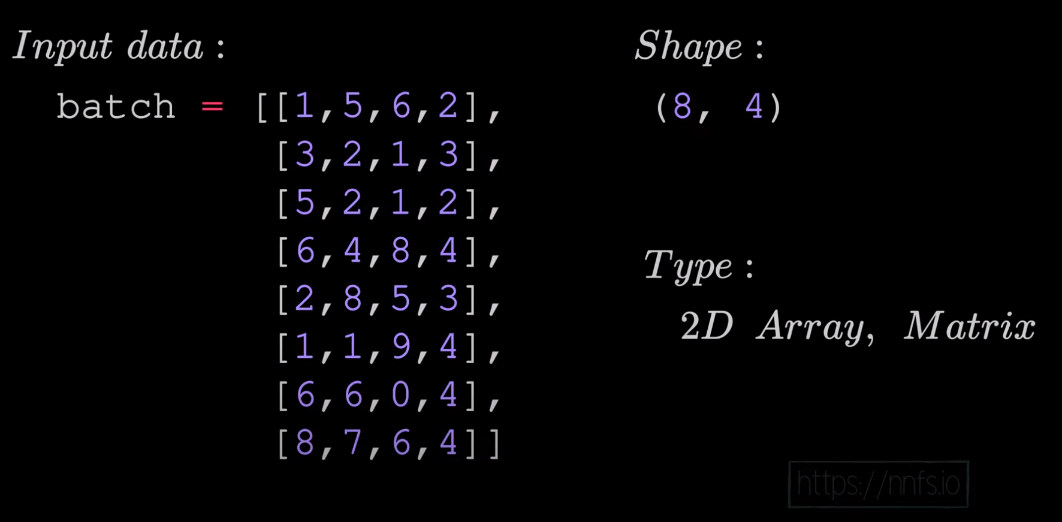
# CH2 NOTES – Coding Our First Neurons

See Practice – Ch2.ipynb in Solar-Panel-Monitoring folder.

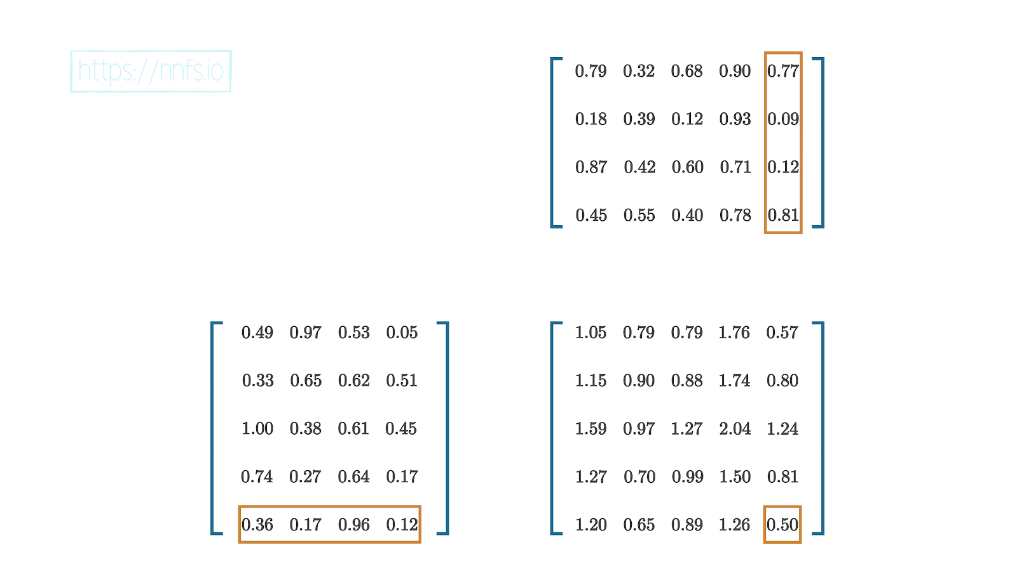
In the context of computer science and deep learning, a **tensor** object is an object that can be represented as an array.

Just as we can write a **matrix** as a list of lists in Python, we can also write a **vector** as a list or an array. Keep thinking of vectors as arrays —- a one-dimensional array is just a vector (or a list in Python). When multiplying vectors, you either perform a dot product or a cross product. A cross product results in a vector, while a dot product results in a scalar.

“In the example here, the [1, 2, 3, 2.5] data are somehow meaningful and descriptive to the output we desire. Imagine each number as a value from a different sensor all at the same time. Each of these values is a feature, and together they form a **feature set**, also called an **observation or a sample**.”

lists are useful containers for holding a feature set and multiple feature sets which make up a batch of observations. This list of lists could be made into an array since it is homologous.

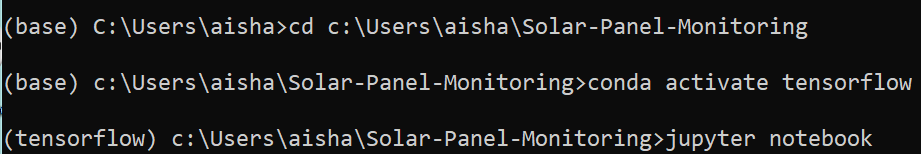
In the example bellow: the matrix product is produced by taking two matrices (the size of the second dimension on the left matrix (ex- 5,4) should match the size of the first dimension of the right matrix (ex- 4,7)) and perform dot products on all of them in all combinations, resulting in a list of lists of outputs, or a matrix; this operation is called the **matrix product:**

**Transposition** modifies a matrix in a way that it’s rows becomes columns and columns becomes rows****.

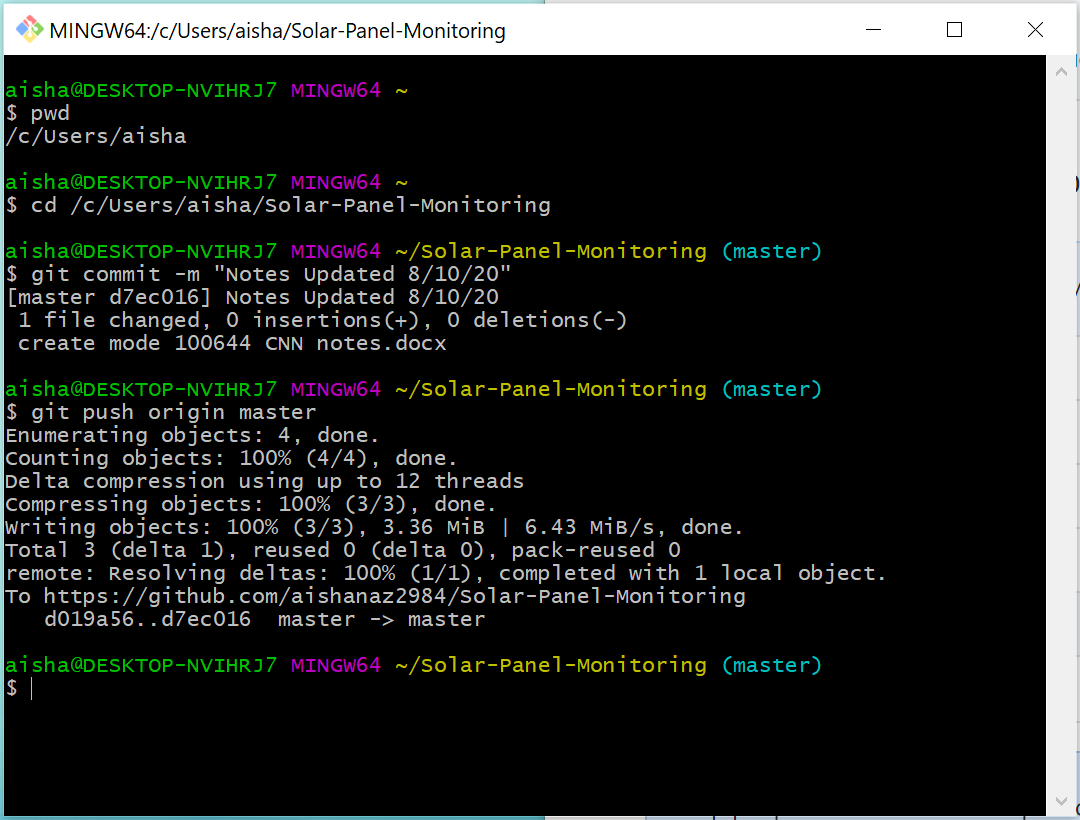
We can perform the matrix product on row vectors and column vectors instead of the dot product, which results in a matrix.

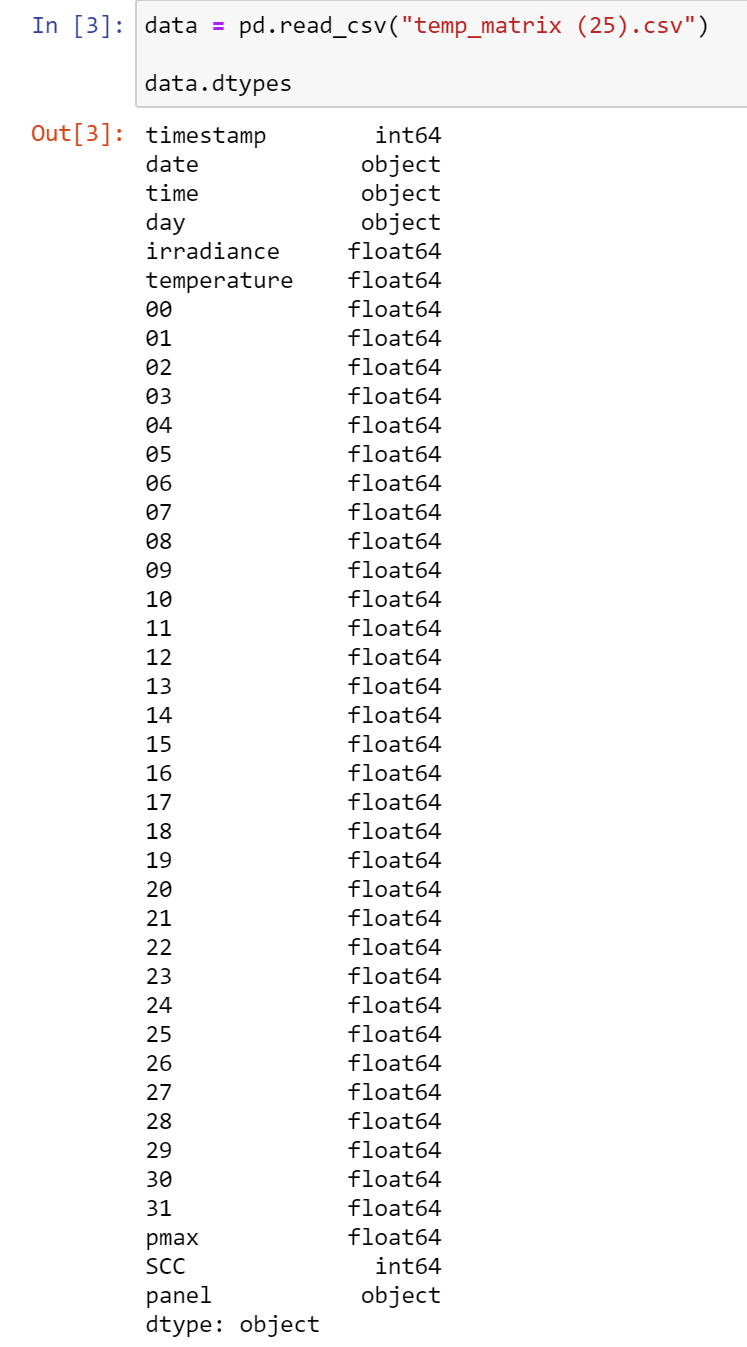
# Project Notes:

Starting project:



Pushing progress to GitHub:

Do “Git add .” first (the period implied add all, you can replace that with the name of a specific file too), before doing the Git commit command, otherwise an error will appear: “changes not staged for commit”.



These are the pandas data types.

[A resource about pandas dataframes](https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python)

Next I need to split that data into frames:

Date, time, day, timestamp, panel – data 1

Irradiance, temperature, 00-31, pmax, scc – data 2

[A resource on how to Handle Missing Data using: fillna, dropna, and interpolate](https://www.youtube.com/watch?v=EaGbS7eWSs0&t=332s)

After the missing data has been handled the concat method can be used to bring the revised data back together.

Next we’ll want to perform feature scaling but in order to do this we’ll want to separate the input and output data into different frames,

[A helpful resource explaining the difference betweek loc, iloc, and ix](https://www.youtube.com/watch?v=xvpNA7bC8cs&t=2s)

## Requirements for working with data in scikit-learn:

Notes from this video, but applying to this project: <https://www.youtube.com/watch?v=hd1W4CyPX58>

Question: is this supervised learning or unsupervised? (Supervised?)

* Each row of the data is an **instance** (also known as: sample, observation, record)
* Each column is a **attribute** also known as: (feature, predictor, independent variable, input, regressor, covariate)
* Each value being predicted is known as the **dependent variable** (also known as: target, outcome, label, response)

There are 2 types of Supervised Machine Learning:

* **Classification** is supervised learning in which the response is categorical
* **Regression** is supervised learning in which the response is ordered and continuous

(The type depends on what you are trying to do with the data)

These are the requirements for working with data in scikit-learn:

The Attributes and Dependent Variables are **separate objects**

The Attributes and Dependent Variables should be **numeric**

The Attributes and Dependent Variables should be **NumPy arrays**

The Attributes and Dependent Variables should have **specific shapes:**

* Specifically the Attribute object (in the case of our data set this is clean\_ind, and dirty\_ind), should have 2 dimensions, in which one dimension is the rows (instance- the time of day), and the other is the columns (attributes - the sensors)
* The Dependent Variable object (in the case of our data set this is clean\_dep, and dirty\_dep), should have 1 dimension, and it should have the same magnitude of the first dimension of the Attribute object (the number of rows). This way there is one response to each instance.

Understanding Input & Output shapes in CNN

<https://towardsdatascience.com/understanding-input-and-output-shapes-in-convolution-network-keras-f143923d56ca>