*The field of Computer Vision has made huge progress in the last few years. Convolutional neural networks have greatly boosted the accuracy of image recognition models and have a ton of applications in the real world. In this article I’ll cover how they work, some real-world applications, and how to code one with Python and Keras.*

**

**

Photo by [Amanda Dalbjörn](https://unsplash.com/photos/UbJMy92p8wk?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/search/photos/eyes?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

For most of us, seeing is a part of our everyday life. We use our eyes to find our way through the world around us. We use them to communicate and comprehend. I probably don’t need to tell you that vision is super important. It’s such an essential part of our day. I mean, can you even imagine, not seeing?

But what if I asked you to explain how seeing works? How do we understand what our eyes interpret? Well, first you look at something, and then… what? The brain is like a super complicated computer that developed naturally over millions of years. **We’ve gotten really good at recognizing various patterns and objects.**

**A ton of technologies have been based on natural mechanisms.** Take cameras for example. The shutter controls the amount of light, similar to the pupils that we have. The lens in cameras and eyes focus and invert images. Cameras and eyes both have some way to sense the light and convert it to signals that can be understood.

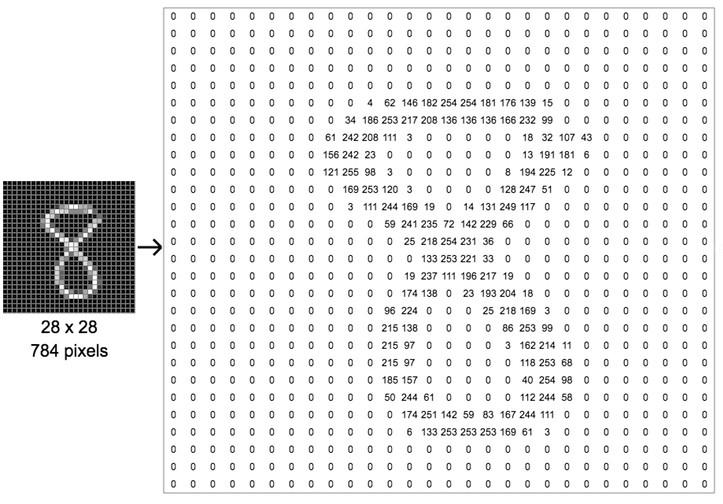




Photo by [Alfonso Reyes](https://unsplash.com/photos/xVg65Jh4CI4?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/search/photos/camera?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

But obviously, we’re not all just moving cameras with arms and legs. The cameras we currently have obviously can’t fully understand what they’re taking pictures of. If they did that would be kinda scary. **To the camera and computers, a picture is just a bunch of numbers in an array.**

****

****

Digit 8 from [MNIST](http://yann.lecun.com/exdb/mnist/) dataset represented as an array. [Source.](https://cdn-images-1.medium.com/max/1600/1*av47vApmzuM0AN21VaIcSA.png)

So how on earth can we create programs that can tell us if a dog is a dog or if a cat is a cat? This is the problem we’re trying to solve with **Computer Vision**.

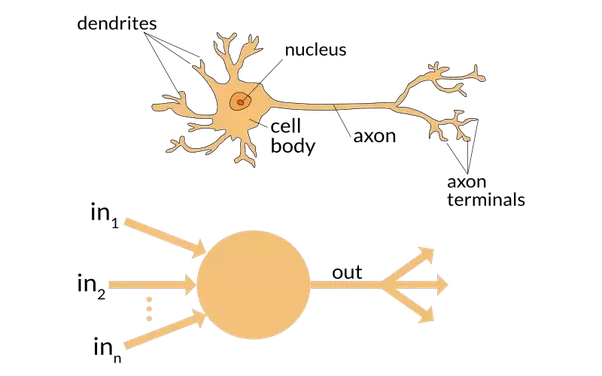
And this is how neural networks can help us out!

# **How Neural Networks Work**

Artificial Neural Networks (ANN) are programs loosely based on the human brain. **Neural nets consist of many connected neurons.** Some of these neural nets can have millions of nodes and billions of connections!

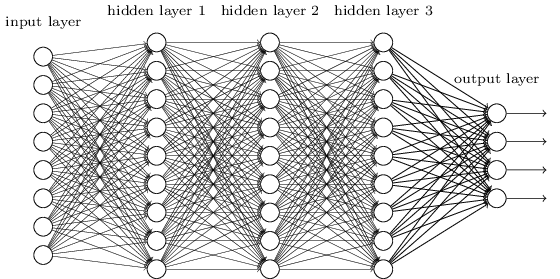
**A neuron is basically a function that takes in inputs and returns an output.**

****

****

Artificial neurons are modeled off biological neurons. [Source.](https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network)

A neuron by itself can’t do much. But the fun begins when you have a ton of neurons linked together. **Different layers/structures of neural networks let you do a ton of cool things.**

****

You can get something like this!

Each neuron is generally associated with some kind of weight. Basically when one connection is more important than another. Let’s say we have a network that wants to tell you if the picture is a [hot dog or not](https://medium.com/@timanglade/how-hbos-silicon-valley-built-not-hotdog-with-mobile-tensorflow-keras-react-native-ef03260747f3). Then we’re going to want the neurons that contain features of the hot dog to matter more than the features of a regular dog.

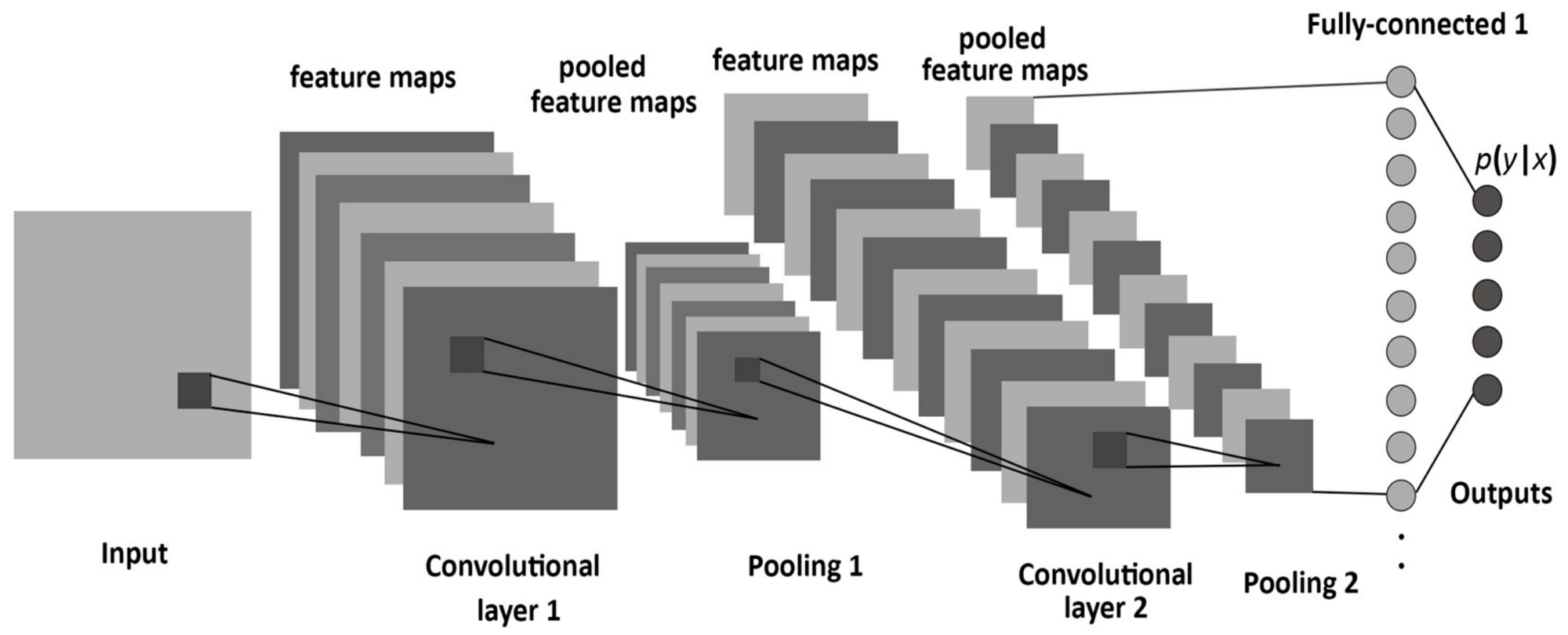
The weights of a neural network are learned through training on a dataset. It will run a bunch of times, changing its weights through backpropagation with respect to a loss function. The neural network basically goes through the testing data, makes predictions, then sees how wrong it is. Then it takes this score and makes itself slightly more accurate. Through this process, a neural network can learn to improve the accuracy of its predictions.

I won’t be covering back-propagation or loss functions in this article but there are a ton of great resources like [this](https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e) that cover these topics!

**Convolutional Neural Networks (CNN) are a special type of neural network. They perform really well when applied to image datasets.**

# **Convolutional Neural Networks**

****

****

A diagram of Convolutional Neural Networks. [Source.](https://www.mdpi.com/entropy/entropy-19-00242/article_deploy/html/images/entropy-19-00242-g001.png)

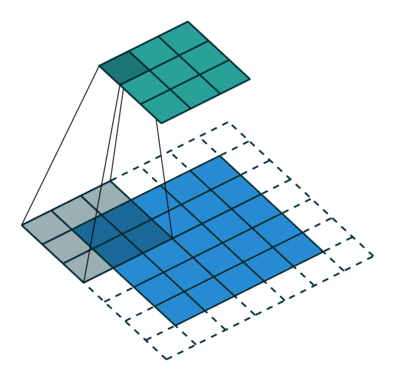
As I previously mentioned, pictures are seen by the computer as a bunch of numbers in an array. **The different layers of a CNN apply functions to these arrays to extract various features from the images and to reduce the complexity of an image.**

Let’s walk through some steps involved in training a CNN on a hot dog detector.

First, we initialize the CNN with random weights. This basically means the network is completely guessing. Once it makes its prediction, it will check how wrong it is using a loss function and then update its weights to make a better prediction next time.

**CNNs contain layers known as convolutional layers and pooling layers.** You can imagine what happens during the convolutional layer this way.

Pretend you have a picture and a magnifying glass. Put your magnifying glass at the top left corner of the picture and look for a specific feature. Note down if it’s there or not. Repeat this process by slowly moving through the image.



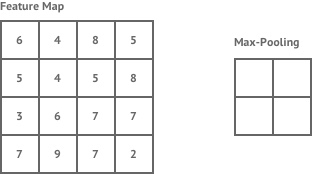
Visualizing feature extraction in a convolutional layer. [Source.](https://github.com/vdumoulin/conv_arithmetic)

**The convolutional layer creates a bunch of feature maps.**

For a CNN that is used to describe different images like animals or faces. The features that the first convolutional layer looks for could be the different edges of the objects. It’s like making a list of the different edges in the picture. This list is then passed on to another convolutional layer which does a similar thing, except it looks for bigger shapes in the image. This could be a leg on an animal or an eye on a face. Ultimately, the features are taken in by a fully-connected layer which classifies the image.

Pooling layers are also used with convolutional layers. This takes another magnifying glass but it doesn’t look for features. Instead, it takes the biggest value in a region to reduce the complexity of the image.

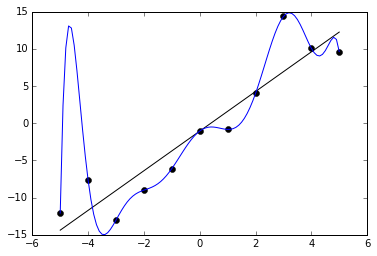




Pooling in a CNN. [Source.](https://shafeentejani.github.io/assets/images/pooling.gif)

This is useful because most images are big. They have a ton of pixels which makes working with them hard for the processor. **Pooling lets us reduce the size of the image while still keeping most of the important information.** Pooling is also used to prevent overfitting, which is when the model becomes too good at identifying the data we train it on and doesn’t work well for other examples we give it.





An example of overfitting on a linear dataset. [Source.](about:blank)

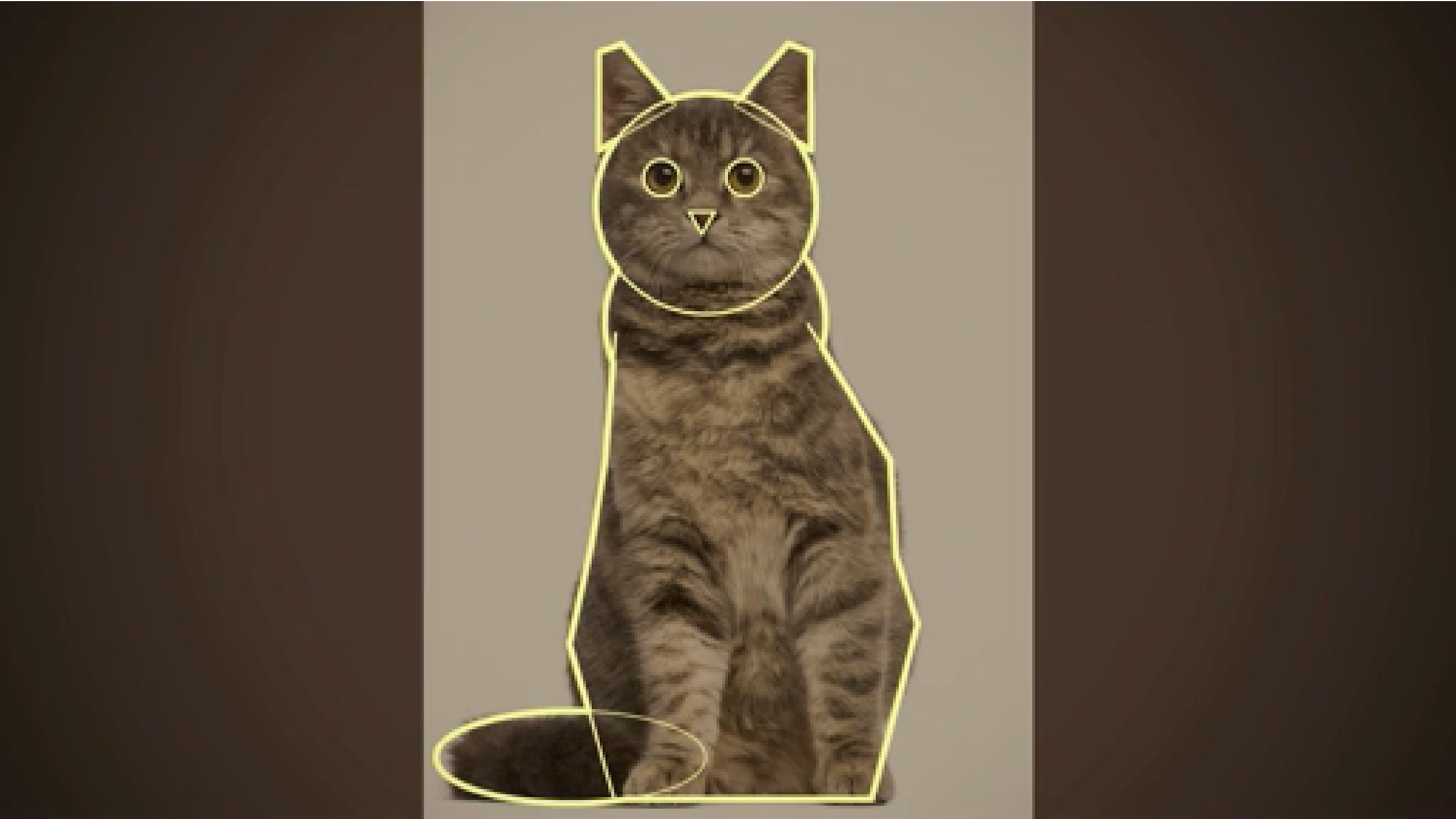
As you can see in the picture, the data in this plot can be represented by a linear line. The model in blue is obviously hitting all of the data points, but if we try to get it to predict something else, it won’t be able to. In terms of our CNN, this might mean that it will be super accurate with images we train it with, but won’t be able to give us correct predictions on other pictures.

Finally, we flatten the structure of the CNN into one super long feature. We’re basically just putting all our data together so we can pass it on to a fully-connected layer to make predictions.

## **Why are Neural Nets Better?**

Pretend for a moment that we weren’t using a neural network. How would we approach this problem? Let’s say we’re trying to write a program that identifies cats. We can try to represent the cat by looking for certain shapes.

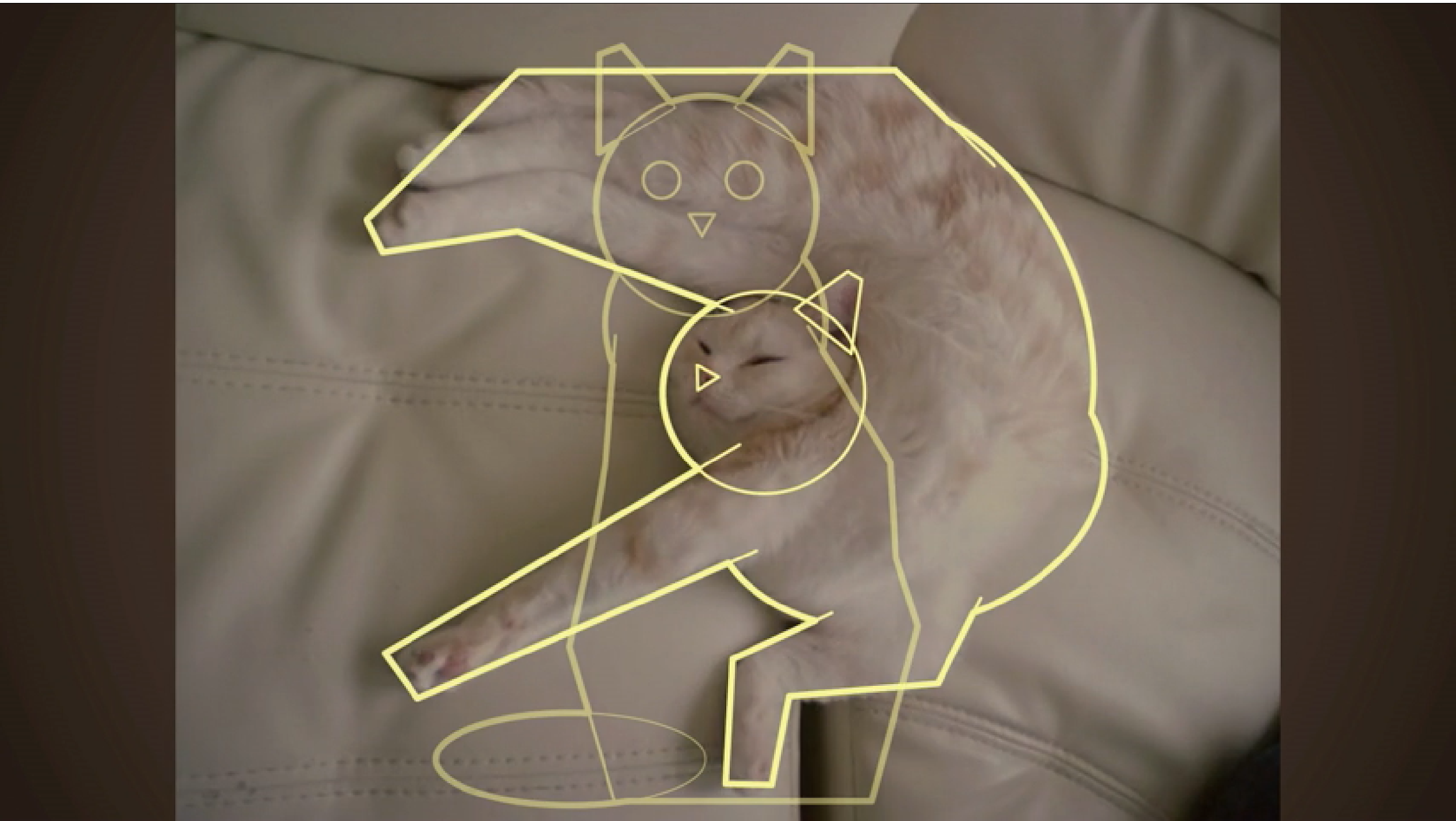




Cat shapes from Computer Vision Expert Fei-Fei Li’s TED Talk.

Seems easy, right? But wait a minute. Not all cats look like this. What if you have a cat stretched out? We need to add some more shapes.





More cat shapes from Computer Vision Expert Fei-Fei Li’s TED Talk.

By this point, it should be pretty clear that telling the computer to look for certain shapes won’t work. Cats come in all shapes and sizes. And this is assuming we’re only looking for cats. What if we want a program that can classify all kinds of pictures?

This is why using neural nets is so much better. You can let the computer set its own rules. **By using highly advanced algorithms, neural networks can classify images with a crazy high level of accuracy.** Some models have already beaten humans at this task!

# **Some Really Cool Ways We’re Applying Computer Vision**

As algorithms become more efficient and hardware more powerful, we may be able to accomplish tasks with neural nets that are closer to the realm of science fiction. But that doesn’t mean we’re not doing a bunch of cool things with the technology right now!

## **Retail**

You’ve probably heard of it in the news. [**Amazon Go**](https://www.amazon.com/b?ie=UTF8&node=16008589011), the e-commerce giant’s cashier-less grocery stores. You walk in, pick up some stuff, and then walk out. The system automatically charges you for what you take. **Cameras that cover the ceilings keep track of the items you pick up.** While the system isn’t perfect and possibly vulnerable to shoplifting. It’s going to be super interesting to see how this idea develops over the next couple of years.

## **Autonomous Vehicles**

In my opinion, self-driving cars are some of the coolest things that are being worked on right now. [**Waymo**](https://waymo.com/)**,** originally Google’s self -driving car project, **Uber, and Tesla are some companies currently developing vehicles that can navigate roads on their own.**

Waymo’s fleet of self-driving cars has covered over 10 million miles of road! The average travels about 12,000 miles every year. In total, that’s more than 800 years of driving!





One of Waymo’s self-driving cars. [Source.](https://waymo.com/ontheroad/)

## **Healthcare**

In healthcare, CNNs are being used to identify many different kinds of diseases. **By training on certain datasets of cancers or other medical conditions, neural networks can figure out whether or not something is wrong with a high accuracy rate!** By having a neural network extract features and find patterns in the data, it can make use of information in pictures that we never would have thought of!

Microsoft is developing its [**InnerEye**](https://www.microsoft.com/en-us/research/project/medical-image-analysis/)project which helps clinicians analyse 3D radiological images using deep learning and CNNs. This helps medical practitioners extract quantitative measurements and plan effectively for surgeries.

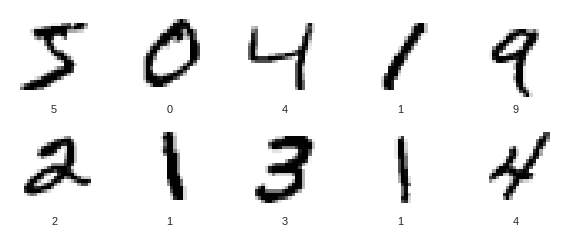
# **Creating a Convolutional Neural Network with Keras**

Now that we understand some of the intuition behind how a CNN is supposed to work. We can create one in **Python** with **Keras**, a high-level API written in Python. Keras will help us write easy to understand and super readable code.

You can start by installing [**Anaconda**](https://www.anaconda.com/download/) and running conda install keras in the command interface. Then you can use **Jupyter Notebooks** to begin programming with Python. You can also use [Google’s Colaboratory](https://colab.research.google.com/drive/1oSE2zRroFcjZkBrrWv-p0tnWR7_yOYrH) if you want to run everything in the cloud.

We’re going to be working with the **MNIST dataset** that is part of the Keras library. It contains *60,000 training examples and 10,000 test examples of handwritten digits.* Let’s get started!





The first few training examples in the MNIST dataset.

First, we’re going to want to import everything we need from the Keras library. This includes the Sequential model which means that we can build a model really easily by adding layers. Next we‘re going to import the Conv2D (Convolution2D), MaxPooling2D, Flatten, and Dense layers. The first 3 are self-explanatory and the Dense layer helps us build our fully-connected layer.

We’ll need **Keras Utils** to help us **encode the data** to ensure that it’s compatible with the rest of our model. This makes it so that the digit 9 isn’t treated as better than 1. Lastly, we’ll import the MNIST dataset that will be used to train the model.

After importing the dataset we’ll need to split it into the training data and the test data. The training data is what we’re going to teach the neural network with. The test data is what we’re going to use to measure accuracy. We’re going to *reshape* the data to match the format needed for **TensorFlow backend.** Next we’ll *normalize* the data to keep the range of values around 0 to 1. and *categorically encode* the MNIST labels.

Great! Now we can start building our model. We’ll start off by creating a Sequential model which is a *linear stack of layers*. As you can see in the code below, this makes it super easy for us to add more layers to the model.

After we’re done building the model, we’ll compile it. This model uses the *Adam optimizer* which is a type of gradient descent algorithm used to adjust the weights. The loss function that our model uses is *categorical cross entropy* which tells our model how far off we are from the result. The metrics parameter is used to define how the performance will be evaluated. It’s similar to a loss function but won’t be used during the actual training process.

We’ll fit or train our model on the training set. The *batch size* determines how many images we’ll consider during every iteration. The number of *epochs* determines how many times the model with iterate through the entire set. After a certain number of epochs, the model will essentially stop improving.

The *verbose* value determines whether or not the model will tell us the progress of the model and the *validation data* determines how the model will evaluate its loss after every epoch.

Finally, we’ll print out how accurate our model is. The final result should be between 98% and 99%

You can find the complete code on [my GitHub](https://github.com/uyxela/mnist-keras) or you can run it on [Google Colaboratory](https://colab.research.google.com/drive/1oSE2zRroFcjZkBrrWv-p0tnWR7_yOYrH).

# **Key Takeaways**

* Neural Networks are loosely based off of the way our brain interprets information.
* Convolutional Neural Networks work particularly well with images.
* There are a ton of real world applications for Computer Vision.