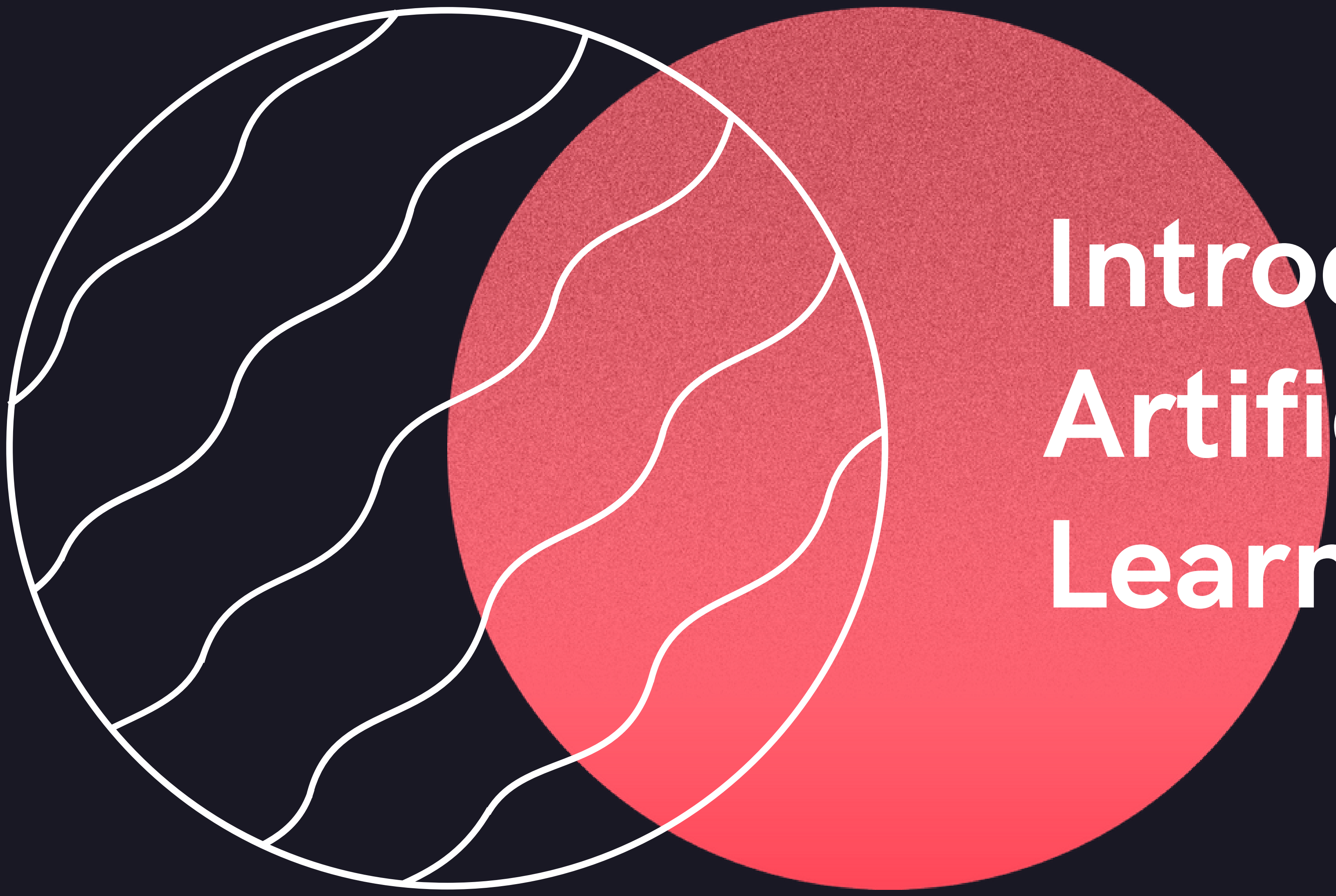

Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning

- In Keras <u>Dence</u> is define a layer of connected neurons
- Successive layers are defined in sequence, hence the word sequential.
- The np.array is using a Python library called numpy that makes data representation particularly enlists much easier.
- The optimizer generates a new and improved gues
- Convergence is the process of getting very close to the answer

- Fashion-MNIST is available as a data set with an API call in TensorFlow. We simply declare an object of type MNIST loading it from the Keras database. On this object, if we call the *load data* method, it will return four lists to us. That's the training data, the training labels, the testing data, and the testing labels.
- We have 3 layer NN this time. The last layer has 10 neurons in it because we have ten classes of clothing in the dataset. This should always match.
- Our images are 28 by 28, so we're specifying that this is the shape that we should expect the data to be in.
- Flatten takes this 28 by 28 square and turns it into a simple linear array.
- Hidden mid-layer has 128 neurons in it.
- Sequential: That defines a SEQUENCE of layers in the neural network
- Flatten: Remember earlier where our images were a square, when you printed them out? Flatten just takes that square and turns it into a 1 dimensional set.
- Dense: Adds a layer of neuronsEach layer of neurons need an activation function to tell them what to do. There's lots of options, but just use these for now.
- Relu effectively means "If X>0 return X, else return 0" -- so what it does it it only passes values 0 or greater to the next layer in the network. means negative values will be thrown way.
- Softmax takes a set of values, and effectively picks the biggest one, so, for example, if the output of the last layer looks like [0.1, 0.1, 0.05, 0.1, 9.5, 0.1, 0.05, 0.05, 0.05], it saves you from fishing through it looking for the biggest value, and turns it into [0,0,0,0,1,0,0,0,0] -- The goal is to save a lot of coding!
- on_epoch_end called when the epoch finishes.

The training loop does support callbacks. So in every epoch, you can callback to a code function, having checked the metrics. If they're what you want to say, then you can cancel the training at that point. "callbacks="

when we reshape eg: (28 ,28, 1). Here '1' is just means that we are tallying using a single byte for color depth. As we saw before our image is our gray scale, so we just use one byte.

Convolution: image processing, it usually involves having a filter and passing that filter over the image in order to change the underlying image. For every pixel, take its value, and take a look at the value of its neighbours. If our filter is three by three, then we can take a look at the immediate neighbour, so that you have a corresponding three by three grid. Then to get the new value for the pixel, we simply multiply each neighbour by the corresponding value in the filter. "Conv2D"

Pooling is a way of compressing an image. A quick and easy way to do this, is to go over the image of four pixels at a time, i.e, the current pixel and its neighbors underneath and to the right of it. Of these four, pick the biggest value and keep just that. MaxPolling2D. We moving 1 pixels around so dimantion 2 less 1 for x 1 for y

A really useful method on the model is the model.summary method. This allows you to inspect the layers of the model, and see the journey of the image through the convolutions.

Previous slide explains convolutions and got a glimpse for how they worked. By passing filters over an image to reduce the amount of information, they then allowed the neural network to effectively extract features that can distinguish one class of image from another. Pooling compresses the information to make it more manageable. This is a really nice way to improve our image recognition performance.

The Keras API gives us each convolution and each pooling and each dense, etc. as a layer. So with the layers API, I can take a look at each layer's outputs, so I'll create a list of each layer's output. I can then treat each item in the layer as an individual activation model if I want to see the results of just that layer.

Under the hood, TensorFlow is trying differen filters on your image and learning which ones work when looking at the training data. As a result, when it works, you'll have greatly reduced information passing through the network, but because it isolates and identifies features, you can also get increased accuracy.

It has feature of the image generator is that you can point it at a directory and then the sub-directories of that will automatically generate labels for you.

ImageDataGenerator class available in Keras.preprocessing.
flow_from_directory method for get it to load images from that directory and its sub directories.
It's a common mistake that people point the generator at the sub-directory. It will fail in that circumstance. You should always point it at the directory that contains sub-directories that contain your images. The names of the sub-directories will be the labels for your images that are contained within them.

The RMSprop, where you can adjust the learning rate to experiment with performance. This looks a little different than before when you called model.fit. Because now you call model.fit_generator, and that's because we're using a generator instead of datasets.

Each epoch is loading the data, calculating the convolutions and then trying to match the convolutions to labels.

We saw the training with a validation set, and we could get a good estimate for the accuracy of the classifier by looking at the results with a validation set.

Using these results and understanding where and why some inferences fail, can help you understand how to modify your training data to prevent errors like that.