University of Essex
MSc Artificial Intelligence
Module: ML_PCOM7E July 2025 B
Unit 11: Individual Presentation Transcript

Transcript

Note: This transcript was automatically prepared using Adobe Premiere Pro and corrected only for transcription typos.

00:00:00:00 - 00:00:39:00

Hello. This is Matthew Bullen, and this is my Machine Learning PCOM7E July 2025 B final presentation. Let's get started. The task at hand was to build, train and evaluate the predictive accuracy of a machine learning model using the CIFAR-10 image recognition dataset. The CIFAR-10 dataset is a collection of 60,000 images, all small thumbnails divided across ten categories such as bird, car, cat, etc.

00:00:39:02 - 00:01:06:25

The methodology, or the preliminary steps of the methodology for the project, were first to open a new Google Colab notebook, load the Python coding packages needed for the model, and download the CIFAR-10 dataset. Once the dataset was downloaded, I ran it through two preliminary pre-processing or data cleaning steps. The first was to normalize the pixel RGB values for each image

00:01:06:28 - 00:01:40:17

so they fell between 0 and 255 for the red, green, and blue color channels. And this was just to ensure that every image in the data set would be usable. The second pre-processing step was to convert the text labels for each image into numerical 0 or 1 values. So, for example, an image of a car would receive an encoded label value of one

00:01:40:20 - 00:02:00:25

because the label car applied to it, while every other potential label that could apply to that same image would receive a value of zero in the vector array, the encoded vector array, that describes the image’s label.

00:02:00:28 - 00:02:36:11

And with that done, and before jumping into the results of model training, I think it's worthwhile to go over why machine learning models need to use training sets separate from testing sets. One of the biggest reasons that training sets need to be separated from validation sets is in order to avoid overfitting a model. So, for example, if you only test a model using the same data that you trained it with, you're essentially answering your own question.

00:02:36:13 - 00:03:14:15

You'd have no way, no objective way, to know or to measure how accurate the model would be when faced with data points it has never encountered before. And as related note, it's a good practice to separate validation sets from training sets in order to reduce the risk of inadvertent human bias or data cherry picking. In the model's training set, an example of bias or data cherry picking would be loading a model's training set with only duplicates of the very same image.

00:03:14:18 - 00:03:56:17

That model would most likely be 100% accurate, but have essentially no predictive value whatsoever, because its underlying training data was so massaged and skewed by human intervention. Also, just as a programing side note, a completely separate and independent validation or testing set means that you can iterate over the training data as much as you'd like without any concern about it affecting or influencing the results of validation set [testing].

00:03:56:20 - 00:04:11:03

And having that separate or independent validation set also makes it easier to make iterative adjustments to the parameters you use to train the model on the training data on the fly as you go.

00:04:11:05 - 00:05:01:04

For this project, I use a convolutional neural network, or CNN. I use a CNN, because they're industry standard with robust code packages available. They're well understood in the research literature. The code that's used to create and train them is readily amenable to hyperparameter adjustments during training and during validation testing, which means that it's very easy to swap in and out different pieces of the model as you're working with it, which makes it much faster to land on the most accurate, or the most likely accurate, model available for the dataset you're working with.

00:05:01:07 - 00:05:35:03

Related to that, let's briefly go over what exactly a CNN is in terms of its architectural components. Broadly summarized, the core component of a CNN is a kernel or a matrix, a matrix of weights. That kernel passes over larger matrices of input data sequentially. And as it does, it performs element-based multiplication using whichever values it's encountering inside the data window it's working with at the time.

00:05:35:06 - 00:05:59:12

The results of that processing are then assigned to a single output value that is loaded into a results matrix, which is then used by subsequent steps in the model. And on this slide, we have an example of a CNN kernel in action. (Just to be scrupulous, this is an example that I copied from a Google example.)

00:05:59:12 - 00:06:38:05

I did not create this illustration, but it is an excellent illustration for seeing how a kernel moves over training data and then outputs results, with the blue matrices on the left representing the training data and the kernel moving across it, and the green matrix on the right representing the output values of those operations. Then on this slide, we see an excerpt of some of the code used to create the model itself.

00:06:38:08 - 00:07:05:06

Again, this is just an example that I've copied from a Google example, but it was the basis for the code that I used inside the project itself. And I include this code snippet because the Python code used to create the CNN model in this project is very straightforward. It's very human readable. And you can see each layer and component of the model listed in sequence.

00:07:05:09 - 00:07:57:17

Moving on. Training a model depends on configuring its input hyperparameters. So for a starting point, I defaulted to some fairly standard hyperparameter settings. As we can see from the slide, I started with 5 epochs, a batch size of 64 images, the Adam optimizer function, the categorical cross-entropy loss function, which is also industry standard and well-known, and the Softmax function for the final activation layer, again, because it's very standard, very well-known, and readily amenable to outputting human readable

00:07:57:20 - 00:08:29:07

validation testing results. And again, just to be scrupulously accurate, these are default values that I then modified extensively throughout the course of the project. But I got the idea for them from looking at a Google example. So moving on to model training, how did I train the model? The model was trained eight times using different variations on that initial set of hyperparameters from the previous slide.

00:08:29:10 - 00:08:56:00

For a quick summary, the first three runs stuck with the same default hyperparameters, but vary the number of epochs starting with 5, then 25 and 50. We'll see this later on, but I determined that 25 epochs looked like the sweet spot, or the ideal number for the greatest accuracy of model validation testing. So I stuck with that going forward.

00:08:56:02 - 00:09:24:28

After settling on using 25 epochs, I then varied model training by adjusting dropout rates between 0.5 and 0.25, and by swapping out or swapping in different optimizer functions, such as the AdamW optimizer function, which is the variation on the first Adam function used, as well as the Stochastic Gradient Descent or SGD optimizer function.

00:09:31:06 - 00:10:02:24

So with that in front of us, I think it would be worthwhile to run through the model training results quickly. And you'll see from the very first training run of only 5 epochs that this was a test run or a dry run. While it was far more accurate than random chance, just on its face, if you have very few epochs and all default values, it was highly unlikely that this set of parameters would result in the most accurate model possible.

00:10:02:27 - 00:10:49:03

So I then reran the training using 25 epochs, which did increase accuracy substantially to approximately 75%. But when I increased the number of epochs to 50, I quickly noticed that the model's accuracy remained fairly steady at above 72%, but it did not exceed the 74% from the previous run. And given that the accuracy had declined slightly despite doubling the number of epochs, I took that as a sign that the model had reached a point of diminishing returns for using additional epochs, and that more epochs were not likely to provide much value in terms of increased validation testing accuracy.

00:10:49:06 - 00:11:13:07

So after settling on 25 epochs as the working number to use for further training runs, I reran the model using 25 epochs, while adjusting the dropout rate to 0.25, the dropout rate being the number of processing nodes that are culled

00:11:13:10 - 00:11:51:09

from model layer iterations. But, as we saw before, model accuracy remained approximately the same at around 72%. There was no significant increase in accuracy, which led me to the conclusion that any further variations on the existing, or the already used, hyperparameters were unlikely to increase model accuracy, no matter how many minor [hyperparameter] tweaks or how many further training runs might have been undertaken.

00:11:51:11 - 00:12:17:18

So instead of tweaking the existing hyperparameters, I swapped out the optimizer function. Instead of using the Adam optimizer function, I switched over to AdamW, which is an expanded and related optimizer function. Again, as before, model accuracy remained approximately the same at around 73%.

00:12:17:20 - 00:12:48:23

I then revamped the same training session using a tweaked dropout rate of only 0.25 instead of 0.5, but again, no improvement to the model's accuracy. In fact, there was a minor decrease in accuracy to around 72%, which led me to the conclusion that there would likely be no significant difference worth investigating between the Adam versus AdamW optimizer functions.

00:12:48:25 - 00:13:35:02

Then, last up, I ran two training runs using 25 epochs, but swapping in the Stochastic Gradient Descent optimizer function. And I chose the SGD optimizer function because, well, just as a general matter, it's fairly efficient, comparatively speaking, for larger datasets like the CIFAR-10 dataset with 60,000 entries, but more importantly because it uses smaller batches. Because it uses smaller batches, that can help reduce the risk of outlier points or other aberrations or unexpected variations in the underlying data radically skewing the results.

00:13:35:04 - 00:14:32:07

It helps reduce the importance of outliers so that the substantive or most common or most consistent part of the data takes prominence in the model [results]. And as we've seen before, adjusting the dropout rate, even using the SGD optimizer, did not make a material difference in model validation testing accuracy. And as a general matter, after using three different optimizer functions, each with largely similar accuracy results, my takeaway was that it could be possible that the data set itself might not be amenable to a much higher model accuracy value range than what we've already seen, which would be ranging between 72 and 74%.

00:14:32:10 - 00:15:05:01

And here on this slide, we see each of the training runs with their validation accuracy values. And just reiterate, 25 epochs with a 0.5 dropout rate, using the Adam optimizer was the most accurate at 74.19%. So what does all of this mean? Or what would be the takeaways from these training runs? I think the first takeaway would be that increasing the number of epochs can increase accuracy.

00:15:05:03 - 00:15:35:24

And in this case, we found the greatest accuracy after 25 epochs, but it decreased [in] accuracy at 50 epochs. Or, in other words, it had a point of diminishing returns where further epochs did not provide greater accuracy, and, in fact, started to impair accuracy. Similarly, decreasing the dropout rate from 0.5 to 0.25 did not appear to provide any significant improvement in model accuracy.

00:15:35:27 - 00:16:10:18

Using the Keras AdamW optimizer rather than the Adam (no “W”) optimizer likewise did not seem to have much effect on model accuracy. And the same thing for using Stochastic Gradient Descent: the SGD optimizer provided remarkably similar values in terms of model validation accuracy, but it did not dramatically increase or decrease accuracy, so it had no particular effect in either direction.

00:16:10:20 - 00:16:45:21

Finally, what would be the lessons learned, or how could the results of these training sessions be [related] to findings in the literature? And I think there are several, first among them, tuning hyperparameters. It's worthwhile as an exploratory effort to see

if model accuracy can be increased. But more is not always better. More tuning does not necessarily result in greater model accuracy.

00:16:45:23 - 00:17:26:08

Similarly, model accuracy improvements aren't necessarily linear in relation to each other. So, for example, increasing every configurable hyperparameter value will not necessarily increase model accuracy. It could well be the case that decreasing values for certain parameters in combination with increased or omitted values for other hyperparameters could increase accuracy. So the relationship is more interconnected than it is linear.

00:17:26:11 - 00:18:06:16

And finally, I think, a valuable takeaway is that optimizer functions by themselves are not silver bullets. Several should be tried and the results for each compared, and it could well be the case for many data sets that certain types of optimizer functions simply aren't cut out for the type of data you're working with.

That’s my presentation. Thank you.