Annette Bazan

ITAI 2376

02/16/2025

Lab 04 Reflection Journal: CNN

Lab 04 is centered around CNN or Convolutional Neural Networks and works with MNIST dataset. By working through the dataset, the architecture was something tweakable, training the CNN model with different parameters to get a high accuracy to analyze the performance of the model. In my reflective journal I will go over new concepts and concepts I now truly understand not just in theory but with hands-on work.

Working with the keras. Sequential model was challenging at first my code was getting many errors. But I had to understand it better, so I imagine its like a toy train with different types of cars connected together in a line. Each layer in the model would be like each car of the toy train. The input layer would be the engine car that starts the train. Taking in the image we want to analyze. The convolutional layers are like cargo cars that have special filters. The filters look for specific features /patterns in the image like curves or even edges. Now for the pooling layers that shrink the size of the image while keeping important information. So, these would be smaller cargo cars, and this makes them go faster and be lighter. While the flatten layer is a connector car that transforms information into a single line making it easier to process. The dropout layer is a random switch which turns off some parts during training. Helping the model to learn to be more robust and not rely too much on any single part. Now the dense layers I would compare it to the passenger cars on a train, it is where all the information is combined and processed to decide. An output layer would be the caboose which is at the end of the toy train. It gives the final answer/prediction. The keras. Sequential model reminds me of a toy train which made it much easier to understand. The layers are different cars that work together in sequence to process information and make predictions. Each layer has a specific job and are connected in a linear order to form the complete model.

When I started playing with the parameters, I encountered so many type errors and runtime errors and had to play with my parameters and add code and take some code off just to get it run through and show me my loss and accuracy. The first Conv2D layer is like adding a layer with special magnifying glasses(filters) that help see corners or edges in the shapes. The maxpooling2D layer makes it easier to handle while keeping the important information about the images/shapes. Now adding another Conv2D layer means I am adding more filters to help see more details and patterns. So, adding another Maxpooling2D layer is going to make it even smaller and more manageable. Now when I

added my third that's when I had issues but realized my numbers were the issue and the spelling in the code and I had an indent, so these were just user errors in my syntax that I kept running into at first. I was frustrated and almost didn't add my parameters but I'm glad after checking indents, spelling and changing the numbers a few times for it to run correctly. I was able to move forward and see that with my filters it gave me high accuracy at the end of all my inputs. By having the 3 maxpooling2D layer and Conv2D it helps the model learn more features and reduce the size of the information from the dataset to make the model more efficient.

For the optimizer Adam I changed to RMSprop. RMSprop automatically adjusts the learning rate for each parameter based on dimensions of recent gradients. This can lead to faster convergence and better performance compared to using a fixed learning rate. RMSprop is usually preferred for RNNs but it can sometimes perform better on image classification tasks as well. It seemed like a solid alternative to Adam, and I believe it attributed to my high accuracy at least partly. The other choice I made was using Sparse categorical cross-entropy. Which avoids the extra step of one-hot encoding, saving memory and computation time. The code became slightly simpler as I took out the need to use to_categorical function. It directly works with integer labels which was the format of the MNIST dataset's target labels. After I changed it to RMSprop and sparse categorical cross-entropy the model did not run properly, and I had to turn part of the code in an earlier cell into markdown instead of code so it would run properly and display the outputs.

Model.fit () is a core function that starts the training process you can think of it like a student beginning their studying with a textbook. x_train and y_train are the training data and labels. I think of them as the textbook content and the answers at the back of the book. Batch_size determines how many pages of the textbook the student can study at a time before taking a break to understand. I used batch_size = 128 which means the student would study 128 images/text before updating their understanding. The larger batch_size can speed up the training, but it might require more memory. While the flip side for a smaller batch_size can lead to more noisy updates but it might generalize better. Epochs represents the number of times a student reads the entire textbook. So, I changed epoch=15 meaning the student will go through the entire training dataset 15 times. More epochs can potentially lead to better training but also increase the risk of overfitting. Which happened when I changed the epoch to 15. I ran into errors and had to change the parameter for the epochs to equal 10. Validation_split=0.1 which to my understanding is like setting aside 10% of the textbook pages as practice questions. After each epoch(chapter) the student tries to answer these practice questions to see how well they are learning and to avoid memorizing the textbook instead of understanding the concepts. By changing the batch size to be larger the risk you take is missing some details or getting

overwhelmed with too much information at once. But it can speed up the overall learning process. While smaller batch size means it can lead to better learning but might take longer overall. The downside to more epochs is overfitting which I encountered, and I had to spend time fixing the code and figuring out what was the balance between too much and just right. But fewer epochs can cause underfitting which means it might not have learned enough to perform or grasp the concepts.

Overall, I learned a few new concepts and tried to make it more understandable after reading the notebook as if I was explaining it to my son. To help me understand why I kept running into errors, some of them were simple spelling issues, indents that were not needed and removing some code, adding new code and a new import to run the optimizer and sparse categorical cross-entropy, it took longer than I like to admit. After a few hours checking and trying to run code I had to use google to search for more material to read and help my understanding. I read about transfer learning while I was looking for help to explain what I was missing to make the code run properly. It is an aspect of CNNs I would like to explore further. Theoretically I thought I understood CNNs but after doing this hands-on lab it helped take theory into practice and make it feel more familiar than when I started this lab, I was very lost and confused but after doing more research and explaining it to my son in more simple ways. It helps me truly understand what the dataset was providing us with its experience.

CITATIONS:

1. stackoverflow.com/questions/57751417/what-is-meant-by-sequential-model-in-keras
2. keras.io/guides/sequential_model/
3. keras.io/getting_started/faq/
4. www.bmc.com/blogs/keras-neural-network-classification/