Nicholas Martz
COGS 181 Deep Learning Final

Convolutional Neural Networks

Abstract

Convolutional Neural Networks (“CNNs”), similar to our brain, depend on feature recognition to classify things in certain ways. Advances with the structure of the CNNs have reduced losses to more accurately predict classification-tasks. This allows computer-vision tasks to be feasible and recent innovations have prospered because of this. In this project, I will introduce and compare different types of CNNs and tune different hyperparameters such as the number of layers, batch and epoch sizes, and different activation, pooling, and optimization functions. I will be performing this on part of the Intel Image Classification dataset (https://www.kaggle.com/puneet6060/intel-image-classification) that contains the images for the forest, mountain, and sea environment categories. When comparing my custom CNN to AlexNet, AlexNet was able to outperform my model and obtain a higher test accuracy. Changes to create more deeper and accurate CNNs will allow society to give more confidence in machine learning.

Introduction

Articles are published daily on new deep learning innovations about deep learning which leads to an increase in its popularity. My project is focused on the Convolutional Neural Network model (“CNNs”). CNNs consist of a multi-layer process and are most commonly used among image classification tasks where an input image is given. The model learns and trains itself on the features of an image to be able to classify it correctly. The model’s architecture includes Convolutional Layers (Convolution and Pooling operations) which are then flattened out and fed into the traditional Artificial Neural Network (ANN). This process allows the model to learn two necessary aspects such as feature detection and scale invariance when mapping an image input to an output variable.

Methods

2-Hidden-Layer CNN vs AlexNet

I wanted to see how my model would compare against the AlexNet CNN on the given dataset. I thought that the AlexNet CNN would outperform the 2-Hidden-Layer CNN and wondered if the test accuracy would result in a significant difference.

The 2-Hidden-Layer CNN that I used in this project consisted of two Convolutions. First, it applied 64 3x3 filters to the input image, then it applied 32 3x3 filters to the resulting feature map. Both Convolutions also utilized ReLU activation functions and a 2x2 Max Pooling window in the process. After flattening the output into a single vector, I applied it to a 128-unit fully connecting layer before the final output is produced.

Next, I performed the AlexNet CNN on the dataset. Instead of only two Convolutions, AlexNet consists of eight layers (five Convolutions followed by three fully connected layers) which has a higher complexity than the 2-Hidden-Layer model. It uses a larger number of filters, filter sizes, and strides. Also, AlexNet implements batch normalization and dropout in the model. Batch normalization is used to scale features to speed up learning and to allow each layer to learn more independently. The Dropout approach is used for regularization that “shuts-down” parts of the neural net to prevent overfitting.

1-Hidden-Layer vs 2-Hidden-Layer

The Convolutional operation takes in an input image and a feature detector filter to create a feature map which contains values representing how prominent the feature is in certain areas. The feature map has a reduced size, compared to the input image, which allows the CNN to process it easier. Usually, people combine the Convolutional and Pooling operations as the “Convolutional Layer”.

In this part of my project, I will explain the procedure of implementing two different Convolutional operations.

For the 1-Hidden-Layer CNN, I started by inputting a 150x150 size image into a Convolutional layer with 32 3x3 filters and a ReLU activation function. Next, I performed 2x2 Max Pooling on

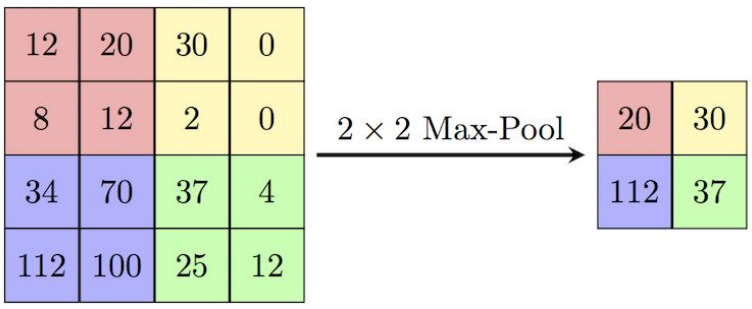
the feature map and flattened it. After, I fed the single vector into a hidden layer with 128 units and a ReLU activation function. Lastly, I output this into three units (one for each class) using a softmax function.

Due to model capacity issues, I added another experiment which was to create a 1-Hidden-Layer CNN that follows the same structure above but with a smaller input image (64x64). The smaller input image would allow the model to learn the image classification task better. I decided to do this with the 2-Hidden-Layer CNN as well for a less-bias analysis.

For the 2-Hidden-Layer CNN, I started by inputting a 64x64 size image into a Convolutional layer that consisted of 64 filters which were a size of 3x3 using ReLU as the Activation function. Next, I fed the feature map into a Max Pooling layer with a pool size of 2x2. After the first hidden layer, I fed the output through another Convolutional layer with 32 3x3 filters, another ReLU Activation function, and Max Pooled it again with a size of 2x2. Once feeding it through the second Convolutional layer, I flattened the cells of the feature map into a single vector. After many trial and error attempts, I chose to feed the single vector to a hidden layer that consists of 128 units with a ReLU Activation function. Lastly, I outputted this into three units (one for each class) using a softmax function.

Max Pooling vs Average Pooling

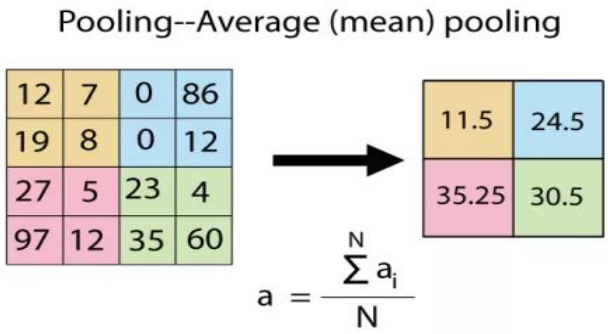
Max pooling is a type of pooling where one extract the largest number in a filter. For example, refer to the representation below.



Given a 4x4 feature map, we perform Max Pooling in a 2x2 filter which results in a 2x2 Max Pooled Feature Map. Notice that 20 is the biggest in the first 2x2 area, 30 is the biggest in its respective area, and so forth. Basically the Max Pooling method is compressing the feature map by taking the maximum value in the filter and detecting how prominent a certain feature is

in that area. Also, notice how all of the other values in the filter are dropped and aren't considered in the next steps. This makes max pooling an advocate for extreme feature detection such as edges.

Average pooling is a type of pooling where one extracts the average number in a filter. For example, refer to the representation below.



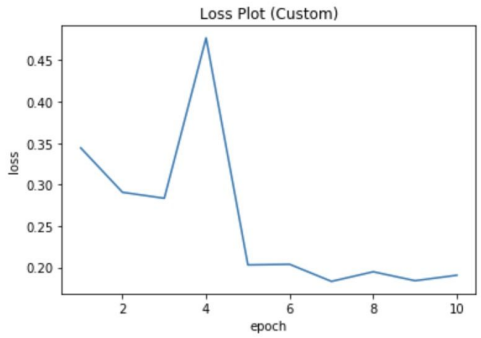
Given a 4x4 feature map, we perform Average Pooling in a 2x2 filter which results in a 2x2 Average Pooled Feature Map which takes the averages of each region or filter respectively. Different from Max Pooling, Average Pooling takes into account all values in the filter which means all values are used to construct the output and extracts feature more "smoothly".

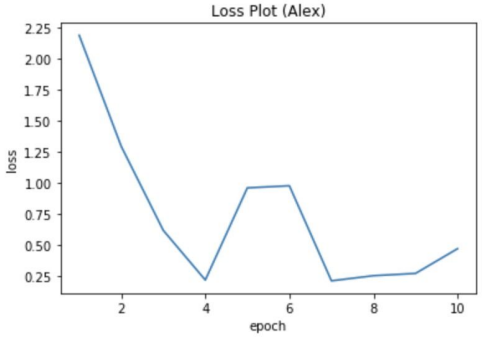
In this part of my project, I compared two of the above Pooling functions on the same dataset given the same Convolutional layers to detect how it would effect my dataset and which Pooling function would produce better test accuracy.

Experiments

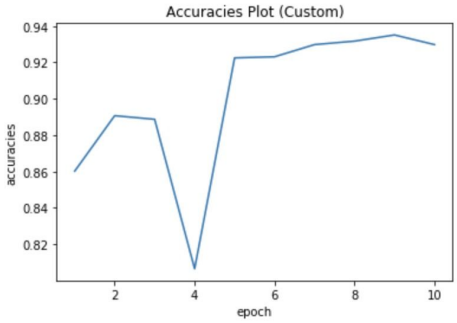
2-Hidden-Layer CNN vs AlexNet

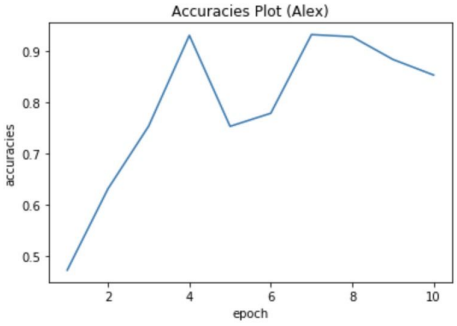
After running both models, I compared the losses and accuracies to determine which model would be a better fit to the Intel Image classification dataset. Going against my initial hypothesis, that AlexNet would outperform the 2-Hidden-Layer CNN model, the results proved that AlexNet might be too complex and the 2-Hidden-Layer CNN model might work better with the dataset.





Taking a look at the graph above, both models show to have a loss trending downwards with the 2-Hidden-Layer CNN model having a slightly lower loss. Also, notice that the AlexNet model starts at a very high loss and learns fairly quickly compared to the custom CNN model which starts at a much lower loss but both resulting in a similar result. It is important to note that both graphs show a spike which illustrates a peak in loss value which may be as a result of a “bad” batch that is inconsistent with the other batches and made the model perform at an unlucky state.





Next, let’s take a look at the accuracies between the models which shows an upwards trend. Again, we can see the 2-Hidden-Layer model outperforming the AlexNet and the unlucky batches which explains the valleys in the graphs. Similar to the losses, both plots end up with similar results which performs with a fairly high accuracy.

Overall, although the custom CNN models outperform the AlexNet, we cannot conclude that it is better for this given dataset. To draw a more confident conclusion, we would have to perform an experiment with more epochs which would give us a more steady incline and more accurate

interpretation of which model is truly better for the dataset. However, we can conclude the the *2-Hidden-Layer* CNN model outperforms AlexNet given this specific 10 epoch analysis.

1-Hidden-Layer vs 2-Hidden-Layer

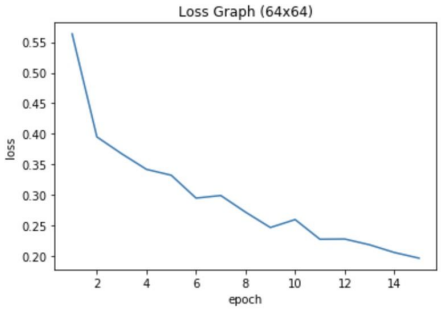
The complexity of a CNN model can heavily affect our testing accuracy. Choosing how many layers usually depends on how large the given dataset is. If the given dataset is small, we would use a more simplistic model that contains less parameters which is where the 1-Hidden-Layer CNN would be more beneficial. For example, if the model is classifying if an object is a watermelon or banana, it is common for the model to compare parameters such as color and shape. If a 2-Hidden-Layer CNN was implemented on this dataset, the model would start to compare irrelevant features and overfit the data. Since the 2-Hidden-Layer CNN is more complex, it would add more parameters like smell and feel and without any sensory inputs, these irrelevant parameters would negatively affect our accuracy.

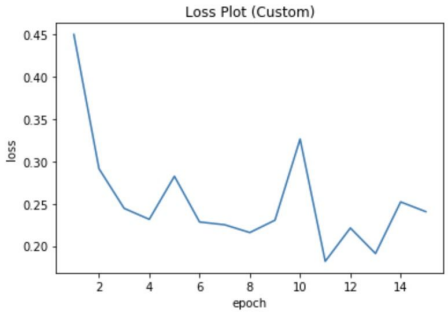
A 2-Hidden-Layer CNN model would be more beneficial with a large dataset because of its complexity which contains more parameters. For instance, if the model is classifying something more similar such as whether or not an object is a banana or plantain, the model would have to factor in different parameters like sugar level, starch content, color, size, etc. If a 1-Hidden-Layer CNN was used on this dataset, it would underfit due to its properties and negatively affect the accuracy of the model.

Overall, a CNN model's complexity should be determined based on the properties of the given dataset. To obtain the optimal testing accuracy, the number of Convolutional layers in a CNN cannot be too small nor too large.

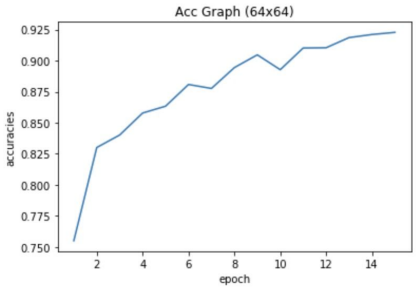
Given the Intel Image dataset, I wanted to compare 1-Hidden-Layer CNN model to a 2-Hidden-Layer CNN model and which might yield a better test accuracy. Below are graphs that illustrate the loss and accuracies of each model.

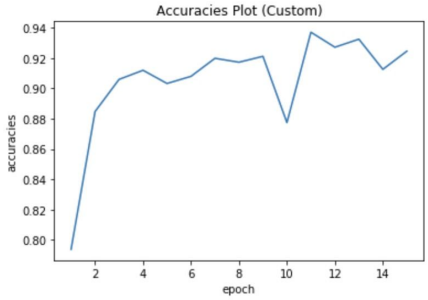
After running the 1-Hidden-Layer Convolution on the 150x150 size input images, I got a constant result for the loss and accuracy. It seems that the model wasn't learning due to model capacity issues with large-size images. Therefore, I decided to run the 1-Hidden-Layer CNN on an image set with the size of 64x64.





Taking a look at the graphs above, it is noticeable that both have a downward trending loss. Both models seem to have resulted in very similar loss values when it comes to a 64x64 input image one could not come to a conclusion of which model would be a better implementation to the dataset. However, with the initial comparison on 150x150 images and the 1-Hidden-Layer having a constant loss, it was easy to conclude that a 2-Hidden-Layer would be optimal.





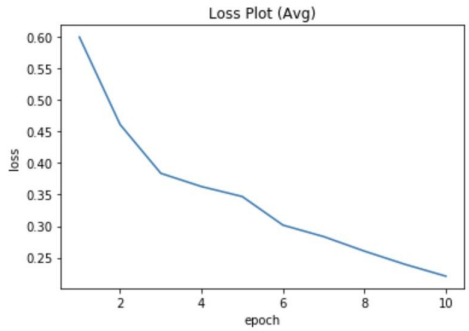
The above graphs both show the accuracies trending upwards and obtaining high values which means our models are learning well. They both have very similar resulting accuracies with the 2-Hidden-Layer CNN having a slight edge in performance. However, this could just be due to performing well on a certain “lucky” batch. Overall, since the accuracies are so close together and we are only doing 15 epochs, I cannot come to a conclusion on which may be a better model. But, for this specific implementation with 15 epochs, the 2-Hidden-Layer CNN worked slightly better.

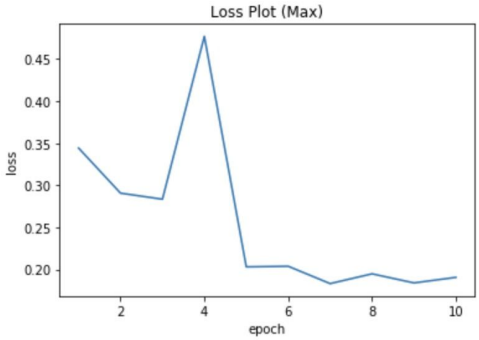
Max Pooling vs Average Pooling

Pooling is usually performed after an activation function which gives our CNN numerous amounts of benefits. One benefit of Pooling is that it reduces the size of the feature map which makes it faster to process. The feature map gets smaller when we reduce the parameters that

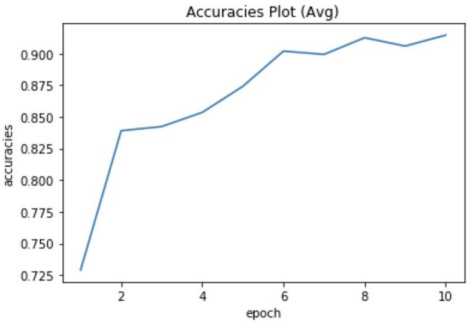
go into the final layers, which prevents overfitting by removing non-relevant (noise) information and extracting features, just how humans do when classifying objects. Another benefit is that it gives our CNN a property call spatial invariance. This means that the CNN doesn't care about where certain features are located within an input image, it will be able to detect it no matter what, if the feature is present. The feature could be distorted (titled, closer, etc..) which gives our CNN flexibility by being able to classify images regardless of orientation.

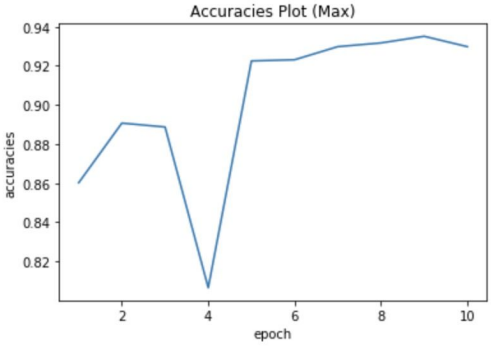
When comparing between the two different pooling functions we notice some notable differences. First, let's take a look at the loss produced as the model iterated through each epoch.





Both loss graphs show a general trend of the loss decreasing. Although this might be harder to notice in the Max Pooling Loss graph, if time permitted and more epochs were added, the general trend would end up looking more smoothly like the graph on the left. In the case of a spike, the model might have just been inconsistent with the specific batch being tested on and produced a noisy result. Also notice that Max Pooling starts and ends with a lower loss than Average Pooling.





Next, looking at the accuracy plots, notice that both graphs illustrate a trend in increasing accuracies. Although it is harder to see the trend in the Max Pooling Accuracy graph, the tested batch in the fourth epoch might have been an unlucky one which was inconsistent with the trend and produced a noisy accuracy output. Also, notice that Max Pooling has slightly better accuracies than Average Pooling.

Overall, it seems that the properties of Max Pooling, extreme edge detection, perform slightly better than the smoothing of Average Pooling. However, to make a more confident conclusion one would have to test many more epochs and see if the trend continues.

Conclusion

I was able to draw some important inferences through my experiments. In order to have access to a GPU, I had to connect to campus wi-fi which wasn't always accessible for me. If it was more accessible, my tests would have been more thorough which would allow me to produce a more confident conclusion. I was able to conclude that my dataset didn't need a complex model such as AlexNet to obtain a high accuracy. The 1-Hidden-Layer CNN and 2-Hidden-Layer CNN both obtained over 90% which would be sufficient for this classifying this dataset. If I was to run my experiment again, I would run the all the CNNs with more epochs so noise wouldn't affect the results as much and create a more steady trend. This would allow me to analyze the results more confidently and draw a stronger conclusion due to less bias in each batch/epoch.