COE 379L: Project 4

Ashton Cole

May 3, 2024

1 Introduction

Neural networks are a powerful tool to perform various classifcation and regression tasks. One interesting feature of neural networks, compared to other machine learning models, like a linear regression or K-nearest neighbor classification, is that they grant the user significant leeway in their design. In a dense Artificial Neural Network (ANN), each layer can have an arbitrary number of perceptrons with a particular activation function, and an arbitrary number of layers can be added. The possibilities are greatly increased when other types of layers, like convolutional, pooling, and dropout ones, are brought into consideration.

There are not any immediately obvious rules of what kind of network structure should be used. How many layers should be used for an image classification problem? How many perceptrons should be in each layer? The large *hyperparameter* space makes it intimidating to fine-tune a neural network model. This raises an interesting problem to explore.

The goal of this project is to investigate how the architecture of a neural network impacts its accuracy. Hyperpararameter sweeps of dense ANN architectures will be performed to glean any apparent patterns. Training is done on three unique image classifcation problems, to see how dependent the results are on the dataset. The results are used to identify both the optimal architectures and general relationships between hyperparameters and accuracy.

2 Methodology

2.1 Project Structure

This project is organized into several Jupyter notebook files, as well as a data folder. Additional output files are generated when the notebooks are run.

- `ann-beans.ipynb`: A Jupyter Notebook file which conducts a hyperparameter sweep for the *beans* dataset.
- `ann-hurricane.ipynb`: A Jupyter Notebook file which conducts a hyperparameter sweep for the *hurricane damage* dataset.
- `ann-mnist.ipynb`: A Jupyter Notebook file which conducts a hyperparameter sweep for the *MNIST* dataset.
- `preprocessing.ipynb`: A Jupyter Notebook file which performs the train-test split on the hurricane damage dataset's directory.
- `data/`: A folder to hold the hurricane damage dataset.
  - `data_all_modified.zip`: An archive of the hurricane damage dataset.

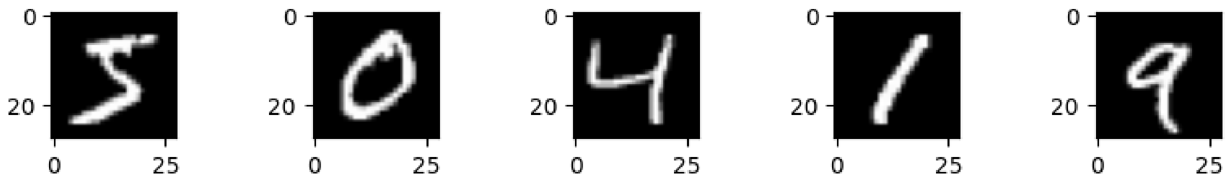

Figure 1: Example pictures from the *MNIST* dataset.

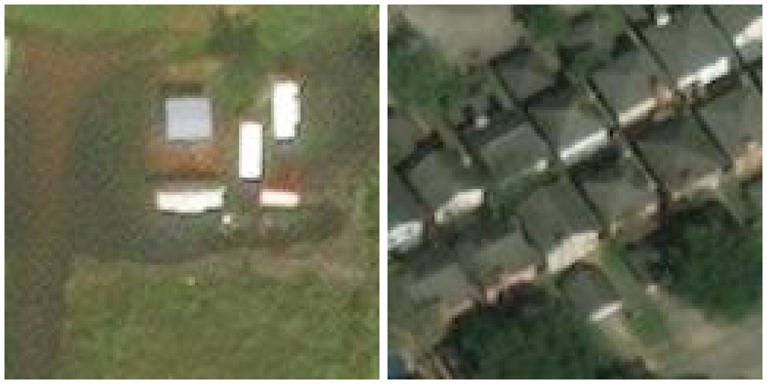

Figure 2: Example pictures from the *hurricane damage* dataset, of properties with flooding (damage) and without flooding (no damage).

2.2 Datasets

Three datasets are used for this study, to evaluate whether or not certain results are tied to a particular dataset: one of handwritten numbers, one of hurricane damage, and another of beanstalk leaves.

First, the *MNIST* [3] dataset from class is revisited. This Modified National Institute of Standards and Technology database set contains 70,000 28×28-pixel black-and-white images of handwritten digits. These are labeled 0-9, as seen in Figure 1

Second, the *hurricane damage* [1] dataset from Project 3 is revisited. This set contains 17,057 128×128-pixel color satellite images of properties. These are labeled as either having "damage" or "no damage" by flooding associated wtih a hurricane, as seen in Figure 2.

In addition, a new *beans* [2] dataset from the TensorFlow Datasets module is considered. This set contains 1,295 500×500-pixel color images of leaves of beanstalks. Leaves are either categorized as healthy or having angular leaf spot or rust diseases, as seen in Figure 3.

2.3 ANN Hyperparameter Sweep

Neural networks are constructed using TensorFlow, specifically, the `Sequential` model type available through the Keras submodule is used.

Model construction and hyperparameter sweeping are automated by the Keras Tuner module. Two dense hidden layers are used, each having a varying number of 64, 128, 192, or 256 perceptrons. To simplify the sweep, only the number of perceptrons in each layer is varied. Activation functions are held constant: the hidden layers use the "ReLU" function, and the output layer uses the "softmax" function. This structure is summarized in Table 1. For training, the optimizer is "adam" and loss is calculated using "sparse categorical crossentropy." Finally, the models are trained for 10 epochs, in batches of 32 images.

During the sweep, trained model checkpoints and their metadata are saved to a `tuner/` directory by Keras Tuner. In addition, the training and validation losses and accuracies are logged after each epoch to a CSV file by means of a callback function. Accuracy is used as a simple metric to evaluate the quality of models. Training, validation, and testing data accuracy are specifically compared against each other, to ensure that overfitting is not an issue.

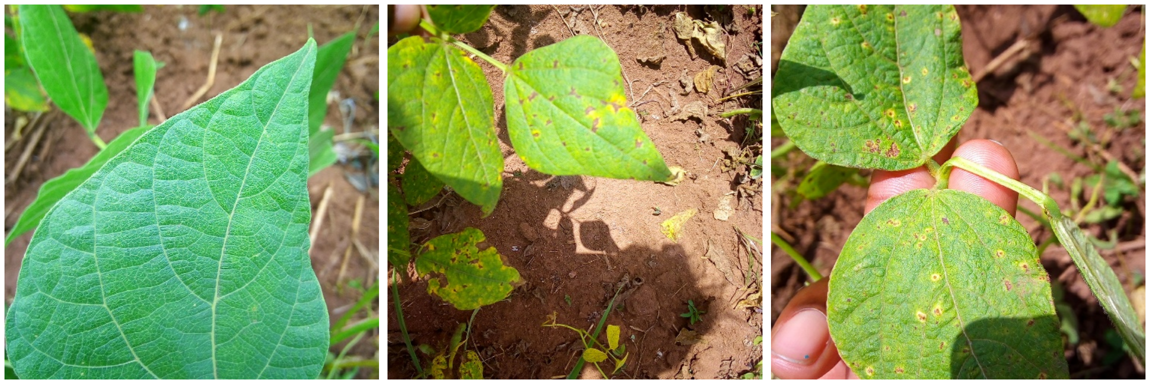

Figure 3: Example pictures from the *beans* dataset, of beanstalk leaves that are healthy, have angular spots, or have rust.

Table 1: Outline of the sequential neural network structure used for this study.

Layer	Type	Input Shape	Number of Perceptrons	Activation Function
0	Flatten	$s \times s$	N/A	N/A
1	Dense	s^2	64, 128, 192, 256	ReLU
2	Dense	Depends	64, 128, 192, 256	ReLU
3	Dense	Depends	Depends	Softmax

The number of layers used and hyperparameter combinations tested are admittedly limited, because additional layers increase complexity and storage requirements exponentially. Even when only considering two hidden layers, trained on one image set, for 16 different combinations of perceptron counts, over a third of the testing machine's disk storage is consumed. Additional layers also increase the chances of encountering the vanishing gradient problem. This arises from back propagation, i.e. a "long" derivative chain rule, used to minimize the cost function of the model.

2.4 Setup and Execution

This study can be replicated by running the Jupyter Notebook files on a Jupyter Notebook server with a Python 3 kernel. The TensorFlow, TensorFlow Datasets, and Keras Tuner packages should be installed into the environment. First, the `data_all_modified.zip` file should be extracted within the `data/` folder. The `preprocessing.ipynb` file should then be run to conduct a train-test split for the *hurricane damage* dataset. Then, the other notebook files may be run to conduct the parameter sweeps. The model checkpoints and their data will be output to a generated `tuner/` directory. CSV logs will be output to the source directory.

3 Results

In this section, first, tables are provided showing the training, validation, and testing accuracies for each model at the tenth epoch. Then, graphs show how the training and validation accuracies vary for each model across training epochs.

put description here

matrices of accuracy, validation accuracy, per dataset

takes about an hour to execute each sweep

put in observations

Table 2: Impact of number of hidden-layer perceptrons on model accuracy, for training, validation, and testing data, trained for 10 epochs on the *MNIST* dataset.

Second Layer	First Layer Perceptrons											
Perceptrons		64			128			192			256	
64	0.991	0.975	0.973	0.993	0.976	0.977	0.995	0.977	0.978	0.995	0.976	0.979
128	0.992	0.975	0.975	0.993	0.973	0.976	0.994	0.977	0.978	0.994	0.974	0.979
192	0.993	0.969	0.974	0.993	0.972	0.979	0.994	0.977	0.977	0.994	0.977	0.979
256	0.993	0.971	0.973	0.994	0.976	0.975	0.994	0.975	0.977	0.994	0.978	0.979

Table 3: Impact of number of hidden-layer perceptrons on model accuracy, for training, validation, and testing data, trained for 10 epochs on the *hurricane damage* dataset.

Second Layer	First Layer Perceptrons											
Perceptrons		64			128			192			256	
64	0.771	0.763	0.768	0.680	0.668	0.740	0.752	0.773	0.765	0.751	0.774	0.770
128	0.777	0.784	0.770	0.761	0.768	0.770	0.664	0.668	0.742	0.664	0.668	0.745
192	0.664	0.668	0.725	0.662	0.753	0.758	0.664	0.668	0.751	0.723	0.719	0.749
256	0.664	0.668	0.664	0.664	0.668	0.733	0.664	0.668	0.742	0.664	0.668	0.709

Table 4: Impact of number of hidden-layer perceptrons on model accuracy, for training, validation, and testing data, trained for 10 epochs on the *beans* dataset.

Second Layer	First Layer Perceptrons											
Perceptrons		64			128			192			256	
64	0.637	0.466	0.500	0.663	0.376	0.672	0.708	0.444	0.570	0.652	0.376	0.578
128	0.552	0.617	0.586	0.645	0.353	0.609	0.708	0.571	0.695	0.603	0.556	0.711
192	0.683	0.406	0.711	0.605	0.406	0.555	0.723	0.368	0.641	0.669	0.752	0.695
256	0.661	0.541	0.648	0.705	0.519	0.742	0.691	0.586	0.547	0.746	0.504	0.516

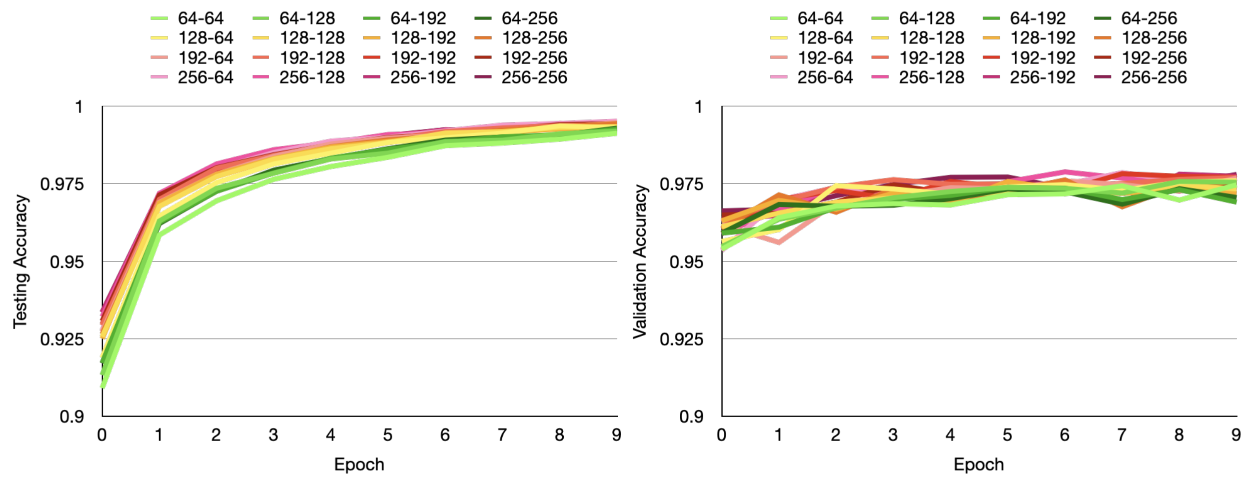

Figure 4: Training and validation accuracies for the *MNIST* dataset across training epochs.

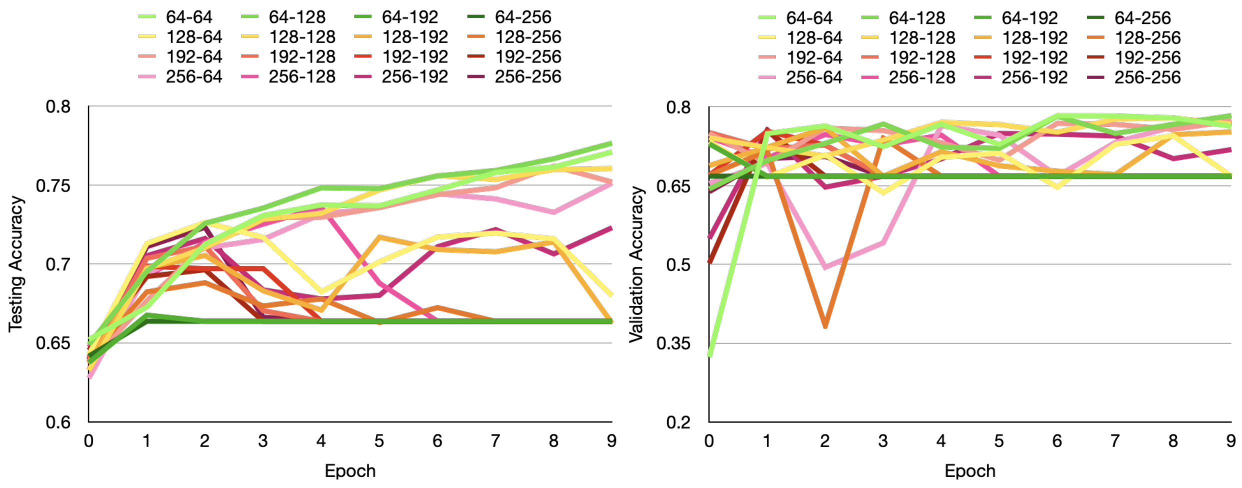

Figure 5: Training and validation accuracies for the *hurricane damage* dataset across training epochs.

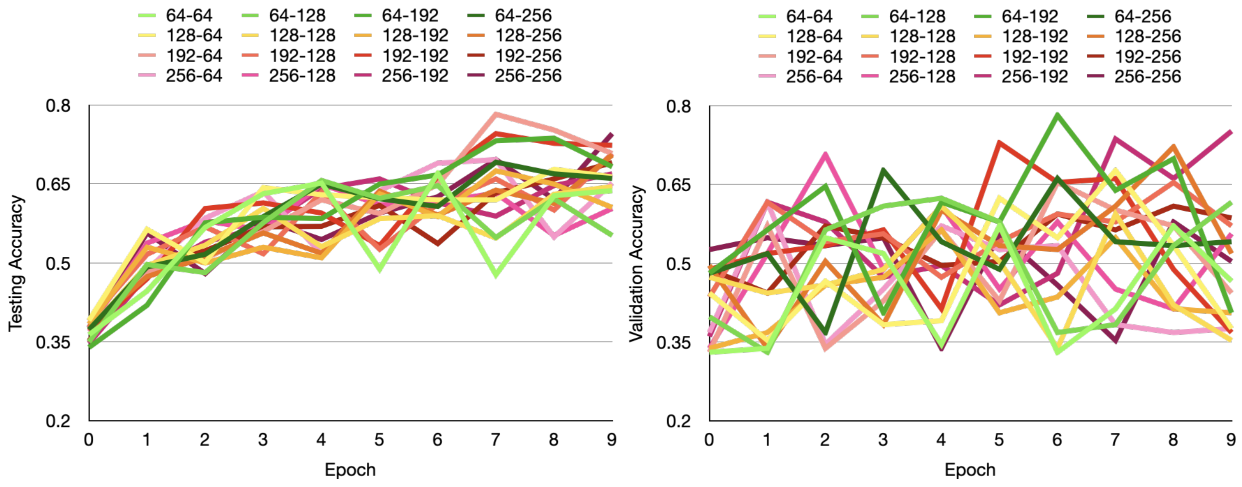

Figure 6: Training and validation accuracies for the *beans* dataset across training epochs.

4 Conclusions

overfitting

vanishing gradient

bottlenecking of first layer

image problem complexity

try higher resolution, batches

more layers

cnns

References

[1] Quoc Dung Cao and Youngjun Choe. Detecting damaged buildings on post-hurricane satellite imagery based on customized convolutional neural networks, 2018.

[2] Makerere AI Lab. Bean disease dataset, January 2020.

[3] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.