# Final Project

Alexis Hampton Department of Computer Science
CUNY Queens College
Flushing, NY 15213
alexis.hampton38@qmail.cuny.edu

December 11, 2024

**Abstract**

The model discussed below combines the fast and very powerful convolutional neural network with the simple and quick linear regression model.

## 1 Introduction

When I first saw the final project I originally wanted to code a normal neural network, but the only way to get the neural network to work with images was to make it a convolutional neural network. Converting the images to weights did not work very well for me, so I eventually decided on using TensorFlow's CNN implementation to handle the image convolution and a more complicated neural network than I could make myself.

When thinking about how to make it different, I really wanted to merge two different machine learning models together, since that seemed doable. None really seemed to fit in a memorable way. Deep neural networks were just neural networks with more than 3 layers, which seemed like cheating. And then merging a ResNet and CNN didn't work either because while they seemed pretty similar they had extremely different architecture.

So, I decided to look away from deep learning machine learning models and to the more simpler ones we learned in the beginning of the semester. In the past couple of weeks while trying to build my neural network from scratch I had wrote a liner regression model, because I had mistakenly believed that neural networks were made of lots of logistic regression models and a final linear regression model, all interweaving together.

I thought it might be nice to learn better weights by passing them into a different model and seeing what happens. I had wanted to use a random forest model, but the libraries available don't work for Intel-based macs, so I couldn't use them. Instead I chose to rely on the linear regression model I had built myself. Partly also because I still wanted more of my code to be in my final project rather than relying so much on libraries to do all the work for me.

In the section following I will discuss papers I read, one of which inspired my CNN design. Then in the methodology section, I will explain the architecture of the model and all the data issues I had with it. Following will be the experiment which tests if more convolution layers lead to better accuracy. And finally, the dismal results of the model will be discussed.

## 2 Literature Review

### 2.1 ImageNet Classification with Deep Convolutional Neural Networks

The first paper I read was "ImageNet Classification with Deep Convolutional Neural Networks". With a very deep CNN, they classified 1.2 million images for the ImageNet LSVRC-2010 contest. In their paper, they discussed the CNN that they used in the contest. They detail the architecture, how they combated overfitting, and their amazing results that placed them at the top of the contest.

When I first skimmed this paper, I was inspired by a misread of them using pooling after a neural network layer. But after attempting and failing to do something that is quite impossible, I was inspired by the fact that they had multiple convolution layers before a pooling layer. I also found it quite strange that they trained their CNN on 2 GPUs which ran in parallel but it still took 5 to 6 days. It is reasonable given the large amount of data but as I don't have as much computing power, which will become very apparent later on in this paper, I can only imagine how much work is being done at a time.

In the paper, they also mentioned using ReLU as if everyone else wasn't, which is likely because it's an old paper. I thought it was fascinating to read a paper that provided groundbreaking innovation.

Their results were very impressive, with the model predicting the images correctly, and then predicting the next probable classes very closely to the actual class. Their error rates were very impressive as well, as they constantly beat their peer's error rates.

That and their architecture inspired me to make my own. In my CNN, I took mostly the same layer layouts, with mostly the same filter and neuron sizes for convolution. In their first layer they had a 11 x 11 x 3 convolution layer, but I made mine 11 x 11 x 32 so it would run faster. I also changed the number of neurons for the fully connected layers which will be explained in detail in the Methodology section.

### 2.2 Recent Advances in Convolutional Neural Networks

The second paper I read was "Recent Advances in Convolutional Neural Networks". They first discussed the history, architecture, and inner workings of CNNs, before compiling a list of recent advancements others have made with it.

The history confirmed a lot of the knowledge I already knew about CNNs but failed to apply in the past 3 weeks of trying to write my own CNN from scratch.

But I found the list of advancements in CNN research to be very compelling.

I was very impressed by the idea to change the loss and activation functions. A lot of the ideas they put forth seemed very math heavy or required a lot of computational power. And some even seemed built for different kinds of models like GANs. I tried to implement the hinge loss function from this paper into my CNN model, but it requires a strict [-1,1] target which I didn't train my CNN on, since I kept it with multi-class classification.

Pooling functions were also something changeable brought forth by the paper, but I did not want to change it because of my previous inspiration in the other paper.

But when it came to activation functions, it seemed that one of the bigger concerns was the speed. ReLU, ELU and the others mentioned were quite fast while legacy ones like sigmoid and tanh were very slow. So I stuck with ReLU since it's speed was confirmed in the paper.

The paper also touches on different regularization methods, one of which is data augmentation. They mentioned mirroring images, shifting them, rotating them, slicing them up, and many more functions. And this coupled with the advice from class, I decided to augment my training data before running it.

The paper then mentions all the ways CNNs are used to classify images, from object detection to posing, to recognizing text. It explores the way different researchers have approached these problems.

## 3 Methodology

The model trains a normal CNN with 2-4 convolutional layers and 2 fully connected layers that feed into the 6-fold classification layer. The second to last layer's weights are taken and then fed into a linear regression model as features. The linear regression's final accuracy is used as the accuracy for the entire model.

As expressed before, I was inspired by "ImageNet Classification with Deep Convolutional Neural Networks" to play with the amount and sizes of the convolutional layers. I ended up not using their model, but used parts of it, and the numbers for the sizes.

### 3.1 Data

Before the CNN begins, I collect all the data from the training and validation sets from the dataset. For each image I had to change the target label to be 0 to 4 to represent the bird, boat, bus, cat, and cow classes. The label would also be replaced by later files, so if an image was classified as cat by cat-train.txt, then bus-train.txt could change it to be classified as bus if they both were in the same image.

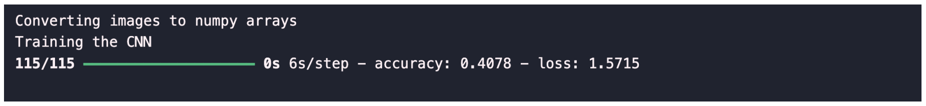
The number 5 was reserved for images that did not have any of the aforementioned classes in it. And unfortunately, most of the dataset contained those.

I also resized all images to be 400 x 400, so they were all the same size as per TensorFlow recommendations. And then all images were normalized to be between 0 and 1 also per TensorFlow recommendations.

#### 3.1.1 Data issues

While training the data, my kernel crashed a lot. Eventually, I figured out it was due to the large dataset. So, instead of using all 14,000 images, I just used around 4,000 (the 2008 images).

When augmenting the data I wanted to mirror, flip, and rotate 40 percent of the data and keep those 3 versions as well as the original, but it became too much data. Now, I only augment 5 percent of the training data and keep 1 augmented version chosen pseudo-randomly as well as its original version. It's not a lot of data, but the kernel doesn't crash anymore.



```
Converting images to numpy arrays
Training the CNN
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - accuracy: 0.4078 - loss: 1.5715
```

Figure 1: A simpler CNN model trained using data augmentation only on images that were of class: bus, boat, bird, cat, and cow with training accuracy of 40 percent. Validation accuracy could not be calculated because the kernel crashed.

### 3.2 CNN model

The CNN has 2-4 convolution layers with pooling layers following, inspired by the "ImageNet Classification with Deep Convolutional Neural Networks" paper. I originally wanted to try it with 1 convolution layer, but the images were so big it would have taken 6 hours to train one CNN, so I started at 2 instead. I kept the activation functions as ReLU for all the layers.

The first convolution layer is of size 11 x 11 x 32, followed by a max pooling function with pool size 2 and stride 4. The second convolution layer is of size 5 x 5 x 48, followed by the same max pooling function with pool size 2 and stride 4. The third convolution layer is size 3 x 3 x 192, followed by a max pool layer of pool size 2 and stride 4. The fourth convolution layer is the same as the third.

The first fully connected layer has a variable neuron size. This is to get the correct number of weights for the linear regression model. So it's always the length of the training image data. The second fully connected layer has 64 neurons, so each training data for the linear regression model has only 64 features.

The last layer of the CNN has 6 nodes that test for classes: bird, boat, bus, cat, cow, and none. None is there because there are other unrepresented classes and it feels wrong to falsify data to make everything fit within 5 classes.

The last layer is only to see the training accuracy and isn't used outside of that.

### 3.3 CNN Training Architecture

During training, two CNN models are used. One is to get the training weights to be fed into the linear regression model. It's trained on the training image data and the training target. The second is to get weights for the validation. It uses the validation image data and the validation target. This validation data is used to check the accuracy of the linear regression model. The validation CNN model also uses the same number of neurons as the first. So if the training CNN has 3637 images (with augmentation), then it's first connected layer will have 3637 neurons. The validation data which only has 868 images will still be trained with 3637 neurons.

### 3.4 Linear Regression

The linear regression model is the same as the one in the slides from class. It is trained as a 1 verses all linear regression to make it multiclass.

The accuracy function is a little different from a regular accuracy function. Since the number of training data depends on the number of neurons from the CNN, the training data can have more or less data to work with.

As stated above, it might have 868 validation images, but the incoming validation weights from the CNN will be 3637. To get around this, only the first 868 weights will be used. Though in practice, the lowest of the two is used.

The target vector is also changed to be in the range [-1,1], with the current class being 1 and all others being -1.

The linear regression model's accuracy is used for evaluation.

## 4 Experimentation

The experiment is to check whether more CNN layers is better than less. This project only uses 2 to 4 layers with the architecture above. Everything else is the same. The only thing that changes is the number of layers.

### 4.1 Description of the dataset

As explained above, only the 2008 images data is used because of frequent kernel death. Only the bird, boat, bus, cat, and cow classes are tested. There is also a None class in case any of the images falls outside of these, which they frequently do.

Data augmentation is only used on 5 percent of the data which is rotated 90 degrees, mirrored, or flipped upside down.

All of the images are resized to 400 x 400, and all the pixels are normalized to be between 0 and 1.

The target vector only stores one class which is usually the last class file seen as explained above.

### 4.2 Description of the performance measurement

The performance is measured using the linear regression model's accuracy.

### 4.3 Validation Process

Five-fold cross validation is used to validate the model. The data is shuffled before split into 5 groups. It's then trained with one group taken as validation and the other four combined to be training data.

### 4.4 Results

The plotted results are in the figures 2, 3, and 4 at the end of the document.

I like how the accuracies tend to cluster together. It makes sense since there's likely an equal amount of images for each class after being labeled in the code. But it is quite strange that only figure 3, the accuracy for 3 convolutional layers, has more spread out accuracies. It co uld be that their clustering is due to sharing the same training and validation weights, with only the target functions impacting their accuracy.

The accuracies don't change much. For each amount of convolution layers they remain in the 0.43   0.57 range.

## 5 Discussion

The results are very bad. There isn't a single accuracy over 0 percent, but it seems expected. The linear regression model is only receiving weights as features and using that to predict a class. Originally I was going to use flattened images, but I could not get that to work, so I used the weights instead, thinking that maybe it could work out, but it didn't.

I had hoped the linear regression model might learn the weights and improve upon them, but that didn't necessarily work the way I expected it to.

The data issues explained above could also contribute to the poor results, like the small amount of data trained, or image targets being overwritten, or the large amount of data that belongs to none of the 5 classes.

## 6 Conclusion

In summary, training weights to predict class labels isn't a very good idea. But it is a novel one. If I had planned better, perhaps I could have found a way to get those flattened images and then continued training the weights in linear regression while using those images as training data. But, on the bright side, this model shows a way in which one shouldn't think about machine learning.

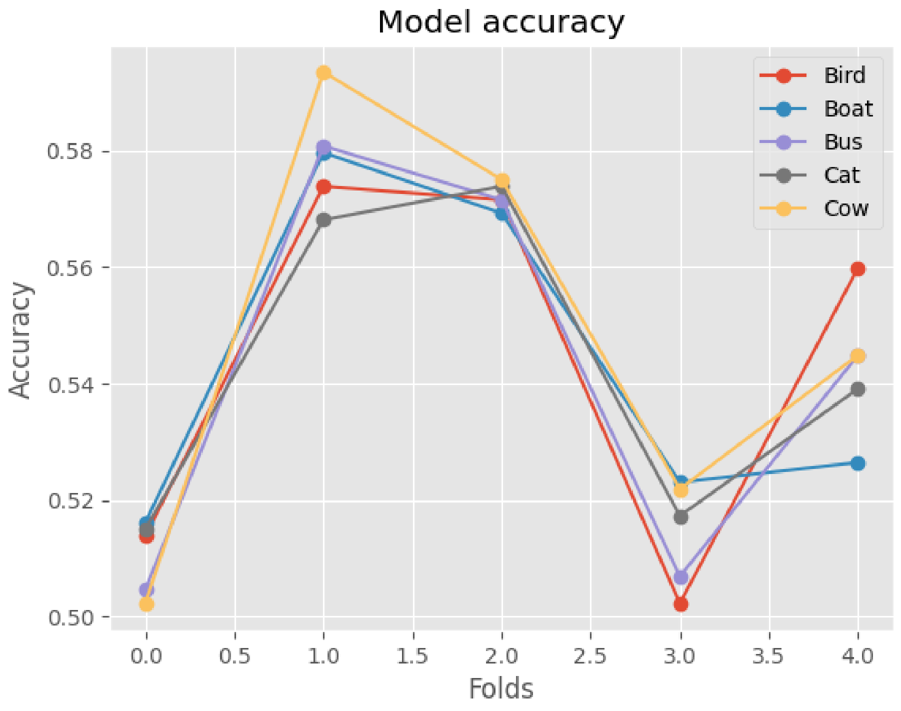


Figure 2: Model accuracy for 2 convolution layers

## References

[1]Gu, Jiuxiang, et al. “Recent Advances in Convolutional Neural Networks.” Pattern Recognition, vol. 77, May 2018, pp. 354–377, https://doi.org/10.1016/j.patcog.2017.10.013.

[2]Krizhevsky, Alex, et al. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

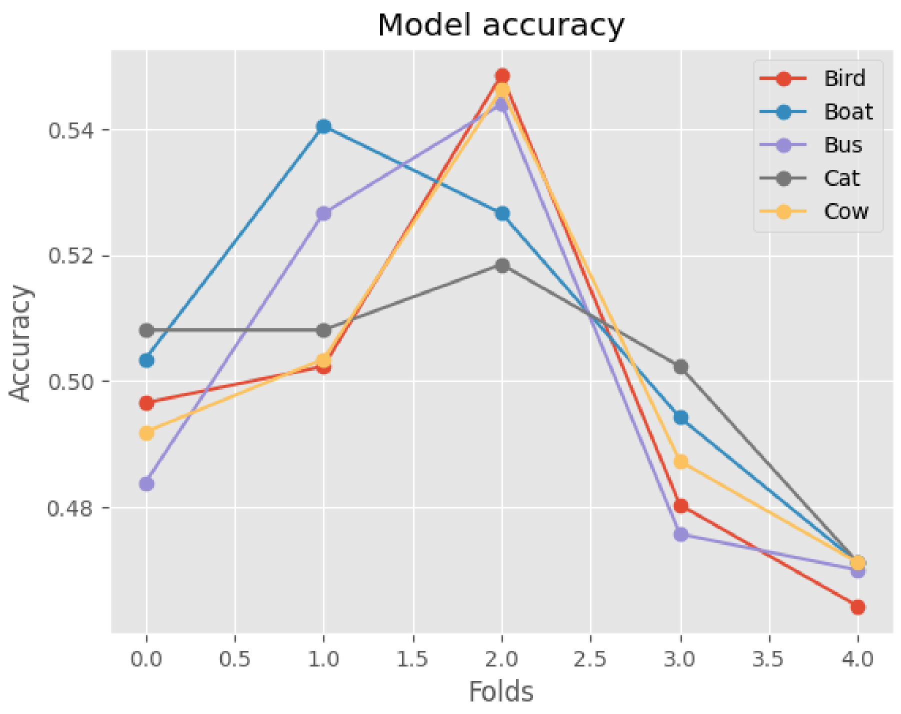


Figure 3: Model accuracy for 3 convolution layers

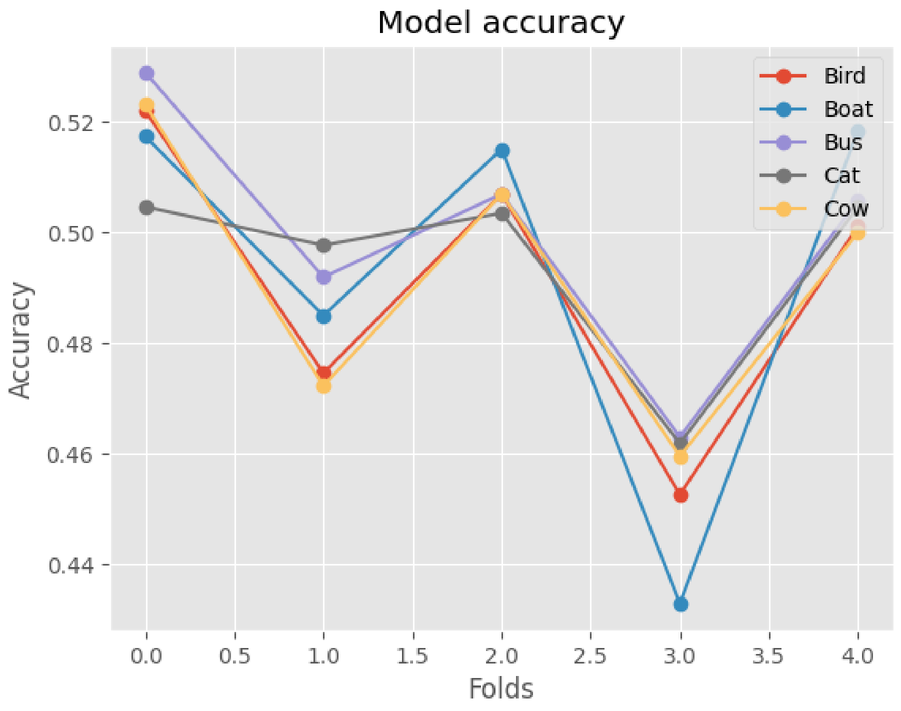


Figure 4: Model accuracy for 4 convolution layers