Leveraging Neural Networks for Poultry Disease Identification

Tom Hocquet tphocquet@ucsd.edu
Zaki Ahmed zahmed@ucsd.edu

Abstract

This study applies Convolutional Neural Networks (CNNs) to a large dataset of chicken excrement images for early detection of avian diseases, a critical issue in poultry agriculture. The dataset, from the Kaggle repository "Poultry Pathology Visual Dataset," includes images from Tanzania, classified into four categories: Healthy, Coccidiosis, NewCastle Disease, and Salmonella.

We developed a sophisticated neural network, LNN-PDI, and used augmentation techniques to handle real-world variability. Optimization methods, including a transition to ADAMW and experimentation with different pooling and activation functions, were employed for model refinement. Despite computational constraints, our model achieved an accuracy of 89%, demonstrating the potential of CNNs in disease detection. Our findings highlight the importance of balanced datasets, strategic optimization, and adequate computational resources in developing effective tools for early avian disease detection.

1. Introduction

With an increased demand on food production in the agricultural industry, avian diseases pose a larger threat than ever to the global food supply and in particular individual chicken farms. Early detection of diseases is of paramount importance and early detection using minimally invasive and involving methods would provide the greatest benefit to both humans, poultry and the industry. Leveraging the predictive potential of Convolutional Neural Networks (CNN's) on a large dataset of chicken excrement images to identify diseases was our goal.

2. Data

2.1. Overview

The dataset was sourced from the Kaggle repository *Poultry Pathology Visual Dataset*, a dataset consisting of poultry fecal images taken from other repositories of images taken by mobile phones. The fecal images were taken in Tanzania between September 2020 and February 2021 [1]. Each of these images were classified into 4 distinct classes- Coccidiosis, Healthy, New Castle Disease, and Salmonella. The dataset was compiled with the purpose of not only improving rapid detection of common poultry diseases but also to create a dataset designed for machine learning research making it a perfect fit for our use case.

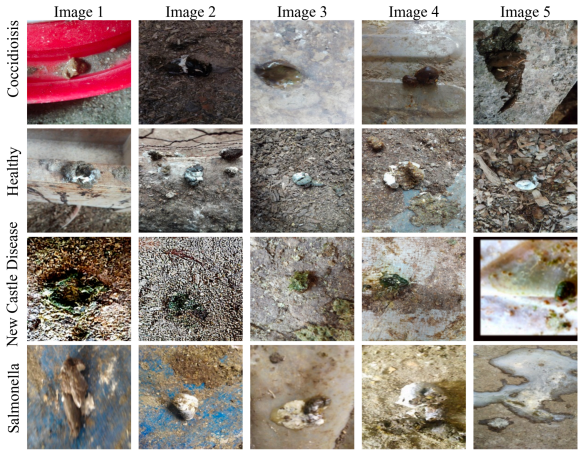

Figure 1. Five Example Images of each class label (top to bottom) Coccidiosis, Healthy, New Castle Disease and Salmonella. Unaugmented Images.

All images in the dataset were resized to a size of 224x224 pixels- allowing for easy integration with existing networks as the size is a long standing precedent with neural network architectures. The data is split into three static directories- train, validation and test. The training images consist of 100,000 images per

class and the validation set 10,000 images per class. The test set consists of 70,500 images with an average of 17,625 images per class as the exact image count is variable in the test set.

2.2. Augmentation

The dataset images were augmented for the purpose of assisting generalizability and robustness on any model trained. Train and Validation have each image augmented using a variety of augmentation methods while the test set contains unique images with some being augmented. These augmentation methods included modifications such as blur, image rotation, color inversion, grayscale and more. The augmentation process not only simulates the inherent variability in real life data (especially considering the data comes from mobile phone photos) but also artificially expands the dataset providing a richer training set for our machine learning models.

3. Model

3.1. Network Architectures

For the design of our convolution neural network, we follow the architecture illustrated below.

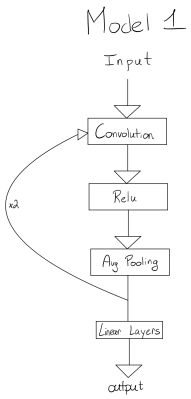

Figure 2.1 Drawing of the architecture of Model One

3.1.1. Model One. The first model we used was a model used in homework 4 of the class (figure 2.1). This set the benchmark for the following models as this was a simple convolution neural network, which included 2 convolution layers, an average pooling layer, then two linear layers. We then used a softmax function due to the multi class nature of our problem. The accuracy of our first neural network was 65%. This is promising, but leaves a lot of room for improvement in future neural networks.

3.1.2. Model Two. The second network architecture (figure 2.2) consisted of three convolution layers that were each followed by batch normalization. This is then followed by a max pooling and a linear layer, then a softmax function. The accuracy of this model was 73%. This showed some improvement, but it was clear that if we wanted to generalize our model, we needed to radically modify our architecture. We also had computational limits as our initial models were trained on CPU. These limitations coupled with a goal for a more powerful model led to the design of our final model.

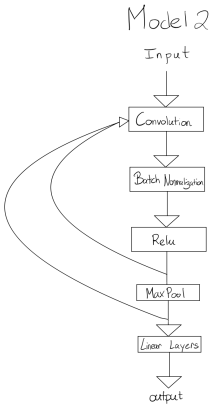

Figure 2.2 Drawing of the architecture of Model Two

3.1.3. LNN-PDI / Model 3. For the final model we got access to a graphics card compatible with CUDA alleviating our previous model's limitation in the data loading process and not the model of the neural network. Drawing influences from some of the current state of the art models, we decided to add layers to our neural network.

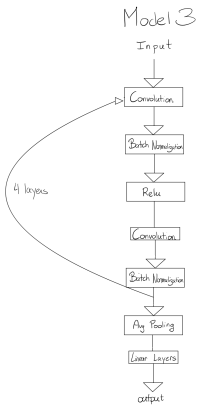

Figure 2.3 Drawing of the architecture of LNN-PDI

Our neural network used the architecture described in figure 2.3. That is, a convolution layer that can alter the number of features to detect patterns, followed by a normalization layer to make sure that each value is 0-mean and 1 standard deviation so that each feature is weighted equally. Adding this layer makes sure that each feature is weighted equally. We follow this with ReLu and then another set of convolution and normalization that work very similarly to the process described above. To help make sure that deeper layers can at least perform as well as shallower ones, we decided to add a shortcut connection. We then use average pooling and a linear layer that maps the pooled features to create the logits for each class. Then we use a softmax function to determine the predicted class of the output. This model has the ability to learn a lot more features and we did our best to make it balanced so that different features can be learned depending on the data. In this case, this model showed a significant improvement. After running this model we saw an accuracy of 89% on the validation and 88% test set. This is a great improvement to our previous models and shows that our architecture is a suitable one for our problem.

3.2. Optimization Methods

Another way our models differed is in the optimization techniques used. The first model used stochastic gradient descent as it is the cheapest and simplest to use, which was a benefit for running on a cpu. This did have some drawbacks as the function was taking very long to converge. Because of these drawbacks, for our second model we opted to use ADAM.

ADAM has benefits such as generally faster convergence compared to SGD, and its lower number of hyper parameters make it easier for tuning. This does use more memory than SGD, but we felt the advantages of ADAM outweighed. After doing some research, we found that ADAMW, which implements the ADAM but with weight decay, seemed to be a popular choice for convolution neural networks. Due to the weight decay, ADAMW often regularizes better and can generalize better. This does mean that like ADAM it has the downside of more cost per iteration, but ADAMW seems to be a better choice than ADAM for a project like this. In conclusion, for our final model we decided to go with ADAMW due to the benefits of its fast convergence, good generalization and lower number of hyper parameters to find.

3.3. Pooling and Activation Functions

We had some choice in the pooling function and the activation function. For our first model we used average pooling as it has the benefit of retaining most of the information from the input and produces a smoother feature mapping. For the second model we wanted to try a different pooling function, so we tried the maximum pooling function. Some benefits of maximum pooling is that the most prominent features get enhanced, which helps the deeper layers to learn. Model 2 which used max pooling is probably not deep enough to use the full benefits of max pooling, but the final model definitely is and we see the benefits from it.

For our activation function we used ReLU. ReLU is computationally inexpensive and due to the positive nature of the function it helps create more efficient models. We also used a softmax function at the end of the model to be able to classify in one of the four possible classes.

For our final model, we decided to use a single max pool function over many of them in order to detect the important features but not lose feature information across the layers as that could lead to problems. Each layer has a ReLU function for the benefits described previously. We decided to use an average pooling function at the end to catch all the information without overfitting and reducing dimensionality. Due to the nature of our problem, our last step is a linear layer with a softmax function to classify the input into one of the four classes.

3.4. Compute Environment

A large factor in our model architecture and experimental design considerations was the limitations of our computational environment. Our first two models were initially developed on, and ran on a 2017 Macbook Pro with 16 GB memory, I7-7920HQ 4-core CPU and a Radeon Pro 560 with 4GB of VRAM. Our final model with more layers required more compute resources to implement and run efficiently hence why we switched our compute environment to a Windows 10 system with 32 GB memory, AMD Ryzen 5 5600X 6-core CPU, and a RTX 4070 with 12 GB of VRAM. The bulk of the speedup (approximately a 20x speedup) from switching machines came from CUDA acceleration on the second Windows machine- the prior MacOS machine experienced incompatibility issues with Pytorch's GPU acceleration. Both machines were running on Python 3.9.5 and Pytorch 2.2.1.

4. Experiment

4.1 Experimental Design

The experiment ran a classical train, validation and test split. The test set is around 17% of the size of the training set, and is larger than 10,000. More details on the sizes and augmentation of the data is discussed in section 2.1. Each model was evaluated on the validation set provided in the data. We used the results from the validation set to help guide us in creating the subsequent models. We also used per class accuracy to make sure that our models were not generalizing to one class, but this was never an issue in any of our models, likely due to the balanced nature of each of the classes. We also looked to minimize false negatives (labeling a fecal matter as healthy when it is in fact not healthy) as those can be incredibly dangerous as our goal is to detect the diseases as fast as possible. We judge that false positives would not be as much of an issue as each picture of the false positives can be independently reviewed by human pathologists to verify if the chicken fecal matter is contaminated. Post confirmation, the proper procedure to take care of the chicken can be initiated. This would be a relatively similar process to the way that the data was labeled.

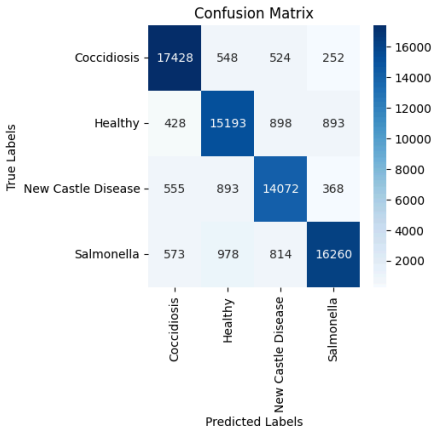

Figure 3.1 a confusion matrix of each classification on the test set of LNN-PDI

4.2 Results

	Precision	Recall	F1
Coccidiosis	92%	93%	92%
Healthy	86%	87%	87%
New Castle	86%	89%	87%
Salmonella	91%	87%	89%

Figure 3.2 Table of results of Precision, Recall and F1 scores of LNN-PDI

Our final model resulted in a test accuracy of 88%. For our previous models we used the validation set in order to preserve the integrity of the evaluation. This is best practice as we do not want to fit the test data to our model, and it is also a way of testing "real world" data as a test for our model. Our validation scores went from 65% for the first model, to 73% in the second model. This showed we were going in the right direction and for the final model we went with a much deeper network which resulted in an 89% validation accuracy.

Model	Train Acc.	Valid Acc.	Mean Loss of last 100	Time
1	71%	65%	1.19	26:33.5
2	77%	73%	0.72	89:47.3
3	**92%**	**89%**	**0.27**	**248:58.6**

Figure 3.3. Table of results of 3 models with Train Accuracy, Validation Accuracy, the last 100 mini-batch Loss of the last epoch and the total runtime of each model.

5. Conclusion

In conclusion, with the potential of convolution neural networks in image classification, we are able to create a model that detects avian diseases early with high accuracy. Our transition from a simple model to more complex architecture, as well as our strategic optimization and iterative enhancement led to a substantial increase in our model accuracy. Our final model, LNN-PDI, used multiple layers and ADAMW as our optimization, we achieved an impressive result of 88% accuracy given our constraints.

Moreover, our work emphasizes the importance of a balanced dataset and a rigorous experimental design to ensure that our model is robust and reliable. Focusing on reducing the false negatives, we aimed to create a tool that could genuinely assist farmers detect diseases in poultry to the effect of greatly improving the health of both the animals and the humans that consume them.

Future research could try deeper architecture to increase accuracy as we had computational limits. Future research could also look into multiple data types in order to create a more comprehensive approach to this problem. Ultimately, this study not only highlights the feasibility of using CNNs for disease detection in poultry, but opens possible avenues for applying similar methodologies across the agricultural domain, potentially transforming the disease management landscape.

References

[1] Jayavrinda Vrindavanam, Pradeep Kumar, Gaurav Kamath, Chandrashekar N, and Govind Patil. (2023). Poultry Pathology Visual Dataset [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DS/3951043

[2] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. (2020). Original ResNet-18 Architecture [Figure]. ResearchGate. https://www.researchgate.net/figure/Original-ResNet-18-Architecture_fig1_336642248

[3] Karpathy, Andrej. (n.d.). CS231n Convolutional Neural Networks for

Visual Recognition. Stanford University. https://cs231n.github.io/convolutional-networks/

[4] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. (2016). Deep Residual Learning for Image Recognition. arXiv. https://arxiv.org/pdf/1607.06450.pdf