EXPLORING MUSHROOM CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Kevin Donohue

Dr. Scott Thatcher, Thesis Advisor
Department of Computer and Data Sciences
School of Science and Mathematics
2024

TRUMAN STATE UNIVERSITY
Kirksville, Missouri

Table of Contents

List of Tables

List of Figures

Abstract

This thesis explores image recognition modeling using convolutional neural networks (CNNs) and their ability to classify mushrooms. A dataset was created by gathering images of mushrooms using a custom Google search engine. Each image in the dataset was opened in the GIMP (GIMP Development Team 2024) image editing software, where they were cropped and exported to directories. The images were then split into training and testing sets for analysis, and the Augmentor (Contributors 2023) module was utilized to generate 10,000 augmented training samples from the training set. Python modules TensorFlow (Authors 2023) and Keras (al. 2023) were used to evaluate four different image recognition models and their ability to predict four classes of mushrooms. After learning from the training set images, the four models were deployed to make predictions on the testing set images and their performances were compared. The thesis concludes with the selection of the best model according to accuracy and loss metrics. The selected model, ResNet50, is implemented in the Mushroom Image Recognition App.

1 Introduction

Image recognition has become an essential tool in various fields, ranging from self-driving cars to medical imaging. A common type of model used for these tasks is convolutional neural networks (CNNs), which utilize a framework of neurons inspired by the way neurons connect in the human brain. In this thesis, three pre-trained CNN models and one CNN model built from scratch are deployed to predict four types of spore dispersal mechanisms found in mushrooms: gills, pores, ridges, and teeth. The ability to identify mushrooms is significant for mushroom foraging, a practice that involves searching for, identifying, and harvesting mushrooms. Mushroom foragers consider various factors when identifying mushrooms, such as taste, seasonality, and habitat. While many characteristics can be used for mushroom identification, this thesis focuses on the physical appearance of mushrooms. The research explores CNN models and their ability to classify mushroom types from images.

2 Methods

2.1 Data Collection

There is no existing dataset for this task, so a custom search engine was developed in Python using an API for specified image collection. The custom search engine takes any Google search query and returns 200 image URLs from the results. This tool was used to gather thousands of URLs representing the spore dispersal mechanisms specified in the queries. This process was repeated for the remaining three classes. The collected URLs were then sorted into four text files, which were opened in Python and parsed, exporting each image to labeled directories in the process.

From these directories, the images were opened in GIMP image editing software, where duplicates, irrelevant images, and invalid data were removed. The resulting dataset consisted of 400 images, with 100 images for each class. Using GIMP, each image was cropped to focus on the undercarriage of the mushrooms, minimizing unnecessary noise in the images. After the cleaning process, the images were exported back into their respective directories with appropriate file names. See Figure 2.1 for a random sample of 16 images from all four directories.

Subset	Percentage	Total
Training	80%	320
Testing	20%	80

Table 2.1: Subset Composition

With data collection and preprocessing complete, the dataset was ready to be split into training and testing subsets. The scikit-learn (Developers 2023) library for Python was used to perform the split, and the resulting subsets were organized into new directories. See Table 2.1 for the composition of these subsets.

Figure 2.1: Image Display

2.2 Data Augmentation

Due to the small sample size of the data, it was decided to increase the number of training samples to enhance the robustness of the convolutional neural network models and prevent overfitting. Data augmentation techniques were applied to the 320 training images, generating 10,000 augmented training samples. The primary goal of data augmentation is to improve the robustness and accuracy of machine learning models, allowing them to perform well even on small or poorly representative datasets (Mumunia and Mumuni 2022). These techniques create new samples by altering existing images, resulting in unique training samples that diversify the training set.

The following data augmentation techniques from the Augmentor module were applied:

- **Flip Left-Right**: Horizontally flips the image with a probability of 50%.
- **Rotate**: Rotates the image up to 180 degrees left or right with a 50% probability.
- **Gaussian Blur**: Applies a Gaussian blur to smooth the image, averaging pixel values weighted by a Gaussian distribution, and reducing noise and detail with a probability of 20%.
- **Zoom**: Zooms into the image with a probability of 20%, scaling between 1.1x and 1.5x with a probability of 30%
- **Random Brightness**: Randomly adjusts the brightness by a factor between 0.7 and 2 with a probability of 30%.
- **Random Contrast**: Randomly adjusts the contrast by a factor between 0.7 and 2 with a probability of 30%.
- **Random Distortion**: Applies random distortions to the image based on a grid of specified width and height with a probability of 30%.
- **Shear**: Skews the image, tilting straight lines while preserving their parallelism with a probability of 30%.

- **Crop Random**: Randomly crops the image to 80% of its area with a probability of 50%.

The augmented training set of 10,000 images contains approximately 31 unique samples for each of the original 320 training images, incorporating different combinations of augmentations. See Figure 2.2 for a visualization of these techniques applied to the same image of a mushroom with ridges.

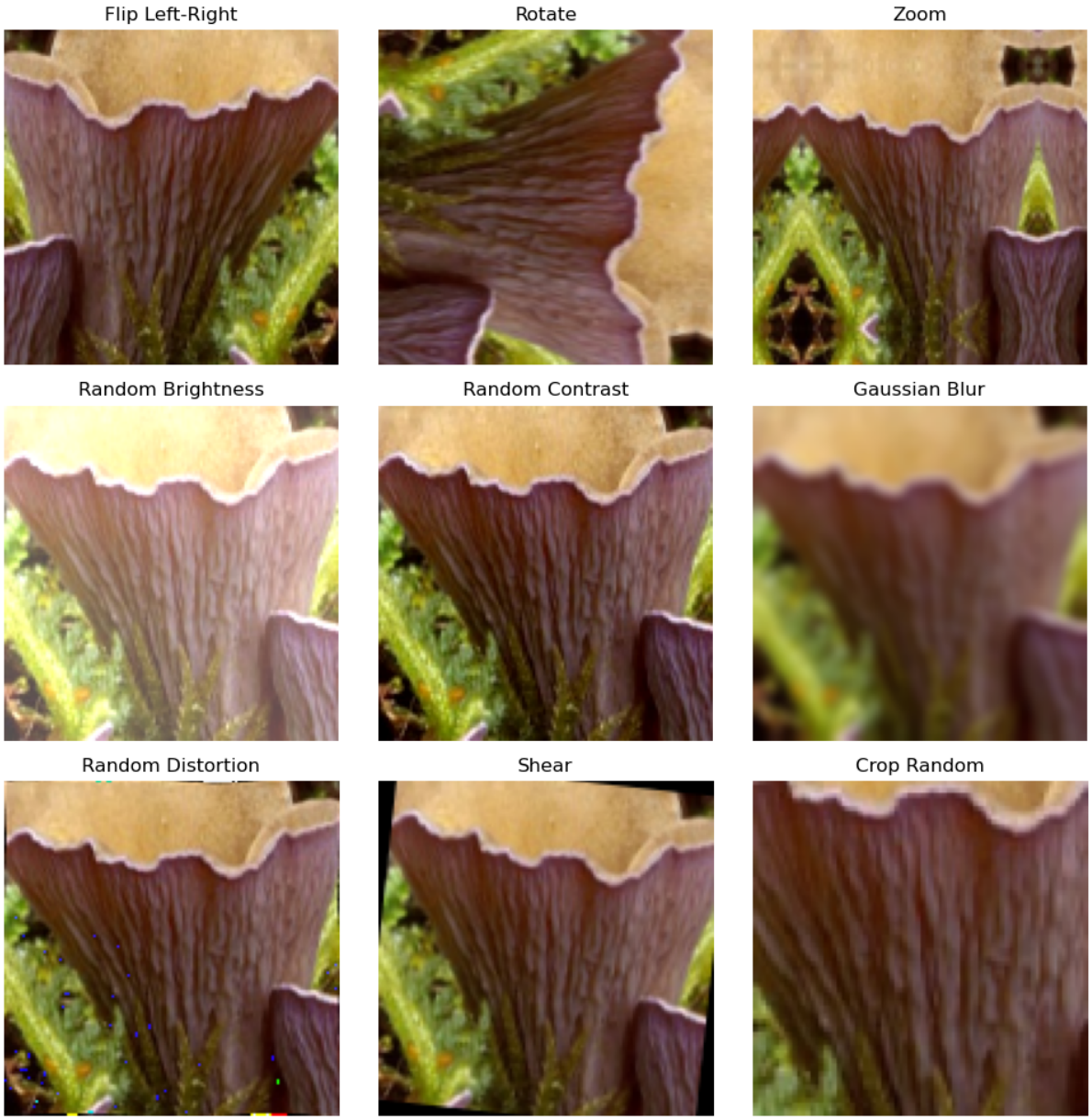


Figure 2.2: Augmentation Techniques

2.3 CNN for Image Classification

With an augmented training set, the data were ready for modeling. Four CNN models were fitted to the data: VGG16, ResNet50, MobileNetv2, and the model made from scratch. The architecture of all four of the CNN models consist of an input layer, hidden layers, and output layers. The images are processed in the input layer, the features are learned in the hidden layers, and classification takes place in the ouput layers.

2.3.1 Input Layer

In the input layer, each image is resized to 128 by 128 pixels with three color channels. These channels measure the intensity of red, green, and blue (RGB) in each pixel. Each pixel is assigned three numerical values corresponding to the intensity of red, green, and blue in that pixel.

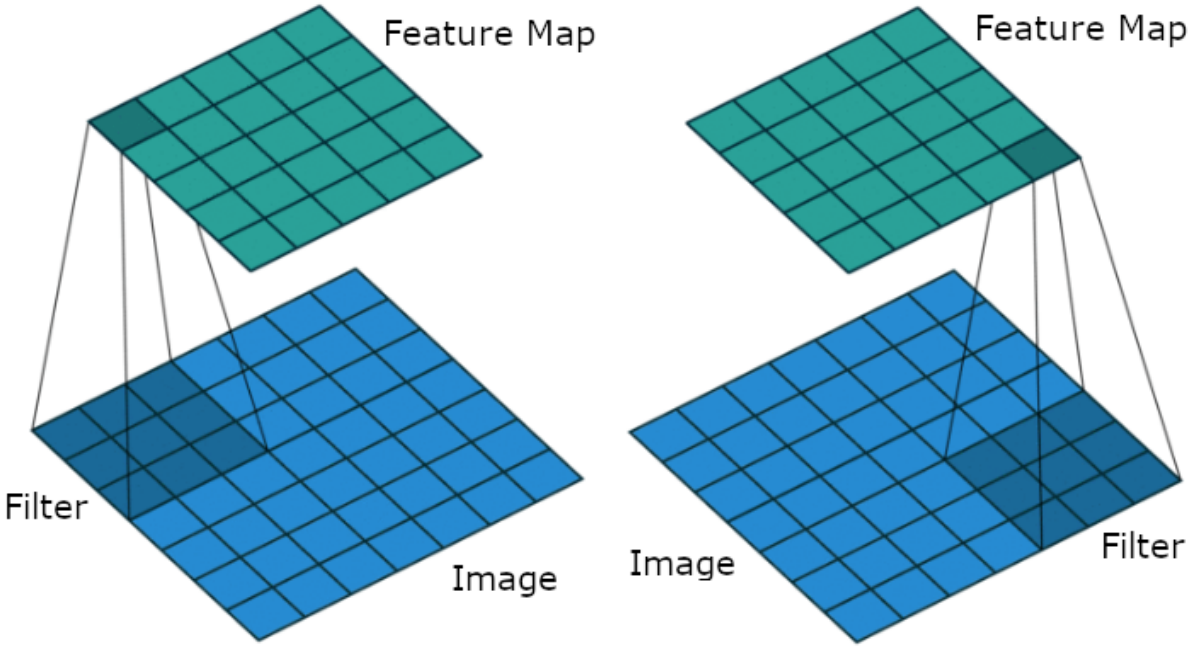
2.3.2 Hidden Layers

The values from the input layer are sent to the hidden layers, where the features used for classification are learned. The hidden layers applied in this research include convolutional layers, max-pooling layers, dropout layers, and bottleneck layers. These operations help the network refine its feature learning while maintaining efficiency.

2.3.2.1 Convolutional Layers

The first hidden layer is the convolutional layer, which applies a set of filters to the input to produce a feature map (Amidi and Amidi 2024). A filter in a CNN is a small matrix of randomized weight values that scans across the input image, pixel by pixel, in each of the three channels. Each position in the filter corresponds to a neuron that processes a small patch of the image. For example, a 3 by 3 pixel filter applied to an RGB image has three separate 3 by 3 matrices for the Red, Green, and Blue channels. As the filter slides, or convolves, over the image, it processes a new patch at

each position. See Figure 2.3a and Figure 2.3b for illustrations of a filter applied at two different positions.



(a) Position a

(b) Position b

Figure 2.3: Convolution

At each position, the neuron performs element-wise multiplication between the filter's weights and the numerical pixel values from the RGB channels. The resulting values are summed, producing a single scalar value for that position. These scalar values, computed across all positions, form a matrix called a feature map. This map represents the output of the convolutional layer after the filter has scanned the entire image.

A visualization of how a feature map is generated from image input values and a filter can be seen in Figure 2.4 from (Baeldung 2024) and Figure 2.4, where a scalar value of 31 is calculated for a specific position on the feature map.

Element-wise multiplication is the operation of multiplying corresponding elements of the image patch matrix and the filter matrix and then summing the results. This operation is performed at each position as the filter slides over the image, producing a feature map. Each value in the feature map corresponds to the output of a neuron processing a specific part of the image.

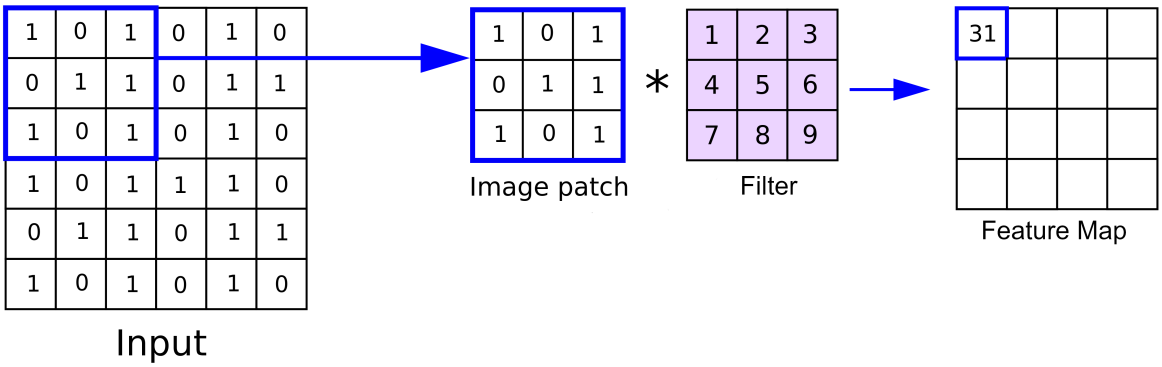


Figure 2.4: Element-Wise Multiplication

Feature maps highlight particular features or patterns, such as edges or textures, that the filter detects. Their dimensions are determined by the size of the input, the size of the filter, the stride, and whether padding is applied. In this thesis, 'same' padding is employed to ensure that the filter learns each pixel at the same number of positions. See Figure 2.3a and Figure 2.3b for visualizations of how a filter convolves over an image to generate a feature map.

2.3.2.2 Max-Pooling Layers

Max-pooling layers are a type of hidden layer applied after convolutional layers. These layers reduce the spatial dimensions of feature maps by selecting the maximum value from small regions of the image (Shanmugamani 2024). This operation retains the most significant information while decreasing model complexity and improving model efficiency.

In this thesis, max-pooling layers were incorporated into the model from scratch to speed up its training process. For example, max-pooling with a 2 by 2 filter significantly compresses the information in the data while preserving critical details. See Figure 2.5 for a visualization of a max-pooling operation.

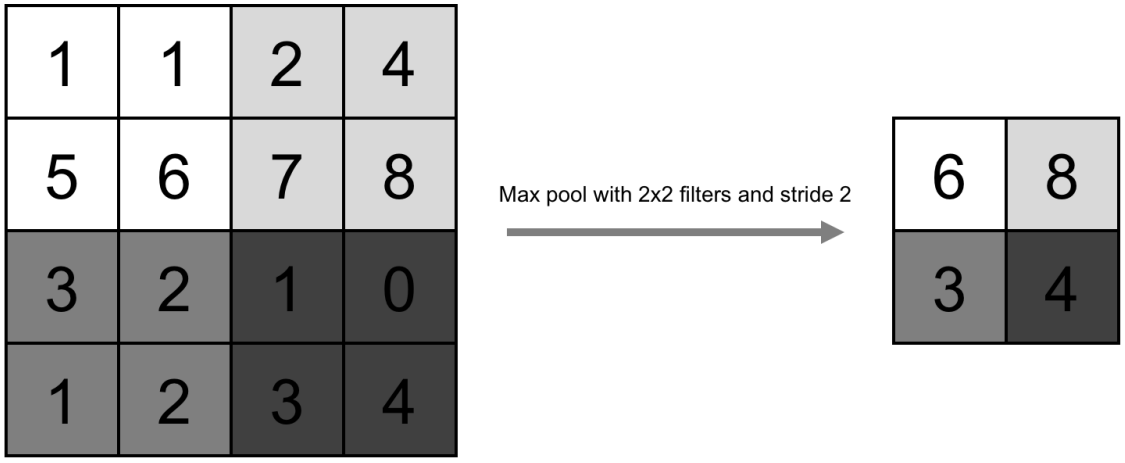


Figure 2.5: Max-Pooling

2.3.2.3 Dropout Layers

Another hidden layer found in convolutional blocks is dropout layers. During training, dropout layers randomly deactivate a fraction of neurons, preventing the model from becoming too dependent on specific neurons and thus helping prevent overfitting (DotNetTutorials 2024). Overfitting occurs when the model becomes too dependent on features learned from the training set and cannot generalize well when making predictions on new data. See Figure 2.6 for the effect of dropout layers on networks.

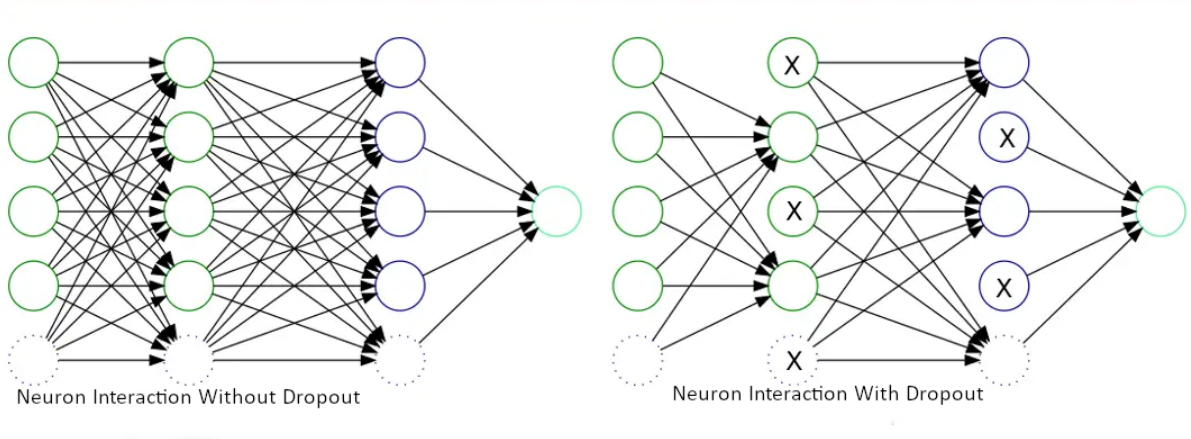


Figure 2.6: Dropout Layers

When evaluating the models on the testing set, the neurons that were dropped during training

are present in the model. Since random neurons are dropped during each training iteration, each neuron becomes more diverse at different iterations, creating a more generalized model. In this thesis, dropout layers are applied in all four models.

2.3.2.4 Bottleneck Layers

A significant architectural feature found in the MobileNetV2 and ResNet50 models is the bottleneck layer. A bottleneck layer is a layer in the network that has fewer neurons than the layers preceding and succeeding it (Marchi and Mitchell 2024). This layer with fewer neurons creates a narrow section in the network, or a bottleneck. The bottleneck layer compresses the most relevant and useful information from the input features into fewer neurons to reduce the complexity of the model. See Figure 2.7 for a visual representation of a bottleneck layer and network architecture.

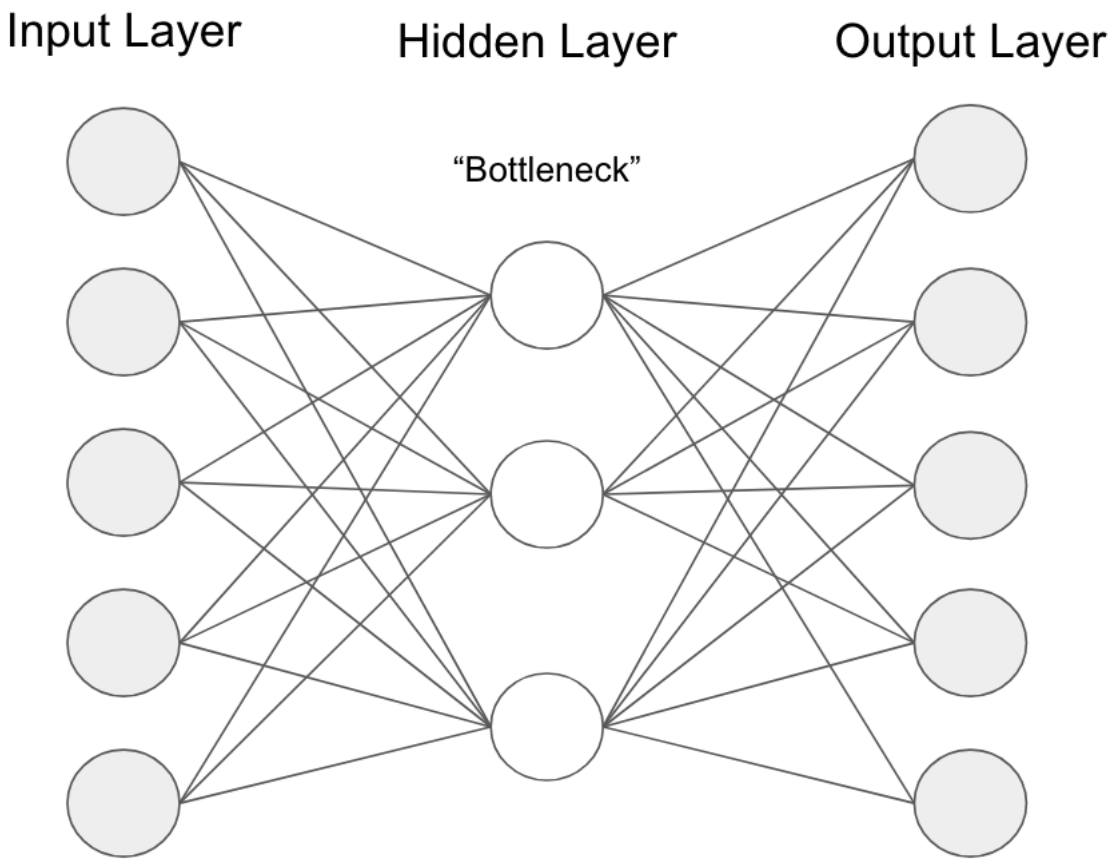


Figure 2.7: Bottleneck Layers

2.3.3 L2 Filter Regularization

L2 filter regularization is not a hidden layer in itself, but a technique that can be applied to convolutional layers to help the models converge. L2 regularization, also known as weight decay, encourages the model's weights to stay small by penalizing large values, as shown in red (Valverde 2018). This results in the weights being concentrated around 0, as shown by the circular blue weights near the origin. As the regularization penalty increases, the weights are pushed toward the edges of the circle, illustrating how L2 regularization minimizes large weight values. Conceptually, L2 regularization uses a circle as a hard constraint for weights. If the weights become too large and move outside of the constraint region, the coefficients are regularized and move back towards the boundary of the constraint region. In this research, L2 regularization is applied so that the models can learn more generalized features of the mushrooms. The weight values seen in Figure 2.8 correspond to the learned features of the input data.

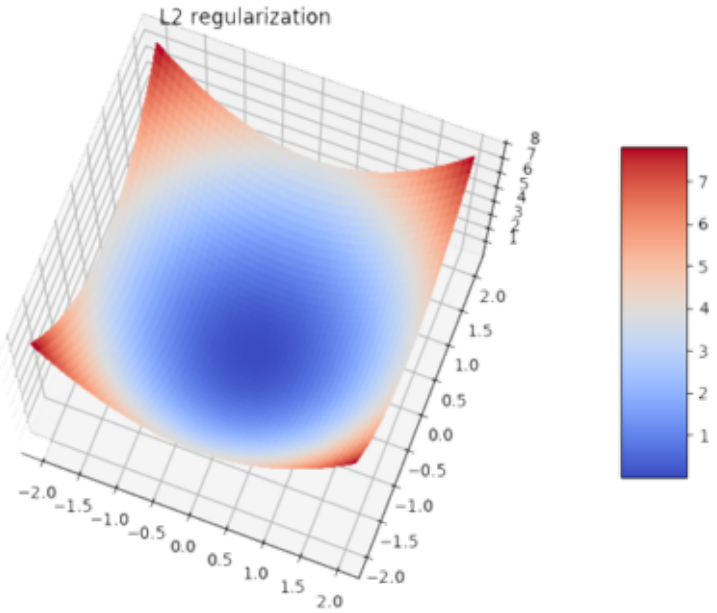


Figure 2.8: L2 Filter Regularization

2.3.4 Output Layers

In a fully connected layer, each neuron connects to every neuron in both the preceding and subsequent layers, summarizing the information across the network. This layer transforms the two-

dimensional feature maps produced by earlier layers into a one-dimensional vector, called logits, which numerically represents the features the model has learned from the input data. The logits are then passed through a softmax activation function, which converts them into a probability distribution over the target classes, ensuring the probabilities sum to one. The class with the highest probability becomes the model's predicted output. For example, in the architecture depicted in Figure 2.9, the softmax layer predicts one of three classes for classification.

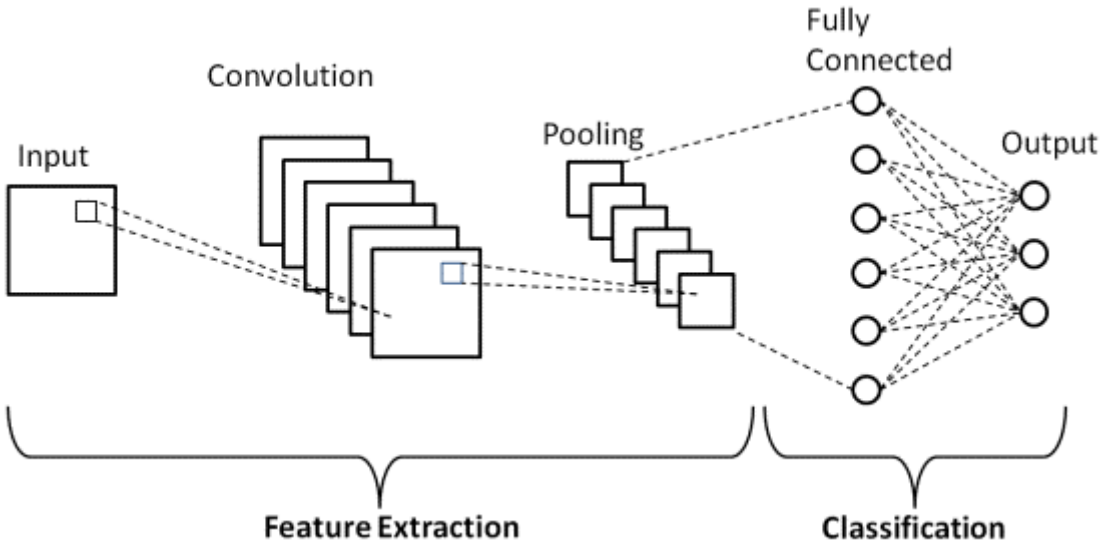


Figure 2.9: Network Architecture

2.3.5 Deep Learning

Deep learning refers to the processes that occur in the input, hidden, and output layers, which are repeated over time, or at each epoch. An epoch refers to one complete iteration of the entire training set through a neural network. During each epoch, the model iterates over all the training examples, makes predictions on the testing set, computes errors based on the difference between predictions and actual labels, and updates its weights to reduce those errors. Models often improve incrementally with each subsequent epoch, and the filters become more specialized, focusing on the critical features needed to classify the mushrooms.

2.3.6 Transfer Learning

Transfer learning is a powerful technique in deep learning where a pre-trained model is adapted to solve a different but related problem. In this research, three pre-trained models are deplouyed to leverage the information learned from the ImageNet (Deng et al. 2009) database to make predictions on the mushroom dataset. ImageNet which contains over a million labeled images across 1,000 categories, which help these models to generalize well to the mushroom dataset. Transfer learning significantly reduces the training time and can produce more accurate classification results.

2.4 Model Structure

2.4.1 VGG16 Model

VGG16 (Simonyan and Zisserman 2015) is a pre-trained, deep convolutional network model for image classification. The "16" in VGG16 refers to the number of layers with weights that the network has, which includes 13 convolutional layers and 3 fully connected layers. VGG16 uses 3 by 3 convolutional layers stacked on top of each other, with max-pooling layers in between. While VGG16 is not the deepest model, it still has a large number of weights.

2.4.2 ResNet50 Model

ResNet50 (He et al. 2016) is a complex pre-trained model named after the "50," which refers to the number of layers in the network, specifically 49 convolutional layers and one fully connected layer. ResNet50 is structured using bottleneck layers. The core unit of ResNet is the bottleneck block, which includes a 1 by 1 convolution for reducing dimensions and a 3 by 3 convolution for processing features. ResNet50 is the deepest model with the largest number of weights.

2.4.3 MobileNetV2

The MobileNetV2 (Sandler et al. 2018) pre-trained model was introduced by Google in 2018 and uses bottleneck layers to preserve information while reducing the number of weights and computations. It is optimized for low-latency and lower computational power, making it suitable for image recognition tasks on smartphones. MobileNetV2 has fewer weights than VGG16 and ResNet50, making its training time much quicker.

2.4.4 Building CNN Model from Scratch

The model from scratch was created with 3 convolutional layers with 256, 512, and 1024 filters, creating a similar number of weights as the pre-trained models. The model from scratch was built with 2 by 2 max-pooling, dropout, and kernel regularization in all 3 convolutional blocks to improve convergence. See Table 2.2 to compare the complexities of the model architectures.

Model	Number of Weights	Training Speed
VGG16	14,813,006	Slow
ResNet50	23,980,942	Moderate
MobileNetV2	10,638,862	Fast
New Model	10,866,158	Slow

Table 2.2: Model Architecture Comparison

3 Results

While each model is structured differently, each of them is designed for image classification. To evaluate these models, they were trained to learn features from the augmented training set and deployed to make predictions on the testing set. See Figure 3.1 to visualize how the models make predictions on the testing set.

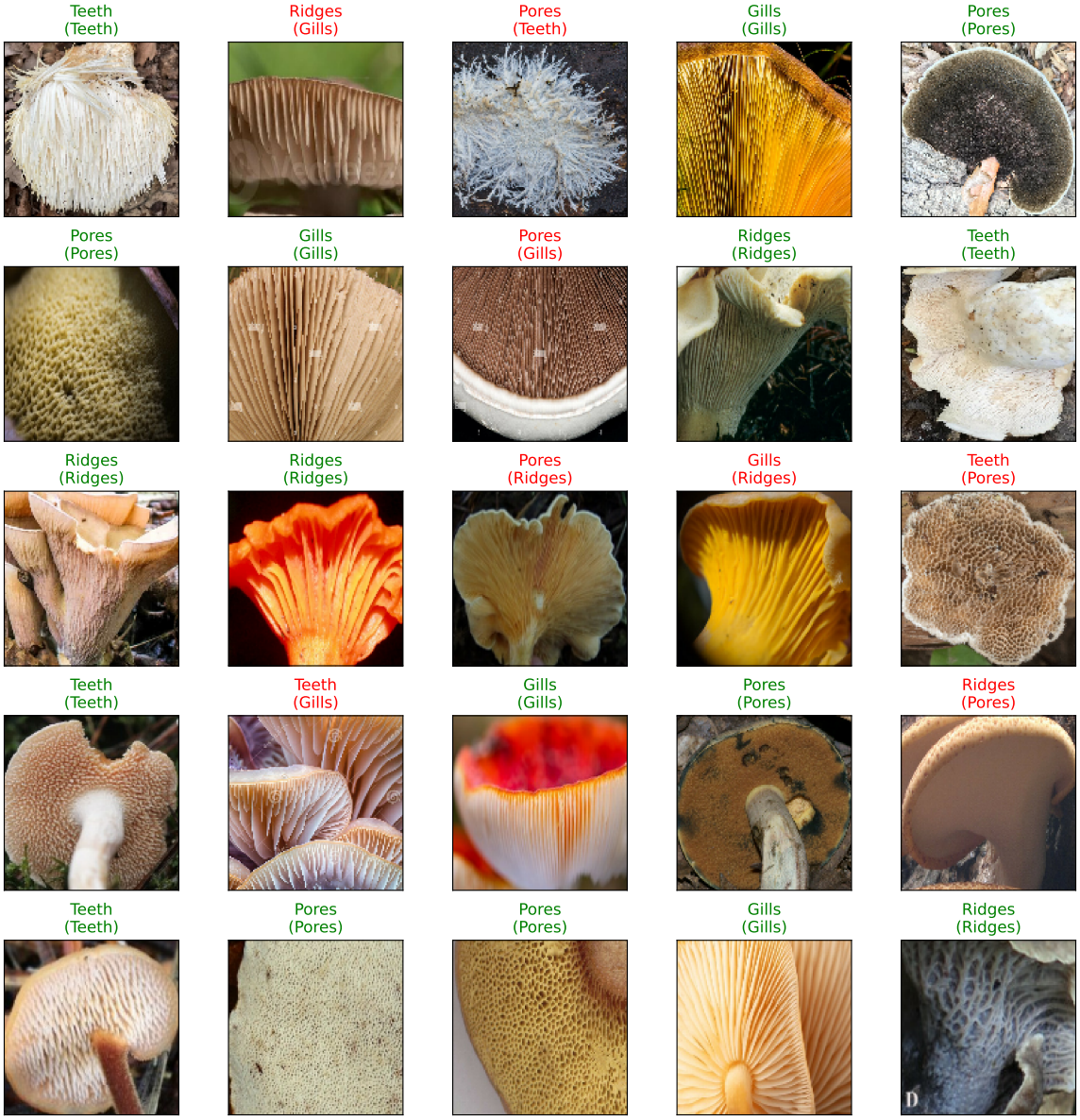


Figure 3.1: Prediction

The predictions on this 25-image sample from the testing set show 18 correct predictions in green

and 7 incorrect predictions in red, resulting in a validation accuracy of 72%. After each model made predictions on all 80 of the testing set images, their performances were evaluated based on accuracy and loss metrics.

3.1 VGG16 Model Performance

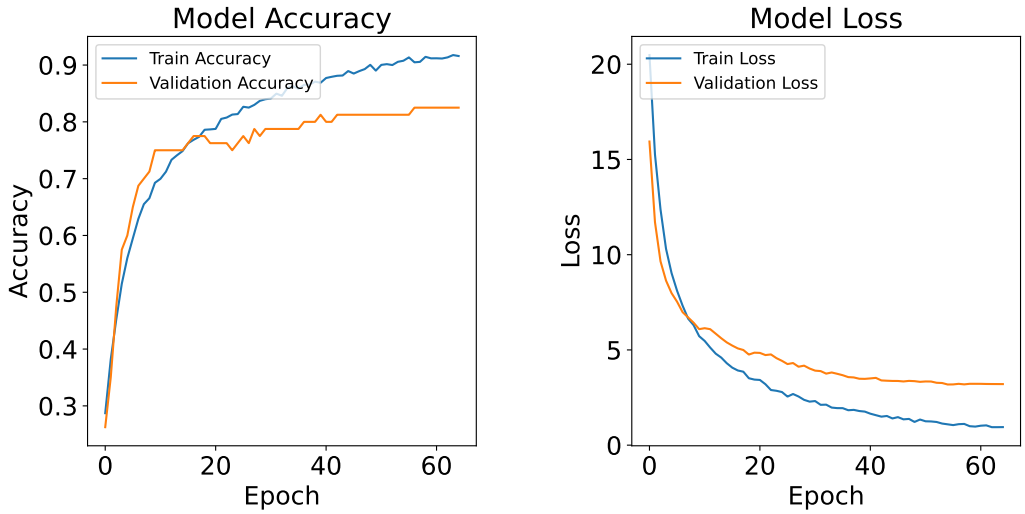


Figure 3.2: VGG16 Model Accuracy & Loss Plots

The graphs in Figure 3.2 show the model reaching a validation accuracy of 0.82, indicating that the model performs well on the testing set. The model reached its highest validation accuracy at the 56th epoch, where the loss begins to plateau. While there is still some overfitting, the model learns the training data well.

	precision	recall	f1-score	support
Gills	1.00	0.90	0.95	20.00
Pores	0.76	0.80	0.78	20.00
Ridges	0.86	0.90	0.88	20.00
Teeth	0.70	0.70	0.70	20.00
accuracy	0.82	0.82	0.82	0.82

	precision	recall	f1-score	support
macro avg	0.83	0.82	0.83	80.00
weighted avg	0.83	0.82	0.83	80.00

Table 3.1: VGG16 Classification Report

The confusion matrix in Figure 3.3 shows that the model correctly predicted 18 out of 20 instances for both gills and ridges, demonstrating strong performance in these categories. For pores, the model achieved 16 correct predictions out of 20, while for teeth, it was accurate 14 times out of 20, indicating slightly lower performance for this class. Overall, the model's highest accuracy achieved on the validation set was 82%.

3.2 ResNet50 Model Performance

The plots in Figure 3.4 show that the model reached a validation accuracy of 0.86, which is greater than the validation accuracy of the new model and VGG16. The model performs well on the new data and reached its highest validation accuracy at the 14th epoch as the loss continued to plateau.

	precision	recall	f1-score	support
Gills	0.90	0.90	0.90	20.00
Pores	0.81	0.85	0.83	20.00
Ridges	0.94	0.85	0.89	20.00
Teeth	0.81	0.85	0.83	20.00
accuracy	0.86	0.86	0.86	0.86
macro avg	0.87	0.86	0.86	80.00
weighted avg	0.87	0.86	0.86	80.00

	precision	recall	f1-score	support

Table 3.2: ResNet50 Classification Report

The confusion matrix seen in Figure 3.5 shows the ResNet50 model's performance on the validation set. gills were correctly identified in 18 out of 20 instances, while pores, ridges, and teeth were each predicted accurately 17 out of 20 times. This uniformity in performance reflects the model's ability to generalize well across most categories. With a total of 69 correct predictions out of 80, the highest accuracy on the validation set stands at 86%.

3.3 MobileNetV2 Model Performance

The plots shown in Figure 3.6 show the model reaches a validation accuracy of 0.81 at the 7th epoch before it begins to overfit the training data. The accuracy and loss plots suggest that while the model is capable of capturing important features early on, it eventually loses its ability to generalize effectively. The model becomes too dependent on the features learned in the training data when it is making predictions on the testing set.

	precision	recall	f1-score	support
Gills	0.95	0.90	0.92	20.00
Pores	0.61	0.95	0.75	20.00
Ridges	0.89	0.85	0.87	20.00
Teeth	1.00	0.55	0.71	20.00
accuracy	0.81	0.81	0.81	0.81
macro avg	0.86	0.81	0.81	80.00
weighted avg	0.86	0.81	0.81	80.00

Table 3.3: MobileNetV2 Classification Report

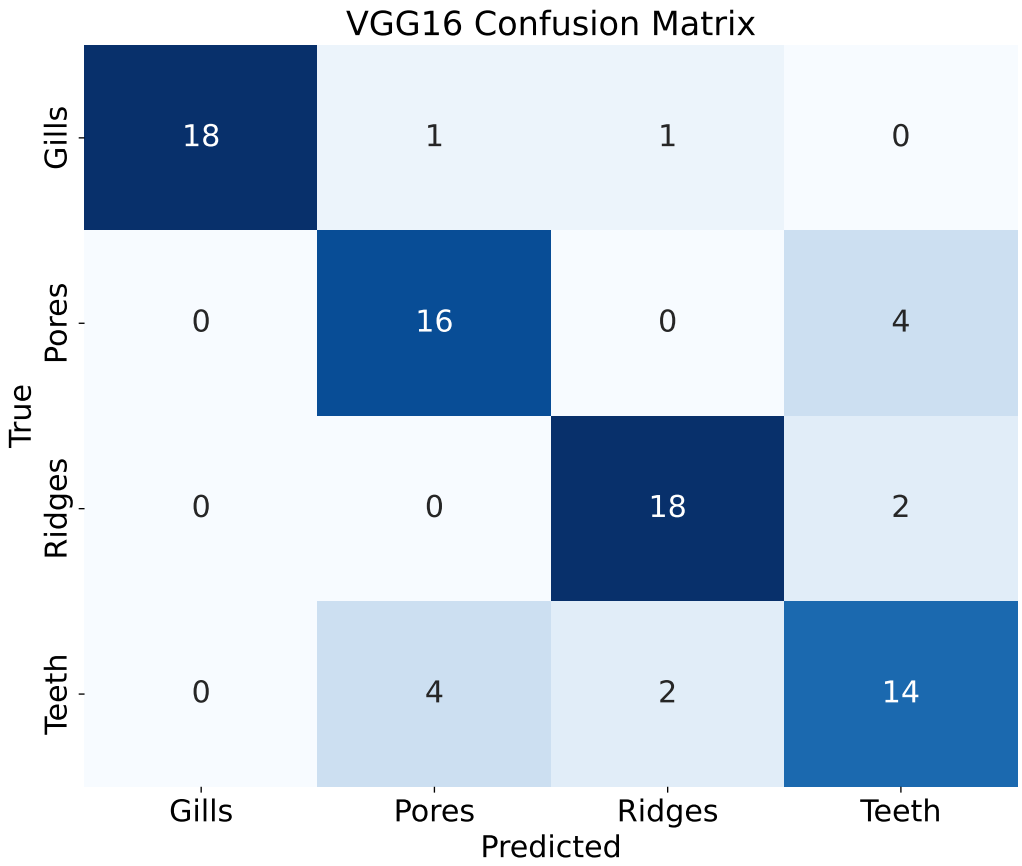


Figure 3.3: VGG16 Model Confusion Matrix

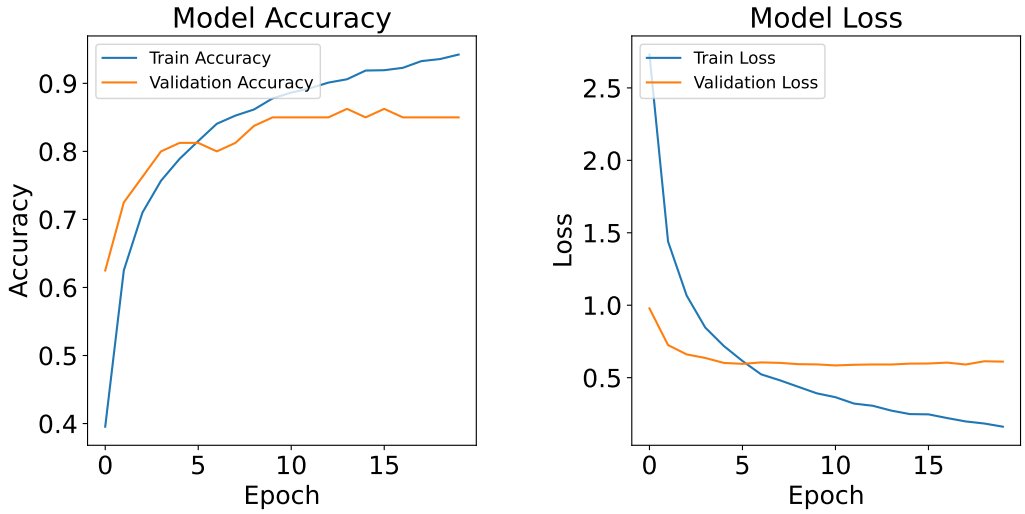


Figure 3.4: ResNet50 Model Accuracy & Loss Plots

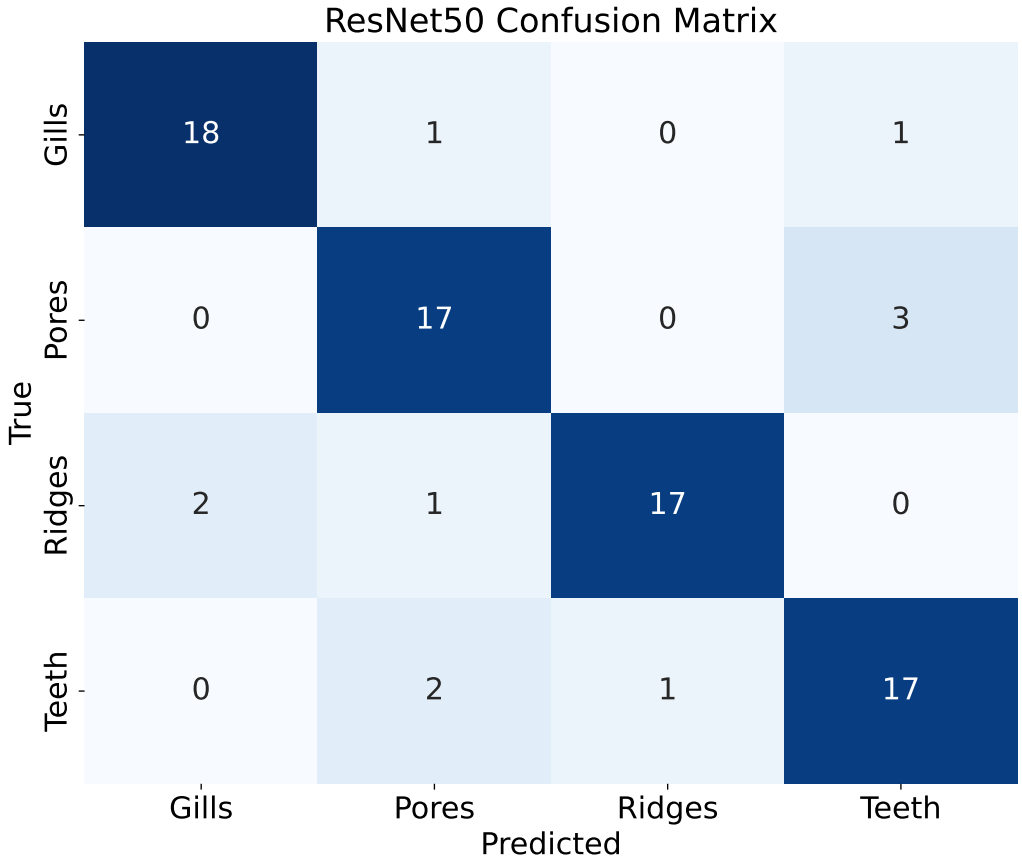


Figure 3.5: ResNet50 Model Confusion Matrix

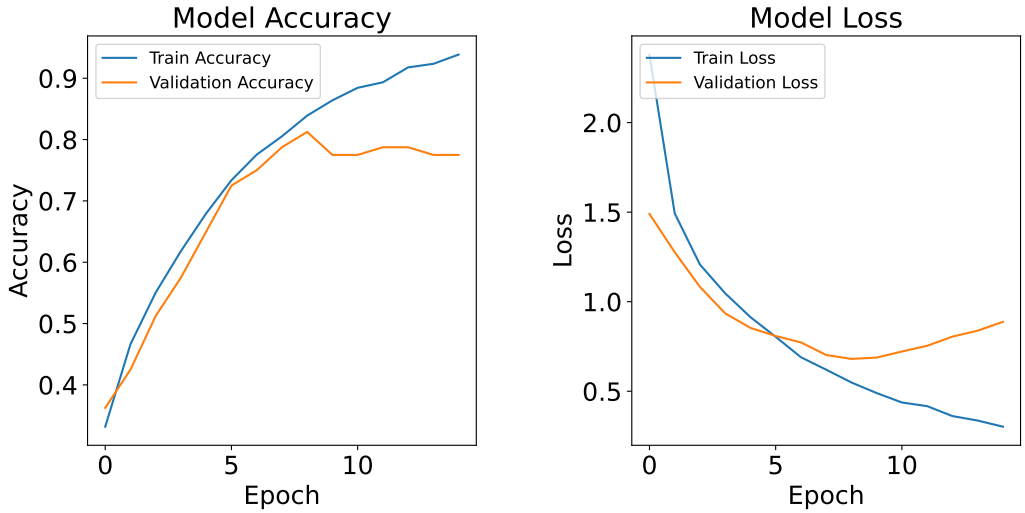


Figure 3.6: MobileNetV2 Model Accuracy & Loss Plots

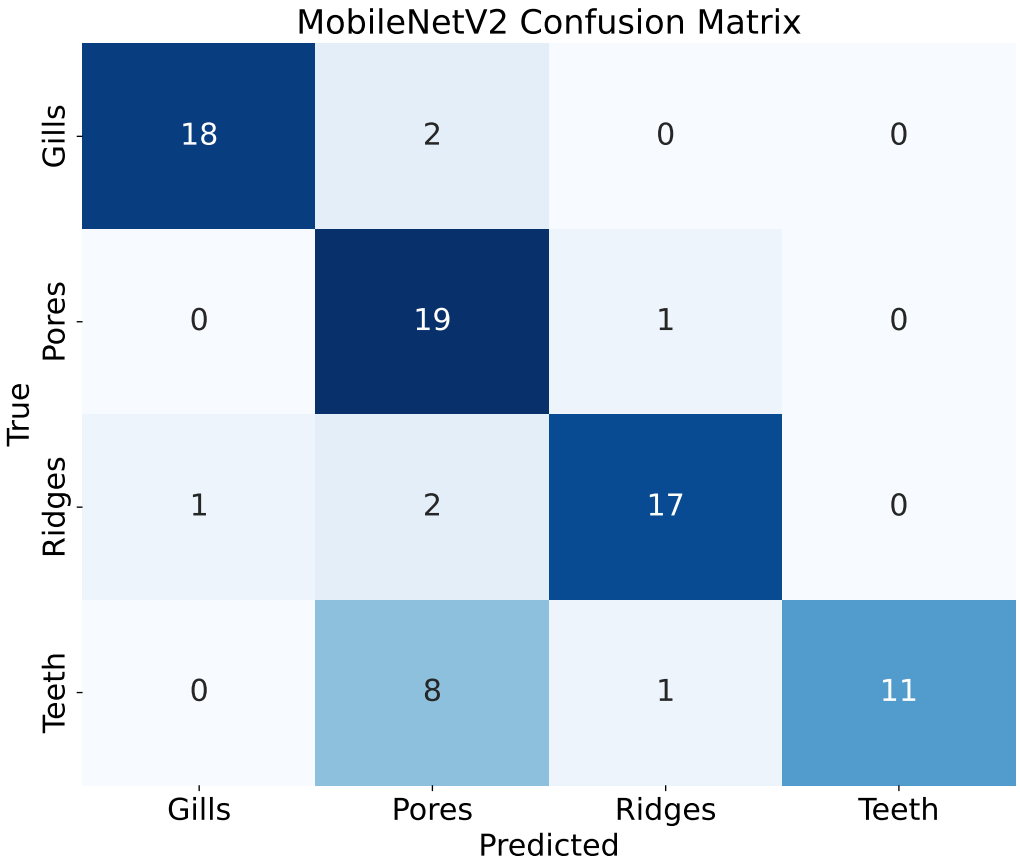


Figure 3.7: MobileNetV2 Model Confusion Matrix

See Figure 3.7 for the MobileNetV2 model's confusion matrix, which shows the model correctly predicted 18 out of 20 instances of gills, 19 out of 20 for pores, 17 out of 20 for ridges, and 11 out of 20 for teeth. Notably, the 19 correct predictions for pores represent the highest precision achieved by any model for a single class. However, the relatively lower precision for teeth kept the overall accuracy down. Overall, the model achieved a total of 65 correct predictions out of 80, resulting in an accuracy of 81% on the validation set.

3.4 New Model From Scratch Performance

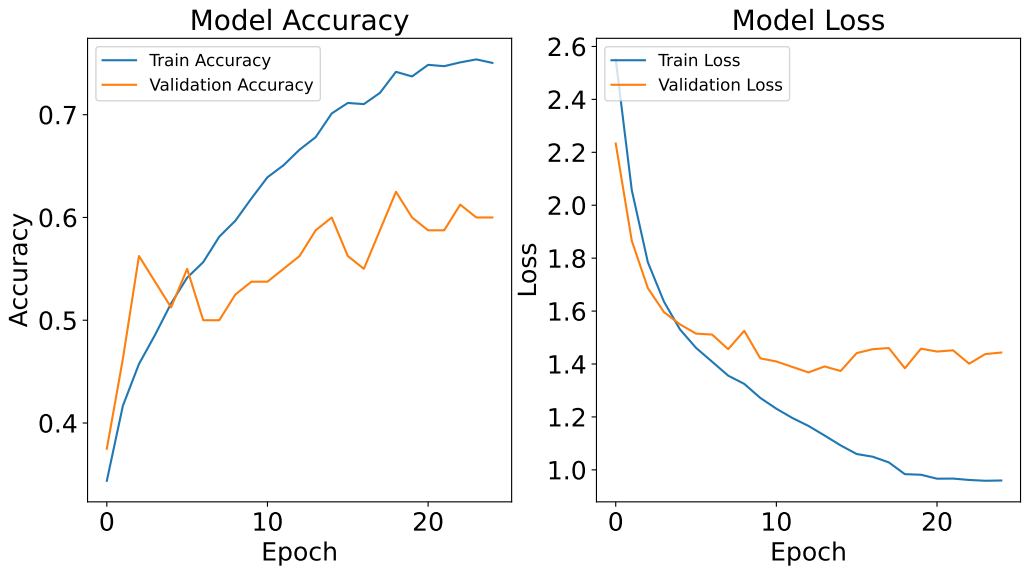


Figure 3.8: New Model Accuracy & Loss Plots

The graphs in Figure 3.8 show the model reaches a validation accuracy of 0.62. During training, the validation accuracy and loss plateau around the 15th epoch, suggesting that the model struggles to improve beyond a certain point. This behavior indicates that the model is either underfitting the data or is unable to capture important patterns.

	precision	recall	f1-score	support
Gills	0.73	0.40	0.52	20.00
Pores	0.48	0.70	0.57	20.00

	precision	recall	f1-score	support
Ridges	0.67	0.60	0.63	20.00
Teeth	0.73	0.80	0.76	20.00
accuracy	0.62	0.62	0.62	0.62
macro avg	0.65	0.62	0.62	80.00
weighted avg	0.65	0.62	0.62	80.00

Table 3.4: New Model Classification Report

The confusion matrix seen in Figure 3.9 shows that when the new model made predictions on the testing set, gills were correctly identified in only 8 out of 20 cases, the lowest accuracy for any class, while pores were predicted correctly 14 times, ridges 12 times, and teeth 16 times out of 20. The model demonstrated its best performance when identifying mushrooms with teeth but struggled notably with gills. Overall, the model correctly classified 50 out of 80 samples, yielding an accuracy of 62% on the validation set.

3.5 Model Comparison

Model Name	Validation Accuracy
New Model	0.62
VGG16 Model	0.82
ResNet50 Model	0.86
MobileNetV2 Model	0.81

Table 3.5: Model Comparison

As seen in Table 3.5, the ResNet50 model achieved the highest performance on the validation set,

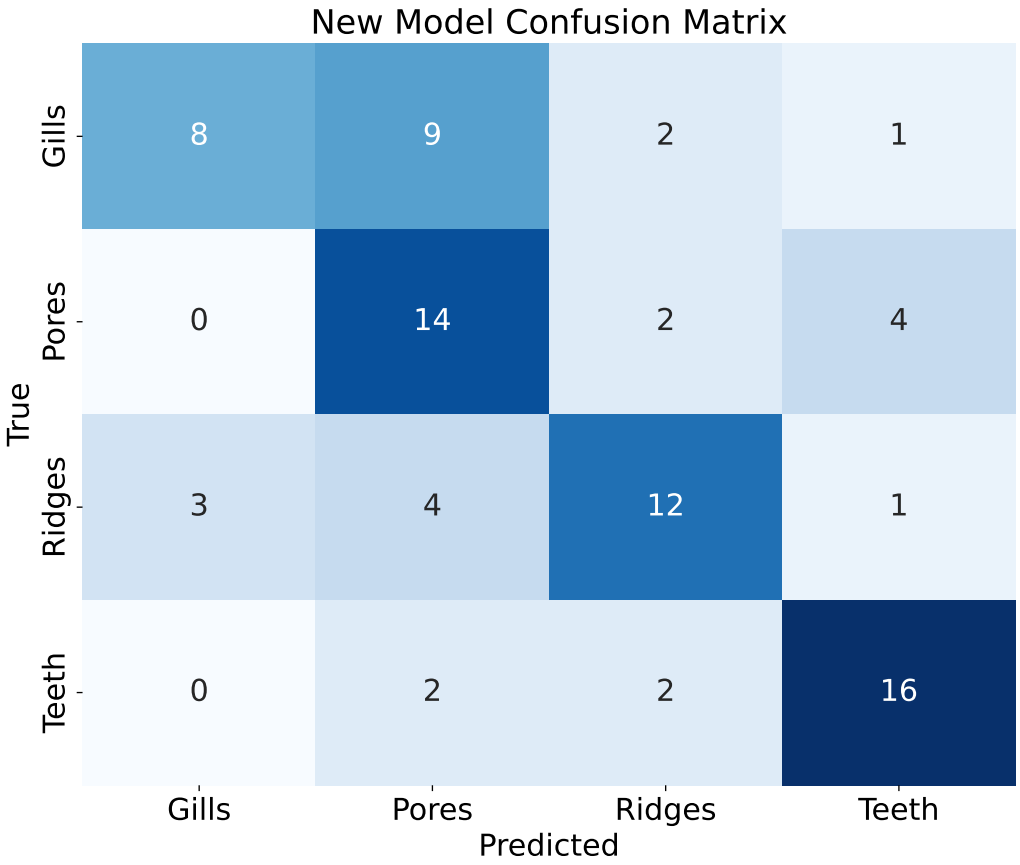


Figure 3.9: New Model Confusion Matrix

demonstrating superior accuracy in making predictions. MobileNetV2 and VGG16 also performed well, delivering strong and reliable results, though slightly behind ResNet50. In contrast, the model trained from scratch struggled to achieve comparable accuracy. This is primarily because it did not utilize transfer learning, unlike the pre-trained models, which benefited from starting with pre-existing weights learned from extensive datasets such as ImageNet.

3.6 Shiny App Development

The artifact for this thesis is an app that allows users to upload an image of a mushroom's undercarriage, which is then classified into one of four categories: gills, pores, ridges, or teeth. The app was developed using the web application framework, Shiny (Chang et al. 2024). When a user accesses the app, the ResNet50 model is deployed to classify the image the user inputs. The app is not intended to determine if mushrooms are safe to consume; rather, it can be used as a supplement for species identification. Try the app here: Mushroom Image Recognition App

4 Discussion

For future research, the model trained from scratch could be enhanced by training on a massive dataset like the 1.2 million images from ImageNet, and then reevaluated for predictions on the mushroom dataset. Access to such a large and diverse training set could enable the model to learn more distinct patterns, potentially improving the classification of the four mushroom types.

All three pre-trained models—ResNet50, MobileNetV2, and VGG16—achieved similar results on the mushroom dataset despite their differing architectures. This suggests that the specific features learned by these architectures may converge when fine-tuned on a smaller dataset like the one used in this thesis. The high performance of the pre-trained models on the mushroom dataset highlights the robustness of transfer learning across various model designs.

Future exploration could focus on techniques to prevent overfitting, as this was an issue encountered with all the pre-trained models on the mushroom dataset. The pre-trained models tended to learn the features of the mushrooms too well before overfitting techniques were applied.

References

al., François Chollet et. 2023. "Keras: The Python Deep Learning API." https://keras.io/.

Amidi, Shervine, and Afshine Amidi. 2024. "CS 230 - Convolutional Neural Networks Cheatsheet." https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

Authors, TensorFlow. 2023. "TensorFlow: An End-to-End Open Source Machine Learning Platform." https://www.tensorflow.org/.

Baeldung. 2024. "Convolutional and Fully Connected Layers in Neural Networks." https://www.baeldung.com/cs/neural-networks-conv-fc-layers.

Chang, Winston, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2024. *Shiny: Web Application Framework for r*. RStudio, PBC. https://CRAN.R-project.org/package=shiny.

Contributors, Augmentor. 2023. "Augmentor: Image Augmentation for Machine Learning." https://augmentor.readthedocs.io/en/latest/.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. "ImageNet: A Large-Scale Hierarchical Image Database." In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–55. IEEE.

Developers, Scikit-learn. 2023. "Scikit-Learn: Machine Learning in Python." https://scikit-learn.org/stable/.

DotNetTutorials. 2024. "Dropout Layer in CNN." https://dotnettutorials.net/lesson/dropout-layer-in-cnn/.

GIMP Development Team. 2024. *GIMP: GNU Image Manipulation Program*. The GIMP Team. https://www.gimp.org.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning for Image Recognition." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–78.

Marchi, Leonardo De, and Laura Mitchell. 2024. *Hands-on Neural Networks*. Packt Publishing.

Mumunia, Alhassan, and Fuseini Mumuni. 2022. "Data Augmentation: A Comprehensive Survey of Modern Approaches." *ScienceDirect* Volume Number (Issue Number): 1. https://doi.org/10.1016/j.example.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–20.

Shanmugamani, Rajalingappaa. 2024. "Deep Learning for Computer Vision: Max Pooling." https://www.oreilly.com/library/view/deep-learning-for/9781788295628/d6fe3d14-d576-40e3-b76d-c60bc316ba80.xhtml.

Simonyan, Karen, and Andrew Zisserman. 2015. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In *3rd International Conference on Learning Representations, ICLR 2015*. https://arxiv.org/abs/1409.1556.

Valverde, Juan Miguel. 2018. "Difference Between L1 and L2 Regularization: Implementation and Visualization in TensorFlow." https://laid.delanover.com/difference-between-l1-and-l2-regularization-implementation-and-visualization-in-tensorflow/.