A Computer Vision Approach to Bee Species Classification

Anthony Geglio* †, Nehal Salik* †, Alex Putman * †
*College of Computing,
Michigan Technological University
Houghton, MI 49931

†Denotes equal contribution
ajgeglio@mtu.edu, msalik@mtu.edu, amputman@mtu.edu

***Abstract*—This project implements a Support Vector Machine and convolutional neural network models of varying complexity to identify the most accurate model to classify images of bees. The challenge is made difficult due to the fact that each image has a colorful background which is the flower the bee is pollinating. Bombus and apis pollinate a variety of flowers and tend to be attracted to the same vibrant colors. The comparison shows that transfer learning and using a deep residual neural network architecture is by far the most effective way to classify this data.**

***Index Terms*—Image Classification, Computer Vision**

I. Introduction

Bees play an important role in the ecosystem, and their populations are endangered. The automated classification of bee species with computer vision can be applied to scientific studies on bee species, and aid bee population management. In the natural world, the background can include a large amount of variation in color and shapes of objects not important to the classification task, and small insects are small and unlikely to be cropped to a high resolution. The goal of this study is to explore the efficacy of a support vector machine, and three convolutional neural networks (CNN) with increasing complexity and demonstrate their ability to perform a bee classification on the data provided.

In this project, we source a data set from a Driven Data Competition to identify Honey Bees and Bumble Bees given an image of these insects [1]. The classification goal is to predict for each image whether it is a bumble bee (Bombus) or a honey bee (Apis). The final aim is to build a model that is able to successfully identify honey bees and bumble bees using a photograph, even when those photographs have great variety in their backgrounds, positions, and image resolutions. With this goal, we will also aim to identify the best performance machine learning model to perform the classification.

The image data set is a crowdsourced image collection which includes bees in the natural environment. The images are intentionally challenging to classify. Some challenges are: the position and visibility of the bee varies as does its scale relative to the background; the images tend to be blurry, and difficult to see the bee body and/or wings, both of which are the primary features which could aid a classifier; lighting conditions are highly variable; and the background includes vibrant colors from the flowers, which are not unique pollination targets for either bee, but can be present in either. We use this data set and experiment with Support Vector machines with feature extraction techniques. We then explore various custom architectures of a convolutional neural network, including a standard 3-layer convolutional neural network which can classify the mnist data set with high accuracy, followed by a custom CNN architecture modeled after resnet 20, and finally we use transfer learning from the resnet 50 using the ImageNet[2][1] weights classifier to test out a more state-of-the-art technique.

II. Background and Motivation

The goal of the given Driven Data competition is to compare different machine learning approaches to the classification of bees as a honey bee or a bumble bee. These images are from the BeeSpotter[1] and have different behavior and appearances. However, given the wide range of backdrops, positions, and image resolutions, it can be difficult for machines to distinguish them.

The spread of colony collapse disorder has emphasized the importance of wild bees as pollinators. Currently, collecting data on wild bees requires a lot of time and effort. Bee Spotter makes this procedure easier by utilizing data provided by citizen scientists. Experts must still inspect and identify each image's bee. Being able to identify bee species from images is a challenge that will help researchers to collect field data more promptly and efficiently. Pollinating bees play an important role in ecology and agriculture,

[1] https://www.image-net.org/

and illnesses such as colony collapse disorder pose a threat to their survival.

In this project we build and demonstrate various machine learning techniques to classify bees effectively, thus helping the biologists examine the bee colony collapse disorders. The starting point is to determine the genus - Apis (honey bee) or Bombus(bumble bee) - using supervised learning with images and labels of the insects.

III. Related Work

Interest in the study of the Bee is not new and scientists have been researching in this field for a very long time. The advancement in technology, increase in computational power and everyday advance in computer vision and image classification techniques has provided scientists the opportunity to dive deep into the issues related to bee colony collapse and their illness. Bees are very important natural pollinators and play a crucial role in ecology and agriculture. There has been increasing research conducted in the various aspects of Bee images classification [3], [4], [5]. A simple search in the Google scholar for bee images classification for the past decade gave us approximately 17000 results. The majority of researchers used some form of convoluted neural network (CNN) to approach the problem. One of the researcher achieved an accuracy of 99 percent on wild bees using the google inception [5]

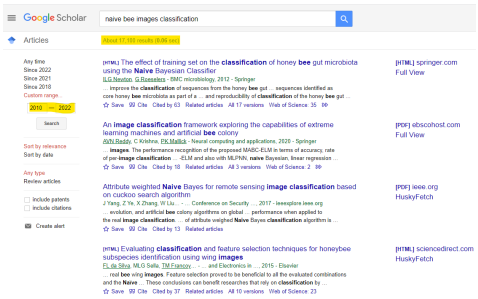

Figure 1: Google Scholar search result for Bee images classification

IV. Data

The data is a class-balanced set of images containing Apis (honey bee) and Bombus (bumble bee). All of the images in the data-set are cropped and scaled to 50x50x3 rgb color images. In this data, there are 1,654 bee images. Figure 2 shows an ideal high resolution cropping of the respective bee species, our image data set is much lower in resolution. The images vary in terms of distance from subject, position of the bee, and background. Figure 3 shows a set of images which are examples from the data set being used. As you can see, the backgrounds have extreme variation, and the bees are not cropped to be the primary focus of the image. Table I summarizes the data set.

Table I: Data Summary

Class	shape	Label
All Data	1654,50,50,3	[0,1]
Bombus	827,50,50,3	1
Apis	827,50,50,3	0

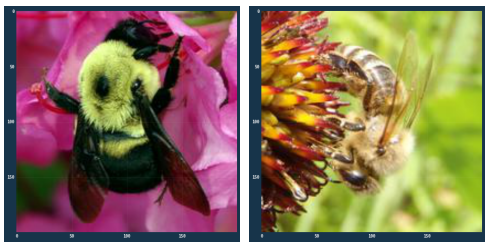

Figure 2: Example images of bombus (left) and apis (right)

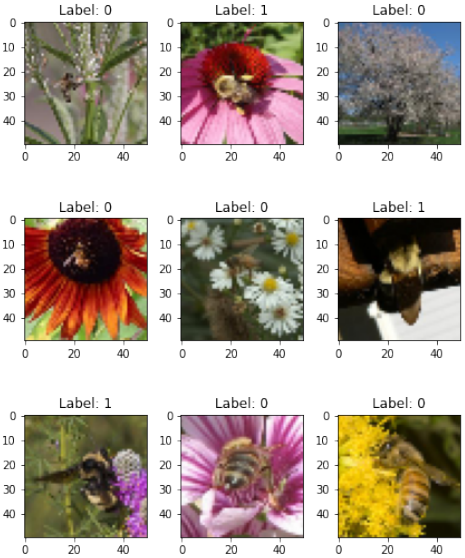

Figure 3: Example images of bombus (left) and apis (right)

V. Exploratory Data Analysis

The first step of the project is to load, transform and understand the images of the naive honey bees and naive bumble bees. This initial step, in a more technical term is called the exploratory data analysis (EDA) except for the loading part. In the EDA section, we have started from very scratch and have first generated a test image, just to show how an image looks like mathematically and how the computer understands them. Figure 4 shows the image generated from sample test data. In the next step, we performed some image augmentation tasks just to set

a context, as these techniques are going to come into use in the latter half of this project. Figure 5 shows the various image augmentation such as cropping, rotating, and flipping performed on a sample image. Next, we converted a sample image into an array of data just to explore its different color channels and to find out if removing any particular channel could be useful or not in analyzing the image. Figure 6 and 7 show the original images and the images with different selective color channels. To explore how the different color density varies in the images, we plotted the kernel density plot, which is shown in 8. To further observe the difference between the honey and bumble bee, we plotted their kernel density schema.

If we compare 9 and 10, we can see how different the color channels are for both the kind of bees. The honey bee on a blue flower has a strong peak on the right side of the blue channel. However, the bumble bee which has a lot of yellow (red + green) for the bee and the background, has a perfect overlap between the red and the green color channels.

Color channel information is very useful in most image classification problems, however, in our case, it is very distracting. Here, the bees themselves are very similar in color and also they are on top of the flower, and the flowers themselves are very vibrant in color. This gives us an intuition that the color of the flowers may be distracting from separating honey bees from the bumble bees. Hence we might need to convert the images into black and white for further use in our project.

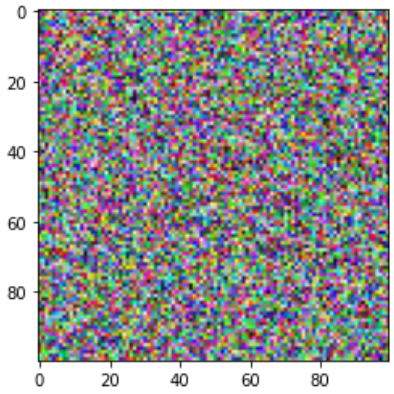

Figure 4: Image generated from sample test data

VI. Success Evaluation Criteria

We'll use our trained model to generate predictions for our test data. To see how well our model did, we'll calculate the accuracy by comparing our predicted labels for the test set with the true labels in the test set. Accuracy is defined as the number of correct predictions divided by the total number of predictions,

Figure 5: Sample augmented image

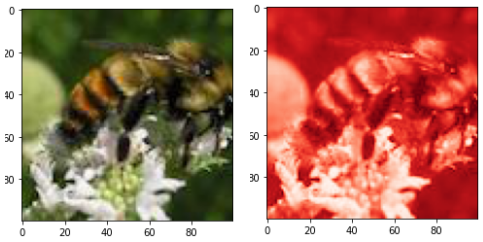

Figure 6: original sample image(left) and one with only rec channel(right)

which is applicable in a data set such as this with a balanced class. The baseline accuracy is 0.5 which is the performance of a model that randomly guesses.

VII. Pre-processing Procedures

Data is randomly shuffled and split 70/20/10 with a train, test, and validation hold-out set respectively for each experiment. To scale the data, typically we use min-max scaling with some exceptions. The CNN networks will be constructed using tensorflow. The network will undergo training with batched/shuffled tensor data over multiple epochs or until the test accuracy no longer improves over several epochs. Based on the size of the data-set, a batch size of 32 and 64 are be tested for the model fitting.

Image pre-processing will include Image manipulation with PIL, Image vectorization, scaling, and PCA. The Support Vector Machine (SVM) uses PCA and image vectorization to create features to make the learning feasible. HOG requires a normalized color value distribution to measure consistent accuracy,

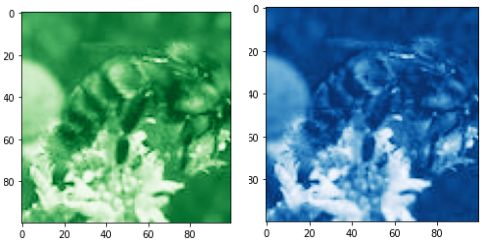

Figure 7: Image with only green channel(left) and blue channel(right)

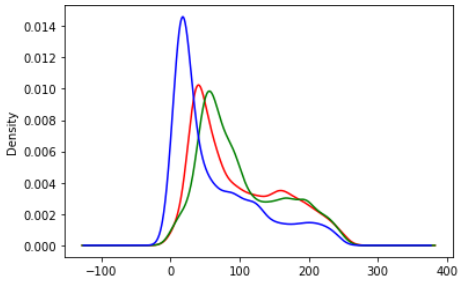

Figure 8: Kernel density plot of the sample image

and as such, data pre-processing for the model was performed on the standard non-augmented dataset by converting each of the images to grayscale.

The CNN architectures create features automatically through the convolutional kernels and pooling layers used.

For the Vanilla CNN and custom Resnet20 architectures described below, the image pixels are divided by 255 for min-max scaling of the images. In the the resnet50 experiment, the ImageNet[2] weighs are transferred. The preprocessing expected for this architecture use a built-in function where the images are converted from RGB to BGR, then each color channel is zero-centered with respect to the ImageNet dataset, without scaling.

Image Augmentation is also tested in the CNN architectures where the images are flipped, zoomed and cropped to increase the size of the database. With only 1,654 samples, this will step will attempt to provide more information for the model, reduce bias and over-fitting.

VIII. Experimental Design

We conduct three machine learning experiments for bee classification. 1) Using a support vector machine with image manipulation 2) Using a basic 3-layer CNN with pooling and drop-out 3) Using a deep 20-layer CNN with residual layers as described in [6], and, 4) Transfer learning with the ImageNet[2] weights using the Resnet50 architecture. The training and validation curves in each experiment and the best parameters for the respective models are recorded.

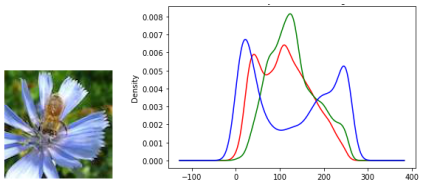

Figure 9: Honey bee (left) and its kernel density plot (right)

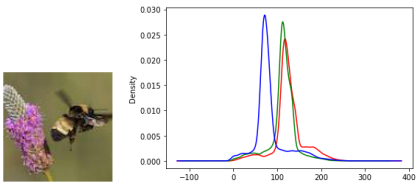

Figure 10: Bumble bee (left) and its kernel density plot (right)

The support vector machine will use cross-validation over a range of parameters, while the convolutional neural networks use optimization with back-propagation over several epochs, and are tested with various learning rates and optimizer functions to determine the best model. The learning curves of the best results are plotted and summarized to validate our recommendation of the best bee image classifier.

IX. Overview of Architectures

A. Support Vector Machine

A Support Vector Machine (SVM) method was built and utilized to attempt to identify the species of bee through image classification. For this portion of the experiment, the python Sklearn package SVM.SVC classification model was leveraged to train and test the model. For feature description image processing, a histogram of oriented gradients (HOG) object detection system was applied via the python Skimage library.

To perform object detection, HOG splits the image into cells of certain pixel ratios based on values found in gradient computation. Cells can be split into different dimensions; this project has utilized square dimensional cells. Gradient magnitude is measured between the cells to provide a vote weight; these weights can be visualized to the apparent object detection:

The Pixels-per-Cell (PPC) was found to be impacting to the performance of the SVM. As such, the PPC was parameterized to optimize accuracy. Additionally, two separate SVM kernels, radial basis function (RBF) and linear, were trained against the parameter set. The linear kernel SVM was able to obtain a maximum of 64.4 percent accuracy; the 8x8 PPC was the best selected parameter for this kernel. The RBF performed slightly better and with much less computational strain, resulting in a maximum of

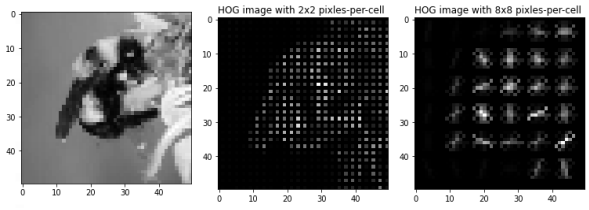

Figure 11: Visualized object detection of different sized cell grids within HOG.
Left: Grayscale sample — Center: Applied HOG with 2x2 Pixels-per-Cell — Right: Applied HOG with 2x2 Pixels-per-Cell

69.6 percent accuracy; the 10x10 PPC parameter performed the best. The linear kernel SVM ran the fastest at 10x10 pixels per cell approximately 20 minutes, and slowest at 2x2 pixels per cell, approximately 80 minutes. Run time increased linearly as pixels per cell decreased. For the RBF kernel, the SVM ran in about 2 minutes for each cell iteration.

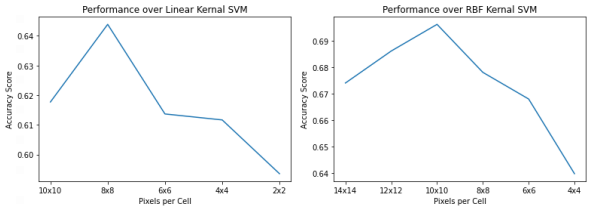

Figure 12: Model Summary of the SVM performance

B. Convolutional Neural Networks (CNN or convnets)

1) "Vanilla CNN": The first CNN model tested will be referred to as the "Vanilla" CNN. This is a baseline CNN architecture which is similar to the one used in the Keras API documentation which can successfully classify the MNIST data set [2]. The MNIST data has been used by many studies early on which demonstrate the efficacy of convnets on image recognition, including this detailed comparative analysis from 1998 [7].

This baseline architecture is built with a series of 3 convolutional layers with 32, 64, and 64 filters respectively with kernel size 2, and stride of 1. Max pooling is used to down sample the image, and dropout is used to prevent over-fitting. The model summary can be inferred from Figure 13. The best results were found with the Adam optimizer with a learning rate of 0.001. The dropout was increased to 0.5, above which the model could not train, therefore the issue of over-fitting was not overcome. Figure 14 shows that the testing loss reached a minimum value around 6 epochs, and over-fitting began around 16 epochs. The maximum testing accuracy achieved was 76%, and achieved 74% accuracy predicting the hold-out set. Training takes around 1 minute to complete. This simple method achieved an accuracy greater than random guessing which shows promise that the CNN network can automatically extract unique features of the the respective bee species.

[2]https://keras.io/examples/vision/mnist_convnet/

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_46 (Conv2D)           (None, 49, 49, 32)        416

max_pooling2d_29 (MaxPoolin  (None, 24, 24, 32)        0
g2D)

conv2d_47 (Conv2D)           (None, 23, 23, 64)        8256

max_pooling2d_30 (MaxPoolin  (None, 11, 11, 64)        0
g2D)

conv2d_48 (Conv2D)           (None, 10, 10, 64)        16448

flatten_14 (Flatten)         (None, 6400)              0

dropout_14 (Dropout)         (None, 6400)              0

dense_14 (Dense)             (None, 128)               819328

prediction (Dense)           (None, 1)                 129

=================================================================
Total params: 844,577
Trainable params: 844,577
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Model Summary of the vanilla CNN

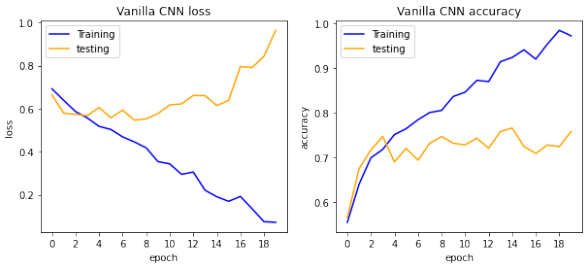

Figure 14: Training and validation results of the vanilla CNN

2) "Pseudo Resnet20": The next custom architecture referred to here as the 'pseudo Resnet20', was built with the Resnet 20 architecture design criteria described in the following article [6] where the authors find that deep networks can be built with shortcut connections that reduce the exploding/vanishing gradient problem and prevent over-fitting. The shortcut is created simply by identity mapping the output from previous layers (x) and then doing element-wise addition to the deeper layer output (F) F + x followed by the non-linearity, typically Relu activation. This is what the authors refer to as the "Residual Function". Furthermore, the authors achieved superior classification accuracy with

data sets such as CFAR10 than previous architectures lacking the shortcuts.

The model summary is too cumbersome to copy here, however the model architecture can be summarized as follows: An initial convolutional layer 3x3 kernel with 16 filters, followed by max pooling 2x2. The following layers are blocks which contain 3 sets of 2 convolutional layers, with intermittent identity mapping layers in-between sets. Each block has a consistent filter size, and contain 3 identity mappings, each mapping is added to the output of the following two layers. Each convolutional layer is batch-normalized prior to relu activation. There are three blocks with filter sizes (16, 32, 64) each having 2x2 kernels. After the convolutional blocks, global max pooling is performed followed by flattening and then the output binary classification. In summary there is 1 convolution/max pool + 6 convolutional layers (block 1) + 6 convolutional layers (block 2) + 6 convolutional layers (block 3) + global average pooling + 1 output dense layer. This attempts to replicate the Resnet20 architecture based on the description in the paper.

Many configurations were tested including the Adam, Adamax, and SGD optimizers, with learning rates from 1e-1 to 1e-3, and learning rate decay schedules. The sets batch sizes were 16, 32, and 64, and 128. The best result was from the Adam optimizer with a constant learning rate of 1e-3, and a batch size of 32. Regardless of the outcome, the test accuracy ended significantly lower that the training indicating over-fit and unable to generalize. Training would fit the data reaching up to 100% accuracy after 25 epochs, and testing would level out at 60% accuracy. The validation hold-out set was tested after each experiment calculating an accuracy almost identical to the final test result. The best accuracy achieved is plotted in Figure 15. Training 100 epochs takes 1 hour to complete with the bee data set.

Another version of the Resnet20 architecture is tested provided by a different member in the team. This architecture was inspired by an interpretation used in a public Github repository[3]. This produced a training accuracy of 99 percent and testing accuracy of 78.17 percent. The accuracy vs epoch plot and loss vs epoch plot is shown in the figure 16. This provides a more successful implementation of the architecture described in [6].

3) Resnet50 transfer learning: The Resnet50 architecture with transfer learning of the ImageNet[2] weights are used. The Resnet50 arhitecture uses the same configuration as Resnet20, but with 50 convolutional layers. The Resnet50 architecture outperforms Resnet20 in image classification tasks according to [6]. Because the actual Resnet50 model requires an

[3]https://github.com/yasharvindsingh

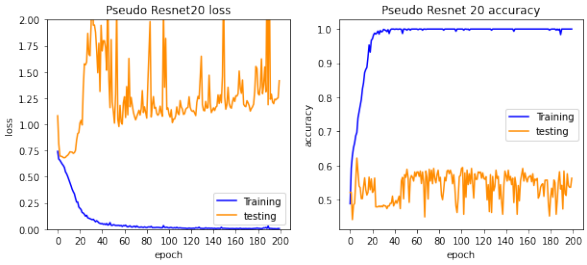

Figure 15: Training and validation results of the 'pseudo Resnet20' CNN

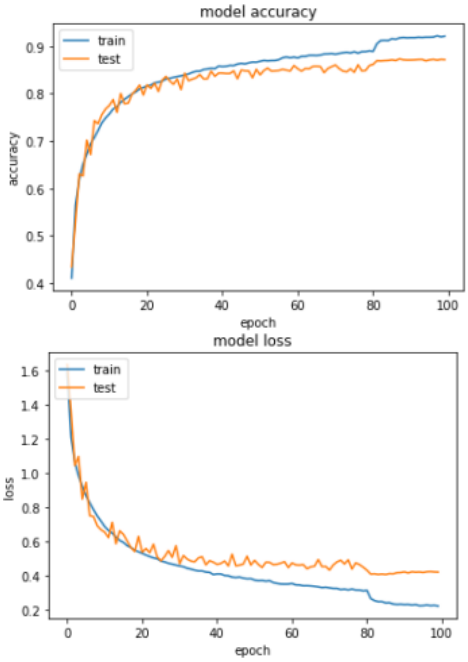

Figure 16: Accuracy(top) and loss(bottom) plots testing another version of ResNet20 architecture with better results

input shape of at least 200x200, an Upsampling layer (5x5) is added to the input which simply copies each pixel in the bee image from 1x1 to 5x5, enabling the Resnet50 to perform the ImageNet[2] weighted convolutional kernels. Additionaly, the default preprocessing function from the Keras Resnet50 applications was applied to raw RGB bee images to ensure consistent input channel configuration and scaling as intended. In the the resnet50 preprocessing, the images are converted from RGB to BGR, then each color channel is zero-centered with respect to the ImageNet[2] dataset, without scaling. The only layer that is trained in this model is the final classification layer.

The results of this experiment show that the Resnet50 architecture can reach close to 90% accuracy, 89% on the holdout, demonstrating the CNN can automatically capture different features born from the bees despite their insignificant size and shape

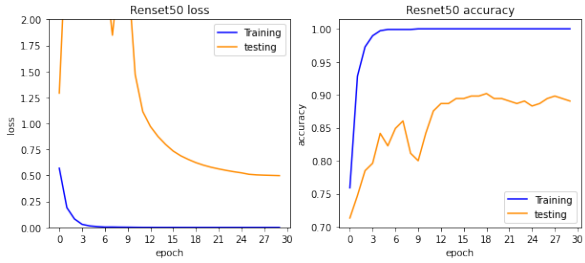

Figure 17: Training and validation results of the Resnet50 CNN

compared to the vibrant, varying backgrounds. The following Figure 17 shows that Resnet50 was able to learn the training after about 6 epochs, and testing reached the highest accuracy around 18 epochs. One observation of this experiment which was different from all others is that validation accuracy could improve even after training reached a minimum loss. This may demonstrate the efficacy in the residual function to prevent a vanishing gradient. Also, training 100 epochs only takes only 20 minutes because the upper layer weights are set to be un-trainable.

4) CNN Using Image Augmentation: The Resnet50 model reached nearly 90% accuracy, however image augmentation is explored to improve that. A worthwhile consideration is the colors in the image do not contribute to the classifier because the backgrounds are very vibrant, and the bee species themselves are similar in color. An addition, basic augmentation such as cropping and flipping has been used successfully to improve image classifiers [8].

Figure 18 shows a random sample of black and white transformed bee images, mixed with flipped and cropped black and white images. Images were transformed using PIL [4]. This doubled the size of the data to 3308 which was then shuffled and split into a training, testing, and validation hold-out set.

The vanilla CNN was tested on the the black and withe plus augmented data set. Figure 19 shows the training data quickly becoming overfit. Dropout was increased by adding 0.5 dropout after the convolutional layers, and 0.5 dropout after the first dense layer. Batch normalization was added, and learning rates and optimizers were adjusted. More epochs were run which resulted in 68% accuracy on testing, and 65% on the hold-out which demonstrates no improvement on generalizing ability of the model.

In Figure 20, the Resnet50 architecture is tested with the augmented data set containing 3308 bee images. The accuracy was not significantly increased so an additional fully connected network was added with two convolutions (dense 128, and dense 64) and 0.25 dropout. No significant change was observed.

[4] https://pillow.readthedocs.io/en/stable/index.html

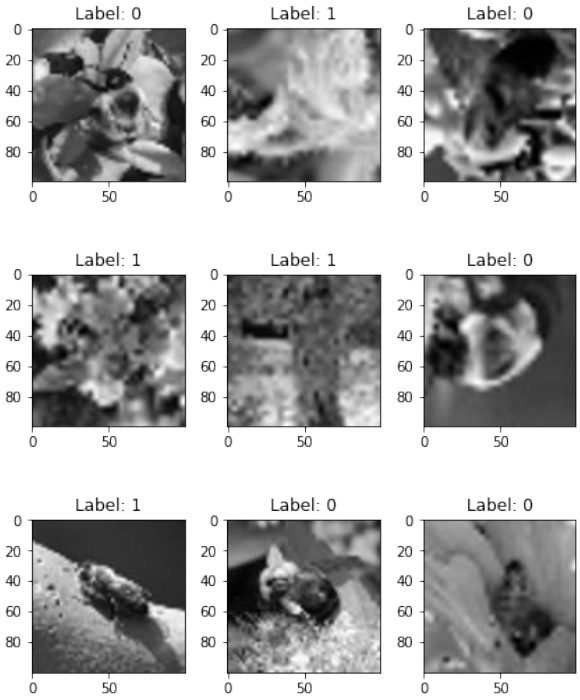

Figure 18: Random sample of augmented bee images

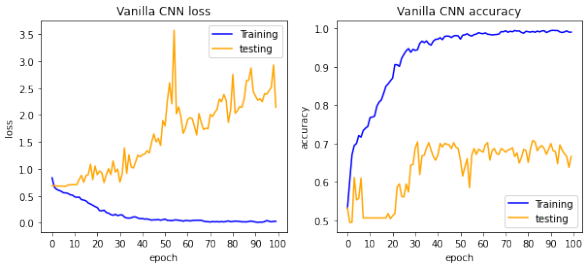

Figure 19: Vanilla CNN training and validation results on the augmented data set

The accuracy of the test and hold-out sets were both 87% indicating that augmentation did not improve the model.

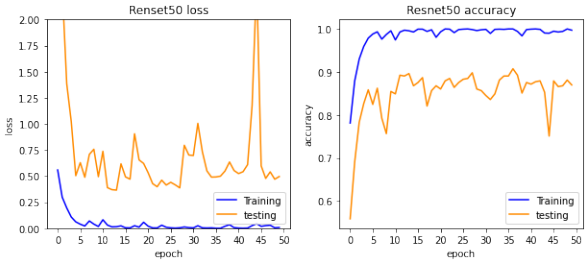

Figure 20: Resnet50 CNN training and validation results on the augmented data set

Table II: Model performance

Model	Data	Accuracy	Train Time
SVM	HOG	69.6	80 min
Vanilla CNN	Original	72.	1 min
Vanilla CNN	Augment BW	65.	1 min
Resnet20	Original	78.17	180 min
Resnet50 Transfer	Original	89.	5 min
Resnet50 Transfer	Augment BW	87.	5 min

XI. Conclusion

Performing these experiments shows that a convolutional neural network is better suited to bee image classification than the Support Vector Machine. Furthermore, CNNs do not require the extra feature extraction step as the SVM which increased the complexity of that model. The vanilla CNN experiment reached greater accuracy than the SVM without any preprocessing other than min-max scaling of the bee images, and completed in about 1 minute using standard free online GPU. The SVM required about 20-80 minutes to train and validate the best pixel parameter. The vanilla CNN, however, was also not able to overcome over-fitting, therefore it is unlikely that a more complex model with this architecture would yield better results.The Resnet20 model demonstrated an increase in accuracy over the Vanilla CNN at 80% and took about 3 hours to train. The Vanilla CNN would quickly become over-fit and the only solution, dropout, if too high would prevent training at all. Adding layers to the Vanilla CNN would only increase over-fitting. This demonstrates the efficacy of the residual blocks in preventing over-fitting and enabling deep networks to train. The transfer learning approach was the highest accuracy classifier reaching about 90%. The model was trained for 100 epochs and took 20 minutes, however, Figure 17 shows that best accuracy was achieved around 25 epochs therefore it requires 5 minutes to train on this data set. This result confirms the importance and overall versatility of transfer learning in the field of image classification and computer vision.

X. Results

The results of the experiments are summarized in Table II. Using the Resnet50 architecture with transfer leaning of the ImageNet weights yielded the highest accuracy classifier for the bee image data set used in this study.

XII. Contributions

This project was completed with equal contribution by the three authors. Tony Geglio focused on experimental design, CNN architectures, and image augmentation, with contributions to related works and introduction. Nehal also contributing to testing the CNN architectures did the background, EDA, and literature review. Alex explored image vectorization, image manipulation with PIL, and the Support Vector Machine classifier.

References

[1] DrivenData. Naive bees classifier. [Online]. Available: https://www.drivendata.org/competitions/8/naive-bees-classifier/page/31/

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[3] D. De Nart, C. Costa, G. Di Prisco, and E. Carpana, "Image recognition using convolutional neural networks for classification of honey bee subspecies," vol. 53, no. 1, p. 5. [Online]. Available: https://doi.org/10.1007/s13592-022-00918-5

[4] W. Kelley, I. Valova, D. Bell, O. Ameh, and J. Bader, "Honey sources: neural network approach to bee species classification," vol. 192, pp. 650–657. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921015544

[5] V. LaVezzi, D. Robert *et al.*, "Bombus species image classification," *Computer Vision and Pattern Recognition*, vol. 11, pp. 1–8, 2020.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," pp. 2278–2324, 1998.

[8] J. Wang, L. Perez *et al.*, "The effectiveness of data augmentation in image classification using deep learning," *Convolutional Neural Networks Vis. Recognit*, vol. 11, pp. 1–8, 2017.