Analysis of Honey Bee Images using Keras

Submitted by Rohit Vincent(18200396)

Dataset chosen based on first announcement

About the Dataset

The dataset chosen for the particular analysis is The BeeImage Dataset: Annotated Honey Bee images. The dataset contains a csv file along with a folder with bee images. The fields in the csv file are described below:

field	file	date	time	location	zip code	subspecies	health	pollen_carrying	caste
description	File name in bee_imgs folder	Date of video captures	Time of day of video capture (military time)	Location (city, state, country)	Zip Code to numerically describe location	Subspecies of Apis mellifera species	Health of a bee	Presence of pollen on the bee's legs	Worker, Drone, or Queen bee

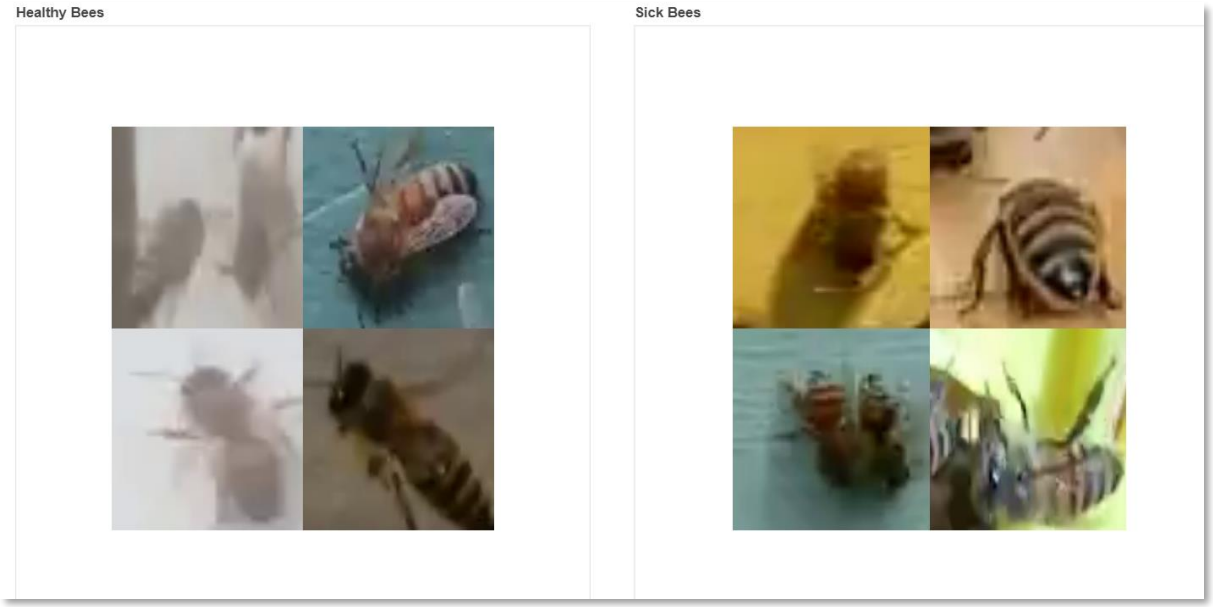
Problem Formulation

Honey bees are very important species and are main pollinators in the environment. Our client is a company which runs Honey bees farms around the world. But regular check-ups of honey bees require man-power and are time-consuming. Moreover check up of these hives causes disturbance in the bees lifestyle. Our client needs a more innovative solution to efficiently manage these Bee hives.

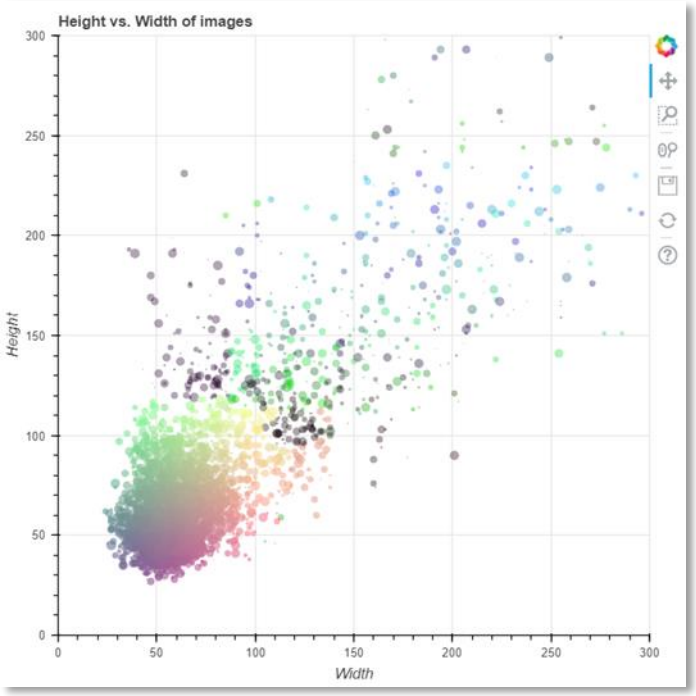
To achieve this we have collected over 5100 bee images from still time-lapse videos of bees. Each image is accompanied with the details provided in the csv like health, subspecies, datetime etc. Our solution is to analyse these images and come up with an interesting model so the client can install cameras in all of their bee hives to monitor the bees automatically and efficiently.

Analysis & Model Generation

Following are some of the sick & healthy bees from the dataset:



Initially we explore some of the metrics provided in the dataset and see what we can do with it.



The following scatter plot using bokeh shows the size of the images provided. We can see most of the images are of size (100,100) whereas some are larger than that. To use all the images we will rescale these images to 100,100 using the image Generator. This size is set as a global variable in the program as width_image & height_image.

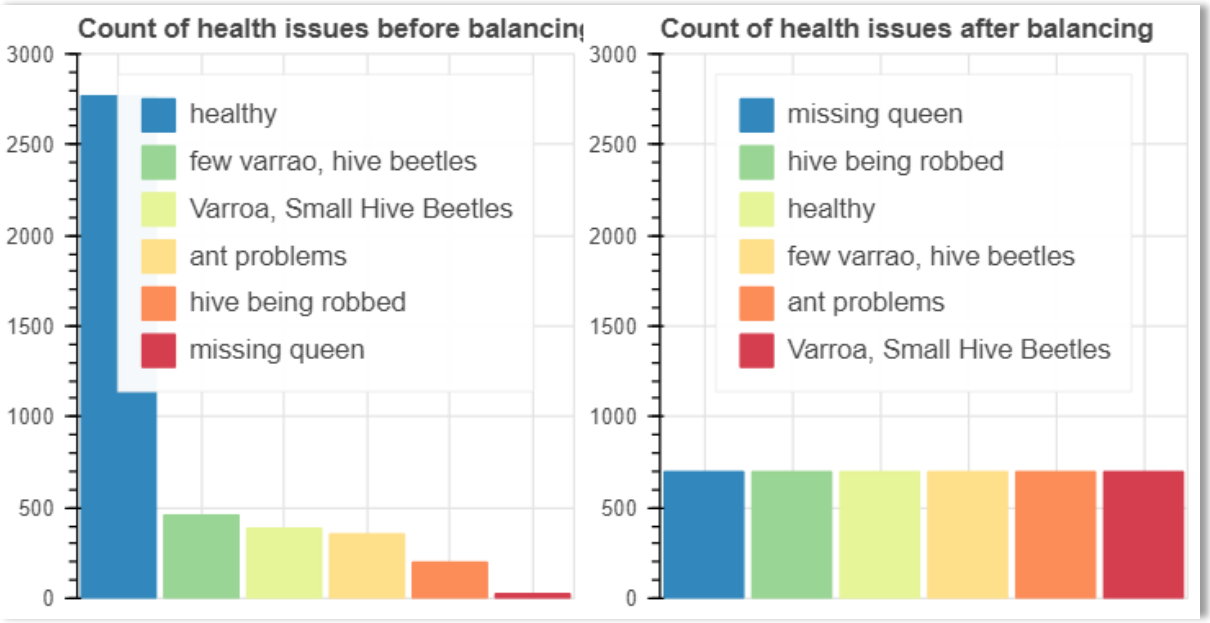
We decided to create a model to predict the health of a bee using Keras based on an image. This will allow our client to detect if there are unhealthy bees in the hive and if there is anything wrong happening over there. For this we take the dataset and do the following preprocessing(Function preprocess() in honeybee.py):

1. Remove bee details where there are missing values.
2. Remove bee details where we do not have an image file.

We then split the data into training, validation & test sets where the size of validation & test is 10% each of the whole dataset.(size is a global parameter and can be adjusted).

Balancing the training data

We noticed that the data is imbalanced for each health issue due to there being more bees which are healthy and the divergence of problems are not balanced. In order for the model to not overfit any of the issues we balance the training set by setting the training set into equal number of entries for each health issue. This is done using function balance(). Following bar graphs show the count of entries for each health issue before & after balancing.



We will be creating a Convolutional Neural Network to classify a image to the any of the above categories. For this we need to assign a numeric representation of each of the categories. The following code using panda function get_dummies to do this.

```
train_y = pd.get_dummies(train_bees['health'])
validate_y = pd.get_dummies(validate_bees['health'])
test_y = pd.get_dummies(test_bees['health'])
```

We use the ImageDataGenerator from Keras to do some random operation on the training dataset. These include random operations of rotating by 0-180 degrees, zooming by 10%, shifting horizontally & vertically by 20% & horizontal & vertical flipping.

```
imgDG = ImageDataGenerator(
    rotation_range=180,
    zoom_range=0.1,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True)
```

Neural Network Model

Our CNN model consists of the following Layers:

1. A convolutional layer using 16 filters of dimension 3. It uses a rectified linear unit activation function.
2. A maxpoll2d layer using a reduction factor of 2.
3. A convolutional layer using 16 filters of dimension 3 with the Relu activation function.
4. A flatten layer.
5. A dense layer to change to label size using an activation of softmax.

This model is then compiled using a adam optimizer which is based on stochastic gradient descent algorithm and is used to reduce the categorical_crossentropy loss for this case.

```
model=Sequential()
model.add(Conv2D(16, kernel_size=3, input_shape=(width_img, height_img,3), activation='relu'
model.add(MaxPool2D(2))
model.add(Conv2D(16, kernel_size=3, activation='relu', padding='same'))
model.add(Flatten())
model.add(Dense(train_y.columns.size, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

We use the Keras fit_generator with a batch size of 60(global parameter batch_size which can be adjusted) for 50(parameterized using epoch_no) epochs with steps_per_epoch equal to the size of the training set divided by the batch size.

```
steps = np.round(train_X.shape[0]/batch_size,0)
# Train the model for the training set
history = model.fit_generator(imgDG.flow(train_X,train_y, batch_size=batch_size)
                              ,epochs=epoch_no
                              ,validation_data=[validate_X, validate_y]
                              ,steps_per_epoch=steps
                              ,callbacks=[early_stop, save_best])
```

early_stop & save_best are callback functions for keras to perform optimization of the model created.

Earlystopping is a inbuilt callback function is keras which is used to stop training in case epochs do not give any better value during further iterations. In this model we check if the accuracy of the validation set improves for 20 epochs before stopping training. This allows unnecessary training epochs and improves performance.

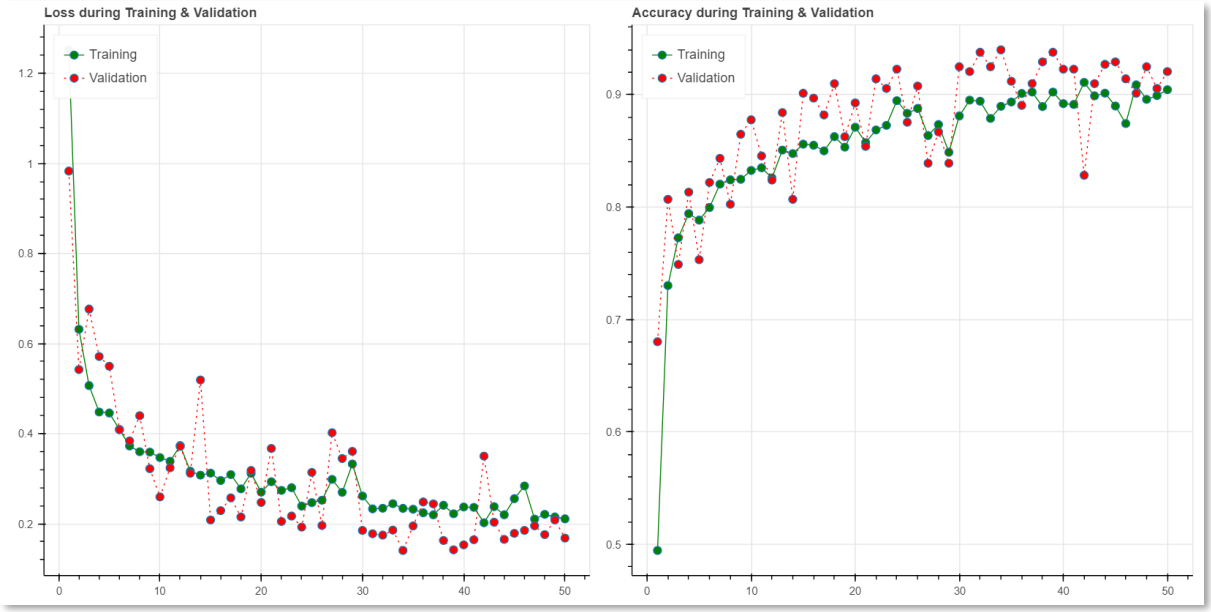
```
early_stop = EarlyStopping(monitor='val_acc', patience=20, verbose=print_flag)
```

ModelCheckpoint is another function in keras which is used to save the weights of the model during each epoch only if it is the best. This function is called at every epoch and saves the model only if it is better than the previous model.

```
save_best = ModelCheckpoint('healthy_model'
                            ,monitor='val_acc'
                            ,verbose=print_flag
                            ,save_best_only=True
                            ,save_weights_only=True)
```

Results of initial model

Following graph shows the changes in Loss & Accuracy in the Training Set & Validation Set for each epoch.



We can see that with every epoch the loss is reduced using the adam optimiser & the accuracy of the model increases. Evaluate the model with the test set & check the accuracy. Following are the results & metrics of the model (divided for each subspecies). An accuracy of 93.6% was obtained for the model.

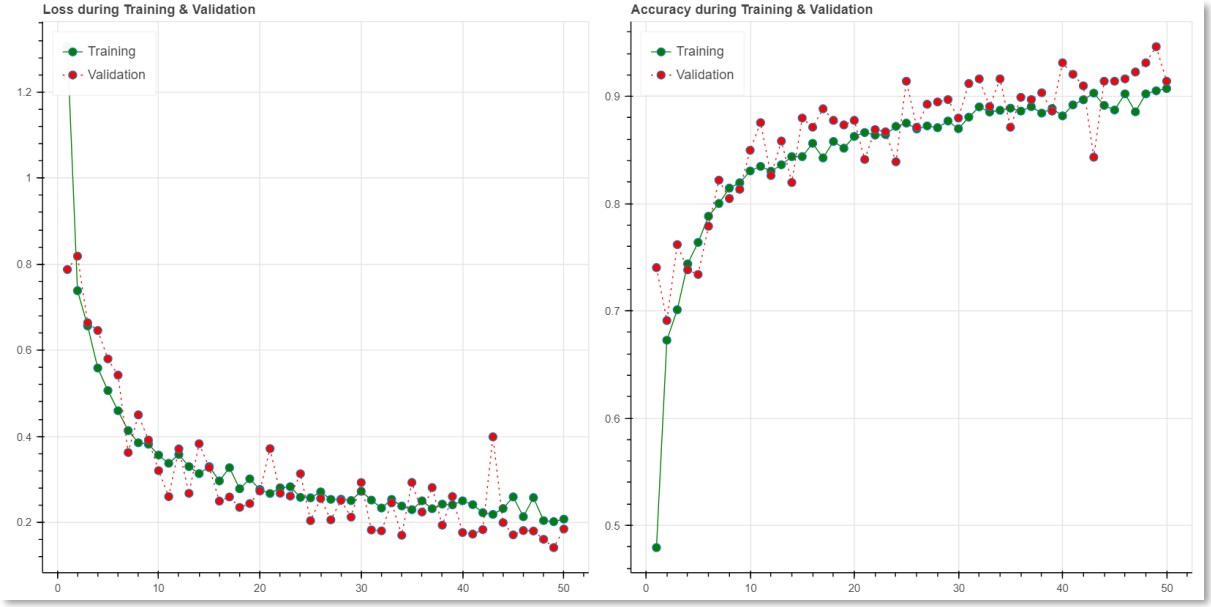
```
Model was created with Loss of 0.1450532457384939 with Accuracy: 0.9362934362934363
                            precision    recall  f1-score   support

Varroa, Small Hive Beetles       0.67      0.72      0.69        36
              ant problems       0.98      1.00      0.99        52
  few varrao, hive beetles       0.80      0.74      0.77        61
                   healthy       0.99      0.98      0.99       332
         hive being robbed       0.86      0.97      0.91        33
             missing queen       1.00      1.00      1.00         4

               avg / total       0.94      0.94      0.94       518
```

Reduce Overfitting

We can reduce overfitting by adding dropouts to the above model. Two Dropout layers were added which drops 20% of the node connections. This reduces the likelihood that the network will overfit the training data. The new model is shown below:

```
model=Sequential()
model.add(Conv2D(16, kernel_size=3, input_shape=(width_img, height_img,3), activation='relu',
model.add(MaxPool2D(2))
# Add dropouts to the model
model.add(Dropout(0.2))
model.add(Conv2D(16, kernel_size=3, activation='relu', padding='same'))
# Add dropouts to the model
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(train_y.columns.size, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

The loss & accuracy for the new model is shown below:

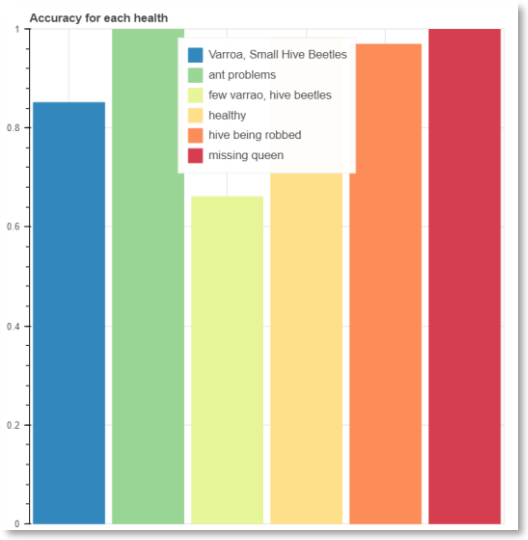


The new model was evaluated and returned an accuracy of 94.01% which is better than the previous model. The metrics for this final model are shown below:

```
Model was created with Loss of 0.1449333887563436 with Accuracy: 0.9401544401544402
                          precision    recall  f1-score   support

Varroa, Small Hive Beetles       0.70      0.78      0.74        36
              ant problems       0.96      1.00      0.98        52
  few varrao, hive beetles       0.83      0.72      0.77        61
                   healthy       0.99      0.98      0.99       332
         hive being robbed       0.89      0.97      0.93        33
             missing queen       1.00      1.00      1.00         4

               avg / total       0.94      0.94      0.94       518
```

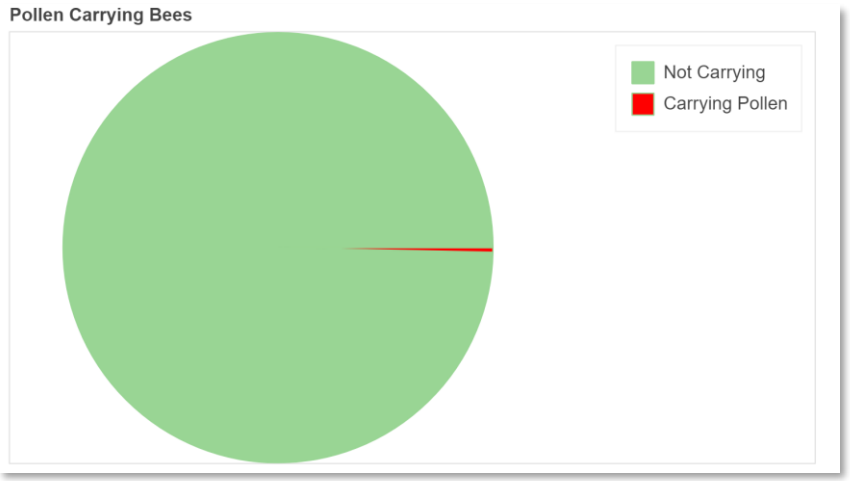
Solution

The created model can be used by the client to identify issues with health of bee hives and take necessary action. The model has an accuracy of 94.01% with accuracy for each health types shown in the bar plot.



Future Improvements

1. A model could be created to classify which subspecies a bee belongs to see cross-interaction & migration of bees across hives.
2. A model could be created to see if a bee is carrying a pollen or not. The below pie chart shows the distribution of bees which carry pollen and those which does not. For the current dataset, the values are very biased towards bees not carrying any pollen and cannot be used to train a model. In the future when we have a more entries for bees with pollen this could be done.



3. We can tweak the hyperparameters to add more filters in the convolutional layers or by experimenting with different epoch sizes.