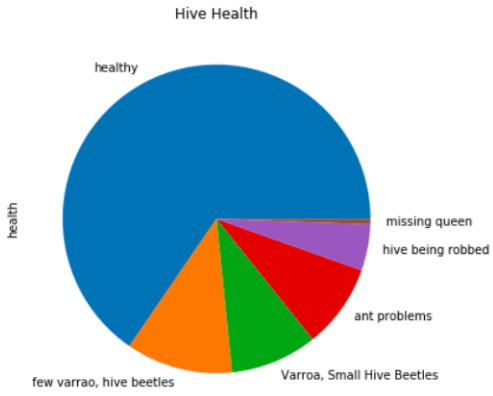
Individual Final Report
Xiaochi Ge

This project goal is to use deep learning network to classify Bees Subspecies and Hive Health. The dataset contains 5,172 bee images annotated with location, date, time, subspecies, hive health condition, caste, and pollen carrying. Ruyue Zhang did Keras subspecies classification, I did Keras hive health classification, Yijia Chen did Pytorch subspecies classification, and Phoebe Wu did Pytorch classification. Everyone in our group did parts of EDA.

My work is focus on Keras Hive Health Classification.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

bees = pd.read_csv('../input/bee_data.csv')
'''
plt.figure(figsize=(6,6))
bees.health.value_counts().plot(kind = 'bar')
plt.title('Hive Health')
plt.show()
'''
unhealthy = bees.loc[bees['health'] != 'healthy']
#unhealthy
unhealthy.count()

healthy = bees.loc[bees['health'] == 'healthy']
healthy.count()

bees.health.value_counts()

New = bees.replace(['hive being robbed', 'few varrao, hive beetles', 'Varroa, Small Hive Beetles', 'ant problems', 'missing queen'],
            'unhealthy')

New.head()
```

The first thing I did is to categorize the “health” column. The original distribution of “health” column look like this:



Therefore, I put all other categories together, except “healthy”, and name the new group “unhealthy”.

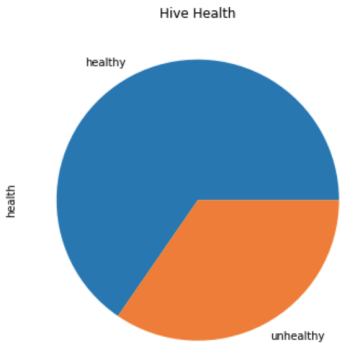


Figure 4

Basically, I use CNN model, and I try with different layers, different activation function, and different optimizer, add normalization, and various batch size and epochs.

```
#model architecture
def cnn():
    model = Sequential()
    model.add(
        Conv2D(input_shape=(train_img.shape[1], train_img.shape[2], train_img.shape[3]), filters=50, kernel_size=(3, 3),
               strides=(1, 1), padding='same', kernel_initializer='he_normal'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(filters=50, kernel_size=(3, 3), strides=(1, 1), padding='same'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=50, kernel_size=(3, 3), strides=(1, 1), padding='same'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    # dense layer with 50 neurons
    model.add(Dense(50, activation='relu'))
    model.add(Dense(7, activation='softmax'))

    adam = optimizers.Adam(lr=0.001)
```

Furthermore, I play with two different training models: fit( ) and fit_generator( ). In Keras, fit( ) can pass the whole dataset at once, therefore, it is good for smaller datasets; while fit_generator( ) could avoid duplicates in data, consequently, it is good for larger datasets. Additionally, generator train the model batch by batch, and it can help to augment images.

```
#fit trainning data model
training1 = model.fit(train_img, y_train, batch_size=64, validation_split=0.2, epochs=30, verbose=1)

training2 = model.fit_generator(generator.flow(train_img, y_train, batch_size=32)
                                , epochs=30, verbose=1)
#print(training2.history.keys())

#accuracy = training1.history['acc']
#loss = training1.history['loss']
```

After trying various experiments, I find that “Adam” optimizer, without normalization, would produce a satisfied accuracy. However, different layers, batch size, number of

epochs do not change a lot. Different training model does not change a lot as well in this case.

- **With Normalization (Epochs = 30)**

	Training 1(Test Accuracy)	Training 2(Test Accuracy)
Batch Size = 32	**0.631**	**0.631**
Batch Size = 64	**0.631**	**0.631**

- **Without Normalization (Epochs = 30)**

	Training 1(Test Accuracy)	Training 2(Test Accuracy)
Batch Size = 32	**0.995**	**0.992**
Batch Size = 64	**0.989**	**0.987**

Conclusion:

After several experiments with normalization, various batch sizes, and different training sets, we find out that, without normalization, we can find a satisfied accuracy result. There are no much distinctions in accuracy when we change the batch size from 32 to 64.

Normalization is not always a good thing. I was stuck by low accuracy and abnormal loss for several days, and also try different layers, optimizers, etc., but the accuracy and loss still cannot improve. After I quit the normalization, the accuracy improves a lot.

The other issue I had is type error, and type error will produce incorrect data.

Percentage of copy code: 34/166 = 20.48%

References

Preda, Gabriel. “Honey Bee Subspecies Classification.” *RSNA Pneumonia Detection Challenge | Kaggle*, 2018, www.kaggle.com/gpreda/honey-bee-subspecies-classification.

Pukhov, Dmitry. “Honey Bee Health Detection with CNN.” *RSNA Pneumonia Detection Challenge | Kaggle*, www.kaggle.com/dmitrypukhov/honey-bee-health-detection-with-cnn.

MAITYH, AYAN. “Honey Bee Image Classification.” *RSNA Pneumonia Detection Challenge | Kaggle*, 2018, www.kaggle.com/ayanmaity/honey-bee-image-classification.

“Image Preprocessing.” *Keras Documentation*, keras.io/preprocessing/image/.

“Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python.” *EliteDataScience*, 8 Feb. 2018, elitedatascience.com/keras-tutorial-deep-learning-in-python.

“Model Class API.” *Keras Documentation*, keras.io/models/model/.