# HoneyCheck: A Sweet Approach to Bee Health Classification

## 1 Introduction

Although bees are tiny, they have a very significant role in the Ecosystem – they are a keystone species. Bees are responsible for pollinating about a third of the agricultural crops. So, you can imagine what would happen if the population of bees drops drastically – many species could go extinct [2,3]. Beekeeping is a good way to maintain and even increase the bee population. Regularly checking up on the bees and hives could help in letting the bee population thrive [1]. The objective of this project is to classify a bee as healthy or not based on its image.

Section 2 discusses the formulation of the Machine Learning problem along with a description of the dataset. Section 3 describes the preprocessing done to prepare the data. Section 4 presents the ML models used along with their results and finally, section 5 provides a conclusion along with limitations and further enhancements.

## 2 Problem Formulation

This project will try to predict whether a bee is healthy or not, by looking at a picture of the bee and analysing its features. Each datapoint in the dataset is a combination of the pixel representation of the bee's image, and its classification label – 0 or 1 (unhealthy or healthy bee, respectively). This makes it a supervised binary classification problem.

The dataset, called "The BeeImage Data: Annotated Honey Bee Images", was created by Jenny Yang and the 2nd version of the dataset was posted to Kaggle in 2018. The dataset was created by extracting images of bees from time-lapse videos (submitted by other people) of bees around their respective beehives. [1]

The dataset contains columns like date, time, location and zip code of where the bee was found, subspecies, whether it was carrying pollen, the caste of the bee (Worker, Drone or Queen), an image of the bee and finally a classification of whether it is healthy or has one of the 5 unhealthy categories (Few Varroa - Hive Beetles, Varroa - Small Hive Beetles, Ant Problems, Hive being Robbed, Missing Queen). There are a total of 5172 entries in the dataset, out of which 3384 entries are of the healthy bees.

For each image, the there is a 64x64 size array for each of the 3 colours (RGB), i.e., each image datapoint contains 64x64x3 attributes. Each of these attributes have an integer value between 0 and 255, representing the intensity of a colour in that pixel. 0 corresponds to zero intensity of the colour, and 255 corresponds to full intensity. So, for a particular point on the image, it is made up of some combination of intensities of red, green, and blue pixels.

## 3 Data Preprocessing & Splitting

Several steps were taken to clean and preprocess the data from the original dataset, they are described in this section. The date, time, location, and zip code columns are dropped, as they are not important for the image classification model. The columns on subspecies, whether the bee was carrying pollen, and the caste were dropped as well. These features could be important in differentiating features of Bee Species A from B. If the dataset is mostly Species A, then it is possible that the model predicts species B as unhealthy. But since these features are a little complicated to include in the image classification problem, they will be skipped for the scope of this project.

There are 3384 healthy samples and a comparatively small number of samples for each of the multiple unhealthy classes (Table 1). It is an unbalanced dataset for an image classification task. Reducing the healthy samples to approximately 500 and keeping it a multiclass classification problem, felt like there was not enough data to train the model properly. Hence, it was converted into a binary classification problem (Refer Table 2) and the number of healthy samples was reduced to 1788. The final data set is balanced and has 3576 datapoints. My hypothesis is that the model will be able to learn the features of a healthy bee and would be able to distinguish between healthy and unhealthy bees.

| Health Classification | # of Samples |
|---|---|
| Healthy | 3384 |
| Few Varroa, Hive Beetles | 579 |
| Varroa, Small Hive Beetles | 472 |
| Ant Problems | 457 |
| Hive Being Robbed | 251 |
| Missing Queen | 29 |

*Table 1 Original Distribution of samples in each class*

| Health Classification | # of Samples |
|---|---|
| Healthy | 3384 |
| Unhealthy | 1788 |

*Table 2 Distribution of samples in 2 Classes after changing labels.*

The pixel matrix has the representation of all 3 colours, which leads to a more complex dataset, and would require more storage space and computing power. The Grayscale of these values are taken, so that the number of attributes for each datapoint decreases, and training can be faster. Grayscale represents the pixel value between white and black (from 0 to 255). Further, each of the attributes in the pixel matrix have been normalized (divided by 255), resulting in continuous attribute values between 0 and 1. These normalized matrices are used as inputs to the Convolutional Neural Network Model.

For the k-Nearest Neighbours model, it is required that each datapoint is in the form of a 1-dimensional vector. This would result in a 2-dimensional matrix for the entire dataset. This is known as horizontal stacking (Figure 1). The final dimensions of the matrix representing the full dataset are 3576x4096 (number of datapoints x number of pixels for each sample).

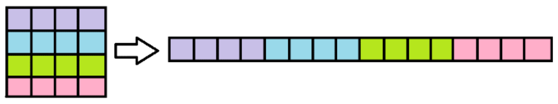


*Figure 1 Horizontal Stacking 2D -> 1D, for each image*

For the data I will use an 70-20-10 split to train, validate and test the model. I have picked this split as I have a balanced dataset, and so could skip over the methods that that are recommended for training models of imbalanced datasets. The data has been randomly sampled, so that all three sets individually represent the distribution of the entire dataset. The training set has 2503 samples, the validation set has 715 samples and the testing set has 358 samples. This kind of split was chosen to maximize the size of the training set.

## 4 Methods

### 4.1 Model Selection

There were two models picked to perform this binary image classification task – k - Nearest Neighbours (kNN) and Convolutional Neural Networks.

The kNN algorithm is a simple and relatively intuitive Supervised Machine Learning Model that is based on the idea that samples near each other, tend to be similar. Since it gives the class of a datapoint based on the other datapoints around it, it seems like a good algorithm for the image classification problem. The training data is plotted in space, and when a new datapoint comes in, it's k nearest neighbours are found and a class is assigned based on the majority class of the neighbours. The nearest neighbours are found using the Euclidean distance between the new point and the rest of the training data [4]. The algorithm should be able to separate the datapoints in space based on their category, allowing for easy classification of test datapoints, which is why this model was picked.

Image data is usually of high dimension. For a model, to be able to learn these many features require a lot of parameter training [7]. Convolutional Neural Networks are usually used for Image related tasks for a couple of reasons. The first is that CNNs are easily able to reduce the dimensions of the data without losing any relevant information [6,7]. They are also able to extract features from the images using edge detection and are good at recognizing patterns across images as well [6]. These pros of the CNN model made it an appealing choice for this image classification task.

## 4.2 Loss Function

The loss function used for the kNN model is 0/1 loss. This loss function seems like an intuitive choice for a classifier. It has a value of 0 if the prediction is made correctly, and a value of 1 if the classification is made incorrectly [4]. Since the k-NN algorithm doesn't have a loss function that it tries to minimize during training, the 0/1 loss seems like a good function to evaluate the performance of the model [5].

For the CNN Model, Binary Cross Entropy loss function is used. For a binary classification problem, for each class, a probability is calculated, probability that the datapoint belongs to that class. We want to maximize the probability calculated for the correct and correspondingly minimize the probability for the incorrect class [10]. If a correct class was predicted with a higher probability, lesser the loss is. Correspondingly, if an incorrect class was predicted with a higher probability, the loss is larger. Binary Cross Entropy can achieve this by calculating the difference between the predicted probability distribution (of the labels) and the actual probability distribution (of the labels) [11].

## 4.3 Training Process

For kNN, the optimal value of k needs to be found for the final model. For this, the training data is fit for various values of k and the labels for the validation and train sets are predicted. Once all validation accuracies are obtained, the k value for which the validation accuracy is the highest, becomes the value of k for the final model. The graph below (Figure 2) shows the accuracies for different values of k. From the graph it is seen that for k=6, the validation accuracy is the highest. For this model the training and validation errors are: 15.51% and 19.03% (The error is given by 1 – accuracy). There isn't much of a difference between the two, meaning there isn't any underfitting or overfitting.

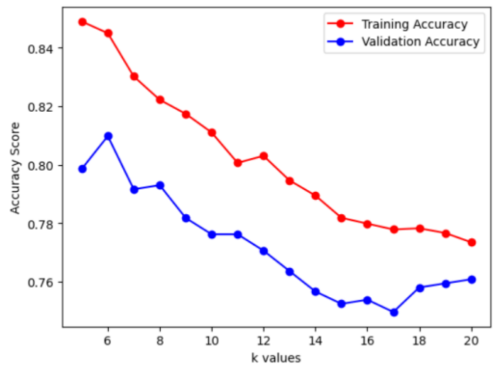


*Figure 3 Accuracy Plot for k-NN*

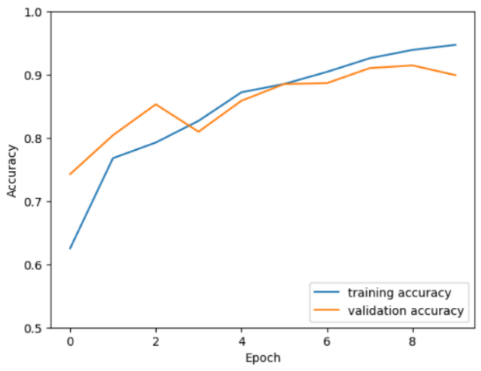


*Figure 2 Accuracy Plot for CNN*

Next the CNN model is trained. It was trained for 10 epochs on the training set, and the weights of the model are finetuned with the help of the data in the validation set – it keeps a check over the model during training, so that the models doesn't learn the weights for just the training set. The final training and validation errors are: 5.27% and 10.07%. Again, there isn't a very large gap in the errors, so it's not like the model has learnt the features of the train set very well, and then is unable to get a good accuracy for the validation set.

The training accuracy is mostly always slightly better than the validation accuracy, or another way to put it is the training error almost always underestimates the validation error. Both models here follow this trend [4, 13], which means they are a good fit for the problem at hand.

### 4.4 Test Set Results

The test set is 10% of the entire dataset, and the points in it are randomly sampled so that it represents the distribution of the whole dataset, i.e., it has approximately the same number of samples from each class. The test set has 358 samples, and they have not been seen by the models during training.

For both the models that are trained, the 3 sets (train, test, and validation) contain the same datapoints. Different subsets of the data are not given to both models. Since the same data is given to both models to train and both models are evaluated on the same test set, it becomes easier to compare them and pick the better model.

The k-NN model obtain an error of 18.72% on the test set, while the CNN model achieves an error of 10.89%. The error of both these models doesn't increase too much from the respective train and validation models.

## 5 Conclusion

Looking at the validation and test errors of the two models, it can be inferred that the Convolutional Neural Network has performed better than the k – Nearest Neighbours algorithm. Both validation and test errors were lower for the CNN model as compared to kNN. Along with this, the test error (10.89%) of CNN is very close to its validation error (10.07%), which means the model was able to learn the overall features of the images well. Had the errors been very different, it would have meant that the model had learnt the features of only part of the data (the training set) and wasn't able to make correct predictions on the unseen test set.

Convolutional Neural Network looks to be the better model for the binary image classification problem at hand, and it is able to correctly classify a bee's health with 89.11% accuracy.

### 5.1 Limitations & Future Enhancements

While the accuracy of both the models is not bad per se, there are still ways to improve it further for better results and easier further analysis. Including more datapoints for the unhealthy bees in all 5 categories, is a good starting point. This would lead to being able to classify the bees into 1 of 6 categories so the beekeeper could know exactly what is wrong with the bees or the hive and can take the specific necessary action as soon as possible.

Another area for further enhancement is being able to incorporate the data with information like the subspecies of the bees, whether it was carrying pollen and the caste of the bee (worker, drone, or queen). Different subspecies could have slightly different features in the images and could result in the model mistaking that as an unhealthy bee. The same goes for the other two features as well.

Incorporating this kind of additional information could lead to a superior model and far better results, which could help the beekeepers make quicker and informed decisions on how to help the bees.

## 6 References

[1] https://www.kaggle.com/datasets/jenny18/honey-bee-annotated-images

[2] https://www.earthday.org/wp-content/uploads/species/bees.pdf

[3] https://www.globalcitizen.org/en/content/importance-of-bees-biodiversity/

[4] Alexander Jung. Machine Learning: The Basics. Springer, Singapore, March 2022.

[5] https://stats.stackexchange.com/questions/420416/does-knn-have-a-loss-function

[6] Image Processing using CNNs: A beginners guide, Mohit Tripathi, May 2023 https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/#:~:text=Why%20do%20we%20use%20CNN,%2C%20image%20segmentation%2C%20and%20classification.

[7] Why are Convolutional Neural Networks good for image classification, Praffул Mishra, May 2019, https://medium.datadriveninvestor.com/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8

[8] Categorical Cross Entropy in Tensorflow and Keras in Python, Koolac (2022) https://www.youtube.com/watch?v=rkULCW_h09k

[9] Convolutional Neural Networks (CNN), https://www.tensorflow.org/tutorials/images/cnn

[10] Understanding binary cross-entropy/log loss: a visual explanation, (2018), https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[11] What you need to know about Sparse Categorical Cross Entropy, 2023, https://rmoklesur.medium.com/what-you-need-to-know-about-sparse-categorical-cross-entropy-9f07497e3a6f

[12] Bee Health Classifier, Shaylynn Morphew, 2020, https://www.kaggle.com/code/shaylynnmorphew/bee-health-categorizer

[13] Validation error less than training error? 2015, https://stats.stackexchange.com/questions/187335/validation-error-less-than-training-error

[14] Training and Test Error: Validating Models in Machine Learning, https://rapidminer.com/blog/validate-models-training-test-error/

[15] Binary Cross entropy, TensorFlow, https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy

[16] KNN Image Classification in Python, Salaar Khan, 2021, https://www.youtube.com/watch?v=Rj9gSsB7kdw

# Appendix

Code File:

In [1]:
```
#imports
import pandas as pd
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
```

## Reading the CSV and balancing the data

In [2]:
```
data = pd.read_csv("bee_data.csv")
data.head()
```

Out[2]:

| | file | date | time | location | zip code | subspecies | health | pollen_carrying | caste |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 041_066.png | 8/28/18 | 16:07 | Alvin, TX, USA | 77511 | -1 | hive being robbed | False | worker |
| **1** | 041_072.png | 8/28/18 | 16:07 | Alvin, TX, USA | 77511 | -1 | hive being robbed | False | worker |
| **2** | 041_073.png | 8/28/18 | 16:07 | Alvin, TX, USA | 77511 | -1 | hive being robbed | False | worker |
| **3** | 041_067.png | 8/28/18 | 16:07 | Alvin, TX, USA | 77511 | -1 | hive being robbed | False | worker |
| **4** | 041_059.png | 8/28/18 | 16:07 | Alvin, TX, USA | 77511 | -1 | hive being robbed | False | worker |

In [3]:
```
data.subspecies.value_counts()
```

Out[3]:
```
Italian honey bee        3008
Russian honey bee         527
Carniolan honey bee       501
1 Mixed local stock 2     472
-1                        428
VSH Italian honey bee     199
Western honey bee          37
Name: subspecies, dtype: int64
```

In [4]:
```
data = data.drop(['date', 'time', 'location', 'zip code', 'subspecies', 'po
data.head()
```

Out[4]:

| | file | health |
|---|---|---|
| **0** | 041_066.png | hive being robbed |
| **1** | 041_072.png | hive being robbed |
| **2** | 041_073.png | hive being robbed |
| **3** | 041_067.png | hive being robbed |
| **4** | 041_059.png | hive being robbed |

In [5]:
```
data.health.value_counts()
```

Out[5]:
```
healthy                      3384
few varrao, hive beetles      579
Varroa, Small Hive Beetles    472
ant problems                  457
hive being robbed             251
missing queen                  29
Name: health, dtype: int64
```

In [6]:
```
#0 is unhealthy
#1 is healthy
data['health_categorical'] = np.where(data['health']!= 'healthy', 0, 1)
data.head()
```

Out[6]:

| | file | health | health_categorical |
|---|---|---|---|
| **0** | 041_066.png | hive being robbed | 0 |
| **1** | 041_072.png | hive being robbed | 0 |
| **2** | 041_073.png | hive being robbed | 0 |
| **3** | 041_067.png | hive being robbed | 0 |
| **4** | 041_059.png | hive being robbed | 0 |

In [7]:
```
a = data.health_categorical.value_counts()
print(a)
print("0 values : " + str(a[0]))
```

```
1    3384
0    1788
Name: health_categorical, dtype: int64
0 values : 1788
```

In [8]:
```
# Need to make the dataset balanced for 0 and 1 entries
data_1 = data[data['health_categorical'] == 1]
data_1.health_categorical.value_counts()
```

Out[8]:
```
1    3384
Name: health_categorical, dtype: int64
```

In [9]:
```
data_1_sampled = data_1.sample(n = a[0])
len(data_1_sampled)
```

Out[9]:
```
1788
```

In [10]:
```
data_final = pd.concat([data_1_sampled, data[data['health_categorical'] ==
data_final.head()
```

Out[10]:

| | file | health | health_categorical |
|---|---|---|---|
| **1269** | 005_301.png | healthy | 1 |
| **4029** | 032_549.png | healthy | 1 |
| **2918** | 010_595.png | healthy | 1 |
| **3100** | 018_050.png | healthy | 1 |
| **3608** | 015_1116.png | healthy | 1 |

In [11]:
```
print(data_final.health_categorical.value_counts())
print("\nDataset Size:" + str(len(data_final)))
```

```
1    1788
0    1788
Name: health_categorical, dtype: int64

Dataset Size:3576
```

In [12]:
```
data_final = data_final.drop(['health'], axis=1)
data_final.head()
```

Out[12]:

| | file | health_categorical |
|---|---|---|
| **1269** | 005_301.png | 1 |
| **4029** | 032_549.png | 1 |
| **2918** | 010_595.png | 1 |
| **3100** | 018_050.png | 1 |
| **3608** | 015_1116.png | 1 |

In [13]:
```
#Randomizing the dataset for the train/test split
data_final = data_final.sample(frac = 1)
data_final = data_final.reset_index(drop = True)
data_final.head()
```

Out[13]:

| | file | health_categorical |
|---|---|---|
| **0** | 010_755.png | 1 |
| **1** | 008_290.png | 1 |
| **2** | 039_025.png | 0 |
| **3** | 038_116.png | 0 |
| **4** | 003_282.png | 1 |

## Reading Image Files: ¶

In [14]:
```
directory = "C:\\Users\\myname\\OneDrive\\Documents\\Machine Learning\\Proj
Bee_images = [image.load_img(directory+img_name,target_size=(64,64)) for im
```

In [15]:
```
fig = plt.figure(figsize=(5, 3))
Img_idx = [55, 300, 1900, 2000]
for i in range(0, 4):
    Img = Bee_images[Img_idx[i]]
    fig.add_subplot(1, 4, i+1)
    plt.imshow(Img)
    plt.axis('off')
    plt.title("label: " + str(data_final.iloc[Img_idx[i]]["health_categoric
```

In [16]:
```
#Converting images to pixel matrices
pixel_matrix = [np.array(image.img_to_array(i)) for i in Bee_images]
np.shape(pixel_matrix)
```

Out[16]:
```
(3576, 64, 64, 3)
```

In [17]:
```
#Converting images into grayscale
gray_matrix = []

for i in range(0, np.shape(pixel_matrix)[0]):
    gry_img = cv2.cvtColor(pixel_matrix[i], cv2.COLOR_BGR2GRAY)
    gray_matrix.append(gry_img)


np.shape(gray_matrix)
```

Out[17]:
```
(3576, 64, 64)
```

In [18]:
```
#normalizing pixel intensity values - all values will be between 0 and 1 no
gray_matrix = [i/255.0 for i in gray_matrix]
```

In [19]:
```
#reshaping the 64x64 matrix representing each image, to a 1x4096
def horizontalStacking(gray_matrix):
    gray_reshaped = []

    for i in range(0, np.shape(gray_matrix)[0]):
        gray_reshaped.append(np.reshape(gray_matrix[i], (-1,)))

    #np.shape(gray_reshaped)
    return gray_reshaped
```

## Train/Test Split for kNN

In [20]:
```
gray_reshaped = horizontalStacking(gray_matrix)
```

In [21]:
```
n1 = int(np.floor(0.7 * np.shape(gray_reshaped)[0]))
n2 = n1 + int(np.floor(0.2 * np.shape(gray_reshaped)[0]))
```

In [22]:
```
final_data = np.array(gray_reshaped)
train_x_knn, val_x_knn, test_x_knn = final_data[:n1,:], final_data[n1:n2,:]
```

In [23]:
```
train_x_knn.shape, val_x_knn.shape, test_x_knn.shape
```

Out[23]:
```
((2503, 4096), (715, 4096), (358, 4096))
```

In [24]:
```
train_y_knn = data_final.loc[:n1-1,:]["health_categorical"]
val_y_knn = data_final.loc[n1:n2-1,:]["health_categorical"]
test_y_knn = data_final.loc[n2:,:]["health_categorical"]
```

In [25]:
```
train_y_knn.shape, val_y_knn.shape, test_y_knn.shape
```

Out[25]:
```
((2503,), (715,), (358,))
```

## Train/Test Split for CNN

In [26]:
```
gray_matrix_arr = np.array(gray_matrix)
gray_matrix_arr.shape
```

Out[26]:
```
(3576, 64, 64)
```

In [27]:
```
train_x_cnn, val_x_cnn, test_x_cnn = gray_matrix_arr[:n1,:], gray_matrix_ar
train_x_cnn.shape,val_x_cnn.shape, test_x_cnn.shape
```

Out[27]:
```
((2503, 64, 64), (715, 64, 64), (358, 64, 64))
```

In [28]:
```
train_y_cnn = data_final.loc[:n1-1,:]["health_categorical"]
val_y_cnn = data_final.loc[n1:n2-1,:]["health_categorical"]
test_y_cnn = data_final.loc[n2:,:]["health_categorical"]

train_y_cnn.shape, val_y_cnn.shape, test_y_cnn.shape
```

Out[28]:
```
((2503,), (715,), (358,))
```

In [29]:
```
n_trainsamples = n1
n_valsamples = n2-n1
n_testsamples = gray_matrix_arr.shape[0] - n_valsamples - n_trainsamples
```

In [30]:
```
train_x_cnn = train_x_cnn.reshape((n_trainsamples, 64, 64, 1))
val_x_cnn = val_x_cnn.reshape((n_valsamples, 64, 64, 1))
test_x_cnn = test_x_cnn.reshape((n_testsamples, 64, 64, 1))
train_x_cnn.shape, val_x_cnn.shape, test_x_cnn.shape
```

Out[30]:
```
((2503, 64, 64, 1), (715, 64, 64, 1), (358, 64, 64, 1))
```

## CNN Model

In [31]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64,
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1))
```

In [32]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        320

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 64)        36928

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 64)                589888

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 645697 (2.46 MB)
Trainable params: 645697 (2.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

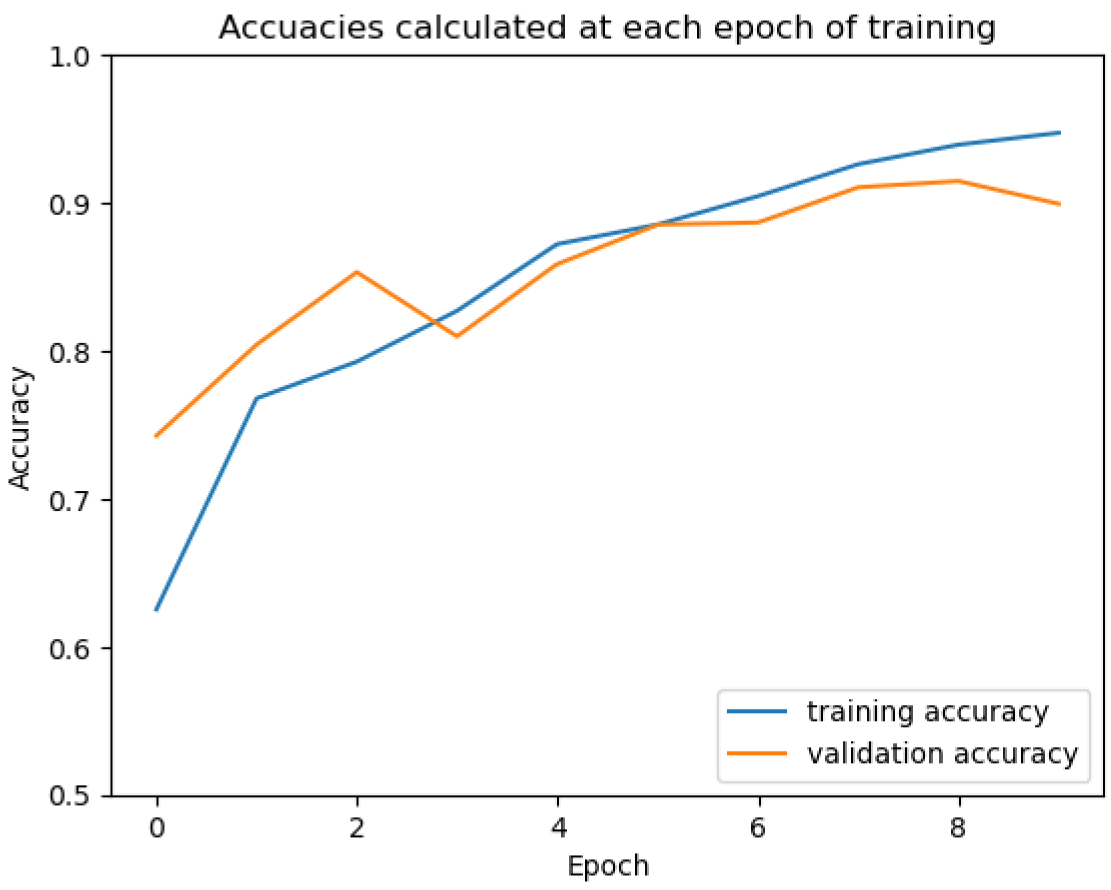
In [33]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [34]:
```
history = model.fit(train_x_cnn, train_y_cnn, epochs=10,
                    validation_data=(val_x_cnn, val_y_cnn))
```

```
Epoch 1/10
79/79 [==============================] - 16s 161ms/step - loss: 0.5934 - a
ccuracy: 0.6252 - val_loss: 0.4675 - val_accuracy: 0.7427
Epoch 2/10
79/79 [==============================] - 12s 154ms/step - loss: 0.4397 - a
ccuracy: 0.7679 - val_loss: 0.3828 - val_accuracy: 0.8042
Epoch 3/10
79/79 [==============================] - 12s 153ms/step - loss: 0.3995 - a
ccuracy: 0.7926 - val_loss: 0.3304 - val_accuracy: 0.8531
Epoch 4/10
79/79 [==============================] - 12s 153ms/step - loss: 0.3549 - a
ccuracy: 0.8270 - val_loss: 0.3335 - val_accuracy: 0.8098
Epoch 5/10
79/79 [==============================] - 12s 151ms/step - loss: 0.2772 - a
ccuracy: 0.8722 - val_loss: 0.2770 - val_accuracy: 0.8587
Epoch 6/10
79/79 [==============================] - 12s 152ms/step - loss: 0.2533 - a
ccuracy: 0.8853 - val_loss: 0.2974 - val_accuracy: 0.8853
Epoch 7/10
79/79 [==============================] - 12s 152ms/step - loss: 0.2166 - a
ccuracy: 0.9045 - val_loss: 0.2519 - val_accuracy: 0.8867
Epoch 8/10
79/79 [==============================] - 12s 151ms/step - loss: 0.1804 - a
ccuracy: 0.9261 - val_loss: 0.2538 - val_accuracy: 0.9105
Epoch 9/10
79/79 [==============================] - 12s 152ms/step - loss: 0.1457 - a
ccuracy: 0.9393 - val_loss: 0.2472 - val_accuracy: 0.9147
Epoch 10/10
79/79 [==============================] - 12s 150ms/step - loss: 0.1255 - a
ccuracy: 0.9473 - val_loss: 0.2639 - val_accuracy: 0.8993
```

In [53]:
```
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label = 'validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title("Accuacies calculated at each epoch of training")
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

Out[53]:
```
<matplotlib.legend.Legend at 0x20cbe139a50>
```



In [36]:
```
test_loss, test_acc = model.evaluate(test_x_cnn,  test_y_cnn, verbose=2)
```

```
12/12 - 1s - loss: 0.3058 - accuracy: 0.8911 - 559ms/epoch - 47ms/step
```

## kNN Model

In [37]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [38]:
```
knn = KNeighborsClassifier(n_neighbors=10)
```

In [39]:
```
knn.fit(train_x_knn, train_y_knn)
```
Out[39]:
```
KNeighborsClassifier(n_neighbors=10)
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

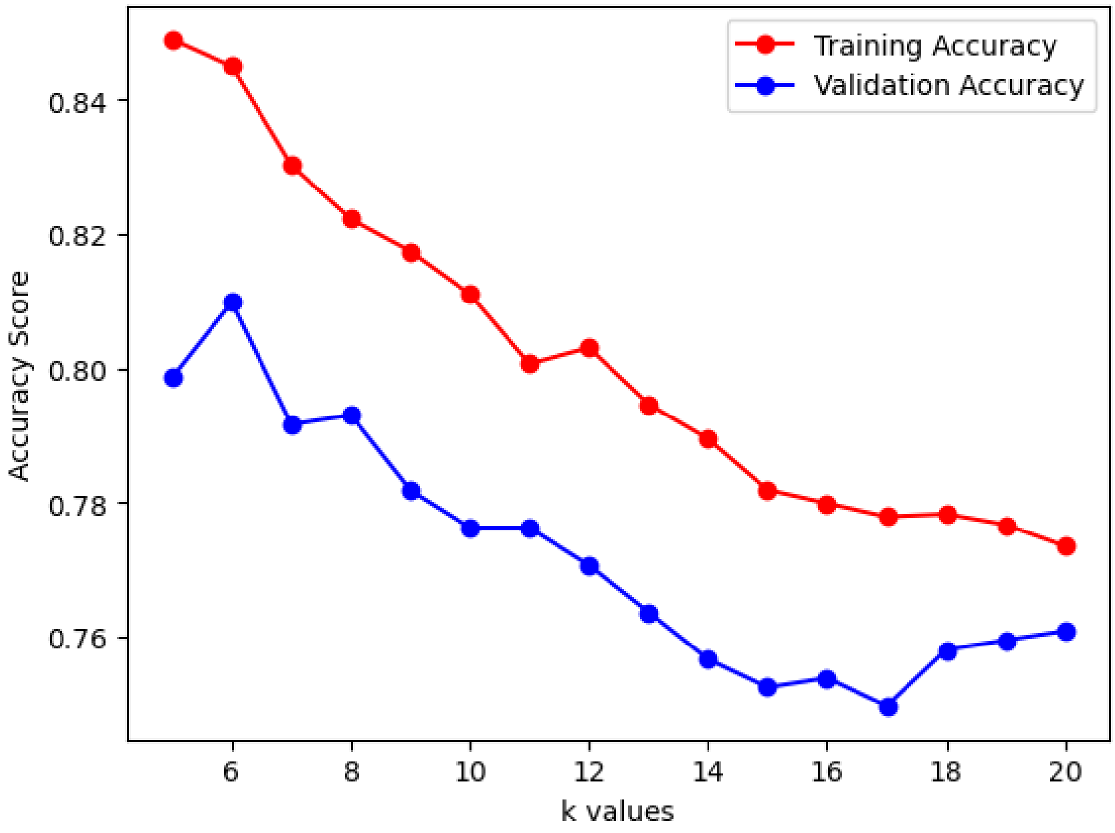
In [40]:
```
y_pred = knn.predict(test_x_knn)
```
In [41]:
```
accuracy = accuracy_score(test_y_cnn, y_pred)
```
In [42]:
```
accuracy
```
Out[42]:
```
0.7932960893854749
```
In [43]:
```
results_training = []
results_validation = []
for k in range(5, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_x_knn, train_y_knn)

    #Training Accuracy
    y_pred_train = knn.predict(train_x_knn)
    acc_train = accuracy_score(train_y_knn, y_pred_train)
    results_training.append(acc_train)

    #Validation Accuracy
    y_pred_val = knn.predict(val_x_knn)
    acc_val = accuracy_score(val_y_knn, y_pred_val)
    results_validation.append(acc_val)
```
In [44]:
```
#selcting the best k value
k_final = 5 + np.argmax(results_validation)
```
In [45]:
```
k_list = np.arange(5, 21, 1)
k_list
```
Out[45]:
```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

In [54]:
```
plt.plot(k_list, results_training, 'ro-', k_list, results_validation, 'bo-'
plt.xscale
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel("k values")
plt.ylabel("Accuracy Score")
#plt.title("Training and Validation Accuracies for different values of k")
```

Out[54]:
```
Text(0, 0.5, 'Accuracy Score')
```



In [47]:
```
max(results_validation), np.argmax(results_validation)
```

Out[47]:
```
(0.8097902097902098, 1)
```

In [56]:
```
results_training[1], results_validation[1]
```

Out[56]:
```
(0.8449860167798642, 0.8097902097902098)
```

In [49]:
```
knn_final = KNeighborsClassifier(n_neighbors=k_final)
knn_final.fit(train_x_knn, train_y_knn)
y_pred_test = knn_final.predict(test_x_knn)
accuracy_testing = accuracy_score(test_y_knn, y_pred_test)
accuracy_testing
```

Out[49]:
```
0.8128491620111732
```

In [ ]:
```

```