Bee Image Classification Using Convolutional Neural Networks

September 2023

1 Introduction

The objective of our machine learning project is to develop a model that can accurately distinguish images of bees from all other images. Images containing bees are labeled with 1, and images without bees are labeled with 0. This binary classification problem holds significant real-world implications as it has several applications in agriculture, environmental conservation, and beekeeping. For instance, accurately detecting the presence of bees can lead to increased crop yields and reduced pesticide risks, thereby benefiting both agriculture and ecosystems. By accurately classifying bee images, this project contributes to addressing critical ecological and agricultural challenges, benefiting both the environment and society.

The report is organized as follows: Section 2 describes the dataset, features, and labels. Section 3 presents the methodology, including data preprocessing and the selection of models. Section 4 discusses model comparisons, training, validation, and test results, and selects the final method. Finally, Section 5 concludes the report and suggests future improvements. The code can be found in Appendix A and references in Appendix B.

2 Problem Formulation

2.1 Data Points, Features, and Labels

This project is classified as supervised learning as the model is provided with both the images and their corresponding binary labels. During the training phase, the models learn from labeled data, identifying patterns and features that enable them to distinguish between bee and non-bee images.

- Data Points: We have a total of 30,000 images in our dataset, with a distribution of 10,000 bee images and 20,000 non-bee images. With each image represented by a 3D tensor of RGB values, the input is multi-dimensional.
- Features: The features are the pixel values of each image. The images depict either bees or various natural scenes that are common in images with bees, such as landscapes, flowers, animals, and other insects. We chose related images to help the algorithm distinguish between bees from their natural background (flowers & landscapes) as well as distinguish bees from other similar insects.
- Labels: The labels are binary, with a value of 1 representing bee images and 0 representing non-bee images, such as

3 Methods

3.1 Dataset Description and Preprocessing

The dataset used for this project has been collected from multiple sources including the Intel Image Classification dataset and The BeeImage Dataset. The full list is too long to be fully listed here. As such, to give due credit, all datasets used for this project, and links to them, are listed in Appendix B.

The datasets combined contain 30,000 images split between bee and non-bee images. Each image varies in dimensions and color. The preprocessing/cleaning steps include resizing images to a uniform size, normalizing pixel values, and performing in-place data augmentation with techniques like image rotation and color manipulation.

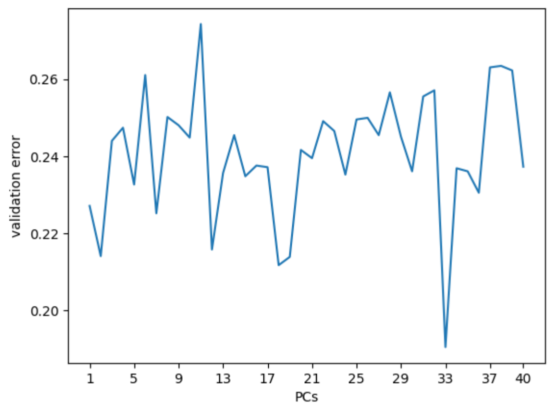
3.2 Feature Selection

In this project, the features are the pixel values of the images. We made a deliberate choice not to manually pick specific features because CNN models excel at feature extraction, which eliminates the need for manual feature engineering.

3.3 Convolutional Neural Network (CNN)

We chose Convolutional Neural Networks (CNNs) as the first model for this image classification task. CNNs are well-suited for image-related tasks, as they can autonomously learn to extract relevant features from raw pixel data through convolutional layers. Additionally, CNNs have the ability to capture spatial relationships within images. They are effective at recognizing patterns, objects, and features in images by considering how pixels relate to each other in terms of their positions and structures. This ability to analyze the spatial arrangement of features makes CNNs a natural choice for image classification.

3.4 Principle Component Analysis (PCA) On A Neural Network



Graph 1.1: A graph of validation error against the number of remaining Principal Components(PCs)

We chose Principal Component Analysis (PCA) in combination with a CNN as our second machine-learning model for image classification. Principal Component Analysis defines new attributes (principal components or PCs) as mutually orthogonal linear combinations of the original features.[1] Using PCA in the preprocessing step allowed us to significantly reduce the data's dimensionality while preserving most information. PCA, essentially, compresses the image, thereby removing noise from the image and allowing the CNN to focus on key details. This could improve the network's ability to differentiate between bee and non-bee images.

Choosing the correct number of principal components for the compressed image is challenging. Too few, and the image loses too much detail for the CNN to effectively distinguish the two types of images. Too many and the advantages of image compression become negligible. To solve this, we ran a simulation by training 14 different CNNs for each number of remaining PCs from 1 to 40 with a limited data set. This allowed us to observe any general trends in validation error with respect to the number of PCs. From graph 1.1 we observe that the validation error is the lowest for 33 PCs. Therefore, we chose 33 PCs for the final model.

3.5 Integration of PCA with CNNs

CNN is classified as a supervised machine learning model as the model is provided with both the images and their corresponding binary labels. On the other hand, PCA is an unsupervised technique, used for dimensionality reduction and feature extraction without any labeled data. Hence, PCA's integration with CNNs in our machine learning project may create ambiguity between supervised and unsupervised aspects. In this case, PCA serves as an unsupervised preprocessing step to reduce the dimensionality of the data before being used in a supervised learning context with CNNs. Therefore, both methods combined form a supervised learning model.

[1] Shereena&David (2015)

3.6 Choice of Loss Function

$$\frac{1}{m}\sum_{i=1}^{m} y_i log(\hat{y_i}) + (1-y)log(1-\hat{y_i})$$

Where

$$y_i = label$$
$$\hat{y_i} = prediction$$

Since this classification problem only involves two classes, we use binary cross-entropy as a loss function for **both** models since it is uniquely suited for binary classification problems: binary cross-entropy is simple and fast to compute. Furthermore, the logarithm in the binary cross-entropy will undo the exponential behavior of the sigmoid activation function. Large gradients are important for gradient-descent algorithms to make sufficient progress in each iteration;[2] the logarithm will avoid the vanishing gradient problem, where, with large values, the gradient approaches 0 and learning slows down.

3.7 Model Validation

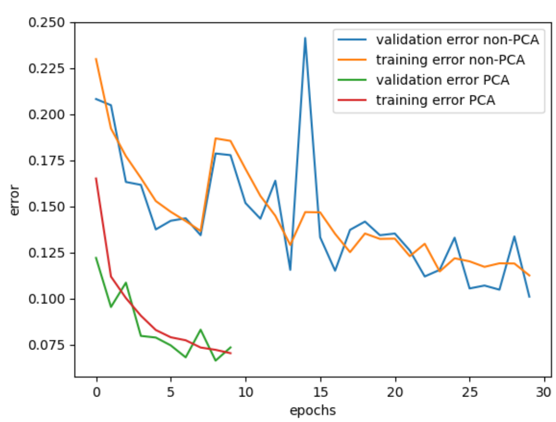
We have split the dataset into three subsets: training, validation, and testing sets. The data split is as follows:

- 70% of the data is allocated to the training set: This large training set is important for the model to learn the underlying patterns in the data.
- 15% of the data is allocated to the validation set: The validation set is used to evaluate the model's performance, tune the hyperparameters, and monitor for overfitting.
- 15% of the data is allocated to the testing set: The testing set is used to evaluate the model's final performance.

By plotting training and validation loss and comparing it with training and validation accuracy, we can look for trends such as overfitting (if validation loss starts increasing while training loss keeps decreasing) or underfitting (if both losses are high).

4 Results

4.1 Training and validation errors



Graph 2.1: A graph of error against epochs for PCA-based and pure CNNs

[2] oW_

We compared the training and validation errors of each model across 10 epochs of training for the PCA-based CNN and 30 epochs of training for the pure CNN. Each epoch is a forward pass and backpropagation through the CNN with the entire training dataset. The training and validation error consistently decreased with the number of epochs as models learned from the data. From all epochs, we chose the versions with the lowest validation error for both the PCA-based CNN and CNN models. From Figure 1 we can see that the PCA-based CNN model reached the lowest validation error of 6.6% at epoch 9 while the pure CNN model reached the lowest validation error of 10.1% at epoch 30.

The reason for the pure CNN having 3 times more training epochs than the PCA-based CNN is due to hardware limitations: the machine on which both were trained required 2 hours to train the pure CNN through 30 epochs, while it required 5 hours to train the PCA-based CNN through only 10. This is due to applying the costly PCA compression algorithm to each of the approximately 25,000 images in each epoch. Due to time constraints, we could not train either for longer. Even so, the PCA-based CNN far outperformed the pure CNN model already at the second epoch in terms of validation error. Therefore, the final choice of model is the PCA-based CNN.

4.2 Test set

The test set used for our evaluation was constructed using augmented versions of the images that were not included in either the training or validation sets. To ensure a balanced representation of both bee and non-bee images, the test set was shuffled, creating an even distribution of these categories. Finally, we obtained a test error of 8.2% from the PCA-based CNN model. This shows the model is capable of accurately classifying images of bees from all other images.

5 Conclusion

In summary, our project aimed to develop two distinct models for image classification to differentiate between bee and non-bee images. We created both a Convolutional Neural Network (CNN) and a PCA-based CNN. The PCA-based CNN consistently performed better than the standard CNN, with a 8.2% test error rate. Training and validation errors for both models decreased at approximately the same rate and neither show signs of overfitting. Therefore, we can confidently select the PCA-based CNN as the superior model.

While the results are promising, there is room for improvement. Future directions may include exploring additional data augmentation techniques, experimenting with different CNN architectures, and fine-tuning hyperparameters to further enhance the model's performance. Additionally, collecting more diverse training data could contribute to even more robust classification capabilities.

In conclusion, our findings indicate that the PCA-based CNNs method is the most suitable for accurately classifying bee and non-bee images. This highlights the importance of integrating dimensionality reduction techniques like PCA with deep learning models for image classification.

6 Bibliography

- oW_. (n.d.). What makes binary cross entropy a better choice for binary ... https://datascience.stackexchange.com/questions/53400/what-makes-binary-cross-entropy-a-better-choice-for-binary-classification-than-o

- Hussain, I. (2021). An ultimate guide to color image compression using PCA in python. Medium. Available at: https://towardsdatascience.com/dimensionality-reduction-of-a-color-photo-splitting-into-rgb-channels-using-pca-algorithm-in-python-ba01580a1118

- Shereena, V. B. & David, J. M. (2015, June). SIGNIFICANCE OF DIMENSIONALITY REDUCTION IN IMAGE PROCESSING . https://core.ac.uk/download/pdf/212147876.pdf

7 Appendix A code

```
from os.path import join
import datetime
from os import listdir
from os import getcwd
from os.path import isfile, splitext
import numpy as np
import keras
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib.pyplot import imread
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sns


def display_conf_matrix (conf_mat):

    plt.figure(figsize=(8,6), dpi=100)
    sns.set(font_scale = 1.1)


    ax = sns.heatmap(conf_matrix, annot=True, fmt='d')

    ax.set_xlabel("Predicted Diagnosis", fontsize=14, labelpad=20)
    ax.xaxis.set_ticklabels(['Negative', 'Positive'])

    ax.set_ylabel("Actual Diagnosis", fontsize=14, labelpad=20)
    ax.yaxis.set_ticklabels(['Negative', 'Positive'])

    # set plot title
    ax.set_title("Confusion Matrix for the Diabetes Detection Model", fontsize=14, pad=20)

    plt.show()



# a method to correct an incorrect directory structure for the keras imagedatagenerator
def correct_directory_structure(train_percent, test_percent, validation_percent, *dpaths): #
    dpaths: directories with classes of data
    import random
    from math import floor
    cwd = getcwd() + '\\'
    dir_list = listdir(cwd)
    #check that we are not overwriting directories train,test,validation
    assert not any (d in dir_list for d in ['train', 'test', 'validation']), 'cannot correct
     directory structure, as some files will be overwritten'
```

```python
    assert all (d in dir_list for d in dpaths), 'the listed directories are not in the current path'

    [os.mkdir(f"{i}") for i in ['train','test','validation']]
    [os.mkdir(f"{i}\\{d}") for i in ['train','test','validation'] for d in dpaths]

    data_gen = ((d,listdir(d)) for d in dpaths)
    for d,i in data_gen:
        shuffled_files = random.sample(i,len(i))

        train_end = floor(len(shuffled_files)*train_percent)
        test_end = floor(train_end + (len(shuffled_files) - train_end)*test_percent/(test_percent + validation_percent))

        train_data = shuffled_files[0:train_end]
        test_data = shuffled_files[train_end:test_end]
        validation_data = shuffled_files[test_end:]

        for m in train_data:
            try:
                os.rename (f"{cwd}{d}\\{m}", f"{cwd}train\\{d}\\{m}")
            except:
                print ("hit and error, moving forward")
        for m in test_data:
            try:
                os.rename (f"{cwd}{d}\\{m}", f"{cwd}test\\{d}\\{m}")
            except:
                print ("hit and error, moving forward")
        for m in validation_data:
            try:
                os.rename (f"{cwd}{d}\\{m}", f"{cwd}validation\\{d}\\{m}")
            except:
                print ("hit and error, moving forward")

image_shape = (400,400,3)

# returns the iterators for data in directories so the model can be trained on batches of data without needing much RAM
def get_image_iterators(train_dir = 'train',
                        val_dir = 'validation',
                        test_dir = 'test',
                        mode = 'binary',
                        batch = 64,
                        size = image_shape[:2],
                        processing = None):
    cwd = getcwd() + '\\'
    image_gen = ImageDataGenerator(
        rotation_range = 90,
        width_shift_range = 0.1,
        height_shift_range = 0.1,
        shear_range = 5,
        brightness_range = (0.7,1.),
        horizontal_flip = True,
        vertical_flip = True,
        preprocessing_function = processing
    )
    # these already shuffle the data
    train_set = image_gen.flow_from_directory(train_dir, class_mode = mode, batch_size = batch, target_size = size)
    val_set = image_gen.flow_from_directory(val_dir, class_mode = mode, batch_size = batch, target_size = size)
    test_set = image_gen.flow_from_directory(test_dir, class_mode = mode, batch_size = batch, target_size = size)
    return (train_set, val_set, test_set)

#correct_directory_structure(0.7,0.15,0.15, 'Bees', 'Not_Bees')

# for this function to work, the directory structure must be:
```

```
# train/
#    |
#               class_1/
#    |                  image1.jpg
#    |                  image2.jpg
#    |                  ...
#    |
#               class_2/
#    |                  image1.jpg
#    |                  image2.jpg
#    |                  ...
# test/
#    |
#               class_1/
#    |                  image1.jpg
#    |                  image2.jpg
#    |                  ...
#    |
#               class_2/
#    |                  image1.jpg
#    |                  image2.jpg
#    |                  ...
# ...

train,validation,test = get_image_iterators(batch = 32)

class SaverCallback (tf.keras.callbacks.Callback):
    def __init__ (self, model, test_data, save_path, batch = 10): # test_data must be either a keras sequence or dataset
        self.best = test_data if type(test_data) == float else model.evaluate(test_data, callbacks = [self], batch_size = batch)[1] #init val accuracy
        self.save_path = save_path
        self.model = model
    def on_epoch_end (self, epoch, logs = None):
        assert logs.get('val_accuracy') != None

        if self.best < logs['val_accuracy']:
            self.best = logs['val_accuracy']
            self.model.save(self.save_path)

class accuracy_recorder (keras.callbacks.Callback):
    def __init__ (self, init_compression = None):
        self.accuracies = pd.DataFrame (columns = ['compression', 'epoch', 'val_acc', 'val_loss', 'acc', 'loss'])
        self.compression_factor = init_compression
    def on_epoch_end (self, epoch, logs=None):
        assert logs.get('val_accuracy') != None, 'no validation set for val_acc_recorder'
        self.accuracies = self.accuracies._append({'compression': self.compression_factor,
                                     'epoch' : epoch,
                                      'val_acc': logs['val_accuracy'],
                                      'val_loss': logs['val_loss'],
                                      'acc': logs['accuracy'],
                                      'loss': logs['loss']}, ignore_index = True)

    def set_compression (self, new_compression):
        assert new_compression > 0, 'cannot have pca with n_components = 0'
        self.compression_factor = new_compression

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
       ers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Flatten(),                    input_shape=image_shape),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
    tf.keras.lay
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

```
model.compile (
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics= ['accuracy']
)

CNN_model_path = join(getcwd(), 'models', 'CNNmodel')
saver = SaverCallback(model, 0., CNN_model_path)
recorder = accuracy_recorder()

model.fit(train,
        validation_data = validation,
        callbacks = [recorder, saver],
        epochs = 30,
        batch_size = 10,
        initial_epoch = 20)

csv_data_path = join(getcwd(),'CSVdata', 'CNNtraining2.csv')
recorder.accuracies.to_csv(csv_data_path)
model.evaluate (test)

# let's plot the data
val_error = 1 - recorder.accuracies['val_acc']
error = 1 - recorder.accuracies['acc']
epochs = recorder.accuracies['epoch']
plt.plot(epochs, val_error, label='validation error non-PCA')
plt.plot (epochs, error, label = 'training error non-PCA')
plt.xlabel('epochs')
plt.ylabel('error')
plt.legend()
plt.show()

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from scipy.stats import stats
import matplotlib.image as mpimg

compression = 33
def compress_images(img):

        # Split into channels
        blue, green, red = cv2.split(img)

        # Scale the data between 0 and 1 for all channels
        r = red / 255
        b = blue / 255
        g = green / 255

        # Fit and transform data in PCA to reduce dimensionality
        red_pca = PCA(n_components=compression)
        red_pca.fit(r)
        red_trans = red_pca.transform(r)

        blue_pca = PCA(n_components=compression)
        blue_pca.fit(b)
        blue_trans = blue_pca.transform(b)

        green_pca = PCA(n_components=compression)
        green_pca.fit(g)
        green_trans = green_pca.transform(g)

        # Reconstruct the images
        r_arr = red_pca.inverse_transform(red_trans)
        b_arr = blue_pca.inverse_transform(blue_trans)
        g_arr = green_pca.inverse_transform(green_trans)

        # Merge channels into one
        compressed_image = (cv2.merge((b_arr, g_arr, r_arr)))
```

```
        return np.array(compressed_image)


modelPCA = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
                           input_shape=image_shape),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])

modelPCA.compile (
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics= ['accuracy']
)

train_cmp,validation_cmp,test_cmp = get_image_iterators(processing = compress_images)
PCA_model_path = join(getcwd(), 'models', 'PCAmodel')
saver = SaverCallback(modelPCA, 0., PCA_model_path)
recorderPCA = accuracy_recorder()
modelPCA.fit(train_cmp,
            validation_data = validation_cmp,
            epochs = 40,
            callbacks = [saver, recorderPCA])


train_cmp,validation_cmp,test_cmp = get_image_iterators(
                                    processing = compress_images, batch = 8)

from functools import reduce
stack_features_and_labels = lambda fl1, fl2: (np.vstack((fl1[0], fl2[0])), np.hstack((fl1
    [1], fl2[1])))

# take some data and load it into ram so it's fast to train with it
def limit_data (data, included_batches): # requires an iterator
    holder = next(data)
    for i in range (1, included_batches):
        holder = stack_features_and_labels (holder, next(data))
    return holder

limited_train_data = limit_data(train_cmp,3)
limited_test_data = limit_data(test_cmp,1)
limited_validation_data = limit_data(validation_cmp,3)

trend_tester_model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
                           input_shape=image_shape),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])

trend_tester_model.compile (
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics= ['accuracy']
)
validation_accuracies = accuracy_recorder()
save_path = join(getcwd(), 'trend_tester_temp')
trend_tester_model.save_weights(save_path)

```

```python
#what compression we start at
start_compression = 40
#how many times we train a new model under one compression
reps = 14

for i in range (start_compression, 0, -1):
    validation_accuracies.set_compression(i)
    compression = i
    for _ in range(0, reps):
        trend_tester_model.load_weights(save_path)
        trend_tester_model.fit(limited_train_data[0],
                               limited_train_data[1],
                               validation_data = limited_validation_data,
                               batch_size= 8,
                               epochs = 20,
                              callbacks = [validation_accuracies])

csv_data_path = join(getcwd(),'CSVdata', 'trend_tester4.csv')
validation_accuracies.accuracies.to_csv(csv_data_path)
dat = pd.read_csv(join(getcwd(),'CSVdata', 'trend_tester4.csv'))._append(pd.read_csv(join(getcwd(),'CSVdata', 'trend_tester3.csv')))
dat.groupby('compression')['val_acc'].mean().plot()
```

8 Appendix B datasets

- Bhathena, Jehan. “Weather Image Recognition.” Kaggle, 26 Nov. 2021, www.kaggle.com/datasets/jehanbhathena/weather-dataset
- DePie. “Butterfly Image Classification.” Kaggle, 24 June 2023, www.kaggle.com/datasets/phucthaiv02/butterfly-image-classification
- Gerry. “Birds 525 Species- Image Classification.” Kaggle, 20 Apr. 2023, www.kaggle.com/datasets/gpiosenka/100-bird-species
- Rey, George. ”Bee or wasp?” Kaggle, https://www.kaggle.com/datasets/jerzydziewierz/bee-vs-wasp
- Yang, Jenny. “The Beeimage Dataset: Annotated Honey Bee Images.” Kaggle, 16 Sept. 2018, www.kaggle.com/datasets/jenny18/honey-bee-annotated-images