Group 3 - MA 3 (Batch 2)

Dimaculangan, Rob Ian

Gutierrez, Gaerlan John

Hernandez, John Robynn

Pangilinan, Patrick

Recto, Antonio Julian

Topic: Identifying Mushrooms Using Convolutional Neural Network: A Multiclass Classification Problem

---

Introduction

The problem of identifying the class of a mushroom from its image is a classic example of image multi-classification. One can solve this problem by using a convolutional neural network (CNN). The four classes in the dataset (conditionally_edible, deadly, edible, and poisonous) can be treated as separate classes to be recognized by the model. The goal of the model is to learn the features of the different species of mushrooms from the images and predict the correct class for an unseen image.

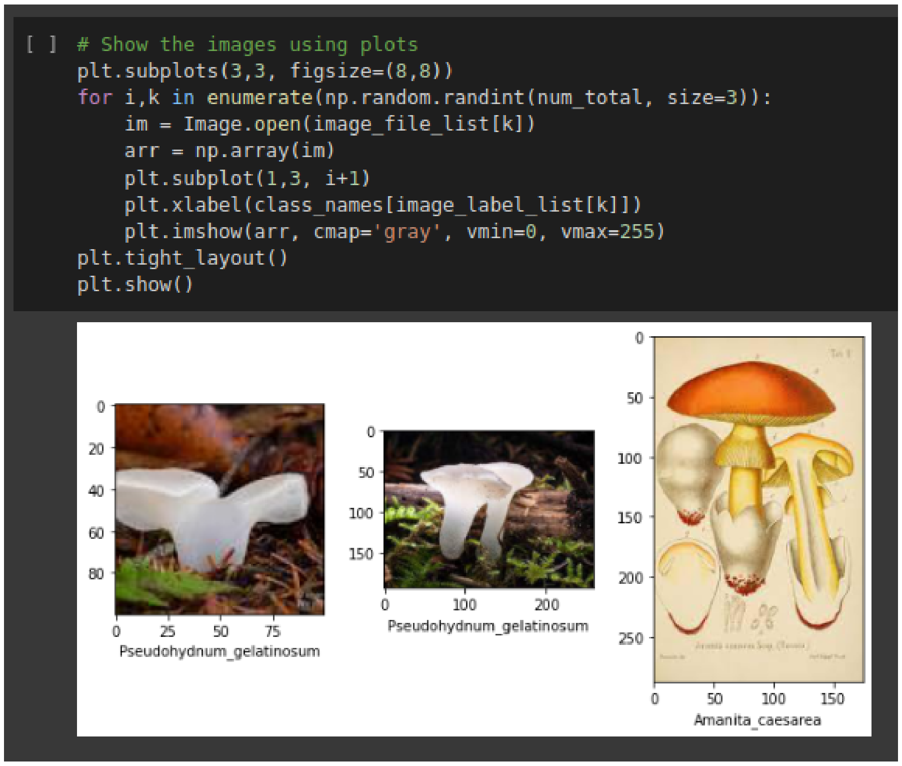
The description of the classes are as follows:

Conditionally Edible: These mushrooms are considered safe to eat, but only under certain conditions. They may have specific preparation requirements or may only be safe to consume in small quantities. It's important to consult a reliable source before consuming any mushrooms in this class. This class has 420 images in the dataset.

Deadly: These mushrooms are highly toxic and can cause serious illness or death if consumed. They should be avoided at all costs. This class has 1190 images in the dataset.

Edible: These mushrooms are safe to eat and are commonly consumed as a food source. They may have a distinct taste and texture, and can be prepared in a variety of ways. This class has 2475 images in the dataset.

Poisonous: These mushrooms are not safe to eat and can cause symptoms such as nausea, vomiting, and organ damage. They may be mistaken for edible mushrooms, so it's important to be able to accurately identify them. This class has 4696 images in the dataset.



```
[ ] # Show the images using plots
    plt.subplots(3,3, figsize=(8,8))
    for i,k in enumerate(np.random.randint(num_total, size=3)):
        im = Image.open(image_file_list[k])
        arr = np.array(im)
        plt.subplot(1,3, i+1)
        plt.xlabel(class_names[image_label_list[k]])
        plt.imshow(arr, cmap='gray', vmin=0, vmax=255)
    plt.tight_layout()
    plt.show()
```

Sample Images from the Mushroom Dataset

The dataset in question is gathered from Kaggle on Mushrooms by Kuno-Williams, D. (2022). and is found to be suitable for use in this type of problem seeing as according to Kaggle, it has a usability score of 7.50, having 100% completeness and credibility, only having a low score on compatibility with 33% due to not having a complete file description, however, this does not affect the performance of the neural network in classifying the mushrooms as the file description is irrelevant in testing and the classes are already labeled appropriately.

The model for this type of problem doesn’t just benefit a single group or organization of people, but rather multiple groups of people seeing as it has a wide variety of implications which could be used by a wide range of organizations and individuals to improve food safety, protect public health, and support scientific research and conservation efforts.

Specifically, it can help the following:

Mushroom growers and sellers: Accurate identification of mushrooms can be important for growers and sellers to ensure that they are providing safe and high-quality products to consumers. A CNN model could be used to help identify mushrooms in a farming or harvesting operation, or to verify the identity of mushrooms in a retail setting.

Government regulatory agencies: Accurate identification of mushrooms is also important for government regulatory agencies that are responsible for monitoring food safety. A CNN model could be used to help identify mushrooms that may be mislabeled or misidentified, which could pose a risk to public health.

Mycologists and researchers: Mycologists and researchers who study mushrooms could also benefit from this type of research. A CNN model could be used to help identify mushrooms in the wild, or to aid in the classification and identification of new species.

Outdoor enthusiasts: People who enjoy hiking, camping and foraging for mushrooms could use a CNN model to identify mushrooms found in nature. This could help them to avoid accidentally consuming poisonous mushrooms.

Environmental groups: Environmental groups could use a CNN model to monitor and identify mushroom species to protect and conserve them.

People who are interested in foraging for mushrooms and those working in the field of agriculture, food safety and health can benefit from a solution to this problem. The model can help identify the type of mushroom and whether it is safe to consume. The model can also be used as a reference guide to help identify species of mushrooms that are rare or difficult to recognize. Additionally, the model can help to detect harmful or poisonous mushrooms, and prevent accidental poisoning, thereby saving lives.

Related Literature

Convolutional Neural Network (CNN) is widely used in various applications such as image classification, object detection, and segmentation. CNNs have also proven themselves useful as a promising method to automatically and accurately identify and classify different species of mushrooms.

The identification of mushrooms is a challenging task because of the diversity of appearances and complex characteristics of different species of mushroom. Traditionally, mushroom identification relies on morphological and anatomical characteristics of the mushroom, which is time-consuming, reliant on expertise, and most importantly, can be subject to human error. With the use of CNN in mushroom identification, it can provide an accurate and more efficient method for identifying mushrooms.

A study by Preechasuk et al. (2019) applied CNN to classify 10 different types of poisonous mushrooms and 35 types of edible mushrooms. The authors used the dataset of mushroom images and implemented a CNN-based model to identify the type of mushroom. The results showed that the authors achieved an accuracy of 74.00% of classifying 45 different types of mushrooms.

Another study conducted by Zhang et al. (n.d) used CNN to identify poisonous and edible mushrooms most prominent in China. The authors crowd-sourced their dataset composed of edible and poisonous mushrooms gathered from the internet - they collected 250 poisonous mushrooms and 200 edible mushrooms. Additionally, they compared different models such as ResNet50, AUC, precision, recall, F1, and DenseNet121. ResNet50 got the highest accuracy with 75%.

Ketwongsa et al. (2022) aims to identify 5 different species of poisonous and edible mushrooms common in Thailand. The dataset used in making the CNN is composed of 5 species and 623 images. Results of their proposed model showed 98.5% accuracy in classifying 5 different species of mushroom.

The usage of CNN in mushroom identification has shown promising results in terms of its accuracy and efficiency. The deep learning framework is a reliable and efficient method for both identification and classification of mushrooms, making it a viable tool for real-life applications such as food safety, environmental monitoring, and scientific research.

A similar study to this paper in 2017 by Koivitso, et al. has the goal to develop a smartphone application that classifies wild mushrooms as either edible or not edible using a convolutional neural network model. To train the classifier, large amounts of pictures of classified mushrooms are required. The input images are normalized in size and shape, and only the appearance of the mushrooms is used for classification, without considering other factors such as smell and location. The CNN performs implicit feature extraction through its convolutional layers, and the network structure is defined in such a way that it is able to extract the information necessary for the classification task. Instead of providing a binary output, the user will be provided with the probability of the mushroom being edible, which will help them make a decision on whether to collect it for further research. The aim is to provide a tool for deciding whether to collect a mushroom, not for deciding whether to eat it. This study provided us insight into the different parameters that could be useful in tackling our specific problem.

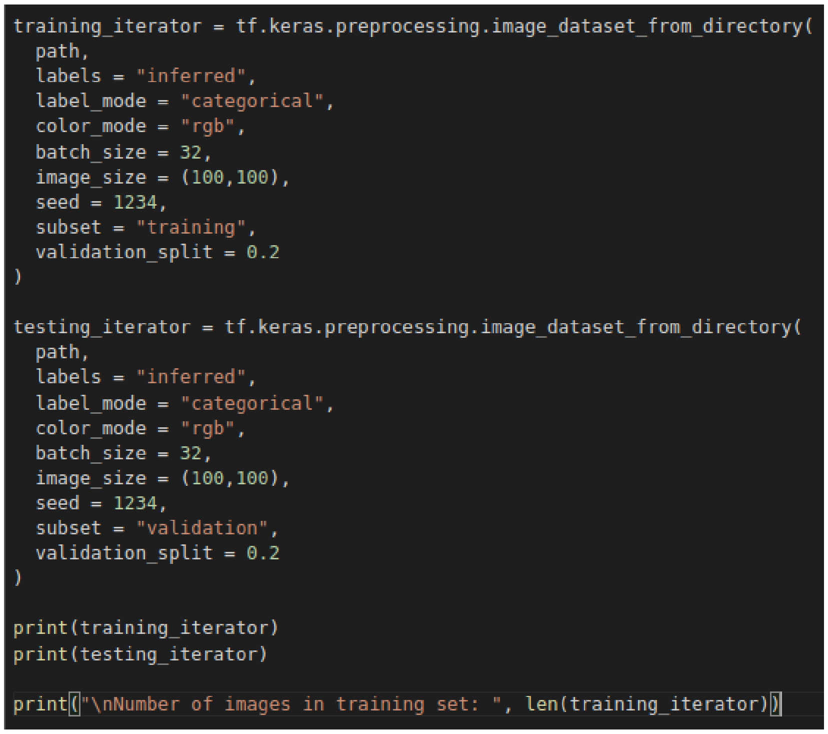
One article by Wang, B. (2022) discusses the significance of fungi in the ecological systems and the fact that only a small percentage of the estimated 3.8 million fungal species have been discovered. The article specifically mentions three species of macro-fungi - Morchella, Tuber melanosporum, and Cantharellus cibarius - which produce mushrooms and are classified based on their properties such as being edible, medicinal, or poisonous. The article highlights the importance of edible mushrooms as a diverse group of fungi. This goes to show how much a study into this field could help with the identification of different types of mushrooms.

Lastly, a study by Lee, J. et al, (2022) goes into how hallucinations, illness, and even death are common outcomes of hazardous mushroom exposure and intake situations all around the world. One contributing cause is that it can be challenging for general public collectors to tell some dangerous mushrooms apart from their edible equivalents. The goal of the project is to develop a smartphone application to classify mushroom types based on images taken in the field using a convolutional neural network (CNN). The app aims to help people distinguish edible from poisonous mushrooms, as harmful mushroom consumption can cause serious health issues. The study showed that the CNN-based classification method was able to achieve high sensitivity and specificity (ranging from 89% to 100%) in classifying mushrooms into two-, three-, and five-classes using field images with diverse backgrounds. The study also developed an Android app that transfers the trained model from the server and provides users with probability scores for the correct genus classification.

Discussion of the Experiment

Splitting and Preprocessing the Data

In splitting the dataset, it wasn't simple seeing as the folder structure of the dataset contains the 4 main classes, but each of those classes contained the folders of individual mushroom species. The researchers tried looping through the directory using os.walk() and storing the file paths in an array which would then be labeled with their appropriate classes. Eventually, the researchers landed on a more simple option which is shown in this code:



```
training_iterator = tf.keras.preprocessing.image_dataset_from_directory(
  path,
  labels = "inferred",
  label_mode = "categorical",
  color_mode = "rgb",
  batch_size = 32,
  image_size = (100,100),
  seed = 1234,
  subset = "training",
  validation_split = 0.2
)

testing_iterator = tf.keras.preprocessing.image_dataset_from_directory(
  path,
  labels = "inferred",
  label_mode = "categorical",
  color_mode = "rgb",
  batch_size = 32,
  image_size = (100,100),
  seed = 1234,
  subset = "validation",
  validation_split = 0.2
)

print(training_iterator)
print(testing_iterator)

print("\nNumber of images in training set: ", len(training_iterator))
```

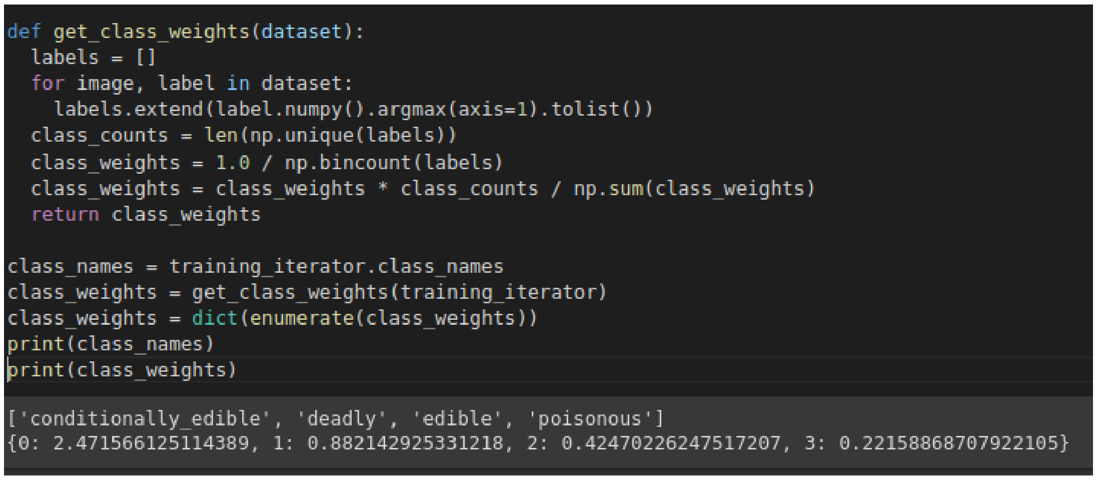
Code for splitting the dataset into training and testing sets

The code is splitting the data into two sets: a training set and a testing set. The training_iterator is created using the preprocessing.image_dataset_from_directory function and is used to specify the training data. The function takes several parameters such as the path to the directory where the images are stored, the labels for the images, and the type of labeling used. The color_mode is set to "rgb", the batch size is set to 32, and the image size is set to 100x100. The seed parameter is set to 1234 to ensure that the random number generator used to shuffle the data is reproducible. The subset parameter is set to "training" to indicate that this iterator is for the training set. The validation_split parameter is set to 0.2, which means that 20% of the data will be used for validation.

The testing set is created in a similar way, with the only difference being the subset parameter is set to "validation". This indicates that the testing set consists of the data reserved for validation during the training phase. Finally, the code prints the training and testing iterators and the number of batches in the training set.

Calculating the Weights for the Dataset

The dataset can be considered imbalanced seeing as the class with the lowest amount of images goes down to 400, while the class with the highest number of images goes upward to 4000, which could skew the results unfavorably. This was done by the 4th test, after realizing that the imbalance dataset could be giving bad results after training the model.



```
def get_class_weights(dataset):
  labels = []
  for image, label in dataset:
    labels.extend(label.numpy().argmax(axis=1).tolist())
  class_counts = len(np.unique(labels))
  class_weights = 1.0 / np.bincount(labels)
  class_weights = class_weights * class_counts / np.sum(class_weights)
  return class_weights

class_names = training_iterator.class_names
class_weights = get_class_weights(training_iterator)
class_weights = dict(enumerate(class_weights))
print(class_names)
print(class_weights)
```

```
['conditionally_edible', 'deadly', 'edible', 'poisonous']
{0: 2.471566125114389, 1: 0.882142925331218, 2: 0.42470226247517207, 3: 0.22158868707922105}
```

.

Code for computing the weights for each class

This code calculates the class weights of a given dataset. Class weights are used in training machine learning models to account for imbalanced datasets, where some classes have more samples than others. The higher the weight of a class, the more the model will focus on it during training. The code calculates the class weights by first extracting the labels of each image in the dataset, counting the frequency of each label, and normalizing the weights so that they sum up to 1.

The function get_class_weights(dataset) takes a dataset as input and returns the calculated class weights. The function first creates an empty list called labels to store the labels of each image. It then iterates through the dataset using a for loop, extracting the label of each image and converting it to a 1-dimensional numpy array using label.numpy(). The function then finds the index of the maximum value of the label array using argmax, and converts it to a list using tolist(). This index represents the class label of the image, and it is appended to the labels list.

Next, the function calculates the number of unique labels in the labels list using np.unique(), and stores it in the class_counts variable. It then calculates the frequency of each label in the dataset by dividing 1 by the number of occurrences of each label in the labels list, using np.bincount(). This frequency represents the weight of each class. The function then normalizes the class weights by multiplying them by the number of classes and dividing by the sum of the weights, so that they sum up to 1. Finally, the function returns the class weights.

The rest of the code gets the class names from the training dataset using the class_names property of the training dataset, calculates the class weights based on the frequency of each label in the training dataset using the get_class_weights() function, and converts the class weights to a dictionary where the keys are the indices of the labels. Finally, it prints the class names and class weights.

Testing and Fine-Tuning the Model

Legend

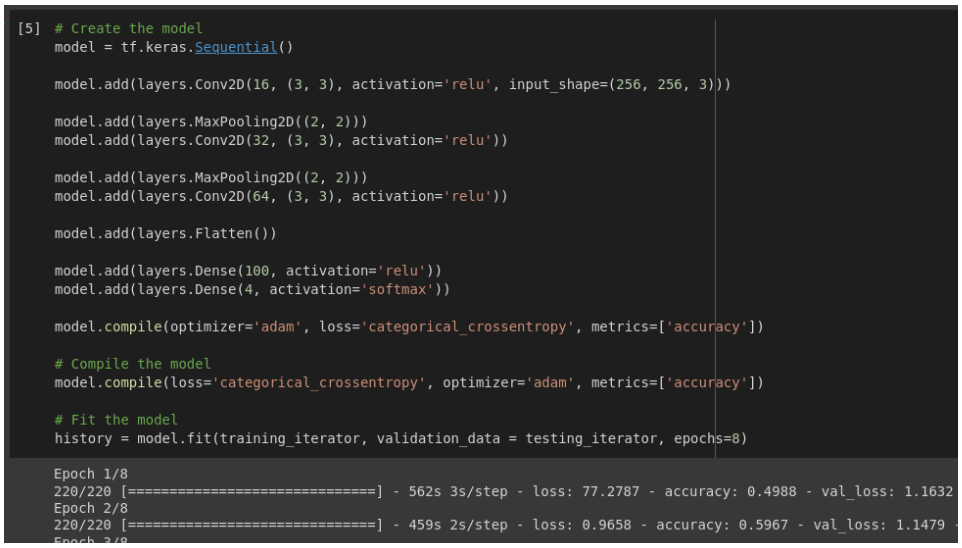
L = Loss Function

O = Optimizer

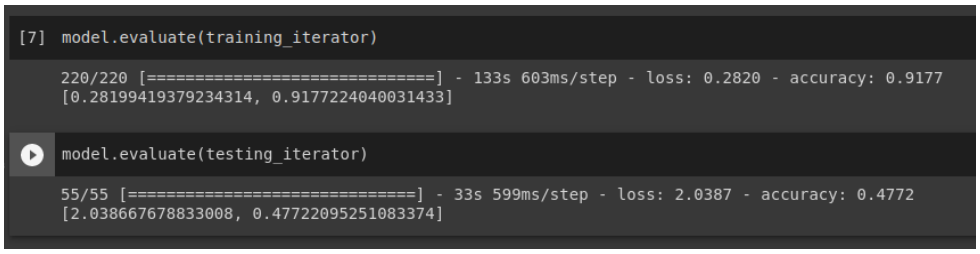
E = Epochs

First Test (Batch Size: 32)

- **L**: categorical_cross_entropy; **O**: adam; **E**: 8;
  - **Layer 1**: 16
  - **Layer 2**: 32
  - **Layer 3**: 64
  - **Fully Connected Layer 1:** 100
  - **Output Layer Nodes**: 4
  - **Accuracy**: 48%; **Loss**: 2.0387;



```
[5] # Create the model
    model = tf.keras.Sequential()

    model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(256, 256, 3)))

    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(32, (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(100, activation='relu'))
    model.add(layers.Dense(4, activation='softmax'))

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

    # Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Fit the model
    history = model.fit(training_iterator, validation_data = testing_iterator, epochs=8)

    Epoch 1/8
    220/220 [==============================] - 562s 3s/step - loss: 77.2787 - accuracy: 0.4988 - val_loss: 1.1632
    Epoch 2/8
    220/220 [==============================] - 459s 2s/step - loss: 0.9658 - accuracy: 0.5967 - val_loss: 1.1479 -
    Epoch 3/8
```



```
[7] model.evaluate(training_iterator)

    220/220 [==============================] - 133s 603ms/step - loss: 0.2820 - accuracy: 0.9177
    [0.28199419379234314, 0.9177224040031433]

    model.evaluate(testing_iterator)

    55/55 [==============================] - 33s 599ms/step - loss: 2.0387 - accuracy: 0.4772
    [2.038667678833008, 0.47722095251083374]
```

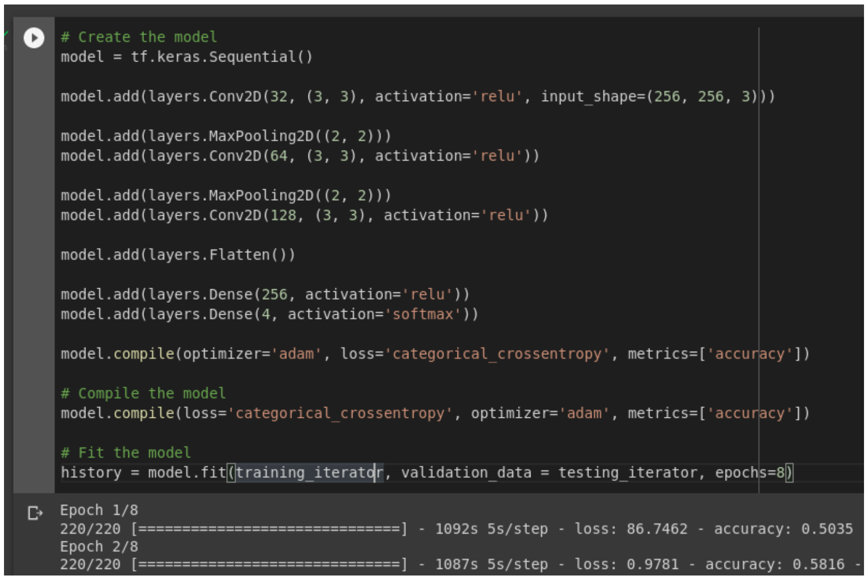
For this test, the researchers decided to start off small with only a few filters per layer, starting with 3 layers. The accuracy is low which might mean that this is not the right approach. The validation accuracy is not very high, which means that the model is not able to correctly predict the classes in the validation set. There are a few things that could be causing this low accuracy:

Imbalanced class distribution: As discussed earlier, the imbalance between the classes in the training set could be causing the model to have difficulty predicting some classes, but this hasn't been realized by the researchers at this point.

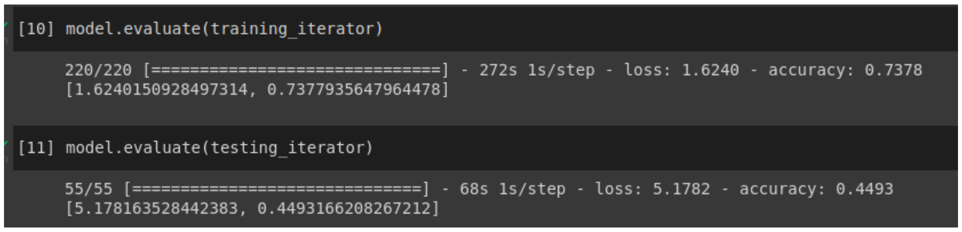
Overfitting: The model might be overfitting to the training data and not generalizing well to new data. This can be seen by the high accuracy on the training data but low accuracy on the validation data.

Second Test (Batch Size: 32)

- **L**: categorical_cross_entropy; **O**: adam; **E**: 8;
    - **Layer 1**: 32
    - **Layer 2**: 64
    - **Layer 3**: 128
    - **Fully Connected Layer 1:** 256
    - **Output Layer Nodes**: 4
    - **Accuracy**: 45%; **Loss**: 5.1782;



```
# Create the model
model = tf.keras.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))

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(128, (3, 3), activation='relu'))

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))

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

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
history = model.fit(training_iterator, validation_data = testing_iterator, epochs=8)

Epoch 1/8
220/220 [==============================] - 1092s 5s/step - loss: 86.7462 - accuracy: 0.5035
Epoch 2/8
220/220 [==============================] - 1087s 5s/step - loss: 0.9781 - accuracy: 0.5816 -
```



```
[10] model.evaluate(training_iterator)

     220/220 [==============================] - 272s 1s/step - loss: 1.6240 - accuracy: 0.7378
     [1.6240150928497314, 0.7377935647964478]

[11] model.evaluate(testing_iterator)

     55/55 [==============================] - 68s 1s/step - loss: 5.1782 - accuracy: 0.4493
     [5.178163528442383, 0.4493166208267212]
```

For this test, the researchers decided to increase the amount of filters per layer to see different results. However, the results seemed to have gotten worse, going down slightly from the first test, with roughly the same problem as the first test.

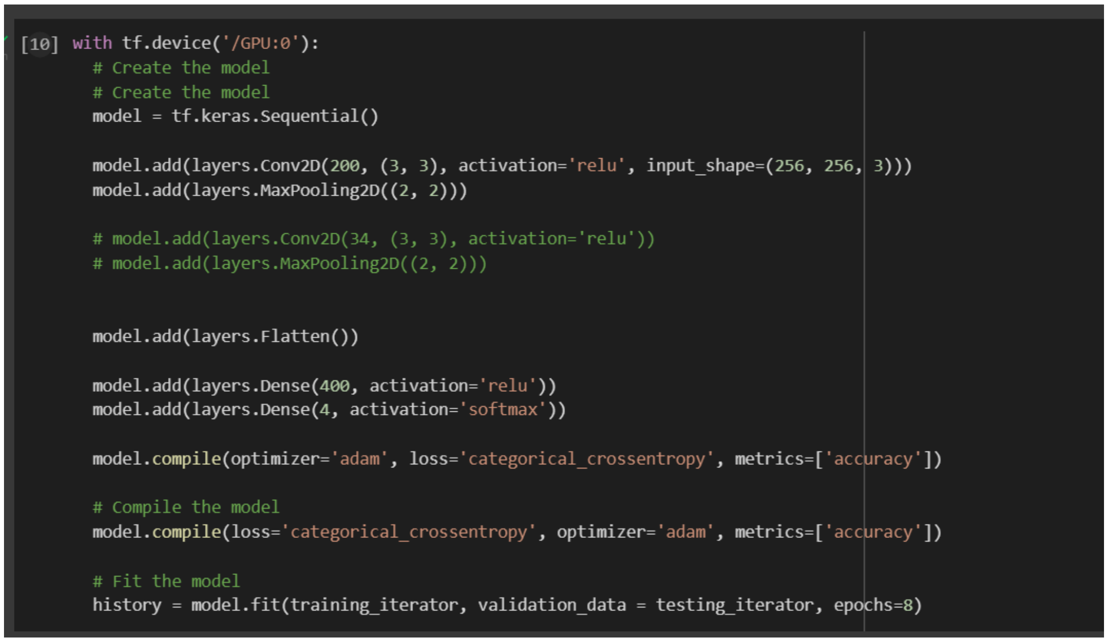
A decrease in the training loss and an increase in the training accuracy over the epochs suggest that the model is learning from the training data. In other words, the model is making fewer errors and improving its performance on the training data as the number of epochs increases.

However, the stability of the validation loss and accuracy values indicate that the model may not be generalizing well to unseen data. The validation accuracy value of around 0.45 suggests that the model is only making correct predictions in about 45% of the cases. This could again be due to overfitting, where the model has learned the training data too well and is not able to generalize to new data.

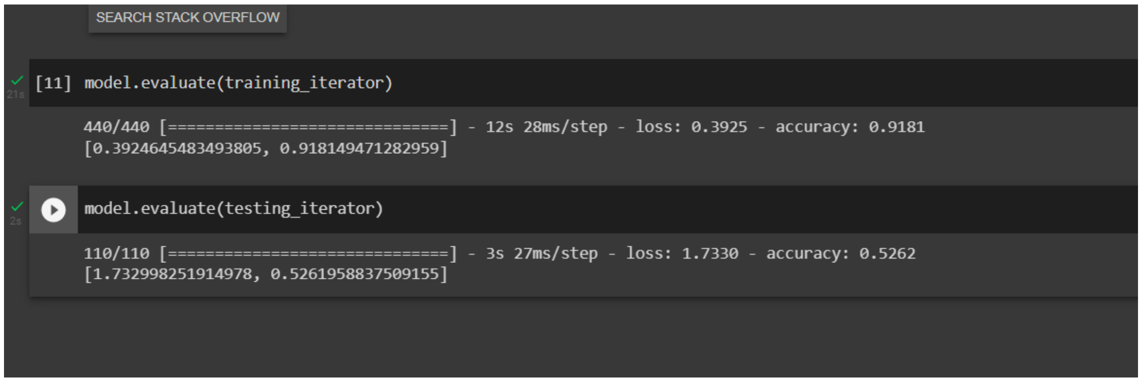
Another possibility is that the choice of model architecture is not appropriate for the problem. A different model architecture with fewer parameters or a simpler structure may be better suited for this problem.

Third Test (Batch Size: 16)

- **L**: categorical_cross_entropy; **O**: adam; **E**: 8;
  - **Layer 1**: 200
  - **Fully Connected Layer 1:** 400
  - **Output Layer Nodes**: 4
  - **Accuracy**: 53%; **Loss**: 1.7330;



```
[10] with tf.device('/GPU:0'):
       # Create the model
       # Create the model
       model = tf.keras.Sequential()

       model.add(layers.Conv2D(200, (3, 3), activation='relu', input_shape=(256, 256, 3)))
       model.add(layers.MaxPooling2D((2, 2)))

       # model.add(layers.Conv2D(34, (3, 3), activation='relu'))
       # model.add(layers.MaxPooling2D((2, 2)))


       model.add(layers.Flatten())

       model.add(layers.Dense(400, activation='relu'))
       model.add(layers.Dense(4, activation='softmax'))

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

       # Compile the model
       model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

       # Fit the model
       history = model.fit(training_iterator, validation_data = testing_iterator, epochs=8)
```

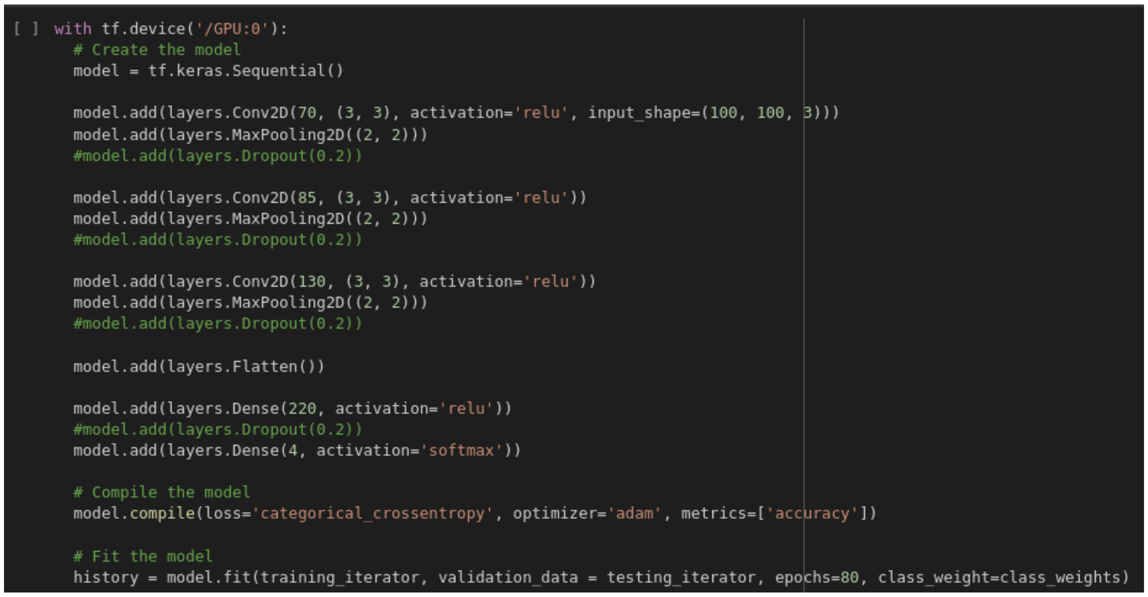


```
[11] model.evaluate(training_iterator)

     440/440 [==============================] - 12s 28ms/step - loss: 0.3925 - accuracy: 0.9181
     [0.3924645483493805, 0.918149471282959]

     model.evaluate(testing_iterator)

     110/110 [==============================] - 3s 27ms/step - loss: 1.7330 - accuracy: 0.5262
     [1.732998251914978, 0.5261958837509155]
```

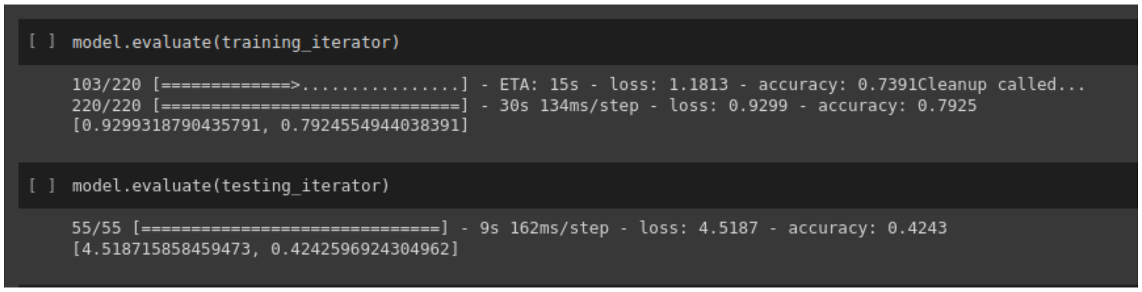
For this test, the researchers decided to cut down the amount of convolutional layers to see if the model will perform better under a different architecture. From the output, it is clear that the training and validation losses are not improving much after the first epoch, which suggests that the model may not be learning effectively. The training accuracy is also stuck at around 0.5262, which although is an improvement over the previous tests, this is still close to random guessing and is not appropriate for a problem as high stakes as identifying mushrooms for consumption. All further subsequent tests, testing different amounts of filters, layers and optimizers yielded the same exact accuracy percentage.

Fourth Test (Batch Size: 32)

- **L**: categorical_cross_entropy; **O**: adam; **E**: 80;
    - **Layer 1**: 70
    - **Layer 2**: 85
    - **Layer 3**: 130
    - **Fully Connected Layer 1:** 220
    - **Output Layer Nodes**: 4
    - **Accuracy**: 42%; **Loss**: 4.5187;



```
[ ] with tf.device('/GPU:0'):
      # Create the model
      model = tf.keras.Sequential()

      model.add(layers.Conv2D(70, (3, 3), activation='relu', input_shape=(100, 100, 3)))
      model.add(layers.MaxPooling2D((2, 2)))
      #model.add(layers.Dropout(0.2))

      model.add(layers.Conv2D(85, (3, 3), activation='relu'))
      model.add(layers.MaxPooling2D((2, 2)))
      #model.add(layers.Dropout(0.2))

      model.add(layers.Conv2D(130, (3, 3), activation='relu'))
      model.add(layers.MaxPooling2D((2, 2)))
      #model.add(layers.Dropout(0.2))

      model.add(layers.Flatten())

      model.add(layers.Dense(220, activation='relu'))
      #model.add(layers.Dropout(0.2))
      model.add(layers.Dense(4, activation='softmax'))

      # Compile the model
      model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

      # Fit the model
      history = model.fit(training_iterator, validation_data = testing_iterator, epochs=80, class_weight=class_weights)
```



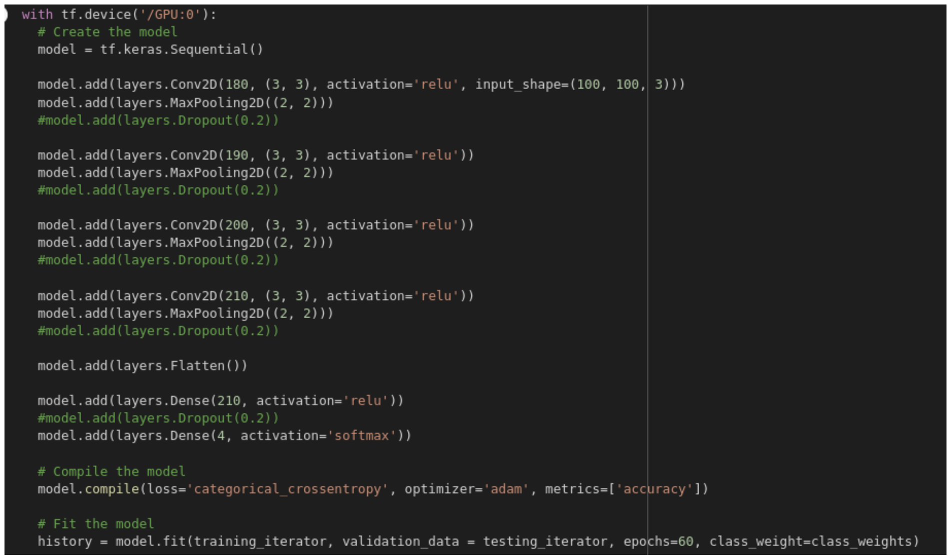
```
[ ] model.evaluate(training_iterator)

    103/220 [=============>...............] - ETA: 15s - loss: 1.1813 - accuracy: 0.7391Cleanup called...
    220/220 [==============================] - 30s 134ms/step - loss: 0.9299 - accuracy: 0.7925
    [0.9299318790435791, 0.7924554944038391]

[ ] model.evaluate(testing_iterator)

    55/55 [==============================] - 9s 162ms/step - loss: 4.5187 - accuracy: 0.4243
    [4.518715858459473, 0.4242596924304962]
```

This is the test where the researchers started computing for the weights of each class for use in the model in order for the dataset to have a balanced distribution when testing the model. The researchers also experimented with higher epochs seeing as the previous tests that resulted in low accuracy under roughly the same parameters as this test had continually rising accuracy, but were getting cut off by the low epoch value.

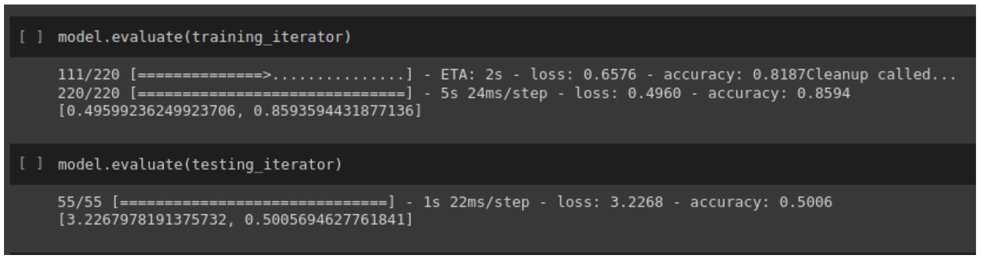
The output shows that the model is improving over time as the accuracy increases and the loss decreases on the training set, however the accuracy on the validation set is not improving significantly and is fluctuating around 0.4. The model could still be improved with the right parameters

Fifth Test (Batch Size: 32)

- **L**: categorical_cross_entropy; **O**: adam; **E**: 60;
    - **Layer 1**: 180
    - **Layer 2**: 190
    - **Layer 3**: 200
    - **Layer 4**: 210
    - **Fully Connected Layer 1:** 210
    - **Output Layer Nodes**: 4
    - **Accuracy**: 50%; **Loss**: 3.2268;

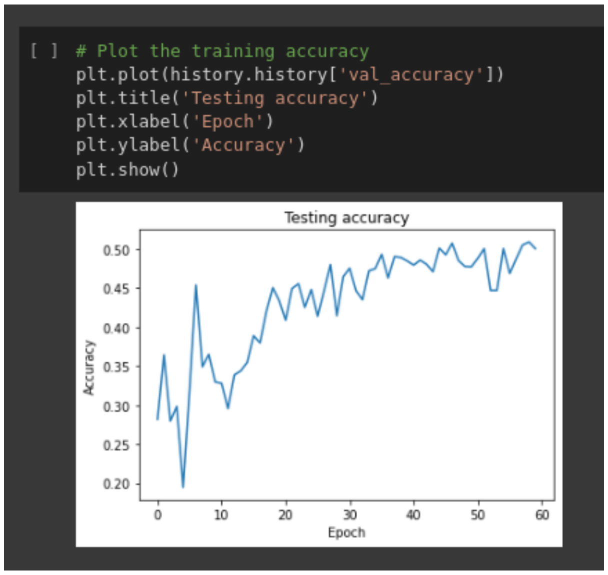


```
with tf.device('/GPU:0'):
  # Create the model
  model = tf.keras.Sequential()

  model.add(layers.Conv2D(180, (3, 3), activation='relu', input_shape=(100, 100, 3)))
  model.add(layers.MaxPooling2D((2, 2)))
  #model.add(layers.Dropout(0.2))

  model.add(layers.Conv2D(190, (3, 3), activation='relu'))
  model.add(layers.MaxPooling2D((2, 2)))
  #model.add(layers.Dropout(0.2))

  model.add(layers.Conv2D(200, (3, 3), activation='relu'))
  model.add(layers.MaxPooling2D((2, 2)))
  #model.add(layers.Dropout(0.2))

  model.add(layers.Conv2D(210, (3, 3), activation='relu'))
  model.add(layers.MaxPooling2D((2, 2)))
  #model.add(layers.Dropout(0.2))

  model.add(layers.Flatten())

  model.add(layers.Dense(210, activation='relu'))
  #model.add(layers.Dropout(0.2))
  model.add(layers.Dense(4, activation='softmax'))

  # Compile the model
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

  # Fit the model
  history = model.fit(training_iterator, validation_data = testing_iterator, epochs=60, class_weight=class_weights)
```



```
[ ] model.evaluate(training_iterator)

    111/220 [==============>..............] - ETA: 2s - loss: 0.6576 - accuracy: 0.8187Cleanup called...
    220/220 [==============================] - 5s 24ms/step - loss: 0.4960 - accuracy: 0.8594
    [0.49599236249923706, 0.8593594431877136]

[ ] model.evaluate(testing_iterator)

    55/55 [==============================] - 1s 22ms/step - loss: 3.2268 - accuracy: 0.5006
    [3.2267978191375732, 0.5005694627761841]
```

In this test, the researchers decided to add an additional layer and reduce the epochs to 60 to preserve some time and for the reason that the model gets diminishing results by the middle of the epoch range.



```
[ ] # Plot the training accuracy
    plt.plot(history.history['val_accuracy'])
    plt.title('Testing accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.show()
```

In general, it can be observed that the loss decreases and the accuracy increases over the epochs, although there is some fluctuation. The model is not yet well-optimized, as there is still a gap between the training accuracy and validation accuracy, and the validation accuracy is not improving much after the 20th epoch. Many other tests were done to improve upon this model, but none of the other parameters passed the 50% mark which might suggest flaws in the dataset that was used for this study.

Conclusion

Training CNNs for more complex datasets such as multi classification requires an immense amount of computing power. None of the members have strong enough GPUs to handle the CNN training, so the testing is initially done in Colab in order to take advantage of the computing resources available such as the GPU processing the Colab provides. Despite that, there are still limitations with the use of the GPU computing seeing as there are rate limitations and it gets disabled after a certain time period is reached. Later on, the model was trained using Kaggle which doesn't only provide a stronger GPU, but also provides more lenient limitations on using the GPU. Without the GPU processing, each epoch takes more than 20 minutes to complete.

Categorical_crossentropy and adam are used in order to have consistency with the results. Softmax Categorical Cross Entropy is the most suitable to the dataset seeing as all the other loss functions aren't suitable for multi class classification problems.

The results are stuck at around 45-50% even with different changes to the parameters. The accuracy goes up continually from the first few epochs, but starts plateauing around the 45% mark, eventually landing at 50%. The data was initially imbalanced, with the gap in the amount of images ranging from 400 (conditionally edible) up to 4000 (poisonous). This is addressed by computing weights to balance the dataset. The accuracy follows a more typical pattern when trained with the proper weights (starts small, then grows as the epochs pass). However, the accuracy still hovers around 45-50%, but there is a more steady increase with each change to the parameters. The loss throughout each test goes up as the epochs pass, the reason for this could be overfitting.

The model is found to not be satisfactory enough even for general use, only reaching at most an accuracy of 50% which is not acceptable when dealing with potentially dangerous mushrooms. The following can be concluded from these results:

Insufficient training data: The model may not have been trained on enough data to accurately identify the different classes of mushrooms.

Poor quality of training data: The quality of the training data might be low, with examples of the classes being mislabeled or inconsistent.

Overfitting: The model might have overfitted to the training data, memorizing the examples instead of learning general features to identify the different classes of mushrooms.

Inadequate architecture: The architecture of the CNN model might not be appropriate for the task, not having enough layers or capacity to accurately identify the different classes of mushrooms.

Even though the model could be more improved upon, the researchers couldn't get better results due to time constraints and limitations within the dataset considering the amount of time it takes to conduct a test trial with the limitations of using CPU computation to conduct the tests. On the other hand, while GPU computing provides faster results, there are still limitations with Colab and Kaggle only giving users a limited amount of time to use the GPU resources which left the researchers having to use CPU for the majority of the experimentations. Despite the low accuracy of 50%, this study could still have value in several ways:

Benchmark: The study can be used as a benchmark for comparison with future studies. The model can serve as a baseline for improvement and can demonstrate the complexity of the task at hand.

Insight: The study may still provide insight into the difficulties and limitations of using CNNs for mushroom classification, which can inform future research efforts in the field.

Feature Engineering: The study can inform feature engineering efforts, such as the selection of appropriate images and the methods used to extract meaningful information from those images.

Data limitations: The low accuracy could be attributed to limited or unrepresentative data, which may highlight the need for larger and more diverse data sets for training and testing.

Overall, a study with a low accuracy can still be valuable in providing information and guidance for future research efforts.

Recommendations

Although the model isn't considered a success, there are insights that could be gathered from this research for future studies into this type of model. First off, data simplification could help with processing the data as there are fewer factors for the model to take into account. For example, the data could've been simplified to just 2 classes: edible and poisonous seeing as these are the common variants of mushrooms that could be found in the wild.

Something else to take into consideration is checking if the training data is insufficient, seeing as the model in this research may not have been trained on enough data to accurately identify the different classes of mushrooms. There is also the concern of using poor quality of training data. The quality of the training data could be potentially low, with examples of the classes being mislabeled or inconsistent. It is important to keep in mind the quality of the training data to have better results.

Furthermore, the use of GPU as the source of computing power is recommended rather than CPU computing for faster performance and processing as the training of data is a time consuming effort.

References

Ketwongsa, W., Boonlue, S., & Kokaew, U. (2022). A New Deep Learning Model for the Classification of Poisonous and Edible Mushrooms Based on Improved AlexNet Convolutional Neural Network. Applied Sciences, 12(7), 3409. MDPI AG. Retrieved from http://dx.doi.org/10.3390/app12073409

Koivisto T., Nieminen, T. & Harjunpää, J. (2017). Deep Shrooms: classifying mushroom images. Retrieved from https://tuomonieminen.github.io/deep-shrooms/

Kung-Williams, D. (2022). *Mushrooms A collection of edible and inedible mushrooms.* Kaggle. DOI: 10.34740/kaggle/dsv/3690937 Retrieved from https://www.kaggle.com/datasets/derekkunowilliams/mushrooms

Machine Learning-Based Classification of Mushrooms Using a Smartphone Application Lee, J. J., Aime, M. C., Rajwa, B., & Bae, E. (2022). Machine Learning-Based Classification of Mushrooms Using a Smartphone Application. Applied Sciences, 12(22), 11685. MDPI AG. Retrieved from http://dx.doi.org/10.3390/app122211685

Preechasuk, J., Chaowalit, O., Pensiri, F., & Visutsak, P. (2019). Image Analysis of Mushroom Types Classification by Convolution Neural Networks. Proceedings of the 2019 2nd Artificial Intelligence and Cloud Computing Conference. doi:10.1145/3375959.3375982

Wang B. (2022) Automatic Mushroom Species Classification Model for Foodborne Disease Prevention Based on Vision Transformer. Journal of Food Quality. vol. 2022, Article ID 1173102, 11 pages. https://doi.org/10.1155/2022/1173102

Zhang, B., Li, Z. & Zhao, Y. (n.d.) USING DEEP CONVOLUTIONAL NEURAL NETWORKS TO CLASSIFY POISONOUS AND EDIBLE MUSHROOMS FOUND IN CHINA. Retrieved from https://arxiv.org/pdf/2210.10351.pdf