Name That Mushroom!

The world of mycology is a veritable universe unto itself, in the soil around all of us. Forest floors invite exploration and discovery for young and old, novices and experts. And the world of mushrooms is diverse enough that even a seasoned mushroom hunter will find mushrooms she can't always identify. Mushrooms are at once alluring and anxiety-causing; the right mushroom can be delicious, but the wrong mushroom can be a toxic experience. Imagine if a computer could help you identify which mushroom you're looking at? In this project, we will train a machine learning model on images of different common mushroom genus, to see if it can be like having a mushroom expert in your pocket!

1. Data

The series of mushroom images comes from the Kaggle website, link below:

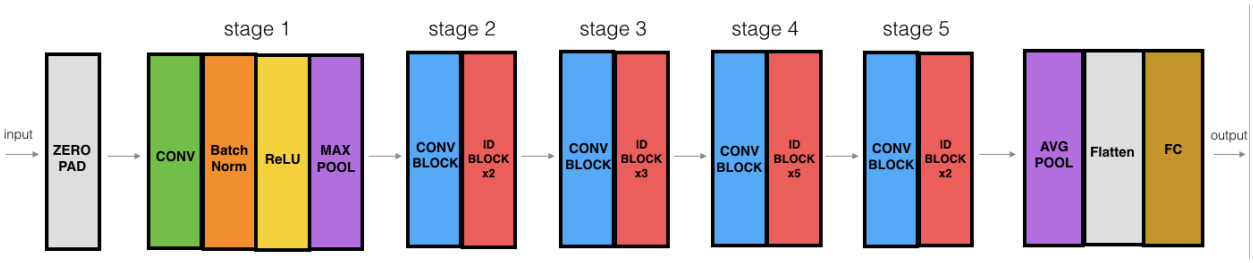
- Kaggle Dataset

2. Method

Neural networks, with their ability to identify and look for patterns in visual data, are particularly useful for image classification. Two neural network types are explored:

1. **Convolutional Neural Network (CNN):** Convolutional methods "convolve" the individual pixel data, aggregating them in useful ways for pattern recognition.

2. **Residual Neural Network (ResNet):** Extremely deep learning (i.e., many layers) suffer from issues with vanishing gradients. ResNet algorithms have been in use for a while, but in the last several years have gained prominence for their ability to deal with vanishing gradients with techniques such as skip connections, allowing deeper learning and greater classification accuracy.



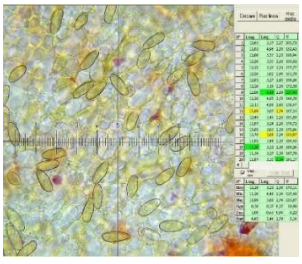
The ResNet50 Model

3. Data Cleaning

The Kaggle dataset is pre-organized into subfolders, each representing a common mushroom genus. While the majority of the images are perfectly fine for use, there are a couple data cleaning opportunities:

- **Problem 1:** The original dataset contains images with writing, highlights, arrows and circles, probably gleaned from books and websites about the subject of mushroom classification. While a human can ignore these image features as clearly non-natural, a computer may not be able to.

- **Problem 2:** The original dataset contains images, some microscopic, of spores of the genus types.



- **Problem 3:** Several images which show sliced mushrooms. While these are not a natural form for mushrooms in the forest, different mushrooms can have different discolorations when cut and exposed to air. Ultimately, I decided to leave these images in the dataset to preserve this potentially useful, if confusing, information.

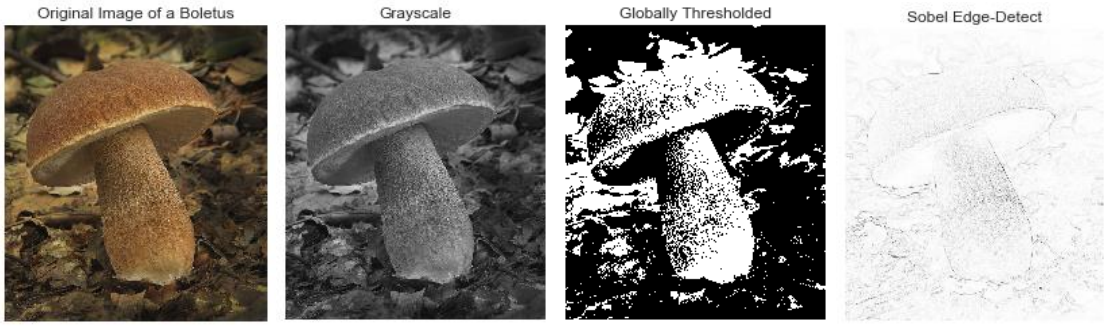
4. EDA

EDA Report

Exploratory data analysis consisted of preprocessing the images to a common size format. All images in the dataset are already color images with RBG channels.
The images came in a wide variety of image sizes and aspect ratios, however; some wider than tall, some taller than wide.

Looking at the images, mushrooms were fairly well centered, which presented an opportunity for cropping each image around the center, as shown below:

Also explored were manipulations of images including grayscale, thresholding and edge detection (using the sobel algorithm):



In the end, these grayscale manipulations didn't improve accuracy and hence didn't warrant use.

Returning to the color images, the dataset presented two significant challenges for any classification algorithm. (For human beings, too!)

First, **Variety Within Class:** Mushrooms within a single genus can vary significantly in how they present themselves. A great example of this is the enormous variation that can be found within the Hygrocybe genus:

Similarity Across Class: Kind of a corollary to all the variation within a genus, is that two different genera can end up looking pretty similar to each other. Here's a few left vs. right examples to illustrate this. Sometimes, a Suillus can look like a Boletus; a Lactarius can look like a Russula; and an Amanita can look like an Entoloma. Certainly, even a human would be confused by these similarities!

5. Splitting the Dataset

The various training, validation and hold-out test sets are constructed as follows:

1. Data Augmentation: Create sibling images of each image in the dataset using flipping and rotation schemes from the Pillow library
2. Use Python for-loops, lists and Numpy to read each image in sequence as a new row in an array (“training_data”). Alongside, a separate array (“training_labels”) encoding each row with its original category, i.e., the subfolder the image came from.
3. Use SciKit-Learn train_test_split to randomly shuffle the labelled data and assign a certain percentage (in this case, 10% of the original data), which the model can never see during training, as a hold-out test set after model training.
4. One-Hot encode the label arrays in both the training and test sets.
5. Using TensorFlow, compile the model to be tested.
6. Finally, train the model on the training set. During training, set aside a certain percentage (in this case, 20% of the training set) of the training set as a validation set. Stop the algorithm when the validation set accuracy fail to improve for numerous iterations (i.e., plateaus), using EarlyStopping callbacks.

6. Machine Learning

To explore the kind of model required for a problem of this scope, we’ll take a look at three models, ranging from simple to complex: We’ll start with a dummy stratification regression, then use a Convolutional Neural Network, then finally the ResNet50 model already described.

6a. Dummy Stratification

This model simply randomly assigns one of the seven categories to each image. When you flip a coin, the percentage of correct guesses will be close to 50%. When you are flipping a seven-sided coin, however, a correct guess will be much rarer. Indeed, this model yields the following accuracy (roughly 14%) on the test set:

```
0.13902439024390245
```

6b. Convolutional Neural Network

This model uses convolutions of local groups of pixels in an image, in an attempt to look for patterns. This kind of model has demonstrated to be quite effective with simple multi-class problems like MNIST handwritten digits and two-class mushroom problems. The model results in about 130,000 trainable parameters:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 298, 298, 5)       140
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 149, 149, 5)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 147, 147, 15)      690
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 73, 73, 15)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 15)        2040
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 35, 35, 15)        0
_________________________________________________________________
flatten (Flatten)            (None, 18375)             0
_________________________________________________________________
dropout (Dropout)            (None, 18375)             0
_________________________________________________________________
dense (Dense)                (None, 7)                 128632
=================================================================
Total params: 131,502
Trainable params: 131,502
Non-trainable params: 0
```

In the case of seven mushroom categories, this model seems to lack the ability to discern enough useful patterns. It ends up not doing a whole lot better than the random guess. The fact that this model does well with two-category classification exercises suggests that we would expect this model to perform better if we had access to a greater number of training image.

```
Model performance on test images:
Accuracy = 0.24390244483947754
Loss = 3.2920670974545363
```

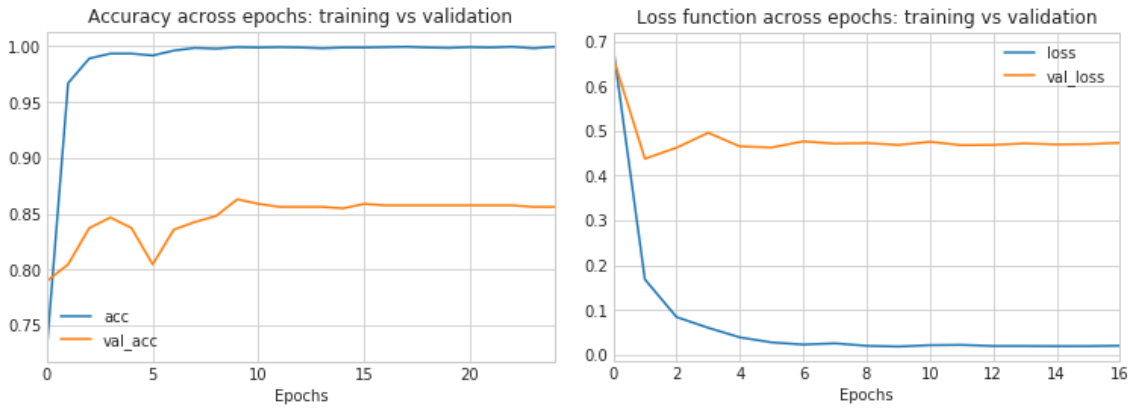
6c. The ResNet50 Model

For this model, the a ResNet50 model weighted for the ImageNet database was added to several Batch Normalization and trainable Dense layers.

The resulting model has over 24 million parameters, and over half a million trainable parameters. The model summary is as follows:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 2048)              23587712
_________________________________________________________________
batch_normalization (BatchNo (None, 2048)              8192
_________________________________________________________________
dense_1 (Dense)              (None, 256)               524544
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 903
=================================================================
Total params: 24,155,783
Trainable params: 563,207
Non-trainable params: 23,592,576
```

Training accuracy and loss functions are shown below, colored for the training set and the validation set.



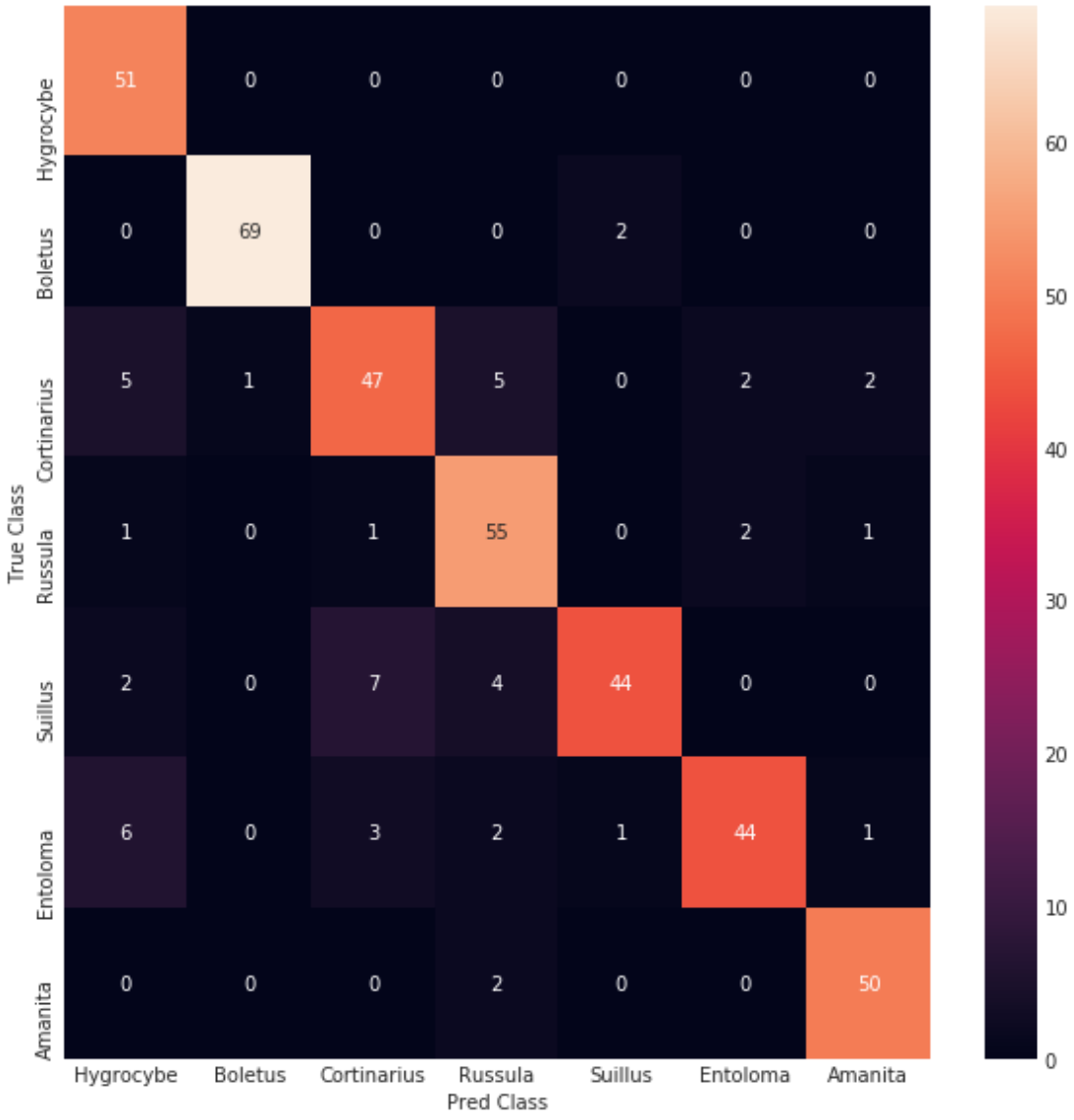
7. Predictions

The model yields the following accuracy on a hold-out test set:

```
Model performance on test images:
Accuracy = 0.8780487775802612
Loss = 0.37541265342293717
```

To be sure, in many computer vision exercises, a significantly higher accuracy can be obtained, often well in excess of 90%. The mushroom image set presents some fairly unique challenges, as has been discussed above. Indeed, it's fair to say that 88% accuracy might be as good as what any true human expert could be expected to achieve based on images of mushrooms alone (i.e., they don't get to touch, smell, cut, or inspect the surroundings of a mushroom, which are clues a mycologist often uses in the identification process.)

A confusion matrix heatmap for the hold-out test set is shown below. Rather than randomly distributed error, the confusion matrix shows particular identification issues; for example, thinking too many images are Hygrocybes, and having a hard time finding all the Cortinarius and Entoloma mushrooms.



A breakdown of each mushroom genus looks like this. On an individual genus basis, accuracy looks quite good. The model seems especially good with Boletus and Amanita mushrooms. Which is good, because Boletus are delicious, while Amanita are poisonous!

	TP	FP	TN	FN	Accuracy	Precision	Recall	F1-score
Hygrocybe	51.0	14.0	345.0	0.0	0.965854	0.784615	1.000000	0.879310
Boletus	69.0	1.0	338.0	2.0	0.992683	0.985714	0.971831	0.978723
Cortinarius	47.0	11.0	337.0	15.0	0.936585	0.810345	0.758065	0.783333
Russula	55.0	13.0	337.0	5.0	0.956098	0.808824	0.916667	0.859375
Suillus	44.0	3.0	350.0	13.0	0.960976	0.936170	0.771930	0.846154
Entoloma	44.0	4.0	349.0	13.0	0.958537	0.916667	0.771930	0.838095
Amanita	50.0	4.0	354.0	2.0	0.985366	0.925926	0.961538	0.943396

8.Future Improvements

a) With more time, it may be possible to break through the 90% accuracy level. While the exact method that will accomplish this is not known, I suspect that preventing overfitting of ResNet50 on training data may prove fruitful.
b) Experimenting with further image manipulations, such as superimposing a sobel edge detect image onto a standard RGB image. This might possibly allow the model to better identify the mushrooms in an image.
c) As a follow on to note (b) above, it may be possible to count the number of mushrooms in an image using edge detect algorithms like sobel or canny. If possible, that might be a useful feature to add to the dataset.
d) And, as always, procuring more high-quality images data of these mushroom classes! This might enable the far simpler CNN model to perform well.

9. Credits

Many thanks to Springboard and Paperspace for access to their Gradient GPU environments which allowed these highly complicated neural network models to train quickly, allowing the great deal of iteration that was required to tune the model performance. Thanks also to Branko Kovac for his encouragement and excellent counsel during this project!