Machine Learning Nanodegree

Capstone project

author: Michael Kögel
date: 8th September, 2018

Project report

Definition

Project Overview

As described in the project proposal, the aim of the project is find a solution to classify species of mushrooms from images with Convolutional Neuronal Network (CNN). The idea was based on the project Deep Shrooms from the University of Helsinki, in which three students developed a smartphone application to classify wild mushrooms to edible/not edible based on image input from the users camera. Such a project has a high potential in terms of application. It can help in the search for edible mushrooms, but it also carries dangers with regard to false positive (false edible) results.

This project differs in case of application in comparison to Deep Shrooms. The aim of the CNN is not about the decision whether a mushroom is edible or not. Instead the application should help to understand the biodiversity of mushrooms in Central European forest areas and help users to distinguish between mushroom species.

Problem Statement

Identifying mushroom species requires a basic understanding of their macroscopic structure. Separating species requires meticulous attention to detail. There is no single trait that mushrooms can be separated into single species. For this purpose, a recognition system is needed.

The main task of my project is to create an application which classifies user inputted pictures of wild mushrooms according to their species, using a machine learning classifier based on a CNN. In result the application calculates the probability for each species of mushrooms, the image may belongs to.

In the first part I explain how the data was collected from databases / websites and their subdivision into groups of their species. After the data collection and data preprocessing part the report includes how well an Deep Residual Network is able to detect mushrooms in my collected images (regarding to the chapter 5 project from the course).

In the first analytical part of my project the preprocessed and transformed data was used to train a small convolutional neuronal network and see how this network performs on the small dataset. In the second part the approach was to leverage a network, pre-trained on a large dataset. I used the bottleneck features of a pre-trained network and build a resulting more complex CNN. The last step includes fine-tuning of the fully connected layer and illustrate how the model can be further improved. At the end of the project, however, an outlook is given as to whether the application is better differentiated between edible and non-edible mushrooms.

Metrics

The problem described is a categorical classification problem. The model responds to an incoming image with the possible type of mushroom. I will evaluate the results by calculating the accuracy of a verification dataset. I will also show the improvement of training and validation accuracy and loss after each epoch of training.

To visualize the performance of the algorithm a confusion matrix for the tests of the CNNs will be used. This allows more detailed analysis than mere proportion of correct classifications. There is also a statement about the cross sensitivity between the species of mushrooms. An interesting evaluation would also be to distinguish the sensitivity and specificity between edible and non-edible mushrooms in the result.

Analysis

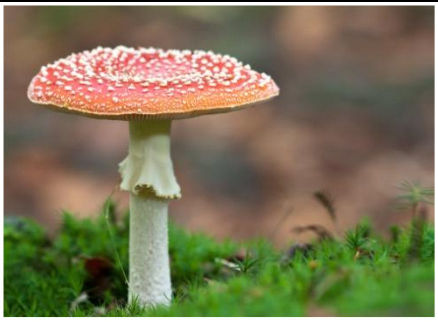
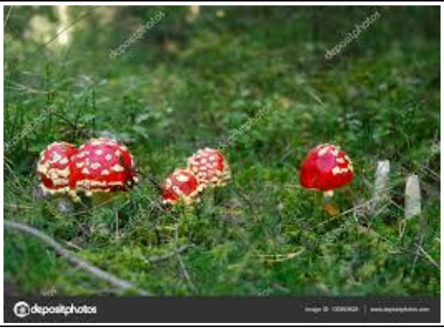
Data Exploration and Exploratory Visualization

The search for a suitable data set or the collection of data was the most time-consuming point of this project. For the beginning, the application should include at least 10 different species of mushrooms and I wanted only to use few training examples to build the image classifier.

I wasn't able to find a matching data record of images that met my requirements. These contained the following points:

- the minimum pixel size of 150 x 150 pixel
- only one mushroom species on the image
- the mushroom must fill between 1/3 to 2/3 of the full image
- a neutral background (no dominant environment elements i.e.: trees, white areas...)
- no text or markings on the image
- side view from above onto the mushroom
- only 1 to 2 mushroom on the image (was ignored if species comes only in groups)
- bright and well illuminated images

The following table provides a brief overview of good and bad images for the data set (see Table 1).

Image of mushroom	Explanation
	**Good image** • the mushroom fills a dominant part of the image • there is only one mushroom in camera focus • background is blurry and has a natural environment
	**Bad image** • to many mushrooms on the image • mushrooms only fills a low area of the image • there is a text at the bottom of the image • the position of view is good

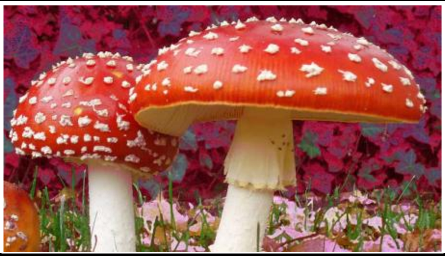
	**Bad image** • position of view is to low • the background is to dominant • the area filled by both mushrooms may is to large

Table 1: Good and bad images for the dataset

To find suitable images for the dataset I used the google images search. With the help of the browser add-on Fatkun Batch Download Image, I was able to collect a large number of images with little effort. In the picture below is the browser add-on with the selection of the images to download according to the user selection (Figure 1).

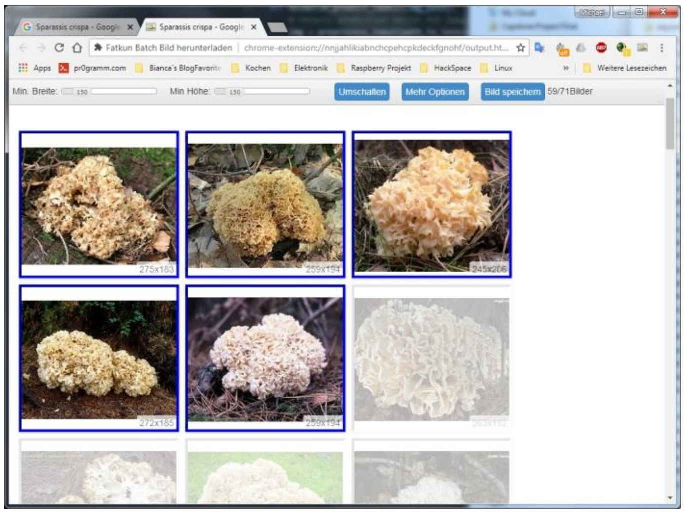


Figure 1: Fatkun Batch Download Browser Add-On with user selection

In result of the data collection part of this project we have:

- 1176 total images of mushrooms
- for training (976), validation (100) and test (100)
- divided in 10 subcategories (mushrooms species)

The data is organized in directories for training, validation and test containing subdirectories per image class (mushroom species):

```
data /
        train /
                001.Amanita muscaria /
                002.Amanita phalloides /
                003.Amanita pantherina /
                004.Galerina marginata /
                005.Boletus /
                006.Cantharellus cibarius /
                007.Gyromitra esculenta /
                008.Armillaria /
                009.Clitocybe /
                010.Sparassis crispa /

        valid /
                001.Amanita muscaria /
                002.Amanita phalloides /
                ...

        test /
                001.Amanita muscaria /
                002.Amanita phalloides /
                ...
```

Side Note: For indices 1 to 5, the mushroom species can be classified as not edible and 6 to 10 as edible. We will use this information at the end for comparison with the Deep Shrooms project.

Algorithms and Techniques

The goal of the project is to develop an application that is able to classify mushrooms according to their species based on images. To do this, the project presents a few effective methods that will be used to build a powerful image classifier, using only very few training examples. In the field of machine learning, CNN is one of the best methods to classify images with relatively little pre-processing.

The main task include the following three parts:

- evaluate the images from a pre-trained ResNET-50 network
- training a simple CNN with data augmentation
- using the bottleneck features of a pre-trained network and perfom fine-tuning

ResNet-50

In the first step of solving the classification problem, I used a pre-trained residual network ResNet-50 to determine the image class with the highest prediction value for every image of our dataset. The weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks.

ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories. Given an image, this pre-trained ResNet-50 model returns a prediction (derived from the available categories in ImageNet) for the object that is contained in the image. The network is 50 layers deep. The architecture can be seen in Figure 2.

My intention was to determine whether a large part of the images would be recognized as mushrooms. One result could be whether other elements on the pictures might have a higher influence on the classification (i.e.: the background). In this case, elements in the data set would have had to be filtered out additionally.

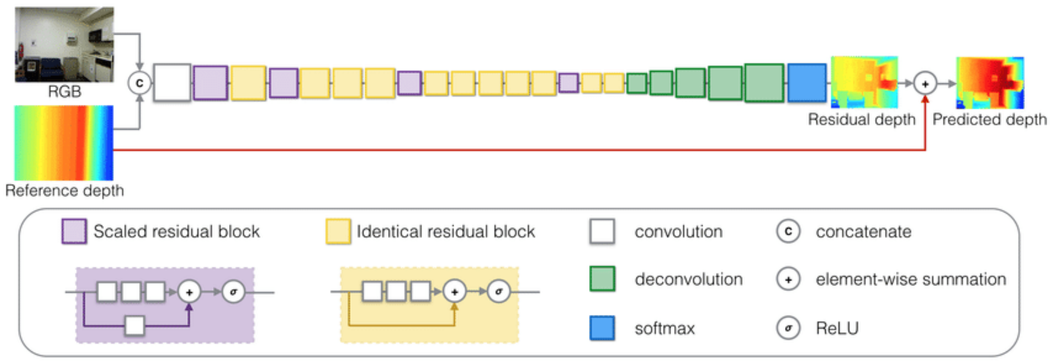


Figure 2: ResNET-50 architecture (https://www.researchgate.net/figure/Our-residual-of-residual-network-architecture-The-network-is-designed-based-on-the_fig1_309738522)

Simple CNN

As a initial baseline I trained a small CNN for the image classification problem. Since there were only a few numbers of training datasets, overfitting should be avoided. If the model exposed too few examples it learns patterns that do not generalize to the image data. The data augmentation step is a good process againt overfitting, but may isn't enough since the augmented samples are highly correlated in one batch.

That's why I have a special look at the entropic capacity of the choosen model (amount of information the model is able to store). A model that can store a lot of information has the potential to be more accurate by leveraging more features, but it is also more at risk to start storing irrelevant features. Meanwhile, a model that can only store a few features will have to focus on the most significant features found in the data, and these are more likely to be truly relevant and to generalize better.

I will use different ways to modulate entropic capacity. A main option is the value for the number of parameters in the mode (number of layer, size of layers). I will also use weight regularization like L1 or L2 regularization, which consists in forcing model weights to take smaller values.

Bottleneck Feature

Another procedure to increase the accuracy of the CNN is to use transfer learning, where a model trained on one task is re-purposed on a second related task. Transfer learning is an optimization that allows rapid progress or improved performance when modeling the second task.

Such a network would have already learned features that are useful for most computer vision problems, and leveraging such features would allow us to reach a better accuracy than any method that would only rely on the available data.

I used the VGG16 architecture, a model from the VGGNet, involving 144 million parameters, contains 16 convolutional layers with very small receptive fields pre-trained on the ImageNet dataset. The Imagenet dataset contains several objects of forms, shapes and figures from nature that are related to our problem. The architecture of the VGG16 can be seen in Figure 3.

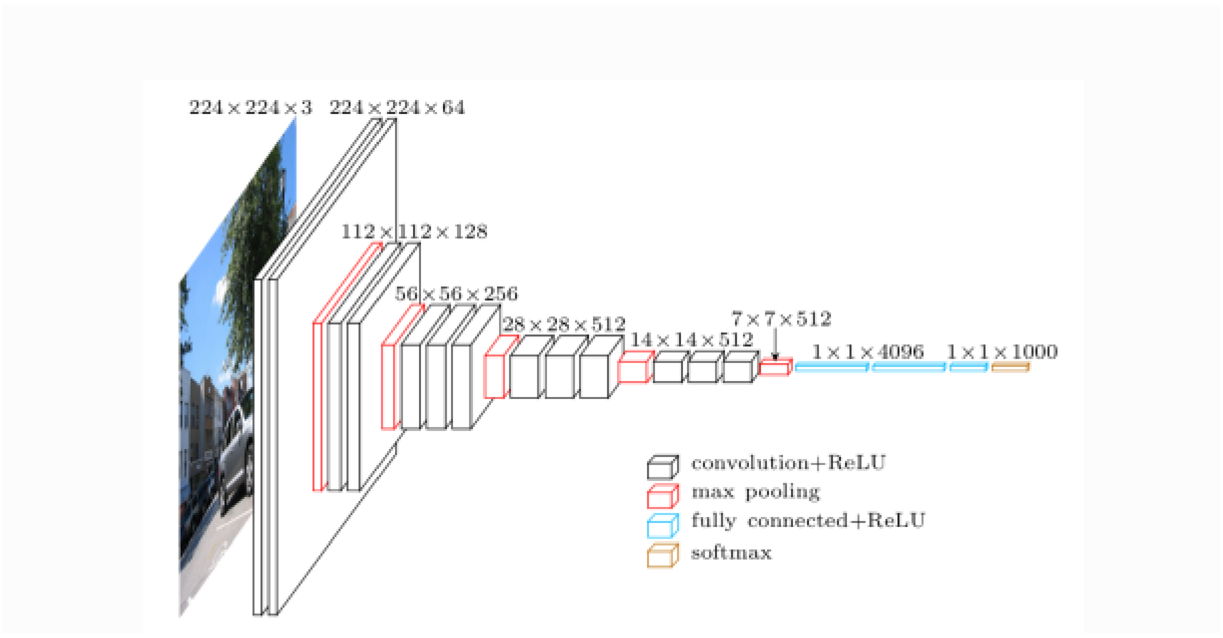


Figure 3: VGG16 architecture (https://www.quora.com/What-is-the-VGG-neural-network)

The strategy is the following: only the first layers, the convolutional part of the model, is instantiated up to the fully-connected layers. The training and validation data is then running once a time through the network and the output will be saved. This is what is called the bottleneck feature. A new fully-connected model on the top of the stored features is trained afterwards. The reason to store the features offline and then adding a new fully-connected model directly is the computational efficiency.

To further improve our previous result, a "fine-tuning" will be done on the last convolutional block of the VGG16 model alongside the top-level classifier. Fine-tuning consist in starting from a trained network, then re-training it on a new dataset using very small weight updates.

Benchmark

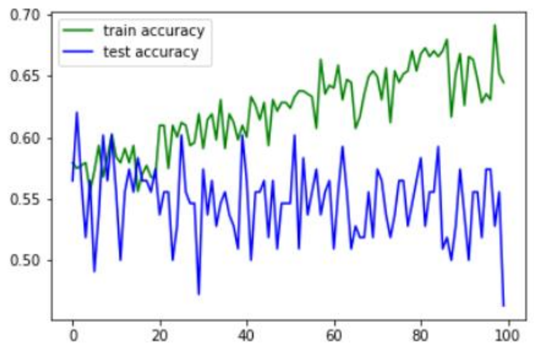
A step-by-step evaluation of the results takes place by comparing the individual models from the simple CNN to the fine-tuned bottleneck feature network in the last step. I will improve the accuracy on the test images by the following target plan:

- Simple CNN without data augmentation: accuracy > 70%
- Simple CNN with data augmentation: accuracy > 80%
- Using bottleneck feature from VGG16: accuracy > 90%
- Fine-tune bottleneck feature network: accuracy > 95%

Furthermore, the results of the classification models are compared with the Deep Shrooms project mentioned at the beginning. The main focus here is to achieve significantly higher accuracy.

Figure 4 include the results from the Deep Shrooms project trained with 536 images. The chart on the left shows both the training and testing accuracy for each of the epochs. It can be seen that while the training accuracy has an increasing trend, the testing accuracy averages a steady 55% accuracy, which is remarkably poor and the model seems to overfit the data. On right chart the histogram of the predictions of the models is shown. It contains the precision and recall for each prediction value. It shows that there is no clear area where the effect of false positive predictions is delimitable.

A higher accuracy and improvement of precision and recall is necessary for the upcoming model.



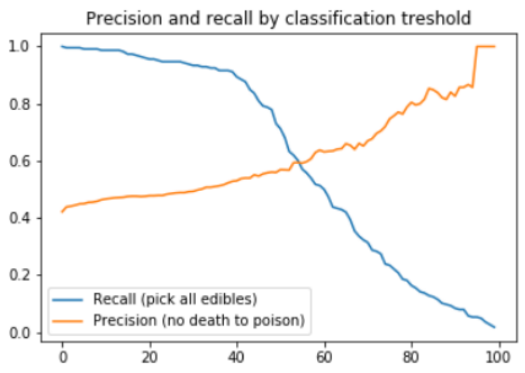


Figure 4: Accuracy / loss and precision / recall from Deep Shrooms results

Methodology

Data Preprocessing

Compared to ImageNet, our dataset is significantly less to train a neuronal network. Certainly, deep learning requires the ability to learn features automatically from the data, which is generally only possible when lots of training data is available - especially for problems where the input samples have high dimensions, like images.

However, convolutional CNNs are by design one of the best models available for most perceptual problems (such as image classification), even with very little data to learn from. Training a CNN from scratch on a small image dataset will still yield reasonable results, without the need for any custom feature engineering. For our job, is it the right tool.

In order to make the most of our few training examples, I used data augmentation to transform the data by a random number of transformations, so that the model would not see twice the exactly same image. This also helps against overfitting (higher generalization).

I used the "ImageDataGenerator" class from Keras which allows to configure random transformations and normalization operations to be done on your image data during training and instantiate generators of augmented image batches (and their labels). Table 2 include the parameter applied on the training data.

Parameter with value	**Description**
rotation_range=15	is a value in degrees, a range within which to randomly rotate pictures
width_shift = 0.1 height_shift = 0.1	randomly translate pictures vertically or horizontally direction by range
rescale = 1./255	is a value multiply the data before any other processing
shear_range = 0.1	is for randomly applying shearing transformations
zoom_range = 0.1	is for randomly zooming inside pictures
horizontal_flip = true	is for randomly flipping half of the images horizontally -- relevant when there are no assumptions of horizontal assymetry
fill_mode = 'nearest'	is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift

Table 2: Parameter for data augmentation

In Table 3 are the results applying the data augmentation strategy to an image from the training data set of an Amanita Muscaria mushroom image.

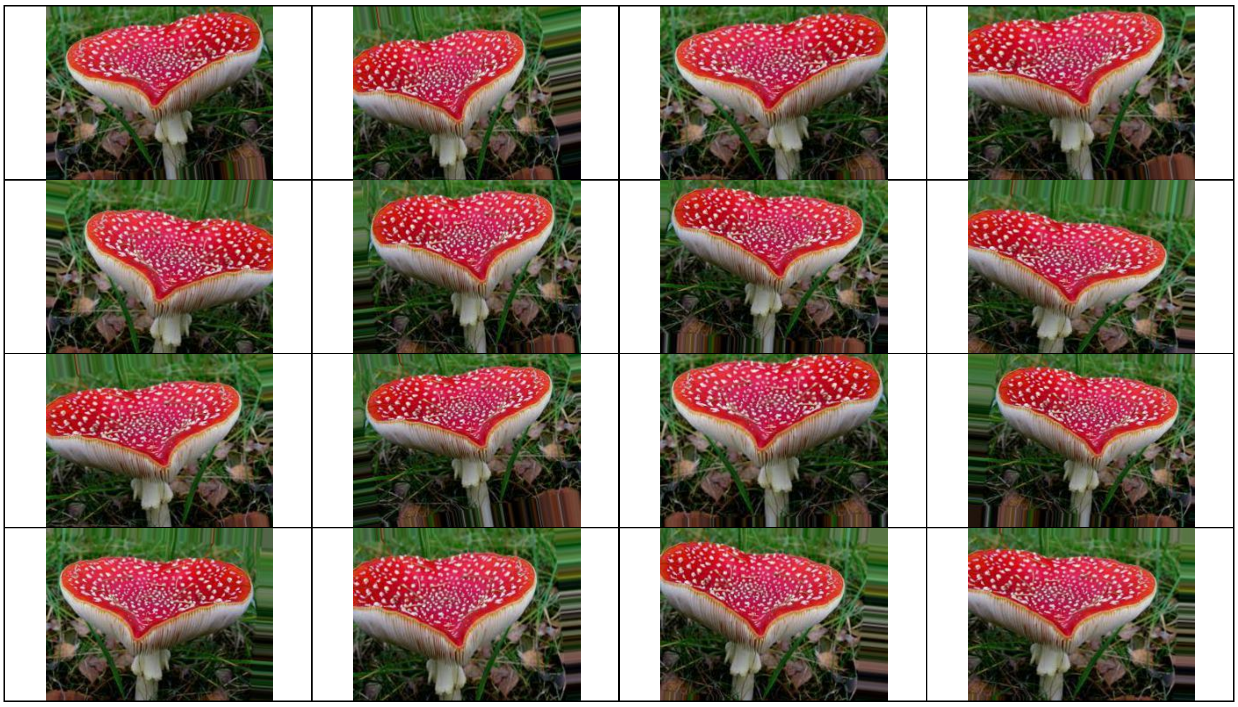


Table 3: Images of mushroom after augmentation process

Implementation

I used the Keras library with TensorFlow as backend. The CNNs in Keras require a 4D tensor as input. All used data arrays in the upcoming implementation used the following shape:

[number_of_samples,rows,columns,rgb_channel]

The number_of_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

I resized all images to square images with a size of 150 x 150 and normalized the pixel values for each color between 0 and 1.

ResNet-50

There is some additional preprocessing for the ResNet-50 pre-trained model in Keras required. In the first step the RGB image is converted to BGR and the pixel are additionally normalized to the same mean value of each image. The model and they trained weights from ImageNet platfrom are loaded and every image from our dataset gets an categorically prediction array from the model.

By taking the argmax of the predicted probability vector, I obtain an integer corresponding to the model's predicted object class, which we can identify with an object category through the use of the pertinent dictionary. The results can be seen in Table 4 in the next section.

Simple CNN

The simple CNN include a with few layers and few filters per layer, alongside with data augmentation and dropout. After each hidden layers I also add batch normalization. The idea behind that is during training to normalize the activations of the previous layer for each batch, i.e. apply a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. The convolutional layers starts with 16 filters and a kernel size of 3. On each convolutional hidden layer the number of filter are doubled up to 256. So for the first hidden layers the model is very sensitive to primitive structures in our images. With subsequent layers the recognized structures in the images become more complex from the primitive types. After each convolution layer we apply a Max-Pooling layer to reduce unnecessary information and to increase the performance of our CNN. It has also positive effects against overfitting.

This is what the layer configuration of the simple CNN looks like:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_1 (Batch (None, 150, 150, 3)       12
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 16)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 74, 74, 16)        64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 36, 36, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 17, 17, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 256)         295168
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 397,174
Trainable params: 396,176
Non-trainable params: 998
_________________________________________________________________
```

The parameter for training are the following (after optimization):

- batch-size: 64
- epochs: 20
- steps per epoch: 128

Bollteneck feature

The implementation of the CNN with usage of the bottleneck feature is done by downloading the VGG16 weights trained on the ImageNet dataset. Only the bottom layer up to the fully-connected layers of the network and their weights are loaded into the application. The mushroom dataset for training, validation and test were processed by the convolutional layers of the CNN. The output has been saved afterwards to train a small fully-connected model on top of the stored features.

This is what the layer configuration of the top model from bottlenck CNN looks like:

```
_________________________________________________________________
Layer (type)             Output Shape           Param #
=================================================================
batch_normalization_1 (Batch (None, 4, 4, 512)       2048
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)            0
_________________________________________________________________
dense_1 (Dense)           (None, 32)            16416
_________________________________________________________________
dropout_1 (Dropout)         (None, 32)            0
_________________________________________________________________
dense_2 (Dense)           (None, 10)            330
=================================================================
Total params: 18,794
Trainable params: 17,770
Non-trainable params: 1,024
```

The configuration of the top model is the following (after optimization):

- batch-size: 32
- epochs: 300

The network to be trained is much smaller, so there is much more performance in training epochs of the network.

Refinement

There are different ways to modulate entropic capacity of the CNN. The main one is the choice of the number of parameters in the model, i.e. the number of layers and the size of each layer. Another way is the use of weight regularization, such as L1 or L2 regularization, which consists in forcing model weights to taker smaller values.

To further improve the previous results, I will try to "fine-tune" the last convolutional block of the VGG16 model alongside the top-level classifier. Fine-tuning consist in starting from a trained network, then re-training it the new dataset using very small weight updates. This can be done in 3 steps:

- build the convolutional base of VGG16 and load its weights
- add the fully-connected model on top, and load its weights
- freeze the layers of the VGG16 model up to the last convolutional block
- retrain the network with the given changes

It is necassary that all layers should start with properly trained weights, because a large gradient of updates in the top layers could wreck the learning rates in the convolutional part of the network.

To prevent overfitting it is necassary to keep the first layers fixed, because they include a high entropic capacity and i dont have the CPU power to perform this high-level learning. I will also do fine-tuning by using a low learning rate I also did before in the CNNs.

Results

Model Evaluation and Validation

ResNet-50

The results are in Table 4. It can be seen that most of the images were recognized as mushroom or as specific mushroom species. That means that less than 1% from the dataset are primary classified as an unexpected object category (possible by a dominant background). The first category that has nothing to do with our project is the acorn and wooden spoon with less than 1% each.

Total number of Images for ResNet-50 prediction: 1176

Ranking	Number of Images	Relative Number	ResNet-Index	Category
1	298	25,3%	947	Mushroom
2	236	20,0%	996	hen-of-the-woods, hen of the woods, Polyporus frondosus, Grifola frondosa
3	182	15,5%	992	agaric
4	152	12,9%	993	gyromitra
5	117	9,9%	997	bolete
6	115	9,8%	991	coral fungus
7	49	4,2%	995	earthstar
8	20	1,7%	994	stinkhorn, carrion fungus
9	2	0,17%	988	acorn
10	1	0,09%	910	wooden spoon

Table 4: Results from ResNet-50 network

Simple CNN without data augmentation

The results from the simple CNN without data augmentation can be seen below. From the two charts out of Figure 5 we see that the CNN is doing very well and there are no indices of overfitting. The train and validation data converges in an appropriate temp.

The validation accuracy of the model is 75%, while the test accuracy on our test data subdirectory is 76%.

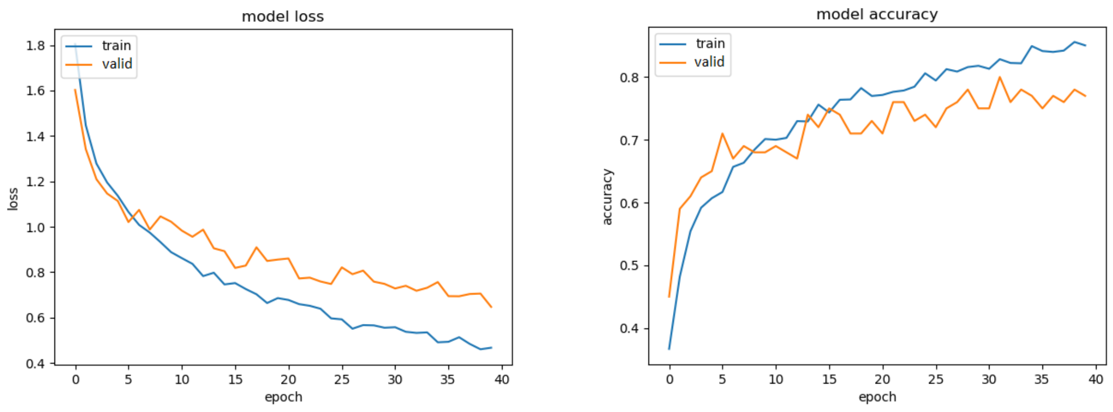


Figure 5: loss and accuracy from simple CNN without data augmentation

Simple CNN with data augmentation

I improved the simple CNN with data augmentation technique. The validation accuracy increased to 85% and test accuracy has a value of 86%. This is an improvement of nearly 10% in comparison to the first simple network.

From the table below the precision and recall from the test data for each category are compared. The values for precision and recall are for some categories quite high. A similar effect as before can be seen. It seems that the network is doing very well on some mushrooms species while other species seems to be classified poor. On category 7 the CNN reaches a precision value of 0.62 which seems that a higher number of images are false categorized to 7. On category 4, it can be seen that only 50% of the mushroom images can be recalled by the CNN. It is possible that there is a high correlation to other mushroom species in the dataset.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	10
2	0.78	0.70	0.74	10
3	0.90	0.90	0.90	10
4	1.00	0.50	0.67	10
5	0.90	0.90	0.90	10
6	0.90	0.90	0.90	10
7	0.62	0.80	0.70	10
8	0.83	1.00	0.91	10
9	0.90	0.90	0.90	10
10	0.91	1.00	0.95	10

Another effect can be seen on Figure 6. In the first epoch, accuracy and loss are converging very well. With increasing epoch, both values doesn't change very much. My first intention is that maybe the network is overfitting on the training data. But the values are quite constant even for higher epochs. It should be notable that the values did not become worse after reaching the epoch 10. The problem is discussed in the conclusion section.

Performing the model fit on the training dataset for this model took a lot of time on a single CPU. The number of epochs has to be reduced to get faster results.

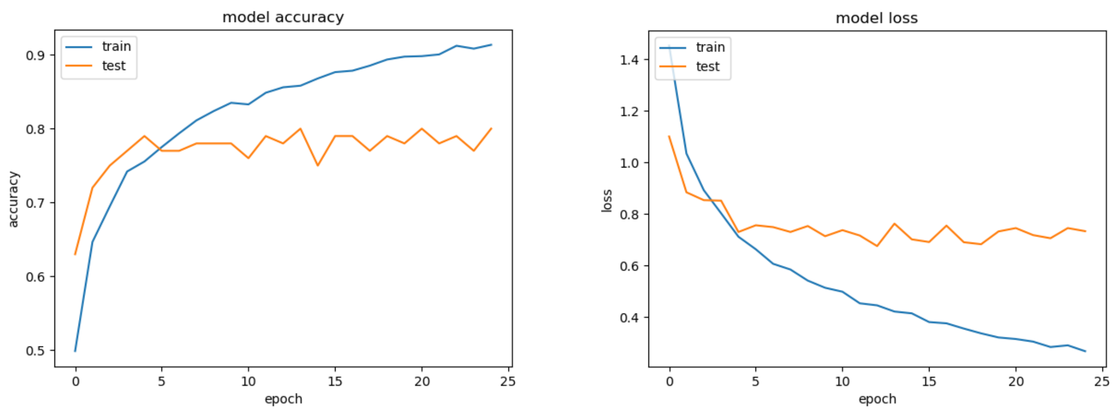


Figure 6: loss and accuracy from CNN with data augmentation

CNN with Bottleneck feature

The results from the CNN with the usage of the bottleneck feature from the VGG16 is listed below. The last chosen configuration gives the best value for the classification model and reaches the high precision and recall values in some categories.

	precision	recall	f1-score	support
1	1.00	0.90	0.95	10
2	0.69	0.90	0.78	10
3	0.86	0.60	0.71	10
4	0.88	0.70	0.78	10
5	0.91	1.00	0.95	10
6	1.00	0.80	0.89	10
7	0.82	0.90	0.86	10
8	1.00	0.70	0.82	10
9	0.60	0.90	0.72	10
10	0.91	1.00	0.95	10
avg / total	0.87	0.84	0.84	100

The validation accuracy was 86% and the test accuracy reaches a value of 85%. Training and validation accuracy are converging with no indication of overfitting in Figure 9.

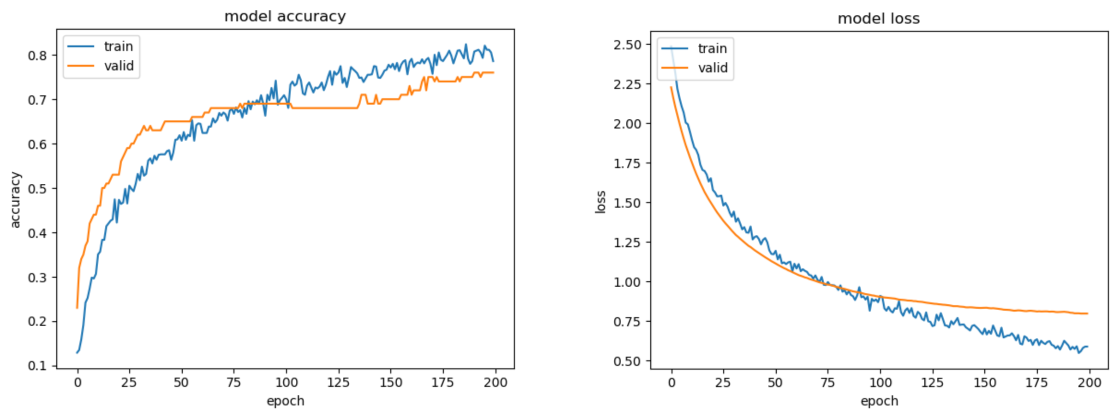


Figure 7: loss and accuracy from CNN with bottleneck feature

CNN with Bottleneck feature and fine-tuning

Freezing the first convolutional layers and adding the pre-trained top model from before was done in the last part of this project. Unfortunately, the results cannot be shown completely, because the calculation on my CPU is too time-consuming. Instead, I will merely give an overview of the results of a trial trial with only 5 epochs.

In Figure 8 the results from the 5 epochs of the fine-tuned bottleneck model can be seen. The preloaded values already make it appear that the model is performing much better on the validation data then on the training data. The number of epochs and the possibility of fine adjustment of the learning parameters from the training phase are too small for an evaluation.

The accuracy on the test data reaches 85%, while the accuracy on the validation data stuck at 76%.

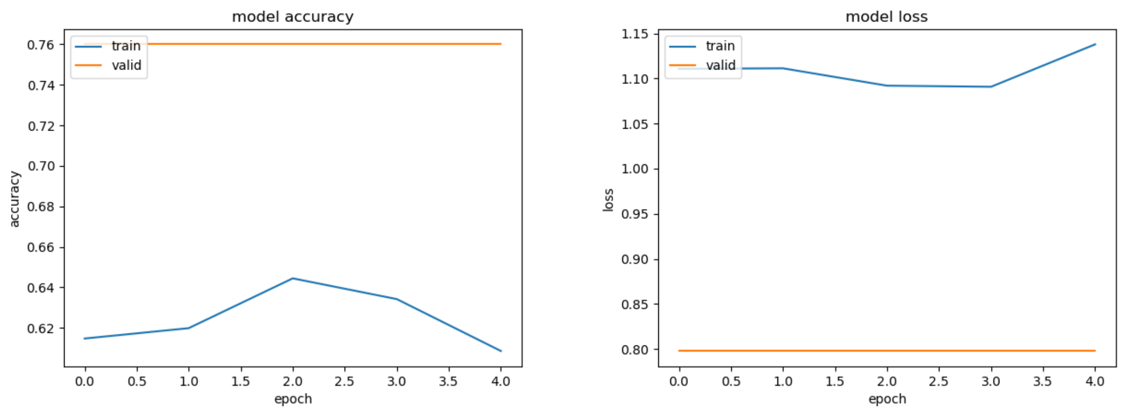


Figure 8: loss and accuracy from CNN with fine-tuned bottleneck feature

Conclusion

Free-Form Visualization

Simple CNN without data augmentation

The best CNN with optimized parameter has an accuracy of 75%. This is a satisfactorily result for a first trial with the collected dataset. In the confusion matrix in Figure 9, it is recognizable that some mushrooms species are classified very well (above 80%), while other categories were recognized incorrectly. The biggest error are that mushrooms from category 8 (Armillaria) were detected as category 4 (Galerina marginata). The main reason is that both mushroom species look very similar.

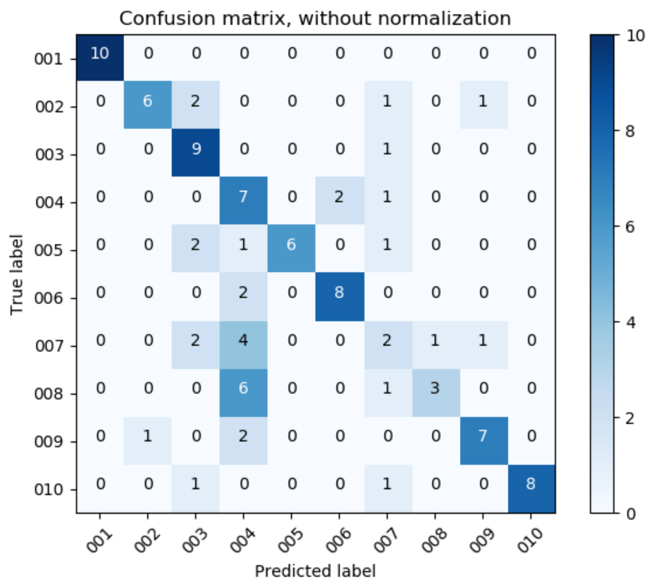


Figure 9: Confusion Matrix from simple CNN without data augmentation

In Table 5 are two images of both species in comparison. Both mushroom types come in groups of mushrooms that may lead the CNN to mix this categorys.

Armillaria	Galerina marginata

Table 5: Comparison between category 8 (Armillaria) and category 4 (Galerina marginata)

Simple CNN with data augmentation

The network reached a very good result. The problem from the first CNN that category 4 and 8 are misclassified seems to be away as seen in Figure 10. It can be seen that there is a high cross-variance is still in category 4. The main reason for this seems to be in the correlation of features in both mushrooms species. Another point is that category 1, 8 and 10 reached the 100% accuracy of the network.

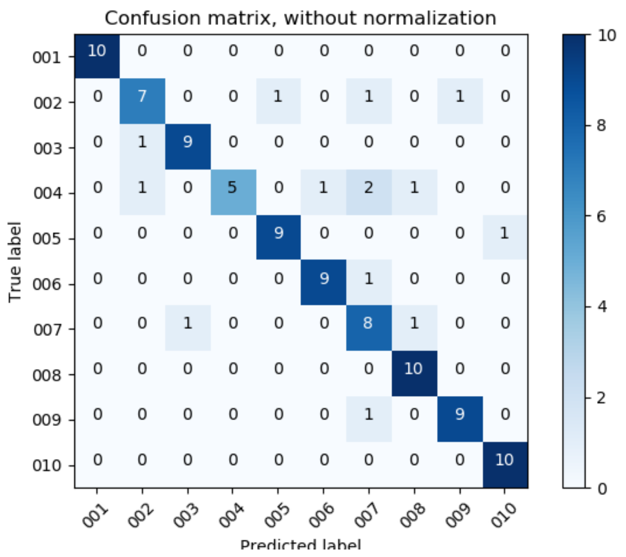


Figure 10: Confusion Matrix from simple CNN with data augmentation

CNN with Bottleneck feature

The improvement in accuracy is not as expected. Reason for this could be this time by not using the data augmentation process. The data effort and the necessary CPU power would have exceeded the possibilities available to me in this case.

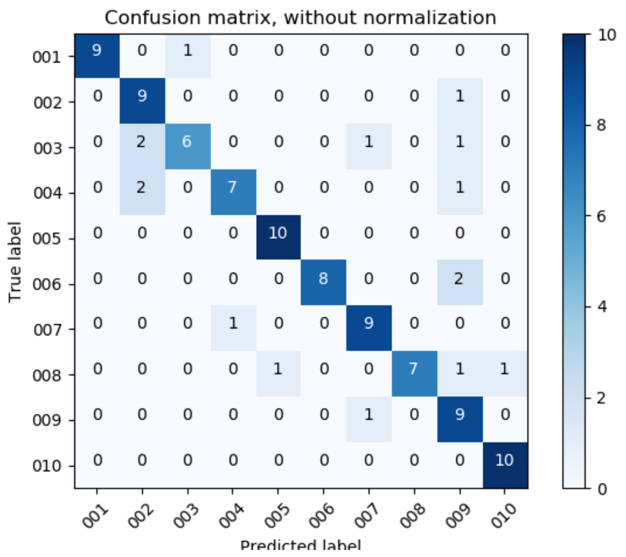


Figure 11: Confusion Matrix from CNN with bottleneck feature

The results from Figure 11 are comparable to the previous CNN. The precision and recall values are little worse compared with the simple CNN.

CNN with Bottleneck feature and fine-tuning

There were not much changes in the confusion matrix in comparison to the previous model. The reason for this is the low number of epochs and the choosen small learning rates to prevent overfitting. The small variation in the validation dataset did not improve the fitting phase of the chosen model.

Reflection

From the prediction from the ResNet-50 network it can be verified that the collected data is very well suited for training a CNN. There are no indicators that other objects appear dominant in the data.

Starting with the simple network it can be seen that this network is able to perform really well on the test dataset. The accuracy reaches a value of 75%. The best predictions can be seen on mushrooms species that are very easy to differentiate - even as human (example: Amanita muscaria). A problem is that, as shown, some mushroom species look very similar and the trained simple CNN has problems to distinguish between similar species.

On the simple CNN with data augmentation we saw that the values for accuracy and loss become much better, up to 86%, and we also saw good results in precision and recall. But it is also notable that the network didn't seem to increase their accuracy after 10. The reason for this could be overfitting but then the error should increase at higher epochs - but that wasn't observed. It seems like the classification limit of this network is reached. It is possible that this is caused by the low number of validation data (=100). A better validation strategy in such cases would be to do k-fold cross-validation, this would require training k models for every evaluation round.

The results from the CNN with bottleneck feature are below the expectations from the implementation section with 85%. The main issue that I wasn't able to use data augmentation on the CNN. The variation in the training data was too low to reach higher values and generalize the feature while training. This problem may can be solved by applying GPU power to the model and adding massive data augmentation for training.

To further improve the previous result, I wanted to "fine-tune" the last convolutional block of the VGG16 model alongside the top-level classifier. Fine-tuning consist in starting from a trained network, then re-training it on a new dataset using very small weight updates. I wasn't able to show the results from the last training and evaluation step. As already described - to fit the model with the augmented data was to time-intensive. A brief overview was nevertheless given. It has been shown again that the network has difficulty to set the parameters to get an accuracy of more than 85%. The main reason is again, in my opinion, the small selection of validation images from the validation dataset.

Resulting the Deep Shrooms project. It could be shown that it is much simpler using CNNs to classify mushrooms by their species whether they are edible or not. The Deep Shrooms project reaches a accuracy of 55%. This value is clearly worse in comparison the best result here of over 85%. In conclusion, it can be deduced from this that the classification into mushroom species is much better suited for image recognition than the distinction into edible or not-edible.

Improvement

To improve the usefulness of the application to the user it is necessary to add more mushrooms species to the network. This will also make the structure of the CNN more complex. Therefore additional enhancements can be done to increase the accuracy of the image classification.

In the report I already discussed steps to increase the accuracy of the model. There are some fundamentally steps that possibly lead to better results for our problem.

- Increasing the training dataset:
  A larger dataset for training and validation increases the possibility that the CNN is doing well generalizing the main features from the image.
- Advanced data augmentation Techniques:
  Doing a more aggressive data augmentation can help that the model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.
- Adding Dropout-layers, increase dropout parameter:
  Adding layers or increase dropout parameters also helps reduce overfitting, by preventing a layer from seeing twice the exact same pattern, thus acting in a way analoguous to data augmentation.
- Fine-tuning one more convolutional block (alongside greater regularization):
  Freezing also parts from the pre-trained convolutional part of the VGG16 should increase the sensitivity of the bottom part of the network to generalize the forms and shapes of mushrooms. There will be a greater regularization in alongside in the CNN.

Links and Sources

https://tuomonieminen.github.io/deep-shrooms/

https://pdfs.semanticscholar.org/8efa/ea4085e64785143e21f1797e9c2c95c8f2f7.pdf

http://www.mushroom.world/

http://www.image-net.org/

https://www.theverge.com/2017/7/28/16054834/mushroom-identifying-app-machine-vision-ai-dangerous

http://www.funga.fi/teema-aiheet/sienten-tunnistaminen/

https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf/RK%3D2/RS%3DPnB3CMxxSoOYRnLD3KKFviCVQvs-

https://keras.io/preprocessing/image/

https://keras.io/

https://www.kaggle.com/keras/vgg16

https://keras.io/applications/#mobilenet