# Capstone Project

Jonas Harnau[1]

**Abstract**
We propose a two-stage model that improves classification accuracy compared to a standard convolutional neural network by making use of bounding box data that is available for a small subset of classes where the classes are relatively homogeneous. Specifically, we apply the model to a dataset of about 22,000 turtle and tortoise images from 22 species; we have bounding box information for some 2,000 images distributed over five species. The first stage is a binary implementation of *Faster R-CNN* Ren et al. [2017] which has the purpose to detect the turtles or tortoises in the image and puts forward a single region proposal. The second stage model never gets to see the full image but rather only the cropped region proposed by the first stage. The model outperforms the benchmark significantly, improving top-1 and top-3 error by about 2pp and 1.25pp, respectively.

**Keywords**
Image Classification — Convolutional Neural Network — Faster R-CNN

[1] *Department of Economics, University of Oxford, United Kingdom, j.harnau@outlook.com*

## Contents



## 1. Definition

### 1.1 Project Overview

While there is an abundance of apps catering to cat and dog lovers, turtle and tortoise aficionados have so far been neglected. We cannot and will not tolerate this injustice any longer. To remedy the situation, we build a model for turtle and tortoise species classification. For conciseness, we shall henceforth refer to the two species together as *turtletoise*. While the main focus of the model is classification, we added a boon to make up for the dry-spell when it came to turtletoise-based procrastination: the model cannot only label the image as a whole but can find potentially multiple turtletoises in the picture, labeling and drawing a box around around each find detection.

Broadly speaking, the proposed project comes from the domain of computer vision. This field was revolutionized in 2012, when Krizhevsky et al. [2012] entered the convolutional neural network (CNN) *AlexNet* into the prestigious ImageNet classification challenge. *AlexNet*, the first CNN used in the challenge, not just won but blew its competitors out of the water. Since then, CNNs have become the dominating force and arguably the state-of-the-art in image classification. Deshpande [2016] gives an overview of seminal papers in the field. An example for another popular CNN is *VGG16* Simonyan and Zisserman [2014] which is rather simple in structure but has many more layers (is much "deeper") than *AlexNet*. While image classification is interesting in itself, combining classification with detection yields some truly fascinating results. Girshick et al. [2014] married these two, introducing *Regions with CNN features* or R-CNNs. While this first R-CNNs worked well, they were rather slow with up to 50 second processing time per image; *Fast R-CNN* Girshick [2015] and *Faster R-CNN* Ren et al. [2017] fix this issue; the latter is

capable of processing about five images per second; *Faster R-CNN* largely builds on *VGG16*. While these R-CNNs bound objects by boxes, *Mask R-CNN* He et al. [2017] goes a step further and detects objects on the pixel level. Parthasarathy [2017] gives a history of R-CNNs.

### 1.2 Problem Statement

From a non-technical perspective, our primary goal is to design a model that takes an image as input and outputs an image annotated with the species of the dominant detected turtletoise. As a secondary objective, the model should also be capable of drawing bounding boxes around potentially multiple detected turtletoises, labeling each one separately.

From a technical point of view, we are interested in the potential to improve classification performance by labeling images based on the label attributed to the most likely detected object in the image, rather than by labeling the image as a whole. This is particularly challenging because we have no bounding box information at all for a majority of species, and if we do, it is available for a sub-sample of images within species. Thus, the total number of images with bounding boxes is small.

The idea is to use a two-staged model. First, we train a *Faster R-CNN* with a binary classifier, turtletoise or not, across the bounding box data. We then use the trained model to predict the single most likely bounding box for all images, including unseen species, and crop the image to that box. This "generalized detection" approach seems promising since turtletoises likely share defining characteristics. The model can learn these on just a few species and extrapolate the information to other, unseen, species for which we do not have bounding box information. This approach seems less promising for other settings with more heterogeneous groups: a network trained to detect turtletoises may have problems to detect cars instead, for example. Second, we take the cropped images output by the first stage model and use them as the training data for a CNN. This second stage model does not get to see the original, un-cropped image. Given that we feed only a single cropped region per image to the second stage, the output of the second stage CNN is a single classification vector containing probabilities for each species.

A pleasant side effect of the two-stage model is that it can be adjusted to return images with bounding boxes for detected turtletoises annotated with species even if we did not have bounding box information for said species. Further, we can allow the first stage to report that no turtletoise was detected in an image, hence feeding no detection to the second stage such that the output for an image can be "no turtletoise" even though the second stage was never trained for that. However, we do not perform a study to evaluate the performance for this application here. Still, such a study is certainly feasible for a future investigation, if not necessarily with the data at hand for which bounding box information is rather sparse. A quantitative investigation would require data with bounding box information for all species. We could then hold back this information for some species in training, but still reliably evaluate the model performance in the testing stage.

### 1.3 Metrics

Putting ourselves in a turtle fan's shoes, we expect the most likely input to be an image with a single turtle in it. This is why we focus on classification accuracy for the quantitative model evaluation. We would of course prefer the model to correctly classify the turtle species. However, if it fails at that, we at least want the correct species to be among the top contenders. Thus, we choose top-1 and top-3 error rates as our evaluation metrics. The top-k error is the proportion for which the correct label was not included in the k most probable labels according to the model. Of course, a lower error is better. Top-k error is for example used as a scoring metric by the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), so by using this metric we are in good company. The ILSVRC scores based on top-5 error given 1,000 classes; meanwhile, we have only 22 species, we decided to be a little less generous and use top-1 and top-3 errors instead.

## 2. Analysis

### 2.1 Data Exploration and Visualization

We have a total of 21,639 images for 22 turtle and tortoise species downloaded from the ImageNet database Jia Deng et al. [2009]. The minimum number of images per species is 160, the maximum 1,328. For a subsample of 2,151 images distributed across five species we also have data for bounding boxes available; there is a similar number of images with bounding boxes for these five classes with between 408 and 478 images per species. For some images, there is more than one bounding box available. However, this is the exception: there are 75 images with more than one box, 49 of which come from the *Terrapin* species. We then have 2,255 bounding boxes distributed across 2,151 images. We show a randomly drawn image from each of the 22 species in Figure 1; drawing from those with bounding boxes, annotated in red, if applicable.

The draw captures the variety of perspectives, and overall quality encountered in the data. For example, the Leatherback Turtle seems very hard to make out for the human eye, hiding in the ocean, the *Slider* is very much off center, the *Giant Tortoise* image actually has two tortoises in it, and the draw for the *Alligator Snapping Turtle* quite clearly does not show a turtle. Of those, we consider only the last point as a problem with the data. However, for the purpose of this investigation we assume that such misplaced images are not rare and thus do not take action to remedy the situation.

Looking specifically at the images with bounding boxes, they seem to accurately capture the turtletoise position; we may be able to argue about the box for the Leatherback, but that is mostly because it is very hard to locate in the first place.

The random sample also exposes that the image resolution, shown on the axes in pixels, varies across images. This is reflected in the descriptive statistics in Table 1 which shows


**Figure 1.** Example images for each of the 22 species with bounding boxes annotated where available. Image resolution in pixels shown on the axes. *(Best viewed digitally.)*

that we have a broad spectrum of resolutions. The average image is 461 pixels wide and 347 high (computed separately) and thus in landscape format. There is a clustering of images at about 500x350 pixels, spanning at least 25pp of the distribution.

| | Mean | Std | Min | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|---|
| Width | 461 | 223 | 48 | 375 | 500 | 500 | 4000 |
| Height | 347 | 174 | 38 | 286 | 357 | 375 | 3658 |

**Table 1.** Descriptive statistics for image resolution in pixels computed seperately for width and height.

## 2.2 Algorithms and Techniques: Two-Stage Model

As motivated in the Defintion, R-CNNs have become very popular for object detection while CNNs are the state of the art in object classification. We propose a two-staged model that combines them sequentially. We discuss the two stages in turn.

### 2.2.1 First Stage: *Faster R-CNN*

The first stage closely follows *Faster R-CNN* Ren et al. [2017]. This model is capable of combining object detection with classification of the detected objects. However, training the model to achieve this feat requires bounding boxes annotated with class labels for all relevant classes. Since we have such data only for a small subset turtletoise species, we thus cannot simply train a *Faster R-CNN* and call it a day. Instead, we opt to first train a binary *Faster R-CNN* to detect regions containing turtletoises and feed the most likely proposed region of this model to the second stage. Rather than diving too far into the details of the first stage model, we describe it in rather rough terms and try to give an intuition, referring the reader to the original paper for a more detailed explanation. When

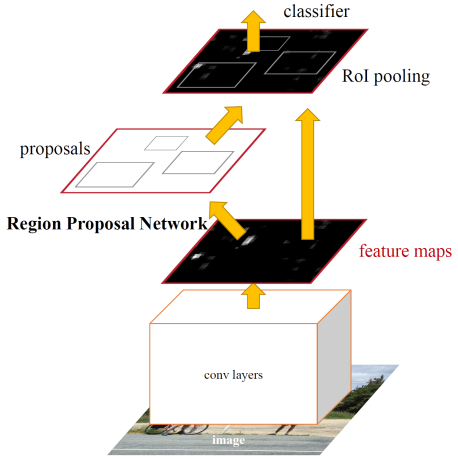


**Figure 2.** *Faster R-CNN*, taken from Figure 2 in Ren et al. [2017]

we discuss the implementation below, we make sure to point out where we diverge from the original paper.

The idea of *Faster R-CNN* is to train object detection and classification simultaneously. The model consists of four stacked components: first a standard convolutional block, second a region proposal network (RPN), third a region of interest (RoI) pooling layer, and fourth a classifier block which produces predictions for bounding boxes and classification of said boxes. This structure is shown in Figure 2. The RPN is the heart of *Faster R-CNN*. The last two components are identical to *Fast R-CNN* Girshick [2015]. To explain how *Faster R-CNN* works, we pick *VGG16* Simonyan and Zisserman [2014] as a concrete example and explain the changes necessary to make an R-CNN out of the CNN; the original paper builds on *VGG16* as well.

**Convolutional Block** The convolutional block that makes up the first component of *Faster R-CNN* consists of almost the entire network: looking at the graph of *VGG16* in Figure 3, it consist of the five convolutional blocks layers up to and including `block5_conv3`. Is it notworthy that the convolutional block is shared between RPN and RoI pooling layer plus classifier. A noteworthy difference to the standard *VGG16* implementation is the input dimension. While fixed at $224 \times 224 \times 3$ in the original CNN, *Faster R-CNN* allows for variable input shapes, rescaling images so the shorter side is 600 pixels while adjusting the longer side to keep the original ratio intact. This implies that the output shape of `block5_conv3` also varies by image.

**Region Proposal Network** The second component is the region proposal network which takes the feature maps of `block5_conv3` as input and returns a number of box-regions along with an "objectness score" for each region. This is arguably the most complex component of the model despite the fact that it consists of only three layers. With respect to the original *VGG16*, we can think of RPN as an insertion between the *block5_conv3* and *block5_pool*. In training, the RPN is

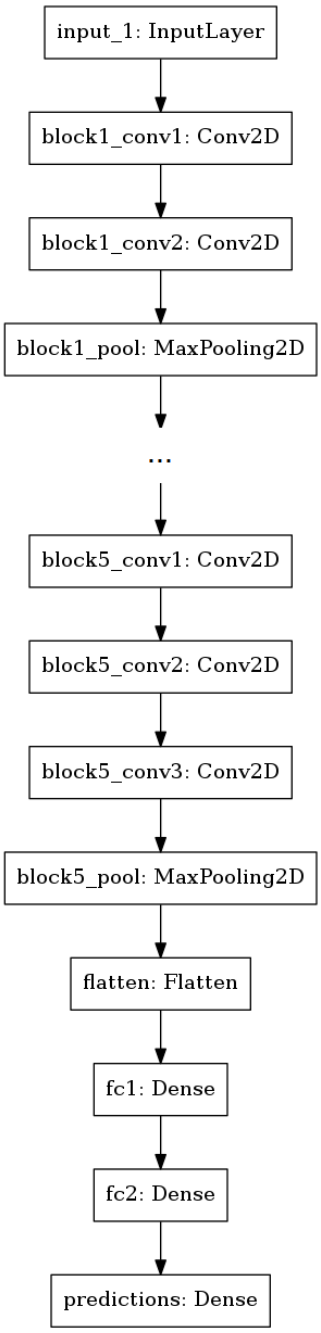


**Figure 3.** Graph of *VGG16* Simonyan and Zisserman [2014], convolutional blocks two to four not shown.

trained directly as well as used to provide region proposals that are fed into the RoI pooling layer discussed below.

The feature maps of `block5_conv3` input into the RPN first encounter a 512-filter $3 \times 3$ intermediate convolutional layer (stride 1, ReLu activation) which connects simultaneously to two $1 \times 1$ convolutional layer: one performs box-regression (linear activation, four coordinates per box), the other classifies whether there is an object or not (sigmoid activation). Indeed, for each $3 \times 3$ window these two output layers generate $k$ box-proposals and $k$ objectness scores where Ren et al. [2017] choose $k = 9$; we discuss the reason for this choice when we discuss RPN training below.

The number region proposals output by the RPN is large and Ren et al. [2017] note that some of them highly overlap. To reduce this redundancy and with it the number of proposal fed into the RoI pooling layer discussed next, they implement non-maximum suppression, using the objectness score for ranking. This algorithm takes the box with the highest objectness score, adds it to the set of kept boxes, and discards any box with an IoU larger than the chosen threshold of 0.7 with the kept box. This process is iterated. We are left with a smaller number of region proposals ranked by objectness score.

When it comes to training, the RPN is both trained directly to improve the quality of the region proposals and features indirectly by providing region proposals to the RoI pooling layer.

Directly training the RPN requires a ground truth. For this purpose, Ren et al. [2017] introduce "anchors", rectangles centered at the center of a $3 \times 3$ window mapped to the original image. Ren et al. [2017] choose three shapes for each of three sizes for these rectangles, yielding a total of nine anchors per window as shown in Figure 4, explaining the choice of $k = 9$ above. We note that the RPN is trained based on the raw region proposals before non-maximum suppression. Given an

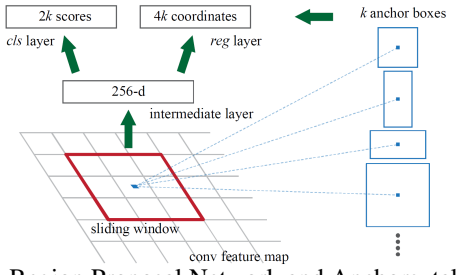


**Figure 4.** Region Proposal Network and Anchors, taken from Figure 3 in Ren et al. [2017] (the intermediate layers in the image has 256 rather than 512 filters)

anchor on the original image, we can compute the intersection over union (IoU) with the ground truth bounding box to decide whether the anchor should be assigned a positive (IoU $> 0.7$), negative, (IoU $< 0.3$), or neutral label. This is done for each of the $9 \times \mathrm{W}_{featuremap} \times \mathrm{H}_{featuremap}$ anchors per image, except those that cross the image boundary. Ren et al. [2017] suggest that dropping boundary crossing anchors both reduces the

number of anchors from about 20,000 to about 6,000 for a $600 \times 1000$ image, but also prevents convergence issues of the network. These remaining anchors constitute the ground truth for training the RPN with a multi-task loss that is a weighted sum of the box-regression and objectness-classification loss. However, rather than using all available anchors per image in training, Ren et al. [2017] suggest to randomly sample 256 of them aiming for an even ratio of positive and negative anchors to avoid biasing towards the more common negative samples.

The indirect appearance of the RPN in training comes through its prediction of regions of interest that constitute the inputs into the RoI pooling layer. These inputs are region proposals after performing non-maximum suppression to reduce redundancy. For the same reason as in direct training of the RPN, at training the number of regions of interest fed to the RoI pooling layer is a sample from the total number of proposals per image and positive regions, that is regions containing an object, are over-sampled to prevent biasing towards the more common background regions.

**Region of Interest Pooling** The RoI pooling layer's job is to take the regions proposed by the RPN which come in heterogeneous shapes and map them to fixed size feature maps, $7 \times 7$ in this case. It takes as input the feature maps output by the convolutional block as well as with the region proposals from the RPN in form of box coordinates on these features maps. Then, the RoI pooling layer performs max pooling for each region proposal. Grel [2017] describes this process in detail on a specific example. We can think of the RoI pooling layer as replacing the `block5_pool` layer in *VGG16*.

**Classifier** The final component of *Faster R-CNN* is again rather similar to the final dense block of the original *VGG16*. It takes feature maps corresponding to a region of interest of the RoI pooling layer as input and returns a classification and a bounding box for each proposal; if there is more than one region of interest, this is done for sequentially for one region at a a time.

In Ren et al. [2017], the block of a flatten and two dense layers (ReLu activation) with 4,096 nodes each is retained, while the final dense prediction layer is replaced with two parallel dense layers, one for classification (softmax activation) that returns probabilities for labels plus background, and one for bounding box regression (linear activation) which returns boxes for each label, but not the background. Just as the output layers of the RPN, the classifier component of *Faster R-CNN* is trained with a multi-task loss that takes into account the performance of both outputs.

While we retain the general structure for all other components of the model, we make some alterations to this component for two reasons. First, we are lacking the computational power to train the original dense block: the two dense layers `fc1` and `fc2` have some 120 million parameters, almost 90% of the total number of parameters in the original *VGG16*. Second, we only have 2,151 images with bounding box information so that we want to introduce some regularization. First, we replace the flatten layer with a global average pooling layer, reducing the output dimension from 25,088 to 512. Second, we half the number of nodes in the two dense layers to 2,048. Third, we introduce 50% dropout Srivastava et al. [2014] for `fc1` and `fc2` to regularize the model. We keep the two parallel classification layers unchanged.

#### 2.2.2 Second Stage: *VGG16*

For the second stage, we consider two slighty variations of *VGG16* Simonyan and Zisserman [2014] as shown in Figure 3. For the first, we only adjust the final dense layer to match the number of nodes to the number of species, namely 22. For the second, we orient ourselves on the classifier component of the first stage. that is, we replace the flatten layer with a global average pooling layer and half the number of nodes in the dense layers `fc1` and `fc2` to reduce dimensionality. We also add 30% dropout to these two dense layers for regularization. This new dense block is shown in Figure 5. We limit

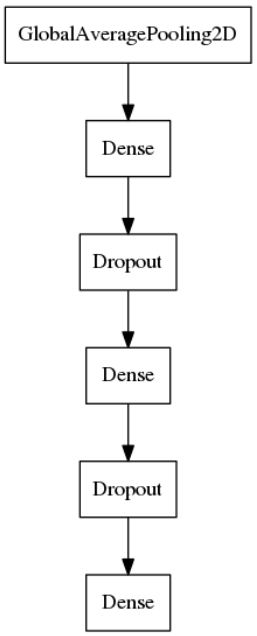


**Figure 5.** Dense block for the second considered graph of the second stage model. The global average pooling layer connects to `block5_pool` of *VGG16* as shown in Figure 3, replacing the block from `flatten` to `predictions`.

ourselves to the same fixed input size as the original network, namely $224 \times 224 \times 3$. To obtain the inputs, we first feed an image trough the separately trained first stage and obtain the bounding box with the highest certainty of containing a turtletoise. Then, we warp the content of this bounding box to the input size of the second stage model. While the choice of *VGG16* is arbitrary, we could have chosen a different CNN as a baseline such as ResNet He et al. [2015], this likely would not substantially impact this investigation as it is about the relative performance of the two-stage model to the benchmark model, rather than the absolute classification performance.

As a caveat for a probabilistic interpretation of the second

stage model output we should take into account the first stage of the model. That is, we should interpret the probability vector of species output by the second stage as a conditional probability given that there is a turtletoise in the proposed region. Thus, to get the probability that the region shows a turtletoise of the species is given by the product of first stage turtletoise score and second stage output:

$$\mathrm{P}(\text{species}) = \mathrm{P}(\text{species}|\text{turtletoise})\mathrm{P}(\text{turtletoise}).$$

### 2.3 Benchmark

The main idea behind the two-stage model is to improve upon a more standard CNN for image classification by pre-selecting the likely region of interest from the image and feeding this region into the CNN, rather than the entire image. Thus, it seems natural to pit the two-stage model against a single-stage model that does not have a first stage. We believe that the most natural benchmark is a CNN that exactly matches the second stage models described above, both for set-up and training. The difference is that the input to the benchmark model is not a warped region of an image output by the first stage, but an entire image warped to $224 \times 224$ pixels. Due to the strong connection between the two-stage model and the benchmark, we discuss the specific implementation and results of the benchmark in tandem with that of the two-stage model below.

## 3. Methodology

### 3.1 Training-Validation-Test Split

Following common practice, we split the data into training, validation, and test set. We decided to set 60% of the data ($\sim 13,000$ images) aside for training, and 20% ($\sim 4,300$ images) each for validating and testing. The bounding box data is not needed at test since the evaluation metrics are top-k-errors which require only labels. Thus, to make the most of the bounding box information, we do not include images with bounding boxes in the test set. Instead, we sample 80% ($\sim 1,700$ images) of the bounding box images into the training, and 20% ($\sim 400$ images) into the validation set. Thus, we stratify not just by species but also by bounding box information. For example, we have 969 images of Leatherback Turtles, 415 of which also have bounding box information. Of the images with bounding boxes, we sample 80% (332 images) into the training and the remaining 20% (83 images) into the validation set. Of the 554 images for which we do not have bounding boxes, we sample 194 ($\sim 0.2 \cdot 969$) into the test set. We split the remaining 360 images without bounding boxes to fill up training and test sets such that we have a 60-20-20 overall split: 250 go into the training and 110 into the validation set leaving us with a total training-validation-test split of 582-193-194 corresponding to 60.06%-19.92%-20.02%.

### 3.2 Data Preprocessing and Augmentation

For all models, first and second stage, and benchmark, we preprocess the data by subtracing the *ImageNet* mean channel-wise as is common practice. For augmentation, we flip the images horizontally during training, but not during validation or testing.

### 3.3 Implementation

We implement all models in python using `Keras` Chollet and Others [2015] with a tensorflow backend.

#### 3.3.1 First Stage

To implement of the first stage we use the GitHub repository `keras f-rcnn` Henon [2017] and adjust it as needed; for the most part, this consisted in replacing the original classifier block as discussed above and implementing validation after each epoch.

We initialize the convolutional block with *ImageNet* weights; the idea is to use transfer learning Zeiler and Fergus [2014], Yosinski et al. [2014]. For the RPN, following the standard in the repository, we initialize the intermediate layer with Gaussian, the objectness-classifier with uniform, and the regression layer with zero weights. Finally, we initialize the dense layers in the classifier with `Keras`'s standard Glorot-uniform distribution Glorot and Bengio [2010] for the kernels and zeros for the biases. The first stage model has 22,356,851 parameters, all of which are trainable.

We train the model for 50 epochs each consisting of 500 randomly drawn images from the training, validating after every epoch on the full validation sample of about 400 images. Within each epoch, we follow the implementation in `keras f-rcnn`; we note that this procedure differs somewhat from the training outlined in *Faster R-CNN*.

First, we draw a single image and train the the convolutional block as well a the RPN on it with back-propagation optimizing with "Adam" and a learning rate of $10^{-5}$; see Ruder [2016] for an overview of stochastic gradient descent algorithms. The idea behind the small learning rate is to prevent the model from adjusting too strongly too quickly, destroying the information contained in the pre-trained *ImageNet* weights.

Second, we use the RPN to predict region proposals for the given image. As discussed above for the RoI pooling layer, we use non-maximum suppression with an IoU threshold of 0.7 to reduce the redundancy in the number of proposals. We sample 32 out of 300 region proposals, aiming for an even split between positive (containing a turtletoise) and negative (background) samples. If this is not possible due to lack of positive samples, we take the available positive samples and fill up with negative samples.

Third, we feed the selected 32 region proposal through the RoI pooling layer to the classifier and train the convolutional block, just trained by the RPN, and the classifier. The RPN itself is not trained. Just as for the RPN, we back-propagate using "Adam" with a learning rate of $10^{-5}$.

For each epoch, we record all four training and validation losses (RPN objectness-classification and box-regression, and classifier block turtletoise-classification and box-regression) along with the accuracy of the turtletoise-classification. We save weights for the model with the lowest validation total

loss, that is summed over the four components. Given that this training took about a day on a temporarily available 4GB GTX970 GPU, we did not have the resources to try out other configurations.

After model training is complete, we use the trained model to predict the bounding box that most likely, as measured by the turtletoise-classification layer, contains a turtletoise for each image in the training and the validation set. At this point, we set the number of regions of interest fed to the RoI pooling layer to 100. A larger number might yield better results but would also increase the number of forward passes through the classifier component and thus increase computational time. Given that the final two-stage model will have to go through the same process for testing and other applications it seemed like a good idea to keep the number of proposals limited. Given the bounding box output by the classifier, we crop the image and resize it to $224 \times 224$ pixels, the input shape of the second stage model. We keep the training-validation split unchanged.

An idea for a refinement could be to crop images conditionally: if the first stage model is less than 50% certain that the best bounding box contains a turtletoise, we could warp the original image, rather than the cropped version. The idea behind this would be to reduce the likelihood of feeding a background region that does not contain a turtletoise at all to the second stage.

#### 3.3.2 Second Stage

The second stage makes use of the cropped images output by the first stage. As discussed in Section 2.2.2, we consider two slightly different graphs both based on *VGG16*. The first merely replaces the final prediction layer to match the number of nodes to the number of species and has 134,350,678 parameters. The second graph replaces the final dense block as in Figure 5 and has 20,006,742 parameters. Based on the two graphs, we train three different specifications. The idea is to use transfer learning, slowly increasing the number of trainable layers from the output towards the input. We believe that this is a sensible approach since the layers closer to the output of the model are know to be more task specific, while the layers towards the input capture more general features Zeiler and Fergus [2014], Yosinski et al. [2014].

**Specification 1** We initialize the first graph with *ImageNet* weights where possible. The new layers are initialized randomly with `Keras` standard Glorot-uniform setting. We train only the last layer which contains 90,134 parameters, fixing all other layers at their *ImageNet* weights. We choose a categorical cross-entropy loss which is natural for a multi-class specification task. We chose "Adam" with `Keras` standard settings as the optimizer. We use a batch size for of 32 which was just about computationally feasible.

**Specification 2** The second specification is based on the second graph, initialized just as the first specification. We make the entire dense block in Figure 5 trainable, leaving us with 5.3 million trainable parameters. We make the entire dense block in Figure 5 trainable, leaving us with 5.3 million trainable parameters. To accommodate the larger computational demand, we had to reduce the batch size for training this model to 25. We use "Adam" with standard settings.

**Specification 3** The third specification has the same graph as the second. We initialize this model from the best weights found in the second specification. Further, now we all layers from the final fifth convolutional block and thus from texttt-tblock5_conv1 up trainable. With that, 12.3 million parameters are trainable. The batch size for training this model was 20. We choose a stochastic gradient descent optimizer with a smaller learning rate of $10^{-4}$ as well as Nesterov-momentum of 0.9. The idea behind the small learning rate is to prevent the model from adjusting too strongly too quickly, destroying the information contained in the pre-trained weights from Specification 2.

All specifications are trained with a categorical cross-entropy loss, which is a natural choice for a multi-class specification task, with back-propagation for up to 100 epochs of 2,598 randomly drawn images training images, corresponding to a fifth of the training sample. Each epoch is followed by an evaluation on a sample of 864 randomly drawn validation images, corresponding to a fifth of the validation set size. We keep the epochs rather short with only a fifth of the training sample size to be able to react to changes in model performance more quickly (we control the number of images seen by the model by choosing a five times larger number of epochs, here 100 instead of 20). We reduce the validation set size proportionally both to decrease computing time and to work against the increased potential to "fit" to the validation set that comes with a larger number of epochs and the hence larger number of decisions made based on validation set performance. We stop training early if the validation loss did not decrease for 20 epochs.

After training has completed, we first find the best model within specification as measured by the lowest validation loss during training to be the best model for this specification. To choose the best model across specifications, we then compute the validation loss, top-1-error and top-3-error over the entire validation set for the best model of each specification. The idea behind this is to decrease the variance in validation set performance induced by training on a smaller validation set. We then choose the best model from the three specifications based on the best performance over the entire validation set.

#### 3.3.3 Benchmark

To maximize comparability to the two stage model, we implement the benchmark models in exactly the same way as the second stage models. That is, the sole difference to the second stage is that the inputs are not cropped images warped to $224 \times 224$ pixels but the original images warped to the same size. Other than that, we implement the same three specifications with the same initializations and the same training settings on the same training and validation sets. We believe that this constitutes a good benchmark for the two-stage model

as it directly answers the question of the worth of the first stage. For example, if the two-stage model with second stage specification **x** does not have lower top-1 or top-3-error than the benchmark model with that same specification on the test set, then we may conclude that the two-stage model does not improve classification performance, at least for that specification. Of course, the best possibly result for the two-stage model would be to have lower top-1 and top-3 error than the benchmark model across specifications.

# 4. Results

## 4.1 Model Evaluation and Validation

We first discuss the results for the benchmark model before discussing the results for the first and second stage in turn, comparing the second stage results to the benchmark.

### 4.1.1 Benchmark

We discuss the training performance of the three specifications for the benchmark model, explain how we select the best model and then evaluate the performance on the test set.

Figure 6 shows the training and validation loss, top-1-error, and top-3-error for each specification. We discuss them in turn.

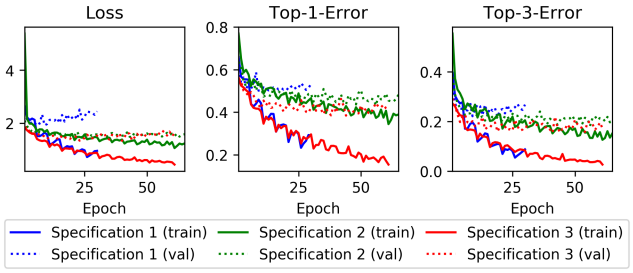


**Figure 6.** Benchmark model training and validation loss, top-1-error and top-3-error over training epochs

The limited flexibility of the benchmark for Specification 1, for which we kept all but the last layer of the original *VGG16* and made only the last layer trainable, shows quickly. This model attains the lowest validation loss of 1.91 after just nine epochs with validation top-1 and top-3-errors of 50% and 22%. Still, the model manages to over-fit quickly as indicated by the diverging training and validation metrics.

The benchmark model based on Specification 2, where we introduced a new dense block including dropout but kept all convolutional layers fixed, performs better, reaching a validation loss of 1.46 after 44 epochs with top-1 and top-3 error rates of 44% and 20%. The dropout seems to regularize the model well: there is little evidence of over-fitting and the training and validation metrics diverge at most slowly. Thus, the training metrics are in fact substantially "worse" than those of Specification 1.

Finally, the benchmark for Specification 3, the model matching Specification 2 except that the last convolutional block is trainable as well, has a minimum validation loss

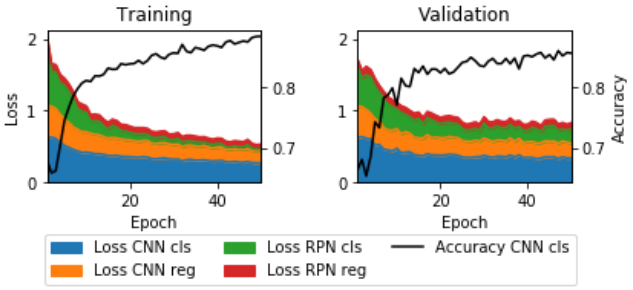


**Figure 7.** First Stage Loss and Classifier Accuracy

of 1.33 after 40 epochs of training for which top-1 and top-3-errors are 40% and 17%. However, this model seems to over-fitting: the validation metrics are slightly lower than those of Specification 2, yet the training metrics track those of Specification 1. This indicate that this model may benefit from additional regularization. However, the two-stage model would suffer from the same issue so that the comparison is still fair; thus we decided not to consider more specifications.

To choose between the benchmark models, we then compute the validation loss, top-1-error and top-3-error over the entire validation set for the best model of each specification. The idea is to decrease the variance in validation set performance induced by training on a smaller validation set. We then choose the best model from the three specifications based on the best performance over the entire validation set. As we

| | | Specification | | |
|---|---|---|---|---|
| | | 1 | 2 | **3** |
| 1/5 | Loss | 1.92 | 1.46 | **1.33** |
| | Top-1-Error (%) | 50.12 | 43.66 | **39.77** |
| | Top-3-Error (%) | 22.11 | 20.46 | **16.70** |
| Full | Loss | 2.02 | 1.56 | **1.48** |
| | Top-1-Error (%) | 51.45 | 46.45 | **42.12** |
| | Top-3-Error (%) | 23.76 | 20.91 | **18.41** |

**Table 2.** Benchmark model results. Best outcome across specifications in bold.

can see in Table 2, we are in the fortunate position that all metrics agree: they all indicate that Specification 3 is the best model with a loss of 1.48, a top-1-error of 42.12% and a top-3-error of 18.41%. We can see that the metrics computed over 1/5 of the validation set are generally more optimistic than those computed over the full set, perhaps indicating that there is some effect of fitting to the validation set.

### 4.1.2 First Stage

We show the first stage training results in Figure 7. The figure shows the losses for RPN box regressor and objectness classifier, and classifier box regressor and turtletoise classifier on the left, and the accuracy of the turtletoise classifier on the right. The total loss, the sum of the individual losses, decreases quickly in both training and validation until about epoch 15. After that, the decrease slows markedly in training and stays about constant in validation. Still, the lowest validation total loss is attained in epoch 43 near the end of training. The classifier accuracy is roughly the mirror image of the losses,

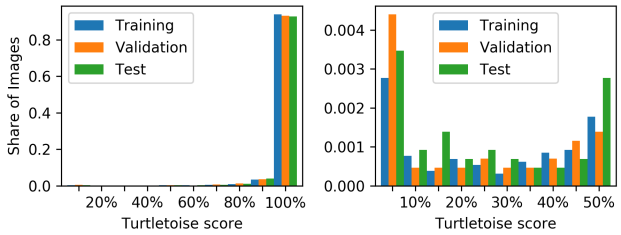


**Figure 8.** First Stage Turtletoise Classification Score, full range on the left, up to 50% on the right

except the speed of improvement slows somewhat earlier at about ten epochs . Looking at the disaggregated losses, the two classifier losses look similar for training and validation. Meanwhile, the RPN losses keep decreasing in training, diverging form the leveling off validation performance. We can take this as evidence that the RPN overfits while the classifier does not. Considering the model, we can explain this by the the lack of regularization in the RPN while the classifier uses 50% dropout between the dense layers. Thus, the model may be improved by introducing dropout in the RPN as well.

Given the purpose of the first stage model is to put forward a suggestion for the most likely region containing a turtletoise, we take a look at what "most likely" means: Figure 8 shows a bar graph of the probability the model attaches to the most likely bounding box containing a turtletoise across images. We shows a results for the whole probability spectrum on the left, while we limit ourselves to the lower half of the spectrum on the right. It is clear that the model is very certain of turtletoise detections for most images since the overwhelming majority, more than 90% of images in each of training, validation, and test set are ranked are classified with 90% certainty or higher. Looking at the lower half of the certainty spectrum, we can see a spike between 0% and 5% which however make up a share less than 0.5% of images per imageset. The the share of images classified with less than 50% certainty, is at most 1.2% per imageset, 125 training, 46 validation, and 54 testing images, and thus rather negligible.

To given an intuition for the turtletoise classification certainty, we sample a random image from each quantile of the certainty distribution from the subset of images in the validation set for which we have bounding box information. In Figure 9 we show these four images annotated with the certainty as well as the ground truth bounding boxes in green and the most likely bounding box output by the first stage in red. Perhaps most impressive is the sample drawn from the lowest quantile: the turtletoises make up a small part of the image and are arguably hard to spot even with the human eye. Still, the best bounding box does a rather good job at finding the relevant region. For the second quantile, the image is slightly out of focus, still the best bounding box includes the turtletoises in the image even though it is too large. The third quantile image is in focus; yet the box turtle in it is half-hidden behind plants. The best bounding box is somewhat shifted from the

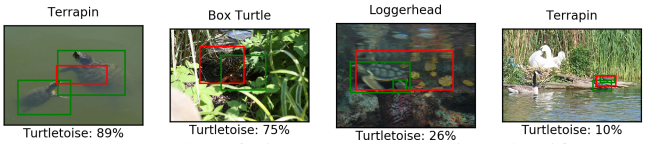


**Figure 9.** Examples of First Stage Turtletoise Classifier Certainty. One image randomly sampled for each certainty quantile. Ground truth boxes in green, highest scoring first stage box in red.

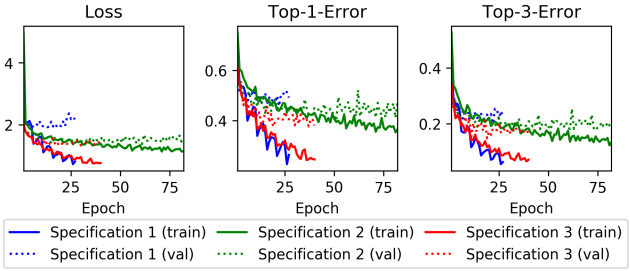


**Figure 10.** Second stage model training and validation loss, top-1-error and top-3-error over training epochs

ground truth but has decent overlap. Finally, the bounding box for the highest quantile captures only part of a ground truth box but does contain a turtletoise; it does not seem obvious why the model chose this specific box. Overall, the first stage model does seem to do a good job to select a cropping region that contains (parts of) a turtletoise, at least for this random draw.

We mentioned before the idea to refine the two-stage model by cropping only images above a 50% turtletoise classification certainty, keeping the original image otherwise. Given the negligible number of images with less than 50% classification certainty and the good detection performance across the certainty spectrum shown in the analyzed random draw, we do not further consider this option.

### 4.1.3 Second Stage

As for the benchmark, we first analyze the three specifications for the second stage model, then select the best one and evaluate it on the test set.

Figure 10 show losses, and top-1 and top-3 errors for the three model specifications. The general appearance of the plots is rather reminiscent of the benchmark model plots in Figure 6. Specification 1 fails to reduce the validation loss after epoch 6 and shows signs of over-fitting evidenced by the quick divergence of training and validation metrics. Specification 2 trains the longest: the validation loss attains its minimum after 61 epochs, While there is some evidence of over-fitting in the later epochs, it is rather muted showing that dropout does its job well. Finally, Specification 3 has a similar validation loss compared to Specification 2 with slightly lower top-1 and top-3 errors; there is evidence of over-fitting for this specification. The lowest validation loss is attained after 19 epochs. We show the lowest validation loss as computed over 1/5 of the validation set in training

| | | Specification | | |
|---|---|---|---|---|
| | | 1 | 2 | **3** |
| 1/5 | Loss | 1.91 (1.92) | 1.38 (1.46) | **1.31** (1.33) |
| | Top-1-E (%) | 46.99 (50.12) | 41.71 (43.66) | **37.61** (39.77) |
| | Top-3-E (%) | 21.64 (22.11) | 18.06 (20.46) | **14.66** (16.70) |
| Full | Loss | 1.86 (2.02) | 1.48 (1.56) | **1.34** (1.48) |
| | Top-1-E (%) | 47.67 (51.45) | 43.11 (46.45) | **38.64** (42.12) |
| | Top-3-E (%) | 22.18 (23.76) | 19.40 (20.91) | **16.46** (18.41) |

**Table 3.** Two Stage model results for training evaluated on 1/5 validation set and evaluation on the full validation set. Best outcome across specifications in bold. Benchmark equivalents in parentheses.

with accompanying the top-1 and top-3 errors in Table 3, also indicating the benchmark performance in parentheses. Specification 3 performs best across all metrics. We also see that the two-stage model outperforms the benchmark model across specification and metrics.

Next, we evaluate the performance of the best performing model within each specification on the full validation set to choose the best model across specifications. Table 3 shows that the ordering coincides with that computed on 1/5 of the validation set but the metrics are somewhat less optimistic. Thus, Specification 3 is the model we choose with a loss of 1.34, top-1-error of 38.64% and top-3-error of 16.46%. Again, the two stage model beats the benchmark in every metric, sometimes by large margins: for example, the top-1-error is some 3.5pp lower for the two-stage than for the benchmark model. However, to truly evaluate the performance we have to look at the test set and this is what we do next.

### 4.2 Justification

So far, the two-stage model performed better than the benchmark across metrics and specifications. However, the two stage model also involves a more involved fitting procedure with perhaps more leaking of information from the validation sets to the weights of the models. For example, the second stage model makes use of information of the first stage model which already made use of the validation set; thus even before training the second stage there is some information spill-over from the validation to the training set, perhaps hampering the reliability of the validation metrics as an out-of-sample metric. Thus, we now consider performance on the test set which neither model has ever seen and which was not touched up until now. If the two stage model outperforms here as well, then we can justify the claim that it improves classification performance compared to the benchmark.

We see in Table 4 that the two-stage model outperforms the benchmark on the test for all metrics of the chosen specification, as well as for the unchosen specifications. Thus, the two-stage model indeed beats the benchmark. Looking at the top-1 and top-3 error, the performance metrics we chose in the beginning, in more detail, we would also argue that the improvement is substantial. For the chosen Specification 3, the top-1-error is with 40.07% almost 2pp lower than the

| | | Specification | | |
|---|---|---|---|---|
| | | (1) | (2) | 3 |
| Loss | Two-Stage | (**1.91**) | (**1.50**) | **1.37** |
| | Benchmark | (2.01) | (1.56) | 1.50 |
| Top-1-Error (%) | Two-Stage | (**48.73%**) | (**44.67%**) | **40.07%** |
| | Benchmark | (51.69%) | (46.37%) | 41.99% |
| Top-3-Error (%) | Two-Stage | (**24.06%**) | (**20.62%**) | **17.25%** |
| | Benchmark | (24.94%) | (22.40%) | 18.48% |

**Table 4.** Results on the test set for benchmark and two-stage model. Best outcome in bold. Outcomes for unchosen models in parentheses.

benchmark value of 41.99%. The top-3-error of 17.25% beats the benchmark value of 18.48% by 1.23pp. Results for the other specifications give a similar impression.

## 5. Conclusion

### 5.1 Free-Form Visualization

In Figure 11 we show eight example images from the test set annotated with the most likely bounding box provided by the first stage. We also show the turtletoise certainty for the box as provided by the first stage, and the top-3 classification proposals as ranked by probability from the best second stage model (based on the proposed box) and the best benchmark model. Even though all bounding box proposals are based on the single box most likely box without a second round of non-maximum suppression after the second stage as is more standard, the boxes generally do a good job, even for images that we would not necessarily consider "easy". For example, the Desert Tortoise is half covered by the arm of a woman, the first Leatherback Turtle example blends into the sand quite nicely and the second Leatherback Turtle example shows many turtles each of which takes up a rather small part of the image. The first stage is clearly very confident that the bounding boxes contain turtletoises: the lowest turtletoise score is 94%.

While the benchmark does better than the second stage model for the Terrapin example, having Terrapin as a top proposal with 16% while the second stage ranks it second with 11%, it appears that this image does not actually show a real Terrapin but rather a model of it. For the Pacific Ridley, both second stage and benchmark get it wrong proposing a Green Turtle as most likely. However, the benchmark model is much more certain of this with 54% compared to 23% for the second stage. Meanwhile, the correct species does not show up in the top-3 for the benchmark model, while it is a close second with 21% for the second stage. The benchmark also mis-classifies the Giant Tortoise as a Desert Tortoise (60%) with the Giant Tortoise (13%) coming third. The second stage on the other hand attaches the correct label with 97% certainty. The Desert Tortoise example is interesting: this is exactly where we would expect the two-stage model to outperform conditional on finding the right box since the image seems quite "busy" with a woman as a "distraction" and the colors seem somewhat off on top of it. And indeed,

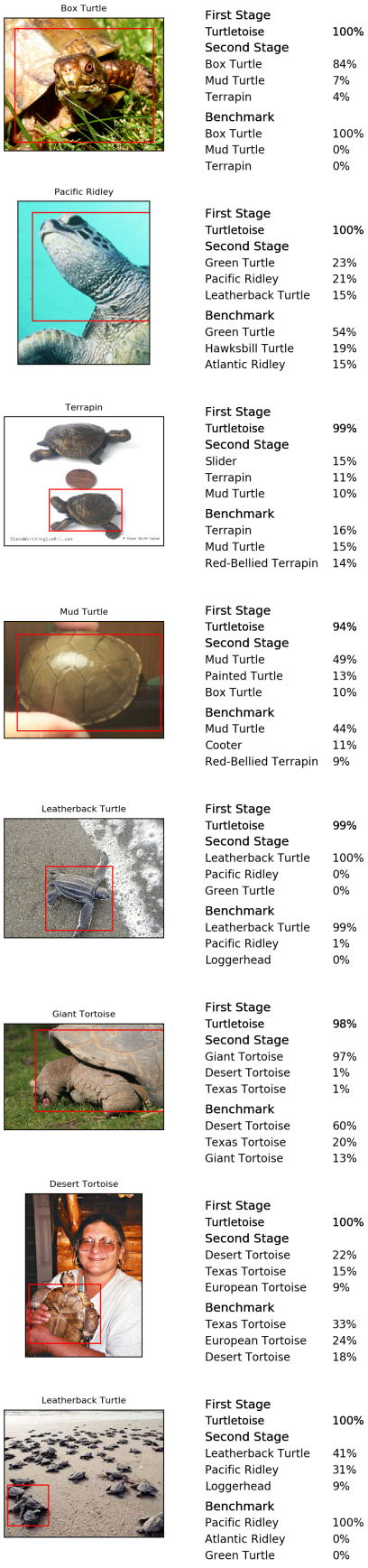


**Figure 11.** Examples Classifications from the Test Set

the second stage attaches the correct label with 22% certainty while the Desert Tortoise comes third for the benchmark which believes that the image most likely shows a Texas Tortoise (33%). We can tell a similar story for the Leatherback Turtle at the bottom which the benchmark model believes shows a Pacific Ridley with virtual certainty. Meanwhile, the two-stage model classifies this image correctly (41%). For the remaining examples, showing a Box Turtle and a Mud Turtle, both models accuractely classify the species.

## 5.2 Reflection

We trained a model that takes an image as input, finds a box containing a turtletoise, and finds the most likely species of it. The model has two stages with the first consisting of a binary *Faster R-CNN* which finds the box, and the second corresponding to a convolutional neural network based on *VGG16* that classifies the species of the turtletoise in the box. The model beats a benchmark single-stage model based on *VGG16* handily when it comes to classification performance as measured in terms of top-1 and top-3 error.

The most interesting part of the project is that we managed to improve classification accuracy even though we did not have bounding boxes for all turtletoise species and generally only for a rather small number of images. This meant that we could not use a full-blown *Faster R-CNN* for both detection and species classification but rather had to adopt the two-stage approach. We were unsure how well the first stage would be able to generalize turtletoise features to unseen species and thus how well it could pick out a bounding box for a majority of images. Given this challenge, we were unsure whether the two-stage model would be able to match let alone beat the benchmark. Thus, we are rather happy with the results.

Admittedly, we could have produced an even tougher benchmark by not just warping the original images to the input size of the CNN but rather by sampling several random areas from the image, warp and classify each one and take the mean over classifications as an output for each image as in commonly done in practice. However, perhaps even more interesting than just the improvement in classification accuracy compared to the benchmark is the opportunity offered by the two-stage model to learn to produce and correctly classify bounding boxes for labels for which no bounding box information is available at all and this is something the benchmark model cannot do. We believe that the free form visualization above shows that the two-stage model has great potential for this. Thus, we believe that the two-stage approach offers exciting opportunities for further research.

## 5.3 Improvement

There is an obvious potential to increase classification performance. For example, we did not pretrain the first stage *Faster R-CNN* other than by using *ImageNet* weights which were not designed for object detection tasks. Thus, pre-training the first stage for this task on data sets such as PASCAL VOC or MS COCO may boost the first stage quality. Next, we could consider replacing the *VGG16* architecture which may

be considered slightly outdated and has been beaten by several models such as ResNet He et al. [2015] or Inception Szegedy et al. [2014]. We also found evidence of over-fitting, indicating too little regularization. Thus, we could use dropout for more layers or introduce batch normalization Ioffe and Szegedy [2015]. Somewhat more tinkering to the specific task at hand, we could employ a "within-image-ensemble": while each image in the data at hand only has a single label, there may be more than one turtletoise of the same species in each image. We could therefore allow *Faster R-CNN* to propose several region with detected turtletoises per image. We could then feed several cropped regions per image to the second stage and take for example the mean over the classification output for all regions belonging to one image. This may improve performance at test time as ensemble learning often does.

An idea aimed at post-training computational speed rather than classification performance is to look at the first stage model and evaluate how similar how closely the objectness score of the RPN lines up with the output of the classifier turtletoise-classification. If the numbers are very similar, we could reasonably reduce the number of regions of interest, possible even two unity. This could substantially reduce the number of forward passes through the classifier block per image and thus speed up computations.

Finally, pursuing a quantitative evaluation of the performance of a two-stage model to both locate and classify potentially multiple unseen species per image would be of interest.

## References


Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

Adit Deshpande. The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3), 2016. URL `https://goo.gl/zAoev2`.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint*, pages 1–10, 2014.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 International Conference on Computer Vision, ICCV 2015:1440–1448, 2015.

Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. *arXiv*, page 10, 2017.

Dhruv Parthasarathy. A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN, 2017. URL `https://goo.gl/V7nCFA`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

Tomasz Grel. Region of interest pooling explained, 2017. URL `https://blog.deepsense.ai/region-of-interest-pooling-explained/`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfittin. *Journal of Machine Learning Research 15*, 15:1929–1958, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385v1*, 7:171–180, 2015.

François Chollet and Others. Keras, 2015. URL `https://github.com/fchollet/keras`.

Yann Henon. keras-frcnn, 2017. URL `https://github.com/yhenon/keras-frcnn`.

Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. *Computer Vision–ECCV 2014*, 8689:818–833, 2014.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256, 2010.

Sebastian Ruder. An overview of gradient descent optimization algorithms. pages 1–14, 2016.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Chapel Hill, and Ann Arbor. Going Deeper with Convolutions. pages 1–9, 2014.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning,*, volume 37, 2015.