2D Image Orientation Estimation of Turtles

Henry Grover

May 2019

1 Introduction

Straightening photos to their upright position is a fundamental problem that has numerous practical outlets, from the interpretation of documents to correcting vacation photos. The priority of being able to read scanned documents lead to a solution that was found long ago. Documents, however, exploit the unique structure of text images such as the layout of each line or the shapes of letters. Other practical orientation estimation areas include simplified angle estimation, such as inferring either portrait or landscape layouts or using line detection to correct tilt. The orientation estimation problem is at heart a convolutional neural network problem. This implementation explores the possibility of inferring rotation estimation from annotated datasets using a densely connected convolutional neural network in order to maintain a consistent orientation of animal images with the intent of feeding these images into a detection network.

2 Design Decisions

In this implementation, various design decisions were made to optimize not only network training, but also for data cleaning and to avoid unreliable results. The network uses a negative loss likelihood gradient function for classification and a custom L1 loss for regression. The regression gradient function uses traditional L1 loss followed by an operation that ensures the following:

$$360 - |y - \hat{y}| \text{ if } |y - \hat{y}| > 180 \text{ else } |y - \hat{y}|$$

to counteract the wrap around problem which occurs when comparing angles such as 1 and 359 that are logically 2 degrees away but numerically 358 units away. Instead, angles above 180° are "wrapped" back towards zero so that when an angle approaches 360, it is actually approaching zero.

Positive degrees were utilized where possible to avoid issues concerning the vanishing gradient problem. When target values are smaller they are at a higher risk of degrading towards zero within recurrent networks with a large number of layers. Luckily DenseNet offers improved information flow, but using degrees rather than radians further reduces the risk of a vanishing gradient.

Pre-processing is used to load images into memory. This is necessary because the images and the annotations are located in separate folders, therefore in order to avoid loading each image and annotation when collecting the current batch (this is equivalent to loading the dataset again at each epoch), they are stored as pairs within a list ready for data augmentation. The pre-processing decision also avoids various errors down the road, such as those arising from lack of data cleaning. For example, the mean and standard deviation must be calculated in order to normalize the images as part of standard transforms. In order to reject images that are either poorly annotated or for some other reason cause an error during augmentation, a test augmentation is applied to each image found in the directory to test if it makes it through without error. This test augmentation ensures that during batch selection the images do not cause any issues during each unique augmentation. Images that are labelled incorrectly or cause an error during test augmentation are not included in the dataset and therefore the final mean and standard deviation are not skewed since we only pre-load the images we intend to train on.

Data augmentation consists of rotating and cropping solely the head of the turtle so that it is facing upward at a universal 0° angle. It can then be augmented by some random angle that is then used as the image label. The region of interest crop of the head is the maximal inscribed rectangle that results from the rotated minimal enclosing bounding box of the original annotation. An example is shown below in Figure 1.

Course angle estimation is trained to estimate within 45° of accuracy using 4 discrete classification angles. This allows a regression network to assume that the input image is no more than 45° away from its true orientation.

3 Neural Network Framework

Neural network framework selection is an important design decision. It can be equally as important as sufficient training data or any optimization. A poor choice in framework can only be optimized so much, where a well selected network framework can be tuned to near perfection. This being said, DenseNet was chosen as it is a newer variation of a recurrent network using a number of fundamentals proven from networks like ResNet. DenseNet utilizes feature reuse, which means the model is easy to train and highly parameter efficient because it eliminates the need for learning redundant feature maps that were previously computed in earlier layers of a dense block.Layers can be considerably narrower than other frameworks because it adds only a small set of feature maps to the "collective knowledge" of the network. This is important because the network includes multiple stages of training including both classification and regression algorithms.

DenseNet is known to be rather memory hungry and takes a longer time to train, but has shown higher accuracy rates on standard datasets. It's implicit deep supervision allows individual layers to receive additional supervision from the loss function from shorter connections that are impeded by fewer layers to get to its destination. On top of higher accuracy, DenseNet offers detailed documentation and simplicity help in analyzing both problems and further performance.

In order to preserve computational efficiency, the current implementation has been designed after the DenseNet-BC concept. This version uses a bottleneck and a compression factor to reduce the size as well as number of feature maps.

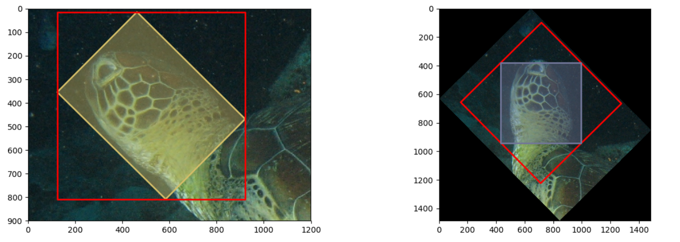
4 Data Augmentation

Data augmentation is crucial to preserve the generality of the features learned by the network. While a ground truth is known from annotation, training with this value results in memorization because it sees the same image and angle during each epoch, leading to overfitting and therefore poor performance outside the training set. Instead, the annotated angle is used to orient the image at it's natural orientation as a universal zero. From here we can augment the image with a random angle and transformation to ensure that the network never sees two of the same image/angle pair. This increases the feature generality so that it is well calibrated for data not previously seen.

During pre-processing, very few images are thrown away due to errors during test augmentation. There are a total of 2674 image in the training set, 940 in the validation set, and 904 in the test set. This means that 99.58% of images are able to be used of the total number of images in the complete dataset, $\frac{4,518}{4,537}$. The dataset itself is rather small, therefore data augmentation is extremely important.

On animal images, the area of interest is specified by a bounding box and this is the only area that should be evaluated, however rotation becomes more complex because we have more true pixels than are solely within the bounding box, therefore a simple maximal inscribed rectangle is less crucial. Instead, a combination of the maximal inscribed rectangle and the minimal enclosing rectangle is needed. The region of interest within each image is the head of the turtle, which is defined by MSCOCO style annotation. With each annotation (there may be multiple in each image), important attributes include the segmentation points around the head, the viewpoint of the image (left side, top, ect.), and the angle of the segmentation (which translates to the angle of the head).

The data is formatted in MSCOCO style with folders of images for each training, validation, and testing, as well as a folder for json files of annotations for each. Using the official python API, the annotations were verified to be accurate for almost all of the images. The API is necessary for this problem because it will be how the segmentation is extracted which specifies where the image should be cropped before it is fed into the estimation network. The segmentation and a labelled theta in radians will act as the input image and corresponding image label to train the network. This minimal enclosing rectangle does not exceed


(a) The annotated segmentation with surrounding minimal enclosing rectangle to be rotated with the image.

(b) The rotated minimal enclosing rectangle from Fig. 1a, with axis aligned maximal inscribed rectangle.

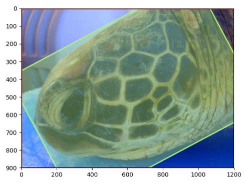
Figure 1: The segmentation and bounding box before rotation, and the rotated bounding box with final minimal inscribed crop.

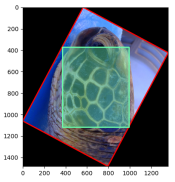
the image dimensions, even if the segmentation does. This ensures that the maximal inscribed rectangle does not include any black background pixels generated during image rotation. The final crop is the maximal inscribed rectangle.

For $[-45, 45]$ degree augmentations, labels are modified in order to maintain positive target values. The range is actually $[0, 45]$ and $[315, 360]$ degrees, essentially equivalent to the 360 degree model with the middle 270 degrees removed. The same L1 error can be used, and the results are do not suffer.

5 Challenges Faced

Before pre-processing, the API must be used to locate each filename and load the specific annotation and related image when the images are selected for each batch. This computation is happening repeatedly during training rather than as a single initialization measure. The other challenge is a logical decision about what to do when the minimal enclosing rectangle is located beyond one of the edges of the image. Currently the region of interest is cropped to preserve pixel correctness and exclude any pixel not present in the bounding box of the original segmentation. While the theoretical algorithm produces good training images, a significant portion of the dataset contains annotations that exceed the dimensions of the image. This limits the ability of the rotating and cropping algorithm to capture a crop that represents enough of the region of interest from the original annotation. If the annotation take up most of the original image, there is a chance that this interferes with the accuracy of final prediction because the network is trained on images with a very small window of the region of interest. Examples of this are provided in Figure 2.


(a) Original annotation with bounding box (entire frame).


(b) Rotated segmentation bounding box (entire rotated frame) with maximal inscribed rectangle.

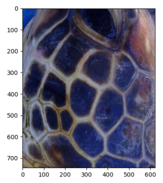

(c) Crop of the rotated image from the maximal inscribed rectangle bounds.

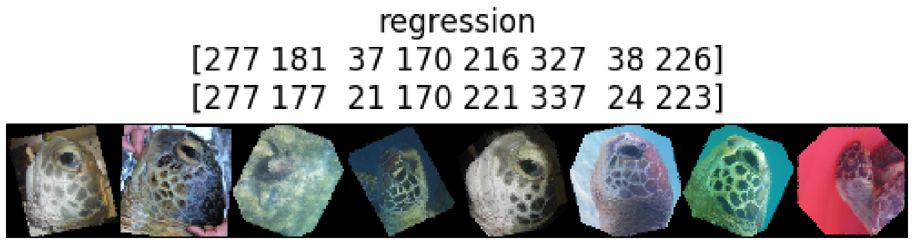
Figure 2: An example where the annotation causes the final crop to only include a small part of the original segmentation.

More work needs to be done to filter out images where the annotation coordinates are severely outside the bounds of the image. When they are rotated and cropped they end up showing only a small part of the original annotation. This can potentially be fixed by making a more intelligent algorithm to go outside the rotated minimal enclosing rectangle of the original segmentation, but still focus on the turtle head as well as not include any of the black background pixels the appear from image rotation. If this intelligent crop is not possible, the image should not be loaded into the dataset, determined by the of areas of the final crop to the original segmentation.

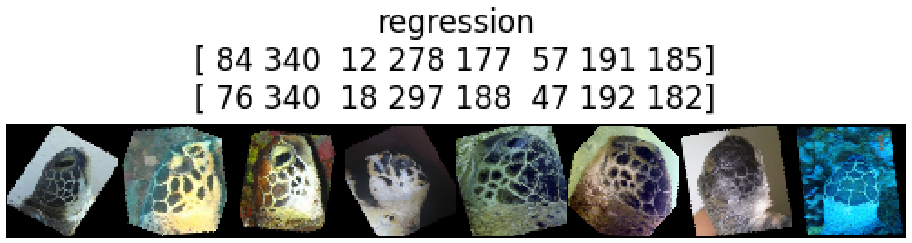
Additional challenges arose around normalization during testing. Means and standard deviations are computed during pre-processing, and used when loading each batch to normalize an image. Originally computed for each set (training, validation, and testing), an error arose where setting the model to either train or evaluate brought an inconsistency to these normalization values when applied to the images. Means and standard deviations cannot be computed over multiple sets of images due to laws of data snooping, however the error lies in the way PyTorch handles these modes. batch normalization uses tracked running statistics to use during batch normalization instead of those generated from the current batch data. This interfered with the resulting prediction angles, therefore inference mode has been temporarily until further investigation can be done as to why a running mean and standard deviation lead to poor testing results. As a side note, this error has been isolated when, all else held constant, eval() was replaced with train() to result in significantly better results.

6 Results

Random examples of images and their predicted output were selected to display the process of rotating the images back to their inferred natural orientation. Below are various examples of a 360° regression algorithm being applied to the turtle dataset. In this case, the cascading network is not used in order to isolate the performance of solely the regression network.



(a) Example 1 of predicted turtle images.



(b) Example 2 of predicted turtle images.

Figure 3: Two example output predictions of turtle head rotations. The top row of angles is the ground truth, and the bottom is the predicted rotation.

The images in Figure 3 show promising results. All predictions are within 20° of their ground truth, and the majority are within 10°. This is only a small subset of images, however they serve to visualize that small error is relatively acceptable. In Figure 4 is the full error distribution in the form of a confusion matrix or true and predicted labels, and Figure 5 is a histogram of angle errors. The mean error is 6.11° and the median error is 3.91°, which means that there is a small tail of errors that extend to high degrees, but it is not significant.

Distribution of error can be seen to congregate around zero with few instances that extend far beyond the mean, and the confusion matrix shows that the majority of predictions follow the ground truth diagonal.

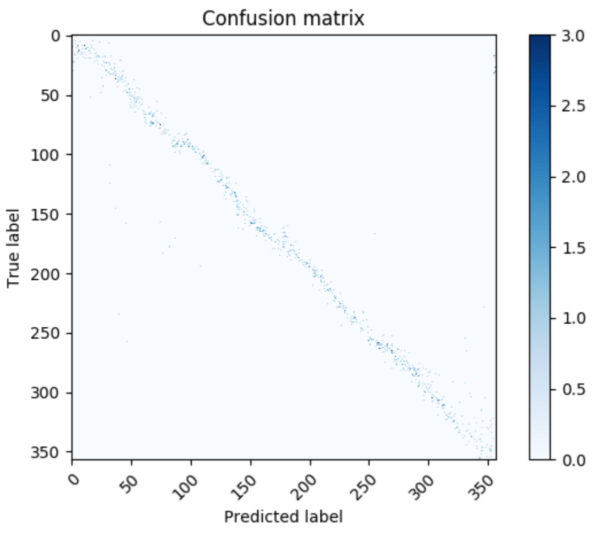


Figure 4: Confusion matrix of the 360° regression network.

Overall the predictions seem to closely follow the negative diagonal line, which represents a true estimate. It is a relatively narrow line that deviates only slightly. The Histogram shows that the vast majority of angle estimates are within 20°, which translates into ±10° on either side of ground truth.

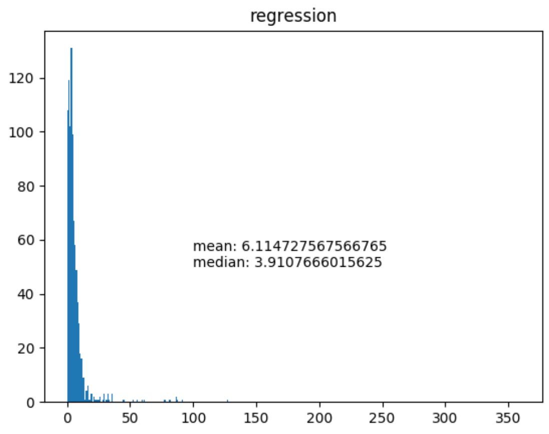


Figure 5: Histogram of error distribution for the 360° regression network.

The cascade network makes use of the classification network with 4 discrete outputs feeding into a regression network that assumes no larger deviation in either direction than ±45°. The classification network results in the following confusion matrix, shown in Figure 6 The histogram of errors was not useful as

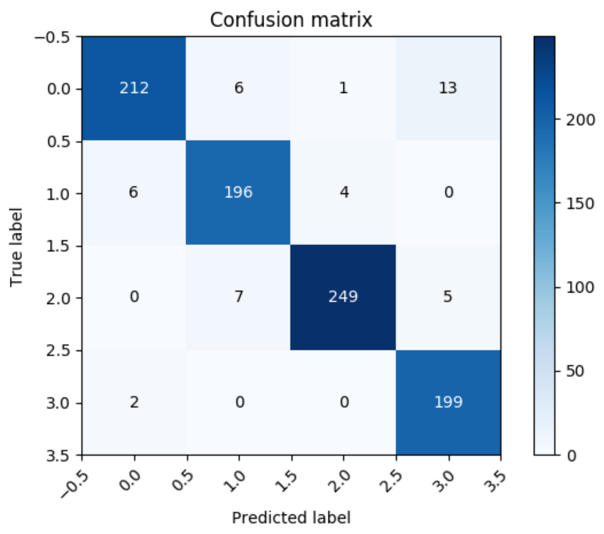


Figure 6: Confusion matrix of the classification network with 4 outputs.

errors are more easily seen in the confusion matrix. The mean error was 0.0833, and the median error is 0.0. When run on the test set, we get an error rate of 4% (44/904). In some cases, the error has been observed to be as low as 2%. Figure 7 shows a few examples with the predicted and ground truth quadrant of the images.

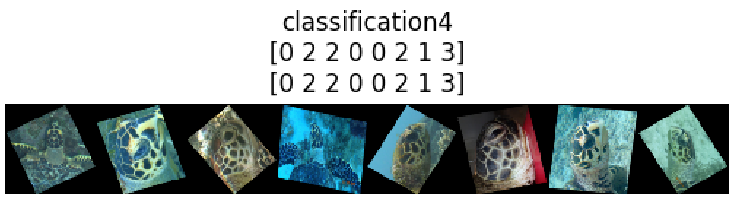


Figure 7: Examples of the classification network with 4 outputs.

The regression network shows good results as well. Figure 8 are the confusion matrix and error histogram graphs for the ±45° regression network. The error and error distribution for the ±45° regression network is considerably smaller than that of the 360° network. This is the advantage of using the cascade network, however the possible downside is that any errors that arise in the

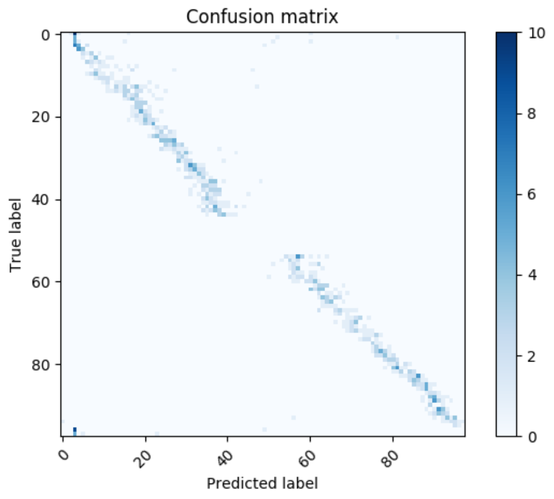


Figure 8: Confusion matrix of the ±45° regression network curved around 45°.

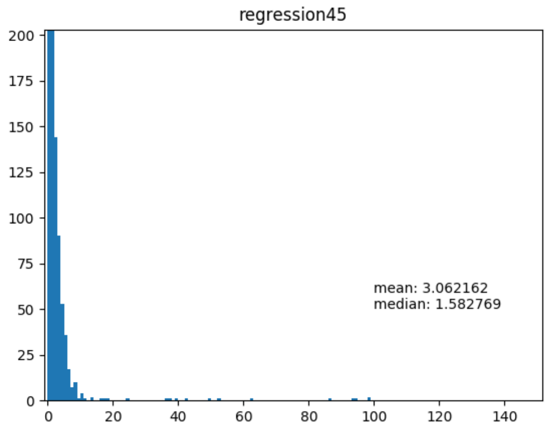


Figure 9: Error distribution of the ±45° regression network.

cascade network is twice as likely to happen, because if either one of the networks are incorrect in their prediction then it is guaranteed to be wrong. A possible solution and an interesting future work is to combine multiple networks to guess the same stage. That is, the 4 output classification network as well as the 360° regression network could both estimate the course estimation and line up their outputs to validate sending it to the stage 2, more precise ±45° regression network, which could also be paired with a second network to double check the final orientation estimation.