Computer Sciences
SCHOOL OF COMPUTER, DATA & INFORMATION SCIENCES
UNIVERSITY OF WISCONSIN-MADISON

CS 771
Learning-Based Methods for Computer Vision
Assignment 1 Report

Name: Sujay Chandra Shekara Sharma
Wisc ID: 9085871714
Wisc Email: schandrashe5@wisc.edu

3.2 Image Processing

3.2.1 Image Resizing

The **Scale** class can be used to resize an input image to a desired target size based on the input arguments. There are two cases handled by the **Scale** class with respect to the target size:

1. If the size argument is a tuple of (width, height), then the image is resized to the target width and height.
2. If the size is a single integer, the smaller edge of the image is matched to the target size, and the longer edge of the image is scaled proportionally.

The **Scale** class also accepts an optional interpolations argument, which determines the interpolation method used for resizing. If not specified, a default set of interpolation methods is used, with bilinear interpolation as the fallback.

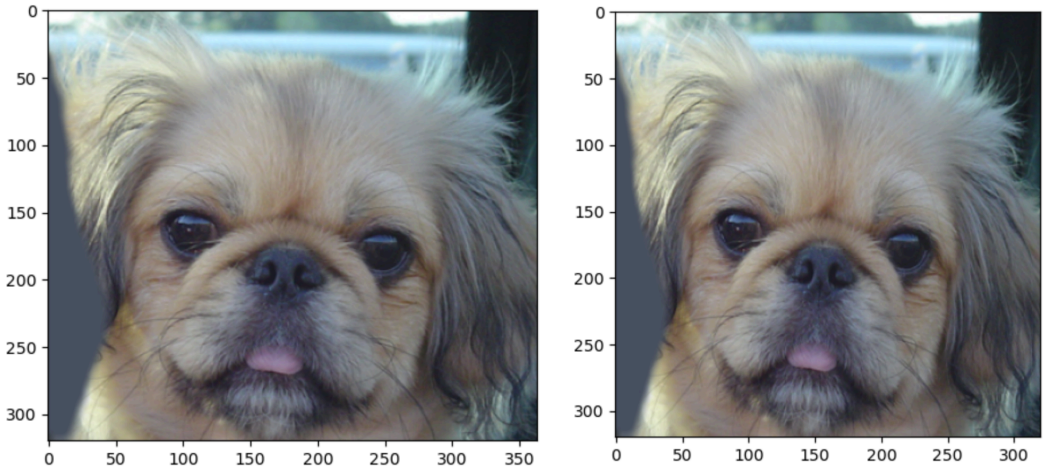


Figure 1: Result of calling Scale(320) and Scale((320,320)) respectively. Image 1 on the left has dimensions (363, 320). Image 2 on the right has dimensions (320,320).

3.2.2 Image Cropping

The **RandomSizedCrop** class can be used to crop a random region within an input image and then resize it to a desired target size based on the input arguments. Similar to the **Scale** class, there are two cases handled by the **RandomSizedCrop** class with respect to the target size:

1. If the size argument is a tuple of (width, height), then the image is resized to the target width and height.
2. If the size is a single integer, then the image is resized such that both the width and height are matched to the target size. One thing to note is that this behaviour is different from the corresponding case for the **Scale** class which scales the longer edge of the image proportionally.

In this implementation, the target area and aspect ratio of the cropped image are randomly sampled from a uniform distribution whose range is specified by an argument. The height and width of the cropped image can then be computed from the target area and aspect ratio as shown below:

$$cropped\ width\ = \sqrt{target\ area\ *\ aspect\ ratio}$$

$$cropped\ height\ = \sqrt{\frac{target\ area}{aspect\ ratio}}$$

After calculating the cropped width and height, **RandomSizedCrop** checks if the crop fits in the original image dimensions. If it does, then the top left corner of the cropped image is randomly selected (design choice to select a corner instead of the center for easier computation of the

coordinates of the cropped image). Then the image is cropped and resized based on the logic specified above. If a valid crop couldn't be found after a specified number of trials, then the entire image is resized to the target size based on the logic specified above.

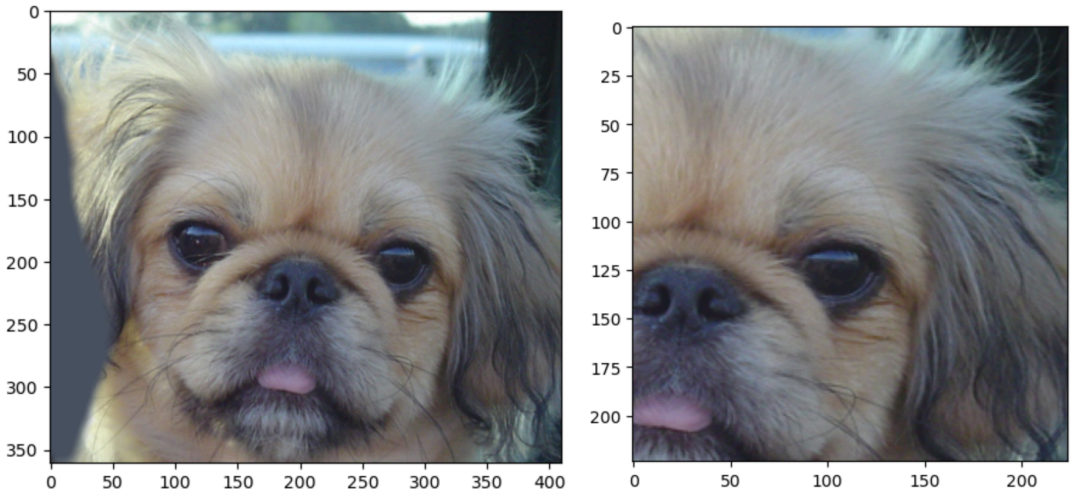


Figure 2: Original image on the left and the result of calling RandomSizedCrop(224) on the right. Image 1 on the left has dimensions (410, 361). Image 2 on the right has dimensions (224, 224).

3.2.3 Color Jitters

The **RandomColor** class can be used to apply random color perturbations to an input image by independently adjusting each color channel (red, green, and blue) within a specified range. Random perturbation factors (alphas) are first sampled for the red, green, and blue channels based on a specified color range argument. The numpy array representing the image is then converted to store floating point values. These factors are then applied multiplicatively to each color channel, adjusting the color intensities. The perturbed image values are finally clipped to ensure they remain within the valid pixel range (0–255) and converted back to uint8.

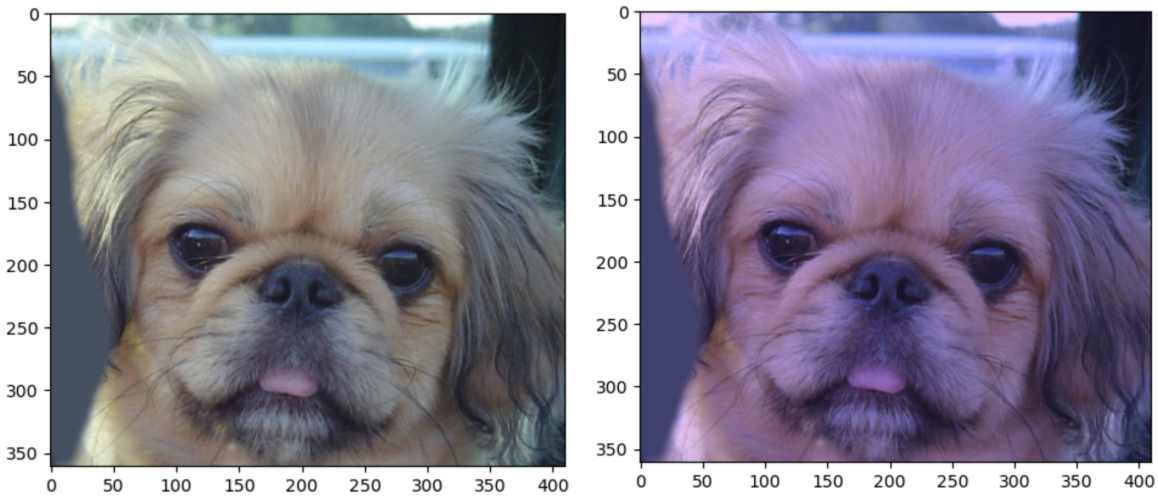


Figure 3: Original image on the left and the result of calling RandomColor(0.2) on the right.

3.2.4 Rotation

The **RandomRotate** class performs random rotations on an input image while preserving the maximal area inside the rotated image such that no empty pixels are included in the final rotated image. A random rotation degree is first sampled from a uniform distribution based on the rotation range input argument. The image is then rotated with the help of OpenCV's **warpAffine** function and the new bounding dimensions are calculated to fit the rotated image without cutting off any part of the original image. A function is then used to calculate the largest possible rectangular area within the rotated image such that no empty pixels are present in the selected rectangular area. This area is finally cropped and returned as the resulting image.

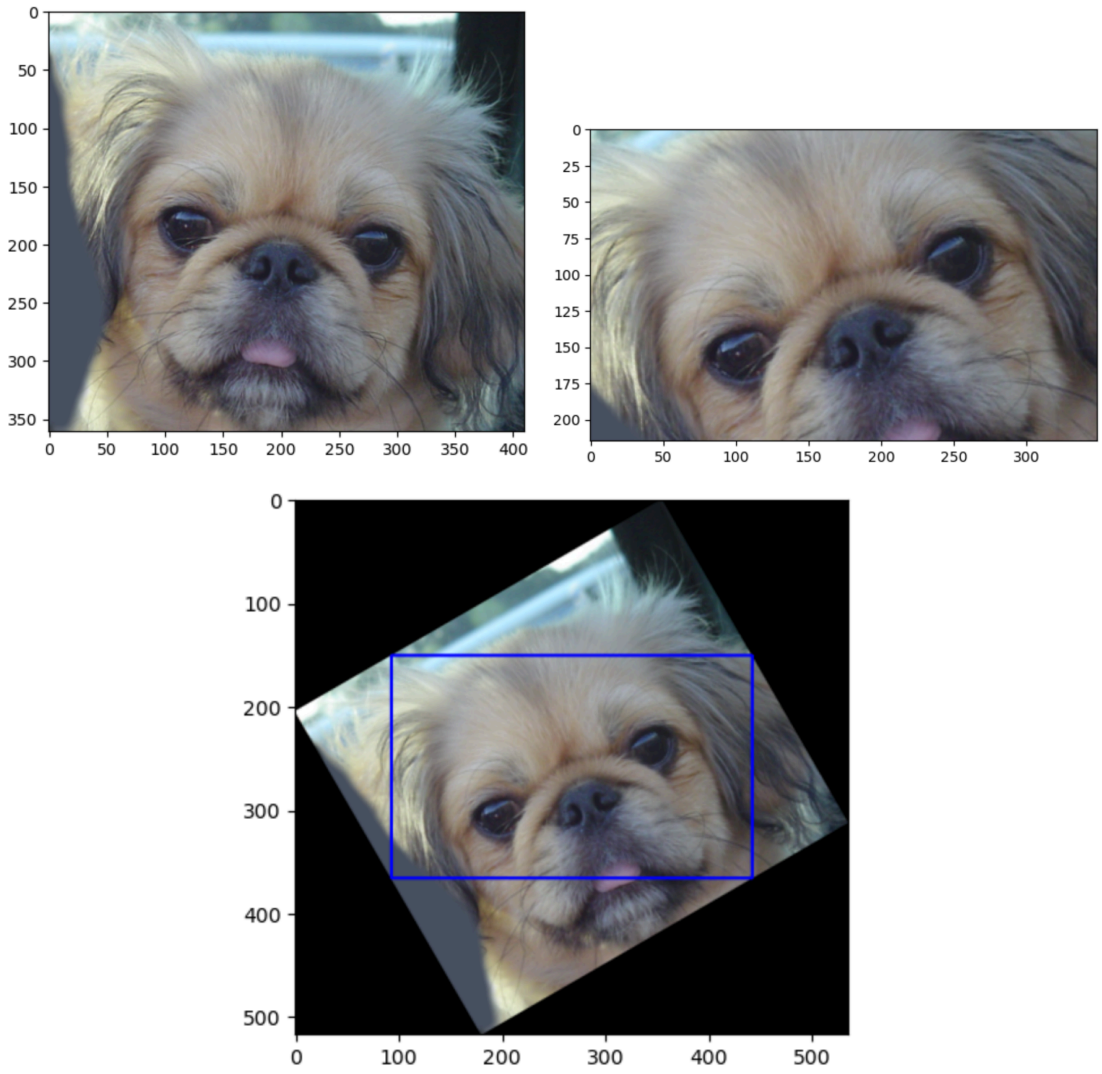


Figure 4: Original image on the top left and the result of calling RandomRotate(30) on the top right. In this case, the image was rotated by exactly 30 degrees. The image at the bottom is an illustration of the rotated image along with its bounding region and the corresponding maximal rectangle highlighted in blue. The dimensions of the original image on the top left are (410, 361). The dimensions of the rotated image on the top right are (349,215). The dimensions of the illustration on the bottom are (535,517).

The screenshots of the proof of how to compute the maximal area rectangle are attached at the end of the report.

3.3 Data Augmentation and Input Pipeline

3.3.1 Benefits of using PyTorch Dataloader

PyTorch **DataLoader** wraps an iterable around the PyTorch **Dataset** primitive to enable easy access to the samples stored in the PyTorch dataset. The main benefits of using **DataLoader** over a custom solution that iterates over each sample of the dataset in a simple loop are:

1. **DataLoader** automatically batches images into mini-batches based on the specified batch size.
2. **DataLoader** reshuffles the data at the end of every epoch of training to reduce the chances of your model overfitting.
3. **DataLoader** supports multi-threading and it uses Python's multiprocessing to speed up data retrieval and process images in parallel.
4. **DataLoader** provides a simple interface that is well integrated with other PyTorch operations making it easier to use compared to a manual solution.

All of these benefits make PyTorch's **DataLoader** simpler to use and more performant compared to a custom solution that iterates over each image in a dataset.

3.3.2 Order of Transformations in Data Augmentation

Yes, the order of the transformations applied during the data augmentation pipeline can have an impact on the final augmented image that is being generated. Consider the case of an input image whose dimensions are (410, 361). If we apply the following two transformations as a part of the data augmentation pipeline - Scale(450), and RandomSizedCrop(100), the results can be quite different based on the order of these transformations.

Scenario 1: Scale(450), RandomSizedCrop(100)

In this case, the entire image is first scaled to be of dimensions (511, 450). After this, a random area of the image is cropped and resized so that the final output image dimensions are (100, 100).

Scenario 2: RandomSizedCrop(100), Scale(450)

In this case, a random area of the image is first cropped and then resized to be of dimensions (100, 100). After this, the cropped portion of the image is scaled so that the final output image dimensions are (450, 450).

As shown in the example above, the dimensions of the resultant augmented image can vary significantly based on the order of transformations.

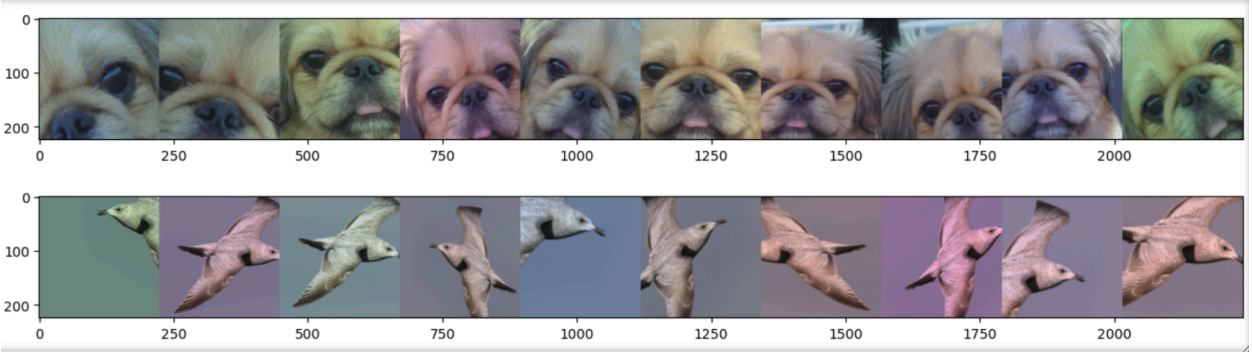
Based on the order of the transformations applied, we could end up with severely distorted images which might not result in effective data augmentation.

Transformed Images

Applying the transformations in the below order results in the following output images:

1. Scale(320)
2. RandomHorizontalFlip()
3. RandomColor(0.15)
4. RandomRotate(30)
5. RandomSizedCrop(224)

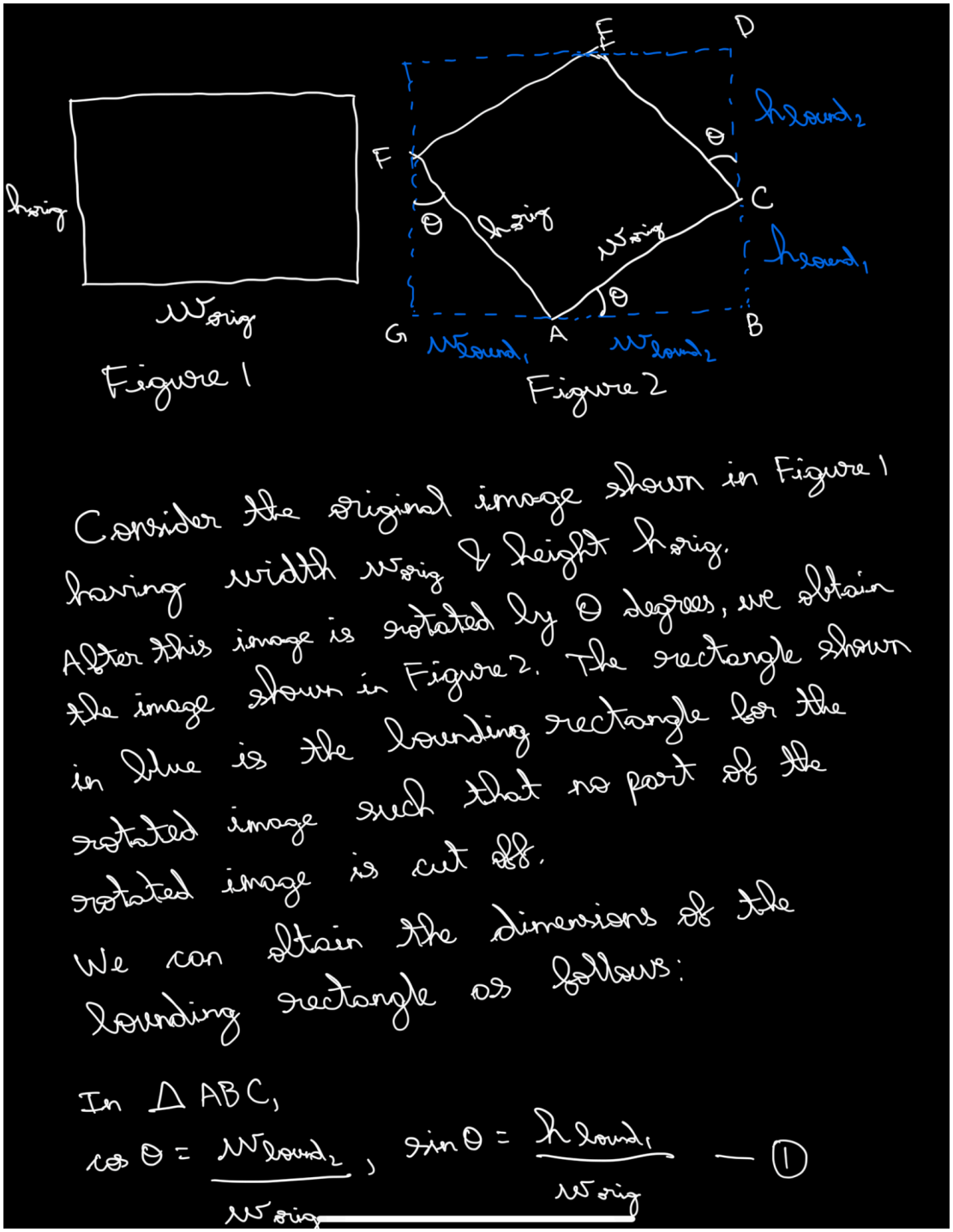
As we can see, using these transformations in a data augmentation pipeline can be an effective technique for increasing the training data size and making our model more robust as we expose the model to variations of an image that can mimic what is seen in the real world so long as the transformations are applied sensibly.



References

1. https://pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/
2. https://stackoverflow.com/questions/16702966/rotate-image-and-crop-out-black-borders/16778797#16778797
3. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#preparing-your-data-for-training-with-dataloaders

Proof of 3.2.4



Consider the original image shown in Figure 1 having width w_{orig} & height h_{orig}. After this image is rotated by θ degrees, we obtain the image shown in Figure 2. The rectangle shown in blue is the bounding rectangle for the rotated image such that no part of the rotated image is cut off.

We can obtain the dimensions of the bounding rectangle as follows:

In ΔABC,

$$\cos\theta = \frac{w_{bound_2}}{w_{orig}}, \quad \sin\theta = \frac{h_{bound_1}}{w_{orig}} \quad -(1)$$

In Δ AGF,

$$\sin\theta = \frac{w_{sound_1}}{h_{sig}} \quad -(2)$$

In Δ CDE,

$$\cos\theta = \frac{h_{sound_2}}{h_{sig}} \quad -(3)$$

We know

$$w_{sound} = w_{sound_1} + w_{sound_2}$$

$$h_{sound} = h_{sound_1} + h_{sound_2}$$

From (1), (2), (3)

$$w_{sound} = h_{sig}\sin\theta + w_{sig}\cos\theta \quad -(4)$$

$$h_{sound} = w_{sig}\sin\theta + h_{sig}\cos\theta$$

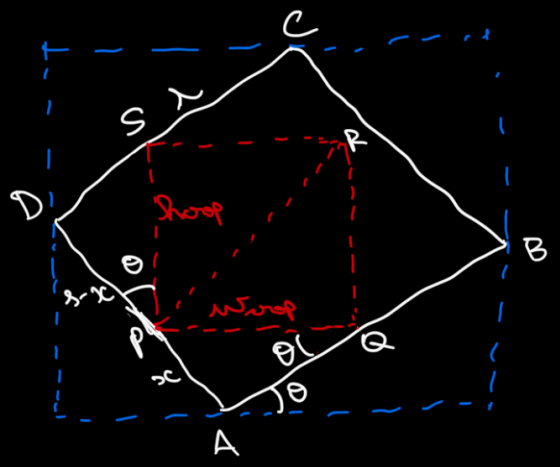


Figure 3

Now consider the maximal area rectangle shown in red in Figure 3. Our goal is to find the width w_{oop} & height h_{oop} of this rectangle. Let us assume that this rectangle touches side AD of the rotated image at point P.

Let $|AD| = s$, $|AP| = x$, $|PD| = s - x$, $|CD| = l$, $s \leq l$

In Δ APQ,

$$\sin\theta = \frac{x}{w_{oop}} \quad - (4)$$

In Δ PDS,

$$\cos\theta = \frac{s-x}{h_{oop}} \quad - (6)$$

From (5), (6),

$$x = w_{oop} \sin\theta \,, \quad s - x = h_{oop} \cos\theta$$

$\therefore$ Maximizing the area of the rectangle $w_{oop} \times h_{oop}$

$\Rightarrow$ Maximizing $(x)(s-x)$

$$f(x) = (x)(s-x)$$

$$\frac{df(x)}{dx} = (s-x) + x(-1) = s - 2x$$

To maximize, $\frac{d\,f(x)}{dx} = 0$

$\therefore x = \frac{s}{2}$

This maximum area is only valid if the diagonal PR fits in the rotated image ie $|PR| \leq |CD|$
$\hookrightarrow \lambda$

In $\triangle APQ$,

$\cot\theta = \frac{AQ}{x} \quad \therefore AQ = x\cot\theta = \frac{s\cot\theta}{2}$

In $\triangle PDS$,

$\tan\theta = \frac{DS}{s-x} \quad \therefore DS = (s-x)\tan\theta = \frac{s\tan\theta}{2}$

Since PR can be written as AQ+DS, we have

$|PR| = \frac{s}{2}(\cot\theta + \tan\theta) = \frac{s}{2\sin\theta\cos\theta} = \frac{s}{\sin 2\theta}$

$\therefore s \leq \lambda \sin 2\theta$ for $x = \frac{s}{2}$ to be the correct solution.

solution.

If $s > \lambda \sin 2\theta$, then PR won't fit in the rotated image & we need to find $x < \frac{s}{2}$ such that all four corners of the desired rectangle are on a side of the rotated image.

$\therefore PR = \lambda$

$\therefore x \cot\theta + (s - x)\tan\theta = \lambda$

$\therefore x\left(\frac{\cos\theta}{\sin\theta} - \frac{\sin\theta}{\cos\theta}\right) = \lambda - \frac{s\sin\theta}{\cos\theta}$

$\therefore x\left(\frac{\cos 2\theta}{\sin\theta\cos\theta}\right) = \frac{\lambda\cos\theta - s\sin\theta}{\cos\theta}$,

$\therefore x = \frac{\sin\theta\,(\lambda\cos\theta - s\sin\theta)}{\cos 2\theta} \quad - (1)$

Now, that we know x for both cases, we can compute w_{crop} & h_{crop}.

If $s \leq l \sin 2\theta$,

From ⑤, ⑥

$$w_{crop} = \frac{s}{2\sin\theta}, \quad h_{crop} = \frac{s}{2\cos\theta}$$

If $s > l\sin 2\theta$,

From ⑤, ⑥, ⑦

$$w_{crop} = \frac{l\cos\theta - s\sin\theta}{\cos 2\theta},$$

$$h_{crop} = \frac{s}{\cos\theta} - \frac{l\cos\theta\sin\theta + s\sin^2\theta}{\cos\theta\cos 2\theta}$$

$$= \frac{s\cos^2\theta - s\sin^2\theta - l\cos\theta\sin\theta + s\sin^2\theta}{\cos\theta\cos 2\theta}$$

$$\therefore h_{crop} = \frac{s\cos\theta - l\sin\theta}{\cos 2\theta}$$

Now that we know w_{crop} & h_{crop}, we can now calculate the horizontal & vertical inserts from the bounding box to get the coordinates of the max area rectangle.

$$x_1 = \frac{w_{bound} - w_{crop}}{2}, \quad x_2 = x_1 + w_{crop}$$

$$y_1 = \frac{h_{bound} - h_{crop}}{2}, \quad y_2 = y_1 + h_{crop}$$