Practiques en Empress i Institutions - Report

6/12/2020

Personal Data

Student: Gael Ruta Gatera

NIU: 1542977

Company

Mitiga Solutions SL

carrer de bosc, 38

Barcelona, 08017

Tutors

Jordi Morera and Roger Pastor

Project Assigned: Volcanic Ash Emisstion Image Detection

The company started a project titled "Volcanic Ash Emisstion Image Detection" with the aim of using machine learning techniques in order to detect whether or not there are ash emissions in images coming from volcanoes with the aim of using it as a fast response triggering system. The machine learning model chosen for task was a Convolution Neural Network (CNN) as the task is defined as an image classification problem and the fact that CNN's are the most popular architecture used for image classification [1]. The source of Data are images of volcanoes being monitored by the Mexican Government's Centro National de Prevencion de Desastres (CENAPRED) [2]. The project was initiated by Marc Miranda and handed to Gael Gatera in February 2020 for a spring semester internship program.

Introduction

Stanford University's online course on convolution neural networks identified 7 challenges will arise as a developer tries to create an accurate image classifier: Viewpoint variation, scale variation, deformation, occlusion, illumination conditions, background clutter, and lastly Intra-class variation. Sometimes, despite high performance metrics, image classifiers are found to be classifying incorrectly due to the aforementioned challenges. Before putting images through the convolutional neural network. It was thought that it would be beneficial to process images before having them going through the machine learning algorithm and hopefully limit the aformentioned 7 challenges. The steps would be followed in this order:

a) Greyscale original image

b) Crop original image to region associated with activity

c) Apply various filters to the images Smoothed processed image.

d) Dominant color contrasting from the region, also known as maximum peak, identified using a Laplacian of the Gaussian (LoG) blob detection algorithm.

All processing was done using different functions and methods from the OpenCV library.

Problem Statement

Before and after the mentioned pre-processing steps will be applied, the following questions can be asked: What does this algorithm actually detect? In the case that we wanted to train the algorithm with more images, would there be the option to make the label of the images more intelligent ? Can this algorithm be applied to other volcanoes ? Therefore, the task in this project is to investigate how much the preprocessing can improve the current implimentation. Due to the unbalanced nature of the images, the majority of the images

Convolution Neural Network - Background

What makes Convolution Neural Networks different that other kinds of Neural Networks is that they are based on convolutions [3]. Convolutions can be thought of as small sliding filters that are 'convolving' or sliding around a chosen medium, in this case images, extracting features of interest and creating layers from them. In our model, the convolution layer depends on a few parameters. The receptive field, how big the filters are. Image strides, the way in which the filters slide around the image. Padding, the amount of filters in the neural network. Lastly, the amount of such filters. In our case, the CNN is used as an image classifier that will have images as inputs and the probability that the images either have an emission of ash or not. As previously defined, neural networks have thresholds that are used to compare the accuracy of the implemented training and finally appropriately classify the image. In our model, an image is classified to have an eruption if the predicted probability of having one is greater than 0.5 and no eruption if less than or equal to 0.5. This can be thought of as a spectrum where the higher the number the most likely an ash emission is occurring and the lower the number the less the probability. While tuning the model, the threshold is a parameter that can be changed accordingly. At every given value in the threshold, the actual predictions will vary, and so will metrics such as False Negatives, False Positives, Precision and Recall. Keeping in mind that changing the threshold can compromise the trade-off between the Precision and the Recall.

Image pre-processing Implementation

After some considerations and literature review, one possible improvement to the overall workflow was identified as the images can benefit from being pre-processed before being inserted into a machine learning model. The idea behind processing the images can be narrowed down to optimizing how computers represent images. Which, for standard colored images is represented as having 3 dimensions: x, y, and a depth of 3 color channels, namely, Red, Green, Blue (or RGB for short) [4]. Also can thought of as a 3D matrix with the equation m x n x 3. Figure below for a representation. Value for red, green and blue, each between 0 to 255 a combination of the 3 values creates the colors.

In our case the images are downloaded as 480 pixels wide, 704 pixels tall, and with the three color channels the image consists of 480 x 704 x 3 = 1'013'760 color combinations. The team identified using the Open CV (Open Source Computer Vision) library as the main tool for pre-processing the images.

Step A - Crop Images to region right above vent

The idea behind cropping images is two fold:

reducing the number of color combination that the convolution kernels have to go around convolving (or calculating the dot product) between the values in the patch and those in the kernel matrix.

Specify the area of activity (right above the vent) which is the only area of interest for our classification problem.

In order to test of the hypothesis the data came from two separate cameras. "popoTlamacas2" which monitored the Tlamacas volcano and "popo1" which monitors the Altzomoni volcano. Due to the different

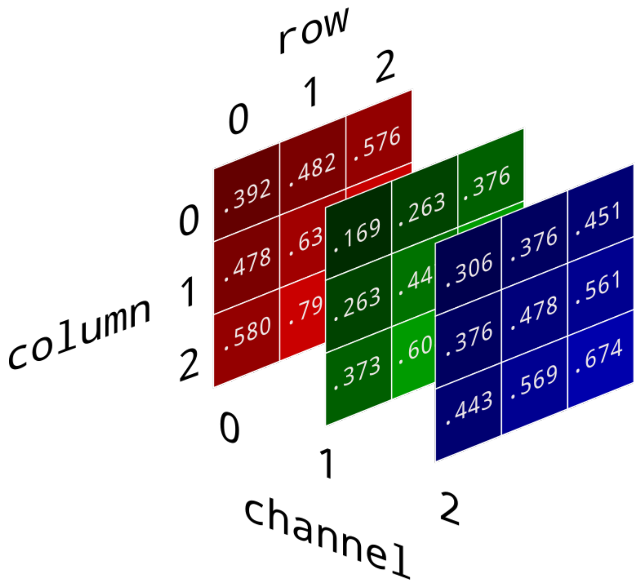


Figure 1: RGB Color Channels [4]

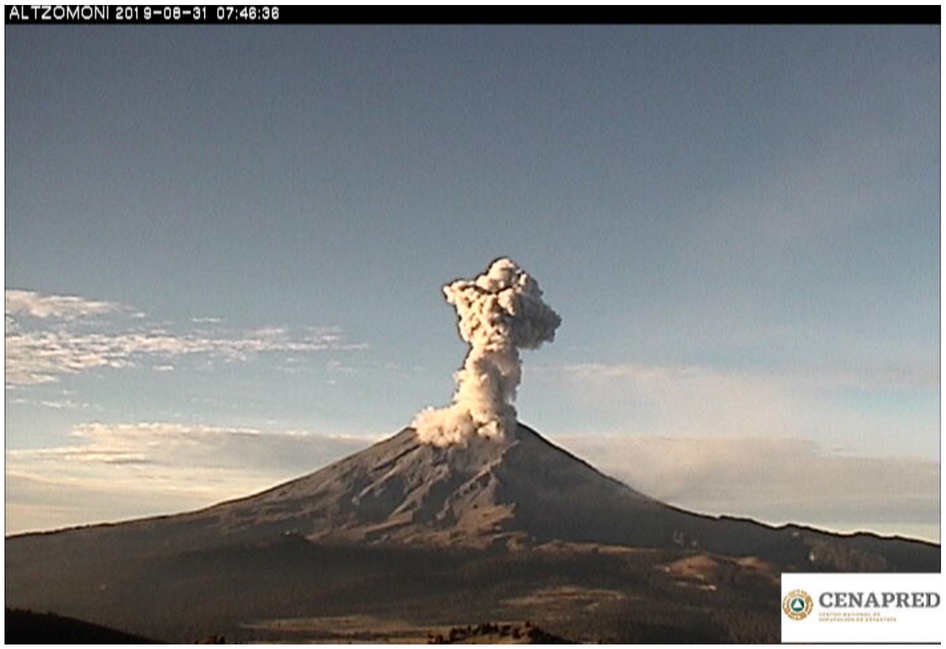


Figure 2: Example of Original Image

positioning of the cameras, they were cropped differently since the area would henceforth be at different places on the picture however the region associate with activity had a width of 360 a height of 184 therefore reducing the color combination to 360 x 184 x 3 = 198'720.

Step B - Greyscaling & Cropping

The reason for greyscaling is in order to reduce the image noise caused by so many pixel values from the RGB channel to simple black and white scales.

Step C - Apply various filters to the images

Just like in one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. LPF helps in removing noise, blurring images, etc. HPF filters help in finding edges in images. After being greyscaled, the following filters were used in order to aid the performance of the blob detection algorithm: 2D conv filter, Gaussian Blur Filter. Different combinations and different oders where tried along the way in order to way in order as often times the blob detection algorithm did not perform as desired especially for distinguishing between light and dark emissions, more on this in subsequent sections.

The assumption here is that the algorithm was detecting bright eruptions (blobs) by selecting the negative responses of the Laplacian together with the maximum value of the gray intensity. On the other hand, when it was only detecting dark emissions it was detecting dark eruptions you should use positive responses to the Laplacian and minimum gray intensity.

2D conv filter

OpenCV provides a '`cv.filter2D()` to convolve a kernel with an image. In our application, we will try an averaging filter on an image with a 5 x 5 kernal. The operation works as following: keep this kernel above a pixel, add all the 25 pixels below this kernel, take the average, and replace the central pixel with the new average value. This operation is continued for all the pixels in the image. Below is a representation of the 5x5 averaging filter kernel will look like the below:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 3: 2D Kernal

Gaussian Blur (smoothed) Filter

The goal of this process is to blur the input images with a low pass filter, which is this case is Gaussian blurring. From reading on the internet, it is seen as an effective method of removing Gaussian noise from an image. The Gaussian distribution in 1 Dimension has the form:

$$G(x) = \frac{1}{\sqrt{2\ \pi\sigma}} e^{-\ \frac{x^2}{2\ \sigma^2}}$$

`Equation 1 - 1D Gaussian`

In 2-D, an isotropic (i.e. circularly symmetric) Gaussian has the form:

$$G(x,y) = \frac{1}{2\ \pi\ \sigma^2} e^{-\ \frac{x^2+\ y^2}{2\ \sigma^2}}$$

`Equation 2 - 2D Gaussian`

OpenCV has the cv2.GaussianBlur() function that uses a Gaussian Kernel where the width and height of the kernel should be positive and odd. Furthermore, the standard deviation in the X and Y directions, sigmaX and sigmaY should be specified accordingly. If the sigma X and sigma Y are given as zeros, they are calculated from the kernel size. The kernel that represents the shape of a Gaussian ('bell-shaped') hump.

Python function: `dst = cv.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]`

Parameters taken:

src - Input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.

dst - Output image of the same size and type as src.

ksize - Gaussian kernel size. ksize.width and ksize.height can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from sigma.

sigmaX - Gaussian kernel standard deviation in X direction.

sigmaY - Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.height, respectively (see getGaussianKernel for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, sigmaX, and sigmaY.

borderType - Pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.

Step D - The Laplacian of the Gaussian Blob detection algorithm

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single graylevel image as input and produces another graylevel image as output.

The Laplacian L(x,y) of an image with pixel intensity values f(x,y) is given by:

$$L(x,y) = \nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2\ f(x,y)}{\partial y^2}$$

`Equation 3`

Since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result [5]. Doing things this way has two advantages:

Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations. The LoG ('Laplacian of Gaussian') kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image [5]. The 2-D LoG function centered on zero and with Gaussian standard deviation has the form:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1\ -\ \frac{x^2\ +\ y^2}{2\ \sigma^2}\right] e^{-\frac{x^2+\ y^2}{2\ \sigma^2}}$$

`Equation 4`

As mentioned, the LoG operator calculates the second spatial derivative of an image. This means that in areas where the image has a constant intensity (i.e. where the intensity gradient is zero), the LoG response will be zero. In the vicinity of a change in intensity, however, the LoG response will be positive on the darker side, and negative on the lighter side. This means that at a reasonably sharp edge between two regions of uniform but different intensities, the LoG response will be:

- Zero at a long distance from the edge,
- Positive just to one side of the edge,
- Negative just to the other side of the edge,
- Zero at some point in between, on the edge itself.

As previously mentioned, the main trouble the algorithm was having performance was is differing between light and dark emissions. A blob detector is not an edge detector which conflicts in purpose from taking the laplacian. There are several implementations of a blob detector. The way it was implemented is using a discretization of the Laplacian combined with a previously filters. Things that had to be kept in mind were, is this Gaussian filtering isotropic (equal sigma for x and y) or anisotropic? What scales (sigma) are you using? The higher the sigma the least prominent the blob will be. How do you set the filter size? This should ensure that the support of the Gaussian is contained.

It gives high response at ridges (negative) and valleys (positive), therefore using a combination of both proved to be challenging. The assumption here is that the algorithm was detecting bright emissions (blobs) by selecting the negative responses of the Laplacian together with the maximum value of the gray intensity. On the other hand, when it was only detecting dark emissions one should use positive responses to the Laplacian and minimum gray intensity. The correct combination was found by applying the 2d conv filter both before and after the guassian bluring low pass filter.

Overall The code had four main sections:

1. Generation of LOG filters
2. Convolving the images with Gaussian filters
3. Finding the maximum peak
4. Drawing the blobs.

Step E - Binarization

The Binarization step was only as an experiment, a lot of trouble came with inputing the Binarized images in Tensorflow however I will still explain our reasoning behind implementing this step. To begin with, is done using NumPy modules. As in previous sections with OpenCV, the images are read in and converted to ndarrays so that pixel values can be easily calculated and processed. In computing terms, Binary in it's basic form is the process of converting numbers into a base 2 system. Other laymen forms of this application wanting a response to be either True or False. If the RGB image is 24-bit (the industry standard as of 2005), each channel has 8 bits, for red, green, and blue—in other words, the image is composed of three images (one for each channel), where each image can store discrete pixels with conventional brightness intensities between 0 and 255. We can follow the lifecycle of the image below, how it is represented as an array and the final result once it is binarized at different thresholds. By binarizing the image we are attempting to give convolution neural network easier work to do. The level of binarization can be adjusted with a certain threshold.

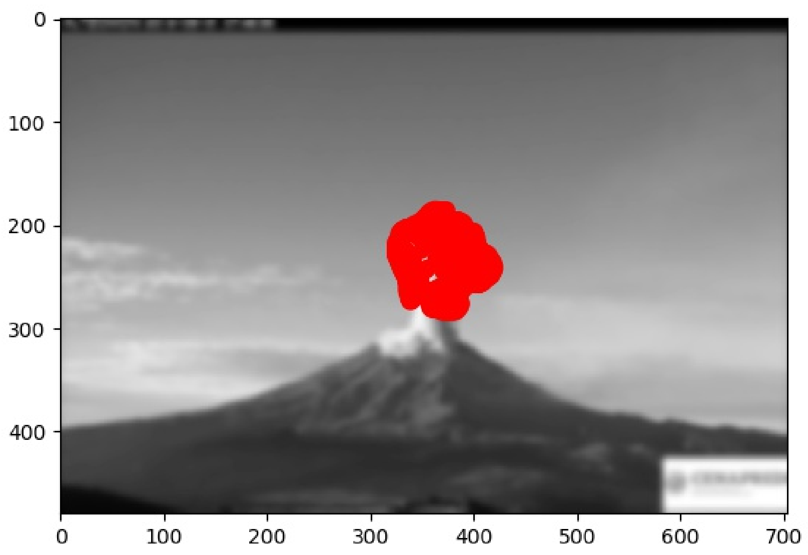


Figure 4: After Blob Detected

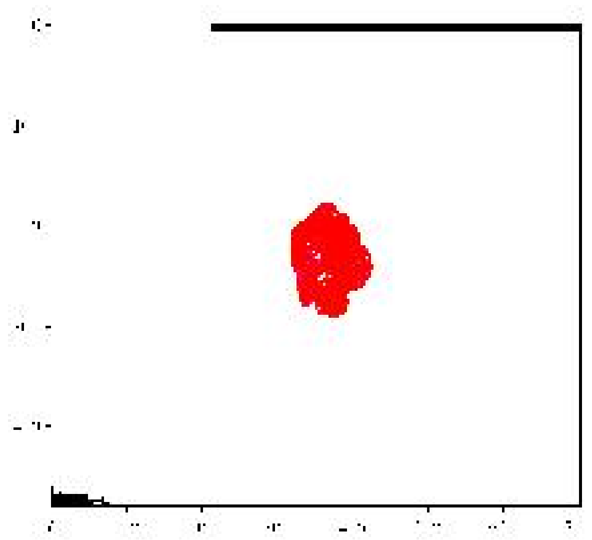


Figure 5: Binarized

Model Validation

Due to the unbalanced nature of our data, meaning that the majority of images did not contain any emissions. Traditionally trained classifiers, simply use accuracy as a validation metric which are good enough when the amount of data should be closer to the same in each class for classifiers. Imbalanced datasets can get very high accuracies when evaluated simply on the accuracy and are called "naive" results [8]. There are different ways of evaluating the mentioned "naive behavior" which will be explained as well as showing the results in our model. The best results where obtained when the images are classified not as if there is a ash emission or not, but if such as feature is distinguishable or not. Therefore with the number of both classes distinguishable and indistinguishable being significantly higher, the model did show good results.

Confusion Matrix

For classification problems such as ours a good yet simple tool that can be used is the confusion matrix. In our case, the confusion matrix gives the False Positive Rate, the ratio between the number of false positives and the number of negatives, and as most of the images do not have any eruption (negative), we cannot get useful information from the mentioned images unless we find the correct metrics. From the confusion matrix we can get the following information.

- Accuracy - The accuracy of the model is basically the total number of correct predictions divided by total number of predictions.
- Precision - The precision of a class define how trustable is the result when the model answer that a point belongs to that class.
- Recall - The recall of a class expresses how well the model is able to detect that class.
- F1 score - The F1 score of a class is given by the harmonic mean of precision and recall (2×precision×recall / (precision + recall)), it combines precision and recall of a class in one metric.

A general illustration of the information a confusion matrix will give is the following.

Original Images

Confusion Matrix - Results

Below is the confusion matrix of the best performing model printed on the terminal. Here this model accurately predicted 4991 times that an emission was undistinguishable for the original images while 120 that it was not.

Precision Recall Curve

From the values obtained in the previous section the model can be evaluated using the Precision-Recall curve. The Precision-Recall curves is useful to see the trade-off between precision -the ratio of true positives and the images predicted as positives- and the recall -or True Positive Rate, the ratio of true positives and the positives-.

Blob Detected Images

Confusion Matrix - Results

Below is the confusion matrix when the previous convolution neural network was implemented on our blob detected images. Here this model accurately predicted 4827 times that an emission was undistinguishable for the original images while 157 that it was not. With this metrix yes the model performed a bit worse but not by a considerable amout.

Precision Recall Curve - Blob Detected

Here we can tell that the model also did not performed much worse than with the original images.

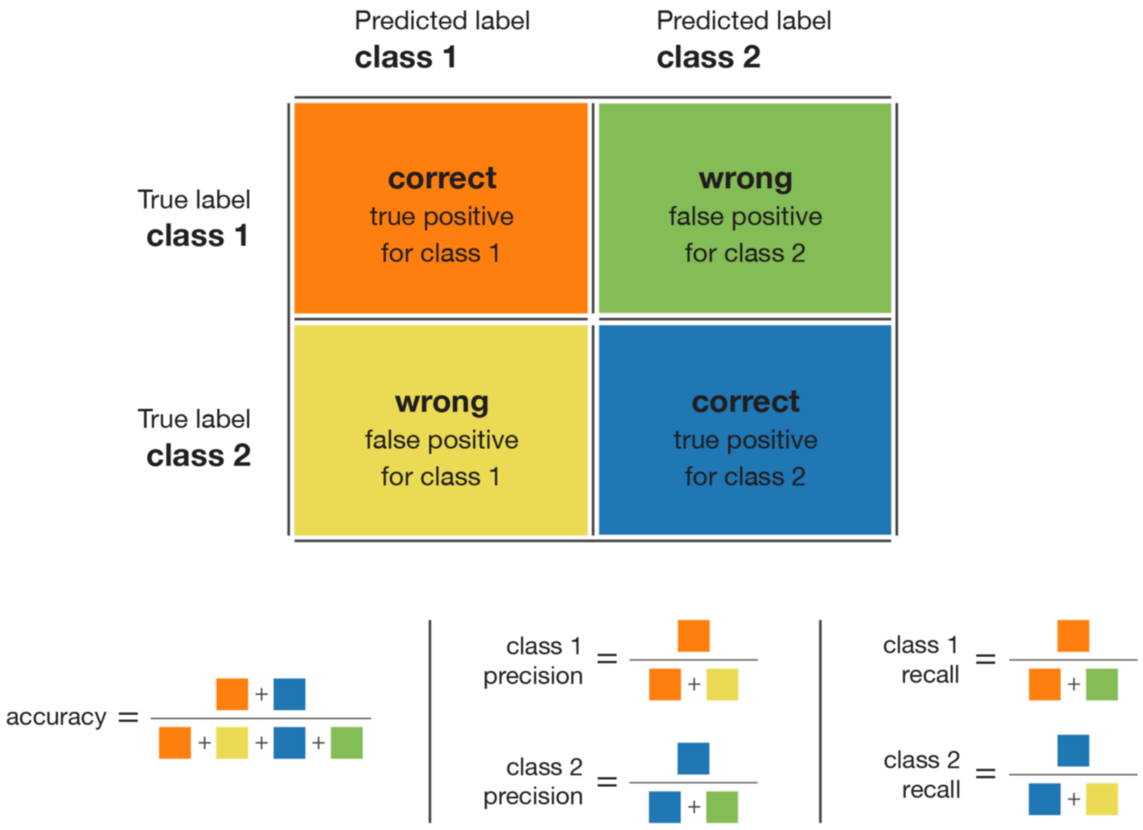
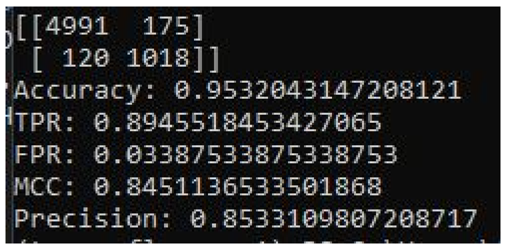


Figure 6: Confusion Matrix Results



```
[[4991  175]
 [ 120 1018]]
Accuracy: 0.9532043147208121
TPR: 0.8945518453427065
FPR: 0.03387533875338753
MCC: 0.8451136533501868
Precision: 0.8533109807208717
```

Figure 7: Results

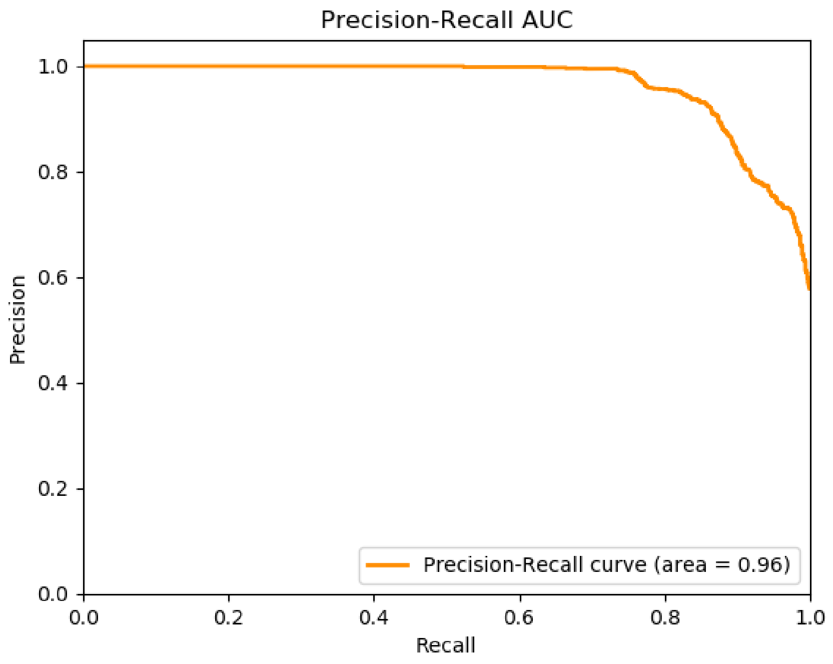
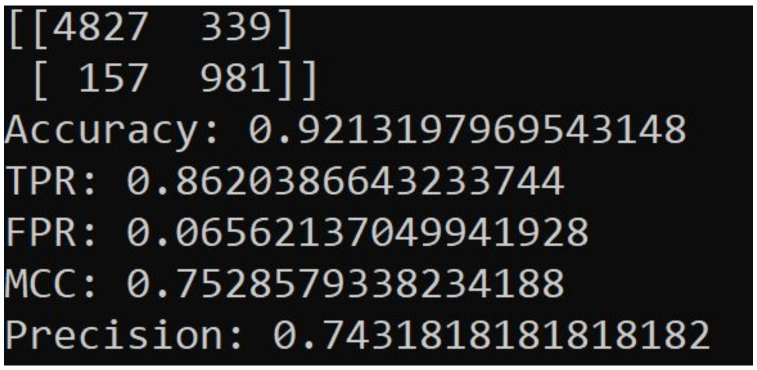


Figure 8: Precion Recall Curve



```
[[4827  339]
 [ 157  981]]
Accuracy: 0.9213197969543148
TPR: 0.8620386643233744
FPR: 0.06562137049941928
MCC: 0.7528579338234188
Precision: 0.7431818181818182
```

Figure 9: Results - Blob Detected

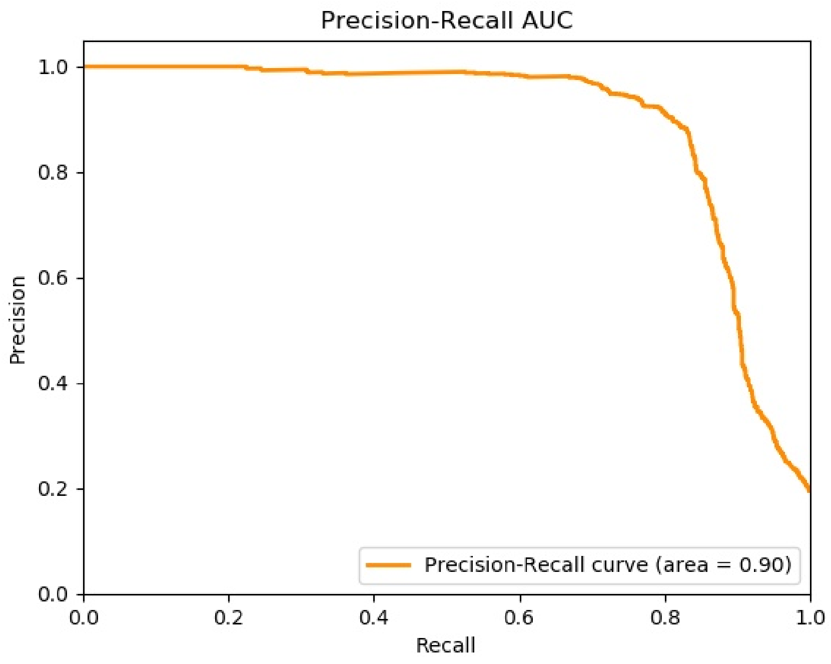
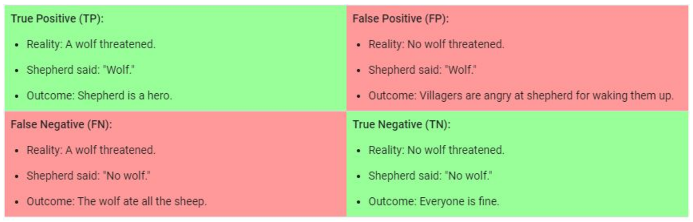


Figure 10: Precion Recall Curv - Blob Detected

Suggestions

In order to test the different hypotheses of and improve on the implementation of the model, they will have to be ran with existing checkpoint files, which contain trained models and saves a lot of time since the model will not have to be retrained. There is a function in the overall model called 'make_pred_test' that only saves the false negatives and false positives. It can be seen from the matrix below that those are mostly the images classed as not having the desired "feature". It would be interesting saving the images that do indeed theoretically have what we are looking for in order to conclude that the model has accurately made the predictions.



True Positive (TP): • Reality: A wolf threatened. • Shepherd said: "Wolf." • Outcome: Shepherd is a hero.	**False Positive (FP):** • Reality: No wolf threatened. • Shepherd said: "Wolf." • Outcome: Villagers are angry at shepherd for waking them up.
False Negative (FN): • Reality: A wolf threatened. • Shepherd said: "No wolf." • Outcome: The wolf ate all the sheep.	**True Negative (TN):** • Reality: No wolf threatened. • Shepherd said: "No wolf." • Outcome: Everyone is fine.

Figure 11: Precion Recall Curv - Blob Detected

References

[1] James Le - HeartBeat - Fritz AI. «www.medium.com.» April 12, 2018. The 5 Computer Vision Techniques That Will Change How You See The World. Accessed:. May 1st 2020.

[2] R. Haralick and L. Shapiro Computer and Robot Vision, Vol. 1, Addison-Wesley Publishing Company, 1992, pp 346 - 351.

[3] B. Rohrer «https://brohrer.github.io/convert_rgb_to_grayscale.html» Signal Processing Techniques

[4] R. Fisher, S. Perkins, A. Walker and E. Wolfart «https://homepages.inf.ed.ac.uk/rbf/HIPR2/copyrght.htm» 2003 Laplacian/Laplacian of Gaussian: Hypermedia Image Processing

[5] R. Fisher, S. Perkins, A. Walker and E. Wolfart «https://homepages.inf.ed.ac.uk/rbf/HIPR2/copyrght.htm» 2003 Laplacian/Laplacian of Gaussian: Hypermedia Image Processing

[6] D. Vernon Machine Vision, Prentice-Hall, 1991, pp 98 - 99, 214.

[7] Rocca, B. (2019, January 27). Handling imbalanced datasets in machine learning. Retrieved from https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28

[8] B. Horn Robot Vision, MIT Press, 1986, Chap. 8.

[9] D. Marr Vision, Freeman, 1982, Chap. 2, pp 54 - 78.

Miscellaneous Sources

https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/

https://academic.mu.edu/phys/matthysd/web226/Lab02.htm

https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

http://fourier.eng.hmc.edu/e161/lectures/gradient/node8.html