### SUNSET SCENE DETECTOR USING SUPPORT VECTOR MACHINE

*Yuankai Wang, Chen Yin*

Department of Computer Science, Rose-Hulman Institute of Technology   
{wangy16, yinc}@rose-hulman.edu

#### Abstract

Getting features from pictures and classifying them is an important and common way of recognizing scenes. With a color-feature based system using Support Vector Machine (SVM), we can distinguish sunset from non-sunset scenes. With a large training set and tuned hyper-parameters for training functions, we increased the performance of our machine and achieved 89.8% accuracy. We changed the threshold to getting different groups of True Positive Rate and False Positive Rate according to our needs. We also used Convolution Neural Network (CNN) to train the pictures. We used both feature extraction and transfer learning using AlexNet, and got 93.8% and 96% accuracy respectively. We also compared its performance with GoogleNet using transfer learning, which correctly recognized 96.5% of the pictures.

**1. Introduction**

Recognizing sunsets in a picture is a difficult problem. We needed to discover more details than the images presented to figure out what these imply. A sunset was somehow obvious when we looked at the image. It had its unique color, it had a certain shape that almost never changes, and in most cases, it would have few noises around it on a picture. We could easily recognize a sunset scene because we knew what it would look like and we had related experience. However, for the computer, it did not have these experiences. We needed to train it to let it know what true sunsets were and what non-sunsets were.

To let a computer “fully” understand it, we needed hundreds or thousands of pictures to cover most of the situations. However, some of the pictures could still be confusing. Pictures in Figure 1 are such examples. They both had a similar color set for the whole picture. It was extremely hard for the computer to figure it out especially when we all perceived that the second picture was a scene of sunset. The halo around the picture was what a sunset should have.



**Fig 1**: Confusing sunset images for computers

Meanwhile, detecting sunsets is also quite interesting. When we saw a sunset, we needed to comprehend which feature invoked us to conclude that is sunset. We needed to consider and attempted to figure out the information by the data we got. This turned a sensitive understanding to a rational analysis on the problem.

We worked on the project in a typical way of doing image recognition, which includes extracting features, managing data, training with inputs, and testing using test cases. We needed to find proper features for it and obtained it from every image. Since we had many features to compare, it was impossible to work on them manually. We needed to depend on machine learning techniques to find it out.

Recognition depended heavily on the color feature of each image. When there was a sunset, the color of the whole picture was quite likely to be affected by it. There would be a shiny point with other surrounding shiny pixels. We got information from color to find it out. We might apply the same technique to other recognitions to an object whose color was a dominant feature.

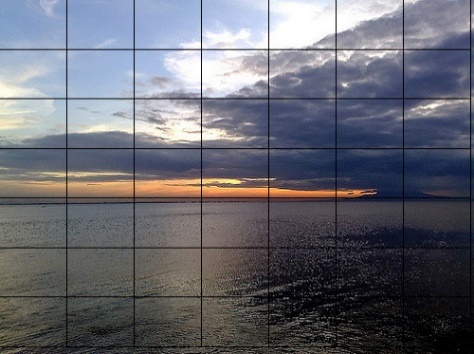
We trained the data with a large set of sunset and non-sunset pictures. We got 294 features for each picture and labeled it to indicate it was a sunset or not. Then, we trained with the features and determined another picture was a sunset with the result of our training.

**2. methodology**

**2.1 Feature Extraction**

As the color is an important feature we intended to inspect, we tried to get color information from it. Even though we usually thought the sunset color was mainly just a combination of red, orange, and yellow, the actual color could differ a lot under different circumstances. Their color features were not universal, but their luminance almost never changed and had larger differences with other scenes. Therefore, we decided to use LST color space to get color information. L means luminance, S means red versus blue, and T means green versus magenta. The unscaled conversion equation to LST color space is; ; . Although L, S, and T were in different scales, we did not have to scale them according to their proportion respected to RGB color space, because we compared these values among images, not on the same image. We just scaled them altogether.

However, getting such information was too general. An image contained much information, and the general data we got does not help much. Therefore, we decided to divide the image into several small sections and gained the data on each of them. There were two ways to accomplish this, dividing grid or dividing edges. Both had their advantages. Because we intended to get the data, which were easy to use, we chose to divide image with a matrix. Since the sizes of the picture were not necessarily divisible by 7, we began on the top-left corner and had to leave out some information on the bottom and the right side.



**Fig 2**: Sunset image with the grid overlaid on

After dividing, we had the data for each small part. However, the data were still just matrices. We wanted some more straightforward features that we could use to compare. Therefore, we calculated the first and second moment of each section on L, S, and T separately and finally got features. [1] The first moment is the mean value of the matrix. We could have a general idea of the color in this section by its mean value. The second moment is the standard deviation. We could know how the values in the matrix distributed by its standard deviation. The value showed whether the color in this part had a large or small difference.

Since the data we got were in different scales, we intended to normalize all the data with the minimum value to be zero and the maximum value to be one. We found the max value and min value in the matrix and used the equation for each data. We treated the output matrix as X, which we used in our training. As we wanted, our normalization applied to all the data together; we included every value from the images we got from feature extraction.

**2.2 Classifiers**

Support Vector Machines (SVM) is a common and useful way to classify things. It will receive a group of data; each data has a label to show which class it belongs. SVM usually handles binary classification. The label is either 1 or -1, which indicates yes or no for the thing we wanted to recognize. When it gets all the data, it will try to find the largest margin between the two kinds of data that differs them. It will allow some errors. However, with its default kernel, which is linear, it may not be able to handle some circumstances when the data have many intersections. Therefore, while classifying, we need to set the kernel to show the way of training. The most common used kernel is radial basis function (RBF) kernel, which is also the one we will use in our classification. We also changed its parameters to get an optimized way of classification.

Convolutional neural network (CNN) based on biological process is a category of neural networks, which is very effective in classification and image recognition. There are four main operations in the convolutional neural nets, which are convolution, ReLu, pooling, and classification. There are many trained networks, like AlexNet, GoogleNet, and ResNet. Since training a new network can be time-consuming, we used some of these pre-trained networks to help us recognize the sunset. The essential purpose of the convolution is to extract the feature from the input image. Convolution preserves spatial relationship between pixels by learning image feature using filters. ReLu operation, which is applied on per pixel, replaces all negative pixels value with zero. Spatial pooling reduces dimension but maintains the most important feature information. The fully connected layer is a multi-layer perceptron that can used for deep learning. The purpose of the fully connected layer is to use these features for classifying the input image into various classes based on the training dataset.

**2.3 Experimental setup**

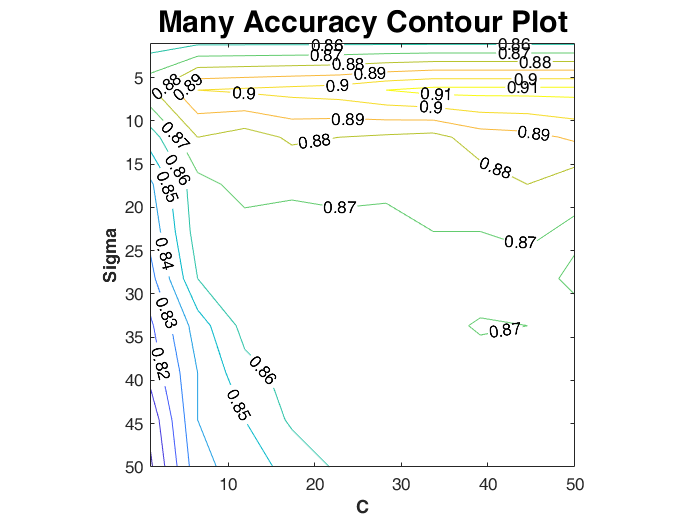
After we prepared all the features, we used them to train the computer. We first selected the training set from its corresponding place in the matrix. We also extracted their labels in the same way. We collected 800 sunset pictures and 800 non-sunset pictures. We trained the computer with the 1600 lines of data. We also collected 300 sunset pictures and 300 non-sunset pictures for validation. For SVM, there are two variables in the function sigma and c, which mean kernel scale and box constraint. We used the validation set on the trained machine to test and to record its accuracy. When we used CNN, we changed the number of classes in the fully connected layers to get better result during the learning.

**3. Results**

**3.1 SVM**

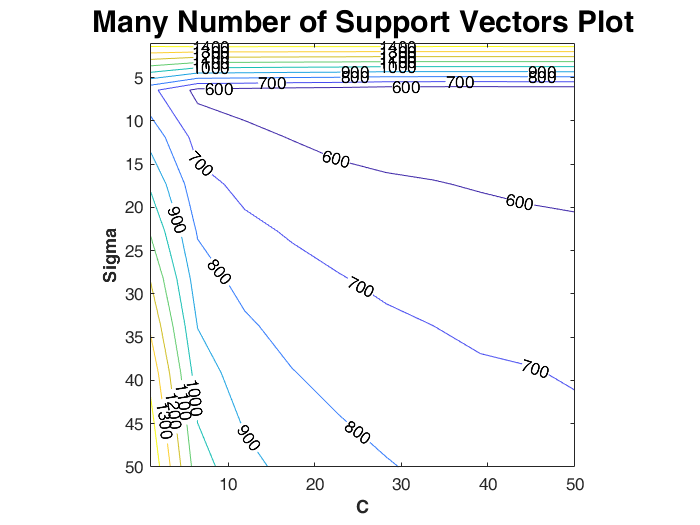
**3.1.1 Tune Hyper-parameters**

To achieve a higher accuracy, we had to tune our hyper-parameters. We looped over a broad range of values and recorded the accuracy of each group of values. Therefore, we finally got a matrix of accuracies with their corresponding sigma and c values. In this first step, we set the sigma from 1 to 50 with a step of 5, and c from 1 to 50 with a step of 5. We drew a picture of the matrix to give us a direct understanding of the performance. [2]

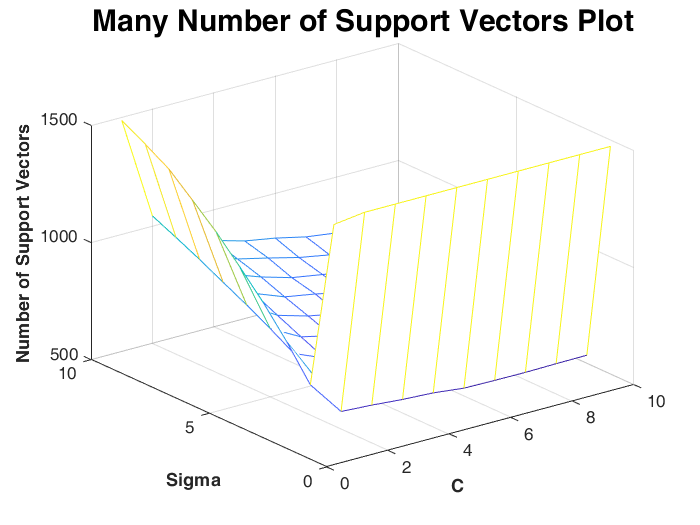


**Fig 3**: The contour plot of the accuracy for the grid search on sigma from 1 to 50 with step of 5, and c from 1 to 50 with step of 5.

Besides getting high accuracy, we also intended to have a reasonable number of support vectors to avoid overfitting. Therefore, we also formed a matrix of the support vectors in the same way.

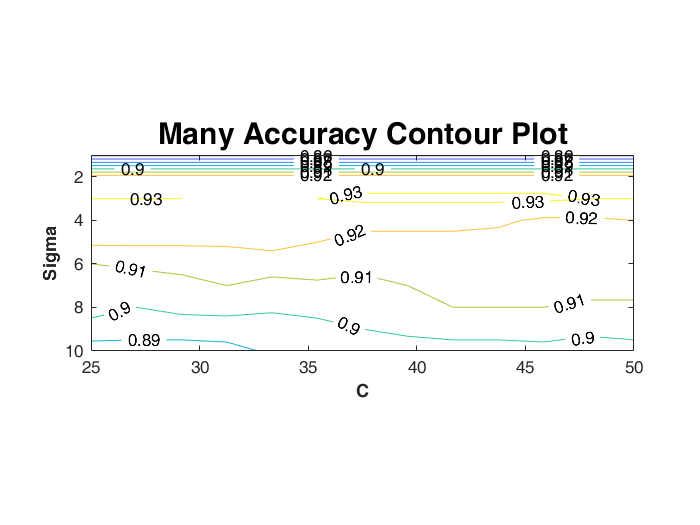


**Fig 4**: The contour plot of the support vectors for the grid search on sigma from 1 to 50 with a step of 5, and c from 1 to 50 with a step of 5.

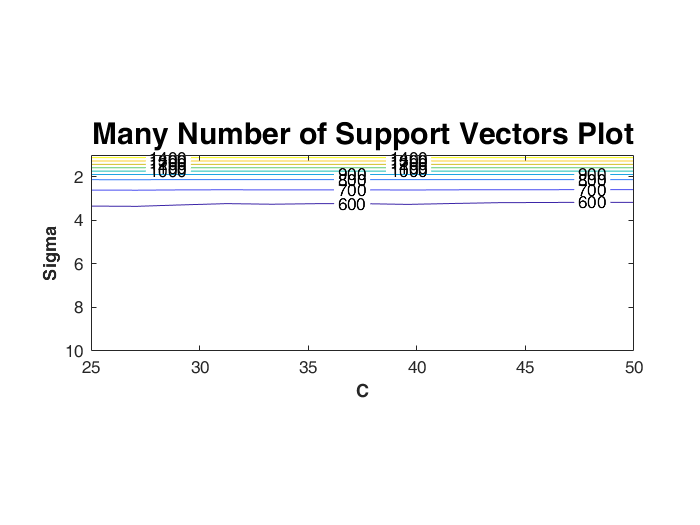


**Fig 5**: The contour plot of the support vectors for the grid search on sigma from 1 to 50 with a step of 5, and c from 1 to 50 with a step of 5.

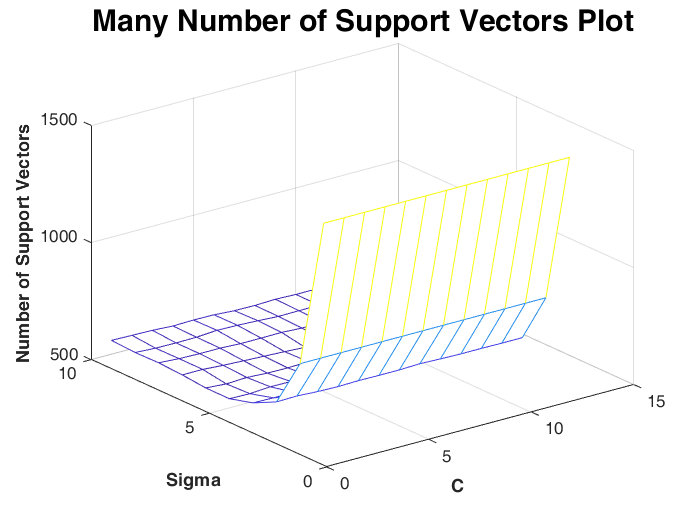
From the picture about the accuracy, we could see that when sigma was in the range 1~10, c was in the range 25~50, the accuracy was obviously higher than other value of sigma and c. From the picture about number of support vectors, the number of support vectors grew dramatically when sigma or c was small. However, the accuracy did not raise much. We believed that it would be better for us to choose the value that had less support vectors to get a higher efficiency with an acceptable reduction on accuracy. Therefore, we reduced the range and formed the new pictures.



**Fig 6**: The contour plot of the accuracy for the grid search on sigma from 1 to 10 with a step of 1 and c from 25 to 50 with a step of 2.

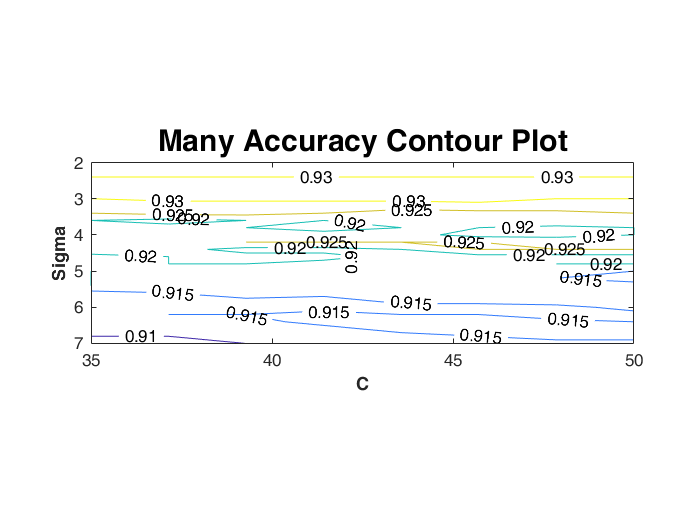


**Fig 7**: The contour plot of the support vectors for the grid search on sigma from 1 to 10 with step of 1 and c from 25 to 50 with step of 2.

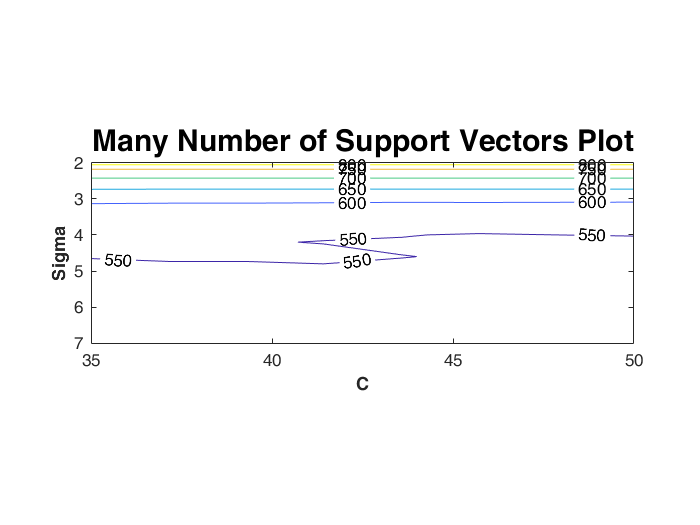


**Fig 8**: The mesh plot of the support vectors for the grid search on sigma from 1 to 10 with step of 1 and c from 25 to 50 with step of 2.

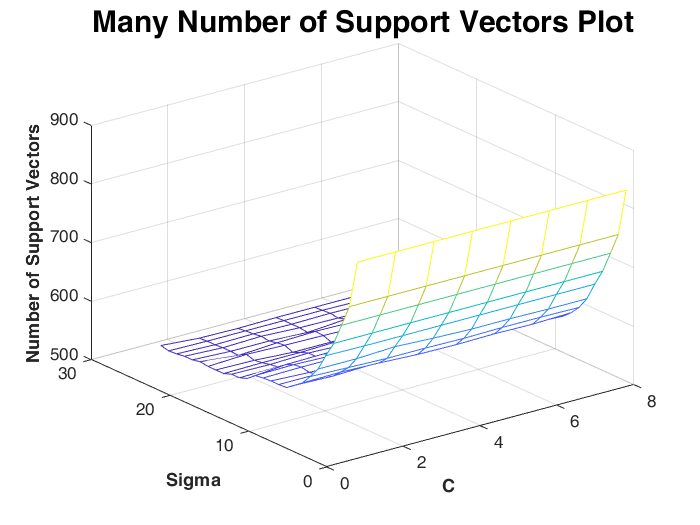
From the accuracy plot, we found that when sigma is in the range 2~4, the accuracy was higher than it was in other locations. From the number of support vectors plots, we found that when sigma was in the range 4~10, the number of support vectors was less than 600. So we decided to constrain sigma to the range 2~7 and c to the range 35~50.



**Fig 9**: The contour plot of the accuracy for the grid search on sigma from 2 to 7 with step of 0.2 and c from 35 to 50 with step of 2.

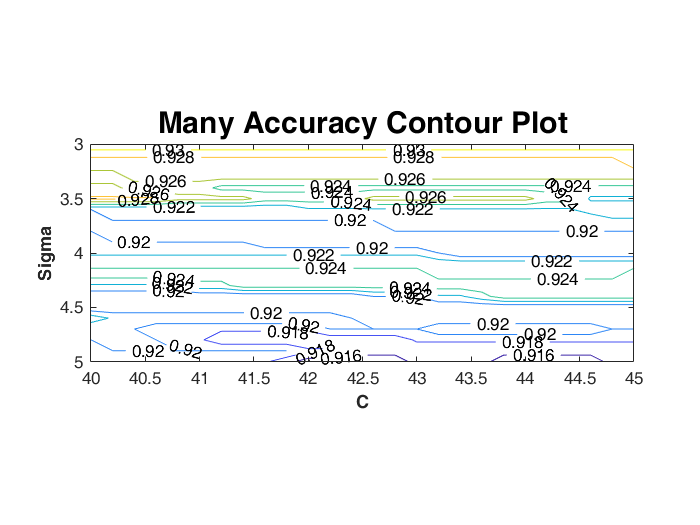


**Fig 10**: The contour plot of the support vectors for the grid search on sigma from 2 to 7 with step of 0.2 and c from 35 to 50 with step of 2.

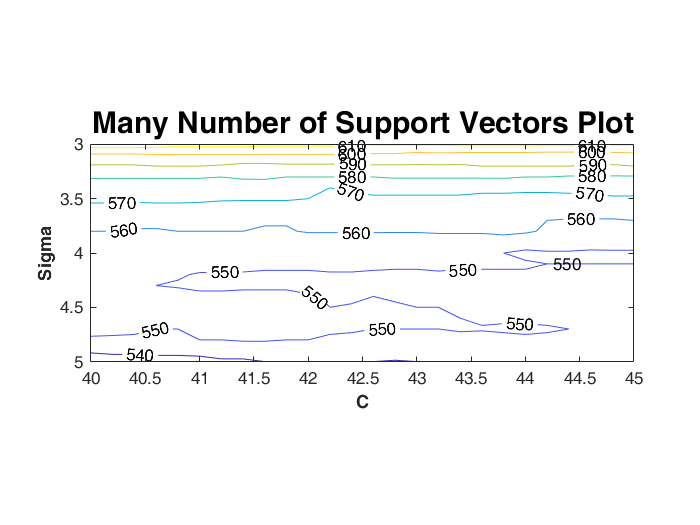


**Fig 11**: The mesh plot of the support vectors for the grid search on sigma from 2 to 7 with step of 0.2 and c from 35 to 50 with step of 2.

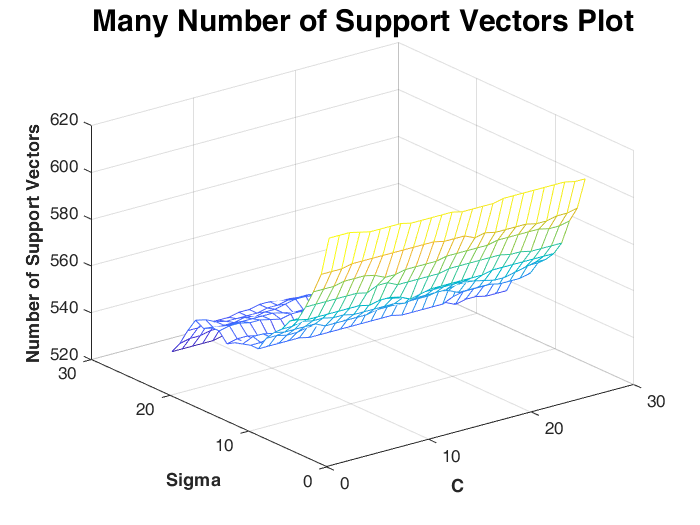
To find reasonable number of support vectors and high accuracy at the same time, we found out that when sigma was in the range 3~5, accuracy could achieve 92% or higher. When c was in the range 40~45, the number of support vectors were around 550, which is fairly reasonable.



**Fig 12**: The contour plot of the accuracy for the grid search on sigma from 3 to 5 with step of 0.1 and c from 40 to 45 with step of 0.2.



**Fig 13**: The contour plot of the support vectors for the grid search on sigma from 3 to 5 with step of 0.1 and c from 40 to 45 with step of 0.2.



**Fig 14**: The mesh plot of the support vectors for the grid search on sigma from 3 to 5 with step of 0.1 and c from 40 to 45 with step of 0.2.

With the new range, accuracy and number of support vectors was stable at 92% and 550, which was a reasonable and acceptable result for us.

Finally, we decided our value for sigma and c to be 4.2 and 40.

**3.1.2 Result with hyper-parameters**

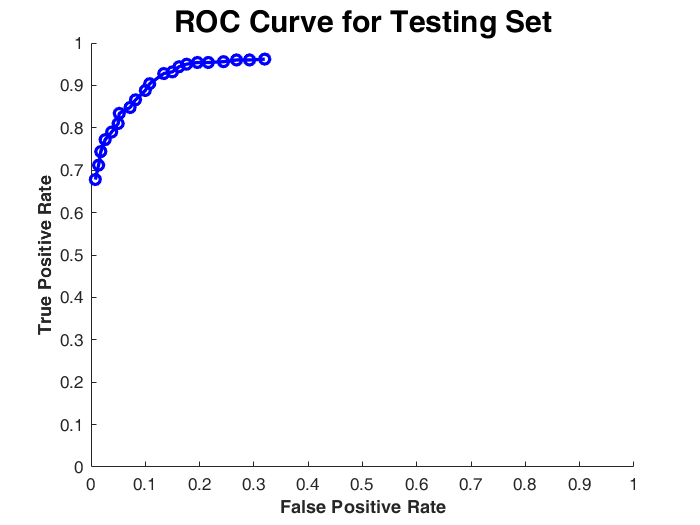
Using this two values for our hyper-parameters for the SVM training, we got 279(true positive), 276(true negative), 24(false positive), 21(false negative) in the validation set. The true positive rate was 93%, false positive rate was 8% and the accuracy was 92.5%. The number of support vectors used was 552. The accuracy was rather high and the number of support vectors was reasonable.

We had 500 sunset pictures and 500 non-sunset pictures for testing. With the same value for hyper-parameters, we got 452(true positive), 446(true negative), 54(false positive), 48(false negative) in the validation set. The true positive rate was 90.4%, false positive rate was 10.8% and the accuracy was 89.8%. The number of support vectors used was 552. The result did not have much difference with the one we got from the validation set. We felt the result was satisfying.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Image set | TP | TN | FP | FN | TPR | FPR | ACC |
| validation | 279 | 276 | 24 | 21 | 93% | 8% | 92.5% |
| Test set | 452 | 446 | 54 | 48 | 90.4% | 10.8% | 89.8% |

**Table 1**: Results from validation set and test set with 552 support vectors.

We tried to change the threshold of our classification. Its default value was 0. When we changed it, it would affect the decision of an image. Therefore, we got several groups of different truth positive rates and false positive rates. When we increased the threshold, some false positives were identified as true negatives, but also some true positives were identified as false negatives. Therefore, both true positive rate and false positive rate decreased. Similarly, if we decreased the threshold, the true positive rate and false positive rate both increased. We went over the threshold from -1 to 1 with 0.1 to be the interval, and generated the curve.



**Fig 15**: ROC curve for the testing set with threshold from -1 to 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TPR | 0.962 | 0.96 | 0.96 | 0.956 | 0.954 | 0.954 | 0.95 |
| FPR | 0.32 | 0.292 | 0.268 | 0.244 | 0.216 | 0.196 | 0.176 |
| TPR | 0.944 | 0.932 | 0.928 | 0.904 | 0.888 | 0.866 | 0.848 |
| FPR | 0.162 | 0.15 | 0.134 | 0.108 | 0.1 | 0.082 | 0.072 |
| TPR | 0.834 | 0.81 | 0.79 | 0.772 | 0.744 | 0.712 | 0.678 |
| FPR | 0.052 | 0.05 | 0.038 | 0.026 | 0.018 | 0.014 | 0.008 |

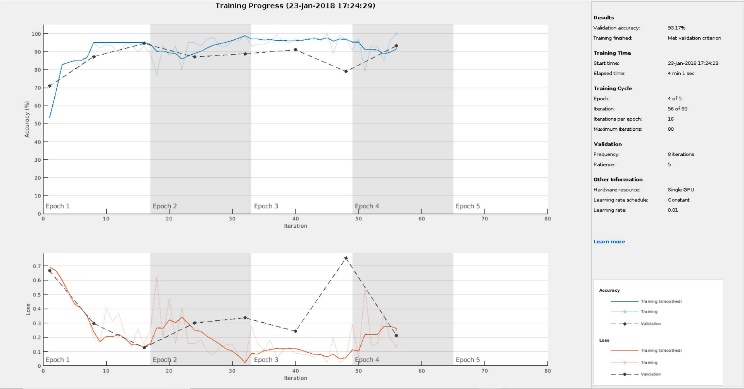
**Table 2**: Different TPR and FPR we used to plot ROC curve.

We calculated the distance of between each point and top-left corner (0,1), and found out the smallest value. When the value of threshold was 0, we had the smallest distance. Therefore, we would use 0 to classify our data.

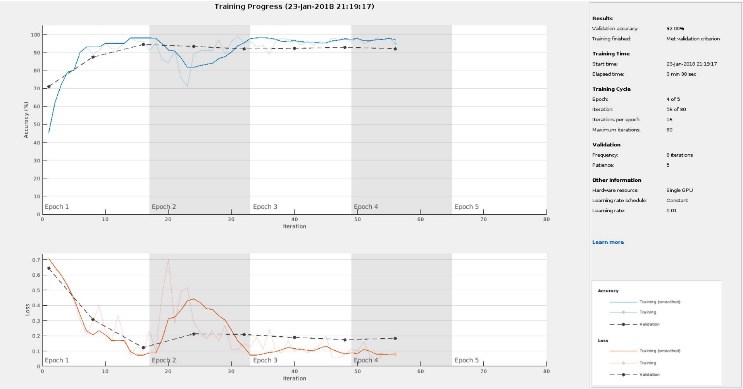
**3.2 CNN**

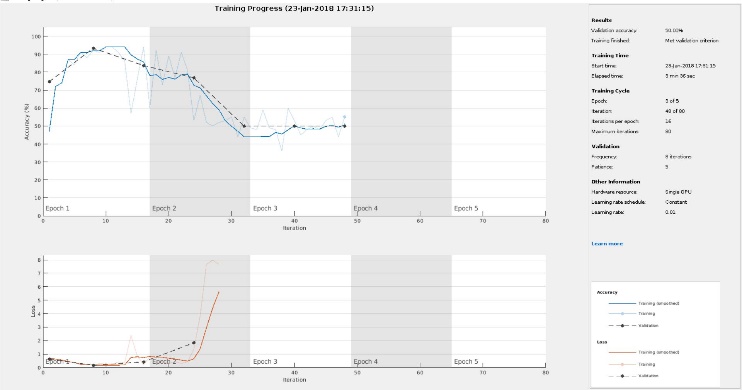
**3.2.1 Transfer learning using AlexNet**

Besides SVM, CNN would also be a great method to train with the data. We used AlexNet for our training. AlexNet contains 25 layers. The images go through each layer to enhance the network to learn the features. We first used the transfer learning method. Since our image set was small compared to the millions of images that the net used to train, we could use most of the pre-trained results. We removed the last three layers and added our own layers to train with the sunset pictures. We changed the number of classes we used in the new layer and got some different results.

Since the result would random because of the random number we got during training, we believed that minor changes would have a small effect. Therefore, we just tried some numbers with large gaps among them.

**Fig 16**: 80 fully connected layers achieved 93.17% accuracy.

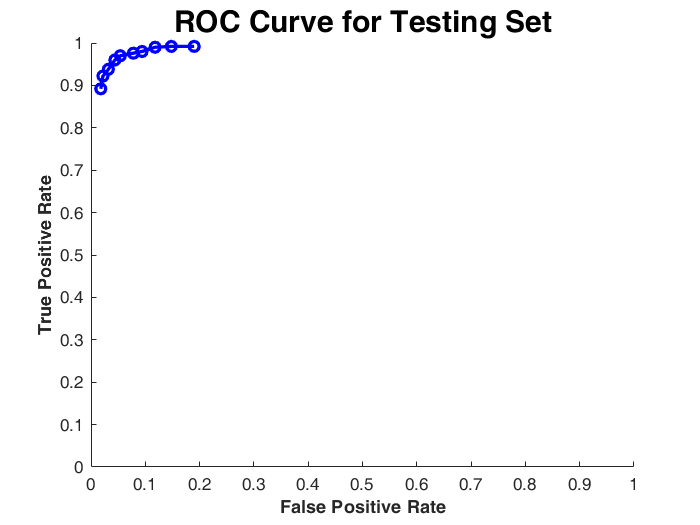
**Fig 17**: 100 fully connected layers achieved 93.5% accuracy.

**Fig 18**: 120 fully connected layers achieved 50% accuracy.

We also tried 50 and 150, but their results were similar to the 120 one, which did not make sense.

We finally found out that 100 is a nice value to use, which had high accuracy and acceptable running time. It took 218.26 seconds to train the new network and took 20.48 seconds to classify the test set. On average, it took 0.02 seconds to classify each image.

We went over the threshold from 0.2 to 0.9 with 0.1 to be the interval. Here is the ROC curve for this result.



**Fig 19**: ROC curve for the testing set with threshold from 0.2 to 0.9.

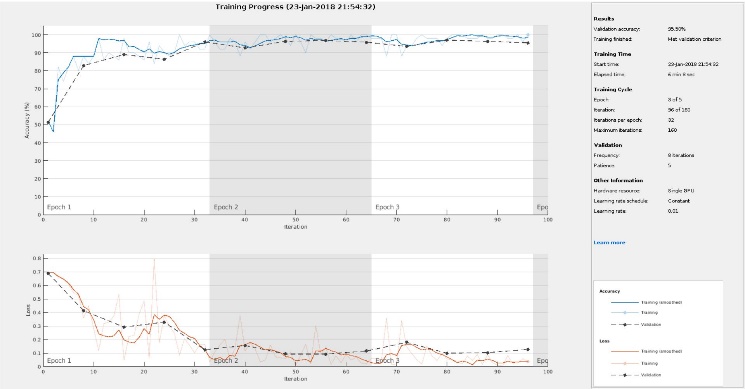
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TPR | 0.992 | 0.99 | 0.988 | 0.976 |
| FPR | 0.19 | 0.144 | 0.106 | 0.082 |
| TPR | 0.97 | 0.96 | 0.926 | 0.91 |
| FPR | 0.054 | 0.04 | 0.028 | 0.018 |

**Table 3**: Different TPR and FPR we used to plot ROC curve.

We calculated the distance of between each point and top-left corner (0,1) and found out the smallest value. When the value of threshold was 0.7, we had the smallest distance. Therefore, we would use 0.7 to classify our data. The TPR we use is 96%, and the FPR we use is 4%. The overall accuracy is 96%.

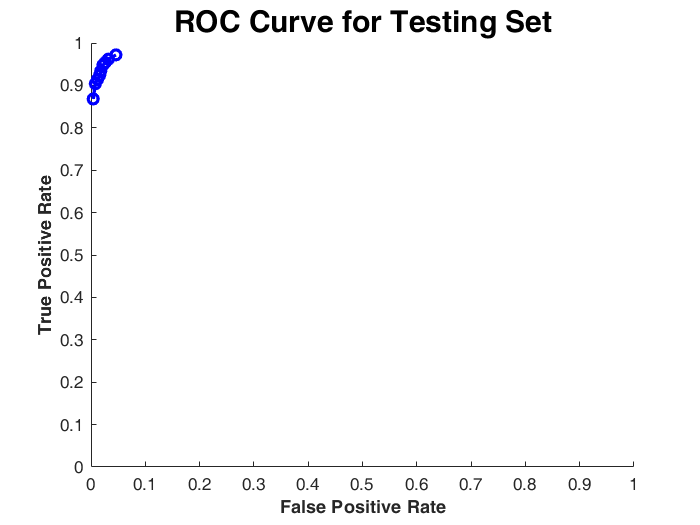
**3.2.2 Transfer learning using GoogleNet**

We also applied our transfer learning method on GoogleNet. GoogleNet has 144 layers, which is much more than AlexNet’s. Therefore, it was reasonable that we would spend more time on training with it, but we could also get a higher accuracy. We ran it with the same parameters we used for AlexNet.

**Fig 20**: 100 fully connected layers achieved 95.5% accuracy.

It took 368.75 seconds to train the new network and took 22.29 seconds to classify the test set. On average, it took 0.0223 seconds to classify each image.

However, after we set the thresholds for both networks, the difference was quite small. We went over the threshold from 0.1 to 0.9 with 0.1 to be the interval. Here is the ROC curve for this result.

**Fig 21**:ROC curve for the testing set with threshold from 0.1 to 0.9.

|  |  |  |  |
| --- | --- | --- | --- |
| TPR | 0.972 | 0.962 | 0.954 |
| FPR | 0.046 | 0.032 | 0.026 |
| TPR | 0.948 | 0.934 | 0.924 |
| FPR | 0.022 | 0.018 | 0.016 |
| TPR | 0.914 | 0.904 | 0.868 |
| FPR | 0.012 | 0.008 | 0.004 |

**Table 4:** Different TPR and FPR we used to plot ROC curve.

We calculated the distance of between each point and top-left corner (0,1) and found out the smallest value. When the value of the threshold is 0.2, we had the smallest distance. Therefore, we would use 0.2 to classify our data. The TPR we use is 96.2%, and the FPR we use is 3.2%. The overall accuracy is 96.5%.

With more layers than AlexNet, GoogleNet has a longer running time, but also a higher accuracy. But both differences were not obvious. Since we can achieve a high accuracy just with AlexNet, we can just use it as the training network. However, if we want to train more complicated thing, GoogleNet might be more powerful on that.

**3.2.3 Feature Extraction Using AlexNet**

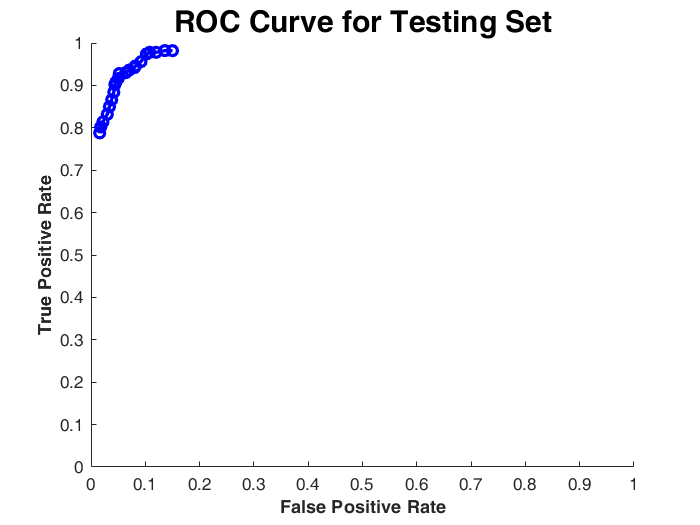
The other method we used is feature extraction. We obtained extracted features from the pre-trained network. We grasped it from a fully connected layer in the net to train our image set. We tried to obtain information from three different layers, “fc6”, “fc7”, and “fc8”. Without tuning the hyper-parameters, we got result their accuracy and runtime shown in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Layer Name | fc6 | fc7 | fc8 |
| Accuracy | 92.5% | 93.8% | 93.8% |
| Runtime | 49s | 47s | 63s |

**Table 5**: Accuracy and run time of “fc6”, “fc7”, and “fc8” before tuning hyper-parameters.

Therefore, we chose to work on “fc7”, which had the highest accuracy and lowest runtime. The default hyper-parameter had a high performance with 95.3% accuracy and only used 296 support vectors. We tried to tune the parameters manually, but we got lower accuracy and higher support vectors number. Therefore, we just kept its default tuning result.

We went over the threshold from -1 to 1 with 0.1 to be the interval. Here is the ROC curve for this result.

**Fig 22**: ROC curve for the testing set with threshold from -1 to 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TPR | 0.982 | 0.982 | 0.978 | 0.978 | 0.974 | 0.956 | 0.946 |
| FPR | 0.15 | 0.136 | 0.12 | 0.108 | 0.102 | 0.092 | 0.082 |
| TPR | 0.942 | 0.936 | 0.93 | 0.928 | 0.916 | 0.908 | 0.902 |
| FPR | 0.08 | 0.07 | 0.064 | 0.052 | 0.05 | 0.046 | 0.044 |
| TPR | 0.884 | 0.866 | 0.85 | 0.832 | 0.814 | 0.802 | 0.788 |
| FPR | 0.042 | 0.038 | 0.034 | 0.03 | 0.022 | 0.018 | 0.016 |

**Table 6**: Different TPR and FPR we used to plot ROC curve.

We calculated the distance of between each point and top-left corner (0,1) and found out the smallest value. When the value of threshold was 0, we had the smallest distance. Therefore, we would use 0 to classify our data.

**3.3 Example Pictures Analysis**

**3.3.1 Using SVM**

After getting the result, we want to step back to the original image to see what kind of image it can correctly classify.

Here are the images that are true positive.



**Fig 23**: Sunset images correctly labeled as sunsets.

Obviously, these two pictures are two types of typical sunset scenes. The first has a center point, which is the Sun. The colors around it change according to their distance to the center point. The overall hue is about red, orange, and yellow. There is also a horizontal line at the bottom, which color is much darker.

The second image presents a cloudy scene. We can hardly find a center point that shows the Sun. However, the overall color keeps the style of sunset scene. It can still be recognized as a sunset.

Here are the images that are true negative.



**Fig 24**: Non-sunset images correctly labeled as non-sunsets.

The first picture is blue-sky scene. There is no sun in this image. At the same time, the main color of this picture is not red, yellow, or orange. The cloud makes a diagonal in the image and the sunset will not distribute like this in the sky. It is easy for the computer to label this picture as non-sunset.

The second picture is a purple flower and green leaves with a pink background. The spatial distribution of the flower is vertical which will not confuse the computer. The color also has a large difference with what a sunset scene should have.

Here are the images that are false negative.



**Fig 25**: Sunset images incorrectly labeled as non-sunsets.

The first image is a picture of sunlight reflected from water. From the overall color of the picture, we can know that it is the light of a sunset scene. However, it is concerning that the information about sunset is vague here. It does not have a center point of light. The color is not distinguishable enough for a sunset scene. The animals in the middle may affect the data of the color because they are almost black. Usually, we will have black at the bottom to be considered as the horizontal line. However, in this picture, it has black at the top of the image, which may be a reason of misclassifying.

The second image displays the sun that is surrounded by noise. Most of the pictures do not have noise around the sun. This one is much more difficult to be recognized as sunset scene because the overall color is blue. Although it has a shining center point, the pixels around it are mainly noise with black color. It does not have the information of the changing colors around it.

Here are the images that are false positive.



**Fig 26**: Non-sunset image incorrectly labeled as sunset.

The first picture is a tree with yellow leaves all over the place. The color is quite similar to the sunset scene. The distribution of the color is broad. The computer can hardly tell it is the light of the Sun or the original color of the object. This picture even has some smoothly changing colors on the side.

The second picture is an orange flower. It also has a similar color and a wide spread of that color. We believe the reason is similar to the first picture. Since we depend heavily on colors, similar colors is the main reason that causes misclassification.

**3.3.1 Using Transfer Learning based on AlexNet**

Here are the images that are true positive.



**Fig 27**: Sunset images correctly labeled as sunsets.

The first picture is a typical sunset scene. There is a shiny circle above the horizon in the center of the image. The sun is so shiny that our neural net confidently classifies this image as sunset picture. Even though the sun has much noise, which is trees, around it, they did not influence the color features much.

The second picture is a city view with sunset. The main color of this picture is orange, pink and purple. Although there is not a sun in the picture, the main colors determine the class. The overall spatial relation might also help it to know it is a sunset scene.

Here are the images that are true negative.



**Fig 28**: Non-sunset images correctly labeled as non-sunsets.

The first picture is a scene of mountains and sea. This picture consists of white cloud, black mountains and a man wearing a red jacket. Obviously, the main color of this picture is hugely different from typical sunset scene. Although this picture has spatial relations of typical sunset scenes, we use color as our feature to classify the image. So it can correctly recognize it.

The second picture shows a rose. The main color of this picture is dark red, light red. This picture does not have a spatial connection with typical sunset picture because of the petals. Although the color is similar to a sunset scene, the spatial information of the color promotes net to make the correct decision.

Here are the images that are false negative.



**Fig 29**: Sunset images incorrectly labeled as non-sunsets.

The first picture is a sunset scene in the picture. However, there is too much noise around the sun. Although the sun is shiny as usual, the influence of the shiny light is largely influenced by the noise around it. The color information is not obvious enough to show the existence of the sun.

The second picture shows sunset clouds. There is not a sun in the picture, so it does not have a shiny point to show the most obvious color feature. Furthermore, the main color is in the typical sunset color range which is yellow, orange, and red. However, the saturation, lightness does not match. The scene is darker than a typical sunset.

Here are the images that are false positive.



**Fig 30**: Non-sunset image incorrectly labeled as sunset.

The first picture is a close picture of some shiny yellow leaves with a brown background. The color is so close to the sunset scene. The bottom part is almost black, which might be considered as the horizon. Therefore, the picture can be quite confusing for the computer to recognize it.

The second picture is a scene of a forest with a river in it. We are quite confused why the network considers it to be a sunset scene. We did not find any similarities between the picture and a sunset scene.

**4. conclusions**

If we were granted another two weeks, we would like to try to use some different kernels. They may be the polynomial kernel, which is also in the Matlab or some other useful kernels from the Internet. Use several different validation sets of images to get more general and accurate hyper-parameters. We also want to process the image first in some different ways to improve the result or just see what difference the method will bring. We would also try to tune the hyper-parameters manually to see their influence on the result. We also want to try some different neural networks to find out their performance in training our sunset image set. This work can be quite time-consuming and need strong hardware support.

If we have another year, we would like to include some other useful features, like the edge of the sun. We may also try some different neural networks. We can also add some other things to classify together. For example, we can train the computer to recognize a scene is a sunset or a sunrise or neither of them. We would like to develop our own neural network and use it to train our data. The network would be specialized for sunset detection. However, we would need much more training pictures to make it has high accuracy.

With our sunset detector using SVM, we can recognize whether an image is a scene of sunset or not. We increased its accuracy and made a trade-off with efficiency. Using CNN to help to recognize sunsets, we got a higher accuracy than merely using SVM in both methods with a small support vector number but ran for a longer time. Both ways have advantages and work well on recognizing sunset scenes. The concept can also be applied to recognize other objects that process obvious features and a large number of training sets.

**5. References**

[1] Boutell, M., Jiebo Luo, and Robert T. Gray. "Sunset scene classification using simulated image recomposition." *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*. Vol. 1. IEEE, 2003.

[2] Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. "A practical guide to support vector classification." (2003): 1-16.