Shadow Detection Using Convolutional Neural Network

Mr. Nasaruddin N.Shaikh[1], Dr. Mrs.A.A.Agashe[2]
[1, 2] Department of Electronics Engineering
[1, 2] Walchand College of Engg.Sangli, Maharashtra, India

Abstract*- We present a practical framework to automatically detect shadows in real-world scenes from a single photograph Previous works on shadow detection requires human intervention to detect shadow.But our framework automatically learns the most relevant features in a supervised manner using multiple convolutional deep neural network (ConvNets).Framework explain procedure of labeling image data, balancing classes, defining layers in CNN, setting training options ,training on GPU and testing test image on trained network .The proposed framework learns features at the pixel level and gives probability to each pixel to be shadow or nonshadow.*

Keywords*-* Convolutional neural networks (ConvNets),Stochastic Gradient Descent with Momentum (SGDM)

I. INTRODUCTION

Shadow is natural phenomenon whose detection and removal is important in many computer vision tasks. Shadows provide useful clues for the scene characteristics which can help in visual scene understanding. Recently, shadows have been used for tasks related to object shape, size, movement, number of light sources and illumination conditions. Shadows have a particular practical importance in augmented reality applications, where the illumination conditions in a scene can be used to seamlessly render virtual objects and their casted shadows. In digital photography, information about shadows and their removal can help to improve the visual quality of photographs. Beside the above mentioned assistive roles, shadows can also cause complications in many fundamental computer vision tasks. They can degrade the performance of object recognition, stereo, shape reconstruction, image segmentation and scene analysis. Shadows are also a serious concern for aerial imaging and object tracking in video sequences. Despite the ambiguities generated by shadows, the Human Visual System (HVS) does not face any real difficulty in filtering out the degradations caused by shadows. We need to equip machines with these same visual comprehension abilities. Inspired by the hierarchical architecture of the human visual cortex, many deep representation learning architectures have been proposed in the last decade. We draw our motivation from the recent successes of these deep learning methods in many computer vision tasks where learned features out-performed hand-crafted features.On that basis, we propose to use multiple convolutional neural networks (ConvNets) tolearn useful feature representations for the task of shadow detection.

II. SHADOW DETECTION FRAMEWORK

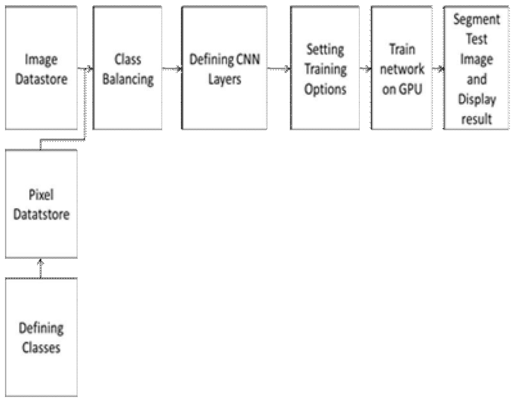


Figure 1. Block diagram of shadow detection

A. Defining Classes

A class is a category in which we need to classify pixels in an image.In our framework, our interest is to detect shadow so we have considered two classes: shadow class and nonshadowclass.We aim to classify each pixel in test image in either shadow pixel or nonshadow pixel.

B. Labeling Images

For training images need a number of images.For indicating a region of interest there is a need to label data according to class present in an image.In our case region of interest is a shadow. In our framework, we have labeled 700 images with shadow and nonshadow class.

There are two types of labeling :

1. Rectangular Labeling
2. Pixel Labeling

If an object is of fixed size then rectangular labeling is beneficial. If a region of interest is non-regular size object then pixel label labeling is beneficial. Shadow does not have fixed size so we have labeled 700 images with pixel labeling.

We have labeled images using image labeler app in Matlab.

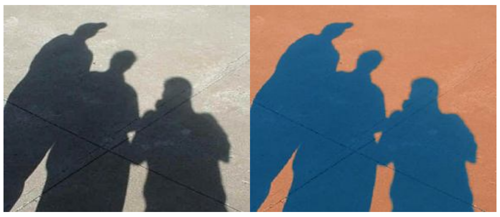


Figure 2. Image and corresponding class labeling

C. Pixel Label Datastore

It contains pixel labeled data of 700 labeled images. All pixel labeled data is stored in a PixelLabelDatastore object so that whenever we need we can read pixel label data for reading pixel-wise classification of image data.

D. Image Datastore:

It contains original 700 images.All images are stored in Image data store object to manage a collection of a number of image files.

Figure 3. Some Images from Image datastore

E. Prepare Training and Test Sets

We have labeled all 700 images with respective classes.At the end of testing test image we need to calculate the accuracy of shadow detection .so we need to calculate the intersection of a union of shadow detected by CNN and original manually labeled image of the same image.so we have to split whole data into Training and Test Sets.We have trained Convolutional Neural Network using 95% of the images from the dataset. The rest 5% of the images are used for testing.

F. Analyzing Dataset Statistics

To see the distribution of class labels in the dataset, we have use countEachLabel. This function counts the number of pixels by a class label. Visualizing the pixel counts by class, we have observed that nonshadow class pixels are more than shadow class pixels. So here class imbalance problem arises.

G. Balance Classes Using Class Weighting

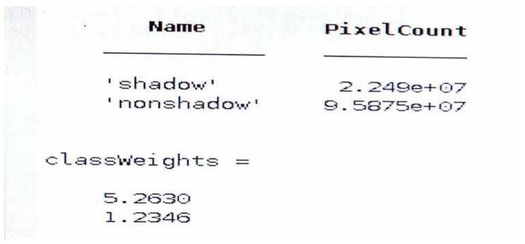
The class imbalance biases learning process in favor of dominant class to fix this class weighting is used.We have used Inverse of frequency weighting where class weights are inverse of class frequency is used.

Procedure for class Weighting:

Total Number Of Pixels = sum(Pixel Count of all classes)

A frequency of class = Pixel Count of class / (Total Number Of Pixels)

Class Weights of class = 1 / (frequency of class)
Class weights of each class are calculated by this procedure.



```
       Name            PixelCount
    ___________        __________

    'shadow'           2.249e+07
    'nonshadow'        9.5875e+07

classWeights =

    5.2630
    1.2346
```

Figure 4. The pixel count of each class and corresponding class weights

H. Creating CNN:

A number of filters: It is specified as a positive integer. This number corresponds to the number of neurons in the layer that connect to the same region in the input. This parameter determines the number of channels (feature maps) in the output of the convolutional layer. We have used 64 number of filters.

Filter Size: It specifies Height and width of filters.Height and width of the filters, specified as a vector of two positive integers [h w], where h is the height and w is the width. FilterSize defines the size of the local regions to which the neurons connect in the input.We have used Filter Size of 3
Number of Classes: It specifies Number of classes to classify inputs. In case number of classes are 2

Normalization: Normalization is the Data Transformation. Data transformation to apply every time data is forward propagated through the input layer, specified as one of the following.

‘Zero-center’-The layer subtracts the mean image of the training set. ‘none’ -No transformation. We have used Zero-center Normalization

Stride: It specifies Step size for traversing input. The step size for traversing the input vertically and horizontally, specified as a vector of two positive integers [a b] where a is the vertical step size and b is the horizontal step size. When creating the layer, you can specify stride as a scalar to use the same value for both dimensions.

Padding: It specifies size of padding to apply to input borders, specified as a vector of four non-negative integers [t b l r], where t is the padding applied to the top, b is the padding applied to the bottom, l is the padding applied to the left, and r is the padding applied to the right.

Cropping: It specifies Output layer size reduction. Output layer size reduction, specified as 'Cropping' and a scalar or scalar vector. We can trim the edges of the full transposed convolution by the same amount or specify vertical and horizontal amounts. If we specify a vector, [vertical, horizontal], the vertical valuetrims the top and bottom, and the horizontal value trims the sides.

I. Defining Layers in CNN:

We have used following layers in our Convolutional Neural Network.

Image input layer
Convolution layer
RELU layer
Max pooling
Convolution layer
RELU layer
Transposed convolution
Convolution layer
Softmax layer
Pixel classification layer

Image input layer: An image input layer inputs images to a network and applies data normalization. Input size is Size of the input data, specified as a row vector of three integer values [h w c], where h is the height, w is the width, and c is the number of channels. Set c to 1 for grayscale images, 3 for

RGB images. In our case, input size is 360*480*3 images with zero-center normalization.

Convolution layer: A 2-D convolutional layer applies sliding filters to the input. The layer convolves the input by moving the filters along the input vertically and horizontally and computing the dot product of the weights and the input, and then adding a bias term. We have used 64 3x3 convolutions with stride [1 1] and padding [1 1 1 1]

RELU layer: It is Rectified Linear Unit (ReLU) layer. A ReLU layer performs a threshold operation to each element of the input, where any value less than zero is set to zero.This operation is equivalent to $f(x)=0$, for $x<0$ and $f(x)=x$, for $x\geq 0$

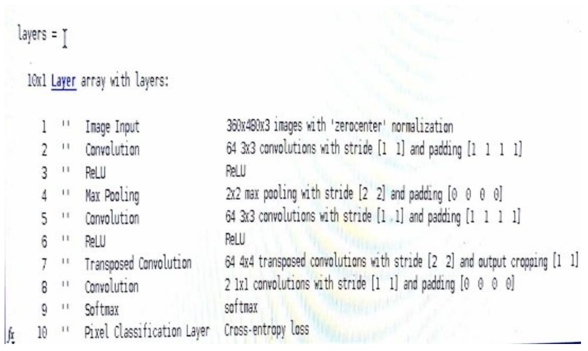
Max pooling layer: A max pooling layer performs down-sampling by dividing the input into rectangular pooling regions, and computing the maximum of each region.We haveused 2x2 max-pooling with stride [2 2] and padding [0 0 0 0]

Transposed convolution: It creates a transposed 2-D convolution layer. A transposed 2-D convolution layer upsamples feature maps. This layer is the transpose of convolution and does not perform deconvolution. We have used 64 4x4 transposed convolutions with stride [2 2] and output cropping [1 1]

Convolution layer: A 2-D convolutional layer applies sliding filters to the input. The layer convolves the input by moving the filters along the input vertically and horizontally and computing the dot product of the weights and the input, and then adding a bias term.We have used 2 1x1 convolutions with stride [1 1] and padding [0 0 0 0]

Softmax layer: A softmax layer applies a softmax function to the input. The softmax function is often used in the final layer of a neural network-based classifier.Softmax classifiers give probabilities for each class label.

Pixel classification layer: It creates pixel classification layer for semantic segmentation.pixel classification layer creates a pixel classification output layer for semantic image segmentation networks. The layer outputs the categorical label for each image pixel processed by a CNN. The layer automatically ignores undefined pixel labels during training. The loss function is cross entropy loss function.



```
layers =

  10x1 Layer array with layers:

     1   ''   Image Input                  360x480x3 images with 'zerocenter' normalization
     2   ''   Convolution                  64 3x3 convolutions with stride [1  1] and padding [1  1  1  1]
     3   ''   ReLU                         ReLU
     4   ''   Max Pooling                  2x2 max pooling with stride [2  2] and padding [0  0  0  0]
     5   ''   Convolution                  64 3x3 convolutions with stride [1  1] and padding [1  1  1  1]
     6   ''   ReLU                         ReLU
     7   ''   Transposed Convolution       64 4x4 transposed convolutions with stride [2  2] and output cropping [1  1]
     8   ''   Convolution                  2 1x1 convolutions with stride [1  1] and padding [0  0  0  0]
     9   ''   Softmax                      softmax
    10   ''   Pixel Classification Layer   Cross-entropy loss
```

Figure 5. Layers in our Convolutional Neural Network

J. Setting Training Options:

It specifies options for training deep learning neural network.

Training options used are:

Solver Name	sgdm
Initial Learn Rate	1e-3,
Max Epochs	100,
Mini Batch Size	1

Initial Weights and Biases: Initial weights is a Gaussian distribution with a mean of 0 and a standard deviation of 0.01. The default for the initial bias value is 0. We can manually change the initialization for the weights and biases.

Solver for training network: Stochastic Gradient Descent with Momentum (sgdm)

Stochastic Gradient Descent

The gradient descent algorithm updates the network parameters (weights and biases) to minimize the loss function by taking small steps in the direction of the negative gradient of the loss,

$$\theta_{\ell+1} = \theta_{\ell} - \alpha \nabla E(\theta_{\ell})$$

where ℓ stands for the iteration number, $\alpha>0$ is the learning rate, θ is the parameter vector, and $E(\theta)$ is the loss function. The gradient of the loss function, $\nabla E(\theta)$, is evaluated using the entire training set, and the standard gradient descent algorithm uses the entire data set at once. The stochastic gradient descent algorithm evaluates the gradient and updates the parameters using a subset of the training set. This subset is called a mini-batch. Each evaluation of the gradient using the mini-batch is an iteration. At each iteration, the algorithm takes one step towards minimizing the loss function. The full pass of the training algorithm over the entire training set using mini-batches is an epoch. We can specify the mini-batch size

and the maximum number of epochs using the'MiniBatchSize' and 'MaxEpochs' name-value pair arguments, respectively.
Stochastic Gradient Descent with Momentum

The stochastic gradient descent algorithm might oscillate along the path of steepest descent towards the optimum. Adding a momentum term to the parameter update is one way to reduce this oscillation. The stochastic gradient descent with momentum update is

$$\theta_{\ell+1}=\theta_{\ell}-\alpha\nabla E(\theta_{\ell})+\gamma(\theta_{\ell}-\theta_{\ell-1}),$$

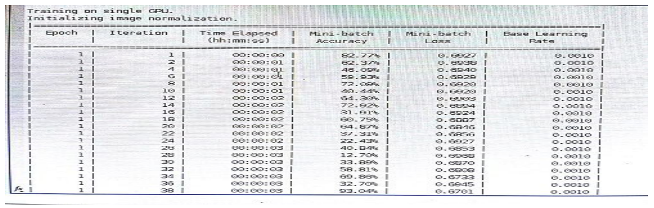
where γ determines the contribution of the previous gradient step to the current iteration. We can specify this value using the 'Momentum' name-value pair argument. To use stochastic gradient descent with momentum to train a neural network, specify solverName as 'sgdm'. To specify the initial value of the learning rate α, use the'InitialLearnRate' name-value pair argument. We can also specify different learning rates for different layers and parameters.

Initial Learn Rate: Initial learning rate used for training, The default value is 0.01 for the 'sgdm' solver and 0.001 for the'rmsprop' and 'adam' solvers. If the learning rate is too low, then training takes a long time. If the learning rate is too high, then training might reach a suboptimal result or diverge.
Max Epochs: It specifies a Maximum number of epochs to use for training. It is an iteration is one step taken in the gradient descent algorithm towards minimizing the loss function using a mini-batch. An epoch is the full pass of the training algorithm over the entire training set

Mini Batch Size: It is the size of the mini batch. Size of the mini-batch to use for each training iteration. A mini-batch is a subset of the training set that is used to evaluate the gradient of the loss function and update the weights.

K. Training Network on GPU:

We have train network using training data(image and pixel datastore), layers and training options. For training 700 images it takes around 1 hour. We have this network on CUDA-capable NVIDIA™ GPU Titan x.



```
Training on single GPU.
Initializing image normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |       82.77% |       0.6927 |          0.0010 |
|       1 |           2 |       00:00:01 |       62.37% |       0.6938 |          0.0010 |
|       1 |           4 |       00:00:01 |       46.09% |       0.6940 |          0.0010 |
|       1 |           6 |       00:00:01 |       59.93% |       0.6929 |          0.0010 |
|       1 |           8 |       00:00:01 |       72.09% |       0.6920 |          0.0010 |
|       1 |          10 |       00:00:01 |       40.44% |       0.6920 |          0.0010 |
|       1 |          12 |       00:00:02 |       64.30% |       0.6903 |          0.0010 |
|       1 |          14 |       00:00:02 |       72.92% |       0.6894 |          0.0010 |
|       1 |          16 |       00:00:02 |       31.91% |       0.6924 |          0.0010 |
|       1 |          18 |       00:00:02 |       60.75% |       0.6887 |          0.0010 |
|       1 |          20 |       00:00:02 |       64.67% |       0.6846 |          0.0010 |
|       1 |          22 |       00:00:02 |       37.31% |       0.6856 |          0.0010 |
|       1 |          24 |       00:00:02 |       22.43% |       0.6927 |          0.0010 |
|       1 |          26 |       00:00:03 |       40.84% |       0.6853 |          0.0010 |
|       1 |          28 |       00:00:03 |       12.70% |       0.6968 |          0.0010 |
|       1 |          30 |       00:00:03 |       33.89% |       0.6870 |          0.0010 |
|       1 |          32 |       00:00:03 |       58.81% |       0.6806 |          0.0010 |
|       1 |          34 |       00:00:03 |       69.86% |       0.6733 |          0.0010 |
|       1 |          36 |       00:00:03 |       32.70% |       0.6945 |          0.0010 |
|       1 |          38 |       00:00:03 |       93.04% |       0.6701 |          0.0010 |
```

Figure 6.Training network on NVIDIA™ GPU Titan x

III. RESULT ON TEST IMAGE

First, resize input image to a size of CNN.In our case size is 360*480. Then apply to trained CNN. We have tested our network for a number of images .some of tested images results are given below:

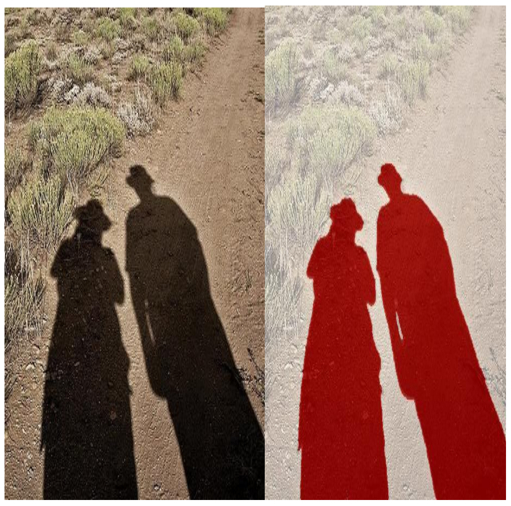


Figure 7. Input Image and Image with shadow detected

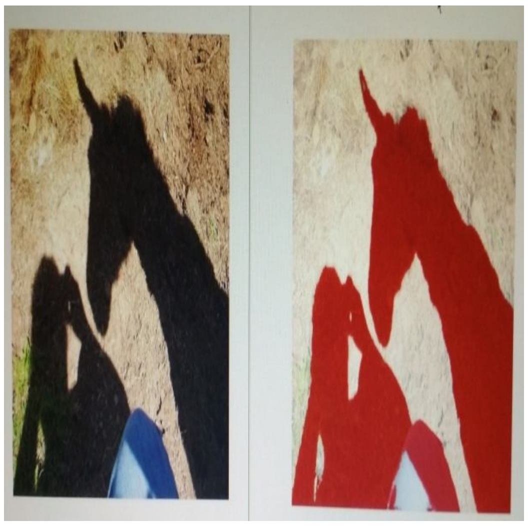


Figure 8. Input Image andImage with shadow detected

IV. CONCLUSION

We presented a use of convolutional neural network to learn the most relevant features to detect shadows from a single image. We showed that our framework performs best on a number of images and it does not depend on the object shape, the environment and the type of scene. In our framework Pre-processing is used to resize an image to fit to kernel of Neural Network.A labeled dataset of shadow created to indicate a region of interest. Class balancing using Inverse of frequency weighting is used to balance weights of classes.CNN is trained to detect shadow in the image. A network is tested on test Image.

V. FUTURE SCOPE

Proposed shadow detection framework can be modified together with the scene geometry and object properties for high-level scene understanding tasks.

REFERENCES

[1] Salman H. Khan, Mohammed Bennamoun, FerdousSohel, Roberto Togneri "Automatic Shadow Removal from aSingle Image" ieee transactions on pattern analysis and machine intelligence, vol. 38, no. 3, march 2016

[2] S. H. Khan, M. Bennamoun, F. Sohel, and R. Togneri, "Automatic feature learning for robust shadow detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recog., 2014, pp. 1939–1946

[3] J. Zhu, K. G. Samuel, S. Z. Masood, and M. F. Tappen, "Learning to recognize shadows in monochromatic natural images," in Proc. IEEE Conf. Comput. Vis. Pattern Recog., 2010, pp. 223–230.