Machine Learning Techniques to Play “Where is Waldo”

Pouneh Aghababazadeh, Jason Alexander Carruthers, Alex Fu, Matthew Gabor, Shunxu Huang, Wei Huang, David Ibrahim, Rachel Lee, David Lin, Lorenzo Martinico, Donald Pinckney, Michael Romero, Amir Sahabi, Doug Sherman, Christina Zhu

Finding Waldo, or more broadly, classifying an image as containing a pattern or not, is an interesting and difficult problem. Such pattern recognition is easy for humans, but computationally addressing this problem is much more difficult. Infinite perversions of the pattern, such as occlusion and transformed perspectives, size, and orientations, make building prediction models computationally expensive, and at best these models create many misclassification errors. This study mainly aims to evaluate the performance of various machine learning techniques and tools in solving this problem. Techniques include logistic regression, feed forward neural networks, decision trees, and support vector machines. Tools we use include Matlab, TensorFlow (python), Caffe(python), and Keras (python). We also use various methods of feature reduction - including traditional methods as well as modifying models to incorporate techniques such as the sliding window algorithm. There is little variation in the advantage of one method over the other as all methods had extremely high specificities, and significantly lower sensitivities. Many methods are augmented with modifications to the data and its preprocessing before feeding input into the machine learning algorithm or tool, which in some cases reflects better in the sensitivity and specificity measurement - however, the margin in improvement is not very large, and sensitivities are still disappointingly low. Ultimately, it seems that these results can be attributed to a weak data set with a very low proportion of positive training samples. There is still more work to be done to discover and develop good techniques and tools to find Waldo, and other patterns, computationally. For this dataset specifically, there may be few options unless more positive training samples are developed. Despite this, results from this study can be built upon to develop techniques for better general pattern image classification.

1. INTRODUCTION

Where’s Waldo presents an interesting problem in the field of object recognition. A Where’s Waldo puzzle presents the reader with a large, colorful image, conflated with visual noise and occlusion, in the hopes the reader has difficulty finding Waldo. This paper will highlight techniques we used and their effectiveness.

Previous techniques were non-generalized, and were unable to be extrapolated to new Where’s Waldo images. Optimal search paths using genetic algorithms [Olson 2015b][Olson 2015a] and template matching [Brownlee 2014] solve the problem in a very confined space, irrelevant to real world applications. For a more detailed history of methods used previously in the hunt for Waldo, review the attached complete report.

Instead of relying on solving a specific subset of the problem (i.e. the known Where’s Waldo puzzles), we opted to research how machine learning techniques can be used to visually understand the Waldo problem fundamentally, and use that understanding to extrapolate to new data sets.

We augmented preprocessed online datasets and qualified our classification as to whether a cropped image had Waldo or not. We approached using logistic regression, feed forward neural networks, including convolutional neural networks (CNNs), TensorFlow’s assorted methods, decision trees, and support vector machines (SVMs). Techniques specific to the field of visual recognition were successful in promising the potential for generalizing the essence of Waldo, but were hindered by low-quality and insufficient data.

2. LOGISTIC REGRESSION

2.1 Method

According to Hofmann, Lampert and Blaschko in “Beyond Sliding Windows: Object Localization by Efficient Subwindow Search”, binary classification was more effective in object recognition [Lampert et al. 2008]. To actually determine the location of an object, it is best to use binary classification in conjunction with a sliding window algorithm. The sliding window algorithm scans the original image by pixel increments to obtain smaller, overlapping sub-images which gets fed into the classifier as input. The classifier outputs the probability of the sub-image containing the object. After calculating probabilities for all the sub-images, the sub-image with the highest probability is returned as the predicted location of the object. We utilized MATLAB's `glmfit` (Generalized Linear Model Regression) to create our logistic regression classifier. Our Waldo classifier outputted a probability which we thresholded at 50%. An image contained Waldo only if the highest probability returned was greater than 50%.

We cropped Waldo's face and hat from the original images and resized those images to 27 by 20 pixels to minimize the number of features. We decided on these dimensions since we were able to reduce the number of features per sample to 72 after using histogram of oriented gradients (HOG). Not using HOG would have yielded 1620 features which is why we decided against raw RGB pixel values as features. HOG has also been shown to improve classification accuracy for human detection [Dalal and Triggs 2005]. These cropped images included different angles of Waldo's head such as his profile. To obtain the HOG features from the raw RGB values of each pixel, we used `extractHOGFeatures` in MATLAB. To increase our sample size, we added mirrored versions of these cropped images to our dataset. Since we used 27 by 20 pixel images of Waldo's head to train our classifier, we resized the original images of Waldo so his head would be as equally sized as possible in the training and testing images.

Using the previously mentioned classifier with the sliding window algorithm, we scanned the testing set images to determine Waldo's location. In addition, we cropped out Waldo's head from these images and reported our classifier's accuracy on the testing set images along with non-Waldo testing set images of the same size.

2.2 Results

Table I. : Logistic Regression Performance

Specificity	Sensitivity	Accuracy	Precision
99.96%	52.63%	99.23%	95.24%

Using the sliding window algorithm, on average, we classified 1.56% of the sub-images as containing Waldo.

2.3 Discussion

Nearly all of the samples occluded the majority of Waldo's body, so we specifically search for Waldo's face in our classifier. We decided not to include other aspects of Waldo, such as torso and legs, for multiple reasons. Firstly, those aspects of Waldo are larger, meaning the associated sub-images would require more features. Thus we would need more samples of those aspects to develop an effective classifier. Also, it would be much more difficult to integrate those multiple classifiers to find Waldo. Finally, Waldo's face is unique while other aspects of him are not and often share qualities with other objects to mislead the searcher.

Unfortunately the method employed lacks the ability to detect Waldo heads of varying size and facing directions. Furthermore, it requires that the user changes the proportions of the image so heads are of size 27 by 20. For future work, ideas and techniques mentioned in [Garg et al. 2011] can be incorporated to better locate Waldo in crowds.

3. FEED FORWARD NEURAL NETWORK

3.1 Method

[Egmont-Petersen et al. 2002] mentions using pixel values as input for an artificial neural network. As such, we decided to explore that avenue ourselves. In order to classify Waldo using Feed Forward Neural Networks, we used three methods: Black and White, RGB, and Color Mapping ([Hasnat et al. 2013] suggests that Euclidean distance can be a helpful metric for mapping similarities in two colored images of faces). Our dataset for black and white images only has one value per pixel (0 - 255), RGB has 3 values per pixel (0 - 255, 0 - 255, 0 - 255) both with 4,096 pixels each (64x64 pixel images). For Color Mapping we mapped each RGB value to it's nearest color value by using the Euclidian Distance, on a subset of 8 colors. Using Keras, "a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano", we fed in pixel values as input features into our FFNN. Using a configuration of two hidden layers, each with the number of nodes equaling half the number of features, we used back propagation to train the neural network over 20 epochs. We also ran the network with configurations of up to 200 hidden nodes in each layer and achieved similar results.

3.2 Results

Feedforward neural networks suffered much more from the disproportionate set of Waldo images then Logistic Regression. The sensitivity of Feedforward neural networks is at best only 8.33%.

Table II. : FFNN Performance

Feature Space	Specificity	Sensitivity	Accuracy	Precision
Color	100%	8.33%	99.32%	100%
B & W	100%	7.14%	99.15%	100%
Color Map	100%	8.33%	99.32%	100%

3.3 Discussion

3.3.1 *Disproportionate Set and rapid convergence.* Given that there were 39 Waldo’s in the entire data set of 5,378 samples (.7% positive samples), the dataset is inherently difficult to train. Furthermore, Waldo isn’t statically appearing in each positive image, meaning in some images he may be missing a hat, or his glasses, or half of his face due to obstruction. In humans “the brain learns to link multiple view-specific categories of an object to view-invariant categorical representation of the object” [Chang et al. 2014]. However, within the images Waldo does appear in, he does not appear in the same place, making it difficult for the neural network to detect a trend in where Waldo might appear, which becomes problematic as our network tries to map discrete pixel locations to Waldo’s appearance. Due to the dataset imbalance, and the Neural Network detecting the input as seemingly random, the model converged very quickly with the assumption that every image did not contain Waldo. This can be seen in the sensitivity of the classifier as it misclassified nearly every Waldo as not Waldo.

3.3.2 *Color mapping helped with feature reduction.* In order to retain color in the feature set without overfitting (given a 64x64 pixel image, 3 values for each pixel would lead to 12,288 features, way more than the 5,378 samples). We mapped Euclidean distance to a set of 8 prespecified colors (all the permutations of 0 and 255 of length 3). This way we are able to retain color as a distinguisher while not having to keep all three values of the color, thereby reducing our input down to 4,096 features.

3.3.3 *Why we used Keras vs. pybrain, other libs.* If you have 4096 inputs, and your first hidden layer is 2048, the network has $4096 * 2048 = 8,388,608$ edges in between the input layer and the first hidden layer, $2048 * 2048 = 4,194,304$ edges between the first hidden layer and the second hidden layer, and 4096 connections between the last hidden layer and the output layer. Considering the number of edges involved, this is a very computationally intensive task, so we wanted a library that could perform these computations in parallel. Keras offered GPU/CPU optimization, so it was ideal for this task, as opposed to libraries such as pybrain where calculations are done in succession, only on the CPU.

3.3.4 *Difficulties.* As mentioned before, the proportion of positive samples to negative samples did bias our findings towards categorizing images as non-Waldo images. However, there were some additional issues regarding the results themselves. The results section for the feed-forward neural net indicates that the sum of the counts of true positive and false negative, which should indicate the total number of actual positive samples, are not always the same. It appears that Keras, therefore, dropped some samples in the preliminary data import so that the total number of samples used in the experiment is lower than what was available. Ideally, all samples would be used. However, given the time constraint, researchers were unable to investigate further, and we encourage other curious minds to attempt rectify these confounding errors.

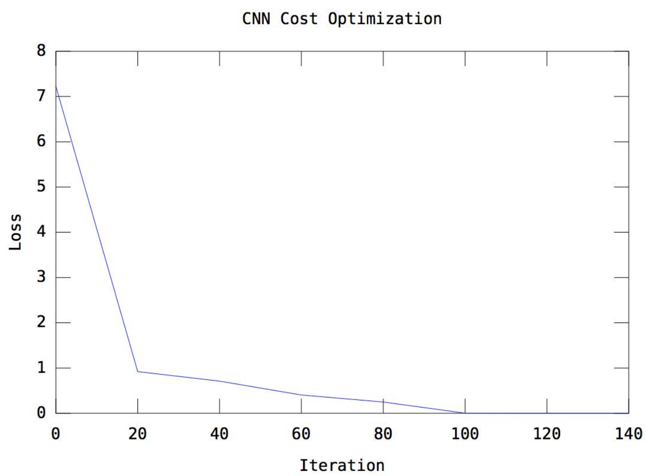
4. CONVOLUTIONAL NEURAL NETWORK (CAFFE)

4.1 Method

We utilized Caffe as our deep-learning framework. Caffe was compiled to utilize CUDA 8.051, OpenBLAS as the Basic Linear Algebra Subprogramming system, and Anaconda python 2.7 to better support high-performance computational packages. We opted for a semi-bounded-box version of input data for classifying Waldo, with each image up-scaled to 256x256. The dataset was retrieved from Jessie Salas' DeepRed [GitHub 2015], and was augmented in order to increase the size of available data. The convolution later consisted of the following layers, as used by [Krizhevsky et al. 2012]. There are 5 convolutional layers, 6 ReLU layers, 3 pooling layers, 2 local response normalization layers, 2 dropout layers, and 3 fully connected layers. Loss was computed via softmax as per Caffe's default. The CNN began training with a learning rate of 0.01. Every 100,000 iterations the learning rate would decrease by 10%. This strategy also stemmed from [Krizhevsky et al. 2012]. The training would use all images except 8, which would be used for testing purposes.

4.2 Results

After a significant training time and extreme heat and noise output the loss function converged to 0:

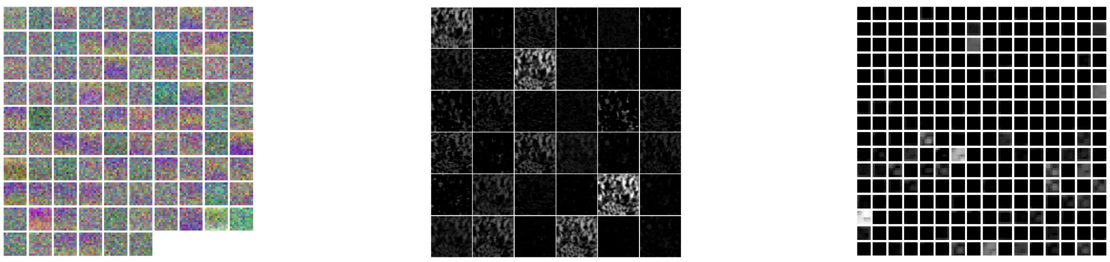


The majority of the data was used for training, so the testing set was extremely small, only 8 samples. The model scored 100% accuracy and precision, correctly marking 4 Waldos and 4 non-Waldos. The Waldo's and non-Waldo's were previously unseen, but had high similarity in relation to the training set, which is both critical and detrimental to the integrity of the model. Copies of the Caffe prototexts can be found in the attached files.

4.3 Discussion

Due to the extreme specificity of the training set, i.e. being pixelated crops of very specific full-bodied Waldos, and small scale of the data set, CNN's translational invariance isn't very effective and cannot account for Waldo in larger images. The variance between a normal, large-scale Where's Waldo images and our full-bodied but miniscule crops of Waldo don't accurately portray him in a real-world' scenario. The only way to use the CNN as we have trained it currently would be subdividing the full image into small sections and checking each one for Waldo's presence via a sliding window method on images of a particular size. Lack of computational power prohibits larger data sets and implementation of data augmentation.

One possibility for speeding up training would be to use Transfer learning, a technique that integrates a pre-existing Convolutional Network in training over the new dataset when there is not enough data. By having a previously trained CNN that classifies similar images, it would be possible to use the values of neurons in the last layer before the output layer as a new vector of features, and train a classification method over all such vectors for the new images. Unfortunately at this moment we weren't able to find a publicly available CNN that has been trained over similar enough images (e.g. cartoon characters). As such a network is not available, it might be of use to train the new images over another existing network, which at the lower level may still be able to recognise some defining features of the images, and use the value of neurons at that layer to classify the images. The small quantity of training data prevents us from adjusting the weight of a pre-existing CNN by training it over the new images, as we might otherwise choose to do in the presence of a larger dataset.



1st Layer   5th Layer After Pooling   6th Fully Connected Layer

Fig. 2: Visualizations of various layers

5. TENSORFLOW INCEPTION MODEL METHOD

5.1 Method

Using the Google TensorFlow pre-trained Inception model, we re-trained the model to recognize waldo using the 64 by 64 pixel dataset [Google 2015]. In order to add more positive waldo images, existing waldo images were flipped horizontally (to preserve face shape) and randomly darkened and lightened. Cross-validation was used to reduce overfitting. Specifically, we are using 500 training steps, a random brightness variance of 10, and the flip-up-down flag to run the retraining on Inception.

5.2 Results

Table III. : Tensorflow results

Specificity	Sensitivity	Accuracy	Precision
79.98%	100%	80%	1.9%

5.3 Discussion

5.3.1 *About TensorFlow.* This section uses the pre-trained network for the Inception model, which was released by Google in 2015. This model, having been trained on over one hundred thousand different images in over a hundred categories has essentially trained itself on shape and edge detection.

When attempting with the raw data (no image flipping or random brightness), we were able to get an accuracy of 79.8% on the test data set. By default, TensorFlow uses 10% of the data for testing, 10% for cross-validation, and the rest (80%) for training. Whereas 100% of the positives (4 images) were correctly identified, only 80% of the negatives (426 images out of 533) were correctly identified.

By using cross-validation after every batch, we can make sure that the model is not overfitting, and will give us a better outcome in the long run. However, adding a random brightness variance of ten and random up-down flips did not seem to change the classification accuracy.

5.3.2 *Why TensorFlow's Inception Model?*. The main reason the Inception model was picked was due to the fact that it was already trained to recognize people and objects [Mordvintsev et al. 2015]. The common denominator of the Waldo images is that his face is always visible, and TensorFlow was able to pick up on this.

5.3.3 *Overcoming Difficulties.* Like the other groups, the small ratio of positives to negatives was detrimental to the overall classification. However, given that there were nineteen total images, the neural network was able to narrow down the total number of 64 by 64 images down from one thousand (10% of the the five thousand images) to a mere 112 in the raw dataset, making Waldo much easier to find.

5.3.4 *Conclusion.* Overall, the neural network still has trouble pinpointing a specific Waldo, but it has at least learned to recognize the facial structure of Waldo and can narrow down the number of tiles needed to search through in order to actually find Waldo in the image.

6. DECISION TREES

For the complete documentation on decision trees, see the attached complete report. The following is an edited version that omits most of the information to save space.

6.1 Method

1.0.0: Our binary decision tree is constructed from and takes as input images. To reduce the feature space and to give more weight to pixels that are more likely to be a part of Waldo, we apply filters to each image, and for each filter extract a new binary feature. Three filters are applied, therefore transforming the 64x64x3 feature space into one of size three. A set of training images are then used to dynamically construct the decision tree. The remaining images from the set of all 64x64 Waldo and non-Waldo images are then fed into the decision tree and classified.

1.1.0: Color map filters are applied independently to each sample image.

1.2.0: For a given binary feature, the respective filtered image is used to obtain an array of regions. The filtering method used to obtain these regions is a recursive four-point stack based flood fill algorithm. Regions of flood-filled pixels are stored by like-color.

1.2.1: The first feature is whether the image has any red and white stripes.

1.2.2: The second feature is detecting whether the image has at least one pair of black glasses.

1.2.3: The third feature is whether the image contains at least one pair of blue pants.

1.3: After acquiring the three features for each image, the binary decision tree is dynamically constructed on 70% of all of the Waldo images and 70% of all of the non-Waldo images. Information gain is used as the splitting algorithm at each node [Amro 2009]

6.2 Results

Table IV. : Decision Tree Performance

Specificity	Sensitivity	Accuracy	Precision
67.84%	74.07%	67.89%	1.64%

6.3 Discussion

Other splitting algorithms may be used instead of information gain [Rokach and Maimon 2005]. Additionally, the benefit of using the twoing criteria as the splitting algorithm is that the decision tree can then operate on features with continuous values, becoming a regression decision tree in the vein of CART [Rokach and Maimon 2005]. The benefit of creating a regression decision tree is that a large image with a lot of noise and containing a single Waldo (a Wheres Waldo image before cropping out all of the people) can be fed into the program, facial detection can be used to identify sub-images with people, feature extraction as described can be used on each of these sub-images, these images can be fed into the decision tree, and the output of the decision tree can be one or more likely coordinates of Waldo within the scene.[Solberg 2009]

7. SVM

7.1 Method

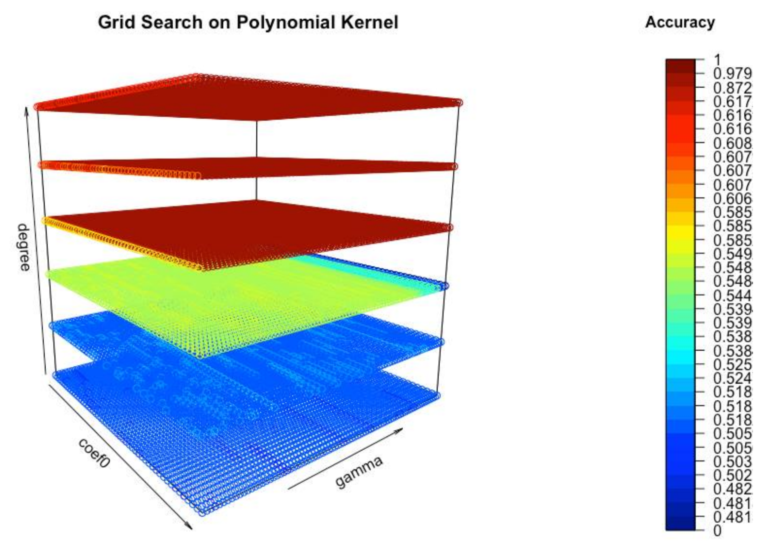
7.1.1 *Pre-Processing the Data.* The data format was in the form of pictures. In order to properly evaluate it, we needed to convert the data into features. To do this we used matlab’s `imread` function. This converted each picture into it’s RGB pixel values. However, since we has 64, 128, and 256 bit images we needed to find a way to combine the different sizes. Some standard compression algorithms simply merge adjacent pixels in order to compress an image. We did this for the 128 and 256 bit images in order to match the size of the 64 bit images. This allowed all available pictures to be used in a single algorithm.

7.2 Results

7.2.1 *Grid Search to Find Optimal Hyperparameters.* Since we have many Kernels to choose from with SVM, and at first glance the data looks like it is not linearly seperable, we used the grid search technique to analyze multiple Kernels. One of the Kernels was the Polynomial Kernel given by,

$$f(u,v) = (\gamma \langle u,v \rangle + c_0)^D$$

The Hyperparameters with this Kernel are $\gamma, D,$ and c_0. We tuned on all three of these parameters and measured the accuracy after to determine the best values. Here is the plot containing the accuracy for each choice of these parameters.



1. Grid Search on γ, D, and c_0 for Polynomial Kernel

We see that degree has the most impact on the accuracy of the model, while extreme γ and c_0 values also lead to worse accuracy for a given choice of D. In general, a kernel with higher degree will give us better Accuracy and Specificity, but worst precision and sensitivity. Meanwhile, change in gamma will have minimal effect on the result, but coefficients appears to not contribute appreciably to the results.

We found that the Radial Kernel produced the best results overall. The following table illustrates the values of D, γ, and c_0 and the associated Accuracy, Precision, Sensitivity, and Specificity.

Table V. : Values on the optimal attributes

Optimal	D	c_0	γ
Accuracy	7.0000	4.6000	4.6001
Specificity	4.0000	4.9000	0
Sensitivity	4.0000	4.4000	7.8001
Precision	6.0000	1.5000	0.0001

7.2.2 *SVM on All Pixel Values.* Once the Kernel was decided, we began to train the data. The first run of the data was to train an SVM (using the libsvm package) on all the pixel values. Since we had RGB values for 64 bit images, this gave us $64^2 * 3 = 12288$ features. We tested many different kernels and measured the accuracy, precision, specificity, and sensitivity for each of them. These were given by

Table VI. : Accuracy Values for Different Kernels

Kernel	Specificity	Sensitivity	Accuracy	Precision
Linear	99.816%	33.929%	98.486%	79.167%
Polynomial	98.631%	33.929%	98.631%	95.000%
Radial	100%	40.00%	98.911%	100%
Sigmoid	97.43%	12.14%	97.846%	8.21%

7.2.3 *Using PCA to Reduce Dimensionality.* Next we tested dimensionality reduction by using PCA. Following singular value decomposition we found the most informative principle components by selecting the right singular vectors that corresponded to the largest 100 singular values. For X as the dataset, the singular value decomposition is given by,

$$X = USV^T$$

Then we selected the right singular vectors V corresponding to the largest singular values and reduced X into

$$\tilde{X} = X[V_1, \cdots, V_{100}]$$

We then trained SVM on $\tilde{X}$. After also transforming the testing set, we measured the accuracy, sensitivity, specificity, and precision as

Table VII. : Accuracy Values for Different Kernels

Specificity	Sensitivity	Accuracy	Precision
100%	46.939%	99.046%	100%

7.3 Discussion

Although model selection when using a SVM is a daunting task, a grid search can often easily determine the best hyper parameters. Using the best hyperparameters per the grid search, we found the Radial Kernel to be the best method according to accuracy, sensitivity, precision, and specificity. After using PCA to reduce the dimensionality, we found the same accuracy; implying that much of the information is found within a significantly smaller subset of the entire space. Future projects would require that we find more positive samples since the proportion of “Not-Waldo” pictures to “Waldo” pictures was substantially off-balance.

8. CONCLUSION

Given these findings, we have the following discoveries. Logistic regression succeeded on classifying Not-Waldo but did not provide a more efficient way of identifying Waldo than random guessing, due to its limitation in computer vision and the lack of Waldo training samples. Feed Forward Neural Networks aimed to solve the problem, and while not having worse running time than the algorithms in [Olson 2015b][Olson 2015a], its Waldo identification suffered due to the small Waldo samples compared to the enormous Not-Waldo samples. TensorFlow identified Waldo correctly every time, but would sometimes produce false positives; a human could comb through each returned positive and correctly ID Waldo, but this raises the problem of relying on a human to complete. Decision trees performed only slightly better than making a random binary classification of each cropped image, due to flawed feature extraction and a small Waldo sample set; after constructing the decision tree, the actual classification takes microseconds. SVMs revealed that Radial Kernels yield the highest proportions of specificity and the like, yet ultimately suffer on overall sensitivity due to an unbalanced data set. CNN’s were largely successful, correctly classifying Waldo and non-Waldo images most of the time. However, due to data, time, and computational limitations, the testing set used was very small. Lastly, while different approaches have various performance and they all have their own strength and weakness. For example, TensorFlow and CNN give the most impressive performances but they both require intensive computational power; SVM runs fairly fast but performs in an unsatisfactory fashion on specificity and precision due to the limitation of the data set available. However, these approaches all faces some common difficulties, such as an unbalanced data set, the limitation on computational power, and a feature space with a very large number of dimensionalities. Among these limitations, the most prominent obstacles is unbalance of the dataset. That is, a large part of the data set is of the non-waldo class, such that the punishment of classifying a true-waldo sample to a non-waldo class is weighted much less in the cost function. It, inevitably, resulted in a very low sensitivity overall in most of our teams. All in all, a more representable/useful data set will contribute appreciably in future performance.

REFERENCES

Amro. 2009. What is “entropy and information gain”? http://stackoverflow.com/questions/1859554/what-is-entropy-and-information-gain. (Dec. 2009). Accessed: 2016-10-22.

Jason Brownlee. 2014. Using OpenCV, Python and Template Matching to play “Wheres Waldo?”. http://machinelearningmastery.com/using-opencv-python-and-template-matching-to-play-wheres-waldo/. (May 2014). Accessed: 2016-10-22.

Hung-Cheng Chang, Stephen Grossberg, and Yongqiang Cao. 2014. Where’s Waldo? How perceptual, cognitive, and emotional brain processes cooperate during learning to categorize and find desired objects in a cluttered scene. Frontiers in Integrative Neuroscience 43, 8 (June 2014). DOI:http://dx.doi.org/10.3389/fnint.2014.00043

Navneet Dalal and Bill Triggs. 2005. Histograms of Oriented Gradients for Human Detection. Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition 1 (June 2005). DOI:http://dx.doi.org/10.1109/CVPR.2005.177

Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. 2002. Image processing with neural networks - a review. Pattern Recognition 35 (Oct. 2002). DOI:http://dx.doi.org/10.1016/S0031-3203(01)00178-9

Rahul Garg, Deva Ramanan, Steven M. Seitz, and Noah Snavely. 2011. Where’s Waldo: Matching People in Images of Crowds. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. 1793–1800. http://www.cs.washington.edu/homes/rahul/data/cvpr2011.pdf

GitHub. 2015. deepRed. https://github.com/JessieSalas/deepRed. (Oct. 2015). Accessed: 2016-10-22.

Google. 2015. How to Retrain Inception’s Final Layer for New Categories. https://www.tensorflow.org/versions/r0.8/how_tos/image_retraining/index.html. (2015). Accessed: 2016-10-21.

Abul Hasnat, Santanu Halder, D. Bhattacharjee, M. Nasipuri, and D. K. Basu. 2013. Comparative Study Of Distance Metrics For Finding Skin Color Similarity Of Two Color Facial Images. National Conference on Advancement of Computing in Engineering Research 3, 2 (2013). http://airccj.org/CSCP/vol3/csit3210.pdf

C Jongkreangkrai, Y Vichianin, C Tocharoenchai, and H Arimura. 2016. Computer-aided classification of Alzheimer’s disease based on support vector machine with combination of cerebral image features in MRI. Journal of Physics: Conference Series 694 (2016). DOI:http://dx.doi.org/10.1088/1742-6596/694/1/012036

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Christoph H. Lampert, Matthew B. Blaschk, and Thomas Hofmann. 2008. Beyond Sliding Windows: Object Localization by Efficient Subwindow Search. CVPR (2008). https://www.computer.org/csdl/proceedings/cvpr/2008/2242/00/04587339.pdf

Kristin A. Linn, Bilwaj Gaonkar, Theodore D. Satterthwaite, Jimit Doshi, Christos Davatzikos, and Russell T. Shinohara. 2016. Control-group feature normalization for multivariate pattern analysis of structural MRI data using the support vector machine. NeuroImage 132 (May 2016), 157–166.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. 2015. Inceptionism: Going Deeper into Neural Networks. https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. (June 2015). Accessed: 2016-10-21.

Mustakim, Agus Buono, and Irman Hermadi. 2016. Performance Comparison Between Support Vector Regression And Artificial Neural Network For Prediction Of Oil Palm Production. Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information) 9, 1 (Sept. 2016). DOI:http://dx.doi.org/10.21609/jiki.v9i1.287

Randal Olson. 2015a. Computing the optimal path for finding Waldo. https://github.com/rhiever/Data-Analysis-and-Machine-Learning-Projects/blob/master/wheres-waldo-path-optimization/Where’s\%20Waldo\%20path\%20optimization.ipynb. (Aug. 2015). Accessed: 2016-10-22.

Randal Olson. 2015b. How I Found the Optimal Where’s Waldo Strategy With Machine Learning. https://www.wired.com/2015/02/created-perfect-wheres-waldo-strategy-machine-learning/. (Feb. 2015). Accessed: 2016-10-22.

Lior Rokach and Oded Maimon. 2005. Data Mining and Knowledge Discovery Handbook. Springer Science+Business Media, Inc.

Anne Solberg. 2009. Hough transform. http://www.uio.no/studier/emner/matnat/ifi/INF4300/h09/undervisningsmateriale/hough09.pdf. (Nov. 2009). Accessed: 2016-10-22.

9. AUTHOR CONTRIBUTIONS

Table VIII. : List of Author Contributions

Author	Contribution
Pouneh Aghababazadeh	Part of FFNN subteam and writing abstract
Jason Alexander Carruthers	Introduction and decision trees
Alex Fu	Retrained Google Inception Engine to recognize Waldos
Matthew Gabor	Helped create Forward Feed Neural Network portion and assisted with discussion section of report
Shunxu Huang	Development/Writeup for SVM using Llibsvm on Matlab
Wei Huang	For logistic regression: Helped collect more data, resized the images to reduce the number of features, wrote the code to run logistic regression and test the performance of the model, provided the calculation and assisted with the writeup
David Ibrahim	For logistic regression: Collected more data (doubled dataset), manually found waldo in the new data, cropped the images of waldo, wrote the code to run logistic regression, wrote the sliding window code, and helped with the writeup
Rachel Lee	Coordinated group meetings and task allocations and helped with Logistic Regression methods
David Lin	Development/Writeup for CNN using Caffe
Lorenzo Martinico	Development/Writeup for CNN using Caffe
Donald Pinckney	LaTeX formatting, assembling final paper, and TensorFlow subteam
Michael Romero	Development/Writeup for CNN using Caffe
Amir Sahabi	Helped write code for FFANN + derive statistics and helped write FFANN portion of methods + discussion
Doug Sherman	Development/Writeup for SVM using Llibsvm on Matlab
Christina Zhu	LaTeX formatting, assembling final paper, and TensorFlow subteam