CS230

CS230 Final Report - Spring 2018 Standardized Object Detection and Classification for Unmanned Aerial Vehicles

Joshua F. Payne[*]
Department of Computer Science
Stanford University
joshp007@stanford.edu

Peggy (Yuchun) Wang
Department of Computer Science
Stanford University
wangyuc@stanford.edu

Abstract

With their high mobility and increasing ubiquity, Unmanned Aerial Vehicles (UAVs) are quickly transforming many different industries and inspiring new approaches to tasks traditionally performed by humans. Tasks of these flavor that involve computer vision as a principle component such as search-and-rescue, infrastructure inspection, and logistics evaluation are what inspired our project's focus: to assess a standardized object of our choice, train deep learning models to detect, localize, and classify it, and report the findings. In this paper, we introduce one family of such objects (geometric shapes with nonstandard alphanumeric characters inscribed), several algorithms that perform well on each of the three computer vision tasks above, and alternative methods we've started to investigate that we feel warrant future exploration. We also attempted to detect, localize, and classify nonstandard human objects in aerial images, which have applications in search-and-rescue, surveillance, and defense, as exploration of the feasibility of aerial image processing.

1 Introduction

Our approach to the challenge of aerial image detection, localization, and classification was inspired by the Object Detection, Classification, and Localization section of the 2018 AUVSI-SUAS challenge[2]. The standard objects described there are geometric, 2-dimensional shapes of any color and a diameter of 12"-24". Painted inside each of these shapes is an alphanumeric character of any color.
For detection and localization, the inputs are 3840$\times$2160 RGB images, which are processed by the YOLO algorithm we've implemented. The output to our detection and localization component is a bounding box around a detected shape, as well as a probability for what the class of that shape is. To narrow down localization, the bounding box smoothed using a bilateral filter and inputted into our k-means segmentation algorithm, which outputs the pixel arrays of each cluster, isolating and centering the geometric shape. This creates a cluster mask that yields the average color of the geometric shape, and calculating the Euclidean norms between these RGB values and the RGB values of several 'template' colors outputs the nearest color of the geometric shape. From this cluster mask, the painted alphanumeric is also isolated, binarized, and reshaped into a 28$\times$28 image. Four 90° rotations of this alphanumeric is inputted into our alphanumeric-trained convolutional neural network (CNN), and the prediction with the highest confidence yields the letter and cardinal orientation of this image.
The concept of processing standard objects in aerial computer vision has applications in search-and-rescue, logistics, and infrastructure, where deep learning models may be trained to process such images.

2 A Brief Overview of the Progression of Our Approach

Our approach to this has undergone several iterations. For localization and detection, we first attempted using an OpenCV implementation of SURF to find 'blobs' in an image. Unfortunately, SURF did not perform well on test

[*]Implementations for the algorithms presented in this paper can be found on Github: https://github.com/Josh-Payne/cs230/

,

images, resulting in a high number of false positives and low accuracy. We then decided to use the YOLO (You Only Look Once) algorithm to detect, localize, and classify geometric shapes in one step.
For segmentation, we attempted using three-cluster k-means but found that the alphanumeric was seldom recognized over deviations in the background. Thus, we decided to use two clusters for the background and geometric shape and apply the mask of the shape's average color to isolate the alphanumeric.
For alphanumeric character classification, we first trained a neural network on characters generated from a font and augmented with rotation, skew, scale, and translation. While this model performed well on examples from the training distribution, it did not generalize well to 'more difficult' handwritten characters. We then used the EMNIST dataset (described in section 4) to build a model that yielded less accuracy, but overall performed much better on general characters due to the higher difficulty of the dataset. We then decided to explore a Siamese Model (which will be discussed in future sections) to see if that would yield a higher accuracy.
We then attempted nonstandard human object detection using YOLO. This is an area of ongoing research since we do not have a good dataset readily available. We were able to get good performance on the training dataset, but we suspect the YOLO model overfitted the training data since it performed poorly on test data. We will continue to refine this model in the future.

3 Related work

3.1 Detection and Localization

The first approach we took was SURF (Speeded Up Robust Features)[3]. SURF finds blobs of interest by using a Laplacian of Gaussian (LoG) with a Box Filter to detect edges. This is similar to the convolutional filters we've seen in class, but uses the kernel to instead approximate a second derivative of the image. To combat the resulting noise sensitivity, the Laplacian filter is convolved with a Gaussian smoothing filter. Unfortunately, this did not perform well on our aerial images, often yielding false positives or no positives at all. Our second approach was to implement the YOLO algorithm by Redmon et al.[14] discussed in class. This Tensorflow implementation was aided by a project by Github user thtrieu entitled 'Darkflow'[5]. YOLO uses a divide-and-conquer approach to approximate bounding boxes for objects in images in 'real-time' (classifying at around 4 fps on our input images on a 16 GB CPU).

3.2 Classification

We initially attempted character classification using a simple neural network and convolutional neural network with little success. Thus, we decided to apply the one-shot learning approach we learned about in class and designed a Siamese neural network for learning internal encodings of examples based on clusters. This method was presented in a paper by Koch et al.[8] and provided an important insight as to how clustering data in higher dimensions could help in comparing a new character to one in a database to determine whether or not they are the same. This encoding representation can be visualized using a technique called t-SNE, presented in a paper by Maaten et al.[9], and was implemented on the MNIST dataset of handwritten numbers by Github user ywpkwon in Tensorflow[15]. Our approach was to train a Siamese Convolutional Neural Network for alphanumeric clustering weights by creating a dataset of positive (two examples of the same class) and negative (two examples of different classes) examples - an unsupervised learning technique. We then used transfer learning by applying those weights to a new model that used examples of matches of template characters of each class with positive examples from the EMNIST dataset and negative examples from the EMNIST dataset. With this model, we saw a training set accuracy of 97.68% and a dev set accuracy of 97.08% after 5 epochs, much higher than the accuracy we've achieved by simply classifying EMNIST characters straightaway. This is likely because a 'random' character picked from the dataset as a match is likely not to be very similar to a given character, while if the CNN guesses an 'I' as a '1', it will lose accuracy. The technique of comparing template characters to semi-standard characters that we'll see in the competition using these encodings is still promising, though. We used contrastive loss for our loss function, RMSProp for our optimization function, and the same layer specs used in our final convolutional model for our hidden layers. We decided to stick with the other model which was fully integrated but feel that the Siamese model is certainly worth exploring.

4 Dataset

Since there is no dataset readily available for our standard objects, we created our own datasets to train our models on for detection and localization. We also used the EMNIST Balanced dataset, a subset of the NIST Special Database 19, comprised of 117,299 handwritten 28×28 numeric and alphabetic characters, for both standard object dataset generation and alphanumeric character classification.
We separated the challenge of creating objects that would be similar to standard objects we might see in the field into three tasks: background generation, shape generation and placement, and character placement. To generate background images that might look like the background of a field, we scraped Google for aerial images of fields

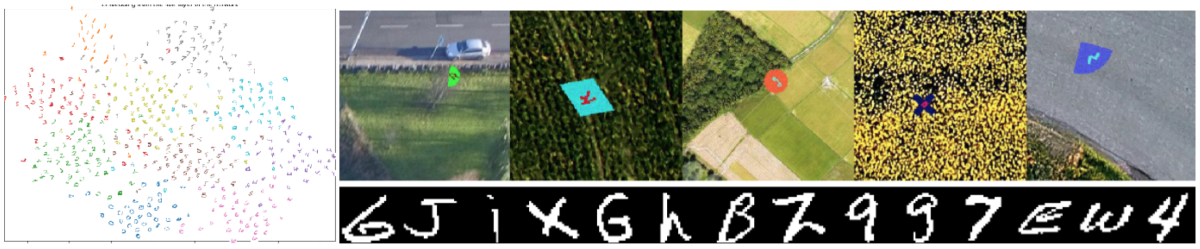
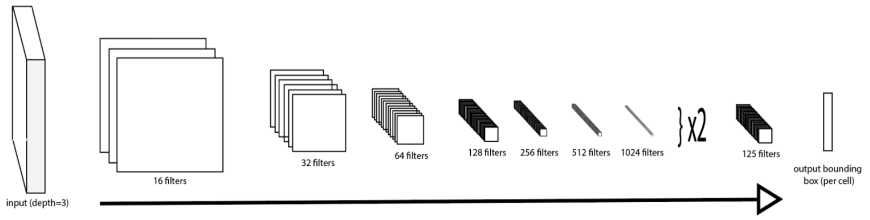


Figure 1: (left) A t-SNE visualization of the 128-dimensional MNIST digit encoding clustering[15], (top right) examples from our generated dataset for training YOLOv2, (bottom right) examples from the EMNIST dataset



Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 2: (left) Our YOLO model, based on Darkflow's Tiny-YOLO model (max pooling layers with a pooling window of 2×2 and a stride of 2 follow the first 5 convolution layers and the 6th is followed by a max-pooling layer with a pooling window of 2×2 and a stride of 1). (right) Our loss is the same as the one presented in Redmon's paper, a combination of bounding box spatial prediction and class prediction accuracy, each weighted with hyperparameter λ).

and forest images and gathered images from our flights and wrote a script that saved 300×300 croppings of these images.
To generate shapes for the YOLO algorithm, we created a template of each of the shapes that we've seen from the competition - circle, half circle, quarter circle, plus sign, star, triangle, and diamond. We then took 28×28 characters, each of a random color with small color variances throughout, from the EMNIST dataset and placed them on the shapes with random rotations and very small random translations from center of shape, which also has a random color and color variation. We then placed this shape and color unit (after random rotation, translation, and scaling between 40 and 75 pixels in diameter) onto the 300×300 background image. We created corresponding XML files for bounding box annotations of where the shape was placed on the 300×300 background images. We generated 1,000 instances per class, with a total set size of 10,000 examples.
For character recognition, we used several augmentation techniques to beef up the EMNIST dataset. Those techniques include rotating characters $\pm$ 45°, height and width shifting, shearing, scaling, and ZCA whitening for better edge internal representation. These were done using Keras' ImageDataGenerator class at runtime.
There are almost no complete datasets available for human detection from UAVs. We attempted to use the Stanford Drone Dataset, however, the downloaded 71Gb zipped file was too big for us to extract, and there was no way for us to only download parts of the dataset. After searching online, our best alternative was images from the Trajectory Forecast Benchmark [18]. We then hand-annotated the images using BBox-Label[4], and wrote a script to convert the annotated .txt files generated from BBox to the Pascal VOC XML annotation format.

5 Methods

5.1 Detection, Localization, and Classification of Geometric Shapes

For this task, we implemented the YOLO method presented in class and Redmon's paper as mentioned before. Redmon's model uses 24 convolutional layers and two giant fully connected layers at the end, but we used the tiny-yolo model from Darknet to train on, which only uses 8 convolutional layers and no fully-connected layers. This is due to the decent results this provided without needing to train for an extremely long time (even on a powerful GPU). This model, trained on 10 classes (one for each geometric shape), started out with a loss of around 114 and ended up with a loss of 1.5 by the second epoch, at which point it converged.

5.2 Segmentation

Once bounding boxes were established we segmented the image into two classes using k-means clustering, minimizing the objective function

$$\sum_{i=1}^{2}\sum_{j=1}^{n} ||{x_i}^{(j)} - c_j||^2$$

where n is the number of pixels in the bounding box, ${x_i}^{(j)}$ is a pixel, and c_j is the cluster center. Coincidentally, we used a similar Euclidean norm technique to calculate the nearest template color to the average color of the mask of the geometric shape:

$$\sum_{i=1}^{3} ||x_i - t_i||^2$$

where i is an RGB channel, x_i is the value of the channel i of example x, and t_i is the value of the channel i of template color t. The isolated alphanumeric image is then binarized, cropped, and reshaped into a 28 $\times$ 28 array to be classified by the ENMIST-trained CNN.

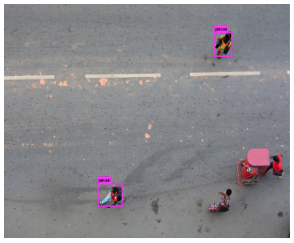


Figure 3: YOLOv3 with threshold .2

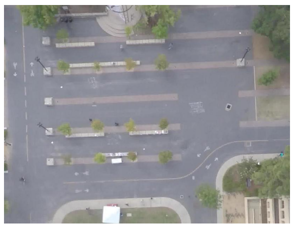


Figure 4: YOLOv3 with threshold .2

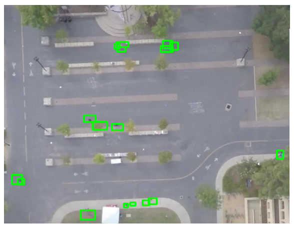


Figure 5: Trained tiny-yolo-voc with threshold .2

$$\begin{cases} v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1-\beta_1)\frac{\partial \mathcal{J}}{\partial W^{[l]}} \\ v^{corrected}_{dW^{[l]}} = \frac{v_{dW^{[l]}}}{1-(\beta_1)^t} \\ s_{dW^{[l]}} = \beta_2 s_{dW^{[l]}} + (1-\beta_2)(\frac{\partial \mathcal{J}}{\partial W^{[l]}})^2 \\ s^{corrected}_{dW^{[l]}} = \frac{s_{dW^{[l]}}}{1-(\beta_1)^t} \\ W^{[l]} = W^{[l]} - \alpha \frac{v^{corrected}_{dW^{[l]}}}{\sqrt{s^{corrected}_{dW^{[l]}}} + \varepsilon} \end{cases}$$

Figure 6: Adam Optimizer

5.3 Classification of Alphanumeric Characters

We initially attempted character classification using a generated dataset and a simple neural network with four hidden layers in Keras (784-neuron input, fully-connected 1024-neuron layer with ReLU activation, dropout layer of threshold τ = 0.2, fully-connected 512-neuron layer with ReLU, dropout layer of τ = 0.2, output 47-neuron softmax). For our loss, we used categorical cross-entropy defined as

$$-\sum_{c=1}^{47} y_{o,c}\log(p_{o,c})$$

where y is the binary indicator (0 or 1) if class label c is the correct classification for observation o and p is the predicted probability observation o is of class c. We chose to use Adam for the optimizer, discussed in class as a combination of RMSProp and Momentum. The update rule for $l \in \{1, ..., L\}$ is presented in Figure 6 where t counts the number of steps taken of Adam, L is the number of layers, β_1 and β_2 are hyperparameters that control the two exponentially weighted averages, α is the learning rate, and ε is a very small number to avoid dividing by zero. Training this for one epoch on a 16-GB CPU took around 20 seconds and had a dev set accuracy of 75%. However, the model did not generalize well to rotated characters, so we decided to move on to a CNN.
For this network, we used 9 hidden layers (784-neuron input, 5×5 kernel size convolutional layer with 32 filters and ReLU activation, a max-pooling layer with 2×2 pooling window, 3×3 kernel size convolutional layer with 32 filters and ReLU activation, a max-pooling layer with 2×2 pooling window, flattening layer, fully-connected 1024-neuron layer with ReLU, dropout layer of τ = 0.5, fully-connected 1024-neuron layer with ReLU, dropout layer of τ = 0.5, output 47-neuron softmax) trained one epoch in around 100 seconds with a test accuracy of around 82% after one epoch on a 16-GB CPU. For both networks, we used categorical cross-entropy as the loss function described above, Adam as the optimizer as described above, a 95% - 5% training/dev dataset split, and a minibatch size of 512 training examples. We used a learning rate of 0.001, β_1 of 0.9, β_2 of 0.999, and ε of 1e-8, set as the default by Keras in the Adam optimizer class. We began with a learning rate of 0.001, but also used a learning rate reducer when the training hit a plateau, with a minimum learning rate of 0.0001 and a factor of $\frac{1}{2}$.
For the Siamese model, we generated around 200,000 positive/negative pairs for training and 20,000 for testing.

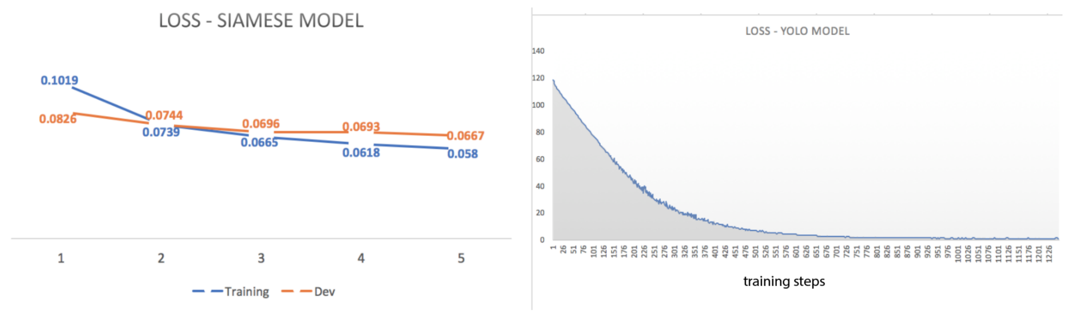


Figure 7: Loss of our Siamese model while training, after 5 epochs (left) and loss of our YOLO model while training, after 4 epochs/1250 steps (right)

We used contrastive loss for the loss function as defined:

$$(1-Y)\frac{1}{2}(D_w)^2 + Y\frac{1}{2}\{max(0, \alpha - D_w)\}^2$$

where D_w is the Euclidean distance between the encodings of the examples and α is a margin value set as a hyperparameter. For α, we used 0.5, and we used RMSProp for the optimizer (defined within Adam, below). The architecture between each network of the Siamese model and EMNIST-trained CNN was otherwise identical.

5.4 Human Detection

We implemented human detection from aerial images by using Darkflow [6]to train the tiny-yolo-voc model from Joseph Redmon's Darknet website. This model was trained on one class for the aerial view of the people. Although the YOLOv3 model was able to detect some humans in aerial photography (Figure 3), it is only able to do so if the human is recognizable. Thus, YOLO was not able to recognize humans in aerial photographs if the humans take up a very small portion of the picture (Figure 4). After training YOLO on the Trajectory Forecasting Benchmark pictures [18] with the default hyperparameters, we were successfully able to detect humans on the training dataset with good accuracy (Figure 5). However, it overfitted to the training set since it had little to no accuracy on the test set, and we are actively researching on ways to improve this model.

6 Discussion

We chose many of the hyperparameters we did based on work people have done in the past. For instance, the hyperparameters we chose for YOLO were identical to the ones chosen in Darknet's Tiny-YOLO model, we only made small tweaks to the hyperparameters chosen in the alphanumeric model from ones people have chosen to train using the MNIST (numbers) dataset, because the inputs were so similar. We did train two YOLO models for the standard objects; one on 10 classes and one that lumped all 10 shapes into one 'shape' class, and we think it's interesting to note that the loss started and ended at around the same time and it took around the same amount of steps to converge, though the 10-class model did take longer per step (but not with linear correlation, only about 1.5 times as long). The YOLO model performed well with localizing shapes - 91.8% dev set accuracy - but performed poorly when it came to classifying shapes due to the loss function hyperparameters (we could only choose to be good at one thing, and since shape detection can be bettered using an additional CNN, we chose better localization performance). For alphanumerics, the model that performed best in practical context with vanilla alphanumeric classification was unsurprisingly the CNN trained on the EMNIST dataset, but the model that had the highest accuracy in training and testing was the CNN trained with our font-generated dataset. We'd also like to note that while the accuracy of our production model only ended up being 86.7%, the Bayes error for the EMNIST dataset can be estimated at around 95%, and with our data augmentation (like rotation), probably more like 90%. While the simple, fully-connected neural network trained much faster, it didn't perform as well in the end. We didn't really need to worry about overfitting due to our use of dropout and low number of epochs.

7 Conclusion/Future Work

With the preliminary results we've seen with the Siamese model, we feel it will be worth exploring (even if this model does perfectly fine in practice). If we had more time, we'd also implement a separate neural network for classifying the shape, because sometimes YOLO confuses certain shapes with others even if it correctly guesses the bounding boxes for shapes. If we had a great deal more time, we'd tackle tougher problems like search-and-rescue operation detection, infrastructure assessment using 3-D internal models and capsule networks, etc...but for now, we're pleased with the results we've gotten with the challenges we chose.

8 Contributions

8.1 Josh Payne

- Built standardized object dataset generator for YOLO algorithm
- Built initial simple NNs for alphanumeric/shape classification (Keras, Tensorflow)
- Built data-augmenting CNN for alphanumerics (Keras)
- Built Siamese CNN model (Keras)
- Implemented YOLO Model (Darkflow) for shape detection and classification
- Built k-means segmenter/color classifier/image formatter
- Wrote paper

8.2 Peggy Wang

- Built image detection and localization system using SURF algorithm
- Built background cropping and random overlay image tools for dataset generator
- Created project poster
- Implemented YOLO Model (Darkflow) for emergent object detection
- Found and annotated aerial views dataset, and wrote script to convert BBox-Label txt annotations to Pascal VOC XML annotations

References

[1] Abadi, Martín, et al. "TensorFlow: A System for Large-Scale Machine Learning." OSDI. Vol. 16. 2016.

[2] AUVSI-SUAS 2018 Competition Rules: http://www.auvsi-suas.org/static/competitions/2018/auvsi_suas-2018-rules.pdf.

[3] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." European conference on computer vision. Springer, Berlin, Heidelberg, 2006.

[4] BBox-Label-Tool: https://github.com/puzzledqs/BBox-Label-Tool

[5] Cohen, Gregory, et al. "EMNIST: Extending MNIST to handwritten letters." Neural Networks (IJCNN), 2017 International Joint Conference on. IEEE, 2017.

[6] Darkflow: a Tensorflow implementation for YOLO: https://github.com/thtrieu/darkflow.

[7] Hartigan, John A., and Manchek A. Wong. "Algorithm AS 136: A k-means clustering algorithm." Journal of the Royal Statistical Society. Series C (Applied Statistics) 28.1 (1979): 100-108.

[8] Keras: The Python Deep Learning library.

[9] Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." ICML Deep Learning Workshop. Vol. 2. 2015.

[10] Maaten, Laurens van der, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.Nov (2008): 2579-2605.

[11] OpenCV: the Open Source Computer Vision Library.

[12] Pandas: the Python Data Analysis Library.

[13] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.

[14] PIL: Python Imaging Library.

[15] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger. CoRR abs/1612.08242 (2016)." (2016).

[16] Siamese 2-D encoding visualization implementation: https://github.com/ywpkwon/siamese_tf_mnist

[17] Stanford Drone Dataset: http://cvgl.stanford.edu/projects/uav_data/

[18] Trajectory Forecasting Benchmark: http://trajnet.stanford.edu/