Automatic Isolation and Identification of Indus Valley Script Graphemes in Curated Images

Brad Costa
bcosta2017@my.fit.edu

Peter Thomas
pthomas2019@my.fit.edu

Abstract

Dating to as far back as 6000 B.C.E., the Indus Valley Civilization left behind a trove of artifacts that have given us a captivating puzzle to solve. Arguably chief among these is the decipherment of the Indus Valley Script. Being able to automatically identify, and eventually decipher Indus Valley Script graphemes in a lab or field setting would be of great benefit to scholars who study this fascinating slice of our human heritage. In this paper we present an investigation into the use of existing pretrained models to automatically locate and identify Indus Valley Script graphemes in curated images of both seals and stamps. We present a brief survey of applicable approaches to similar problems, preliminary results evaluating the "Detecto" Faster R-CNN and YOLOv3, selection of, and experiments with the YOLOv3 model, and finally we present the results, and discuss avenues for further experimentation.

Introduction

The Indus Valley Script (IVS) [1] remains a fascinating and enigmatic piece of Indus Valley Civilization (A.K.A. the Harappan Civilization) [2]. Dating to c. 6000 B.C.E. this society of the Indus River valley left tantalizing clues about their lives and culture. Artifacts range from the mundane to the unique. Once thought to be exceptionally peaceful and egalitarian, scholars are challenging many aspects of earlier held archeological conclusions [3]. Although many decipherments of the IVS have been put forth, there remains no scholarly consensus as to the meaning behind the graphemes that are known to make up the script [4]. Being able to quickly decipher the hundreds of seals and stamps this civilization left would go a long way towards our understanding of their culture. To help facilitate the eventual decipherment and the application of the resultant new understandings of the IVS and the culture that spawned it, an automated system that can quickly and reliably identify IVS graphemes would greatly aid both desk-bound research and, it is our hope eventually, in-situ field work.

Literature Survey

We conducted a search of literature focusing on combinations of terms such as Indus Valley Script, Harappan, OCR, and grapheme. Approaches developed by Google researchers have shown encouraging results for the detection of characters in natural scenes [5] [6]. However, these approaches did not resonate as completely applicable as *DistBelief* is built on massive clusters of CPU cores running training against millions of images pulled from Google's immense Street View databases. We also reviewed approaches to detection and translation of Egyptian hieroglyphic characters [7] [8]. Both approaches rely on the hieroglyphs being divided in a rigid columnar structure and therefore would most likely not translate well to mostly linear structure of IVS stamps and seals. Of particular interest for future investigation is the use of attention mechanisms in similar problems to first identify regions in images where text is most likely present and then feed that information into additional layers [9] [10] [11]. An additional avenue for future research is to utilize novel methods to draw bounding boxes around detected graphemes [12] [13]. Researchers have also applied Markov models to analyze the structure of the IVS with

interesting results [14]. Ultimately, we settled on an initial investigation of the Detecto and YOLOv3 models running on the PyTorch and TensorFlow frameworks respectively [15] [16] [17] [18]. YOLOv3 showed better initial results and was selected for further experimentation.

Images

Source images of seals and stamps were taken from several sources. Primarily the *Indus Script Dictionary* [19] and with permission from Harappa.com [20]. Additional some images were pulled from National Geographic and ISTC websites [21] [22]. All together we were able to utilize 232 source images. To further augment images for training and validation we developed a configurable script that applies various transformations to base images (See §Data Augmentation). Bounding box data with labels was then created in two sets. The first set labeled all characters in the images as the single class "grapheme." The second set labeled each grapheme with its corresponding Harappan character code based on Everson's proposed Universal Multiple-Octet Coded Character Set [23]. Each Harappan character code takes the form HXXX, where XXX is the three-digit sequential character number. Those characters that did not appear in the abridged set (abridged in relation to all known IVS graphemes) were labeled as "grapheme" so they could be included or excluded based on class-name filtering at the time of training. For training purposes, six images were chosen at random from the set of those images that did not contain a single appearance, in our overall set, of a given grapheme.

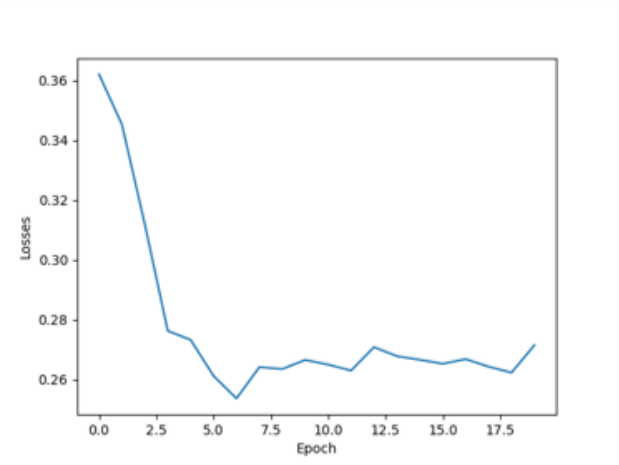
Model Selection

Prior to sourcing the 232 images we evaluated both the Detect-o and YOLOv3 models on an initially limited set of 26 images, 22 for training, and 4 for validation. The Detecto model was trained 2000 iterations over 20 epochs while YOLOv3 was trained with 2500 iterations over 10 epochs. Results are shown in . YOLOv3 was selected due to its slightly better results and for the ease of code implementation and execution.

Data Augmentation

While CNNs have impressive representational power in computer vision, they require vast amounts of training data to generalize well to real-world examples. This can be problematic for many object detection problems, as there is a large cost associated with sourcing large sets of annotated images. This is especially true with the Harappa grapheme dataset. There are only a limited number of sources to draw images of Harappan symbols from (mostly seals and tablets stored by museums). The total number of objects displaying Harappan graphemes around the world is less than ~5000, well below the tens of thousands of images that comprise typical landmark object detection datasets such as COCO and Pascal VOC [24] [25]. In addition, finding annotators who are qualified to accurately label the data is difficult. Unlike with PASCAL VOC and COCO, which contain mostly common place objects, there are only a few qualified experts around the world who have the knowledge and expertise needed to identify these symbols. Even for a simple detection task that does not seek to positively identify each character, it can still be difficult for an amateur annotator to differentiate what portion of the seal forms the grapheme and what is an engraving or damage. This makes data annotations for the Harrapan dataset prone to error, and noise that we can be ill-afforded with such a small sample of images to train from.

Data augmentation is a technique in machine learning for artificially expanding the training set so that it can better represent the distribution of real-world data the model has yet to see. With data augmentation, already collected examples in the training set are modified with some augmentation policy to subtly alter the content of the original image. These types of augmentations typically come in two flavors. One is altering the pixel content of the image by changing the saturation, bloom, or color of



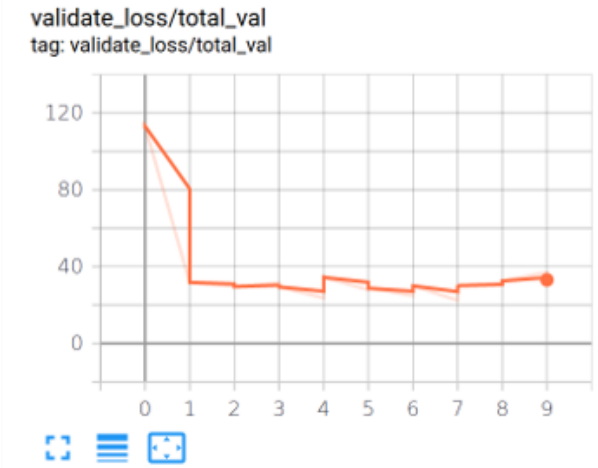


Figure 1: Detecto and YOLOv3 Initial Results

the original image, as well as applying noise filters that change the pixel values of the image with some predefined random distribution, typically, Gaussian. The other type of augmentation category is affine transformation, such as image rotation, translation, scaling, cropping, and flipping the image around an axis [26]. With image classification tasks, where the labels are not tied to any position in the original image, automatic data augmentation is simple. The image can be altered without fear of also changing the intended label. With object detection tasks, however, augmentation is more difficult. Applying affine transformations to the image without also changing the bounding box labels accordingly will result in an unusable image.

It is possible to use hand-cranked methods to apply data augmentation, such as using a human labeler to apply the same set of bounding boxes in a rotated image. However, this does not scale well to larger applications. It became evident early on that hand-labeled data augmentation would be insufficient to generate the large set of training needed to represent the space of all image configurations in our Harappan grapheme dataset.

To handle the expanded set of hand labeled Harappan images as well as to scale for any additional examples that may be added to the dataset later, a data generator that applies pixel and affine transformations automatically was developed. The data generator can accept any number of images at any size. In addition, it is capable of handling both color and grayscale imagery. During training, images are read from disk as needed by the data generator. The generator then automatically applies a set of randomized augmentation policies to the image. Because the augmentations and the necessary label adjustments are generated automatically, there is no limit to the number of training instances that can be produced by the generator. However, there is a limit to the number of representations that are realizable using the transforms listed here, and one must take care that they do not generate a series of images that are too like each other while using the generator. The full set of augmentation policies used by the data generator are listed below with examples provided in ***Error! Reference source not found.*** *and*

Figure 3.

Pixel Value Transforms

1. Noise Filters
2. Saturation
3. Bloom
4. Contrast

Affine Transformations

1. Rotation (-45 ˚ - 45˚)
2. Scaling
3. Translation (Up to $\frac{1}{8}$ of either x or y-dim of image)

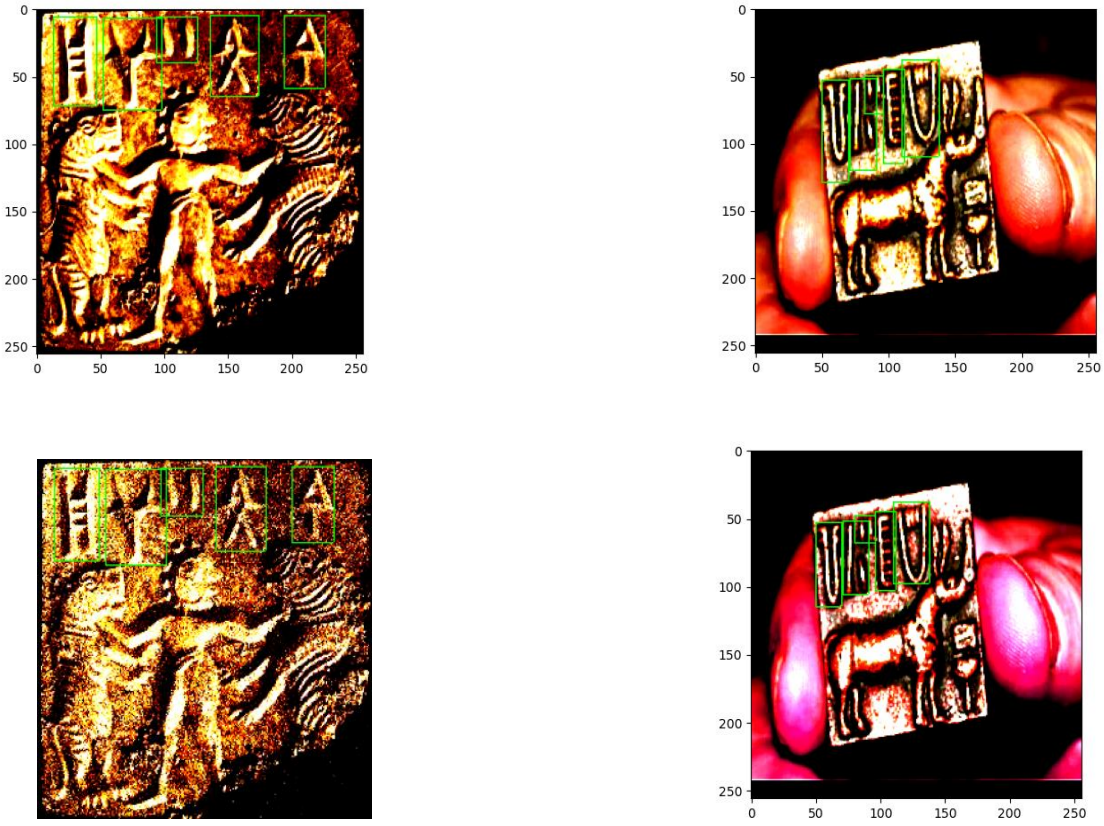


Figure 2: Pixel Augmentations

The bottom row has pixel-level transforms on them, while the top row is the untransformed original image. The bottom image on the left had a Gaussian noise filter applied, while the right had a series of different saturation, bloom and coloring effects applied.

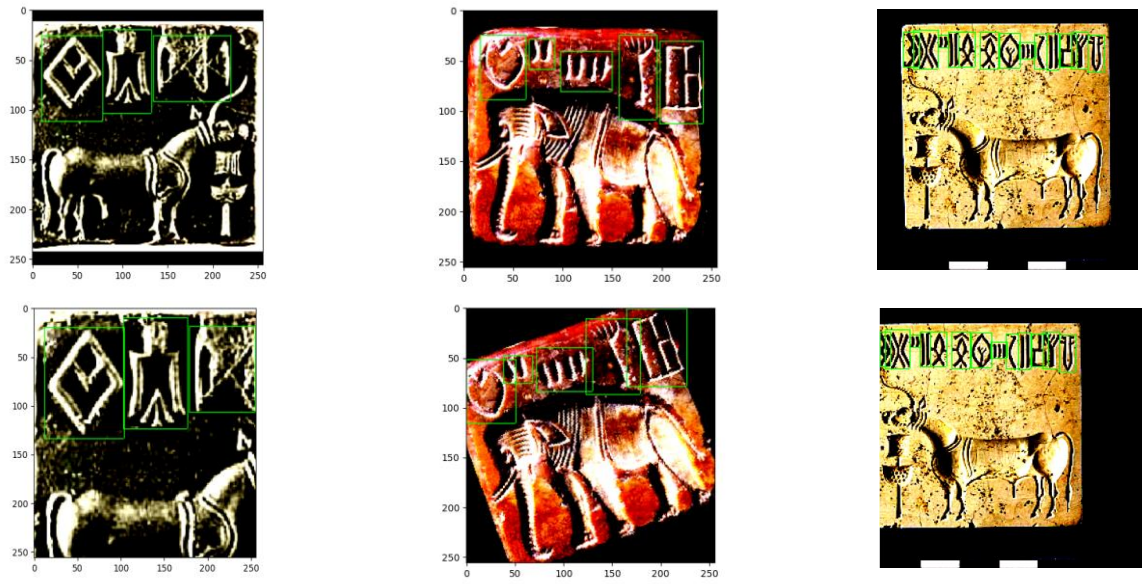


Figure 3: Affine Transformations

As with **Error! Reference source not found.**, the top row consists of unaltered images, while the bottom images have selected affine transformation applied to them. The left column represents a scaling transform, the middle a rotation transform, and the right a translation transform.

While developing the data and augmentation generator, efforts had to be made to ensure that any augmentation that was applied was consistent with the domain of Harappa graphemes that are likely to be encountered in the real world. This limited the scope of augmentation techniques that could be applied. In particular, the orientation of the symbols themselves are intrinsic to their meaning. Mirror reflections of symbols, therefore, are ruled out, as are extreme rotation transforms (e.g., a 90° transform). The rotation transform also has the potential of rotating graphemes outside of the dimensions of the original image, making those bounding box labels useless. A function that removed bounding boxes that bled out of the image by more than a third of their area was used to remove labels in this category.

In addition, model constraints were also a factor in how augmentations were applied. For example, when a rotation transform is applied to an image, ideally one would also apply the same transform to the bounding boxes so that they information content of the bounding box matched that of the original image. However, the model we used to perform the object detection task, YOLOv3, uses a set of anchor boxes to inform predictions that are generated under the assumption that the image they are being applied to is not rotated. Therefore, the hypothesis generated by YOLOv3 is unable to represent rotated bounding boxes.

We dealt with this by keeping the bounding boxes parallel with the y-axis of the original image. To do so, the minimum and maximum values for both the x and y dimension of the box were chosen to generate a new, enlarged bounding box that was able to cover all the original grapheme. This is not ideal, as the resultant bounding boxes also cover other features in the image that do not relate to the grapheme. In addition, the enlarged bounding boxes that were generated in this fashion tended to overlap with each other, sometimes to a great degree. It was feared that this might induce some level of confusion in the model. The representations it learned might overlap multiple graphemes, resulting in single detections covering multiple graphemes instead of keeping them separate, which was desired. The decision was made to apply non-max suppression to rotated bounding boxes based on Intersection-over-Union with their neighboring bounding boxes, removing bounding boxes that overlapped too much with each other. This solution is also non-ideal, as some of the symbols in the rotated image will not have an associated bounding box attached to them during training. Future research could explore this problem by using or creating a model that is effectively able to represent bounding boxes in a rotated space.

Precision and Recall

While we initially utilized validation loss in our initial model selection, validation loss cannot be used as a metric to quantify the model's performance at a given detection task. The validation loss provides some information into how the model improves in performance as it is exposed to more images during training, it cannot be used to compare the performance of the model at the given detection task with other approaches. The loss function used by the model is inherent to the model itself. This loss function will differ with a model's architecture, as different models will optimize different parameters to perform the same task. Comparing a model's validation loss with another is, therefore, unproductive. We need another set of metrics to provide a meaningful measure of the model's performance at the detection task. This set of metrics is the precision and recall of the model.

Precision is the ratio of true positives that the model generates over the entire set of positive predictions provided by the model. Positives correspond to bounding boxes output by the model that are deemed with some degree of confidence to contain an object of interest (OoI). True positives are positive predictions that correspond with a real OoI in the image.

False positives are positive predictions that do not contain a real object of interest in them. The ratio of true positives to the total number of positives generated by the model provides a measure of "trust" in the predictions made by the model. A high ratio, or precision, means that most of the predictions made by the model are accurate. Conversely, a low precision means that the model generates false predictions at a high frequency relative to true predictions.

Recall is a closely related metric that measures the model's ability to detect all the real OoI's in the image. Recall is defined as the ratio of true positives over the sum of true positives and false negatives generated by the model. Negatives are bounding boxes that fall below the confidence threshold required to rate it a positive. They are therefore deemed to not contain an OoI. False negatives are therefore the bounding boxes that are deemed to be negative that contain an OoI in them. A model with high recall will therefore detect most of the real OoI's in the image with a high probability. Conversely, a model with low recall will detect few of the real OoI's in the image.

$$Precision = \frac{T_p}{T_p + F_p} \quad (1)$$

$$Recall = \frac{T_p}{T_p + F_n} \quad (2)$$

where:

T_p = *True Positives*
F_p = *False Positives*
F_n = *False Negatives*

At the heart of both metrics is the determination of what predictions the model deems to be positive or negative. This determination is controlled by the confidence threshold set by the user. For each prediction made by the model, an associated confidence score is assigned to the box that corresponds to the probability that the bounding box contains an actual OoI based on the probability distribution learned by the model. Predictions with confidence scores below the threshold are determined to be negatives, while those above the thresholds are deemed to be positives. Setting a high confidence threshold will therefore reduce the number of positive predictions made by the model, increasing the probability that an OoI will not be detected. Lowering the threshold increases the chance that all OoI's are detected, but at the cost of inducing false detections that contain nothing of interest.

We desire a model that provides both a high precision and high recall. However, a model's performance in one metric is usually inversely proportional to its performance in the other. As the model's recall improves, the precision will tend to degrade, and vice-versa. As both metrics are functions of the confidence threshold, we can increase one or the other by raising or lowering the confidence threshold.

Unlike other OCR tasks, the background of a typical sequence of IVS graphemes can contain other features like those exhibited by the symbols that we want to detect, such as animal or human pictographs. This complicates the recognition task, as there is a possibility that the features that the model learns are also common to the other etchings on the seal or stamp, inducing a degree of confusion that increases the chance that the model will output a false positive. It is important that we can quantify the tendency of the model to produce false positives during detection.

YOLOv3

YOLOv3 is an object detection model that uses a convolutional neural network feature extractor with a fully connected layer prediction head at the end that output scaling factors for anchors to generate bounding boxes, object confidence values, and class scores. What distinguishes this model from its previous iterations (YOLOv1 and v2) is the Darknet feature extractor that forms the front end of the model [16]. Darknet has 53 convolutional layers and a fully connected layer at the end.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4: Darknet-53 Layers

While Darknet's classification performance tends to be lower than other state of the art feature extractors such as Resnet-101 and Resnet-152 [16], it still performs well on most classification metrics. More importantly, it contains fewer layers and, thus, fewer parameters than either Resnet-101 or -152. This makes it better suited for deployment as the backbone of an object detection model being used on an embedded system or other environment with a constrained memory footprint, such as a mobile phone. As the envisioned end goal of this project is to deploy an inference model to a mobile device (perhaps with a symbol classifier on the backend that accepts the cropped bounding boxes of the inference model and outputs classifications), the end state deployment environment was a factor that needed to be considered when choosing our model.

Yolov3 is like other state of the art object detection models in that, instead of generating new bounding boxes for images, it uses a set of prior boxes generated before training begins. The output of the model adjusts those priors to generate its prediction bounding boxes. This simplifies the detection task. Instead of having to localize and generate a bounding box of appropriate size to fit the object detection, the network must learn to localize the OoI and apply regressions to the provided anchor boxes. To generate these anchors, YOLOv3 divides input imagery into a grid of equally sized cells at three different scales. Taking the bounding boxes from the training set, a K-means clustering algorithm is used to form clusters of truth boxes based on their dimensions. The three different scales of anchor size are based on the clusters generated by the K-means algorithm.

The object confidence values are a measure of the probability that a bounding box contains an object of interest based on the underlying probability distribution learned by the model. The confidence threshold at which a bounding box is said to be a "positive" (i.e., a detection by the model) is a value set by the user when post processing the predictions of the model. If we set the confidence threshold low, we'll increase the likelihood that all of the objects of interest in the image will be detected. However, a low confidence threshold also induces a risk that we will accept bounding boxes that do not contain an OoI in them (or a "false positive"). Conversely, if we raise the confidence threshold, we will the probability of generating false positives goes down, but we lower the probability that we will detect all the objects in the image. This is a tradeoff that the model implementer and end user need to consider when using the trained model during inference.

Yolov3 has faster training times for models of similar detection performance. The model divides input images into different regions and outputs objectness scores to each of those regions with predicted regressions and classes. Non-max suppression is applied to the set of bounding boxes and predictions that have higher IOU are removed based on which prediction has the lower objectness score.

Unlike other benchmark object detection models like Faster-RCNN, YOLOv3 does not use any region proposal network to make its predictions. Each input only makes a single forward propagation through the network. Because there is only one pass, YOLOv3 trains faster than other models of comparable detection performance. Class predictions for the model are made with independent logistic classifiers.

Training

YOLOv3 was trained for 40 epochs with 232 training images augmented to 5000 with the active augmentation chain. An initial learning rate of 1e-4 was used. During training, this learning rate was slowly tapered off to a final learning rate of 1e-6. This was so that, as the model approached convergence, we could slow the training of the model so that an optimum local maxima could be identified. Validation was performed on 13 images. This set of imagery was not augmented to provide a good baseline of the model's performance on the dataset as-is. Precision and recall curves were generated for the validation set of imagery at each epoch of training to give a good indication of the progress of the model's performance at the object detection task.

Three different model configurations were trained. First, the model was trained on a set of weights initialized with a random normal distribution centered around 0.0 with a standard deviation of 0.01. The input data to this model was the set of augmented training images with three color channels. This model was the baseline configuration (henceforth referred to as the "baseline" model). Performance in this configuration should give a good idea of YOLOv3's suitability for the problem. Second, a model was trained with the input imagery converted to grayscale. This was to lower the dimensionality of the input space and, therefore, reduce the number of features that the model needed to learn. It was hypothesized that the most salient features that need to be learned for grapheme detection would be contained within the spatial dimensions of the image rather than the color channels. Humans identify different graphemes based on their shape, not their color. Furthermore, the identity and information conveyed by graphemes in the image do not change when the image is rendered in grayscale. Reducing the complexity of the detection problem by removing the color channels could outweigh the information lost from doing so. To test this hypothesis, we trained a model using grayscale imagery. All other training hyperparameters were identical to those used with the color model described above.

Finally, the model was trained with a set of weights provided by the model developer that had been pre-trained on the COCO dataset. The COCO dataset consists of over 200,000 labeled images that display objects from 80 different object categories. Most of these categories are everyday objects such as vehicles, animals, and household items. This method of transfer learning can help improve the performance of object detection models on small datasets such as the IVC dataset. The idea is that the model learns general features of objects of interest in imagery that can help the model even when identifying object classes that it has never seen before. This can improve the performance of the model over a model that has to learn all features needed to identify objects of interest from scratch. It should be noted that there is little cross-over between the domain of IVC graphemes and the objects of interest in the COCO dataset. Using the pre-trained COCO weights did increase the object detection performance of our model, but it is possible that there are other datasets that could be used that exhibit features closer to the ones that we wish to identify in the IVC dataset. One possible candidate is the Street View House Numbers (SVHN) dataset, a landmark dataset that formed the foundation of early research in the use of Convolutional Neural Networks in Optical Character Recognition. The difference in dimensionality between the IVC and SVHN datasets would have to be bridged in some way (SVHN images are only 32x32, while the

average image in the IVC dataset is 400x400), but, if we could overcome the gap in input-space dimensions, the features learned training on the SVHN image set may transfer more effectively to the IVC detection problem. In any case, the model that was trained with the COCO pre-trained weights performed significantly better than the other two configurations, demonstrating that transfer learning will be a critical component of developing an IVC detector that is able to recognize graphemes with the precision needed for researchers in the field. This is even more important if we wish to train the model to not only recognize the broad class of IVC graphemes, but also the individual characters themselves.

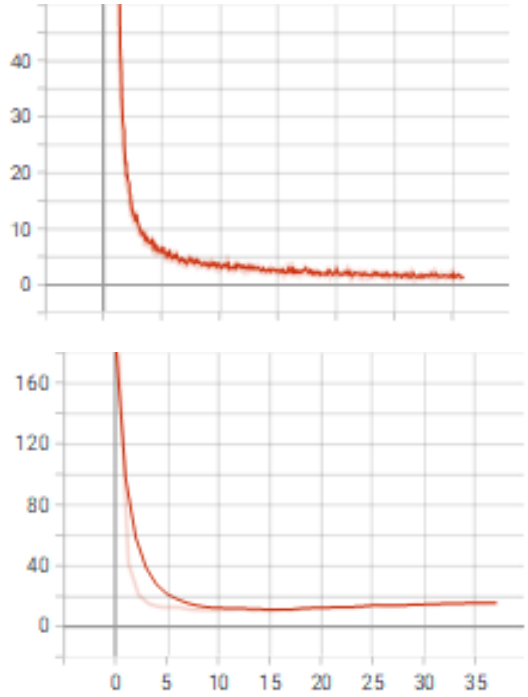


Figure 5: Training (left) and Validation (right) losses for the baseline model.

Note that training had reached convergence at around epoch 20. This was consistent with the observed training behavior in Detecto as well. It is unlikely that there is any benefit for training greater than 25 epochs (though this might change should more labeled data become available).

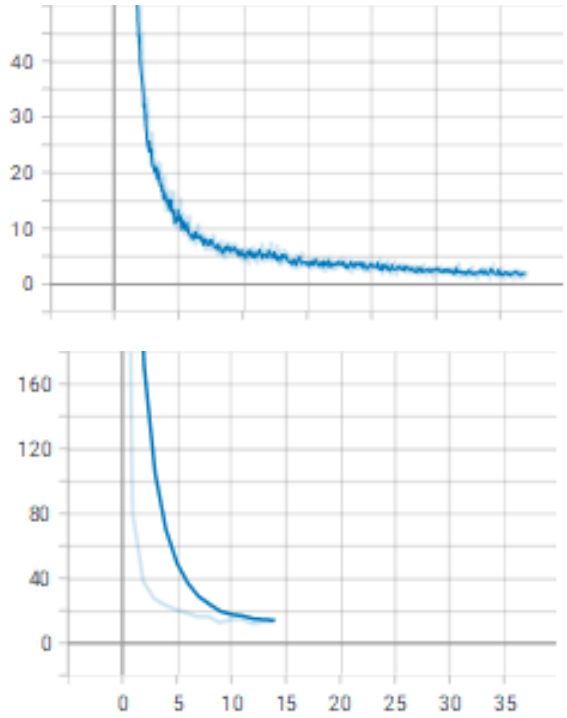


Figure 6: Training (left) and Validation (right) losses for the Grayscale model.

It was hard to track the progress of the grayscale model during validation, as the loss function began to report NaN values halfway through training. We have not found a definitive reason for why this might be the case. However, the convolutional layers of the detection head for YOLOv3 use leak-Rectified Linear Unit (ReLU) activation functions, which are prone to cause NaN values if untuned. This is because the gradient for a ReLU function is always one for inputs greater than zero. Depending on the kernel weights for the filters of the convolutional layers, this can cause the backpropagated gradient to "explode" (I.e., exponentially trend to infinity or negative infinity), resulting in "dead weights" in the network that prevent the model from improving its learned representation any further. It is possible that the model was tuned to work specifically with colored imagery and given the same set of hyperparameters for grayscale imagery, will result in early onset of dead weights. Lowering the learning rate by a factor of 10 (so to 1e-5) should help rectify this problem.

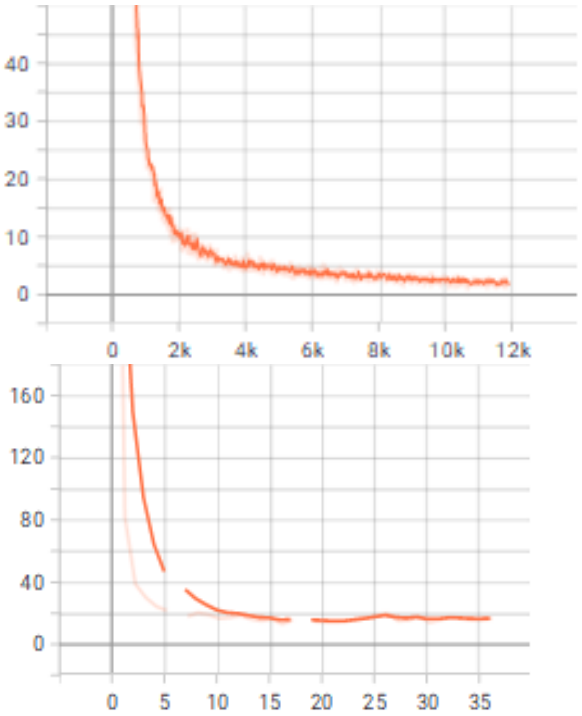


Figure 7: Training (left) and Validation (right) for model trained with COCO weights.

The model with pretrained weights takes longer to reach convergence than the baseline model. This is possibly because it was still extracting salient features from the imagery where the baseline model was unable to do so.

Results

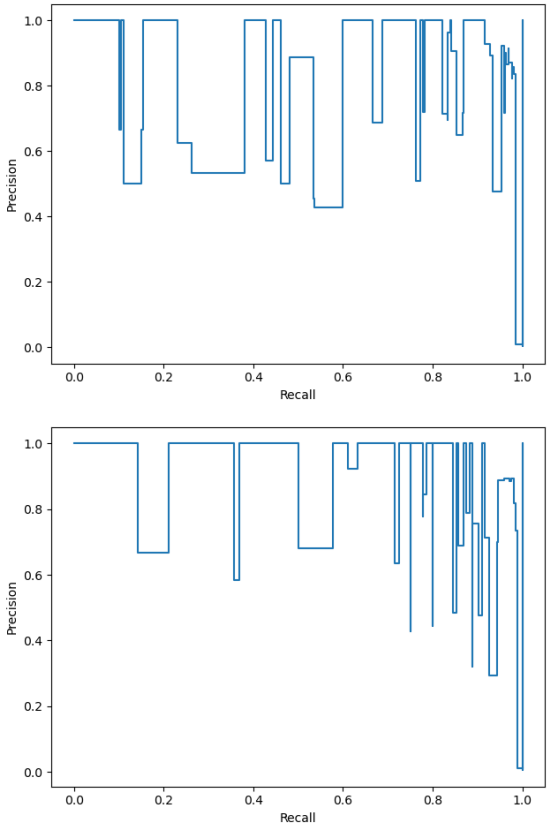


Figure 8: Precision-Recall curves for baseline model (left) and grayscale model (right) for best epoch of training (epoch 22 for baseline model and epoch 20 for grayscale).

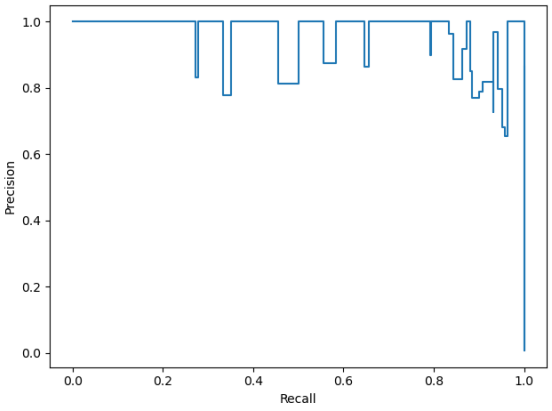


Figure 9: Precision-Recall curve for model pretrained with COCO for best epoch of training (epoch 20)

To produce the precision and recall curves for each model, the precision and recall for each validation image were calculated over a range

of confidence values from a set of 11 equally spaced intervals in the range [0.0, 1.0]. These calculations not only generated precision and recall for each model, but also the number of true positives, false positives, and false negatives that were generated. Based on the precision and recall curves generated for each model, the best epoch for training could be determined based on the area under the curve. The greater the area under the curve, the higher the precision of the model over all values of recall and, thus, the better the model was at keeping the number of generated false positives low while simultaneously detecting all the truth objects in the image.

It should be noted that, due to the lack of data available for the validation portion of model training, it was impossible to generate continuous precision-recall curves. This makes it difficult to definitively determine which model performed the best based on precision and recall alone. The outliers in the validation set could skew the precision and recall in a fashion that is not representative of the distribution of the real-world data. Nevertheless, general trends could still be determined from the precision and recall displayed by the models during all epochs of training that allowed us to determine which model performed the best at the task.

The best precision and recall curves were selected from each model during training based on area under the curve to determine which epoch demonstrated the model's best performance. It was found that the best epoch for the COCO model and the grayscale model was epoch 20, while, for the color model, the best epoch was 22. The precision and recall curves for those epochs of training are displayed in Figure 8 and Figure 9.

Based on the statistics charted by these curves, the COCO model had the best performance on the object detection task. It had a consistently high precision over all values of recall, which can be seen in its PR-curve (Figure 9). The precision for the model only began to decrease after a recall value of 0.8. This can be interpreted as "the model will produce mostly true detections while detecting, on average, 80% of the objects in the image". If we want to increase the detection probability by lowering the confidence threshold, we risk generating false positives after this point. The detection performance of the COCO model can be attributed to the domain knowledge that it had learned previously when training on COCO, which seems to transfer over to the domain of IVC graphemes well. As noted earlier, this would indicate that transfer learning is probably the most lucrative avenue to pursue when improve performance on the detection task.

Another result of note is that the grayscale model seemed to outperform the baseline model. Though both models had, on average, a lower precision than the COCO model (I.e., were generating more false positives during validation), the grayscale model maintained a higher precision over a greater range of recall values than the color model. This was true even after the validation losses of the grayscale model began to produce NaN values, showing that, even with the dead weights in the gray scale model, the model's representation of the detection task was still more precise with the color model. This should be improved with hyperparameter tuning to prevent the spread of dead weights in the grayscale model. This seems to be clear evidence that there are performance gains to be had in lowering the input dimensions of the problem by removing the color channels. Based on this and the performance of the model trained with COCO weights, an approach that could be tried to further improve the detection performance of the model is to perform transfer learning on a dataset closer to the domain of IVC graphemes (potentially the SVHN dataset mentioned earlier) and cast the instances of that dataset into grayscale as well.

Comparison with Detecto Approach

For the final comparison between our approach with YOLOv3 with the actively generated augmentation chain and last semester's

approach using Detector (Faster-RCNN with Resnet 50 feature extractor), we trained the Detecto model for 20 epochs on our expanded training dataset as well as the augmented data that was in the previous semester's dataset (we were unable to apply our augmentation chain to Detecto training, as its model architecture is implemented in PyTorch while ours is in TensorFlow) and collected metrics on the 13 images in the expanded validation set. The validation set contained 73 ground truth bounding boxes in total. Each model's ability to output a prediction that corresponding with each of these boxes (true positives) was measured as well as any boxes the model produced that did not match a corresponding truth object in the image (false positives). The confidence score and the IOU threshold for both models was set at the same value, 0.5. We found that Detecto outperformed our model in all metrics of object detection. Furthermore, it seems that Detecto, even without the active augmented training data, was able to generate accurate predictions for data that it had never seen before (i.e., unlabeled data that we collected off the internet).

	Truth Object	False Object
Detected	108	0
Undetected	*0*	*N/A*

Table 1: Detecto's Detections vs Truth Boxes.

The rows correspond to whether the model detected an object, while the columns correspond to whether the detection corresponded with a truth object or not. The top left cell is the number of True Positives, the top right cell is the number of false negatives, and the bottom left cell is false positives (the bottom right cell, true negatives, cannot be measured in an object detection model, as all portions of the image that do not contain a prediction would be counted as a false negative). Impressively, Detecto was able to generate predictions for all of the truth boxes in the validation set. In addition, it was able to do so without generating any false positives.

It is worth noting that, in Detecto's and YOLO's metrics, the number of true positives will be higher than the true number of objects in the images because overlapping detections will both be treated as true or false positives by the metrics module. A simple fix is to not only compare IOUs between the truth boxes, but to also compare IOUs between other predictions so that those predictions can be removed before metrics values are calculated, perhaps with non-max suppression.

This is not entirely unexpected. Detecto's Resnet feature extractor has more parameters than Darknet and is capable of learning more complex representations of the problem domain. In addition, Faster R-CNN splits the detection problem into two components: 1. Region Proposal Generation to localize locations of interest in the image, and 2. Object Classification and Regression to generate more accurate predictions for the identified object class [34]. What was surprising was that Detecto was able to outperform our model even with the actively augmented training data. There are two reasons for why this might be the case. One is, as mentioned earlier, a more accurate representation of the problem domain can be learned by the Faster R-CNN model. The other is that, since the augmented data used by the Detecto group was hand labeled, a higher proportion of labels were conserved after the affine transformation was applied and the labels that were preserved were of a higher fidelity than those that were automatically generated. We would still recommend (admittedly, with some bias) using our data augmentation approach going forward. It is easily scalable and can handle new data instances more effectively than the hand labeled approach. In addition, it is easier to generate new affine transformations and apply them to the data pipeline with our approach. However, there is room for improvement. Methods for applying higher fidelity labels could be investigated. One method that could be explored is using a different bounding box coordinate scheme (possibly using an additional bounding box parameter, Θ, which measures

the angle of the center line of the bounding box from the y-axis). Most of the standard object detection models in use today cannot be used to learn representations using this bounding box coordinate scheme. Many of the standard models (YOLO, Faster R-CNN, and Single Shot Detector (SSD)) use anchor boxes like the ones mentioned during our description of YOLOv3. While these anchor boxes help the model perform the object detection task, it also restricts the possible representations the model can learn. One approach that could be tried is to use a convolutional neural network architecture to output regression values for the rotated bounding box. In practice, this approach tends to result to inaccurate results due to the sensitivity of the angle prediction.

the showing that further research into using the Faster-RCNN architecture as a grapheme-detector may be worth looking into. This research would have to also examine the tradeoffs for the longer training times for Faster-RCNN as well as the larger number of parameters that would need to be stored in memory when performing inference with the model, an important factor to consider when deploying the model to a constrained memory environment.

	Truth Object	False Object
Detected	305	26
Undetected	21	N/A

Table 2: YOLOv3's Detections vs Truth Boxes.

In contrast to Detecto, YOLOv3 generates many predictions at a much lower confidence, resulting in many overlapping true positives. In addition, YOLOv3 is more prone to generating false positives and misses more of the truth boxes than Detecto. The conclusion is that a Faster R-CNN architecture would provide both a higher precision and recall for the detection problem than YOLOv3. Adding a Faster R-CNN architecture from the TensorFlow model API could be a next step.

Symbol Class Results

After statistics had been gathered for the object detection models, an additional YOLOv3 model was trained on the set of images that had been labeled with the symbol's classes given by Everson [23]. In total, there were 185 symbol classes that covered most of the graphemes in the image (those that were not assigned a specific symbol class were assigned to the general class "grapheme"). In total, there were 1265 different instances in the training set and 34 instances in the validation set. Most of the symbol classes in the data consisted of a singleton, and only a handful had more than ten instances. Q naïve approach was attempted by our model which tried to learn on the whole set of 185 symbol classes to provide a baseline for future detection and classification efforts. A more refined approach in the future could select only those symbols that appear a statistically significant number of times in the dataset. Despite these limitations, those class instances that did show appear in the dataset were identified with YOLOv3 with a surprisingly high precision. The exception to this is "grapheme", as this was an overarching category encompassing many different symbols exhibiting a range of features that overlapped highly with other symbol categories.

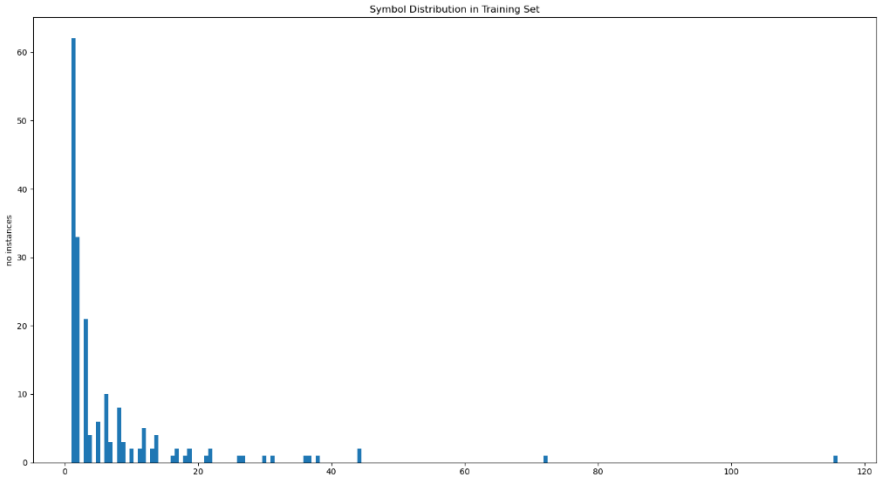


Figure 10: Distribution of symbol instances in the dataset.

Most symbol instances appeared only once in the training dataset. Augmentation helps this somewhat, but not to a degree necessary to recognize low frequency data with any kind of precision. The class instances that appeared most frequently in the data, most notably 'H311' (116 instances in the dataset), 'H127' (72 instances in the dataset) are labeled with relatively high precision.

Symbol ID	*Appearances in the Training Dataset*
H311	116
H127	72
"grapheme"	44
H060	44
H148	38
H376	37
H091	36
H070	31
H189	30
H368	27

Table 3: Ten most common classes that appeared in the training dataset and the number of times they appear.

The YOLOv3 COCO configuration was selected for the classification task due to its performance on the object detection task. All training configurations were kept the same between the classification model and the detection model, with the only difference being the addition of the symbol classes in the dataset and the reduction of epochs from 40 to 30. Additionally, the configurations for the augmentation pipeline for the training of this model was the same as with the previous models.

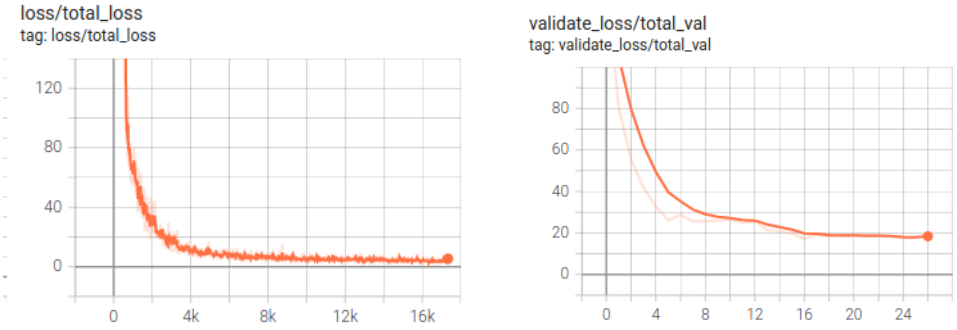


Figure 11: Training loss (left) and validation loss (right) for the YOLOv3 model trained on symbol classes.

It should be noted that the model seems to reach convergence at a much later stage than when it was performing object detection, due to the increased complexity of the task. Running for the full 40 epochs for the next set of training runs may give us a better picture of when the model reaches the desired optimum.

The resulting model fails to identify most of the symbol classes in the image. However, the few symbol classes that appeared in the dataset with a frequency are classified with surprising accuracy. It is important to caveat this statement, however. The metrics we used to evaluate the model's performance on the symbol classes were only gathered on the validation set, which consisted of only six images, insufficient to provide a full picture of the model's performance on the set of real-world IVC imagery. What the metrics do show is that there is potential for the detection and the classification tasks to be performed simultaneously, given enough data instances. As noted, transfer learning on a related dataset (SVHN) should help performance in this area tremendously.

The metrics used to evaluate the performance of the model were the number of true positives, false positives, and false negatives the model generated while performing the detection task. This analysis was constrained to the ten symbol classes that appeared in the training dataset with the highest frequency (given in figure 13). The confidence threshold used to classify a model's detection as a 'positive' was set at 0.5, as with the detection task, and the IOU threshold that was used to determine whether a detection should be classified as a 'True Positive' was set at 0.5. Here are the results for the ten classes:

	Truth Object	False Object
Detected	25	0
Not Detected	0	N/A

Table 4: Detection results for symbol class H311.

As with the detection task, the rows coincide with a model detection, while the columns correspond to whether there was a truth object at the site of the detection or not. The top left cell consists of True Positives, the top right cell False Positives, the bottom left cell False Negatives, and the bottom right true negatives. Based on these results, the model successfully detected all of the H311 symbols in the validation set and generated no false detections for this symbol class.

	Truth Object	False Object
Detected	6	0
Not Detected	0	N/A

Table 5: Detections for Symbol H127

	Truth Object	False Object
Detected	12	1
Not Detected	0	N/A

Table 6: Detections for class "Grapheme"

	Truth Object	False Object
Detected	6	0
Not Detected	0	N/A

Table 7: Detections for Symbol H060

	Truth Object	False Object
Detected	13	0
Not Detected	0	N/A

Table 8: Detections for Symbol H148

	Truth Object	False Object
Detected	6	0
Not Detected	0	N/A

Table 9: Detections for Symbol H148

	Truth Object	False Object
Detected	8	0
Not Detected	0	N/A

Table 10: Detections for Symbol H148

The other three symbols (H070, H189, and H368) were not present in the validation dataset.

As can be seen, the symbols that appeared with a high frequency in the training dataset were detected by YOLOv3 with a high precision (*Note: Some of the numbers reported here are repeat detections, which is the same bug as was present in the object detection module. To report an accurate result, non-max suppression should be applied on the predicted bounding boxes to reduce the chance of repeat detection).* This provides some confidence that an object detection model like YOLOv3 (and Faster R-CNN, if it were trained on the same dataset) could be used to also perform classification on high frequency graphemes with a high degree of accuracy.

Conclusion

In this paper we presented YOLOv3 as candidate model for the identification and classification of Indus Valley Script graphemes in curated images. We utilized 232 source images of Indus Valley Script seals and stamps. Further, we augmented the dataset with a custom data generator that applies a configurable set of pixel augmentations and affine transformations to create thousands of training images. In spite of this active augmentation chain, YOLOv3 still performed at worse at the object detection task than Detecto, showing that the Faster-RCNN architecture is better suited to identifying Indus Valley Graphemes than YOLOv3. It also revealed that our current data augmentation pipeline was not rigorous enough to overcome the greater detection power of the Faster R-CNN model. Efforts to produce higher fidelity labels will need to be considered when making refinements to the automated augmentation pipeline. In addition, while YOLOv3 was able to detect some, select symbol classes with high

precision, it suffered at identifying the vast majority of Harapan symbols in the dataset. However, the results reported by the symbol classification module demonstrate that a combined object detection and classification approach are feasible.

Future Work

Our work identified several areas of potential future improvements and refinement. First, a Faster RCNN architecture trained with our active augmentation chain could be compared to the results output by Detecto trained without the active augmentation chain. Second, an attention mechanism such as VGG [27] employed in front of YOLO may serve to increase accuracy, precision, and recall in noisy images or images with more complex seals as attention mechanisms have been shown to increase accuracy in natural scene text detection [10] [11]. Third, the current approach could be adapted to feed cropped images or coordinate sets to an established OCR model such as Tesseract [28] for identification in conjunction with the Harappan font [29]. Fourth, additional affine transformations could be added to the active augmentation chain, such as 3-d Rotation and image cropping. Fifth, identifying methods of rotating bounding boxes and designing models to accept these rotated bounding boxes may help with preserving all truth labels when a rotation augmentation is performed. This is important because the small size of the dataset makes it important that all labels are preserved during an affine transformation. Currently, some of the labels are removed based on overlap with labels next to it. Rotating the labels with the original image would solve this problem. Finally, identifying methods for robustly training the model on a wider range of symbol classes will need to be found. Currently, the distribution of symbols classes in the dataset is too heavily weighted to a few, select symbol for the model to effectively learn representations for all the Harappan symbols. The simplest, but most difficult to implement, solution to this problem is to devise a “mechanical Turk” process that allows trained researchers in the field to label images of Harappan seals and tablets containing Indus Valley Characters.

References

[1] W. contributors, "Indus script," Wikipedia, The Free Encyclopedia., 20 April 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Indus_script&oldid=1018936730. [Accessed 21 April 2021].

[2] W. contributors, "Indus Valley Civilisation," Wikipedia, The Free Encyclopedia., 14 April 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Indus_Valley_Civilisation&oldid=1017685693. [Accessed 21 April 2021].

[3] E. Cork, "Peaceful Harappans? Reviewing the evidence for the absence of warfare in the Indus Civilisation of north-west India and Pakistan (c. 2500-1900 BC).," *Antiquity,* vol. 79, no. 304, p. 411+, 2005.

[4] M. S. Sharma, "Why we still can't crack the Indus script.," *The Times of India,* p. N/A, 9 February 2020.

[5] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud and V. Shet, "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks," 2013.

[6] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang and A. y. Ng, "Large Scale Distributed Deep Networks," in *Advances in Neural Information Processing Systems 25*, Lake Tahoe, 2012.

[7] R. Elnabawy, R. Elias and M. A.-M. Salem, "Image Based Hieroglyphic Character Recognition," in *14th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, Las Palmas de Gran Canaria, Spain, 2018.

[8] R. Elnabawy, R. Elias, M. A.-M. Salem and S. Abdennadher, "Extending Gardiner's code for Hieroglyphic," *Multimedia Tools and Applications,* vol. 80, p. 3391–3408, 2021.

[9] J. Ba, V. Mnih and K. Kavukcuo, "Multiple Object Recognition with Visual Attention.," in *ICLR*, San Diego, 2015.

[10] L. Dong, D. Zhou and H. Liu, "A Weakly Supervised Text Detection Based on Attention Mechanism," in *10th International Conference, ICIG*, Beijing, China, 2019.

[11] W. Li, K. Liu, L. Zhang and F. Cheng, "Object detection based on an adaptive attention mechanism.," *Scientific Reports,* vol. 10, no. 1, p. N/A, 1 December 2020.

[12] P. Keserwani, A. Dhankha, R. Saini and P. P. Roy, "Quadbox: Quadrilateral Bounding Box Based," *IEEE Access,* vol. 9, pp. 36802-36818, 2021.

[13] Z. Zhong, L. Sun and Q. Huo, "An Anchor-Free Region Proposal Network for Faster R-CNN based Text Detection," *International Journal on Document Analysis and Recognition,* vol. 22, no. 3, pp. 315-327, 2019.

[14] R. P. N. Rao, N. Yadav, M. N. Vahia, H. Joglekar, R. Adhikari and I. Mahadevan, "A Markov model of the Indus script," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 106, no. 33, pp. 13685-13690, 2009.

[15] A. Bi, "detecto," [Online]. Available: https://pypi.org/project/detecto/. [Accessed 2021].

[16] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *ArXiv CoRR,* vol. abs/1804.02767, p. N/A, 2018.

[17] T. Contributors, *PyTorch,* 2019.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia and L, *TensorFlow: A System for Large-Scale Machine Learning,* 2015.

[19] S. M. Sullivan, Indus Script Dictionary, Suzanne Redalia.

[20] O. Khan, J. Turner, I. Aronovsky, M. W. Iqbal, V. Dave, N. Zubair and A. Bhandari, "Harappa.com," [Online]. Available: https://www.harappa.com/. [Accessed 2021].

[21] "Revealing India and Pakistan's Ancient Art and Inventions," National Geographic, 17 April 2013. [Online]. Available: https://blog.nationalgeographic.org/2013/04/17/revealing-india-and-pakistans-ancient-art-and-inventions/. [Accessed 2021].

[22] "Grade 5 Discovers 5000 Years Old Artefacts!," International School of Turks & Caicos, 30 January 2013. [Online]. Available: http://www.internationalschooltci.com/grade-5-discovers-5000-years-old-artefacts/. [Accessed 2021].

[23] M. Everson, *Proposal for encoding the Indus script in Plane 1 of the UCS,* ISO/IEC, 1999.

[24] T.-Y. Lin, G. Patterson, M. R. Ronchi, Y. Cui, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, L. Zitnick and P. Dollar, "COCO Dataset 2020," 2020. [Online]. Available: https://cocodataset.org/#home. [Accessed 2021].

[25] Deep AI, Inc., "Pascal VOC Dataset," Deep AI, Inc., 2012. [Online]. Available: https://deepai.org/dataset/pascal-voc. [Accessed 2021].

[26] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens and Q. V. Le, "Learning Data Augmentation Strategies for Object Detection," 2019.

[27] E. Coto and A. Zisserman, "VGG Image Classification Engine," Oxford, 2017. [Online]. Available: https://www.robots.ox.ac.uk/~vgg/software/vic/. [Accessed 2021].

[28] A. Kay, "Tesseract: an open-source optical character recognition engine," *Linux Journal,* vol. 2007, no. 159, p. N/A, 2007.

[29] A. Parpola, "A Free Complete Indus Font Package Available," Harappa.com, 6 April 2017. [Online]. Available: https://www.harappa.com/blog/free-complete-indus-font-package-available. [Accessed 2021].

[30] A. Parpola, Deciphering the Indus script, New York: Press Syndicate of the University of Cambridge, 1994.

[31] X. Yang, D. He, Z. Zhou, D. Kifer and C. L. Giles, "Learning to Read Irregular Text with Attention Mechanisms," in *International Joint Conference on Artificial Intelligence*, Melbourne, Australia, 2017.

[32] A. Vaswani, N. Shazeer, N. Parmer, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," in *31st Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017.

[33] Z. Wojna, A. Gorban, D.-S. Lee, K. Murphy, Q. Yu, Y. Li and J. Ibarz, "Attention-based Extraction of Structured Information from Street View Imagery," in *Proceedings - 14th IAPR International Conference on Document Analysis and Recognition*, Kyoto, Japan, 2017.

[34] X. Zhu, D. Cheng, Z. Zhang, S. Lin and J. Dai, "An Empirical Study of Spatial Attention Mechanisms in Deep Networks," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South), 2019.

[35] G. Li, Z. Song and Q. Fu, "A New Method of Image Detection for Small Datasets under the Framework of YOLO Network," in *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference(*, Chongqing,China, 2018.

[36] N. Yadav, H. Joglekar and R. P. N. Rao, "Statistical analysis of the Indus script using n-grams," *PLoS ONE,* vol. 5, no. 3, pp. 1-16, 2010.

[37] N. Islam, Z. Islam and N. Noor, "A Survey on Optical Character Recognition System," *Journal of Information & Communication Technology (JICT),* vol. 10, no. 2, 2016.

[38] S. Daggumati and P. Z. Revesz, "Data Mining Ancient Script Image Data Using Convolutional Neural Networks," in *Proceedings of the 22nd International Database Engineering and Applications Symposium, IDEAS 2018*, Villa San Giovanni, Italy, 2018.

[39] S. Palaniappan and R. Adhikari, "Deep Learning the Indus Script," 2017.