**Abstract:**

*Conserving, capturing and reading text hidden away in fragile ancient works of literature has troubled historians for centuries. Most attempts to recover information from ancient documents have relied on physically altering the structure, often resulting in damage to the physical object. In the past year, Seales et al. [****need better citation here****] introduced ground breaking techniques for capturing text hidden away in ancient documents (in their case, an extremely fragile scroll from En-Gedi) while conserving the physical structure. Their methods involve capturing microComputed Tomography scans, constructing a volumetric model, and digitally uncovering the text. The software pipeline consists of multiple components – one of which is the process of isolating voxel located on scroll surfaces, henceforth referred to as segmentation. The aim of this paper is to apply machine learning to the segmentation process. This paper draws heavy influence from the work done by [****CITE PAPER****] where the strengths of Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Conditional Random Fields were combined to achieve state-of-the-art results for image segmentation.*

**Introduction & Related Work:**

There is a collection of ancient manuscripts around the world that remain unreadable. A large amount are too fragile to open, and the text unreadable. While there exists many types of ancient manuscripts, for this paper I am going to focus on scrolls. In particular, I am going to focus on scrolls found in the ancient city of Herculaneum. After the eruption of Mt. Vesuvius, a library containing hundreds of scrolls from antiquity were buried under volcanic ash.

Scrolls exist in three dimensional space, meaning in order to unwrap the scroll, a volumetric model must be generated. After the model is generated, the voxels must be pruned in a way that only the voxels that contain relevant information remain. Segmentation is the process of isolating only those voxels located on papyrus surfaces wherein text exists.

The software process developed in [CITE DR SEALES PAPER] includes a current segmentation algorithm contained within the in-house developed *Volume Cartography* application. The application displays a GUI where the user can draw points on a local scroll layer, afterwards a spline function is interpolated between points. The application then makes use of an algorithm that attempts to maintain the location of the spline function on the scroll layer through the z-dimension. The algorithm constructs eigenvectors around local sub-volumes and makes a greedy decision as to where the next coordinate (in the z-dimension) located on the scroll surface is. While the application produces results, it is time consuming – requiring humans to sort through every slice and verify correctness. The segmentation algorithm is also unable to handle odd scenarios that arise in the volumetric scan.

The scrolls found at Herculean were written on papyrus, and papyrus raises two essential problems during the segmentation process. Papyrus is constructed by pressing together crosshatched pieces from the papyrus plant. Due to the nature of the *pressing* and the general strength of papyrus, the scrolls are easily torn [INCLUDE PICTURE AS EXAMPLE]. Torn surfaces makes the segmentation algorithms greedy decision more ambiguous, and thus results are less consistent. Furthermore, the sheets of papyrus lack rigidity and are easily folded. A fold in the *x,y* plane causes no issue for the segmentation algorithm. It is when the papyrus folds inwards to the *z-dimension* that causes issues. The segmentation application moves perpendicular to the *z-dimension* and only generates single points for each *z-slice.* When a papyrus surface folds inwards in the *z-dimension* the segmentation algorithm tends to lose track of the surface and/or doesn't generate multiple spline functions within a single slice.

By applying machine learning to the segmentation process, the issues raised above can be resolved. Machine learning, and Convolutional Neural Networks (CNNs) in particular, have shown state-of-the-art performance in image classification, object recognition, and pixel-wise classification – even out performing human standards in some cases. For this paper, I apply CNN's for voxel-wise classification on volumetric data – otherwise known as volumetric image segmentation. The application of deep networks towards segmentation on volumetric data remains a relatively undiscovered field. Through the use of three-dimensional receptive fields, CNNs can offer a level of perception not viewable by humans. Since a three-dimensional CNN can offer new fields of view, the problem of papyrus shredding and folding can be resolved. A level of surface cohesiveness through the *z-dimension* can be viewed by a three-dimensional receptive field. For example, the crosshatching patterns are not viewable through a single *x, y* view, but given a view of all three dimensions, the CNN can perceive a continuous scroll surface.

There have been a collection of notable published papers that applied CNN for image segmentation – some of which are two-dimensional networks and more recently, others extended the networks to three-dimensions. *SegNet* is a well known deep network for image segmentation [REFERENCE]. SegNet makes use of an encoder-decoder architecture where shallower downsampling layers are forward passed to deeper upsampling layers. [REFERENCE] claimed that through the use of the decoder stack, predictions appear less blocky. [REFERENCE – large scal video classification with CNN] developed what was referred to as *time information fusion* into CNNs in order to take advantage of local spatio-temporal information. While the input images were still two-dimensional, the data was organized in sequential frames. More recently [REFERENCE – Vnet] applied three-dimension CNNs for volumetric medical image segmentation. The CNN was composed of convolutional layers wherein the receptive fields consisted of three dimensions – thus capturing information in all three dimensions of volumetric scans. A similar implementation was developed by [REFERENCE – 3D CNNs for humans action recognition] where a three-dimensional CNN made classifications on human actions. Similar to [REFERENCE – largse scale video classification] the data consisted of video frames, only instead of two-dimensional input frames, the frames were stacked into three-dimensional volumes.

This paper attempts to apply parts of the network developed in [REFERENCE main paper] extended to a three-dimensional CNN. [REFERENCE main paper] is an extension of the work done by [REFERENCE – semantic image segmenation with deep CNNs and fCRFs] where a fully connected Conditional Random Field (CRF) served as a post processing step after a CNN for semantic image segmentation. [REFERENCE semantic image segmentation] claimed that CNNs outputs are not sufficiently localized for accurate segmentation, and that the application of CRFs can refine local information. [REFERENCE main paper] refined this technique by constructing an end-to-end trainable network that makes use of a CNN connected to a recurrent CRF – referred to as a RNN-CRF. Since the network is constructed in an end-to-end fashion, weight optimization through backpropagation can be conducted in a single system.

**Model:**

**CNN:**

The CNN developed in this paper was developed entirely in house. A full diagram of the CNN can be seen in FIGURE X.

To segment a volume, the output volume must be the same dimensions as the input volume. CNNs often make use of max pooling layers or downsample layers through non-unit strides. Downsampling is the process of reducing image dimensions to capture local information. Since the dimensions must be maintained in the prediction volume, downsampling is an issue for volumetric image segmentation. Not making use of downsampling is also an issue because the strength of capturing local information is not put to use. This network makes use of downsampling in order to perceive local information, followed up upsampling in order to perceive global information. Upsampling layers are commonly incorrectly labeled as deconvolutional layers, the correct term for upsampling layers is convolutional transpose layers. Convolutional transpose layers make use of padding the input volume so that the output volume can have larger dimensions – see FIGURE X1 as an example convolutional transpose layer. The general structure of the CNN is composed of three convolutional layers followed by three convolutional transpose layers. Each convolutional layer makes use of a stride equal to two, thus halving the volume dimensions. Each convolutional transpose layer also makes use of a stride equal to two so that the dimensions of the final output volume are equal to the input volume.

The CNN also makes use of a handful of other modern techniques. In order to prevent learning divergence, convolutional filter weights are initialized using Xavier Initialization. Xavier Initialization initializes weights depending on the number of incoming and outgoing connections. If weights are initialized too small then there is insufficient signal strength through the network. If weights are initialized too large, then the network predictions diverge into random noise. Xavier Initializations resolve both of these problems.

After various experiments, I found that defining receptive field sizes to roughly one fourth of the input volume dimensions produced the best results. Receptive fields, kernels, and filters are commonly used interchangeably. Convolutional layers share weights across the entire input volume. For example, the filter for the first convolutional layer has the dimensions: [z, x, y, number of input channels, number of output channels] → [14, 14, 14, 1, 16]. The number of input and output channels is more correctly labeled as the size of the feature map. Each feature map uses a single convolutional filter across the entire volume – this can lead to blocky predictions (discussed further in section [REFERENCE SECTION]).

All layers make use of batch normalization. Batch normalization is placed after each linear Rectified (ReLu) activation function and prevents divergence. Dropout is applied after the final layer to prevent overfitting. Dropout is a mechanism where a random number of connections (specified by a probability) in the final layer are disconnected. The probability of dropout is a tunable hyperparameter.

**Conditional Random Fields:**

A Conditional Random Field (CRF) is a statistical modeling method than can be applied to machine learning for structured predictions. In the case of this model, the CRF is responsible for voxel-wise classification. In a fully connected CRF, the energy function of a label assignment x is given by:

[FORMULA]

The unary term [FORMULA] measures the cost of voxel *i* taking the correct label. The pairwise term measures the cost of assigning labels to voxels *i* and *j* simultaneously. The unary term is handled by a convolutional filter. The pairwise term is handled by implementing a Gaussian filter, otherwise known as a bilateral filter, where the formula for the labels assigned to voxels *i* and *j* is given by:

[FORMULA]

**Recurrent Neural Network – Conditional Random Field:**

The Recurrent Neural Network Conditional Random Field (RNN-CRF) is nearly identical to the model developed by [REFERENCE main paper]. The general architecture of the RNN-CRF inputs three volumes: the original volume, the output volume from the CNN (not normalized), and the normalized output volume – initially from the CNN, but after the first iteration, the output of the RNN-CRF serves as the third input to the RNN-CRF. The softmax function is used as the normalization function for this network. See FIGURE X2 for a diagram of the general network architecture.

[REFERENCE main paper] developed a clever implementation of a CRF by making use of convolutional layers. The volume dimensions are maintained throughout the RNN-CRF – no downsampling or upsampling occurs. [REFERENCE main paper] labeled the CRF implementation as *a mean-field iteration as a stack of convolutional layers*. There are five steps to the CRF: message passing, re-weighting, compatibility transform, unary addition, normalization. See FIGURE X3 for a diagram of the CRF. After the final step, the output from the normalization is fed back into the message passing step – this is the *recurrent* component of the RNN-CRF. The number of iterations is a tunable hyperparameter.

**Message Passing:**

The message passing step of the CRF is where Guassian Kernels are applied to the input volume. Gaussian Kernel filters rely on capturing two essential components from the input volume: voxel intensity values and voxel locations. The responsibility of the Gaussian Kernel is to penalize voxels with similar values that are far apart. The network developed in this paper implements message passing in two convolutional layers. Both layers are fully connected, so the size of the receptive fields are equal to the volume dimensions. The first layer is a fully connected convolutional layer with learnable filters – this measures voxel intensity values. The second layer is a fully connected convolutional layer with constant filter values. The weights within this filter are equal to the distance between each voxel. The input volume has a single input channel (consisting of the original volume and the normalized output of either the CNN or the RNN-CRF – depending on the iteration number) and *M* output channels.

**Re-weighting Filter Outputs:**

The purpose of the re-weighting step is to take the weighted sum of each filter weight from the M feature maps. This step can be implemented by simply utilizing a single convolutional layer where the filter size is equal to 1x1x1. For this step there are *M* input channels from the message passing step and a single output channel.

**Compatibility Transform:**

The compatibility transform step is responsible for assigning penalties to voxels that share similar properties but are assigned different labels. This is defined in the Potts model as *u(l,l') = [l != l']*, however this model is not flexible. Given a penalty in the Potts model, the penalty is always the same constant value. The compatibility transform provides flexibility through utilizing a convolutional layer with a receptive field size of 1x1x1. Therefore, the weights will learn to apply different levels of penalty depending upon voxel location and intensity. This step should have *l* input channels and *l* output channels.

**Unary Addition and Normalization:**

In the unary addition step, the output from the compatibility transform is subtracted element-wise from the non-normalized output from the CNN. Subsequently, the output volume is normalized via the softmax function. After normalization, if the iteration number is less than the specified hyperparameter, then the normalized output is fed back into the message passing step.

**Data:**

**Input Data:**

The input data for this experiment was collected from a sample scroll from the EDUCE project [HOW DO I REFERENCE THIS?]. The dimensions for the entire scan is 3820x3820x3398. The total size, after compression, on this dataset is toughly 68GB. Due to hardware limitations, training on the entire scroll is out of question. Therefore the scroll was cropped into a more manageable size. There were two training sessions conducted in this experiment – each with different input volume dimensions. Training session number one has an entire training volume of dimensions [396, 396, 100] and an input sub-volume of dimensions [54, 54, 54]. Training session number two has an entire training volume of dimensions [160, 160, 160] and an input sub-volume of dimensions [40, 40, 40]. The microComputed Tomography scan was set to capture one voxel for every 14 micrometers. Each volume crop was sliced in half along the z-dimension so that one half was used as training data and the other half used as prediction data.

**Ground Truth**

Ground truth for the cropped volumes was collected by using the already developed segmentation method (outline in section [REFERENCE section]). To recap, the *Volume Cartographer* application requires the user to define a set of seed points. Subsequently a spline function is interpolated between the seed points. Once segmentation begins, local sub-volumes are passed into an algorithm which makes a greedy decision on the location of the next surface voxel. The segmented points generated by *Volume Cartographer* are placed in PLY files. The points are sparse, and therefore must be filled in. A data script executes a recursive algorithm that first fills in lines between points – making each surface a continuous line – and then expands the line to approximately the width of the actual scroll surface. Expanding each line generates a probability gradient where the highest probability is located at the center of the layer, and the lowest probability is located at the edge of the layer.

**Experiments:**

The entire network and all data handling was written using Google's Tensorflow deep learning API. See FIGURE X4 for a full digram of the network. The learning rate was set to 0.001. The number of training iterations was set to 2,000. The batch size was set to 3. Keeping consistent with modern deep networks for image processing, the softmax function was used after the final layer. The Adam Optimizer was used for optimizing weights through gradient descent. Since each voxel in the ground truth is a probability value, classification isn’t a discrete prediction but instead a linear regression [DR JACOBS IS THIS STATEMENT ACCURATE?].

Due to the nature of the convolutional filter, predictions can appear blocky. A *smoothing* method was developed to overcome blocky predictions. The prediction volume is half of the total volume of the input volume so that edge voxels share the same amount of information as center voxels. Input volumes overlap with a step of nine, and therefore predictions are also overlapped. Overlapping predictions improves the smoothness of prediction volumes – every voxel is an average of nine overlapping volumes.

INCLUDE:

* graph of loss value over time
* graph of accuracy over time?
* Prediction slice next to actual slice
* compare to network without RNN-CRF
  + repeat the first three bullet points

Drawbacks: There is one notable drawback in using Tensorflow for implementing a CRF – Tensorflow has no native support for bilateral filters. The bilateral filter component was left out for this paper.

**Conclusion:**

[REFERENCE SEALES] published ground breaking results and proved that uncovering ancient text from damaged and fragile scrolls is possible without physically destroying the object. One component of the unwrapping pipeline is the process of segmenting voxels located on scroll surfaces. The current software pipeline can handle segmentation, but is time costly, requiring a human to sort through every slice and verify correctness. In this paper, we have discovered the potential in applying machine learning to automate the segmentation process. One drawback in the current deep learning segmentation implementation is the fixed width ground truth. The actual scroll layers vary in width. Also, since *Volume Cartographer* generates points that are not always placed in the center of the scroll layer, expanding the width of the ground truth can often expand past the true scroll surface. The RNN-CRF has shown promising results in picking up variable width layers, but there is still work to do to improve ground truth data. Future endeavors include applying a deep network to the current segmentation process, and/or even developing an entirely new and fully automated segmentation process through machine learning.