**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. For example, at the time of creation, the scrolls were simply wraps of papyrus. Over time, however, the papyrus dries out and, in the case of the Herculaneum scrolls, filled in with volcanic ash. 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 throughout a local scroll surface, where after 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 surface 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 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 greedy decision in the segmentation algorithm 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 deep learning to the segmentation process, the issues raised above can be resolved. Deep 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 pixel-wise classification on volumetric data – otherwise known as volumetric image segmentation. The application of deep networks towards image 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 used two-dimensional data and more recently, others extended the networks to handle three-dimensional data. *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 inputting two-dimensional frames, input data was made up of a set of frames resulting in a three-dimensional volume.

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 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. Since the dimensions must be maintained in the output volume, downsampling volume dimensions is an issue for volumetric image segmentation. Not making use of downsample is also an issue in that 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 [REFERENCE TO OG PAPER]. Convolutional transpose layers make use of padding the input volumes such that the output volume can be of a higher dimension – 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 such 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. All convolutional filter weights are initialized using Xavier Initialization. After various experiments, I found that defining receptive field sizes to roughly one fourth of the input volume dimensions produced the best results. All layers make use of adding biases and biases are initialized to zero. All layers make use of the ReLu activation function. All layers make use of batch normalization.

**Conditional Random Fields:**

**Recurrent Neural Network – Conditional Random Field (RNN-CRF):**

The RNN-CRF is nearly identical to the model developed by [REFERENCE main paper]. The general architecture of the RNN-CRF takes 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 for the CRF component 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 location and voxel intensity value. The responsibility of the Gaussian Kernel is to penalize voxels with similar values that are far apart. [REFERENCE main paper] makes use of *edge-preserving* Guassian filters so that the filters can have fewer parameters despite being as large as the image. [TENSORFLOW CURRENTLY DOES NOT HAVE THIS IMPLEMENTED???]... The message passing step in the CRF can be implemented by simply utilizing a single convolutional layer where the receptive field has the same dimensions as the input volume – a fully connected CRF. 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 filter outputs step is to take the weighted sum of the M filter outputs from the previous step. 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, the penalty is the same constant value for all voxels. 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 hyperparemeter, then the normalized output is fed back into the message passing step.

**Data:**

**Experiments:**

The entire network and all data handling was written using Google's Tensorflow deep learning API.