Summer Internship Report

Introduction

This report summarises all the work done during our two months at UT Austin working under Dr. Chandrajit Bajaj.

Neuropil data

Neuropil is any area in the nervous system composed of mostly unmyelinated axons, dendrites and glial cell processes that forms a synaptically dense region containing a relatively low number of cell bodies. The most prevalent anatomical region of neuropil is the brain which, although not completely composed of neuropil, does have the largest and highest synaptically-concentrated areas of neuropil in the body. For example, the neocortex and olfactory regions.

The data used was derived from the hippocampus region of a rat's brain. The data contains images that were obtained by slicing the part of the brain at discrete intervals and observing those intervals under a microscope.

VolRoverN

VolRoverN is an open source project allowing anyone to use its features to generate physically accurate 3d representations from multiple 2d-contour images. Contours, when being modelled from images can be inaccurate and intersect, which must be removed, taken care of. VolRoverN can automatically remove these imperfections and optionally ensures there is a minimum distance between contours.

The general steps followed by VolumeroverN are:

(i) 2D Processing: Here the intersection between contours is removed in order to maintain the physically accurate representations of structures.

(ii) 2D to 3D: Contour tiler is used to match the same object in different contours to produce a single object in the 3D reconstruction. A heuristic approach is used by employing the use of projections of the contours. Once objects are generated from Contourtiller, ForestTiler makes sure these objects do not intersect while also making sure the original interpolation is left intact.

(iii) 3D processing: The quality of the reconstruction is improved by reducing the total number of triangles and by making the sides of roughly an equal length. Qslim algorithm is used to eliminate edges and then the aspect ratio of the triangles is improved. Repair is done after this as improving the quality generally results in holes or intersections. Complimentary Space and Tetrahedralization allows one to generate the extracellular space once the object's model is complete, within a bounding box of specified size.

Earlier, the available software used algorithms like marching cubes and Boissonnat algorithms which produced blocky reconstructions. These software were made for speed and not for accuracy. These resulted in reconstructions with objects intersecting, and holes. VolumeRoverN solves these issues.

VolRoverN uses scalable algorithms. Reconstruction methods use only the 2 adjacent layers, hence only they are loaded into memory. Increased memory usage is only observed when using higher resolution images or when using more contours, as is expected. But since VolRoverN operates only on pairs of layers, parallelization is the next obvious addition to the already impressive suite of tools and performance characteristics that VolRoverN boasts.

Therefore, this paper demonstrates the capabilities of VolumeRoverN for biological applications. The software gives accurate results for simulations. The results are given in different formats for further processing by different software.

Objective

The main objective is to segment the neuropil data using machine learning algorithms or deep learning frameworks in order to eliminate the step of drawing the contours by hand in the 3D reconstruction of the stack.

Work Done

Unsupervised Learning:

We extracted a 1000x1000 pixels image crop (Fig. 1) from 12007x12007 TIF images available to speed up processing,and to take advantage of the homogeneity of the data.

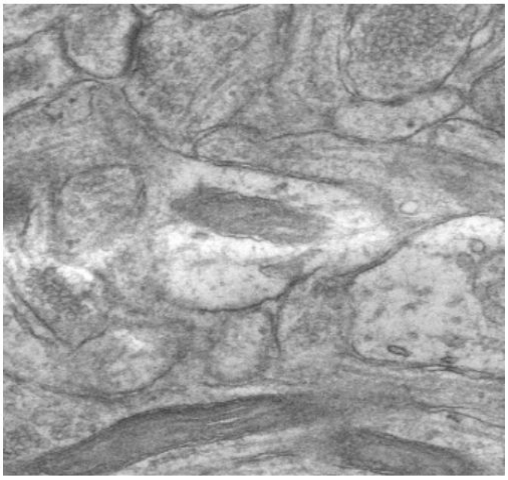


Fig. 1

1. We tried using HDB-Scan, an unsupervised learning algorithm that tries to group related pixels together, which recognized about 190 regions, based on different minimum cluster sizes.

2. To test the accuracy of our code and usage of the algorithm, we generated a simple flat image (Fig 2) form the age old paint application that ships with windows.

3. This image was converted to grayscale and run through HDB-Scan. It detected 7 classes for image Fig. 3 (which is correct)

4. We added some white noise to Fig. 2 and generated the grayscale and detected 4 classes.

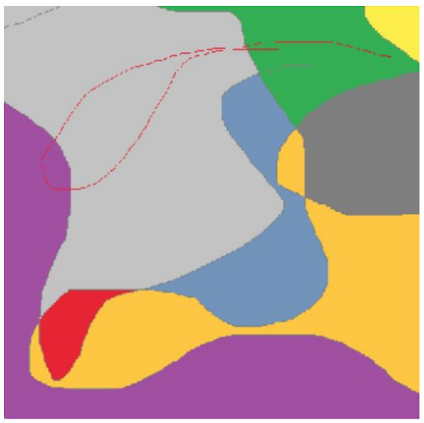


Fig. 2

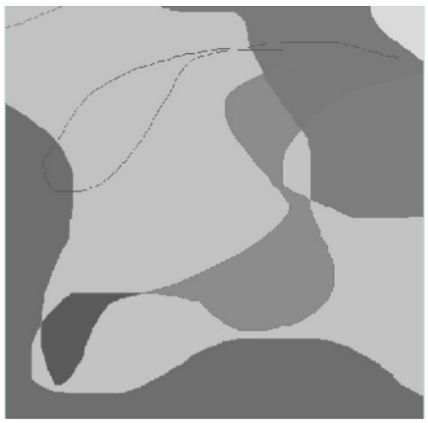


Fig. 3

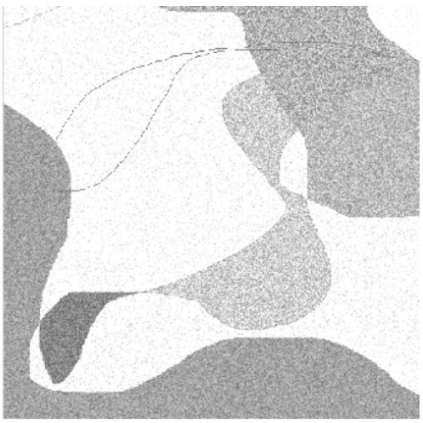


Fig. 4

Conclusion:

We think that the images at hand are too noisy or have too many sub-features, do not have distinct isolation of important features to be used reliably for unsupervised learning.

Canny Image Filter:

We used canny edge detection algorithm to be able to detect the important and major edges in the image. In interest of automatically detecting the contours for a new dataset we tried to detect the contours using canny edge detection algorithm. Various sigma values were tried and the results were as follows:

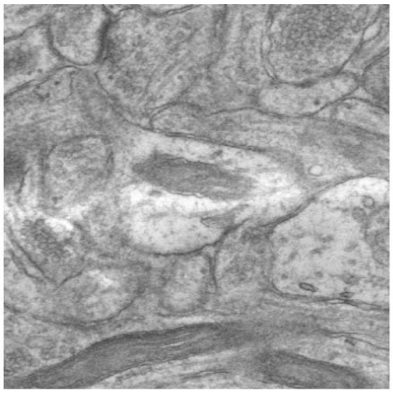


Fig. 5: Original Image

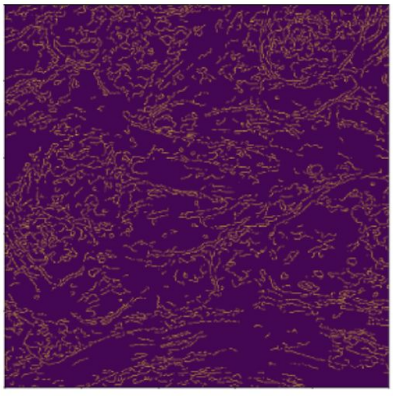


Fig. 6: Low sigma

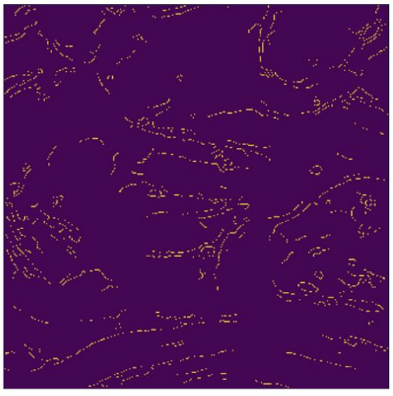


Fig. 7: Sigma = 3 (Best Output)

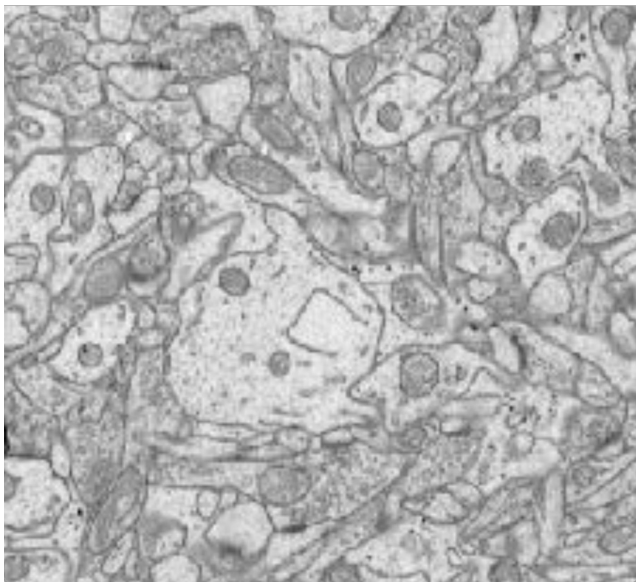
Conclusion:

The canny edge detection algorithm is not able to detect boundaries of the cells along with the extracellular spaces. Since the edges are not clear, we think that an unsupervised algorithm might not be the right choice for classifying grayscale images,

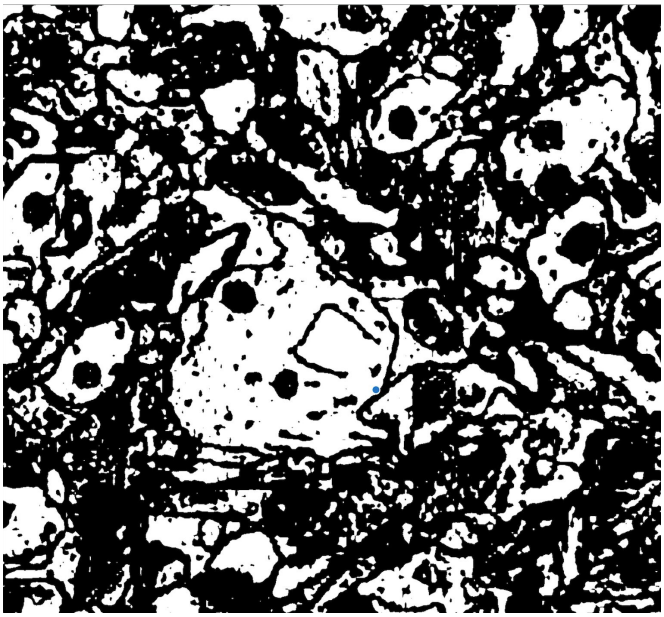
We also tried the edge detection on 200x200 and 100x100 images with original and contrast enhanced images to try and reduce the number of features and make the cellular boundaries more prominent but it yielded similar results.

Labelling through Image Manipulation:

An example of a pre labelled image was found in a powerpoint presentation. This image is used was used to generate labelled data.

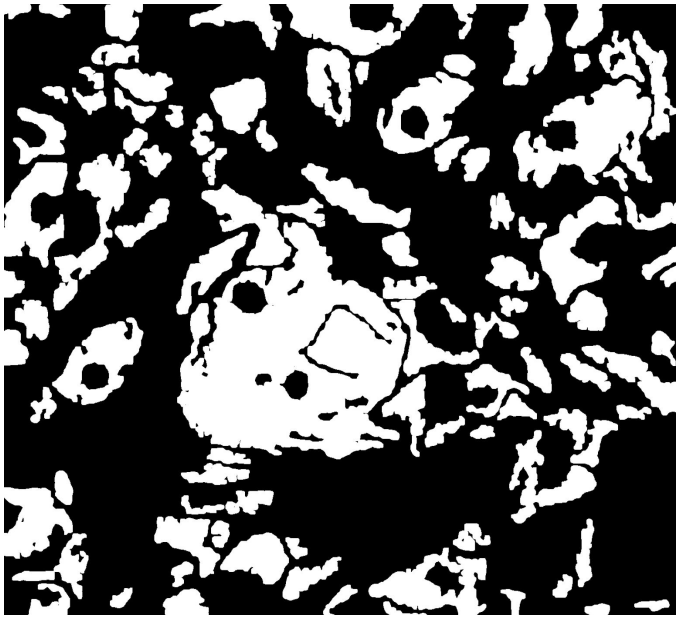


First the image was binary thresholded as the colour of the dendrites and the glial cells seemed lighter than the colour of the axons.

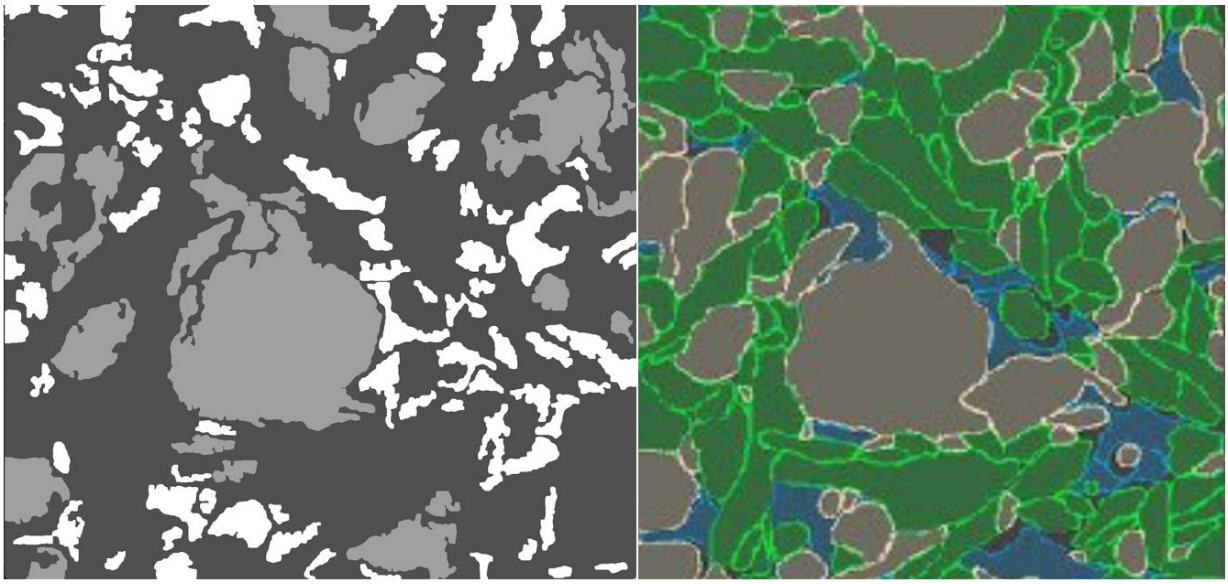


The image obtained is very noisy and hence algorithms to remove small objects are used to get a cleaner image. (morphology.remove_small_objects is used). The algorithm runs a window through the image and wherever it finds an object of smaller area than the minimum area specified by the user, it removes it. After the noise was removed from the image, the image looked like this:

It was observed that dendrites usually had mitochondria in them whereas glial cells were empty. Therefore In the binary thresholded image, white spots that contained black within them were mostly dendrites and white spots without any black spots within them were either small dendrites or glial cells. The latter was true in most cases. Some of the mitochondria were connected to the background in the image and to remove that connection and get a distinct black spot within the whites morphology.erode was used. Erosion works by stripping away pixels from the inner and the outer boundary of objects. Usually the pixels are set to the highest possible value allowed. Therefore in grayscale, the value is set to 255. After erosion the image looks like this:



After this, all the white parts with disconnected black spots in between are classified as dendrites whereas the other parts are classified as glial cells. The coloured image below is the original classification whereas the grayscale image is classified using the methods stated above.



Glial: White

Dendrite: Light Grey

Axon: Dark Grey

Glial: Blue

Dendrite: Grey

Axon: Green

Drawbacks:

1. The assumption that all dendrites have mitochondria in them is false but it holds true in most cases.
2. The accuracy of binary thresholding is not very high but the obtained image is more or less similar to the original classified image.

Joint Embeddings

The paper provides a joint embedding space allowing cross view image retrieval, image based shape retrieval and shape based image retrieval, which is constructed using 3D shape similarity measure as 3D shapes are encoded in a pure form and the similarity measure for 3D shapes is more robust.

Previous Work:

Shape signatures of previous work include construction of signature using geometric properties like volume, distance and curvature parameters, spherical harmonics etc. Subsequent methods used graph representation to obtain description of shape. We use a more efficient approach of Light Field Descriptor where shapes are indexed via a set of 2D projections and views.

Previously for Image retrieval, a method was suggested where the input to the system is a collection of pairs of images and uses Siamese network which discovers the embedded space and maps the images into the space. Whereas, the current method uses a robust distance metric for 3D shapes and then uses a CNN for purifying and mapping the images into embedded space.

Shape retrieval requires projections with clean backgrounds which are not feasible in real world applications. But our system uses CNN for image feature extraction where minimum effort involved in tuning the right parameters for similarity measurement.

a) Construction of embedding space:

- A embedded space is constructed by shape similarity where the distance between them reflects similarity.

The distance between two shapes is given by **$d(i,j) = (F(i) - F(j))^2$** ,

where F(i) and F(j) are the concatenation of viewpoint features vectors H(i,1), H(i,2),...., H(i,k). H(i,j) are based on the histogram of gradients for every projection I(i,v) of an object S(i).

- The embedding space uses a principal component analysis(PCA) to obtain compact version of F of significantly lower dimension while still preserving the pair of distances.
- An Alternate option for space construction is using Distance Matrix. We use MDS to to reduce it's dimension.
- After experimental validation, it is asserted that the reduced distance matrix (D-) is better than the feature vectors(F-).

When a new shape is introduced, say S*. then the distance between S* and S_i can be computed by the formula :

$$d(S*, S_i) = \| P(S^*) - P(Si) \|_2$$

where PS∗ can be solved by L-BFGS while minimising the sammon type error. Then, the top k nearest neighbours are considered.

b) CNN for Image Embedding:

- CNN is trained to map an image depicting an object similar to Si, to a point PI ∈ (D−)such that PI is close to PSi learn the latent connection that exists between an image and the object it features.
- In the training phase, a set of images R(i) are rendered for every S(i) and every S(i) is mapped to corresponding point Ps(i), thereby generating (R(i), Ps(i)) as the training data for our CNN.
- Then the CNN is used to map R(i,r) to Ps(i) disregarding the distracting factors like lighting, viewpoint and background characteristics. The mapping error with Euclidean Loss function is also calculated with inputs R(i) and Ps(i).

c) In testing phase, it is capable of mapping real world images into the same embedded space.

The efficacy and the performance of the method are evaluated using experiments which are categorised into cross view image retrieval, image based shape retrieval and shape based image retrieval.

We now describe the work that was done with Mask R-CNN, the implementation details on TACC, the problems we faced and the progress we achieved with it.

MASK R-CNN

Introduction

MASK R-CNN is an extension of Faster R-CNN that aims to add object instance segmentation along with mask predictions, in parallel to the bounding box predictions. This induces a very minor overhead but adds the advantage of generating masks for each instance of the object along with the bounding boxes. Mask R-CNN is an improvement over the architectures that came before it. CNN, R-CNN, Fast R-CNN, Faster R-CNN are the predecessors of Mask R-CNN, each an evolution of the previous. To understand Mask R-CNN, we must first understand how each of these work, what their capabilities are, and how the successor tries to improve on the latter.

CNN

The Convolution Neural Network used in the paper[1] by Alex Krizhevsky et al. consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. This network contains has 60 million parameters and 650,000 neurons. To train this network, 1.2 million labeled training examples were used. To prevent overfitting, the paper used data augmentation by

horizontally flipping the images. To reduce dependence of the CNN layers and prevent them for learning the intensities, to each training image, multiples of the found principal components were added with magnitudes proportional to the corresponding eigen values times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. The network also utilizes a 'dropout' mechanism by which zero is set to the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in backpropagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This allows the network to be more robust by learning features useful in conjunction with many different random subsets of the other neurons.

The architecture of the CNN is depicted in the figure below.

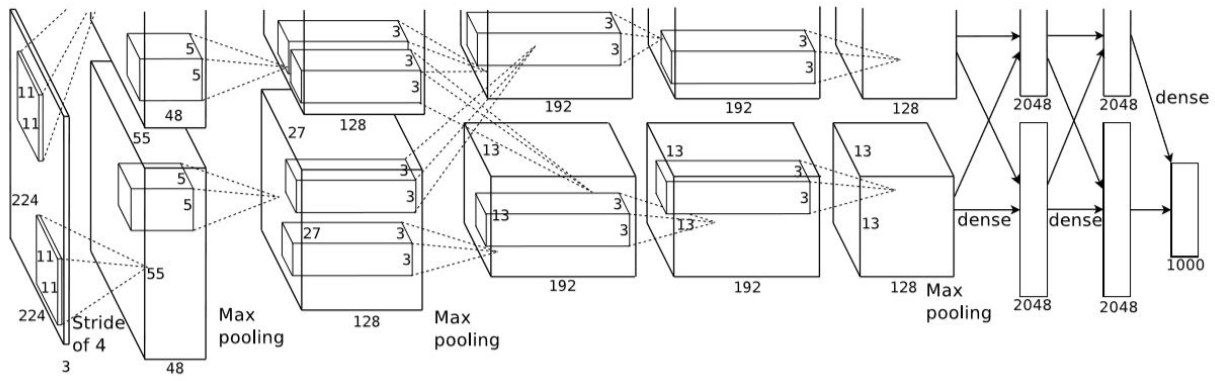


Fig. 8

The network uses stochastic gradient descent, updating the parameters as follows, $\left\langle \frac{\partial L}{\partial w}|_{w_i} \right\rangle_{D_i}$ where where *i* is the iteration index, v is the momentum variable, ϵ is the learning rate and is the average over the i^{th} batch D_i of inputs, of the derivative of the objective function with respect to *w*, at w_i

$$\begin{aligned} v_{i+1} &:= 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}|_{w_i} \right\rangle_{D_i} \\ w_{i+1} &:= w_i + v_{i+1} \end{aligned}$$

R-CNN

The paper[2] takes inspiration from the question '*To what extent do the CNN classification results on ImageNet generalize to object detection results on the PASCAL VOC Challenge*' and as an answer, presents R-CNN. HOG (Histogram of Oriented Gradients) is roughly similar to the cell in V1 region f primates, but one must also realize that detection of objects occurs at other levels also, not just the receptors. To replicate the receptors and the sensory pathways of the visual cortex, this paper uses deep neural networks coupled with CNN's to achieve better object detection, while narrowing the gap between detection and classification. To achieve this result, it focused on two problems: localizing objects with a deep network and training a high-capacity model with only a small quantity of annotated detection data.

The object detection system consists of three modules. The first generates category-independent region proposals. These proposals define the set of candidate detections available to our detector. The second module is a large convolutional neural network that extracts a fixed-length feature vector from each region. The third module is a set of class specific linear SVMs.

R-CNN uses selective search to enable a controlled comparison with prior detection work.

- Selective search is based and constructed on 3 basic principles; Capture All Scales, Diversification, Fast to Compute. To achieve this, hierarchical grouping algorithm was chosen as the basis of selective search. Since Bottom-up grouping was a popular approach to segmentation, it was chosen to generate the groups. This hierarchical features are combined on basis of similarity in texture, color and sizes (sizes of the individual regions). Specifically, for each region one-dimensional colour histograms are obtained for each colour channel using 25 bins. This leads to a colour histogram $C_i = \{ c^1_i, \cdots, c^n_i \}$ for each region ri with dimensionality n =75 when three colour channels are used. The colour histograms are normalised using the L1 norm. Similarity is measured using the histogram intersection.

$$s_{colour}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k)$$

And similarly for texture, for each (t_i^k, t_j^k)

- $s_{texture}(r_i, r_j)$ measures texture similarity. Texture similarity is measured using fast SIFT-like measurements. Gaussian derivatives in eight orientations using σ = 1 for each colour channel. For each orientation for each colour channel we extract a histogram using a bin size of 10. This leads to a texture histogram $T_i = \{ t^1_i, \cdots, t^n_i \}$ for each region r_i with dimensionality n = 240 when three colour channels are used. Texture histograms are normalised using the L1 norm. Similarity is measured using histogram intersection. $s_{size}(r_i, r_j)$ encourages small regions to merge early. This forces regions in S, i.e. regions which have not yet been merged, to be of similar sizes throughout the algorithm. This is desirable because it ensures that object locations at all scales are created at all parts of the image. $s_{size}(r_i, r_j)$ is defined as the fraction of the image that r_i and r_j jointly occupy.

$$s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(im)}$$

Where size(k) denotes the number of pixels used by k

The final similarity measure is given by

$$s(r_i, r_j) = a_1 s_{colour}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j),$$

By combining regions, the following results are obtained. The final stage is a support vector machine (SVM) that classifies the regions and assigns bounding boxes.

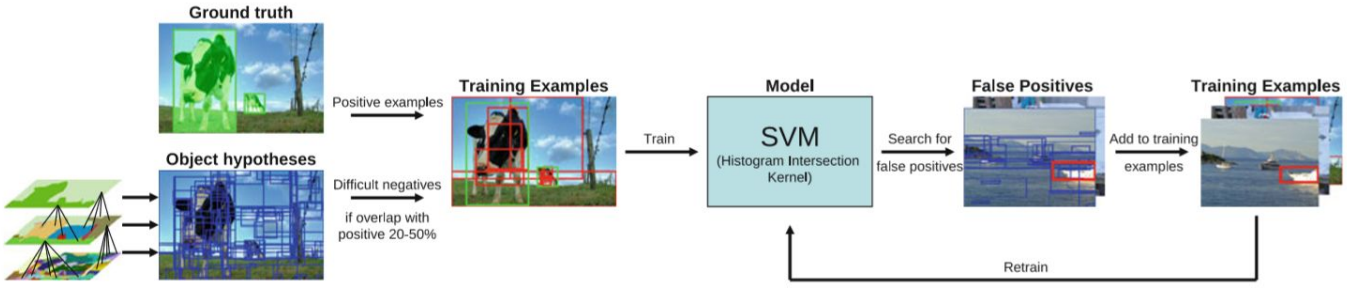


Fig. 9

A 4096-dimensional feature vector from each region proposal using the Caffe implementation of the CNN described by Krizhevsky et al. is used for the feature extraction. Features are computed by forward propagating a mean-subtracted 227×227 RGB image through five convolutional layers and two fully connected layers. (depicted in Fig. 10)

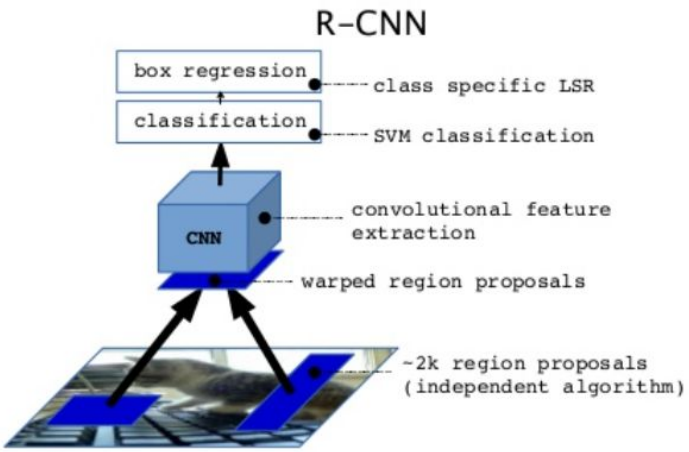


Fig. 10

In the last step, for each class, the extracted feature vector is scored (using an SVM trained for that specific class). After all the regions are scored, a greedy non-maximum suppression that rejects a region is applied, if it has an IoU larger than a certain threshold with another higher scoring region that has been selected.

Fast R-CNN

Fast R-CNN (paper[3]) tries to solve a few problems that plagues R-CNN. It is slow due to the fact that for each proposal, a forward pass of CNN (usually AlexNet) is required. As depicted above, there are three major segments, each with its own training, and hence the whole network is hard to train. Fast R-CNN tries to address these problems by using just a single pass of CNN and later using its output for all region proposals. For a given proposal, the corresponding section of the generated feature map (output of the CNN) is used as the input for the consequent layers.

Hence, Fast R-CNN network takes as input an entire image and a set of object proposals. The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a feature map. Then, for each object proposal a region of interest

(RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

Region Of Interest Pooling (ROI Pooling) uses max pooling to convert the features inside any valid RoI to a feature with fixed dimensions (HxW), defined as hyper parameters to the network. Each RoI is defined by a 4 tuple, 2 of which define the top left corner, and the other 2 the height and the width.

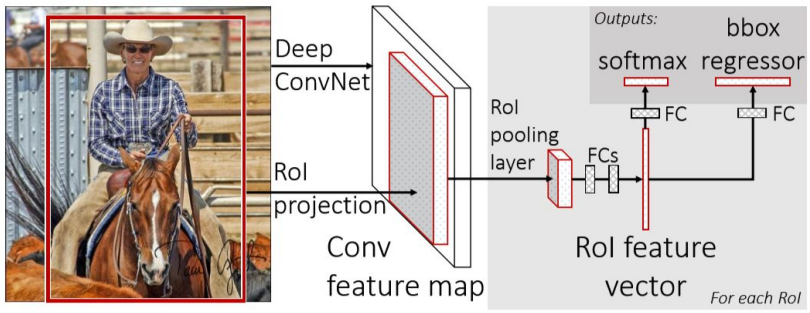


Fig. 11

RoI max pooling, as the name depicts, works by dividing the hxw RoI window into HxW grid of smaller windows, taking the max from each window to represent the value of the respective grid cell.

Fast R-CNN can be initialized with a pre-trained network (the CNN layer), When this is done, three modifications are done to it. First, the last layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net’s first fully connected layer (e.g., H = W = 7 for VGG16 - from the paper). Second, the network’s last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling layers described earlier(a fully connected layer and softmax over K +1 categories and category-specific bounding-box regressors). The network is also modified to take two data inputs: a list of images and a list of RoI’s in those images. In Fast R-CNN training, stochastic gradient descent (SGD) minibatches are sampled hierarchically, first by sampling N images and then by sampling R/N RoIs from each image. Critically, RoIs from

the same image share computation and memory in the forward and backward passes. Making N small decreases mini-batch computation.

A Fast R-CNN network has two sibling output layers. The first outputs a discrete probability distribution (per RoI), $p = (p_0, \ldots, p_K)$, over K + 1 categories. p is computed by a softmax over the K+1 outputs of a fully connected layer. The second sibling layer outputs bounding-box regression offsets, $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$, foreach of the *K* object classes, indexed by k. The parametrization for t^k is calculated as described in the R-CNN paper (previously discussed), in which t^k specifies a scale-invariant translation and log-space height/width shift relative to an object proposal. Each training RoI is labeled with a ground-truth class u and a ground-truth bounding-box regression target v. A multi-task loss L on each labeled RoI is used to jointly train for classification and bounding-box regression, as shown below.

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v)$$

where $L_{cls}(p,u) = -\log(p_u)$ is log-loss for a true class (u) and L_{loc}, is defined over a tuple of true bounding box regression targets for class *u*, $v = (v_x, v_y, v_w, v_h)$, and a predicted tuple $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$. The conditional operator, [u ≥ 1] is one, when u ≥ 1, else it is zero. Background classes are labelled with u=0. For bounding-box regression, L_{loc} is calculated as,

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t^u_i - v_i)$$

Where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

The hyper-parameter λ (above) controls the balance between the two losses. During the backpropagation; assume, for a given image; let $x_i \in R$ be the i-th activation input into the RoI pooling layer and let y_{rj} be the layer's j-th output from the rth RoI. The RoI pooling layer computes $y_{rj} = x_{i*(r,j)}$, in which $i*(r,j) = \text{argmax}_{xi' \in R(r,j)} x_{i'}$. R(r,j) is the index set of inputs in the sub-window over which the output unit y_{rj} max pools. A single xi may be assigned to several

different outputs y_{rj}. During backpropagation, the values are updated by computing the partial derivative of the loss function with respect to each input variable x_i by following the argmax function (as explained above),

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] \frac{\partial L}{\partial y_{rj}}$$

Hence, for each mini-batch RoI (*r*) and for each pooling output unit y_{rj}, the partial derivative $\partial L/\partial y_{rj}$ is accumulated if i is the argmax selected for y_{rj} by max pooling. In back-propagation, the partial derivatives $\partial L/\partial y_{rj}$ are already computed by the backwards function of the layer on top of the RoI pooling layer.

Faster R-CNN

The region proposal in Fast R-CNN[4] is still a bottleneck that is resolved in Faster R-CNN. In Fast R-CNN, these proposals were created using Selective Search, which is a fairly slow process. Faster R-CNN differs from Fast R-CNN in that it uses the feature maps generated by the CNN layer for region proposal, instead of running a separate Selection Search on the image. This architecture requires only input and enables one to reuse the CNN to generate the ROI. The output is an image with bounding boxes and class labels. For the region proposals, Faster R-CNN uses a Fully Convolutional Network. The region proposals are used by Fast R-CNN to generate the masks and bunding boxes.

Region Proposal Networks are to be designed in such a way that allows fast proposals that are accurate and precise, while spanning wide scales of aspect ratios. Faster R-CNN does not use use FPN's, instead uses 'anchor boxes' to identify objects in common aspect ratios.

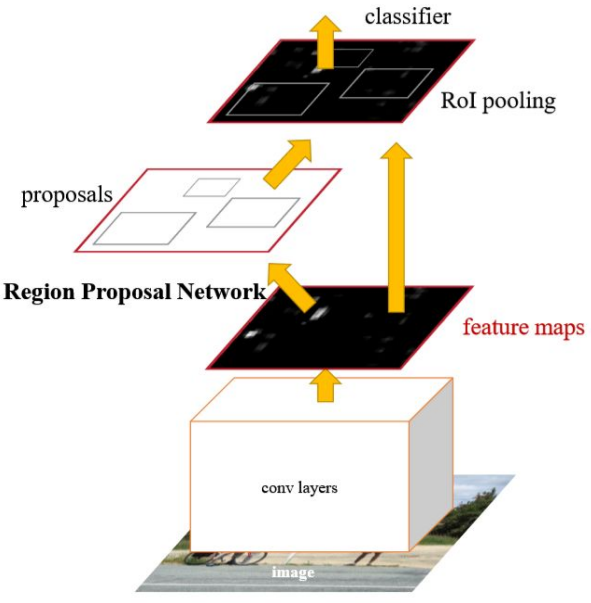


Fig. 12

Faster R-CNN uses a training scheme that alternates between fine-tuning for the region proposal task and then fine-tuning for object detection, while keeping the proposals fixed. This alternation between tasks allows for quick convergence while sharing the CNN features.

To generate region proposals, a small network slides over the convolutional feature map output (conv layers' output). This small network takes as input an n × n spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature. This feature is fed into two sibling fully connected layers—a box-regression layer (reg) and a box-classification layer (cls), as shown in Fig. 13.

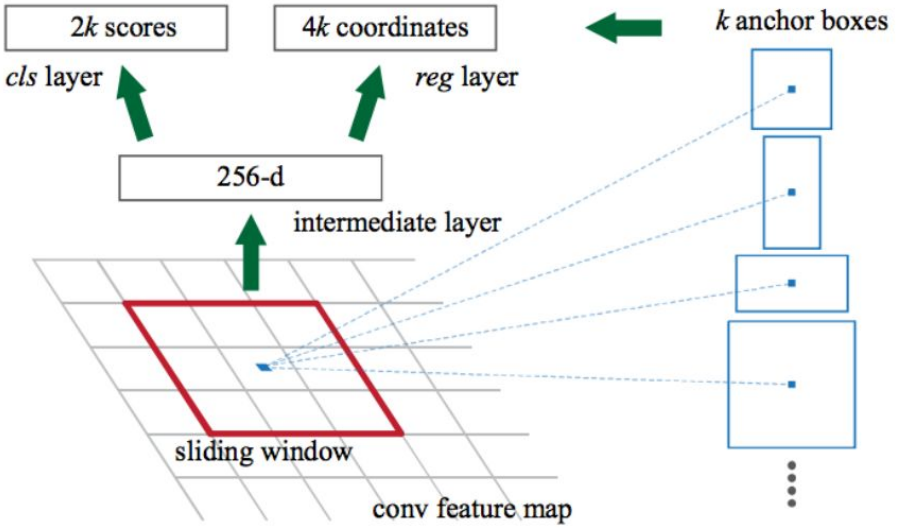


Fig. 13

At each sliding window iteration, k anchor boxes are generated (by the *reg*), each of which has 4 coordinates (2 for the top left coordinate, 2 for height and the width). The *cls* can be thought of as a classification layer, with the two classes being denoting the object presence (object present, object absent), which generates 2 probabilities respectively, for each anchor box. In total, we have 4k parameters for the box coordinates, each with its respective probabilities.

During the RPN training, a positive label is assigned to two kinds of anchors: (i) the anchor/anchors with the highest Intersection over Union (IoU) overlap with a ground-truth

box, or an anchor that has an IoU overlap higher than 0.7 with 5 any ground-truth box. A negative label is assigned to an anchor box if its IoU is less than 0.3 for all ground-truth boxes.

The loss is calculated as follows (below), where *i* is the index of an anchor with p_i being the associated predicted probability that the anchor encloses an object. The value of p_i^* is 1 if the anchor contains an object, else it is zero. t_i and t_i^* are the 4-coordinate vectors for the anchor and the ground truth box respectively. L_{cls} is the log loss over the two classes.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*).$$

For the regression loss L_{reg} , Robust loss is used as R(t_i - t_i^*), as defined in Fast R-CNN. The normalization parameters N_{reg} and N_{cls} are chosen to be the number of anchor locations and min-batch size. The regression loss for the anchor boxes are calculated as,

$$\begin{aligned} t_{\mathbf{x}} &= (x - x_{\mathbf{a}})/w_{\mathbf{a}}, \quad t_{\mathbf{y}} = (y - y_{\mathbf{a}})/h_{\mathbf{a}}, \\ t_{\mathbf{w}} &= \log(w/w_{\mathbf{a}}), \quad t_{\mathbf{h}} = \log(h/h_{\mathbf{a}}), \\ t_{\mathbf{x}}^* &= (x^* - x_{\mathbf{a}})/w_{\mathbf{a}}, \quad t_{\mathbf{y}}^* = (y^* - y_{\mathbf{a}})/h_{\mathbf{a}}, \\ t_{\mathbf{w}}^* &= \log(w^*/w_{\mathbf{a}}), \quad t_{\mathbf{h}}^* = \log(h^*/h_{\mathbf{a}}), \end{aligned}$$

where x_a, y_a represent the coordinates of the top left corner of the anchor box and the asterisked quantities, x^*, y^* denote the top left coordinates of the ground truth boxes. Similarly, the *h, w* which stand for the height, width and height of the anchor and ground truth boxes. The RPN is trained end-to-end by backpropagation and stochastic gradient descent.

For the CNN; the RPN and Fast R-CNN, trained independently ad they modify their convolutional layers in different ways, so a combined way is required to train the CNN layers.

This can be achieved in 3 ways.

- First train the RPN and use the proposals to train Fast R-CNN. The network tuned by Fast R-CNN is then used to initialize RPN, and this process is iterated.
- RPN and Fast R-CNN networks are merged into one network during training. In each SGD iteration, the forward pass generates region proposals which are treated just like fixed, pre-computed proposals when training a Fast R-CNN detector. The backward propagation takes place as usual, where for the shared layers the backward propagated signals from both the RPN loss and the Fast R-CNN loss are combined. But this method is approximate as it ignores the derivative w.r.t. the proposal boxes' coordinates, which are also network responses.

To train the network, the RPN is trained as described above, with the 'conv layer' CNN initialized with the weights of ImageNet. Fast R-CNN network is trained using the proposals from RPN, but initialized with ImageNet (the CNN layer of Fast R-CNN is initialized with ImageNet, further training with the proposals from RPN trained as described above). Now, the conv-layers can be combined. Since the 'conv-layers' in both RPN and Fast R-CNN are more or less same, it is combined and trained, but this time, modifying (training) only the layers unique to Fast R-CNN. The last training iterations can be repeated, or the individual steps be trained better.

Mask R-CNN

The major motivation behind Mask R-CNN is given that Faster R-CNN works well for detecting bounding boxes for objects, can it be improved to detect masks. Based on previous work of improving on the previous established architecture, Mask R-CNN builds on top of Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression (Fig. 14). The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel to pixel manner. A modification was required to make Mask R-CNN be able to predict the masks accurately, precisely on images. To achieve this, RoI Pool (that approximates scaled-down regions) was replaced with RoI Align, that faithfully re-maps the masks from the scaled down regions (as a consequence of CNN's convolution

operations) to the original image's pixel coordinates using bilinear interpolation. Another difference is that Mask R-CNN decouples the mask predictions and the class prediction.

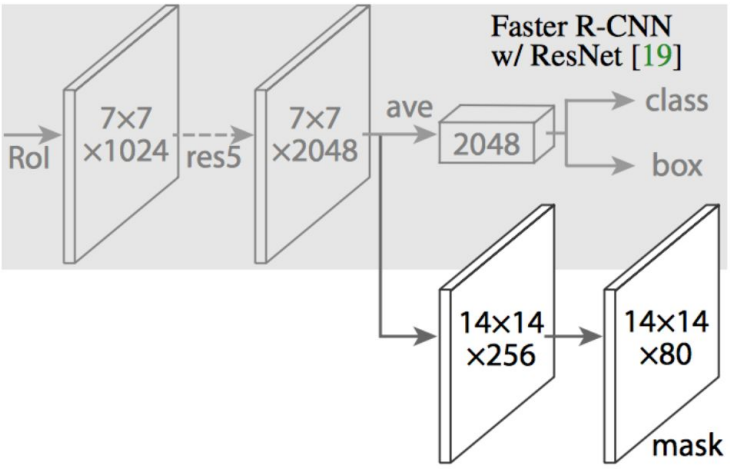


Fig. 14

In Mask R-CNN, a binary mask is predicted for each class independently, without competition among classes, and rely on the network's RoI classification branch to predict the category. In contrast, typical FCNs usually perform per-pixel multi-class categorization, which couples segmentation and classification, which leads to lower performance (experiments from the paper[5]).

The loss function is defined as $L = L_{cls} + L_{box} + L_{mask}$ where $L_{cls} + L_{box}$ represent the same functions as seen in Faster R-CNN, with L_{mask} being added to the loss, to describe the loss due to the mask prediction. The mask branch has a K_m^2 dimensional output for each RoI, which encodes K binary masks of resolution m×m, one for each of the K classes. To this, a per-pixel sigmoid function is applied, and L_{mask} is defined as the average binary cross-entropy loss. For an RoI associated with a ground-truth class k, L_{mask} is only defined on the k-th mask which means that other mask outputs do not contribute to the loss. This allows the network to generate masks for every class without competition among classes.

RoI Align is the major feature that enabled Mask R-CNN to generate faithful masks. The idea behind RoI Align layer is that it removes the harsh quantization of RoI Pool, by properly aligning the extracted features with the input.

To Achieve this, any quantization of the RoI boundaries or bins (i.e., we use x/16 instead of [x/16]) is avoided. Instead, bilinear interpolation is used to compute the exact values of the input features at four regularly sampled locations in each RoI bin, and this result is aggregated (using max or average).

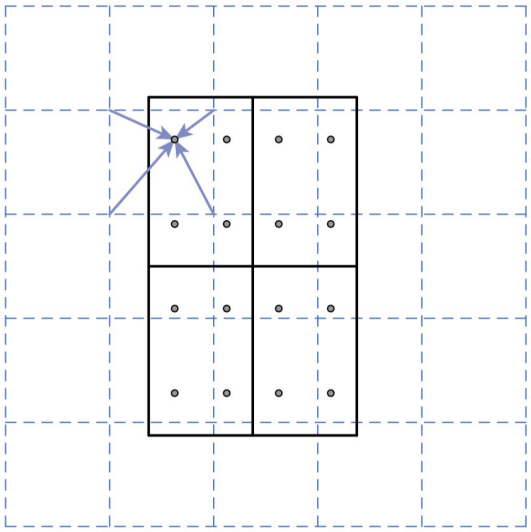


Fig. 15

The dashed grid (Fig. 15) represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the near-by grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points

For example, assume the situation as shown to the right. Assume an image of size 128x128 and a feature map of size 25x25. If we require features of the region corresponding to the top-left 15x15 pixels in the original image, we know each pixel in the original image corresponds to 25/128 pixels

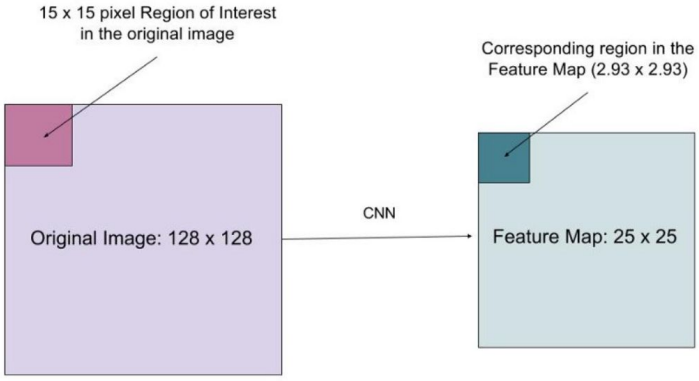


Fig. 16

in the feature map. To select 15 pixels from the original image, we just select 15 * 25/128 ~= 2.93 pixels. In RoI Pool, that value is binned or approximated to 2. Instead, we use bilinear interpolation to get a precise idea of what would be at pixel 2.93. This, at a high level, is what allows us to avoid the misalignments caused by RoIPool. Once these masks are generated, Mask R-CNN combines them with the classifications and bounding boxes

from Faster R-CNN. All the above accumulated effort has led us to Mask R-CNN, that can accurately predict masks and bounding boxes for images, while maintaining reasonable performance to the extent that it is almost real-time (performance of about 5 fps on modern hardware is expected)

TACC

We worked on TACC (Texas Advanced Computing Center) to train Mask R-CNN with the kaggle cell data to try and extract features from the cancer cell data. We were given access to Maverick2, Stampede2 but primarily worked on Maverick2 as it has GPUs to accelerate the training process. But before we could run the training algorithms, we ran into a lot of issues.

Maverick2 was not in sync with the latest machine learning packages and frameworks, so we had to experiment and figure out a way to run the ML tools on Maverick2. We tried building Caffe2, Tensorflow from source, but they required various dependencies that were out of our control to update and upgrade. So, we resorted to building their dependencies which quickly led us to building roughly 15 packages (to name a few, GPU support for the ML tools in the form of CUDA, CUDNN, Nvidia drivers support packages; Glib-C, Tensorflow, Python 3.6, GCC>4.3 with its dependencies as BLAS, ATLAS, MPFR, MPC, GMP) from source, each with its own configuration problems, dependency conflicts, requirements. Solving these issues, testing them and installing them with the proper access permissions for the whole team took us about 2 weeks, after which we had GCC 7, Python 3.6, Anaconda, Tensorflow 1.2 (that was the last version supporting the outdated GPU drivers on Maverick), with working GPU acceleration.

Results

We trained the MaskRCNN on the coco dataset and ran a forward pass for the cancer cell data (Example below).

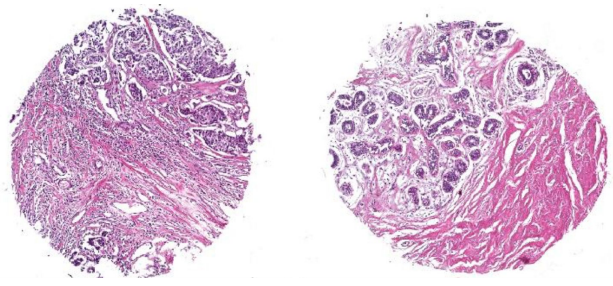


Fig. 17

This was done to see if the MaskRCNN can identify small features in the stain data without the need of any additional training. The result obtained was as follows

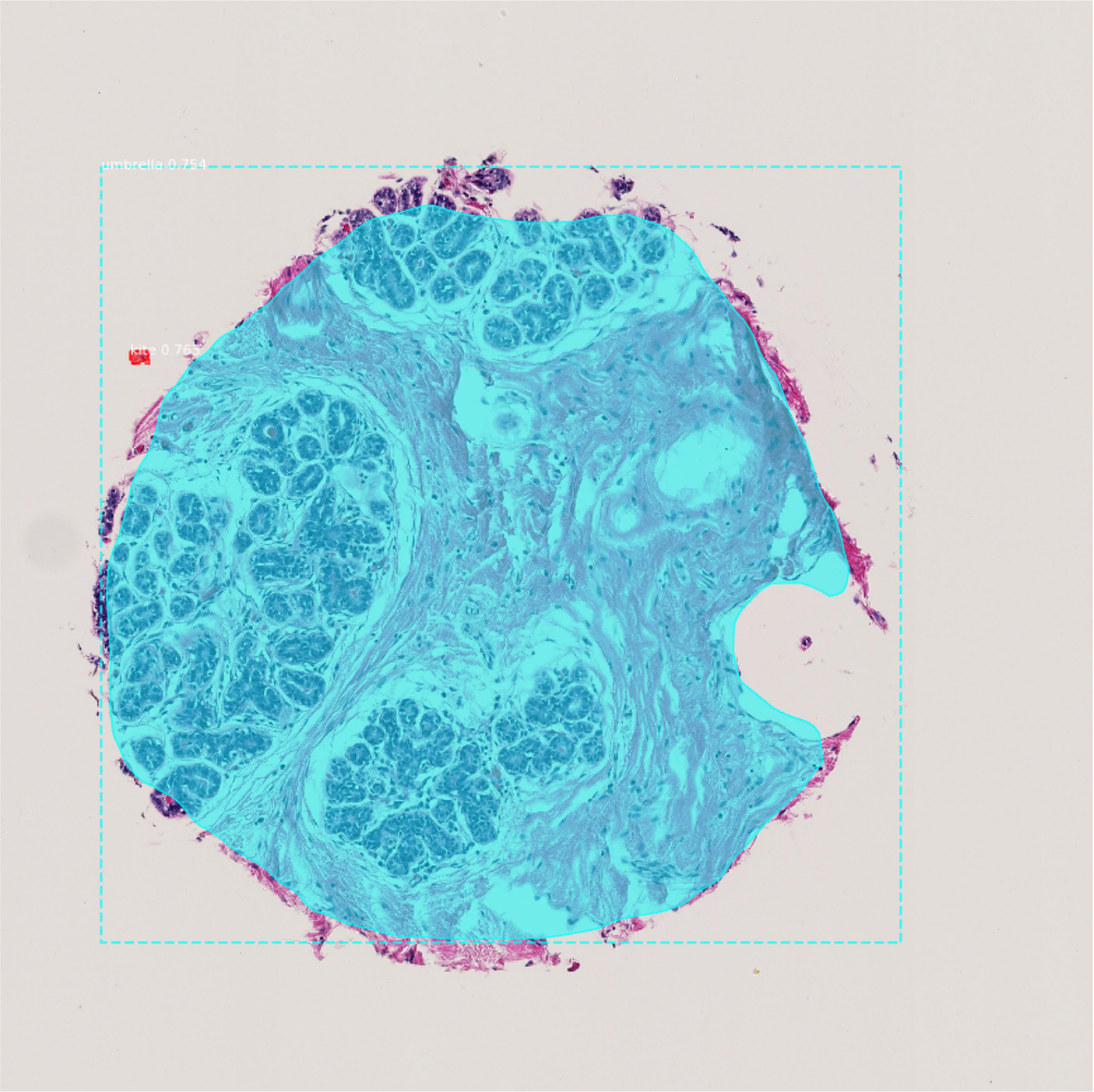


Fig. 18

The cell was identified as an umbrella and a small feature towards the top left was identified as a kite.

Due to the lack of training data for the cancer cell, we decided to train the MaskRCNN with nucleus data obtained from kaggle[6]. The detection of the nuclei could be helpful for the

detection of the carcinogenic features due to a similarity in the shape and it being biological in nature.

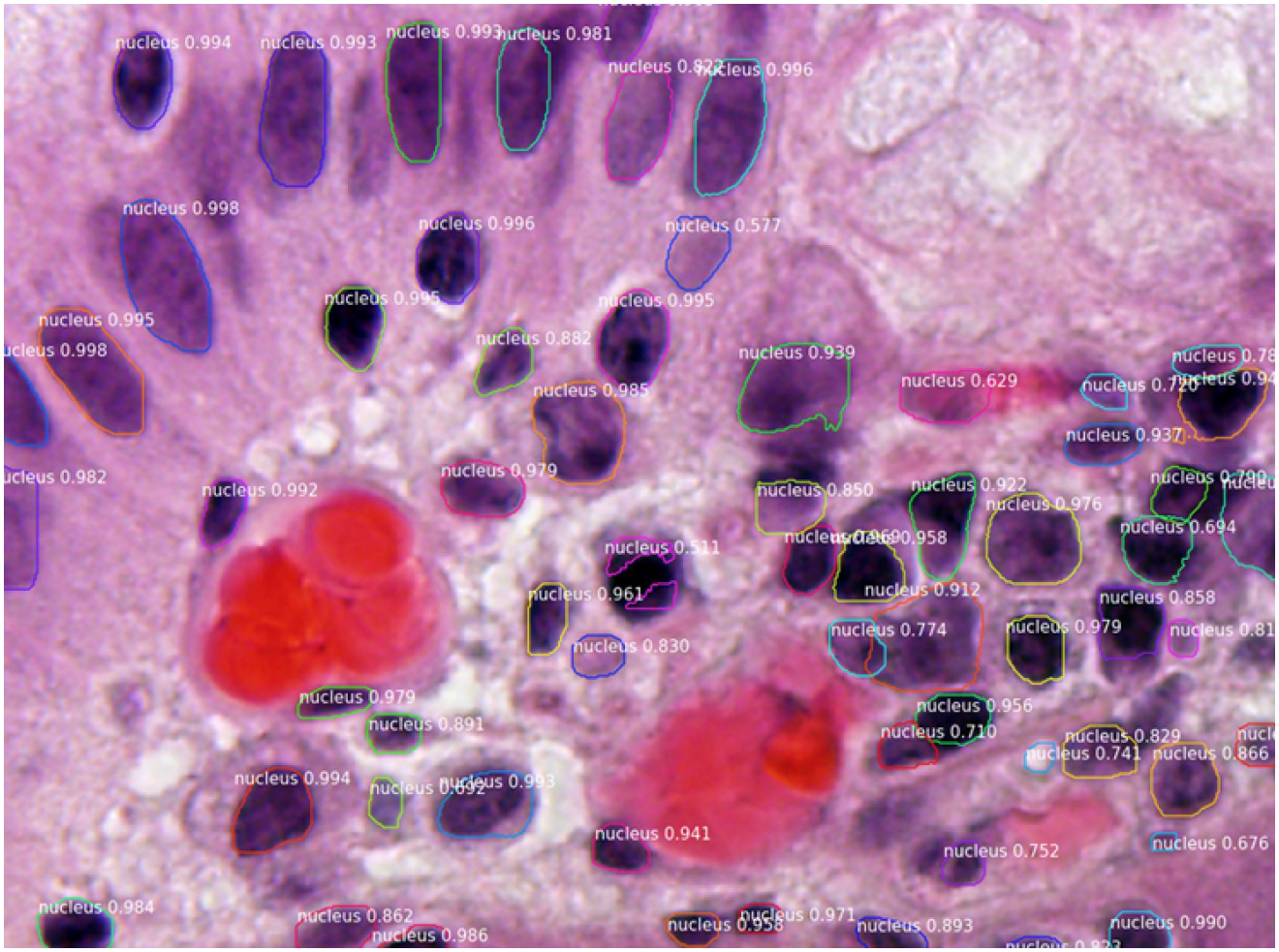


Fig. 19: Nucleus data used for training the MaskRCNN (760 × 572 pixels)

The MaskRCNN was trained with the nucleus data but there was an error with logging the weights. Therefore the results are not available.

Segment Everything

Segment Everything (Learning to Segment Object Candidates: paper[7]) takes a novel approach to detecting and classifying objects by using a class-agonistic classifier to identify all possible patches in an image that might contain objects and then classifying them. This method aims to extract patches while maintaining high accuracy and speed. Three characteristics, high recall, i.e, capturing as many objects as possible in a patch, identify as many regions as possible and accuracy of detection. To achieve this, instead of using low-level features such as edges, superpixels and the similar, this algorithm (or rather the architecture) uses a data driven approach and uses deep-networks to obtain the segmentation proposals. This not only speeds up the architecture generation, but also helps in generalizing it as a whole, so that when a new problem is presented, only the training step is necessary.

Segment everything finds the segmentation masks for 3000 kinds objects in the image. Mask RCNN contains a region proposal network that generates bounding boxes for 80 classes. Bounding boxes are created from the segmentation masks that are generated by Segments Everything and pixelwise labelling is done for the bounding boxes. Therefore, the intersection of these bounding boxes is taken with the bounding boxes generated by the masked RCNN. The resulting boxes are are resized to 14x14x3000. These are again embedded to the size of 14x14x300. This is concatenated to the 14x14x256 features of the bounding boxes from the Mask RCNN. Then the Mask RCNN re-trained and the results are evaluated. This gives us contextual information as to whether adding extra semantic information helps in the detection of masks by the RCNN.

Using MPEG to segment data

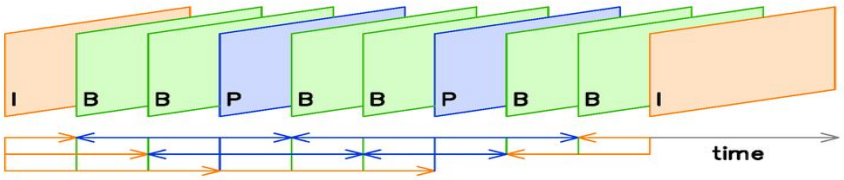
MPEG can be used to transfer the labels on unlabelled neuropil data by using MPEG. It extracts the information from the labelled data and labels the data on new images if similar patterns/ structures are found hence taking advantage of the homogeneity of the data. In this way, a new dataset can be labelled.

MPEG

The MPEG-2 bit stream is basically just a series of coded frames one after the other. It is based on data redundancies like spatial and time based. It mainly consists of three types of frames: I,P and B.

Frame Sequencing

The ordering of I, P, and B frames is fairly flexible in MPEG-2. The first frame must always be an I frame because there is no other data for a P or B frame to reference. After that, I, P, and B frames can be mixed in any order.



Coding a frame in MPEG-2 format always begins by representing the original color frame in YCbCr format. The Y component represents luminance, while Cb and Cr represent chrominance differences. The three components of this color model are mostly uncorrelated so this transformation is a useful first step in reducing redundant information in the frame. Encoding RGB into YCbCr color space reduces the mutual redundancies.

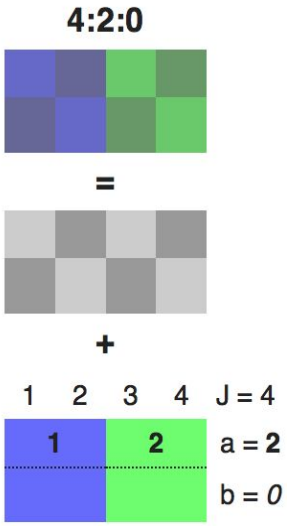
Conversion:
Y = 0.299*R +0.587*G +0.114*B
Cb = 128 - 0.168736*R - 0.331264*G + 0.5*B
Cr = 0.5*R - 0.418688*R - 0.081312*B
R, G and B values are 8 bit long and lies in the range of 0 and 255.

As Human eyes are much more sensitive to luminance than chrominance, so it is common to subsample both chrominance channels. The most commonly used subsampling format is denoted by 4:2:0, which means that chrominance sampling is decimated by 2 in the horizontal and vertical direction. Both chrominance channels are reduced to one quarter the original data rate in this way and the net effect is for total frame data rate to be cut in half with hardly any perceptual effect on image quality. It is illustrated in the following image.

J: horizontal sampling reference (width of the conceptual region). Usually, 4.

a: number of chrominance samples (Cr, Cb) in the first row of *J* pixels.

b: number of changes of chrominance samples (Cr, Cb) between first and second row of *J* pixels.



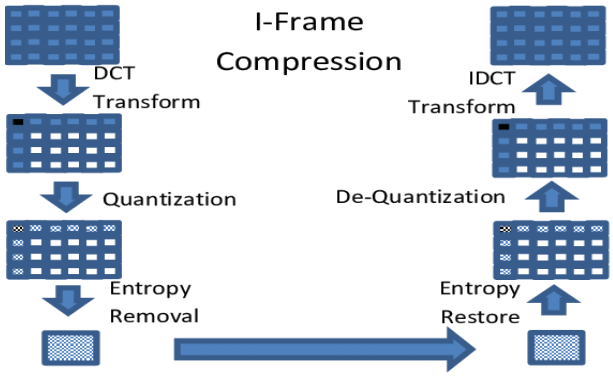
MPEG-2 uses block based coding. This means that a frame is not encoded as a whole; it is divided into many independently coded blocks. A macroblock is 16x16 pixels and is a basic unit of MPEG-2 coding. However, each macroblock is further divided into 8x8 pixel *blocks*. This results in 6 blocks per macroblock -- 4 for luminance and 2 for chrominance (assuming 4:2:0 chroma subsampling).

These block sizes were chosen in part because small sections of a frame of natural video (not computer generated or edited) are likely to be correlated. This correlation helps the next stages of encoding work more efficiently.

Types of Frames in MPEG 2

The next encoding step can vary from frame to frame. There are actually three possible types of frames, called I, P, and B frames.

I frame coding



An I frame is intra or spatially coded so that all the information necessary to reconstruct it is encoded within that frame's segment of the MPEG-2 bit stream. It is a self contained image compressed in a manner similar to a JPEG image. It is usually the reference frame in frame sequencing.
Each block of the frame is processed independently with an 8x8 discrete cosine transform (DCT). This transform generates a representation of each 8x8 block in the frequency domain instead of the spatial domain. Since, as noted above, the pixels in each block of a natural video are likely to be correlated, the resulting DCT coefficients typically consist of a few large values and many small values. The relative sizes of these coefficients represent how important each one is for reconstructing the block accurately.

The general equation for a 2D (*N* by *M* image) DCT is defined by the following equation:

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i).\Lambda(j).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] cos\left[\frac{\pi.v}{2.M}(2j+1)\right].f(i,j)$$

and the corresponding ***inverse*** 2D DCT transform is $F^{-1}(u,v)$, i.e. :

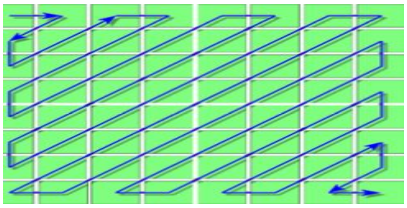
$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

The next step in the process which is Quantization, reduces the amount of information in higher frequency DCT coefficient components using a default quantization matrix defined by MPEG-2 standard

QDCT = round (8 * DCT / Scale * Qi)

where DCT and QDCT represent the coefficient matrices before and after quantization, scale is used to determine quality, and Qi is a perceptually weighted quantization table. Here, a default preset value of Qi is taken.
weights low frequency content higher that high frequency content, mimicking the response characteristics of a human eye. The quality scalar, scale, can be any value from 1 to 31. If desired, MPEG-2 allows the value of scale to be mapped non-linearly to higher compression levels, with a maximum of 112.
This quantization process is lossy because the true floating point values of the DCT coefficients are not preserved. Instead, the quantized coefficients are just an approximation of the true coefficients and the quality of the approximation determines the quality of the frame reconstructed from this bit stream.
Next the 8x8 block of quantized coefficients is arranged into a vector by indexing the matrix in orders given as follows

Run length amplitude/ Variable length encoding is a lossless compression techniques that includes entropy encoding, run-length encoding and Huffman encoding.



Entropy Encoding is a technique where the components of quantized DCT coefficient matrix is read in zigzag order. It is illustrated in the diagram above.

Run Length Encoding is a lossless data compression technique of a sequence in which same data value occurs in many consecutive data elements.

Huffman Encoding is a lossless data compression technique that uses variable length code table for encoding a source symbol. Where, Variable length code table is derived based on the estimated probability of occurrence of each possible value of the source symbol.

For example - MPEG-2 has special Huffman code word (i.e. EOB) to end the sequence prematurely when the remaining coefficients are zero and then it performs variable length encoding for further compression.

char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010

This ordering is approximately low frequency to high frequency for a very important reason small blocks (8x8) of natural video are likely to contain mostly low frequency content. This ordering tends to group non-zero terms at the front of the vector and zero terms at the end.

P Frame Coding

A P frame is inter or temporally coded. That means it uses correlation between the current frame and a past frame to achieve compression. The MPEG-2 standard dictates that the past frame must be an I or P frame, but not a B frame.
Temporal coding is achieved using motion vectors. The basic idea is to match each macroblock in the current frame with a 16x16 pixel area in the past reference frame as closely as possible. Closeness here can be computed in many ways, but a simple and common measure is sum of absolute differences (SAD). The offset from the current macroblock position to the closest matching 16x16 pixel area in the reference frame is recorded as two values: horizontal and vertical motion vectors.

Searching Algorithm

The search to find the best motion vectors is performed in the luminance channel only. Whatever motion vector is found then applies to all three channels of that macroblock.

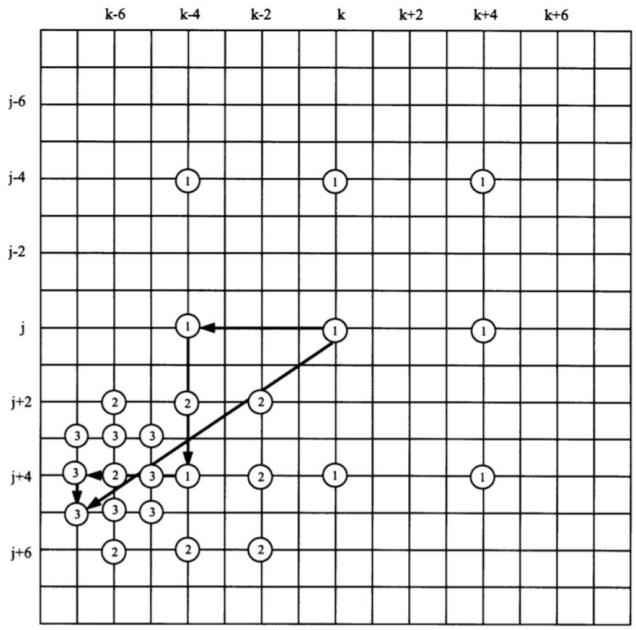
Finding the best-matching block requires optimizing the matching criterion over all possible candidate displacement vectors at each pixel. The full-search logarithmic search, and hierarchical searching algorithms can accomplish this.

Full search

The full-search algorithm evaluates the matching criterion for all possible values within the predefined searching window. If the search window is restricted to a [–*p, p*] square, for each motion vector there are (2*p* + 1)2 search locations. For a block size of *M*N* pixels, at each search location we compare *N*M* pixels. If we know the matching criterion and how many operations are needed for each comparison, then we can calculate the computation complexity of the full-search algorithm. Full search is computationally expensive, but guarantees finding the global optimal matching within a defined searching range.

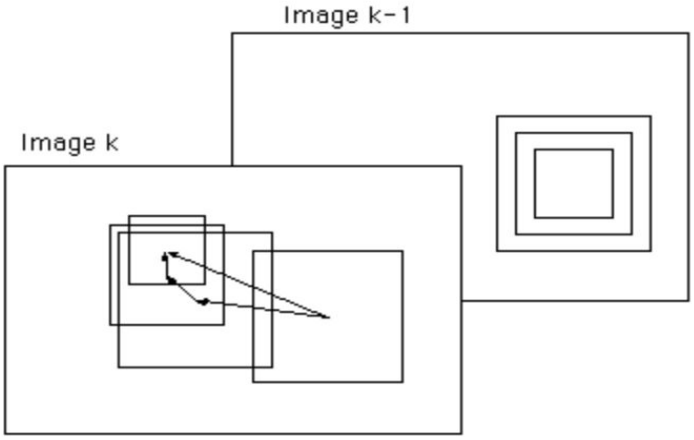
Logarithmic search

Actually, the expected accuracy of motion estimation algorithms varies according to the applications. In motion-compensated video coding, all one seeks is a matching block in terms of some metric, even if the match does not correlate well with the actual projected motion. Therefore, in most cases, search strategies faster than full searches are used, although they lead to suboptimal solutions. These faster search algorithms evaluate the criterion function only at a predetermined subset of the candidate motion vector locations instead of all possible locations. One of these faster search algorithms is the logarithmic search. Its more popular form is referred to as the three-step search. We explain the three-step search algorithm with the figure below where only the search frame is depicted. Search locations corresponding to each of the steps in the three-step search procedure are labeled 1, 2, and 3. In the first step, starting from pixel 0 we compute MAD for the nine search locations labeled 1. The spacing between these search locations here is 4. Assume that MAD is minimum for the search location (4,4) which is circled 1. In the second step, the criterion function is evaluated at eight locations around the circled 1 which are labeled 2. The spacing between locations is now 2 pixels. Assume now the minimum MAD is at the location (6,2), which is also circled. Thus, the new search origin is the circled 2, which is located at (6.2). For the third step, the spacing is now set to 1 and the eight locations labeled 3 are searched. The search procedure is terminated at this point and the output of the motion vector is (7,1). Additional steps may be incorporated into the procedure if we wish to obtain subpixel accuracy in the motion estimations. Then, the search frame needs to be interpolated to evaluate the criterion function at subpixel locations.



Hierarchical Block Matching:

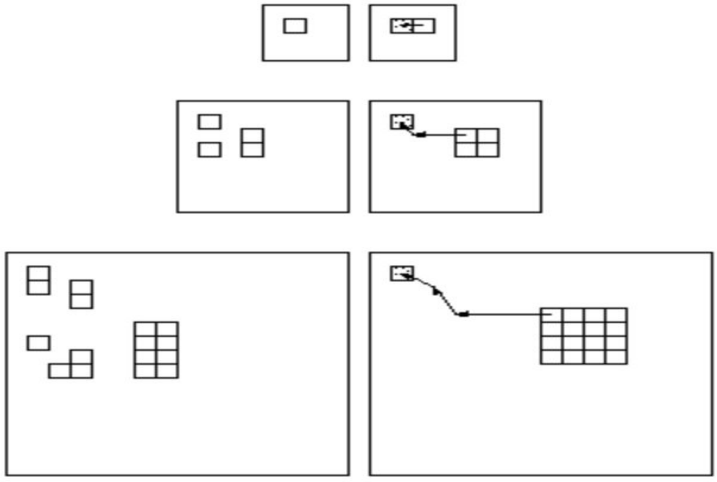
Hierarchical block matching techniques attempt to combine the advantages of large blocks with those of small blocks. The reliability of motion vectors is influenced by block size. Large blocks are more likely to track actual motion than small ones and thus are less likely to converge on local minima. Although such motion vectors are reliable the quality of matches for large blocks is not as good as that for small blocks. Hierarchical block matching algorithms exploit the motion tracking capabilities of large blocks and use their motion vectors as a starting points for searches for small blocks.



A three level hierarchical search. Initially large blocks are matched and the resulting motion vector provides a starting point for a search for a smaller matching block.Bierling developed a hierarchical block matching algorithm that started by matching a large target block with a similar sized block in the past frame. The resulting motion vector is then used as an estimate for a search using smaller target and candidate blocks . At each stage the resulting estimate was used as a starting point for the next stage of the search, as illustrated above. The size of the block was reduced and a match found at each stage of the algorithm, until the matching blocks were the desired size.

It is easy to see how this algorithm might match true motion more easily than those that use only small blocks. But the algorithm is very computation intensive. This computational burden can be eased by modifying the algorithm in three ways.In the early stages, when the target blocks are largest, the candidate motion vectors could be coarse. That is, only a subset of the possible motion vectors in the search space need be considered, as in Gilge's algorithm. Although, the initial estimates might not be ideal, the algorithm would have the necessary flexibility in later stages to make up for this.In addition, the matching criteria could use only a sub-sample of the pixels in the large blocks, thus reducing the number of pixels that need to be compared. For example, only pixels from every second row and column from both the target and candidate blocks might be compared. It is unlikely that such a modification would adversely affect the performance of the algorithm but it would preserve the advantages of using large blocks at the early stages.

A third modification could allow Bierling's algorithm to make better use of the information gathered during each stage. If the size of the blocks for each stage was chosen such that several smaller blocks could fit into one large block, then the approximation generated at one stage can be applied to several blocks during the subsequent stage, instead of just one.



Progressive refinement of a motion vector. The motion vector for each block is applied to four corresponding blocks in the layer below.Combining all three modifications results in a more popular block matching technique that uses image pyramids and block decomposition.

The problem

There exists many choices for the macroblock coding mode under the MPEG-2 standard for P- and B-pictures, including intramode, no-motion-compensation mode, etc.Given below is one of the practical approach to select the best mode.

Practical solution with new criteria for the selection of coding mode:

It is obvious that the near-optimal solution discussed in the previous section is not a practical method because of its complexity. To determine the best mode, we have to know how many bits it takes to code each macroblock in every mode with the same distortion level. The total number of bits for each macroblock, RMB, consists of three parts, bits for coding motion vectors, Rmv, bits for coding the predictive residue, Rres, and bits for coding macroblock header information, R_{header}, such as macroblock type, quantizer scale, and coded-block pattern.

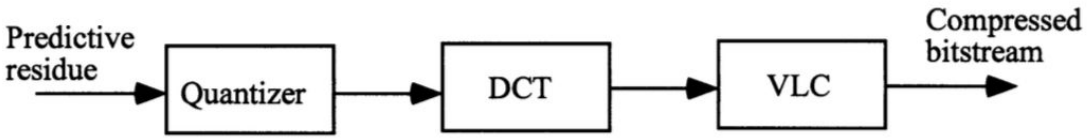
$$RMB = R_{mv} + R_{res} + R_{header}$$

The number of bits for motion vectors, Rmv, can be easily obtained by VLC table lookup. But to obtain the number of bits for coding the predictive residue, one has to go through the three step coding procedure: (1) DCT, (2) quantization, and (3) VLC as shown in figure

below. At step 3, Rres is obtained with a lookup table according to the run length of zeros and the level of quantized coefficients, i.e., Rres depends on the pair of values of run and level:

$$R_{res} = f(run, level).$$

To obtain a much less computationally intensive method, it is preferred to use a statistical model of DCT coefficient bit usage vs. variance of the prediction residual and quantizer step size. This will provide an approximation of the number of residual bits, Rres. For this purpose we assume that the run and level pair in equation above is strongly dependent on values of the quantizer scale, qs, and the variance of the residue, Vres, for each macroblock.



Intuitively, we would expect the number of bits to encode a macroblock is proportional to the variance of the residual and inversely proportional to the value of quantizer step size. Therefore, a statistical model can be constructed by plotting *Rres* vs. the independent variables *Vres* and *qs* over a large set of representative macroblock pixels from images typical of natural video material. This results in a scatter plot showing tight correlation, and hence a surface can be fit through the data points. It was found that equation below can be approximately expressed as follows:

$$R_{res} \approx f(q_s, V_{res}) = \left(K / \left(C q_s + q_s^2\right)\right) V_{res},$$

where *K* and *C* are constants found through surface-fitting regression. If we assume *Rheader* is a relatively fixed component that does not vary much with macroblock coding mode and can be ignored, the above equation can be approximately replaced by:

$$R_{MB'} = R_{mv} + \left(K / \left(C q_s + q_s^2\right)\right) V_{res}.$$

The value of *RMB* reflects the variable portion of bit usage that is dependent on coding mode, and can be used as the measure for selecting the coding mode in our encoder. For a given quantizer step size, the mode resulting in the smallest value of *RMB* is chosen as the "best" mode.

In this type of encoding, motion vectors are a simple way to convey a lot of information, but are not always a perfect match. To give better quality reconstruction, error between the actual macroblock and the predicted macroblock is then encoded. This coding of the

residual is almost exactly the same as I frame coding. In fact the only difference is that the quantization equation becomes

QDCT = floor (8*DCT / Scale * Q_p)

where floor is used in place of round, and Q_p is a different weighted quantization table. The default for Q_p is

Qp = [16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]
[16 16 16 16 16 16 16 16]

During reconstruction, the residual is decoded and added to the motion vector predicted macroblock.

B Frame Encoding:

A B frame is simply a more general version of a P frame. Motion vectors can refer not only to a past frame, but to a future frame, or both a past and future frame. Using future frames is exactly like a P frame except for referencing the future. Using past and future frames together works by averaging the predicted past macroblock with the predicted future macroblock. The residual is coded like a P frame in either case.

Conclusion:

Using the above encoding techniques, MPEG can increase the storage efficiency, stores some frames from which the motion vectors between them are calculated. This allows for reducing the actual information that is stored to represent the original media. Given a sequence of images (labelled stain or neuropil data), with the first one labeled, one can extract the motion vectors which can be used to transfer the labels and boundaries to the next frames (images) in the sequence.

MATLAB CODE

Helper functions
Conversion.mat
convertYuvToRgb.m - Used by loadFileYuv.m and loadFileY4m.m to convert YUV data to RGB
figuresc.m - Easily create a non-standard sized figure
playlast.m - Load and play the last encoded video
quiverplot.m - Show motion vectors of P frames

Process:
A yuv file is taken as input and converted into matlab file using loadFileYuv.m
The generated .mat file is processed using mpegproj.m main function where encoding and decoding is done.

Results:

The encoded, decoded frames and the input .mat file are merged into loadmov.mat.

The following link contains the code and also loadmov.mat file, which is the output obtained:

https://www.dropbox.com/sh/gk3sa6e2kswk8i1/AAAgsQ8_vzDuZGNOL3TENkYLa?dl=0

References:

[1] - Image Net Classification with Deep Convolutional Neural Networks

[2] - RCNN: Rich feature hierarchies for accurate object detection and semantic segmentation

[3] - Fast Region-based Convolutional Network method

[4] - Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

[5] - MaskR-CNN: Facebook AI Research

[6] - Kaggle Dataset

[7] - Learning to Segment Everything

[8] - Image and video compression for multimedia engineering Philip A.Laplante

Appendix:

Huffman coding example:

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.

2. Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

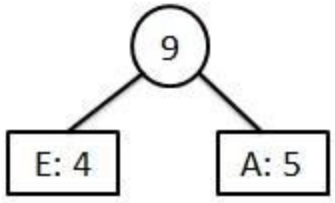
Value	A	B	C	D	E	F
Frequency	5	25	7	15	4	12

Solution:

Step 1: According to the Huffman coding we arrange all the elements (values) in ascending order of the frequencies.

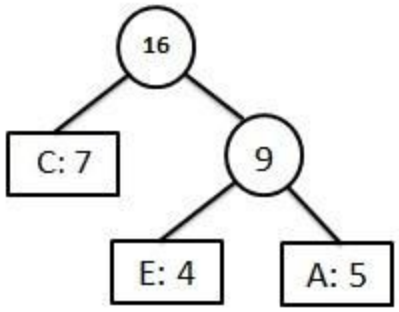
Value	E	A	C	F	D	B
Frequency	4	5	7	12	15	25

Step 2: Insert first two elements which have smaller frequency.



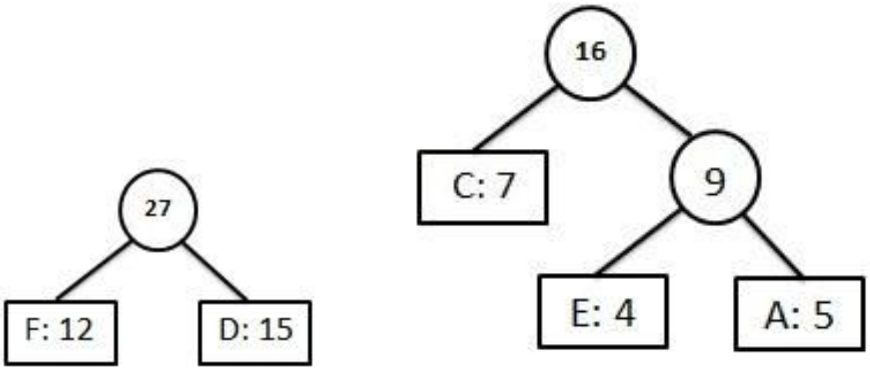
Value	C	EA	F	D	B
Frequency	7	9	12	15	25

Step 3: Taking next smaller number and insert it at correct place.



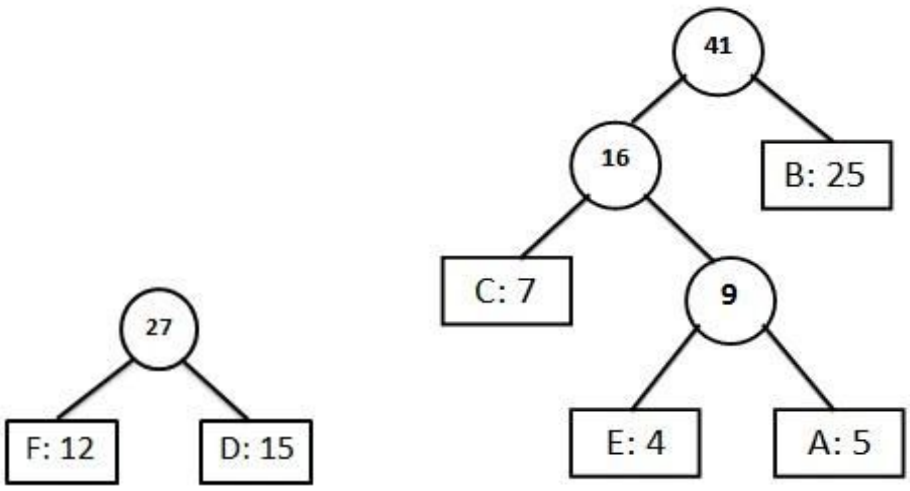
Value	F	D	CEA	B
Frequency	12	15	16	25

Step 4: Next elements are F and D so we construct another subtree for F and D.



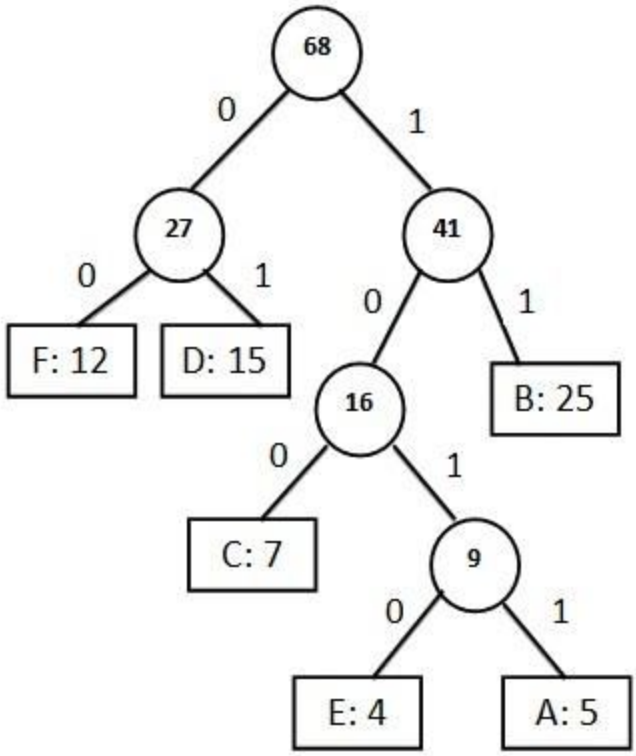
Value	CEA	B	FD
Frequency	16	25	27

Step 5: Taking next value having smaller frequency then add it with CEA and insert it at correct place.



Value	FD	CEAB
Frequency	27	41

Step 6: We have only two values hence we can combined by adding them.



Huffman Tree

Value	FDCEAB
Frequency	68

Now the list contains only one element i.e. FDCEAB having frequency 68 and this element (value) becomes the root of the Huffman tree.

Steps to print codes from Huffman Tree:

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.