Few-Shot Interactive Segmentation

A Thesis
Submitted in partial fulfillment of
the requirements for the degree of
Bachelor's and Master's of Technology
by

Aniket Bhatia
(160020001)

Supervisor:
Prof. Amit Sethi

Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)

July 2021

Abstract

The advent of Deep Learning has revolutionized how we perceive image segmentation. However, deep learning models are data-hungry and without a considerable amount of data the performance starts to deteriorate. In real world it is not uncommon to have a serious lack of datasets, which could result from a lack of annotation masks or the lack of images themselves. We approach this problem on the task of interactive segmentation. The task of interactive segmentation has reached new heights of performance owing to deep learning techniques. However, we see that the existing interactive techniques perform very poorly in data-constrained situations. We apply few-shot learning to the task of interactive segmentation and propose a novel architecture for the same. We create modules to fuse the interaction clicks with the network and to create prior masks for a specific instance in the query image. We also use test-time optimization to boost our performance. Our model achieves state-of-the-art performance on interactive segmentation tasks in a low-shot regime. To the best of our knowledge, few-shot learning has not been applied to interactive segmentation before, and, therefore, apart from our novel architecture, our very pursuit of few-shot interactive segmentation is novel.

Contents

List of Figures

List of Tables

Chapter 1

Literature Review and Related Work

1.1 Few-Shot Segmentation

This is a relatively new field which has evolved from the few-shot learning techniques being applied to classification tasks. Few-shot segmentation has its root motivation dwelling in its importance- being able to adjust to a lack of data. This lack of data can arise due to many reasons. Some primary reasons being a lack of images itself, the other being a lack of annotations when images are available, or in extreme cases both.

Before the introduction of few-shot learning to segmentation, it was used mainly for the task of classification. One of the very first papers in the domain of few-shot segmentation was brought out by the authors of [17]. One simple approach to just do few-shot segmentation would be to do just fine-tune the architecture, which has already been trained on a large dataset, with the insufficient images of a new class and then test it on the test image of that very new class. However, this is prone to the complication that this can cause over-fitting, because a huge number of parameters are being fine-tuned. There is also the added difficulty that one needs to set the optimization parameters, like the learning rate, the number of epochs for the fine-tuning to be carried out.

The authors of [17] propose a novel few-shot segmentation architecture, the first of its kind, which takes inspiration from general few-shot learning architectures. First, I will describe the setting of the problem which the authors of [17] tackled. Let, $S = \left\{\left(I_s^i, Y_s^i(l)\right)\right\}_{i=1}^{k}$ be the support set, where in, $Y_s^i \in L_{\text{test}}^{H\times W}$ is the segmentation mask for the Image I_s^i and $Y_s^i(l)$ is the mask of the i^{th} image in the set for the class $l \in L_{test}$. The aim of the architecture is to be able to learn a model which takes input I_q, the 'query image' and the

support set S, i.e. $f(I_q, S)$, the model is able to predict a decent binary mask $\hat{M}_q$, for the class l. While training the model has access to a large set of images and masks in a pair, $D = \left\{\left(I^j, Y^j\right)\right\}_{j=1}^{N}$, where $Y^j \in L_{\text{train}}^{H \times W}$, is the segmentation mask (semantic) for the image I^j in the training set. Now during testing the query images are only tested on new classes, i.e., $L_{\text{train}} \cap L_{\text{test}} = \varnothing$. This is, obviously, done in the presence of a small support set S, which was described earlier. This is the main distinction between a normal segmentation and what we are dealing with here. In normal segmentation the training phase is composed of training on certain classes, and the testing phase involves testing on a test-set also containing the same classes as the training-set. Although one might expect the L_{test} images to show up in the training set, but the annotations of those particular classes will not be included in the training set, even when the image is. This could also be the case when the images contain objects from multiple classes, which indeed happens for quite a lot of natural images.

The authors of [17] propose a method, where in the first channel or branch, as they call it, takes input an image from the support set S. This image, coming from the support set, has an annotation mask available to the network for use, and this mask is also taken as an input in the first branch. The second branch takes input the query image I_q, which is the image for which the network is expected to predict a segmentation mask. Thus, the input to the first branch is $(I_s, Y_s(l))$, which would help produce the parameters, $w, b = g_\eta(S)$. The other branch is to extract deep features from I_q. Let $F_q = \phi_\zeta(I_q)$ be the features that have been derived from the query image I_q, also let F_q^{mn} be the features at the space-coordinates (m, n). The authors perform a logistic regression at the pixel level, which employ the parameters from the first layers to obtain the mask, $\hat{M}_q^{mn} = \sigma(w^T F^{mn} + b)$. In this case, $\sigma(\cdot)$ is the sigmoid function. Figure 1.1 shows the architecture proposed by [17].

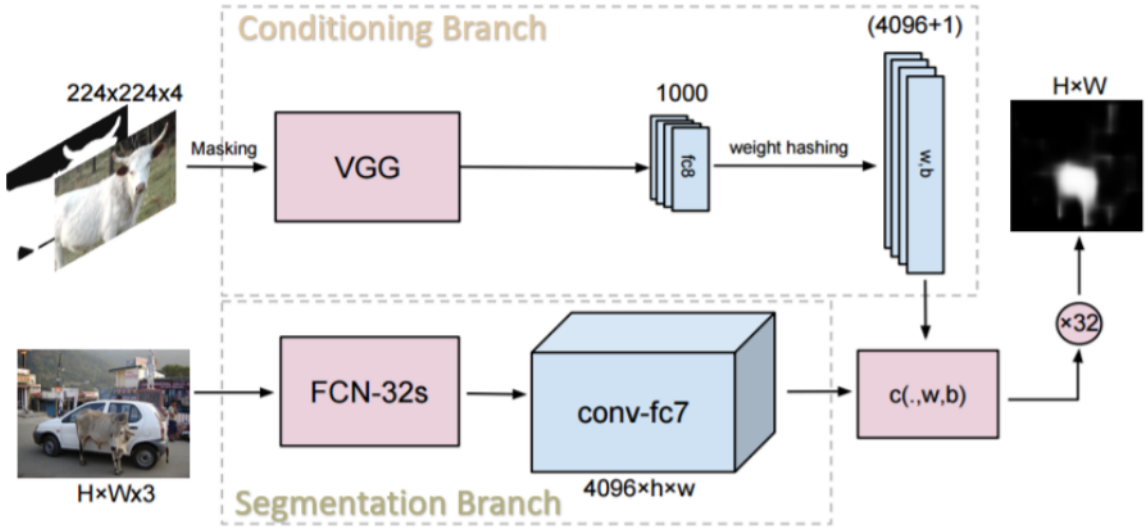

Figure 1.1: Architecture proposed in [17]

Specifically for 1-shot segmentation, they create a support set S, by sampling 5 classes from the 20 classes in the dataset. The dataset which they use is the combination of PASCAL VOC 2012 and SBD. There are obviously, many ways to sample those 5 classes from the total 20 classes in the dataset. The authors call the splitting of 15 classes for training during the training phase and 5 classes for the testing phase, as a 'split'. For split=0, classes 1 to 5 go in the testing phase while classes 6 to 20 go in the training phase. They create 4 such splits and for the results they state the mean IoU obtained for each split, and then average them out. After the support set has been made, a query image I_q is sampled along with its mask M_q (the ground truth mask). This is done in each iteration- Image label pair (I_q, M_q) being sampled uniformly from D_{train}, then a class is randomly sampled from the "approved" classes present in the image. By approved, I mean the classes that are supposed to be present according to the split. The support image is randomly picked from $D_{train} - (I_q, M_q)$ with the class, which as sampled earlier in the query image, present. The task for the model is to then predict the mask $\hat{M}_q$. After the predicted mask is produced by the network, the log-likelihood is maximized as:

$$\mathcal{L}(\eta, \zeta) = \mathop{\mathbb{E}}_{S, I_q, M_q \sim D_{\text{train}}} \left[\sum_{m,n} \log p_{\eta,\zeta} \left(M_q^{mn} \mid I_q, S \right) \right]$$

In the above equation, ζ and η are the parameters of the model, $p_{\eta,\zeta}$ is the probability of the mask given the model output, and the rest variables are as described before.

This now needs to be extended to k-shot. By k-shot I mean, k support images and masks

will be taken in to produce a predicted mask for a query image. The authors do this task by using the k images to produce k sets of model parameters, $w^i, b^{i^k}_{i=1}$. So basically, each one of them can be imagined as an independent classifier, and the authors just combine the predictions of the k masks which will be produced.

Next, the authors of [13] address a few limitations of the techniques being followed. They found out experimentally that CNNs have a proclivity to learn non-discriminative feature maps. So, the authors with their pipeline strive to increase activation inside the objects of the "approved" classes (as per the split) and decrease the activation inside the objects not in the approved classes, and elsewhere. The authors present that as an optimization problem and they come up with a close-form solution for it. The authors also address another problem- learning from very small number of training examples can cause overfitting and inadequate generalizations to the instances present in the query images. The authors target this particular issue by introducing a boosted inference, which was inspired from the traditional ensemble learning techniques which are known to be immune to over-fitting. The first proposal by the authors, namely using the technique to increase activation in the approved classes is used by them during both, training and testing, while the second proposal is applied only during testing, which is specifically targeted to boost performance. They were also the first to report results for few-shot segmentation on the dataset COCO-20^i, which was introduced in [11]. Their method is very naturally extended to the k-shot (i.e. having k training images for each of the class) setting.

In each training episode (for 1-shot segmentation) the authors of [13] choose a pair of support (x_s) and query (x_q) image randomly, along with their binary segmentation masks m_s and m_q respectively. These masks have either 1 or 0 as their elements. m_s is used inside the model to condition the class at hand inside the query image. For loss, the authors use the cross entropy loss between the ground-truth mask (m_q) and the mask which was predicted ($\hat{m_q}$). Now, for the purpose of the weighting the features to focus more on the class at hand, the authors include regularization because it supports higher activations in features on the foregorund and lower activations on the features on the background. Let $\phi_s \in \mathbb{R}^{d\times 1}$ be the feature differences vector which has been normalized over the in-object and out-of-object regions of the image. This is done for the segmentation map in the

support image and is given as,

$$\phi_s = \sum_{i=1}^{wh} F_{s,i}\left[\frac{\tilde{m}_{s,i}}{|\tilde{m}_s|} - \frac{1-\tilde{m}_{s,i}}{wh-|\tilde{m}_s|}\right]$$

. How relevant the features in ϕ_s are is decided by the parameter $r \in \mathbb{R}^{d\times 1}$, which is found out from the following objective function, $\underset{r}{\text{maximize}} \quad \phi_s^\top r, \quad \text{s.t.} \quad \|r\|_2 = 1$. This has a closed form solution given by $r^* = \frac{\phi_s}{\|\phi_s\|_2}$. The authors then use this $r*$ while calculating the similarity map between the query and the support images. This cosine similarity map is given by,

$$\sigma^*_{q,i} = \cos\left(f_s, F_{q,i}, r^*\right) = \frac{(f_s \odot r^*)^T\left(F_{q,i}\odot r^*\right)}{\|f_s \odot r^*\|_2 \cdot \left\|F_{q,i}\odot r^*\right\|_2}$$

Now, coming to the feature boosting segment of the work in [13], where in during testing they try to increase the model's generalizability to classes that were unseen during the training phase. Curiously, they predict both the support and the query mask. They compute the similarity maps as in the above equation, for predicting the support mask. Then the authors calculate the cross entropy loss and update the features in the following manner iteratively,

$$f_s^{n+1} = f_s^n - \nu\partial L\left(\hat{m}_s^n, m_s\right)/\partial f_s^n$$

where $L\left(\hat{m}_s^n, m_s\right)$ is the cross entropy loss and ν is the learning rate. Lastly, the authors do the following to obtain the final query prediction mask where ρ^n denotes the confidence in correct segmentation,

$$\hat{m}_q = \sum_{n=1}^{N} \hat{m}_q^n \rho^n$$

Moving ahead, the authors of [24] use masked-average pooling to create a representation which is object-related, which will in turn help including contextual information. They use cosine similarity to create guidance, which is pixel-wise helping in producing the prediction mask. They do all this alongside creating a novel architecture.

Let, a support image be $I \in \mathbb{R}^{3\times w\times h}$ and let its mask be Y, where w and h are the width and height of the image respectively. Let the output feature maps, as the authors call it, be $F' \in \mathbb{R}^{c\times w'\times h'}$ (here c is the number of channels present in F'). The features F' is firstly resized to the same width and height as that of the support image. The i^{th} element of the representative vector v, is given as,

$$v_i = \frac{\sum_{x=1,y=1}^{w,n} Y_{x,y} * F_{i,x,y}}{\sum_{x=1,y=1}^{w,h} Y_{x,y}}$$

By considering masked average pooling the authors would be able to extract the features selectively from the foreground and those of the background will be disregarded. Further, the authors of [20] propose a model which learns and aligns prototype representations for each of the class present. Having done that, the model performs the task of segmentation in the embedding space. In each of the episode, the model finds out the features from the support and the query image, then prototypes are calculated for using the masked average pooling, as described above. Then they use their prototype alignment regularization on the prototypes to learn the embedding prototypes.

The authors of [19] introduced a new concept of using a prior to add some extra information to aid the prediction of the segmentation mask for the query image. The prior mask would be created by using some features that would be extracted by a pre-trained network. In [19] the authors introduce a block called the prior generation block which does precisely this. This gives the model a first order probability map for the query image.

1.2 Interactive Segmentation

Image Segmentation, the task of partitioning the image into multiple segments, is one thing which finds application in so many different computer vision and image processing tasks. The segments isolated can be the final output or could be further used for processing. Now, Interactive Segmentation hopes to provide a personalized touch to the segmentation by introducing a human in the loop- the human would put a (series of) click(s) on the object which she/he would like to be segmented (and the personalized touch stems from the fact that the user can decide where to put the next 'click' looking at where the segmentation needs to be improved). This 'clicking' to produce a mask for that object is what the interaction is about.

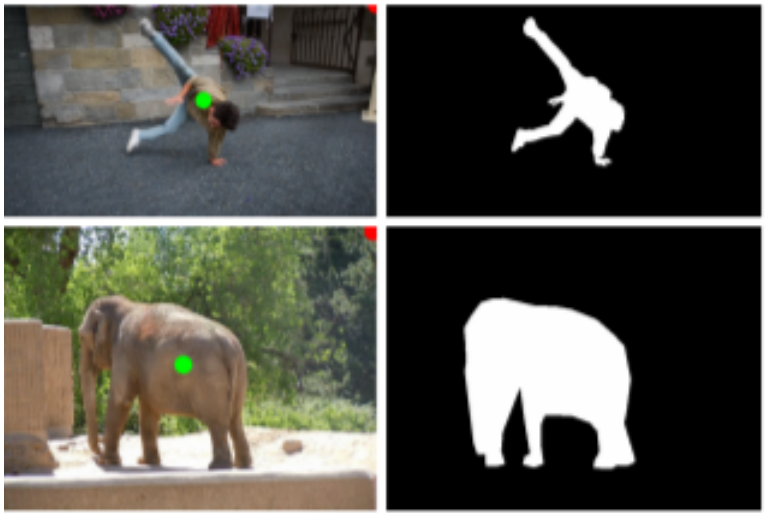

Figure 1.2: Interactive Segmentation [10]

The clicks by the user consists of foreground clicks on the object to be segmented, as shown by the green dot in figure 1.2 and background clicks outside the object of interest shown by the red dot in figure 1.2. So in essence each pixel in the object region is given the label of 'foreground' and the ones outside the object is given the label of 'background'. Effort has been made to get better masks using as less number of clicks possible. Advances in deep Learning and CNNs have led to a major improvement in performance. But this brings us to problems in training with this click-based approach. The problem arises due to the need of a clicking mechanism that would provide clicks on the object for the network to train.

There has been a lot of interesting research in this field. From the outset training posed a problem for interactive segmentation networks because of the way the problem is formulated- clicks need to be put on the objects to be segmented and only the ground truth mask corresponding to that image is to be fed in the system for that training sample.

[21] proposed an approach for click-based training. In the author's notation the user interactions have been denoted by $\mathcal{S}$, $\mathcal{S}^1$ representing all user provided positive (inside the object) clicks and $\mathcal{S}^0$ representing the negative (outside the object) clicks. Euclidean distance transforms were used to convert $\mathcal{S}^1$ and $\mathcal{S}^0$ to channels $\mathcal{U}^1$ and $\mathcal{U}^0$, and for that they introduced an operator f, defined as

$$f(p|A) = min(\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2})$$

$\forall\ q \in A$, where p and q are an element in the euclidean space and A its subset of it. What this basically does is output the minimum distance from a point to the points in a set. So to generate the channels $\mathcal{U}^t$ this transform is applied to all the points (pixels) in the image and A is taken as $\mathcal{S}^t$, $t \in 0, 1$, respectively. The channels or distance transforms then basically look like bumps (or inverted bumps) centered around the clicks. The transform value is truncated to 255.

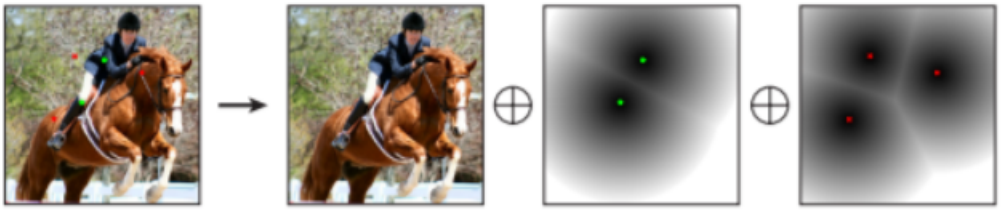

Figure 1.3: Transformation of clicks into separate channels [21]

As shown in figure 1.3 the green (positive) and red (negative) clicks are separately inculcated as two different channels. But the question arises as how to get these clicks in the first place! During the training process its impractical for a human to generate clicks for each image. The authors in [21] provide three strategies used to sample clicks and ultimately combine all the three to simulate user interaction. O has been called the set of pixels in the object (using the ground truth). They define

$$G = \{p_{ij} | (p_{ij} \in O) \cup (f(p_{ij}|O) \geq d)\}$$

which essentially means that the elements of G^c are in the background and up to a certain distance d away from the object.

The positive clicks are sampled randomly from O such that they are at least d_s distance away from each other and at least d_m distance away from the boundary. Negative clicks is where the variety kicks in. Strategy 1 is to sample points randomly from G^c and keep them distanced form each other and the boundary as described above. Strategy 2 is to sample randomly from the negative objects and Strategy 3 is to sample the first negative point randomly from G^c and then to obtain sequential points by,

$$p_{next} = \underset{p_{ij} \in G^c}{argmax} f(p_{ij}|S^0 \cup G)$$

where S^0 contains all the previously sampled negative clicks, which basically means to sample points that are greatest distance form the sampled points and those in G.

By doing this [21] establish quite profoundly on how the training and clicking can proceed seamlessly despite the apparent problems discussed at the start. Freed from the shackles of these problems, research moved further in developing how the interactive segmentation was done. [10] proposed a method titled 'Latent Diversity' which looks at interactive segmentation from a multimodal approach. The authors train two coupled convolutional networks in an end-to-end fashion as shown in figure 1.4. The first network produces a set of possible segmentations. The loss is so devised that it encourages diversity in the set. The second network is trained to select one out of these plausible segmentation masks.

The input to the network is the image X, the clicks S_p and S_n (p denoting positive clicks and n denoting negative ones), distance transforms as found from the clicks and the extracted VGG features. The distance transforms are calculated as,

$$T_p(x) = \min_{y \in S_p} \|x - y\|_2$$

$$T_n(x) = \min_{y \in S_n} \|x - y\|_2$$

T_p and T_n are single channeled intensity maps and are truncated at the value 255.

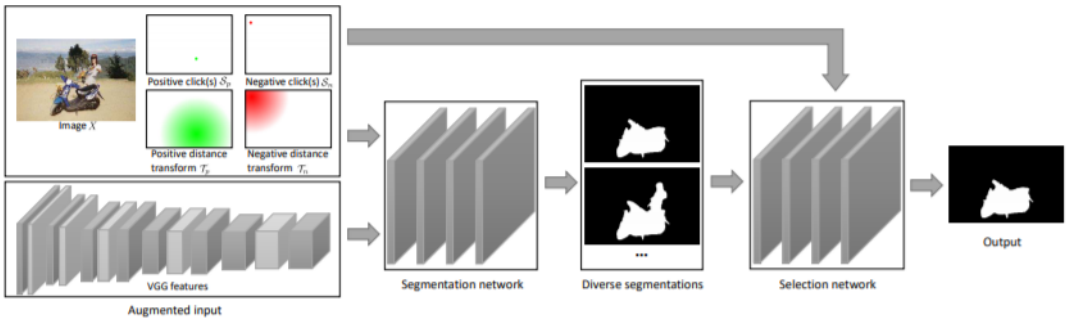

Figure 1.4: Latent Diversity [10]

Now, let X_i be the i^th sample and Y_i be its ground truth mask. The first part of the network generates M masks $f_1,\ f_2\ , ..f_M$. They calculate the hindsight loss $\sum_i \min_m \ell(Y_i, f_m)$, where $\ell(A, B)$ is a loss function defined as,

$$\ell(A, B) = 1 - \frac{\sum_{\mathbf{p}} \min(A(\mathbf{p}), B(\mathbf{p}))}{\sum_{\mathbf{p}} \max(A(\mathbf{p}), B(\mathbf{p}))}$$

where $A(p)$ and $B(p)$ are the value of the masks at pixel p. They introduce some soft constraints,

$$\ell_c\left(\mathcal{S}_p, \mathcal{S}_n, B\right) = \left\|\mathcal{S}_p \odot \left(\mathcal{S}_p - B\right)\right\|_1 + \|\mathcal{S}_n \odot (\mathcal{S}_n - (1 - B))\|_1$$

where $\odot$ is the Hadamard product. They define the loss as a combination of both,

$$\mathcal{L}_f\left(\boldsymbol{\theta}_f\right) = \sum_i \min_m \left\{\ell\left(Y_i, f_m\left(\mathbf{X}_i; \boldsymbol{\theta}_f\right)\right) + \ell_c\left(\mathcal{S}_p^i, \mathcal{S}_n^i, f_m\left(\mathbf{X}_i; \boldsymbol{\theta}_f\right)\right)\right\}$$

Now the next step is to select one of the M masks. They train their 'selection' network with the cross entropy loss,

$$\mathcal{L}_g\left(\boldsymbol{\phi}_g\right) = \sum_i \left(-g_{\diamond_i}\left(\mathbf{Z}_i\right) + \log \sum_{m=1}^{M} \exp\left(g_m\left(\mathbf{Z}_i\right)\right)\right)$$

where Z_i is the selection-network input.

[8] then proposed a scheme 'Back Propogating Refinement Scheme' which later became the base of the current state of the art approach. Their network has an encoder-decoder structure. They use the same strategy as in [21] to convert the user clicks to interaction maps (as they call it). They use cross entropy loss between the ground truth and the predicted mask, and they train the network via stochastic gradient decent.

Coming to the core of this scheme, the user-marked annotations provided as clicks (foreground or background) maybe incorrectly classified by the network. So this scheme strives to go back in the network, as shown in figure 1.5 and correct the prediction of the pixel-annotations provided by the user by formulating it as an optimization problem.

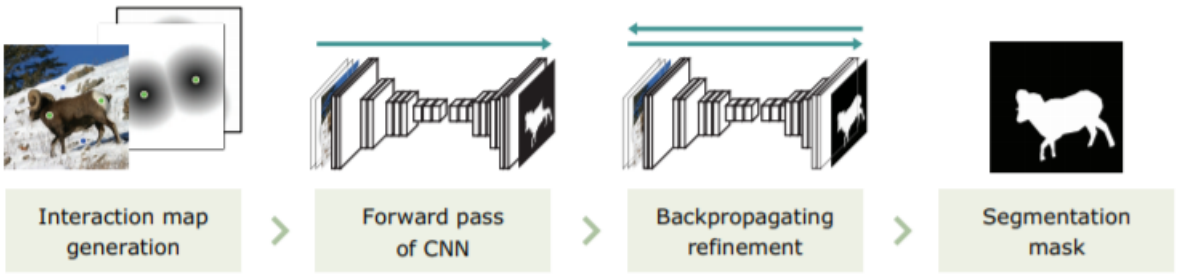

Figure 1.5: BRS flow [8]

The authors present two options, whether to fine-tune the network to yield the correct labels for the user-annotated pixels or to modify the interaction maps which may not have been a perfect representation of the user's wish leading to the final incorrect classification

of those very pixels. But fine-tuning the network may lead to the network losing its learnt knowledge and hence they chose to optimize the interaction maps.

Following the notation used by [8], let z^0 represent the interaction maps which need to be optimized. They define two energies E_C, corrective energy and E_I, inertial energy. The energy function for the interaction maps is then defined as

$$E(z^0) = E_C(z^0) + \lambda E_I(z^0)$$

where λ is to match the scale difference between the two energies. Then the optimal z^0 is found by,

$$\hat{z}^0 = \underset{z^0}{argmin}\, E(z^0)$$

The corrective energy is defined as,

$$E_C(z^0) = \sum_{u \in U} (l(u) - y(u))^2$$

where U is the set of user-annotated pixels, $l(u)$ is the value of that user-annotated pixel- 1 for foreground and 0 for background, and $y(u)$ is the predicted value of that pixel. Hence E_C is a measure of how many of the user-annotated pixels were incorrect, hence the name 'corrective' energy.

The Inertial energy is to prevent too much change in the maps and is defined as,

$$E_I(z^0) = \sum_{x \in N} (z^0(x) - z_i^0(x))^2$$

where N is set of all pixels in the interaction maps, and z_i^0 denotes the initial value of z^0 before BRS. The L-BGFS algorithm in [12] was used to find the optimal z^0. The BRS is done after each forward pass for each new click.

The dazzling thing about this approach is that it can convert any non-interactive method to an interactive one- all that one has to do is to perform BRS after the forward pass!

[18] followed this approach with the feature-BRS (f-BRS) in which the suthors sought to address the high computational cost incurred in BRS stemming from multiple passes through the network. The authors of [18] also introduce an additional zoom-in feature which will be discussed shortly.

The authors reparameterize the problem by realizing that the passes for the optimization needn't be done for the whole network and it suffices to do it for some part of

it. For the same they optimize some intermediate features instead of the interaction maps. In the author's terms let l_i denote label of the i^{th} user provided click and (u, v) its location. Consider a function $f(x)$ which is reparameterized as $f(x, a)$ with a as the introduced variable such that $f(x, p) = f(x)$, i.e. for $a = p$ $f(x, a)$ achieves its original value. The optimization task now is it minimize the objective function,

$$\lambda \|\Delta p\|_2 + \sum_{i=1}^{n} (f(x, p + \Delta p)_{(u_i, v_i)} - l_i)$$

Here Δp would mean the perturbation over the original value and so the first time would correspond to the inertial term and the second on to the corrective term. In terms of the network the reparameterization was done by adding a scale and a bias to a feature in the intermediate stage. Let $F(x)$ be the output of a certain layer for the input x then the reparameterization could be done as $s \cdot F(x) + b$ where s is a vector of scaling coefficients and b a vector of biases (channel-wise application), s and b would now be optimized.

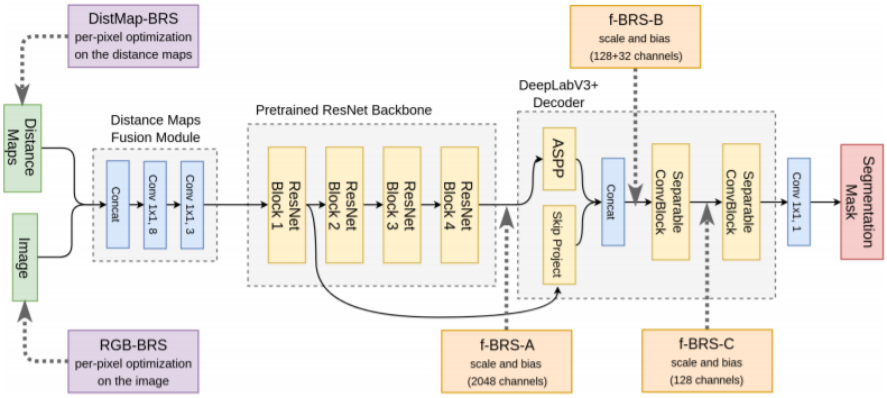

Figure 1.6: f-BRS shown at multiple places [18]

They experimented with various positions of including the s and b but they found f-BRS-B to work the best. Their architecture is shown in figure 1.6.

Zoom-In is a technique which they introduced for capturing some fine details. They realized that a significant accuracy level is achieved after a few clicks. Then with the current prediction a bounding box is created around the predicted area and the image is cropped by the bounding box, which is then upsampled and predicted on again, producing better results without any additional clicks. This is shown in figure 1.7.

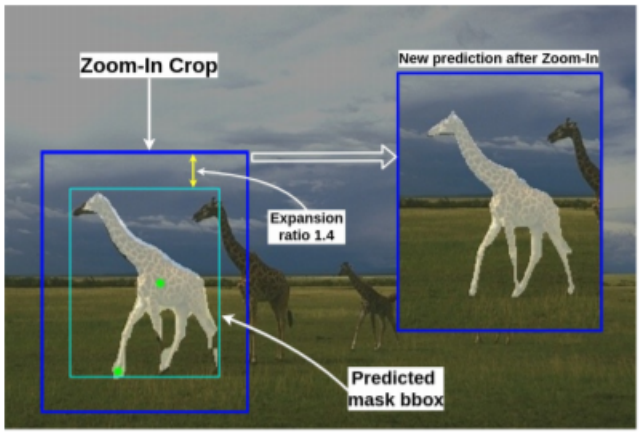

Figure 1.7: Zoom-In, taken from [18]

1.3 Automatic Segmentation

Here I discuss one of the most consequential papers for automatic nuclei segmentation-HoVerNet. This scheme was proposed in [3]. The authors here propose a pipeline to segment nuclei in mutli-tissue histopathology images and simultaneously classify the nuclei in its cancer type. The problems that exist are that there is a huge intra-class variability in nuclei and more often than not nuclei are packed quite close;y resulting in them being overlapped with each other. This network leverages the instance level data for each nuclei and the information embedded in the horizontal and vertical distances of nuclear pixels from the centre of mass of the nuclei. These distances are then used to segregate overlapping and clustered nuclei (separate in the sense that the network is capable of identifying the overlapping nuclei as different nucleus). After segmenting the nuclei, the class of the nuclei is predicted using an upsampling branch.

Firstly, to extract features the authors of [3] use a deep neural network which was inspired by the pre-activated residual network with 50 layers [6]. The authors reduce the downsampling by a factor of 4, from 32 to 8, by making the stride as 1 in the first convolution and removing the max-pooling operations which were performed later. This was done to prevent any loss of information, which would be required to provide a good segmentation result The authors call a chain of successive residual units as a residual block.

After passing the image throguh their modified version of Preact-ResNet50, the authors perform a nearest neighbour upsampling. This is done simultaneously via three dif-

ferent branches, so that classification can ensue along with segmentation. This is shown in figure 1.8.

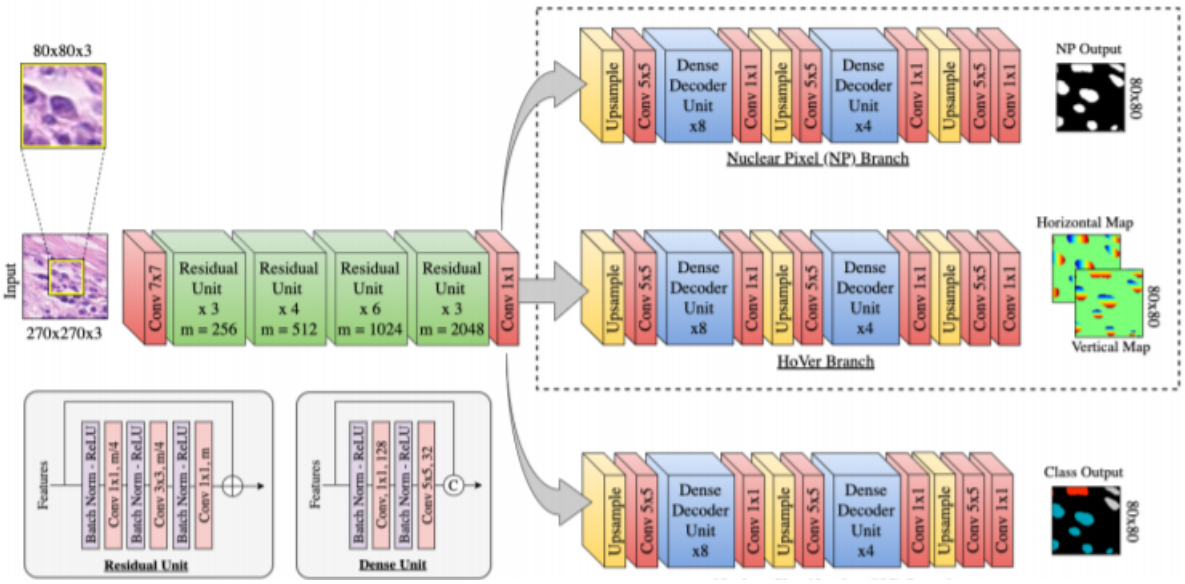

Figure 1.8: Flow of HoVerNet [3]

The nuclear pixel (NP) branch does a pixel classification between foreground and background, i.e. if the pixel belongs to a nucleus or not. The HoVer branch outputs the predicted vertical and horizontal distances of each pixel from its nuclear centre of mass. The nuclear classification (NC) branch is used for classification of a nucleus into its type. For instance segmentation the NP and the HoVer branch work together and for classification NC comes into the picture.

The authors use four sets of weights, in author's notation, w_0, w_1, w_2 and w_3 which would respectively correspond to the weights of Preact-ResNet50 (the part of the network that is common to all the three sub-branches), the NP branch, the HoVer branch and the NC branch. The loss is given as:

$$\mathcal{L} = \lambda_a \mathcal{L}_a + \lambda_b \mathcal{L}_b + \lambda_c \mathcal{L}_c + \lambda_d \mathcal{L}_d + \lambda_c \mathcal{L}_c + \lambda_f \mathcal{L}_f$$

$\mathcal{L}_a$ and $\mathcal{L}_b$ represent the regression loss in relation to the HoVer branch. $\mathcal{L}_c$ and $\mathcal{L}_d$ represent the loss corresponding to the NP branch while $\mathcal{L}_e$ and $\mathcal{L}_f$ are losses in reltion to the NC branch. $\lambda_a, .., \lambda_f$ are scalar weights given to the losses. Consider an input I at the pixel i, in the notation in accordance with the authors, $p_i(I, w_0, w_1)$ is defined as the regression output of the HoVer branch while $q_i(I, w_0, w_1)$ and $r_i(I, w_0, w_1)$ woul represent

the pixel wise softmax outputs from the NP and NC branches respectively. The authors then define $\Gamma_i(I), \Psi_i(I)$ and $\Phi_i(I)$ as the respective ground-truths. $\Psi_i(I)$ is the ground truth of the nuclear binary map. In that map the background pixels by convention are denoted as 0 and the nuclei (foreground) pixels are denoted as 1. $\Phi_i(I)$ is the gorund turth for the type of the nucleus. Lastly, $\Gamma_i(I)$ denotes the ground truth of the horizontal and vertical distances of the pixels in the nuclei with respect to their nuclear centres of mass. A regression loss is calcualted at the output of ther HoVer branch, as mentioned before. $\mathcal{L}_a$ is the mean squared error between the predicted and horizontal and vertical distances and their corresponding ground truths. $\mathcal{L}_b$ calculates the MSE between the x and y gradients of the horizontal and vertical maps and the gradients of their respective ground truths. They are formulated by the authors as follows:

$$\mathcal{L}_a = \frac{1}{n}\sum_{i=1}^{n}\left(p_i\left(I;\boldsymbol{w}_0,\boldsymbol{w}_1\right)-\Gamma_i(I)\right)^2$$

$$\mathcal{L}_b = \frac{1}{m}\sum_{i\in\mathbf{M}}\left(\nabla_x\left(p_{i,x}\left(I;\boldsymbol{w}_0,\boldsymbol{w}_1\right)\right)-\nabla_x\left(\Gamma_{i,x}(I)\right)\right)^2+\frac{1}{m}\sum_{i\in\mathbb{M}}\left(\nabla_y\left(p_{i,y}\left(I;\boldsymbol{w}_0,\boldsymbol{w}_1\right)\right)-\nabla_y\left(\Gamma_{i,y}(I)\right)\right)^2$$

$\mathcal{L}_c$ and $\mathcal{L}_e$ are the cross entropy loss for the NP and NC branches respectively while $\mathcal{L}_d$ and $\mathcal{L}_f$ are the DICE losses for the respective two branches. The two losses are mathematically represented as:

$$\text{CE} = -\frac{1}{n}\sum_{i=1}^{N}\sum_{k=1}^{K}X_{i,k}(I)\log Y_{i,k}(I)$$

$$\text{Dice} = 1-\frac{2\times\sum_{i=1}^{N}\left(Y_i(I)\times X_i(I)\right)+\epsilon}{\sum_{i=1}^{N}Y_i(I)+\sum_{i=1}^{N}X_i(I)+\epsilon}$$

In the above equations X represents the ground truth, Y represents the output, K is the number of classes and ϵ is a smoothness constant. K for $\mathcal{L}_c$ would be 2 because the NP branch entails a binary classification, while for $\mathcal{L}_e$ K would depend on how many nuclei classes does the dataset have. When classification labels for nuclei are not present in the dataset, only the NP branch and the HoVer branch function to thus produce the instance level segmentation.

Chapter 2

Few-Shot Interactive Segmentation

2.1 Introduction

Segmentation has become one of the most important tasks for the functioning modern day computer vision systems. It has many applications like in self-driving vehicles, medical imaging systems, robotic systems, etc. Techniques have been evolving over time for this task ranging from simple thresholding techniques [4], clustering methods [16], histogram-based methods [14], to deep learning techniques. Deep Learning has revolutionized this field and has pushed forth the limits of how much we can aim to achieve. Segmentation has many sub-types and the one that we are going to focus on is Interactive Segmentation. Interactive segmentation strives to incorporate interaction with the user in its model and by doing so it puts human in the loop. This sub-part of image segmentation can be used to segment out some particular instance in the image as opposed to segmenting out all the instances. Being able to choose a particular instance for the predicted mask makes this task even harder than the automatic segmentation task. Interactive segmentation in itself has many different applications, including its use in image editing soft-wares, image-annotating tools, etc. Deep learning has also taken the field of interactive segmentation to new heights. However, deep learning techniques are data-hungry and require a lot of data for good performance. The issue is we may not always have enough training data for all the classes. For example, in classifying or segmenting out a rare species of bird, we may only have a handful (maybe even one) of images for training. This will cause the existing architectures to perform very poorly. This is where the field of few-shot learning came into existence. Few-shot learning models are able to generalize on new classes with only

"few-shots" of data.

Few-shot learning has been extensively applied to classification tasks [9], [22], [7], [1] and automatic semantic segmentation tasks [19], [24], [20]. In our work we propose Few-Shot Interactive Segmentation. To the best of our knowledge, few-shot learning has not been applied to the task of interactive segmentation before. This is an important problem to address because, through our experiments, we find that the current interactive segmentation approaches only work well when there is abundant data, and, as remarked earlier, this very well may not be the case in real-world scenarios. Our proposed interactive segmentation model draws inspiration from the well researched techniques in few-sot segmentation [19], [17] like extracting useful features from very less data and the use of a first order prior mask for the query image. We also employ a test-time optimization technique called feature back-propagating refinement scheme, introduced in [18], to boost our performance. We create baselines from the state-of-the-art methods in interactive segmentation by modifying them for low-shot scenarios. We beat the baseline by a significant margin and set up a new benchmark for the challenging of task of interactive segmentation in low-shot scenarios.

2.2 Problem Setup

Let the support set be $S = \{(I_s^i, M_s^i)\}_{i=1}^k$. We follow the same notation that has been used in as [17]. Support set is a set of image and segmentation mask (binary) pair, where I_s is the image and M_s is the corresponding segmentation mask. The image might contain multiple instances belonging to multiple classes. The support set is therefore formed after it has been decided which classes go into the training phase and which ones go into the testing phase. Let the classes for the training phase belong in the set L_{train}, and the classes for the testing phase belong in the set L_{test}. These sets are created such that $L_{\text{train}} \cap L_{\text{test}} = \varnothing$, which is the key difference between other interactive segmentation tasks and ours. In other interactive segmentation tasks the training and the testing set always contain the same classes (might have a different relative representation from the class), however, in our case the classes in the testing and the training phase are completely disjoint, which makes our task much harder than what has been done before.

The pair as described above refers to the case when we are in the 1-shot paradigm. When

we have more than one image available for reference for a query image we call it k-shot, which would imply we have k images available for one query image for the class at hand. In this case we will have a tuple instead of a pair. We want to ensure that $L_{\text{train}} \cap L_{\text{test}} = \varnothing$, however, the images in the training set might contain instances belonging to classes in L_{test}. This issue is solved by appropriately modifying the segmentation mask associated with that image, in which the pixels corresponding to those instances belonging in the test class will be turned to 0, i.e. background. This is explained in depth in the upcoming section.

2.3 Dataset

The datasets which we have used is one of the most commonly used datasets to evaluate segmentation performance, specifically in the few-shot paradigm. The dataset is PASCAL-5^i [17]. This dataset is created using the dataset PASCAL VOC 2012 [2], and the extended annotations from SDS [5]. For setting up the training dataset, we obviously only include the images that do not overalap with the validation set of the PASCAL VOC 2012 dataset. PASCAL VOC 2012 contains 20 semantic segmentation classes, we choose five, and divide them like $4k + 1$, $4k + 2$, $4k + 3$, $4k + 4$, $4k + 5$, where 'k' represents the fold index, or the fold number. These 5 classes are put into the test set. The rest fifteen classes go into the training set. For our case, as done in [17], we select the fold $k = 0$ to contain the classes of aeroplane, bicycle, bird, boat and bottle in the testing set; the fold $k = 1$ contains the classes of bus, car, cat, chair, cow in the testing set; the fold $k = 2$ contains the classes of dining-table, dog, horse, motorbike, person in the testing set; the fold $k = 3$ contains the classes of potted-plant, sheep, sofa, train, television in the testing set.

The training set is created by including all images in the training set of PASCAL VOC 2012 and SDS which have at least one instance belonging to a class in the training class section (i.e. the remaining 15 classes, after 5 classes, according to the split number, have been removed). The binary mask used is then also modified to contain the foreground only in those portions of the image where those instances are present which belong to the training classes. A very similar process is done to create the testing set, with the differ-

ence that now the images are taken from the validation set in PASCAL VOC 2012 and SDS.

2.4 Proposed Method

Our proposed architecture is shown in figure 2.1. This architecture is built to predict a binary segmentation mask for the query image (I_q), given support image(s) (I_q), its/their binary segmentation mask(s) (M_q) and the clicks (positive (p_q) and negative (n_q)) for the query image. The positive and negative clicks are converted to distance maps before they are sent into the network.

Our architecture has three channels. The bottom-most channel seeks to extract features from the query image subject to the clicks provided, the middle channel is to extract features from the support image, which would enable the model to learn the characteristics of the class at hand, and the top-most channel seeks to create a prior mask for the query image, focusing on the instance of interest in the query image. The top-most channel for its purpose takes input the support and query images, the support masks and the clicks. The clicks before they can be sent into the network need to be converted into distance maps, this will be explained in depth in the upcoming sub-sections.

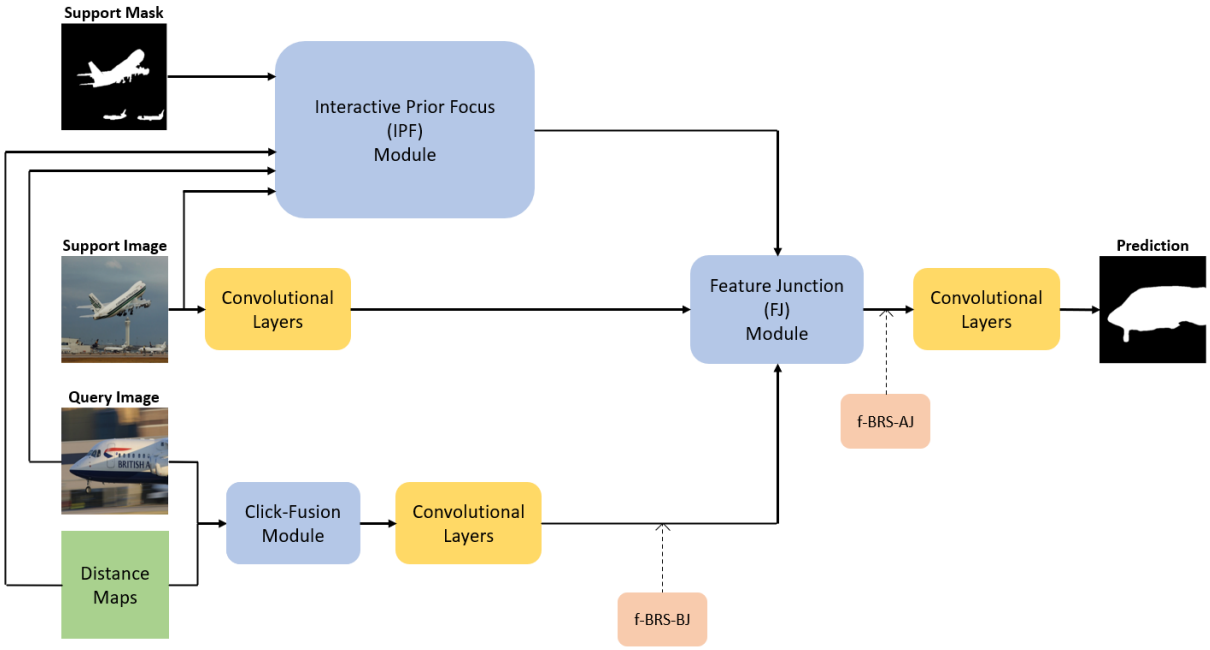

Figure 2.1: Proposed Architecture

In the bottom-most channel, the query image and the distance maps are first entered into the click-fusion module and the resultant is passed through convolutional layers, and the output after these convolutional layers are then given as input to the Feature Junction (FJ) Module. The middle channel is straightforward which involves feature extraction from the support image and the output from that going as input to the FJ Module. The output of the third channel (i.e. the Interactive Prior Focus (IPF) Module), which is the prior mask, also goes as input into the FJ Module. The output from the FJ Module then passes through some final convolutinoal layers before it provides the predicted mask. All the three modules will be explained in the upcoming sub-sections.

2.4.1 Distance Map Creation

We follow a similar technique to incorporate clicks into the network as [18]. Clicks cannot be just passed on in the network as some list or array. As it has been found in previous work [21] it is better to create some form of map and some how fuse the maps with the image and then pass it on to the network. We use the "distance" metric to calculate the map. The value of each pixel in the said "distance map" would be the distance of that pixel from the nearest click. In other words, for each pixel the distance is calculated from each click and the minimum of the distances is assigned to them.

2.4.2 Click-Fusion Module

Our model uses this module, which is tasked with appropriately fusing the distance maps and the images. The internals of this module is shown in the figure 2.2. Firstly, the distance maps are concatenated with the image resulting in a 5 channel output (3 channels from the RGB image and 2 channels from the distance map- one for positive clicks and the other for negative clicks). This 5 channel output is then passed into convolutional layers before it exits as a 3 channel feature. This click-fusion module is fully trainable and is trained during the training phase.

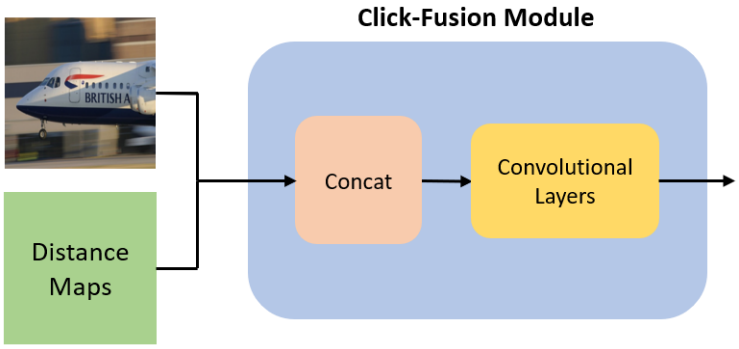

Figure 2.2: Internals of Click-Fusion Module

2.4.3 Interactive Prior Focus (IPF) Module

The aim of this module is to create a prior mask for the query image, which would focus on the instance of interest in the query image. The mask here would contain as its values the probabilities of the pixels belonging to the foreground (the instance of interest). First the high level features are extracted from both the query and the support image a VGG16 network [23] pre-trained on ImageNet [15]. Let the features from the query image be F_Q and that from the support image be F_S. Then the support mask is superimposed on the image (formally, a Hadamard product or a point-wise product is taken). Let, $X_Q = F_Q$ and $X_S = F_S \cdot M_S$, where M_S is the support mask and $\cdot$ is the Hadamard product. Now the job is to be able to have a point-wise correlation between X_Q and X_S, the prior mask should have high values in those regions of the image for which regions of X_S have similarity in X_Q. For calculating the prior mask Y_Q, firstly a cosine similarity is to be found between pixels of X_Q and X_S. Let, $x_q \in X_Q$ and $x_s \in X_S$ then the cosine similarity between the the vectors x_q and x_s is defined as,

$$cos(x_q, x_s) = \frac{x_q^T x_s}{||x_q||\,||x_s||}$$

Then maximum similarity is taken among the the support pixels. Let us call it c_q, and then let $C_Q = [c_1, c_2, c_3, ...c_{hw}]$ where the size of the features was $h \times w \times channels$. Then the query mask, Y_Q, is calculated as,

$$Y_Q = \frac{Y_Q - min(Y_Q)}{max(Y_Q) - min(Y_Q) + \epsilon}$$

We have set the ϵ as 10^{-8}, and is to avoid any division by 0. This prior mask calculation has been adapted from [19]. However, the issue here is that it prioritizes each instance

of the class at hand equally. Our task is to prioritize the instance of interest. We do so by fusing the acquired prior mask in the previous step and pass that along with the distance maps into the click-fusion module. This click-fusion will direct the attention of the probabilities to the instance of region and now this focused mask is the final output of the IPF Module. The parameters of the IPF Module except for those in the Click-Fusion Module are non-trainable. This is done to prevent the parameters being biased towards training classes. The internals of the IPF Module are shown in figure 2.3

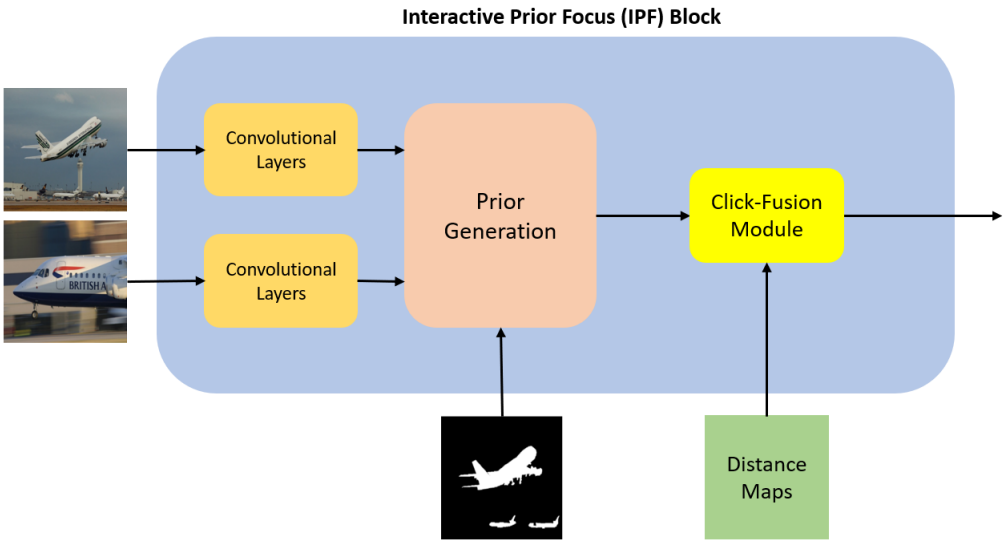

Figure 2.3: Internals of Interactive Prior Focus (IPF) Module

2.4.4 Feature Junction Module

As the name suggests this is the junction of the outputs of the three channels- the middle level features of VGG16 extracted by the convolutional layers of the support image & the click-fused query image and the instance-focused prior mask of the query image (generated by the IPF module). The features are converted to the same size (upsampling is done for this task) and then they are concatenated. Then they are passed through convolutional layers and the output is presented to a final segment of convolutional layers. This module is quite important, especially during test-time, owing to its location at the seat of combination of features coming from all the channels. We apply test-time optimization before and after this block and compare their results.

2.4.5 Training

We use the episodic paradigm for the few-shot aspects of our training. This means that we sample query and support images and create "episodes" in the same way that will happen during test-time. First we define our split, and as explained in the dataset section, split can have values from 0 to 3. This would set which 15 classes are to go in the training-set and which 5 classes are to be put in the testing-set. Then, in case of k-shot segmentation, after the query image has been sampled, k other images containing instance of the same class are chosen. We then choose one particular instance to be sampled from the query image on which we will place clicks.
We use SGD as the optimizer. The momentum is set to 0.9 and the weight decay is set to 0.0001. We train the model for 150 epochs.

2.4.6 Testing

The forming of episodes via sampling the query image and the support image(s) is similar to the training phase. We additionally use a test-time optimization called the feature back-propagating refinement scheme (fbrs) introduced in [18]. The intuition behind this test-time optimization is that at least the clicks that were given by the user as positive or negative should be correctly classified as foreground or background respectively. It is very reasonable to expect this because the user is giving the model the label of the pixel. However, it is quite often the case that even these pixels are wrongly classified. [18] introduced an optimization technique, fbrs, to tackle this. A feature would be chosen from some place in the architecture, let's call it F. Then the feature would be parameterized with a scale (s) and a bias (b), i.e. $F' = s \cdot F + b$. Now this scale and bias would be optimized using the L-BFGS, by forming an objective function depending on the incorrect labels assigned to the places where clicks were placed. Thereafter for the layers that come ahead, F' will be used instead of F.
We have experimented with two places to add the scale and bias. These two places are shown in figure 2.1- they are:

- f-BRS-AJ: Here AJ stands for "After Junction", which means the scale and bias area added to the feature coming out of the Feature Junction (FJ) module.

- f-BRS-BJ: Here BJ stands for "Before Junction", which means the scale and bias area added to the feature going into the FJ module.

f-BRS-BJ would be more computationally expensive because it is deeper into the network as seen from the end. But precisely because of it being deeper, its effect on the final predicted mask would increase, since it will be able to influence a greater number of features. f-BRS-AJ would be less computationally expensive than the former owing to its position near the end of the network, however it is expected that the effect it will have on the predicted mask will decrease, and therefore it is "expected" that it would perform worse than the former option. We experiment with both of them and the results are presented in the next section.

2.5 Results

We evaluate our novel architecture on the PASCAL-5^i. For now we choose split=0, which now sets which 15 classes are set to go in the training-set and which 5 classes go in the testing-set. During testing we present our results by using both f-BRS-BJ and f-BRS-AJ. The performance metric which we use is called Intersection over Union (IoU). This is defined as the ratio of the intersection area between the ground truth mask and predicted mask to the area of their union. In our case we can measure IoU two ways- class wise or an aggregate IoU. The aggregate IoU is called the FB-IoU or the Foreground-Background-IoU. The class wise IoU is called the mIoU which is calculated as the mean of FB-IoUs of different classes.

Since we are venturing into a novel area we need to be very careful in calculating the baselines. We first train and test the way the baseline was supposed to be done. However, it is not fair to do so because the train and test classes are different. Therefore we first train the baseline in the way it was supposed to be, then we fine-tune on the support set and then evaluate on the test set. Fine-tuning on the support set gives an opportunity for the other baseline model to get accustomed with the 5 classes in the testing set. The result of the fine-tuning is taken as the average of five random seed initializations. Our results are for one-click, split=0, and are present in table 2.1 and 2.2

Method	Class 1	Class 2	Class 3	Class 4	Class 5
f-BRS (not fine-tuned)	0.281	0.150	0.306	0.217	0.317
f-BRS (fine-tuned)	0.299	0.161	0.307	0.252	0.342
Ours (f-BRS-BJ)	0.386	0.212	0.392	0.325	0.351
Ours (f-BRS-AJ)	0.408	0.208	0.404	0.321	0.372

Table 2.1: Performance (IoU) on PASCAL-5^i

Method	mIoU	FB-IoU
f-BRS (not finetuned)	0.254	0.263
f-BRS (finetuned)	0.275	0.286
Ours (f-BRS-BJ)	0.332	0.343
Ours (f-BRS-AJ)	**0.343**	**0.357**

Table 2.2: Performance (mIoU and FB-IoU) on PASCAL-5^i

We clearly outperform the baseline by 7%. We also notice that, counter to our prior intuition, f-BRS-AJ performs better than f-BRS-BJ. This is a win-win situation because f-BRS-AJ is computationally less expensive and evidently preforms better than the other.

2.6 Conclusion

Interactive Segmentation has established its importance in today's computer vision systems. However, as we see the existing techniques require a lot of data and perform very poorly in low-shot scenarios. We introduce few-shot learning in interactive segmentation and propose a novel architecture. Our model uses click-fusion module to incorporate clicks into the network. We also introduce the Interactive Prior Focus (IPF) module which generates a prior mask for the query image selectively focusing on the instance of interest. Our architecture on the whole is able to learn useful features even in a data-constraint scenario. We, therefore, pioneer the new sub-field of Few-Shot Interactive Segmentation and set up a new benchmark by beating the baseline by a significant margin of 7%.

Some future work could include using our architecture in conjunction with a vanilla few-shot segmentation (automatic) architecture with our model being tasked with refining the

mask produced by the vanilla few-shot segmentation. This also opens up the possibility of continual learning being incorporated. In a continual learning set up the model would update its parameters each time a click is placed and the output evaluated. This process could conclude in a situation where in our model now no longer requires clicks since it would have had learnt enough from the plenty samples it received clicks on, overtime.

Bibliography

[1] Peyman Bateni et al. “Improving Few-Shot Visual Classification with Unlabelled Examples”. In: *CoRR* abs/2006.12245 (2020). arXiv: 2006.12245. URL: https://arxiv.org/abs/2006.12245.

[2] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. 2012.

[3] Simon Graham et al. “XY Network for Nuclear Segmentation in Multi-Tissue Histology Images”. In: *CoRR* abs/1812.06499 (2018). arXiv: 1812.06499. URL: http://arxiv.org/abs/1812.06499.

[4] Prathima Guruprasad. “OVERVIEW OF DIFFERENT THRESHOLDING METHODS IN IMAGE PROCESSING”. In: June 2020.

[5] Bharath Hariharan et al. “Simultaneous Detection and Segmentation”. In: *European Conference on Computer Vision (ECCV)*. 2014.

[6] Kaiming He et al. “Identity Mappings in Deep Residual Networks”. In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027.

[7] Yuqing Hu, Vincent Gripon, and Stéphane Pateux. “Leveraging the Feature Distribution in Transfer-based Few-Shot Learning”. In: *CoRR* abs/2006.03806 (2020). arXiv: 2006.03806. URL: https://arxiv.org/abs/2006.03806.

[8] Won-Dong Jang and Chang-Su Kim. “Interactive Image Segmentation via Backpropagating Refinement Scheme”. In: *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[9] Seong Min Kye et al. “Transductive Few-shot Learning with Meta-Learned Confidence”. In: *CoRR* abs/2002.12017 (2020). arXiv: 2002.12017. URL: https://arxiv.org/abs/2002.12017.

[10] Z. Li, Q. Chen, and V. Koltun. “Interactive Image Segmentation with Latent Diversity”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 577–585.

[11] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[12] Dong C. Liu and Jorge Nocedal. “On the Limited Memory BFGS Method for Large Scale Optimization”. In: *MATHEMATICAL PROGRAMMING* 45 (1989), pp. 503–528.

[13] Khoi Nguyen and Sinisa Todorovic. “Feature Weighting and Boosting for Few-Shot Segmentation”. In: *CoRR* abs/1909.13140 (2019). arXiv: 1909.13140. URL: http://arxiv.org/abs/1909.13140.

[14] Giuliana Ramella and Gabriella Sanniti di Baja. “Color Histogram-Based Image Segmentation”. In: Jan. 2011, pp. 76–83. ISBN: 978-3-642-23671-6. DOI: 10.1007/978-3-642-23672-3_10.

[15] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[16] B. Sathya and R. Manavalan. “Image Segmentation by Clustering Methods: Performance Analysis”. In: *International Journal of Computer Applications* 29 (Sept. 2011), pp. 27–32. DOI: 10.5120/3688-5127.

[17] Amirreza Shaban et al. “One-Shot Learning for Semantic Segmentation”. In: *CoRR* abs/1709.03410 (2017). arXiv: 1709.03410. URL: http://arxiv.org/abs/1709.03410.

[18] Konstantin Sofiiuk et al. “F-BRS: Rethinking Backpropagating Refinement for Interactive Segmentation”. In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[19] Zhuotao Tian et al. “Prior Guided Feature Enrichment Network for Few-Shot Segmentation”. In: *CoRR* abs/2008.01449 (2020). arXiv: 2008.01449. URL: https://arxiv.org/abs/2008.01449.

[20] Kaixin Wang et al. “PANet: Few-Shot Image Semantic Segmentation with Prototype Alignment”. In: *CoRR* abs/1908.06391 (2019). arXiv: 1908.06391. URL: http://arxiv.org/abs/1908.06391.

[21] Ning Xu et al. “Deep Interactive Object Selection”. In: *CoRR* abs/1603.04042 (2016). arXiv: 1603.04042. URL: http://arxiv.org/abs/1603.04042.

[22] Haipeng Zhang et al. “Sill-Net: Feature Augmentation with Separated Illumination Representation”. In: *CoRR* abs/2102.03539 (2021). arXiv: 2102.03539. URL: https://arxiv.org/abs/2102.03539.

[23] Xiangyu Zhang et al. “Accelerating Very Deep Convolutional Networks for Classification and Detection”. In: *CoRR* abs/1505.06798 (2015). arXiv: 1505.06798. URL: http://arxiv.org/abs/1505.06798.

[24] Xiaolin Zhang et al. “SG-One: Similarity Guidance Network for One-Shot Semantic Segmentation”. In: *CoRR* abs/1810.09091 (2018). arXiv: 1810.09091. URL: http://arxiv.org/abs/1810.09091.