DSA

Bibin Babu

September 11, 2024

Chapter 1

Experiments

Below figure shows the flow chart of whole time evolution of cancer cell.

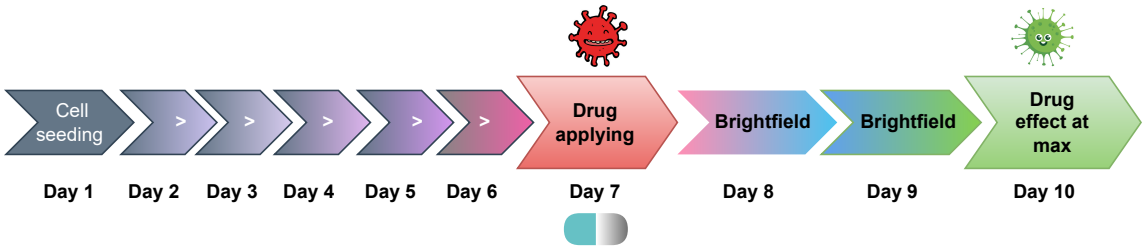

Figure 1.1: time progress

The current dataset we are working with only includes data from day 10 (ie after 3 days of drug application).

Dataset class overview

Dataset Type	Drug Screened	Single Dose	Untreated	Total
Combined Unlabeled	12 (3%)	204 (60%)	150 (37%)	366
Supervised	12 (16.66%)	30 (41.66%)	30 (41.66%)	72

Table 1.1: Dataset Overview

The original images 1.2 are approximately 2500×2500 pixels in size, in 16-bit grayscale, and consist of multiple channels. These channels come from taking images at different focal planes in brightfield microscopy. The number of channels can vary, as you can take images at any number of focal planes.

However, for time efficiency, the current data we have collected contains 3 channels per image.

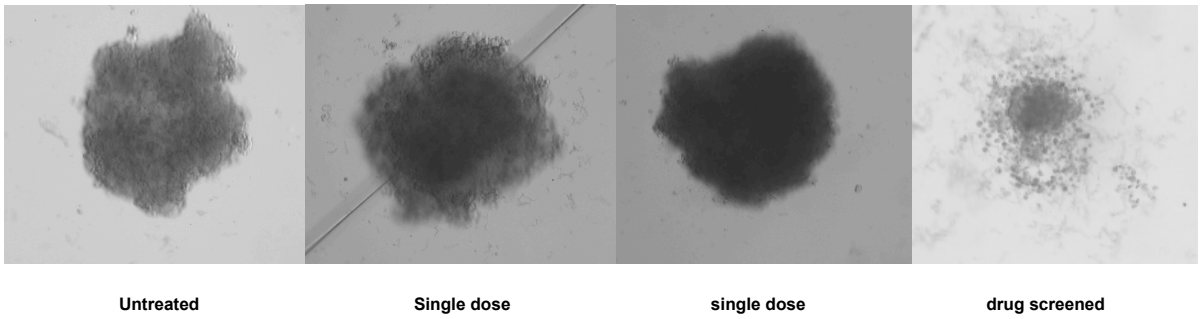

Figure 1.2: originals

It's important to keep the original color depth, aka bit depth. An 8-bit image contains 256 color tones (0-255) per channel, while a 16-bit image contains 65,536 color tones (0-65,535) per channel—in our case, 65,536 shades of gray.

When data augmentation involves significant changes in brightness, contrast, or color, if we have an 8-bit image and lose, let's say, 50 percentage of the tones, we're left with only 128 levels of color and tone. This loss of information will typically manifest as banding, especially in areas of smooth color transitions. This is where we see visible stripes of color with slightly jagged edges forming across the image.

In contrast, a 16-bit image contains 65,536 levels of color and tone. If we lose 50 percentage of those colors and tones, we'd still have over 32,000 levels remaining. This means we'd retain smoother color transitions, preserve edge details, and maintain color and hue accuracy much better. The dynamic range (the difference between the lightest and darkest parts of the image) would be preserved much longer compared to a typical 8-bit image.

Data preprocessing

Detailed study/research/experiments on data augmentation and image preprocessing techniques sepcifically for our 16 bit gray scale image are still need to be done. At the moment since we are trying to create the full pipe line first we used standard data augmentations from SimCLR paper.

1. Normalize the 16-bit image to [0, 1].

2. Do the following augmentations:

   (a) Crop the image randomly, and resize it to 96×96.

   (b) Randomly change the brightness, contrast, saturation, and hue of the cropped patch.

3. For each original image, perform step 2 twice to obtain two augmented images.

Model

The model takes one image and produce a latent representation of the input. The aim of the model is to cluster similar images together in latent space.

Training

The training process follows these steps:

1. We take a batch of images with batch size B.

2. Our dataset class returns two augmented versions for each original image in the batch, resulting in $2B$ images as input.

3. The model produces $2B$ latent representations, independently for each augmented image.

4. For each batch, the two augmentations of the same image are treated as positive pairs, while all others are considered negative samples.

5. We calculate the cosine similarities between the positive and negative pairs. Positive pairs consist of augmentations of the same image, while negative pairs are augmentations of different images. These cosine similarities are then used as input to the loss function described below equation 1.1

$$-\frac{\text{sim}\left(z_i, z_j\right)}{\tau} + \log\left[\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp\left(\text{sim}\left(z_i, z_k\right)/\tau\right)\right] \tag{1.1}$$

where z_i, z_j are positive pairs and z_i, z_k are negative pairs.

Visualisation of before and after preprocessing of image shown in figures 1.3 to 1.8

Variation ideas

1. Each image is considered as RGB since we have 3 channels, and we apply 2 standard augmentations.

2. One channel is considered as the anchor (the most sharpened layer), and the others are treated as the two augmentations.

3. Each image is considered as RGB since we have 3 channels, and we apply more than 2 standard augmentations (research for medical grayscale images).

4. Include the anchor as a positive sample, i.e., 3 augmentations in total (1 anchor as augmentation and the other 2 layers as augmentations).

5. Remove the positive sample j from the denominator of the loss function. Since j is the only image as a positive sample in the sum of the denominator softmax, its contribution will be less.

6. For supervised SimCLR: Ensure that no images from the same breed/class are included in the negative samples.

7. Try another loss function, such as Triplet loss, as shown in the original paper.

Variations implementations:

The two variations tried so far differ only in how they handle the image for data augmentation.

In the first variation, we take a 3-channel image and treat it like a standard RGB image, applying SimCLR-style augmentations to create two augmented versions.

In the second variation, we take a 3-channel image and compute the sharpness of each layer by calculating the magnitude of the gradient of pixel intensities in the x and y directions, which indicates edge strength and provides a measure of how sharp the transitions between pixel values are. The sharpest layer is used as the anchor, while the other two layers are treated as augmentations.

Variation 1:

Input to model (train loader dimension) :

- aug1: torch.Size([16, 3, 96, 96]) (batch size, no of channels, H, W)

- aug2: torch.Size([16, 3, 96, 96]) (batch size, no of channels, H, W)

Model output just after convolution layers: (before applying projetion head):

- torch.Size([16, 512]) (Batch size, standard resnet18 output dimension after avg pooling)
- This output feature will be used for further downstream task.

Model output after projection head:

- torch.Size([16, 20]) (Batch size, no of values in feature vector)
- 2*Batch size is due to the concatenation of aug1 images and aug2 images for the efficient/smart way to calculate the loss.
- No of values in feature vector is a variable which we can change and experiment which will give better accuracy. This output feed to loss function to train model to reduce the loss there by increase the cosine similarity between positive samples in the batch.

Below images after data augmentation and normalisation of 3 channel version:

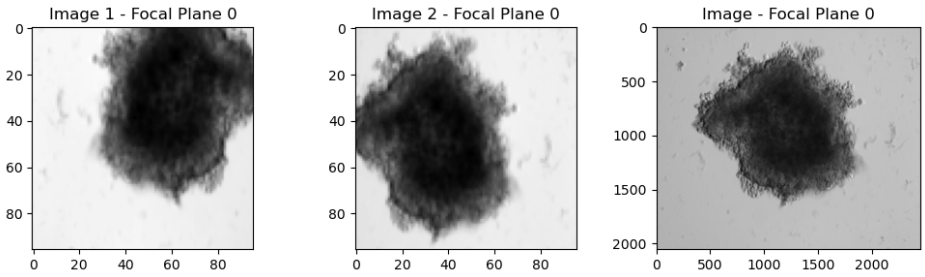

Figure 1.3: op1

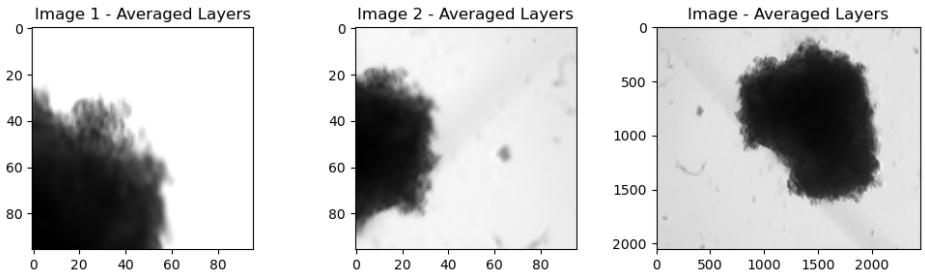

Figure 1.4: op3

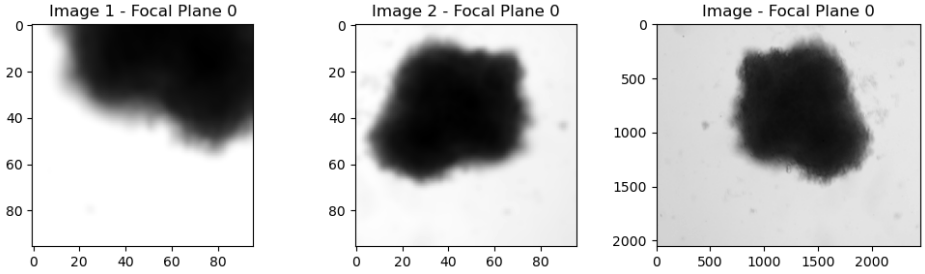

Figure 1.5: Blured augs

Variation 2:

Input to model (train loader dimensions) :

- aug1: torch.Size([16, 1, 96, 96]) (batch size, no of channels, H, W)
- aug2: torch.Size([16, 1, 96, 96]) (batch size, no of channels, H, W)

Model output just after convolution layers: (before applying projetion head)

- torch.Size([16, 512]) (Batch size, standard resnet18 output dimension after avg pooling)
- This output feature will be used for further downstream task.

Model output after projection head:

- torch.Size([16, 20]) (Batch size, no of values in feature vector)

- 2*Batch size is due to the concatenation of aug1 images and aug2 images for the efficient/smart way to calculate the loss.

Below images after data augmentation and normalisation of 1 channel version:

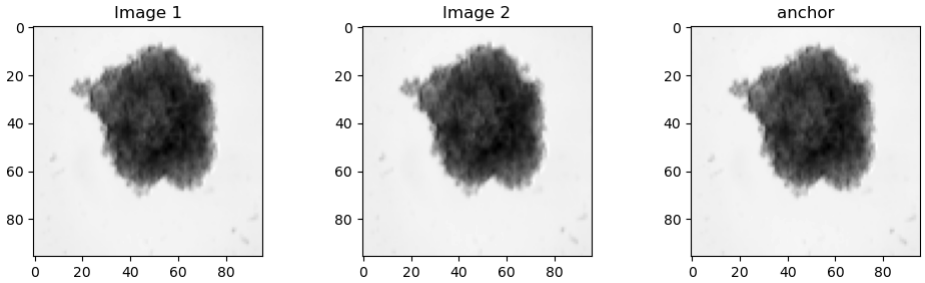

Figure 1.6: 1dop1

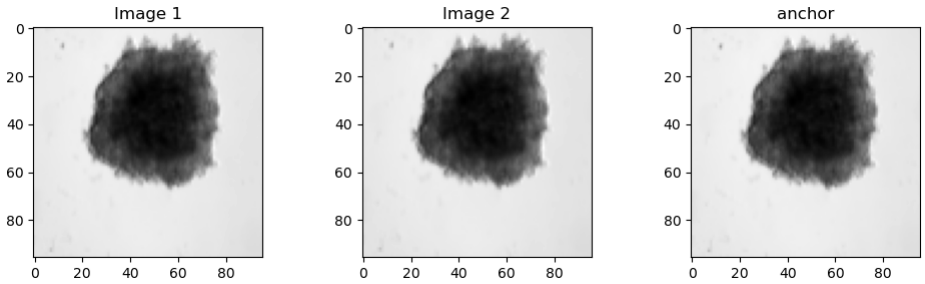

Figure 1.7: 1dop2

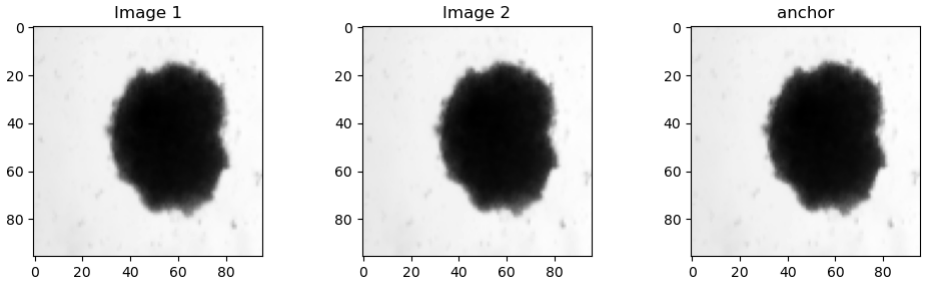

Figure 1.8: 1dop3