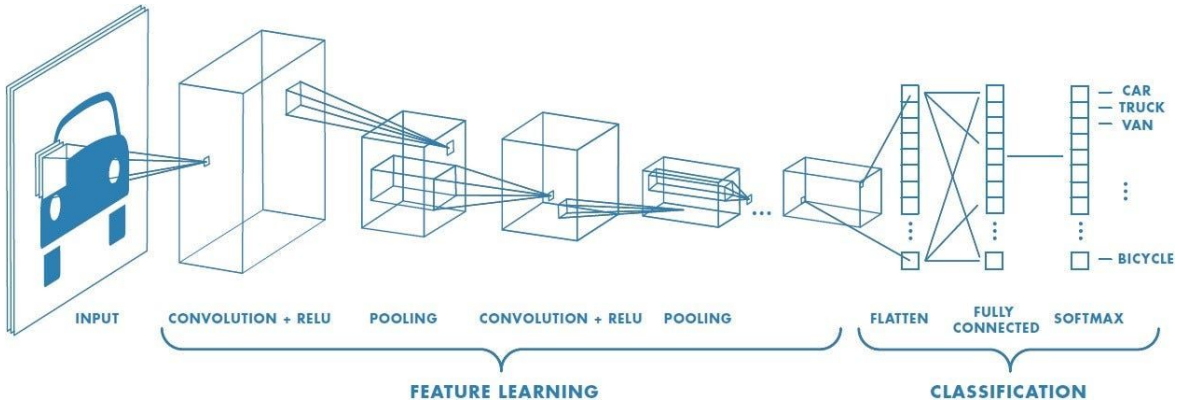
Convolutional Neural Networks and the U-Net Architecture

Presentation by
Maxwell Guevarra

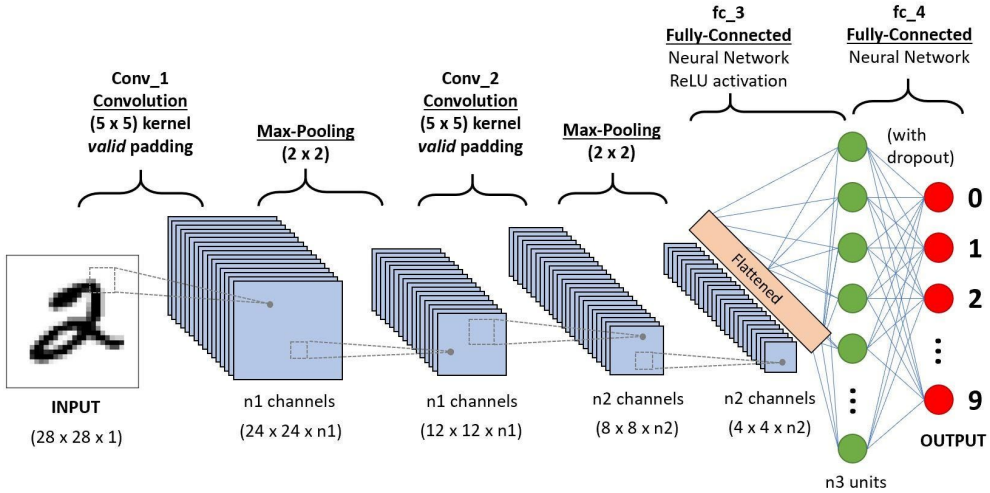
What is a Convolutional Neural Network?

- Deep learning classification algorithm
- Typically used for classifying images with a categorical label
- Images are taken as input, while the classifying label is the output



General Process

- The input images go through a series of filtered layers to determine classification label
  - *Convolution Operations*: takes the most import features of an image
  - *Max-Pooling Operations*: reduces the size of the image features for easier computation



Training a Convolutional NN Model: Basic Structure

General Loop:

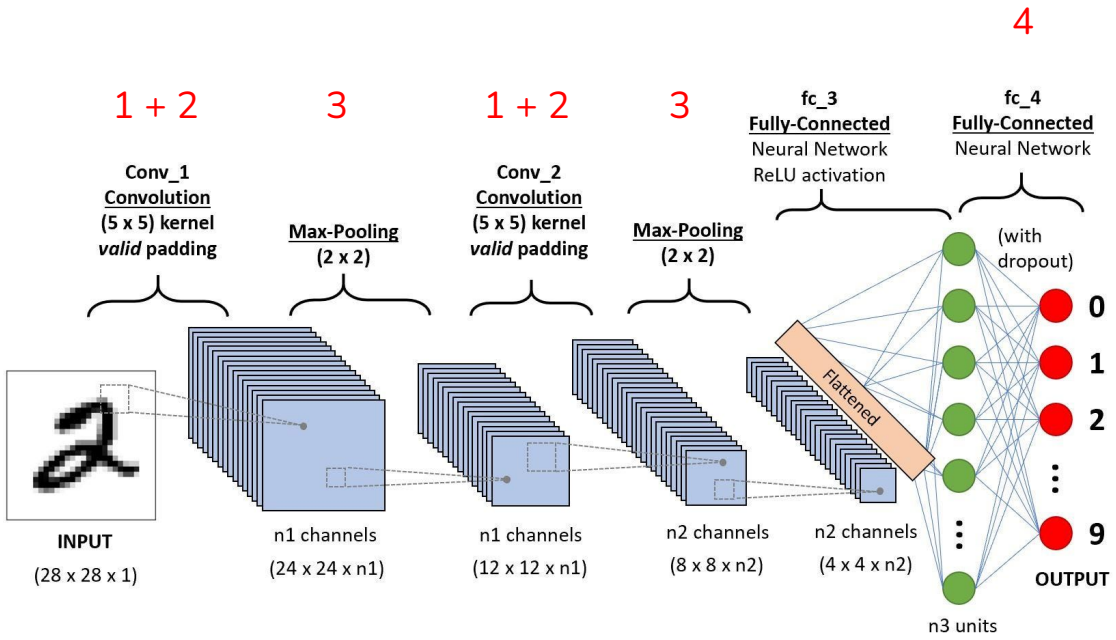
1. Convolutional Layer
2. ReLU Function

   (Can Repeat 1-2)

3. Max-Pooling Operation

   (Can Repeat 1-3)

4. Final Output Layer



Training a Convolutional NN Model: Using the Training Data

Note: Whenever a model needs to be trained, the training dataset is split into a training and validation set to directly compare model performance.

- In small batches, the training data will go through each set of all the convolutional layers, ReLU, and max-pooling operations
- Each categorical output label will be defined by the important features each layer collects as the input image data is fed through the model
- Each time the full training dataset undergoes the model is considered an epoch
- After n epochs, we use the validation set to make predictions on the newly trained model, and compare the output labels with the actual labels

Training a Convolutional NN Model: Loss Function

- Loss = how far off we are for a single prediction
  - We want this value to be small
- While training a model, this value decrease as the training data goes through the model for however many times is needed
- For classification, the method to calculate the loss value is either:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

For Binary Classification:

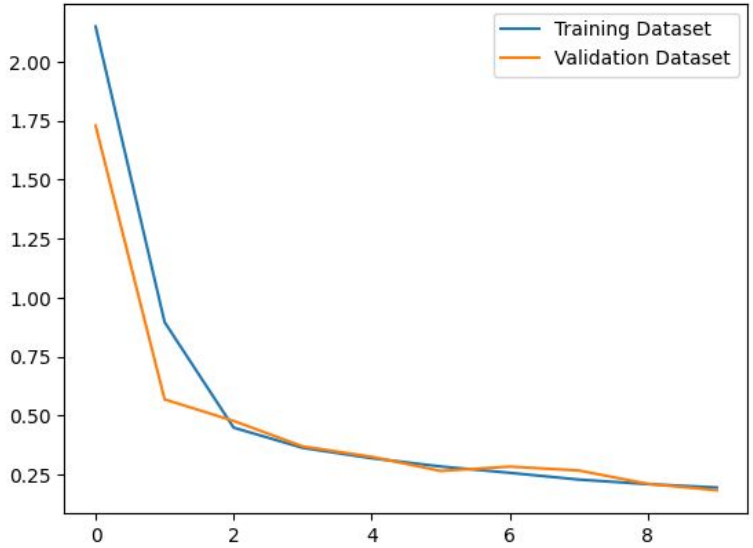
Binary Cross Entropy Loss Function

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i$$

For Multiclass Classification:

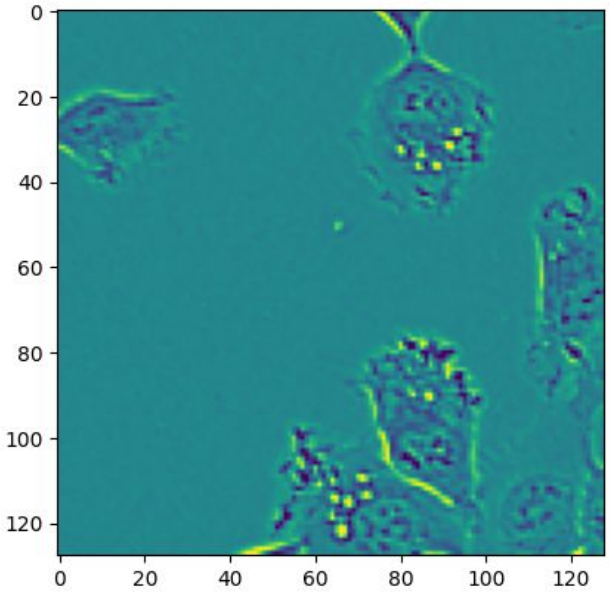
Cross Entropy Loss Function

Training a Convolutional NN Model: Cost Function



- The Cost Function is just the average of the value from the loss function
  - Like the Loss value, we want this value to be small
- Generally, for each iteration of the entire training model undergoing the model (epoch) we want to know what the cost function value is for both the training and validation sets

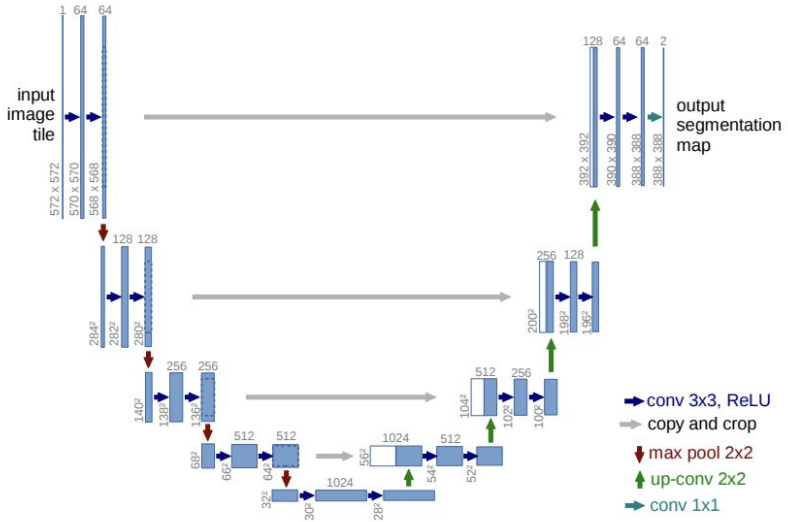
Cell Count Dataset



- The data itself consists of images of cells
- The goal with the model is to determine how many cells are in each image
  - X = image data
  - y = number of cells
- To fine tune the model parameters, the training data will be fitted on training and validation sets
- The final model will be trained on the full training data
- Model is evaluated on the mean squared error (MSE) for predictions on the test dataset

The U-Net Architecture: What is it?

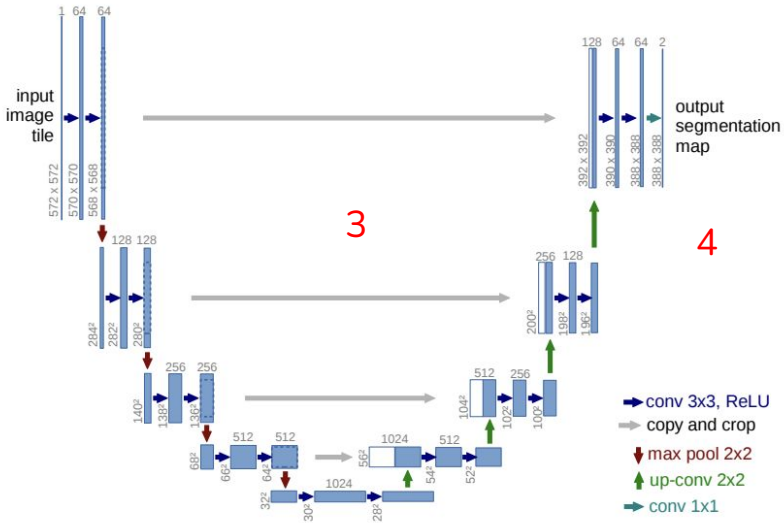
- Type of CNN model that utilizes a series of convolutional layers and max-pooling operations in addition to upsampling operations
- The goal of the upsampling operations are to increase the image resolutions to gather more precise features



The U-Net Architecture: Model Components

- Convolutional Layers:
  - Standard CNN convolutional layers that start at input 1, all the way to input 1024
  - Kernel size is 3 (3x3)
- Max-Pooling Operations
  - Standard CNN max-pooling operations that have a kernel size of 2 (2x2)
- Upsampling Operations
  - Scale factor = 2
  - Increases the resolution of the image by the scale factor

The U-Net Architecture: Model Structure



1

2

3

4

1. Convolutional layers, ReLU, and max-pooling operations are applied in the same manner as a basic CNN model
2. When the output reaches 1024, the upsampling occurs before the next set of layers and operations
3. Once upsampling occurs, the output value from a previous set of layers gets added onto the current layer
4. Now upsampling occurs after each set of layers until the final output layer is reached

Differences in the U-Net Model

- Main difference is the use of upsampling to increase the image resolution
  - In addition to this, whenever upsampling occurs, the next two convolutional layers will have be the exact same as a previous layer's output is being added to the first layer's input
- The inputs/outputs mirroring each other after the halfway point of the model gives it its name of the U-Net model since the sets of convolutional layers are happening twice

Loss and Cost Functions

- This model calls for the use of the binary cross entropy loss function
- Operation is used on outputs created by training sets

```
num_epochs = 10
L_history_train = []
L_history_val = []

for ep in range(num_epochs):

  model.train()
  L = 0

  for x_batch, y_batch in dataloader_train:

    x_batch = x_batch.to(device)
    y_batch = y_batch.to(device)

    outputs = model(x_batch)
    loss = torch.nn.functional.binary_cross_entropy_with_logits(outputs, y_batch)
    L += loss*len(x_batch)
    model.zero_grad() #Erase any previous gradient calculations
    loss.backward()
    optimizer.step()
```

Using a U-Net Model Implementation

```
def __init__(self):
  super().__init__()
  self.conv1 = torch.nn.Conv2d(1, 64, kernel_size=3, padding='same')
  self.conv2 = torch.nn.Conv2d(64, 64, kernel_size=3, padding='same')

  self.conv3 = torch.nn.Conv2d(64, 128, kernel_size=3, padding='same')
  self.conv4 = torch.nn.Conv2d(128, 128, kernel_size=3, padding='same')

  self.conv5 = torch.nn.Conv2d(128, 256, kernel_size=3, padding='same')
  self.conv6 = torch.nn.Conv2d(256, 256, kernel_size=3, padding='same')

  self.conv7 = torch.nn.Conv2d(256, 512, kernel_size=3, padding='same')
  self.conv8 = torch.nn.Conv2d(512, 512, kernel_size=3, padding='same')

  self.conv9 = torch.nn.Conv2d(512, 1024, kernel_size=3, padding='same')
  self.conv10 = torch.nn.Conv2d(1024, 1024, kernel_size=3, padding='same')

  self.conv11 = torch.nn.Conv2d(1024, 512, kernel_size=2, padding='same')
  self.conv12 = torch.nn.Conv2d(1024, 512, kernel_size=3, padding='same')
  self.conv13 = torch.nn.Conv2d(512, 512, kernel_size=3, padding='same')

  self.conv14 = torch.nn.Conv2d(512, 256, kernel_size=2, padding='same')
  self.conv15 = torch.nn.Conv2d(512, 256, kernel_size=3, padding='same')
  self.conv16 = torch.nn.Conv2d(256, 256, kernel_size=3, padding='same')

  self.conv17 = torch.nn.Conv2d(256, 128, kernel_size=2, padding='same')
  self.conv18 = torch.nn.Conv2d(256, 128, kernel_size=3, padding='same')
  self.conv19 = torch.nn.Conv2d(128, 128, kernel_size=3, padding='same')

  self.conv20 = torch.nn.Conv2d(128, 64, kernel_size=2, padding='same')
  self.conv21 = torch.nn.Conv2d(128, 64, kernel_size=3, padding='same')
  self.conv22 = torch.nn.Conv2d(64, 64, kernel_size=3, padding='same')

  self.conv23 = torch.nn.Conv2d(64, 1, kernel_size=1)
```

- 23 Convolutional Layers
  - Inputs: 1 → 64 → 128 → 256 → 512 → 1024 → 512 → 256 → 128 → 64 → 1
- ReLU function between each layer
- Max-pooling/Upsampling operations between each set of layers

What Worked and Didn’t Work

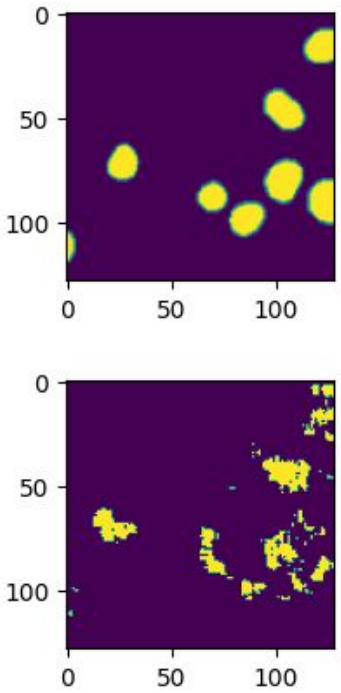
Worked

- Number of Epochs = 20
- Learning Rate (Alpha) = 0.001
- Batch Size increase
- MSE range: 3-4

Didn’t Work

- Number of Epochs < 20
- Learning Rate (Alpha) = 0.01 or 0.0001
- Batch Size < 64
- Modifying model to have less max-pooling operations
- MSE range: 5-6

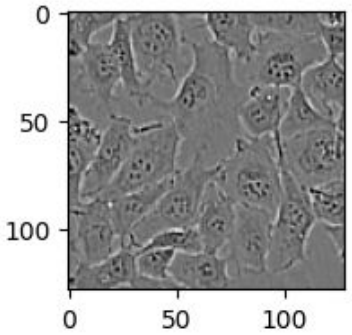
Parameters That Didn’t Work Example

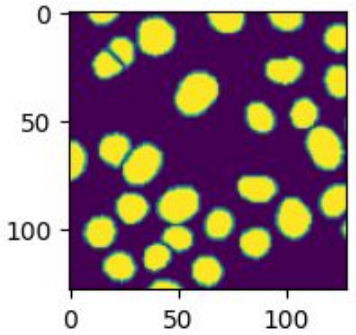


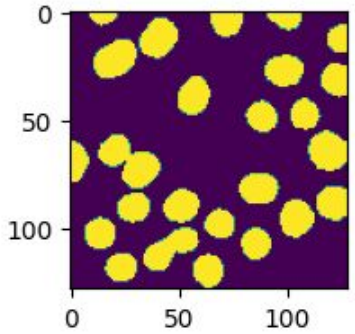
- Number of Epochs = 20
- Learning Rate = 0.01
- Batch Size = 16
- The model had difficulty in differentiating the cells
- Each predicted “cell” does not seem fully shaped

My Final U-Net Model

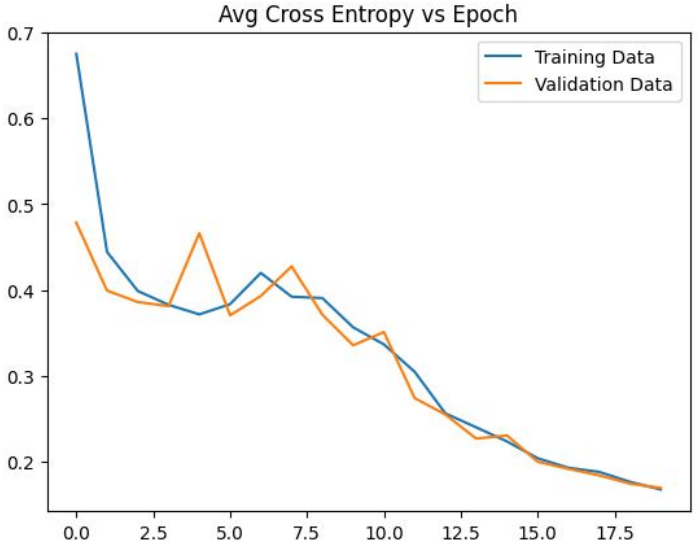
- Entire base structure
- Number of Epochs = 20
- Learning Rate (Alpha) = 0.001
- Batch Size = 64
- Best MSE = 3.31

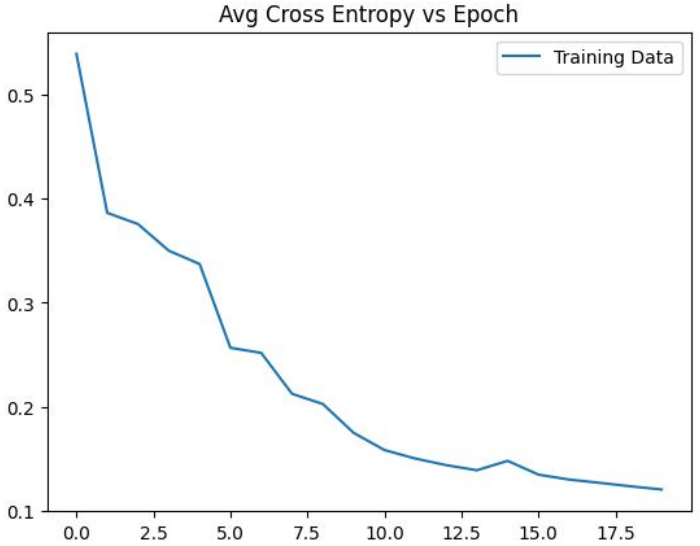




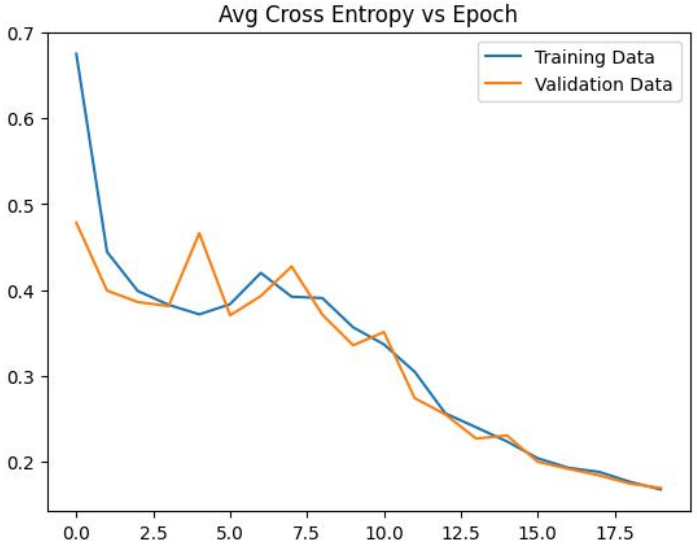


My Final U-Net Model: Convergence Plots





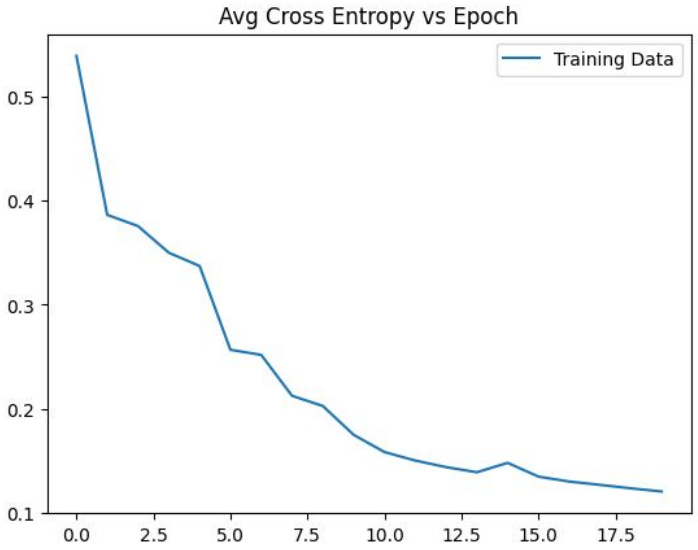
My Final U-Net Model: Convergence Plots Notes



- We see a downward trend in both sets
- While there are a few increases in the cost, this is to be expected
- As long as we see the values decrease and converge, the model is working

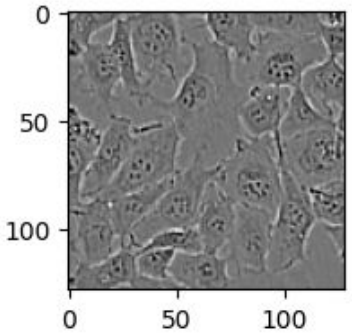
My Final U-Net Model: Convergence Plots Notes

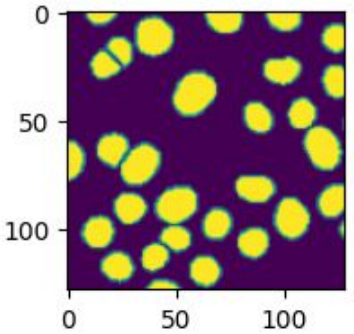
- We see a downward trend for the full training dataset
- Very little increases in cost
- By epoch 16-17 it starts to converge

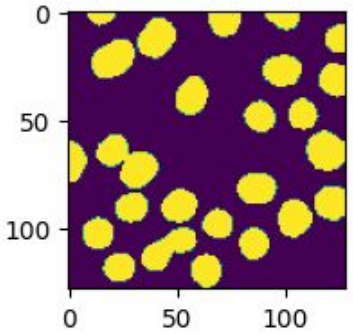


My Final U-Net Model: Prediction Example

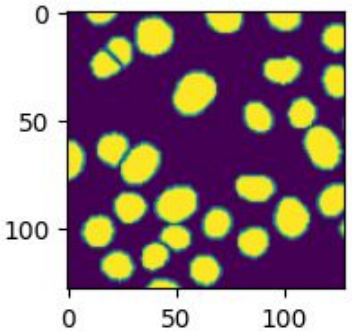
- Original Image (Left)
- Actual Cell Outlines (Right)
- Predicted Cell Outlines (Bottom)





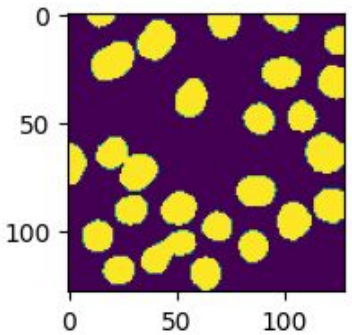


Thoughts on Model Output



Actual

Predicted



- Predicted looks fairly close to Actual
- Each predicted cell is well defined
- Appears to have a hard time differentiating cells that are close to each other

Potential Improvements for the Model

- Instead of having less max-pooling operations, an additional operation along with an eventual upsample might improve the predicted outputs
- Even further testing with the learning rate and batch size parameters
- Changing the input and output values for each convolutional layer
- Testing different kernel sizes for the max-pooling and upsampling operations

Final Thoughts

- Overall, the model performed well with the best MSE value being 3.31
    - In other words, on average our predictions were off by 3.31 cells in terms of counts
- The prediction images, looked well-defined
    - Only ran into problems when detecting cells that were right next to each other
- Even more fine-tuning can only make this model even better, especially with the before mentioned suggestions for improvement

References

Images for CNN Model Background:

https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

Images for U-Net Architecture:

https://arxiv.org/pdf/1505.04597.pdf