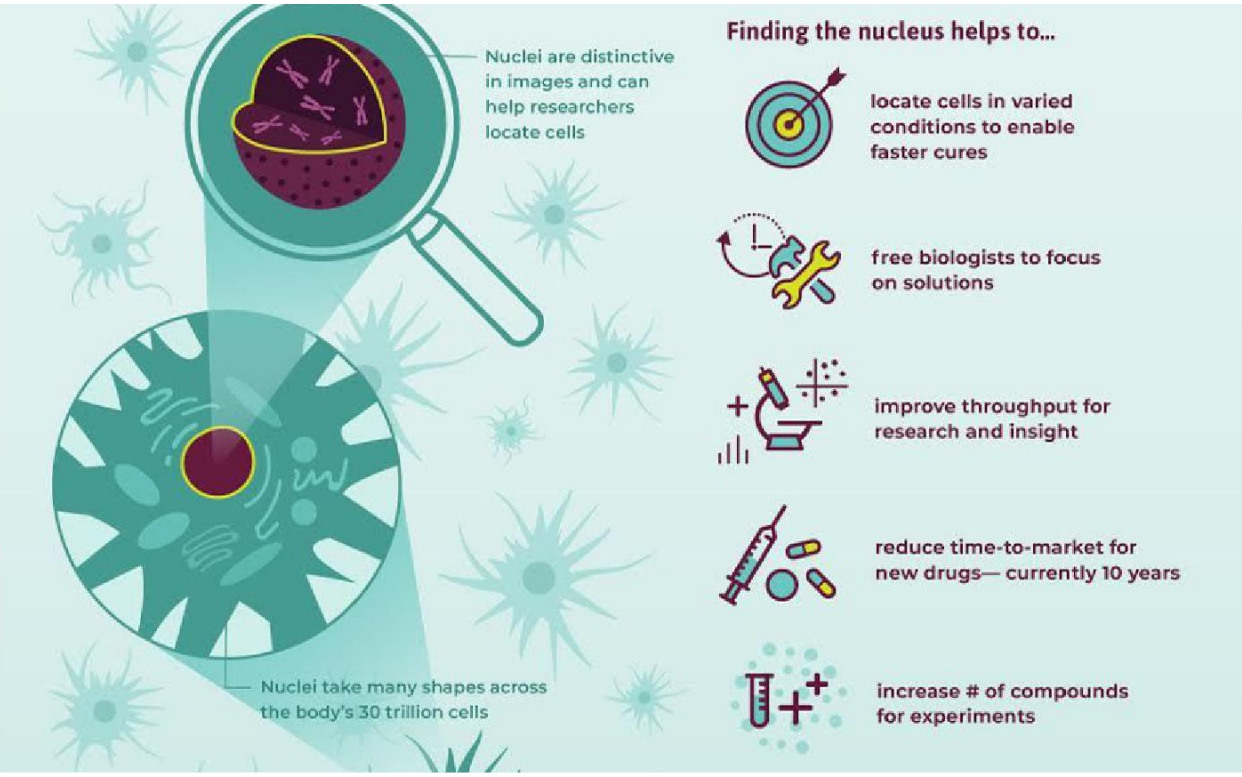
# Find the nuclei in divergent images to advance medical discovery



# Definition

## Project Overview

Imagine speeding up research for almost every disease, from lung cancer and heart disease to disorders. We've all seen people suffer from diseases like cancer, heart disease, chronic obstructive pulmonary disease. Think how many lives would be transformed if cures come faster. So we build nucleus detection project with the help of deep learning. We build model that identify a range ofcells' nucleus and you could help unlock cures faster-from rare disorders to the common cold.

## Problem Statement

Identifying the cells' nuclei is the starting point for most analysis because human body's 30 trillion cells contain a nucleus full of DNA, the genetic code that programs each cell. Identifying nuclei allows researchers to determine each cell in a sample, and by measuring how cells react to various treatments, the researcher can understand the underlying biological processes at work.

I've used this dataset - Find the nuclei in divergent images to advance medical discoveryand thus created a computer model that can identify a range of nuclei across varied conditions. By observing patterns, asking questions, and building a model.

## Dataset

Our dataset contains a large number nuclei images.there are different different type cell available inside our images.

We have already training and testing data available, Each image is represented by an associated ImageId. Files belonging to an image are contained in a folder with this ImageId. Within this folder are two subfolders, Images and Mask,

- Imagesthis folder contains the image file. And every image name is same main folder name
- masksthis folder contains the segmented masks of each nucleus. This folder is only included in the training set. Each mask contains one nucleus.

## File descriptions :

- /train/*- training set images (images and annotated masks) Download
- /test/*- stage 1 test set images (images only, you are predicting the masks) Download

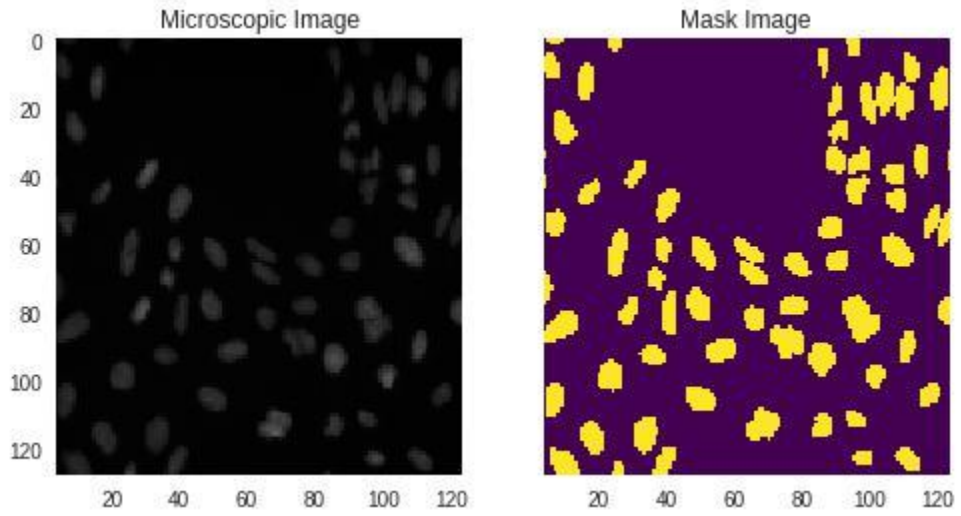
The most important file is train training set images and annotated masks, we have 670 training examples, there are different sizes images in our dataset but we resize those images to fit in our machine learning model, our image resized size is 256 * 256 pixels and depth is 3

# Analysis

## Data Exploration

This dataset contains a large number of segmented nuclei images. The images were acquired under a variety of conditions and vary in the cell type, magnification, and imaging modality (brightfield vs. fluorescence). We have already training and testing data available, Each image is represented by an associated ImageId. Files belonging to an image are contained in a folder with this ImageId. Within this folder are two subfolders, Images and Mask,

Imagesfolder contain microscopic cell images and Maskfolder contains the segmented masks of each nucleus. This folder is only included in the training set. Each mask contains one nucleus.If we combine all masks and make one image we found very similar microscopic images. I show you how microscopic image and after I combine masks images look like.show below picture:




Every images has different pixels and sizes. But we resize image ad make 256 * 256 * 3.

## Metrics

Submissions are evaluated using the Multiclass logarithmic loss.Multiclass logarithmic loss is used to assess predictive models in high-cardinality classification problems. Each image has been labeled with one mask image file. For each image, you must submit a set of predicted mask image (one for every image). The formula is then,

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(p_{ij}\right)$$

where N is the number of images in the test set, M is the number of image class labels as different type Image Mask file, log is the natural logarithm, y is the true mask image and p is predicted mask image.

This project is evaluated on the mean average precision at different intersection over union (IoU) thresholds. The IoU of a proposed set of object pixels and a set of true object pixels is calculated as:

$$IoU\left(A, B\right) = \frac{A}{A} \frac{\cap}{\cup} \frac{B}{B}.$$

The metric sweeps over a range of IoU thresholds, at each point calculating an average precision value. The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). In other words, at a threshold of 0.5, a predicted object is considered a "hit" if its intersection over union with a ground truth object is greater than 0.5.

At each threshold value tt, a precision value is calculated based on the number of true positives (TP), false negatives (FN), and false positives (FP) resulting from comparing the predicted object to all ground truth objects:

$$\frac{TP\left(t\right)}{TP\left(t\right) + FP\left(t\right) + FN\left(t\right)}.$$

A true positive is counted when a single predicted object matches a ground truth object with an IoU above the threshold. A false positive indicates a predicted object had no associated ground truth object. A false negative indicates a ground truth object had no associated predicted object.

The average precision of a single image is then calculated as the mean of the above precision values at each IoU threshold:

$$\frac{1}{|thresholds|} \sum_{t} \frac{TP(t)}{TP(t) + FP(t) + FN(t)}.$$

Lastly, the score returned by the project metric is the mean taken over the individual average precisions of each image in the dataset.

```
def IoU(y_true, y_pred):
    prec = []
    for t in np.arange(0.5, 1.0, 0.05):
        y_pred_ = tf.to_int32(y_pred > t)
        score, up_opt = tf.metrics.mean_iou(y_true, y_pred_, 2)
        K.get_session().run(tf.local_variables_initializer())
        with tf.control_dependencies([up_opt]):
            score = tf.identity(score)
        prec.append(score)
    return K.mean(K.stack(prec), axis=0)
```

In this method has two parameter y_true and y_pred, first we make prec array, we are set the loop with (0.5, 1.0, 0.05). when loop run we found single value for each loop, we have some thresholds like this. (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). when loop run we found single threshold value. for example when loop run first time run we found o.5 value, we compare this threshold to our y_pred .(if predicted y_pred is greater than 0.5 we return 1 else 0). Then we calculate mean IoU score. we add those score value in our list, for each loop same process happen compare thresholds values calculate mean IoU score and add IoU score in our list. we have 10 threshold so our loop run ten time and all those process happen 10 time, so in our list have 10 scores available finally we calculate mean in our score list with axis 0. Then we found our intersection over union score.

## Benchmark Model

The model with the Private Leaderboard score(Intersection Over Union) of 0.614 will be used as a benchmark model. Attempt will be made so that score(Intersection Over Union) obtained will be among the top 50% of the Private Leaderboard submissions.

# Methodology

## Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.In our case our dataset is microscopic cells' nuclei images, there are no any missing values and our data is already separated training and testing, we use **Feature scaling**technique.

**Feature scaling**is a method used to standardize the range of independent variables or features of data. In data processing,it is also known as data normalization and is generally performed during the data preprocessing step. The simplest method is rescaling the range of features to scale the range in [0, 1] or [−1, 1]. Selecting the target range depends on the nature of the data. The general formula is given as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where x is an original value X' is the normalized value in shortest way we are this X = X / 277 in python.

## Implementation

In this project I am build convolutional neural network,below I describe about convolutional neural network.

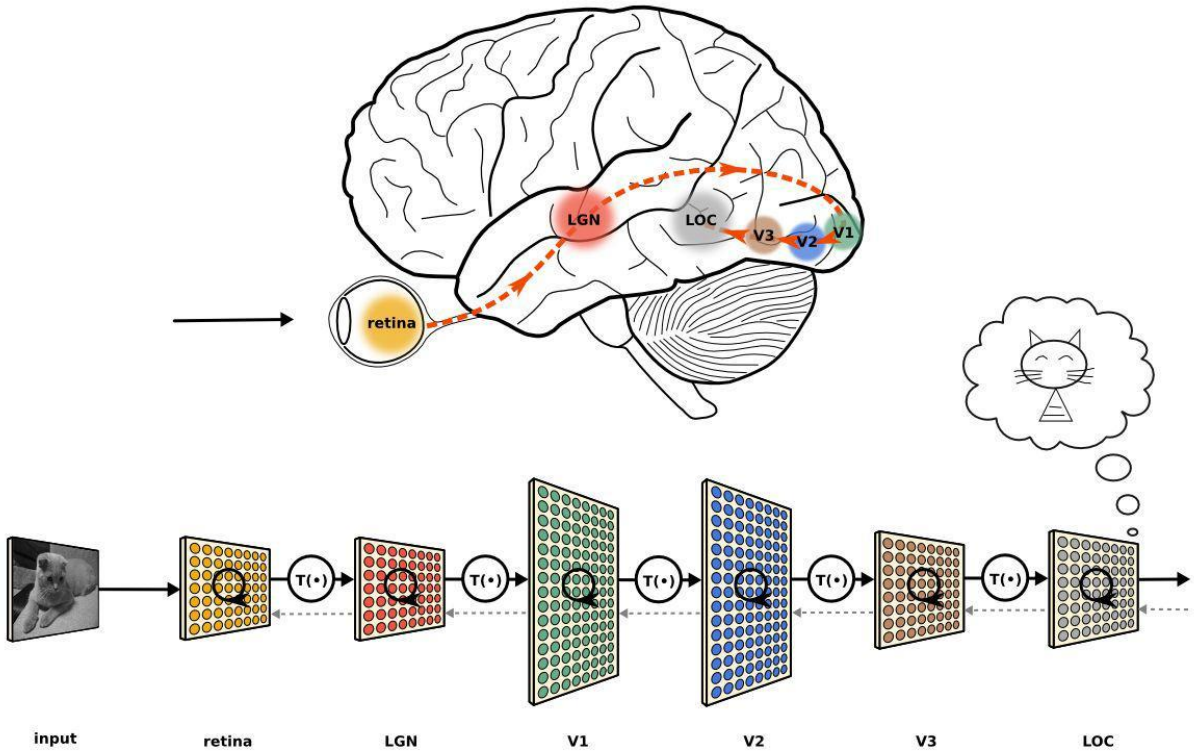
## Convolutional neural network

In 1988 Yann LeCun drew inspiration from the working of the visual cortex of the brain to create the first version of the convolutional neural networks. Alex Krizhevsky and Ilya Sutskever PhD students in Geoffrey Hinton's lab improved the convolutional neural networks algorithm to make it more efficient. They used the convolutional neural nets to win ImageNet competition where they dropped the classification error from 25% to 16%. And this also kickstarted the popularity of neural nets. Since 2012 companies have been increasingly using neural nets at their core of the services. For example, Google for their photo search, Amazon for their product recommendations and Facebook uses neural nets for their automatic tagging algorithms.

The first thing that humans do when they are born is to start recognizing or identifying things. The visual cortex of the brain plays an important role in this identification. This mechanism helped inspire the working of the convolutional neural networks. The visual cortex contains a complex arrangement of cells. These cells are sensitive to some regions of the visual field, called a receptive field. These cells act as local filters over the input space and exploit the strong spatially local correlation present in natural images. Essentially they trigger some neurons as a response to the visual stimuli.

Hubel and Wiesel found out that neurons in the visual context were organized in a columnar architecture and together they produced visual perception. The idea was that each layer performed a specific task. For example, when a cricket player is seeing an incoming ball lots of things happen in the visual cortex of the brain. In the brain we have layers known as v1,v2,v3,v4 and v5 so one layer detects the speed, the other layer detects the color, the other the shape and so on. Together this information in different layers are combined in the visual cortex of the brain

after which the humans will be able to detect the object. This concept of visual cortex in the brain is an inspiration for convolutional neural networks.



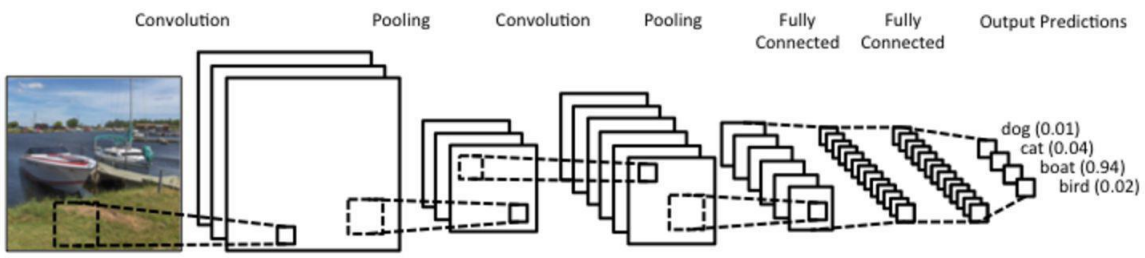

## Architecture of Convolutional neural network

The first two layers are convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches through a set of weights called a filter bank. The result of this locally weighted sum is then passed through a non-linear function such as a ReLU, sigmoid, hyperbolic tan etc. All units in a feature map have same filter banks but all feature maps will have different filter banks in a layer.

The architecture of a Convolutional Neural Network (CNN) is designed to take advantage of the 2D structure of an input image. This is achieved with local connections and tied weights followed by some form of pooling. This results in translation of invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. CNNs are basically just several layers of convolutions with nonlinear activation functions applied to the results.

In a traditional feedforward neural network, we connect each input neuron to each output neuron in the next layer. In CNNs we don't do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands like the ones showed above, and combines their results.

The architecture of typical convolutional neural networks are a series of well-defined stages as shown in the figure below:




## Layers of convolution neural network

A convolutional neural network consists of several layers. These layers can be of three types:

## Convolution Layer

In this layer, what happens is exactly what we saw in case 5 above. Suppose we have an image of size 6*6. We define a weight matrix which extracts certain features from the images:

INPUT IMAGE

| | | | | | |
|---|---|---|---|---|---|
| 18 | 54 | 51 | 239 | 244 | 188 |
| 55 | 121 | 75 | 78 | 95 | 88 |
| 35 | 24 | 204 | 113 | 109 | 221 |
| 3 | 154 | 104 | 235 | 25 | 130 |
| 15 | 253 | 225 | 159 | 78 | 233 |
| 68 | 85 | 180 | 214 | 245 | 0 |

WEIGHT

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| |
|---|
| 429 |

We have initialized the weight as a 3*3 matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output. The value 429 above, is obtained by the adding the values obtained by element wise multiplication of the weight matrix and the highlighted 3*3 part of the input image.

The 6*6 image is now converted into a 4*4 image. Think of weight matrix like a paintbrush painting a wall. The brush first paints the wall horizontally and then comes down and paints the next row horizontally. Pixel values are used again when the weight matrix moves along the image. This basically enables parameter sharing in a convolutional neural network.
The weight matrix behaves like a filter in an image extracting particular information from the original image matrix. A weight combination might be extracting edges, while another one might a particular color, while another one might just blur the unwanted noise.

The weights are learnt such that the loss function is minimized similar to an MLP. Therefore weights are learnt to extract features from the original image which help the network in correct prediction. When we have multiple convolutional layers, the initial layer extract more generic

features, while as the network gets deeper, the features extracted by the weight matrices are more and more complex and more suited to the problem at hand.

## Padding and Strides

As we saw above, the filter or the weight matrix, was moving across the entire image moving one pixel at a time. We can define it like a hyperparameter, as to how we would want the weight matrix to move across the image. If the weight matrix moves 1 pixel at a time, we call it as a stride of 1. Let's see how a stride of 2 would look like.

INPUT IMAGE

| | | | | |
|---|---|---|---|---|
| 18 | 54 | 51 | 239 | 244 |
| 55 | 121 | 75 | 78 | 95 |
| 35 | 24 | 204 | 113 | 109 |
| 3 | 154 | 104 | 235 | 25 |
| 15 | 253 | 225 | 159 | 78 |

WEIGHT

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| |
|---|
| 429 |

As you can see the size of image keeps on reducing as we increase the stride value. Padding the input image with zeros across it solves this problem for us. We can also add more than one layer of zeros around the image in case of higher stride values.

We can see in below image how the initial shape of the image is retained after we padded the image with a zero. This is known as **same padding** since the output image has the same size as the input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 18 | 54 | 51 | 239 | 244 | 188 | 0 |
| 0 | 55 | 121 | 75 | 78 | 95 | 88 | 0 |
| 0 | 35 | 24 | 204 | 113 | 109 | 221 | 0 |
| 0 | 3 | 154 | 104 | 235 | 25 | 130 | 0 |
| 0 | 15 | 253 | 225 | 159 | 78 | 233 | 0 |
| 0 | 68 | 85 | 180 | 214 | 245 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

This is known as **same padding** (which means that we considered only the valid pixels of the input image). The middle 4*4 pixels would be the same. Here we have retained more information from the borders and have also preserved the size of the image.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 18 | 54 | 51 | 239 | 244 | 188 | 0 |
| 0 | 55 | 121 | 75 | 78 | 95 | 88 | 0 |
| 0 | 35 | 24 | 204 | 113 | 109 | 221 | 0 |
| 0 | 3 | 154 | 104 | 235 | 25 | 130 | 0 |
| 0 | 15 | 253 | 225 | 159 | 78 | 233 | 0 |
| 0 | 68 | 85 | 180 | 214 | 245 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

WEIGHT

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

139

## Pooling Layer

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers. Pooling is done for the sole purpose of reducing the spatial size of the image. Pooling is done independently on each depth dimension, therefore the depth of the image remains unchanged. The most common form of pooling layer generally applied is the max pooling.

| | | | |
|---|---|---|---|
| 429 | 505 | 686 | 856 |
| 261 | 792 | 412 | 640 |
| 633 | 653 | 851 | 751 |
| 608 | 913 | 713 | 657 |

| | |
|---|---|
| 792 | 856 |
| 913 | 851 |

Here we have taken stride as 2, while pooling size also as 2. The max operation is applied to each depth dimension of the convolved output. As you can see, the 4*4 convolved output has become 2*2 after the max pooling operation.

## Fully Connected Layer

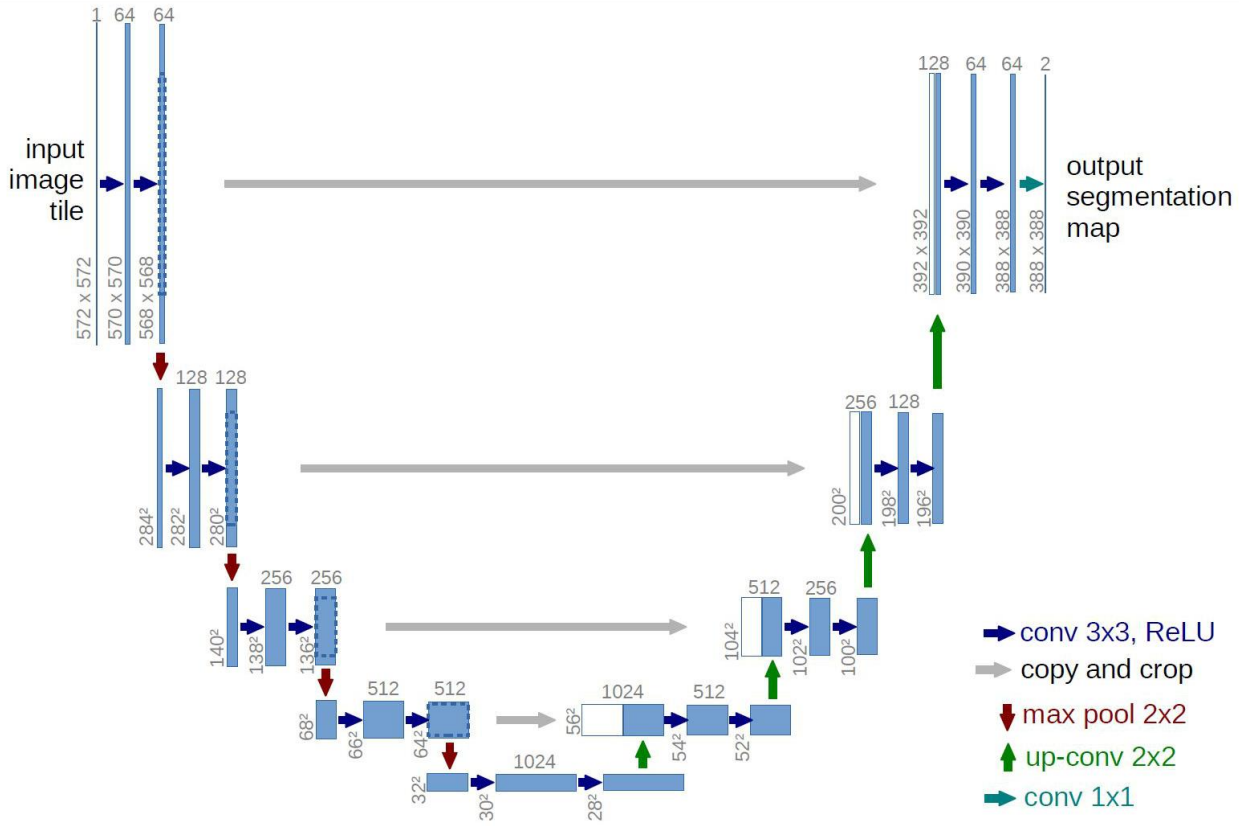
Finally, after several convolutional and max-pooling layers, the high-level reasoning in the neural network is done via fully connected layers. A fully connected layer takes all neurons in the previous layer and connects it to every single neuron it has. Fully connected layers are not spatially located anymore (you can visualize them as one-dimensional), so there can be no convolutional layers after a fully connected layer

Convolutional neural nets (CNN) are automatically trained to learn the values of its filters based on the task assigned. For example, in Image Classification, a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features basically just several layers of convolutions with nonlinear activation functions applied to the results.

## Convolutional neural network architecture

There are several types of architecture available in Convolutional neural nets example inception, U-net, LeNet 4, LeNet 5 and much more. But we use U-net CNN architecture because The u-net is

convolutional network architecture for fast and precise segmentation of images. Up to now it has outperformed the prior best method (a sliding-window convolutional network) on the ISBI challenge for segmentation of neuronal structures in electron microscopic stacks. It has won the Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography at ISBI 2015,and it has won the Cell Tracking Challenge at ISBI 2015on the two most challenging transmitted light microscopy categories (Phase contrast and DIC microscopy) by a large margin (See also our announcement).




U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the

box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The network architecture is illustrated in above image. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers. To allow a seamless tiling of the output segmentation map (see Figure 2), It is important to select the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size.

## Train the Model

The input microscopic images and their corresponding segmentation masks are used to train the network with the stochastic gradient descent implementation of Caffe.we are make only 8 batches. We set 40 Epochs, model take 10% validation data, also we set some callbacks EarlyStoppingand ModelCheckpoint.

- **ModelCheckpoint**:For example: If filepath is model-capstone_project_udacity_by_jimit_jaishwal.h5, then the model checkpoints will be saved with the epoch number and the validation loss in the filename.
- **EarlyStopping**: Stop training when a monitored quantity has stopped improving.

## Refinement

The initial model predict IoU metric score is 80% with just 40 Epochs. I perform following improvements to improve IoU metric score:

- I decrease batch_size 8, and set Epoch size 40. I do that because we have very small dataset only 670 training features.

After those changes, my model now predicts with 80.09% IoU metric score just 40 Epochs on Validation score.

# Results

## Model Evaluation and Validation

At the end of training, I was the achieving following:

| | |
|---|---|
| **Training Loss** | **0.0638** |
| **Training IoU(Intersection over union) score** | **79.96%** |
| **Validation Loss** | **0.0792** |
| **Validation IoU(Intersection over union) score** | **80.09%** |

According our benchmark this model was much good and accurate prediction.

**The final model is made up of the following:**

- We implemented convolution for depth at 16, 32, 64, 128 and 256.
- We implemented Max pooling when we go deep in our neural network architecture.
- We implemented Upsampling in our neural network architecture.

- As activation function we use ReLu all time and as output Layer we use Sigmoid activation function.
- We use Adam as our Optimizer.
- We use IoU score as our benchmark.

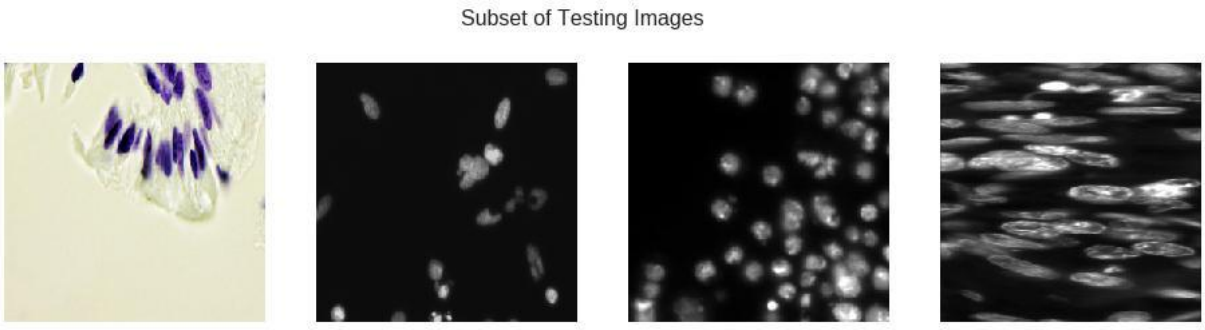
## Justification

This model use in medical research laboratory where researcher identify different type cell's nucleus condition and identify good and bad cell's condition. Since we have 10000 or more images to train on, we expect the IoU score to continue improve.

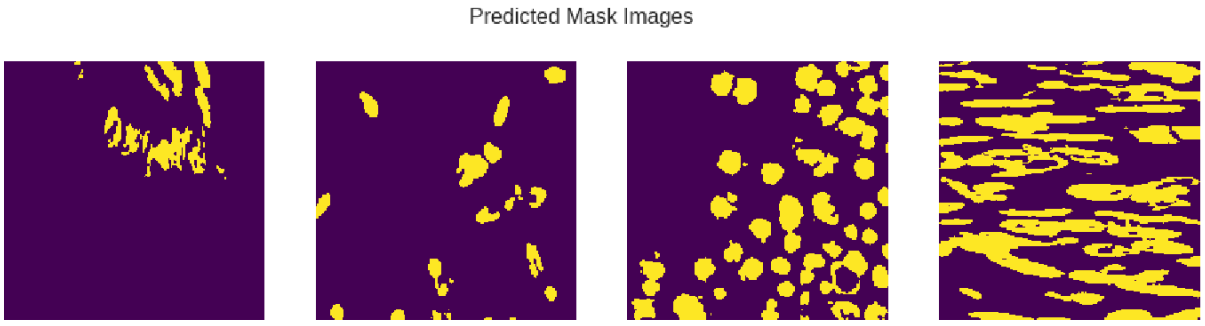
# Conclusion

## Free-From Visualization

To test how well our model working? I create 4 random images from test images and check how prediction is? Hear is actual images.




Here is our predicted masks of images.




We can see our model did a pretty job at predicting data and it did so with a training set of just over 670. If we continue to train with a larger set of 10000 images, we will achieve even batter result.

## Reflection

The process used to define and develop the model can be summarised as follows :

- An initial problem is identified and relevant datasets were obtained.
- The datasets were downloaded and pre-processed.
- Relevant hardware is obtained from cloud to account for huge number of images in the dataset
- We search internet convolution neural network architecture who batter for image segmentation and we found U-net architecture who are good for image segmentation.
- We implement U-net architecture after we build model the model has 1,941,105 trainable parameter
- We train model the model with google codeLab because we have not good computing power.
- After training we visualizing our microscopic image, mask image and predicted mask image.

- Then I predict test dataset, and visualizing randomly 4 test images and predicted masks.
- Finally I save predicted test dataset image masks file in local disk.

Training stage was too difficult to me because of the fact that training and testing took large time despite running on GPU instances.thats way i use google codelab for computing power.

## Improvement

Following are the areas for improvement

- To improve our IoU metric score we need more training images. If we continue to train with a larger set of 10000 or more images, we will achieve even batter result.
- Currently we are use U-net convolutional neural network architecture. We also need to try following architecture and verify the sores.because those model are good for biological image segmentation.
    - Deep Contextual Networks for Neuronal Structure Segmentation
    - AxonDeepSeg

## Reference


- https://en.wikipedia.org/wiki/Feature_scaling
- https://storage.googleapis.com/kaggle-media/competitions/dsb-2018/dsb.jpg
- https://www.kaggle.com/c/data-science-bowl-2018/data
- https://arxiv.org/pdf/1505.04597.pdf
- https://www.datasciencecentral.com/profiles/blogs/polymorphic-malware-detection-using-sequence-classification
- https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional -neural-networks-simplified-demystified/
- https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277

- http://blog.arimaresearch.com/convolutional-neural-network-cnn/
- https://www.youtube.com/watch?v=eHwkfhmJexs&feature=youtu.be
- https://www.kaggle.com/c/data-science-bowl-2018#evaluation
- https://www.kaggle.com/c/data-science-bowl-2018#description
- https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/