Machine Learning Engineer Nanodegree

Detecting cervix type using Deep Learning

Moises Vargas

October 29st, 2017

I. Definition

Project Overview

Cervical cancer has become one of the most important causes of death in womans by cancer at global level [1]. In the past years artificial intelligence has played important role on detection of cervical cancer. Department of Biomedical Engineering of University of Malaya has a review article which study 103 journal paper between 2010 and 2014 which shows works on intelligent systems to cervical cancer using techniques like artificial neural networks, support vector machines, decision trees among others [2].

Cervical cancer can take up to three decades to develop, due this time it is crucial that cervical cancer can be prevented on time allowing patients to receive the appropriate method of treatment. Even with having the proper technology to detect cervical cancer, many women at high risk for cervical cancer are receiving treatment that will not work for them due to the position of their cervix [3]. It is needed to have proper technology to determine which kind of treatment is appropriate for every patient. Intel in partnering with MobileODT opened a challenge in June 2017 with the aim of identify which type of cervix a patient has, given a cervix image [3]. The aim of this project is to develop a deep neural network that can detect which type of cervix has a patient.

Problem Statement

Cervical cancer stages can be detected with current artificial intelligence methods, however there are different treatment depending on patient physiology. Detecting quickly which type of cervix a patient has, would lower the risk of apply the wrong treatment to a patient and will level up the survival possibilities of womans with high risk for cervical cancer. Developing an artificial intelligence method to detect what kind of cervix a patient has, base on a cervix image would potentially help the patient to receive the correct treatment according with his physiology.

This document propose use the LeNet 5 architecture [4] as benchmark model, feed
a resized version of the original data 200x200 pixels, with RGB channels and measure the training, validation and test sets to obtain the baseline accuracy.

This project propose to develop a convolutional neural network(ConvNet) which will receive an cervix image and the ConvNet will predict which type of cervix the image is. The baseline model is a ConvNet with well known architecture LeNet-5 [4], after tuning and image preprocessing the performance will be measured. The aim is to improve the baseline model a get better results by modifying parameters and network architecture.

Metrics

The accuracy metric proposed to use on both solution and benchmark model will measure how well the model is performing by counting the number of correct predictions. For instance for a data set the model will predict for each example which type of cervix this example belongs to, if data set contains 100 examples and only 50 were correctly predicted the accuracy if defines as follow;
50/100 = 0.5 it traduce to the model performance is 50% of accuracy. In general the metric is defined as follow (number of correct predictions)/(total examples)

II. Analysis

Data Exploration

The data used in this project come from kaggle competition [5], data is comprised of image files and are organized in folder according with its cervix type label namely folder are Type_1, Type_2 and Type_3. The initial data from was in 7zip archive compression format, organized as follow a training set train.7z of 5.54 GB and test set test.7z 1.93 GB. Additional set for the three different classes was provided, additional_Type_1_v2.7z 4.49 GB, additional_Type_2_v2.7z 14.61 GB and additional_Type_3_v2.7z 6.66 GB.

After merge train data with additional examples, the total training example are 8210 is distributed as follows, train/Type_1 1438 samples, train/Type_2 4346 samples and train/Type_3 2426 samples.

The Figure below shows how data is imbalanced, being the images of cervix type 2 with most frequencies.

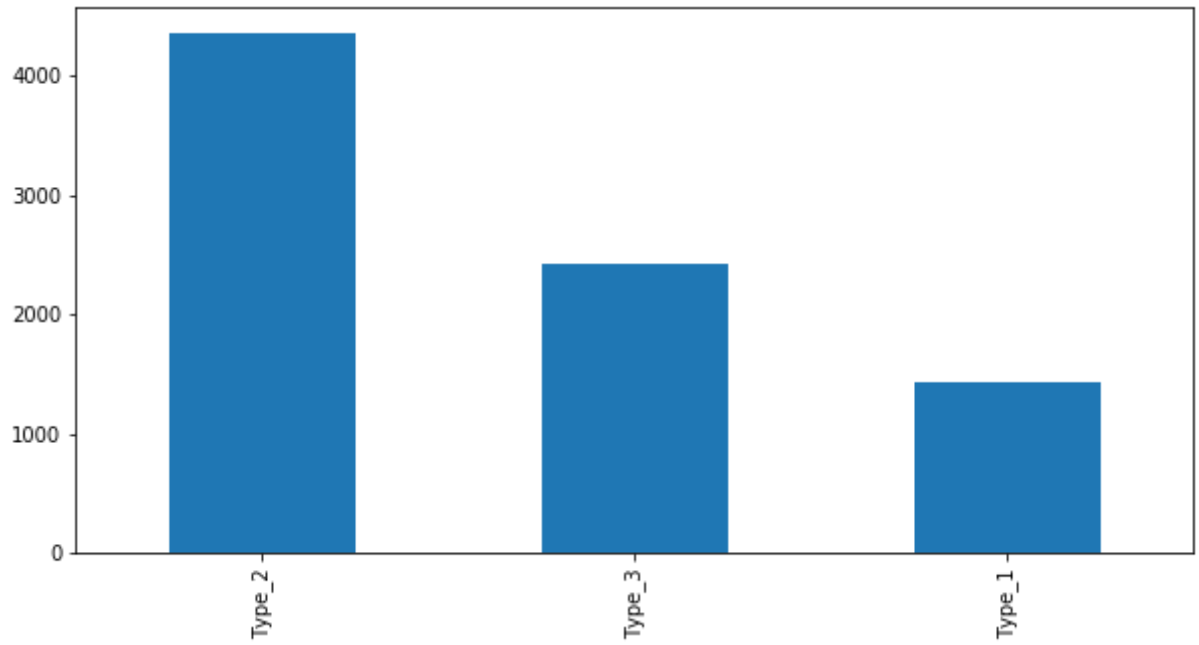


Figure-1

The next figures shows the images pixel distribution.

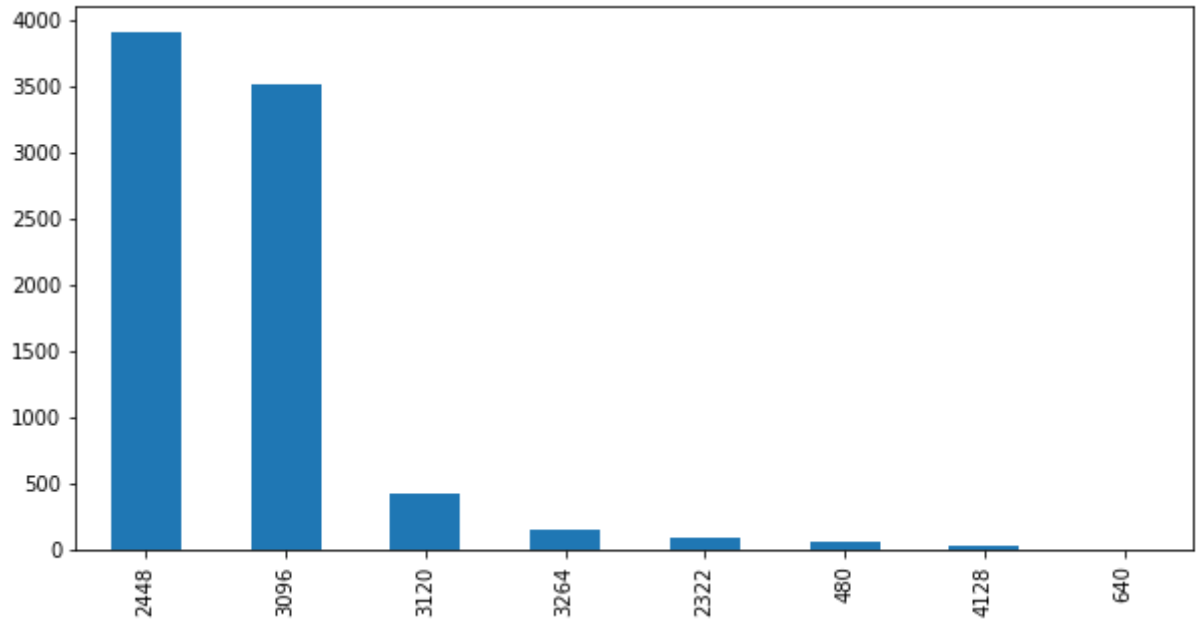


Figure-2

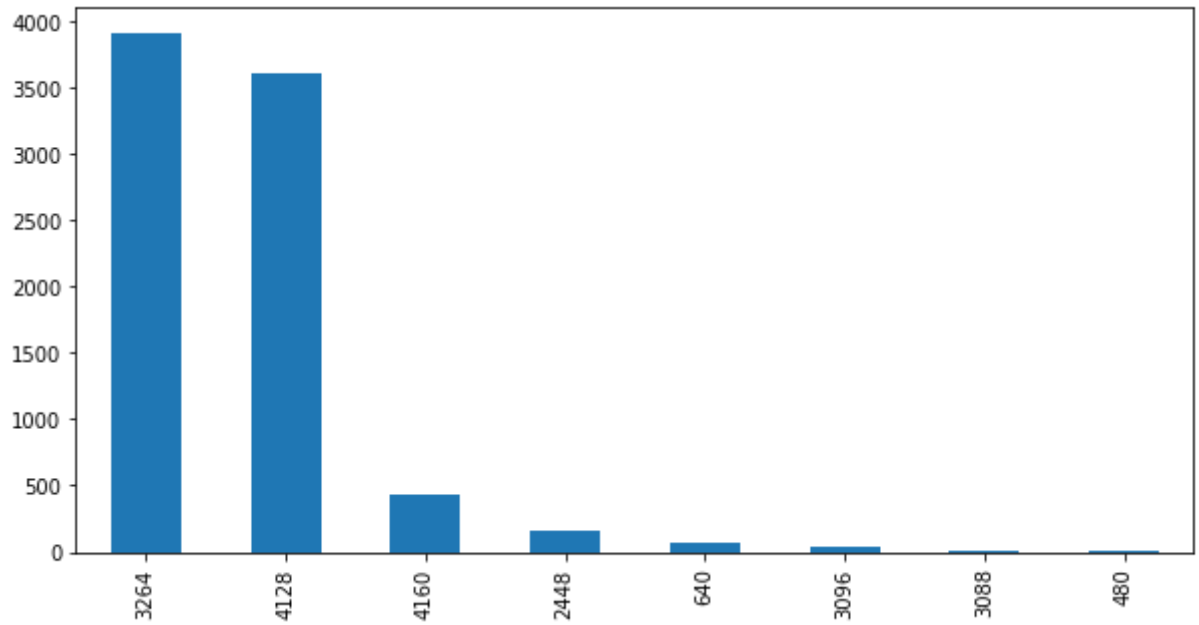


Figure-3

In this section, 33GB of data will be extracted out from its zip files, data will be organized in three directories for its corresponding cervix type i.e Type_1, Type_2 and Type_3.

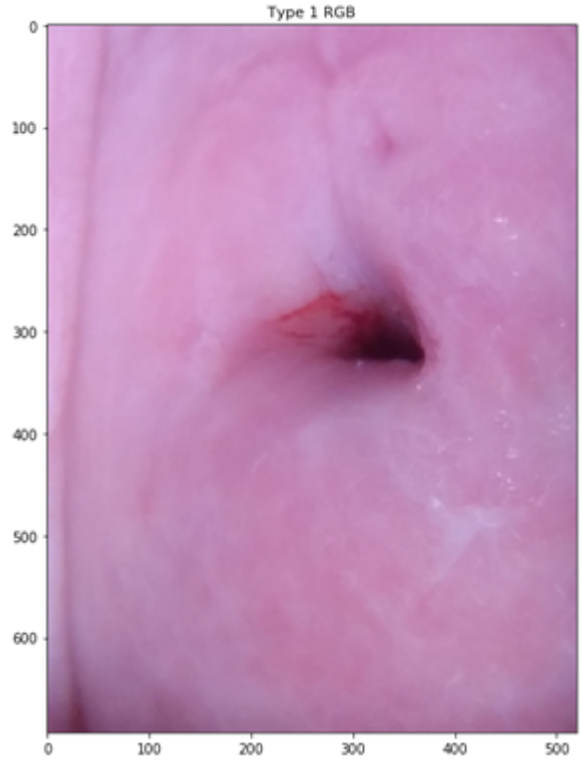
With a imaging processing tool, the images will be inspected to see in which format the images are, and identify possible corrupted images. If corrupted images are detected, they will be removed from the data set.

Explore and create basic counting of height and width of the images and find a good intermediate or reasonable image size to resize and normalize all images. The data set is 33GB of size in disk and this step aims to normalize the image size and reduce the images weight on space disk.

Exploratory Visualization

The images below shows 3 different types of cervix on this data set.

The images are in RGB and HSV to compare this two color spaces of the cervix that helps to differentiate its types.



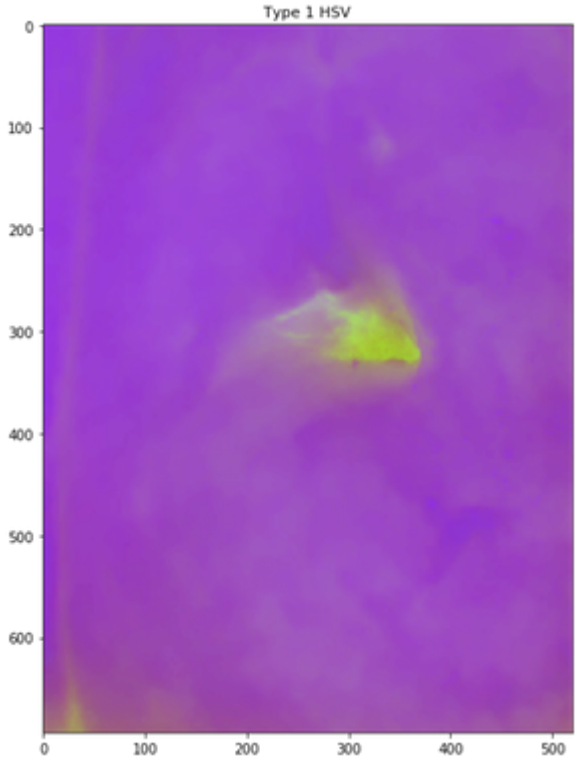
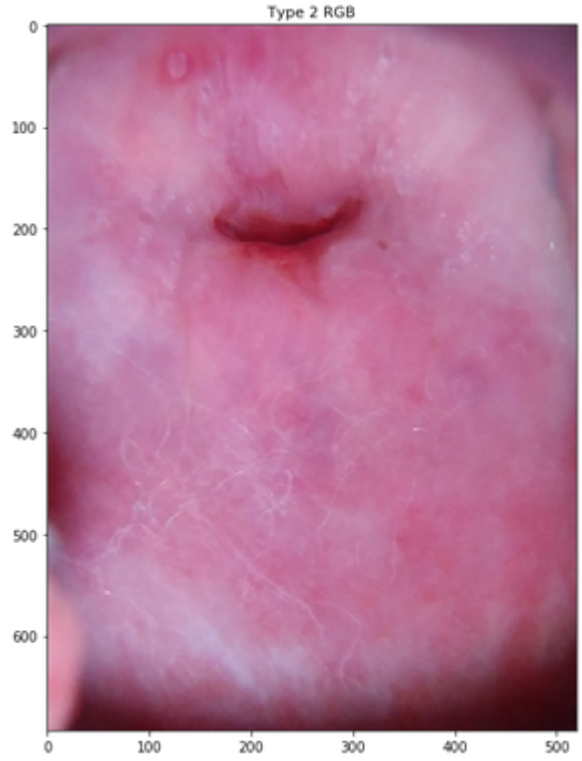


Figure-4



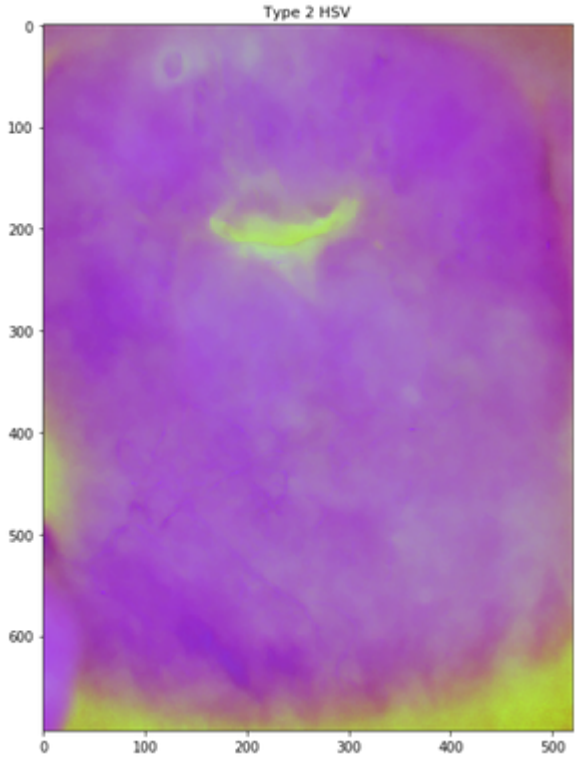


Figure-5

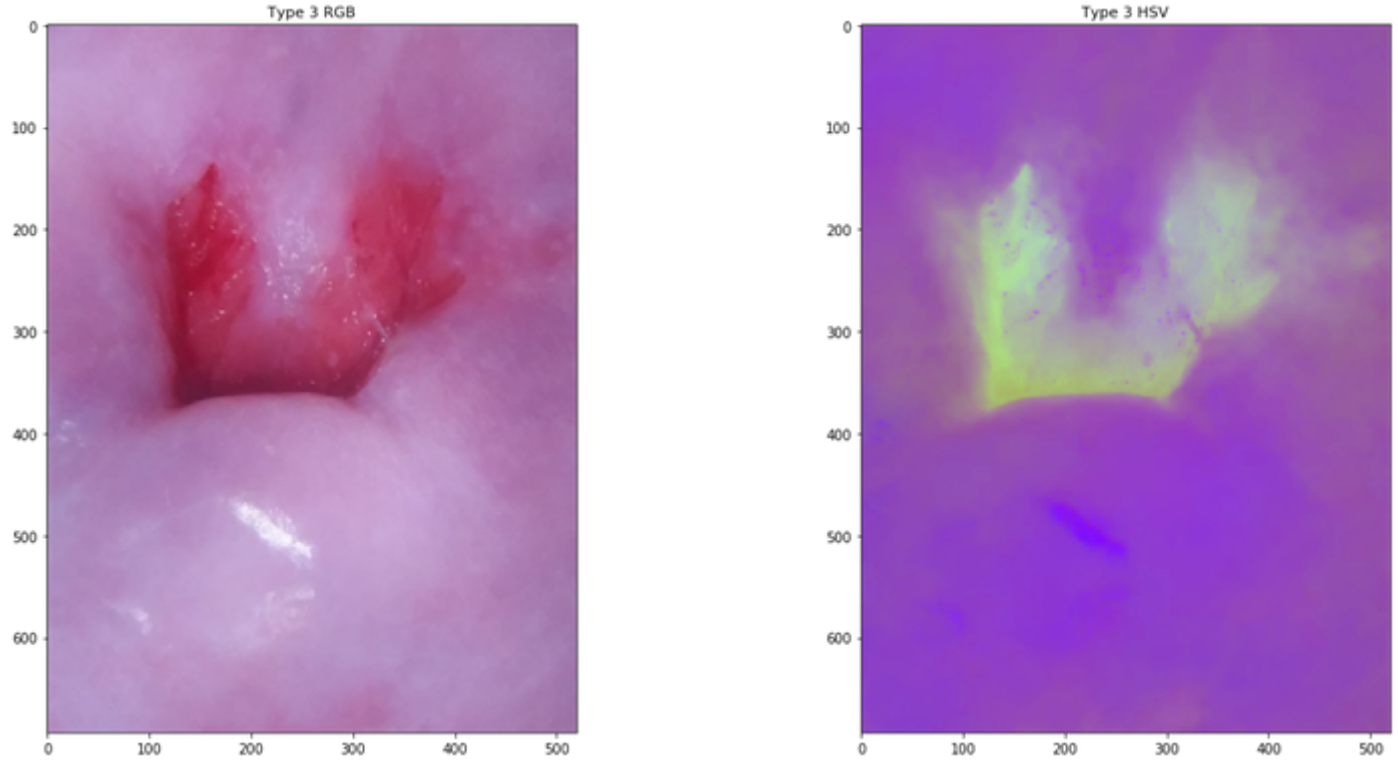


Figure-6

Algorithms and Techniques

As mention before this project use a Convolutional Neural Network (ConvNet) as classifier. The network has several parameters that needs to be tuned namely the parameters are; number of epochs, batch size, learning rate, max pool layers and dropout layers. This technique and its parameters are discussed above.

The ConvNet is trained using back-propagation to learn the weights parameters, this weights are used to activate neurons, the main difference of the ConvNet with respect to a regular neural network is that are specially designed to recognize visual patterns directly from the image pixels [4].

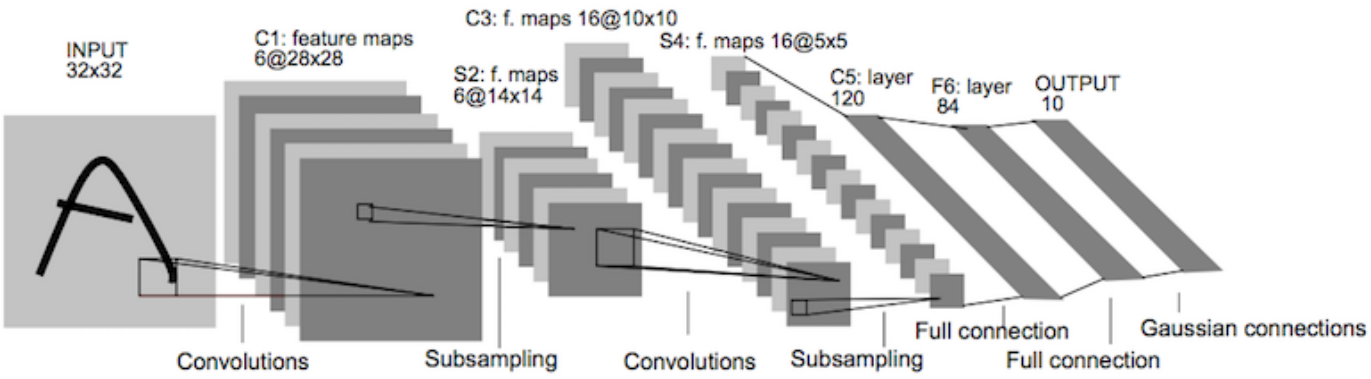


Figure-7 Source: Yan LeCun

The figure above shows the LeNet-5 architecture in which this project base its classifier. In the next paragraphs are intended to explain how this neural network architecture works when receive the image till the output.

Layer name	Size	Channels
Input	32x32	1
Convolution 1 (C1)	28x28	6
Sub-sampling	14x14	6
Convolution 2 (C2)	10x10	16
Sub-sampling	5x5	16
Fully connect (FC1)	120	-
Fully connect (FC2)	84	-
Output	10	-

Table-1 LeNet5 architecture

The table above resumes the architecture of this ConvNet, the input layer it is an image in this example are written digits images of 32x32 pixels, here can be assumed that image is grayscale such that has 1 channel. The dataset of this project will be treat as images with 3 channels color space for example Red, Green and Blue.

The first Convolution layer (C1) it is made by applying a convolution which is basically scanning the image with a patch of smaller size in this case the patch is 5x5 pixels of size and applying to it an activation function. This patch is applied over the image horizontal and vertical, once this process finish the result will be an image of 28x28, This

process will happen six times piling up the first convolution layer for the given image. Finally the dimensions of (C1) layer will be 6@(channels)28x28, this layer represent the automatic extraction of features from the image that was feed in the network.

The second layer is a sub-sampling (S2) that will have the same number of channels as the layer before and the features will be reduce by half ending with dimensions 6@14x14. It means that a given image after passing the (C1) will end up in (S2) with size of 14x14 pixels and 6 transformations that came from the (C1) layer. The sub-sampling use a function that is applied together with a patch, in this case the patch size if 2x2 and the function could be a Max function, for instance an image of 28x28 pixels you apply a Max function with patch size 2x2; every patch will choose one pixel with the max value, this process in 28x28 image will occur 196 times which end with output of size 14x14. This kind of layers aims to reduce the space of feature while retain the relevant features.

The third layer (C3) it is a convolution similar to the first (C1) but with size of 16@10x10 as a result of applying a patch of size 5x5 pixels. Fourth layer (S4) same as (S2) in this point applied to (C3) ending with size of 16@5x5.

Finally this network has two fully connected layer (FC5) its input is an unrolled version of layer (S4) which is a vector of 400 features (16x5x5) and its output is a vector of 100 features. Next six layer (FC6) receives as input vector of 100 features and return a vector of 84 features. The output layer receives the 84 features vector and returns a 10 probabilities vector, the value of each position of the vector represents the probability of that position to be the class that represent the original image that was feed in the input layer.

Beside to the convolution and Sub-sampling layers described above, there is another layer in this project call Dropout that aim to avoid the network overfit the data by randomly turn off connections between layers.

Parameter	**Description**
EPOCHS	Number of training passes
BATCH_SIZE	Batch size in training
learning rate	Initial rate for the optimizer

Table-2 Model parameters

Additional parameters

Neural networks in general works with weights which are the main parameters of the network, it work as the neuron connections and the objective of the network in this case is to classify which type of cervix is given an image, a good result in this task depends on how well this weights parameters get tuned by learning.

All parts of the network that were discussed above convolution layer with activation functions, sub-sampling layers, dropout layers and fully connected layers are the foundation structure that helps to convey the learning to the weight parameters.

The missing piece of this puzzle is the conveyor which is the technique that go interactively through the network updating the weights and this happen controlled by the parameters listed in the table-5. The conveyor that use the ConvNet in this project is well known as AdamOptimizer that as well as others optimizers use Back-propagation to optimize the values of the weights that better perform the neural network.

Since the ConvNet needs image data and usually this kind of data consumes good amount of computational memory it is need to train the network using batches that means passing a subset of data to the network in training process, to control this batches sizes this project use the parameter Batch Size for instance if the data contains 8000 images using batch size of 64 turns out to have 125 batches in training.

The Epoch parameter helps to control how long the training operation will take for instance an epoch of 3 and batch size of 64 with 8000 examples of training will cause the optimizer to learn in 375(3*125) batches where in each batch the network and the optimizer needs to digest 64 images to adjust the weight parameters.

Finally the learning rate which is a initial parameter of the optimizer it is optional since one characteristic of AdamOptimizer if that automatically initialize and update this learning rate. However controlling this parameter by given an initial value will affect the learning results by controlling who fast or slow the optimizer converge to find out the ideal weights.

Benchmark

The benchmark model proposed in this project is the LeNet-5 architecture configuration from the table-4 with a minor modification in the output layer to predict 3 different classes instead of 10. The benchmark use the configuration of table-7. According with the metrics defined in Metrics section, the table-8 reports the measurements of the baseline model that will use to compare with the final model. The data for the baseline model is randomly split in 60% for training 28% and 12% for test.

Parameter	Value
EPOCHS	15
BATCH_SIZE	128
learning rate	0.01

Table-3 base model parameters

Metric	Value
Train accuracy	0.654
Validation accuracy	0.553
Test Accuracy	0.551

Table-4 Baseline model metrics

III. Methodology

Data Preprocessing

As part of the training and predictions tasks, this project implements preprocessing pipeline before feed the data to the neural network. The images are transformed in size and color space. And the data is balanced to have similar number of example per class.

In exploration section the data of this project is described, the original zipped files train.7z additional_Type_1_v2.7z, additional_Type_2_v2.7z, additional_Type_3_v2.7z were unzipped and merged in a train folder. For each of the three classes a sub folder was created and the corrupted files where deleted.

Since the sizes of the images ranges from 3264x2448 pixels to 480x640 pixels, the images were normalized such that all images have the same dimension of 693x520. This helps not only to normalize the size of all images but to reduce the size of bytes of images, for instance an image that size in bytes was 2Mb ends up in 300Kb. The images also are numerically normalized, since pixel values range from 0 - 255, and optimizer prefer receive normalize data around zero, every image was divided by 255 a then subtract 0.5 for all pixel images.

In order to have a balanced dataset it was augmented using a data augmentation tool [6]. Since the majority class are images of type 2 with 4346 samples, the trick used to balance images of type 1 and type 3 was applying transformations for images of type 1 and type 3 in order to have similar number of examples per class. some of the transformation applied were add noise to images, rotation of 90 and 45 degrees and vertical flip.

In order to implement the final model was necessary to create helper functions to preprocessing the input data.

Been able to control the size of the image data when training ConvNet will be useful since it can reduce the amount of computation memory and the training time that the model can take. Also normalizing image data to be zero centered will help the model to converge better, below there are two function definitions one for resize the image and other to scale the image to be zero centered basically by dividing each component of the image with the max value which a pixel can take (255).

```
def resize_image(img, size=(820, 620), debug=False):
    """
    resize image and return it
    """
    h = size[0]
    w = size[1]
    img_ = cv2.resize(img, (w, h))
    return img_
```

```
def scale_image(img):
    """
    normalize the image
    """
    return img / 255.0 - 0.5
```

A function which receive an image and return it as the model needs was implemented to facilitate the image transformation. The code below shows this function named *lamba_layer* which receive an image, create a copy, apply the resize function, change the image color space, in this example this image is being converted from RGB to Gray, finally the image is scaled before return it to the model that will consume this data.

```
def lambda_layer(image):
    """
    pre-process the image before pass it to the
    ConvNet
    """
    img_ = np.copy(image)
    img_ = resize_image(img_, (H_IMG, W_IMG))
    img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2GRAY)
    img_ = img_[:, :, np.newaxis]
    img_ = scale_image(img_)
    return img_
```

Implementation

The implementation was done using pandas to store images information, opencv to read images from file system, scikit-learn as helper to split data in training, validation and tests sets and TensorFlow to implement the ConvNet.

This project use GPU-GRID K520 from AWS 4GB of memory, however all data image can not be feed into the ConvNet cause it will cause memory resource exhaustion, for that reason was necessary to implement a function generator to lazy generate batches of images that will be passed over the ConvNet. The code below shows how this implementation was done, it receives a panda data frame that contain images path and type then map this input to a randomize set of images that will be preprocessed in this generation function, this is done in batches, every time this function is called will generate set of images and types according with the batch size by default is 128.

```
def generator(samples,batch_size=128):
    """
    Generates random batches of data base on image path
    """

    num_samples = len(samples)
    while True:
        randomsamples = samples.sample(frac=1)
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]
```

```
            images = []
            labels = []
            for _, row in batch_samples.iterrows():
                image_path = row['image'].strip()

                image = cv2.imread(image_path)
                label = row['target']

                images.append(lambda_layer(image))
                labels.append(label)

            X_train = np.array(images)
            y_train = np.array(labels)

            yield sklearn.utils.shuffle(X_train, y_train)
```

The evaluate function perform accuracy operation within a tensorflow session and returns how well the model accuracy performs.

```
def evaluate(sess, generator, steps, accuracy_op, dict_opts={}):
    total_examples = 0
    total_accuracy = 0

    for step in range(0, steps):
        batch_x, batch_y = next(generator)
        accuracy = sess.run(accuracy_op, feed_dict={ **{ x: batch_x, y: batch_y },
**dict_opts})
        num_examples = len(batch_x)
        total_accuracy += (accuracy * num_examples)
        total_examples += num_examples
    return total_accuracy / total_examples
```

The test_model function evaluate a stored model on the test set and reports its accuracy.

```
def test_model(test_gen, test_steps, accuracy_op, saver, dict_opts={}, name_model
="."):
    """
    Test a saved model with test data
    """
    if not os.path.exists(name_model+".index"):
        print(name_model, "Model does not exists in disk")
        return False
    with tf.Session() as sess:
        saver.restore(sess, name_model)
        test_accuracy = evaluate(sess, test_gen, test_steps, accuracy_op, dict_opt
s=dict_opts )
        print("Test Accuracy = {:.3f}".format(test_accuracy))
```

The following function with name “training_validation” was created to perform training and validation of the model. The next paragraphs will explain every parameter that his function receive in order to make sense how the model is being trained.

Function “training_validation” parameters:

- name_model:
  The name of the model this will be used to store the model in disk and print results.

- train_gen:
  As explained above this is an initialized generator function with training data

- val_gen:
  As explained above this is an initialized generator function with validation data

- train_steps and val_steps:
  Number of batches which contains both training and validation sets

- train_op:
  tensorflow operation that minimize the loss function of the model.

- accuracy_op:

  tensorflow operation that calculate the accuracy of the model

- saver:

  Object that can be used to store or retrieve a model

- train_opts:

  Options that can be passed to the feed dictionary of tensor training operation

- dict_ops:

  Options that can be passed to the feed dictionary of tensor for validation or test operations

- old_train and old_model:

  To control whether yes or not the training should be on top of a pre trained model.

- store:

  To control whether yes or not store the final model

- epochs:

  Number of epochs the training will perform

The training_validation function store the model in every epoch such that the model can be retrained from one of the epochs results.

```
def training_validation(name_model, train_gen, val_gen, train_steps,
                        val_steps, train_op, accuracy_op, saver,
                        train_opts={}, dict_ops={}, old_train=False,
                        old_model='', store=False, epochs=3):
    """
    Train a model, store the model in disk and return the path
    """

    return_data = {
       'save_path': './'+name_model
```

```
    }

    with tf.Session() as sess:
        if old_train:
            if not os.path.exists('./'+old_model+".index"):
                print(old_model, "Model does not exists in disk")
                return return_data
            saver.restore(sess, './'+old_model)
        else:
            sess.run(tf.global_variables_initializer())

        print(name_model, "Training...")
        print()

        for i in range(epochs):
            for step in range(0, train_steps):
                batch_x, batch_y = next(train_gen)

                sess.run(train_op, feed_dict={**{x: batch_x, y: batch_y}, **train_
opts })

            train_accu = evaluate(sess, train_gen, train_steps, accuracy_op, dict_
opts=dict_ops)
            val_accu = evaluate(sess, val_gen, val_steps, accuracy_op, dict_opts=d
ict_ops)

            #Save trainning per epoch
            saver.save(sess, './'+name_model+str(i+1))
            print("EPOCH {} ...".format(i+1))
            print("Train Accuracy = {:.3f}".format(train_accu))
            print("Validation Accuracy = {:.3f}".format(val_accu))
            print()
```

```
        if store:
            saver.save(sess, return_data['save_path'])
            print(name_model, "Model saved")

    return return_data
```

The LeNet-5 architecture discussed in this project is implemented in the following function using tensorflow

```
def LeNet5(x, init_channels=1, fcx0_len=400):
    mu = 0
    sigma = 0.1

    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, init_channels, 6), mean
= mu, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + co
nv1_b

    conv1 = tf.nn.relu(conv1)

    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], paddin
g='VALID')

    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stdd
ev = sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID')
 + conv2_b

    conv2 = tf.nn.relu(conv2)

    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], paddin
g='VALID')
```

```
    fc0   = flatten(conv2)

    fc1_W = tf.Variable(tf.truncated_normal(shape=(fc0_len, 120), mean = mu, stdde
v = sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b
    fc1   = tf.nn.relu(fc1)

    fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stddev =
 sigma))
    fc2_b  = tf.Variable(tf.zeros(84))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b

    fc2    = tf.nn.relu(fc2)

    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 3), mean = mu, stddev = si
gma))
    fc3_b  = tf.Variable(tf.zeros(3))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

The hyper parameters were initialized as follow

```
BATCH_SIZE = 128
H_IMG, W_IMG = (32,32)
EPOCHS = 6
rate = 0.01
```

The next code shows how the data generators was initialized. Since this project contemplate that a training can star from older trained model the train, validation and test samples were stored in csv files and are retrieved from this files to be consistent with this data over all training process.

```
train_samples, validation_samples = train_test_split(data, test_size=0.4)
validation_samples, test_samples  = train_test_split(validation_samples, test_size
=0.3)

train_generator = generator(train_samples, batch_size=BATCH_SIZE)
validation_generator = generator(validation_samples, batch_size=BATCH_SIZE)
test_generator = generator(test_samples, batch_size=BATCH_SIZE)

train_steps      = math.ceil(len(train_samples)/BATCH_SIZE)
validation_steps = math.ceil(len(validation_samples)/BATCH_SIZE)
test_steps       = math.ceil(len(test_samples)/BATCH_SIZE)
```

The tensorflow operations that use LeNet-5 architecture are shown in the next code

```
x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 3)

logits = LeNet5(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=one_
hot_y)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate=rate)
training_operation = optimizer.minimize(loss_operation)
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

As discussed above the function training_validation receive the training_operation and accuracy_operation and the other discussed parameters to train the model

```
saver = tf.train.Saver()
lenet_5_data = training_validation('lenet5', train_generator, validation_generato
r,
```

```
                                    train_steps, validation_steps, training_operati
on,
                                    accuracy_operation, saver, train_opts={}, dict_
ops={},
                                    old_train=False, store=True, epochs=EPOCHS)
```

finally to test the model the next code was used

```
test_model(test_generator, test_steps, accuracy_operation, saver, dict_opts={}, na
me_model=lenet_5_data['save_path'])
```

Refinement

This section explains how the solution model base on LeNet-5 was built by applying modifications to the network in order to increase the performance of evaluation metrics that indicate better results than the benchmark model. Different experiments were performed with settings combinations specifically to the Convolutional Neural Networks, for example convolutions has filters experimenting by adding or removing filters, adding convolution layers, changing the patch size of the convolution etc. In addition experimentation with dropout layers and polling layers are considered in this section to improve model performance.

Experimentation using other color spaces for the images like RGB, HSV, YCrCb are considered as well as the normalization of the images to be zero centered.

After experimentation, the best model with best accuracy on training, validation and test sets will be chosen as finally solution model.

The first model solution was based on LeNet-5 described in Algorithms and Techniques section, table-5 shows the architecture and table-6 the results of this architecture.

Layer name	Size	Channels
Input	155x155	3
Convolution 1 (C1)	151x151	6
Sub-sampling	75x75	6

Layer name	Size	Channels
Convolution 2 (C2)	71x71	16
Sub-sampling	35x35	16
Fully connect (FC1)	120	-
Fully connect (FC2)	84	-
Output	3	-

Table-5 LeNet-5

The ConvNet outlined in table-9 was trained using the following parameters using different color spaces and obtaining the results that shows table-10

Parameters:

- EPOCHS = 10
- learning rate = 0.01

Color Space	Train	Validation	Test
RGB 3 channels	0.335	0.336	0.320
HSV 3 channels	0.766	0.514	0.523
YCrCb 3 channels	0.748	0.504	0.483

Table-6 Report first solution

As the table-6 shows the best model in the first solution is the one which use HSV Color Space. Since this first solution model does not show big improvements based on the benchmark model, the next steps of improvement were to decrease the learning rate, increase number of epochs and add neurons in fully connected layers FC1 from 120 to 1000 and FC2 from 84 to 250 neurons.

The table-7 shows the results of the model evaluated in the three color spaces. Model with YCrCb color space performs better that the other two however it shows sign of overfitting.

Parameters:

- EPOCHS = 10
- learning rate = 0.0008

Color Space	Train	Validation	Test
RGB 3 channels	0.997	0.641	0.619
HSV 3 channels	0.979	0.648	0.630
YCrCb 3 channels	0.985	0.646	0.623

Table-7 Report solution first improvement

The next improvement was to add dropout layers next to the fully connected layers FC1 and FC2 with value of (0.5) and increased number of epochs to 20. The results are shown in Table-8. This experiment results despite the dropout layers still suffers of overfitting.

Parameters:

- EPOCHS = 20
- learning rate = 0.0008

Color Space	Train	Validation	Test
RGB 3 channels	0.997	0.648	0.620
HSV 3 channels	0.999	0.660	0.663
YCrCb 3 channels	0.997	0.654	0.641

Table-8 Report solution second improvement

Other experiments that were performed are shown in table-9, the intention was to explore different channels of color spaces and some combinations looking for decrease the overfitting. The kernel sizes of the pooling layers will be increased. Only HSV and YCrCb color spaces are considering for this experiment Table-9 shows the results

Parameters:

- EPOCHS = 20
- learning rate = 0.0008

Color Space	Train	Validation	Test
HSV(Hue)	0.912	0.626	0.624
HSV(Saturation)	0.960	0.672	0.656
HSV(H+S)	0.954	0.676	0.658
HSV(S+B)	0.970	0.683	0.637
YCrCb(Y)	0.976	0.648	0.639
YCrCb(Cr)	0.925	0.670	0.630
YCrCb(Cb)	0.924	0.668	0.661
YCrCb(Y+Cr)	0.921	0.658	0.615
YCrCb(Cr+Cb)	0.946	0.686	0.665

Table-9

The results above indicate that the best model according with the validation and test scores is the one that use HSV color space with H+S channels. All the experimented models shows strong signs of overfitting despite the effort to reduce it, in order to intend an improvement on the selected model, five re-trainings keeping the learned weights of the previous trainings were performed using 20 epochs each however the results were not different and the overfitting keep the same.

This project considered to continue the evaluation of this model despite the overfitting due to time restriction. The model architecture obtained after perform the experiments is outlined in Table-10

Layer name	Size	Channels
Input	155x155	3
Convolution 1 (C1)	149x149	6
Sub-sampling	73x73	6
Convolution 2 (C2)	69x69	16

Layer name	Size	Channels
Sub-sampling	32x32	16
Fully connect (FC1)	1000	-
Dropout(0.5)	-	-
Fully connect (FC2)	250	-
Dropout(0.5)	-	-
Output	3	-

Table-10 Model architecture HSV (H+S)

Since the first solution based on LeNet 5 did not bring the expected results, another approach call transfer learning using ConvNet architecture AlexNet [7] was used as improvement. AlexNet is a pretrained neural network which contains 60 million parameters and 650,000 neurons.

The architecture of AlexNet is shown in Figure-8, it has five convolution layers, some max-pooling layers and three fully connected layers. This network was trained to classify 1.2 million images that can fall down into one of one thousand different categories.

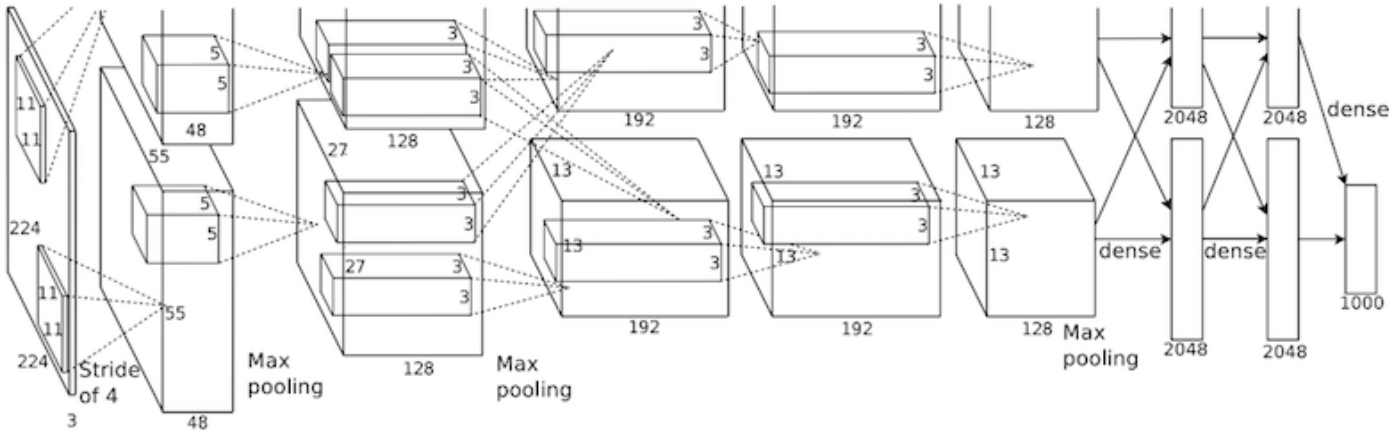


Figure-8 Source: AlexNet [7]

For this experiment The pretrained AlexNet model was used with its convolutions and two fully connected layers, then the fully connected layers and output layers from the previous model LetNet-5 were used here. The Table-11 shows the architecture of the new model.

Layer name	Size	Channels
Input	155x155	3
AlexNet Pretrained	4096	-
Fully connect (FC1)	1000	-
Dropout(0.5)	-	-
Fully connect (FC2)	250	-
Output	3	-

Table-11 AlexNet pretrained model

With the model outlined above several experiments were performed using the same color space as other experiments namely RGB, HSV and YCrCb AlexNet performs better with RGB, the parameters above were used to obtain results of Table-12

Parameters:

- EPOCHS = 100
- learning rate = 0.0008
- Color Space RGB

Metrics	Value
Train	0.959
Validation	0.697
Test	0.659

Table-12 Report solution AlexNet

IV. Results

Model Evaluation and Validation

After the improvements, the model that better performs according with the metrics was obtained using the following parameters.

- Image size (155) height, (155) width
- EPOCHS = 100
- learning rate = 0.0008
- Dropout = 0.5
- Color Space RGB
- AlexNet based architecture

In this section three models are evaluated namely the benchmark model, the intended solution model based on LetNet-5 (Table-10) and The model based on AlexNet architecture (Table-11). The evaluation will be using an unseen test set containing 512 images with the following distribution: type1 87, type2 265, type3 160.

Six images for each category were selected randomly from the unseen test set reporting this probabilities in the following tables for each model. The figures below shows these images.

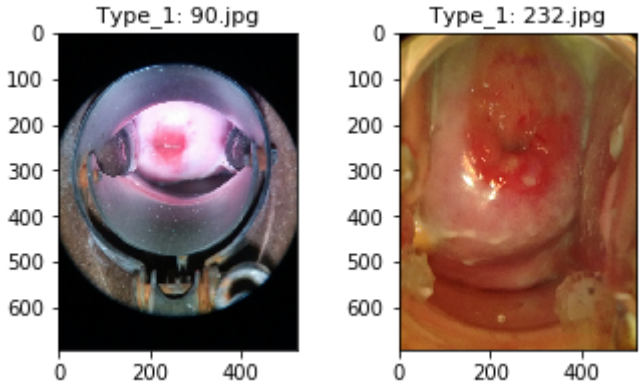


Figure-9 Test Type 1

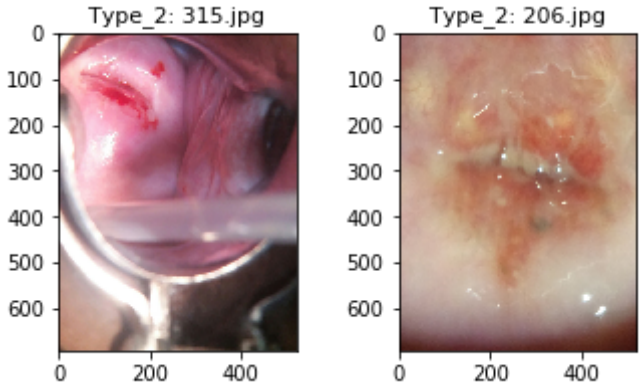


Figure-10 Test Type 2

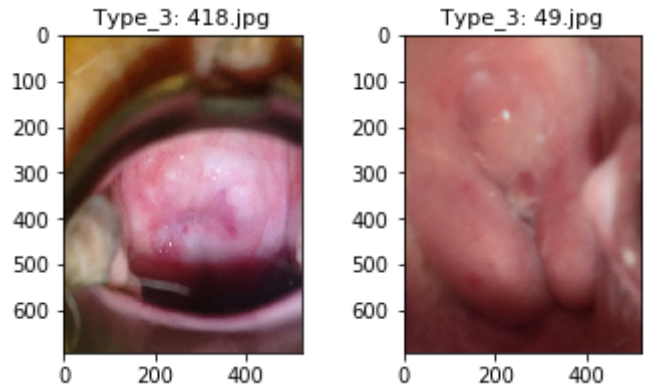


Figure-11 Test Type 3

Unseen test images	Type 1	Type 2	Type 3
90.jpg - Type 1	0.001	0.859	0.140
232.jpg - Type 1	0.181	0.436	0.383
315.jpg - Type 2	0.393	0.326	0.281
206.jpg - Type 2	0.285	0.426	0.289
418.jpg - Type 3	0.361	0.005	0.633
49.jpg - Type 3	0.175	0.423	0.402

Table-13 Test data probabilities - benchmark model

Unseen test images	Type 1	Type 2	Type 3
90.jpg - Type 1	0.01	0.084	0.906
232.jpg - Type 1	0.162	0.587	0.250
315.jpg - Type 2	0.000	0.999	0.000
206.jpg - Type 2	0.002	0.980	0.019
418.jpg - Type 3	0.000	0.996	0.004
49.jpg - Type 3	0.000	0.994	0.006

Table-14 Test data probabilities - LetNet model

Unseen test images	Type 1	Type 2	Type 3

Unseen test images	Type 1	Type 2	Type 3
90.jpg - Type 1	0.000	0.001	0.999
232.jpg - Type 1	0.256	0.551	0.194
315.jpg - Type 2	0.000	0.993	0.007
206.jpg - Type 2	0.258	0.529	0.213
418.jpg - Type 3	0.002	0.984	0.014
49.jpg - Type 3	0.001	0.07	0.929

Table-14 Test data probabilities - AlexNet model

Additionally to the six test images perturbation were added using the augmentation tool[6] with *noise_0.06* obtaining this results on evaluation. The figures below shows these images:

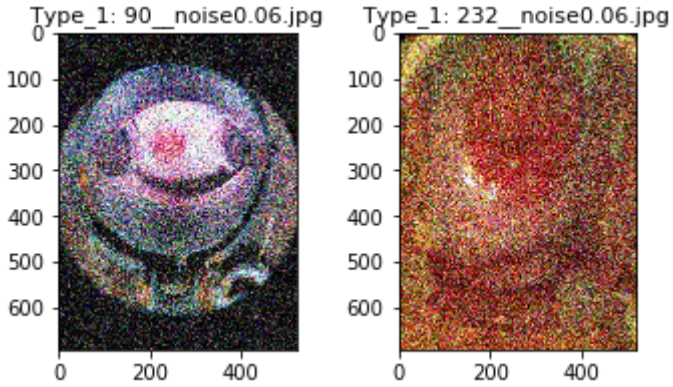


Figure-12 Noise Type 1

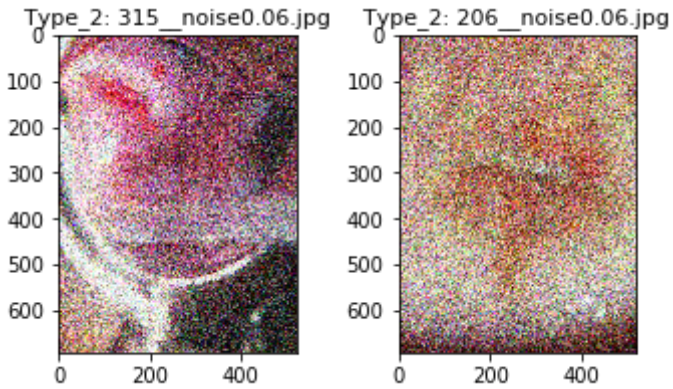


Figure-13 Noise Type 2

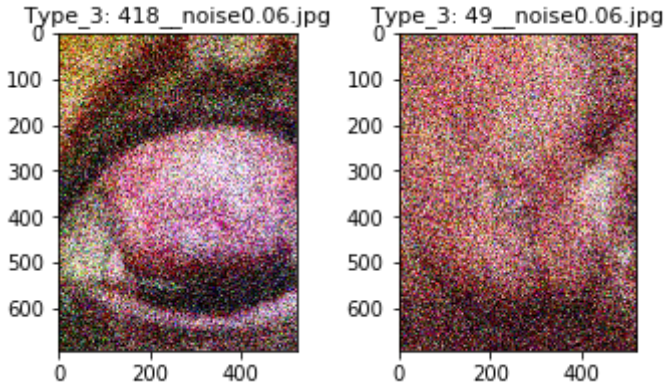


Figure-14 Noise Type 3

Unseen test images	Type 1	Type 2	Type 3
90__noise0.06.jpg - Type 1	0.002	0.908	0.09
232__noise0.06.jpg - Type 1	0.595	0.216	0.19
315__noise0.06.jpg - Type 2	0.513	0.432	0.056
206__noise0.06.jpg - Type 2	0.299	0.427	0.273
418__noise0.06.jpg - Type 3	0.332	0.015	0.653
49__noise0.06.jpg - Type 3	0.163	0.432	0.405

Table-15 Test data probabilities - benchmark model

Unseen test images	Type 1	Type 2	Type 3
90__noise0.06.jpg - Type 1	1.00	0.00	0.00
232__noise0.06.jpg - Type 1	1.00	0.00	0.00
315__noise0.06.jpg - Type 2	1.00	0.00	0.00
206__noise0.06.jpg - Type 2	1.00	0.00	0.00
418__noise0.06.jpg - Type 3	1.00	0.00	0.00
49__noise0.06.jpg - Type 3	1.00	0.00	0.00

Table-16 Test data probabilities - LeNet model

Unseen test images	Type 1	Type 2	Type 3

Unseen test images	Type 1	Type 2	Type 3
90__noise0.06.jpg - Type 1	0.021	0.316	0.663
232__noise0.06.jpg - Type 1	1.00	0.00	0.00
315__noise0.06.jpg - Type 2	1.00	0.00	0.00
206__noise0.06.jpg - Type 2	1.00	0.00	0.00
418__noise0.06.jpg - Type 3	1.00	0.00	0.00
49__noise0.06.jpg - Type 3	1.00	0.00	0.00

Table-17 Test data probabilities - AlexNet model

The evaluation of all of the three models on unseen tests set are shown in Table-19

Model	Accuracy
Benchmark	50.59
LeNet	60.55
AlexNet	58.40

Table-19 Results unseen data

The table above shows the results of the three models, LetNet and AlexNet outperform the benchmark model but is not enough to have a robust model that can be roll out to production.

Justification

The final model solution outperforms the initial benchmark model results. The best validation and test accuracies for benchmark were 0.514 and 0.483. Instead the final model outperform this accuracies with this values 0.676 and 0.658.

However the finally model has a high accuracy in training data about 0.954 which make this model to suffer overfitting. since it has 0.658 of accuracy on test set. For a production use of this solution and since this is critical

health problem this model is weak and will need to perform further improvements that will be mention in this document.

V. Conclusion

Free-Form Visualization

This section is intended to understand how the ConvNet classify the cervix type, for this section intermediary features of the network are plotted to understand what this neural networks sees with different cervix types. Each of the images below are the result of the convolution layer 1 and convolution layer 2 of the network described in Table-10.

The first row of the image are the six filters or channels of the first convolution and the other rows are the sixteen filters or channels of the second convolution. This filters represent the automated extracted features that the network comes up after the training for the given images.

Recalling the results of Table-14 the Figure-15 correspond to the features of image *90.jpg* of Type 1 it was predicted as type 3. Figure-16 features of image *315.jpg* of Type 2 it was predicted as type 2. The last Figure-17 features of image *418.jpg* of type 3 get predicted as type 2.

The network seems to be misclassifying image of type 3 as type 2 and it could be due to the instrument used to perform the screening of the cervix, looks like the first rows of this images are kind of similar.

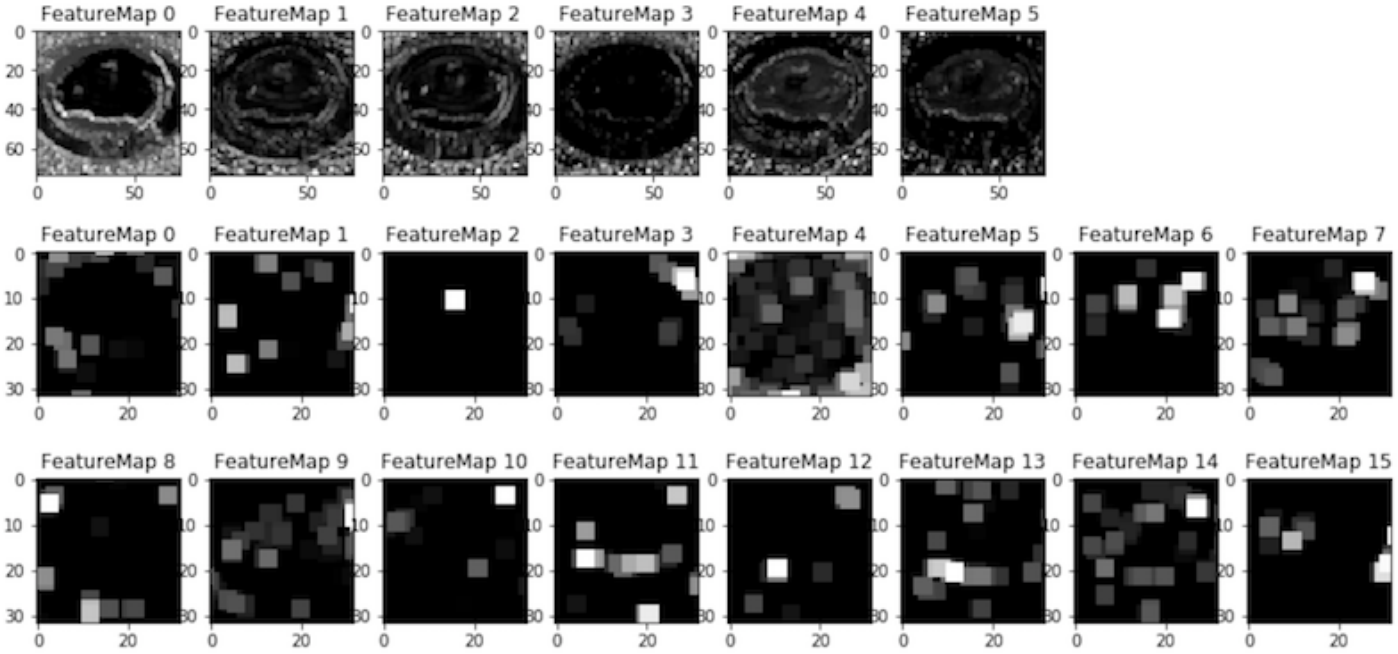


Figure-15 Feature Map 90.jpg Type_1

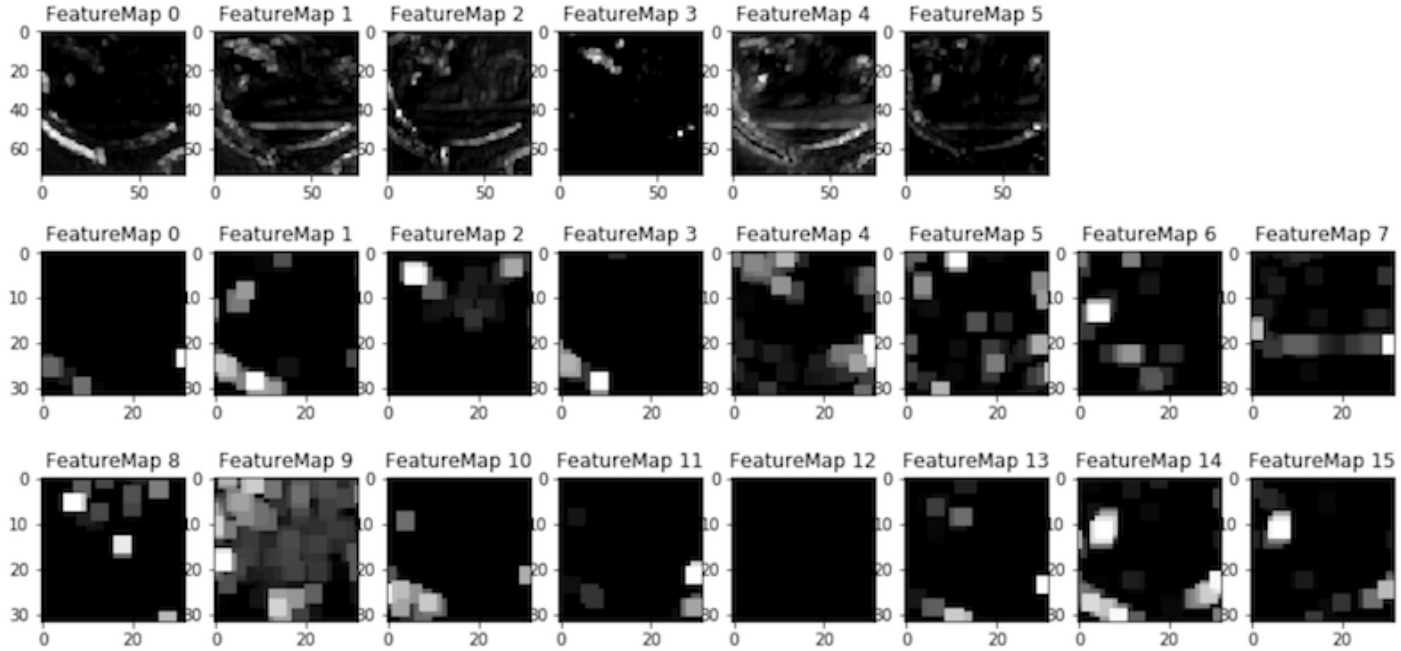


Figure-16 Feature Map 315.jpg Type_2

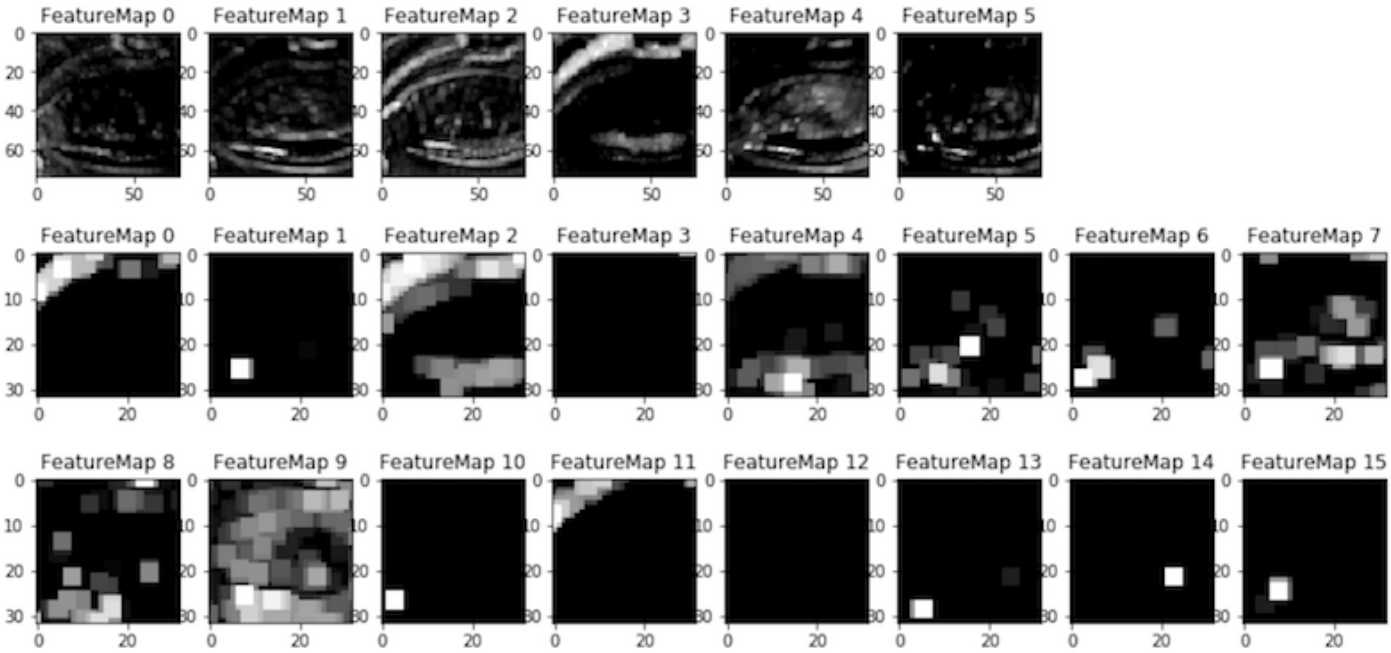


Figure-17 Feature Map 418.jpg Type_3

The network finds difficult to discriminate cervix type for the given images, in the improvement section will be mention another ideas that can lead to a better model that could improve the results in general.

Reflection

Reaching a final solution was a process that take long time of experimentations with different settings trying to overcome the overfitting of data.

The process can be summarized in the following steps:

- Data Exploration

  The data was downloaded from kaggle, uncompressed and analyzed, I found that was necessary to normalize the images sizes to be consistent with the input of the model that will process it. during the process different color space were explored and then applied to the model.

  The images classes were imbalanced and was no easy to apply a proper data augmentation process like rotating images or modifying the lightness or contrast, for that reason a mere duplication was applied to the data to have roughly a equal number of classes.

In the exploration imaging tools like ImageMagick were applied to the image files to detect corrupt data and were found few samples

- Data Modeling

  At the beginning was clear that applying Convolutional Neural Network to images classification problem is probably a good fit to find out a good solution. This project lied out on LeNet-5 Architecture a well known Convolutional Neural Network that has gotten good results of different images classifications like digit classifier and traffic sign predictions among others.

- Model Experimentation

  This process took more time than others since different settings and combination of parameters were performed to pursue the desired results.

  Different assumptions were made in this step like include different color spaces in the experiments to see if there are some channels that can better be consumed by the ConvNet and get betters results.

  Changing the architecture of the ConvNet and a adjusting hyper parameters with the aim of better accuracies were performed in this step.

- Model Refinement

  This is an extension of model experimentation that were performed by revisiting the results of the experimentations looking for hints that lead to extra parameters or architecture tuning.

- Evaluation

  The evaluation was performed through the experimentation process after every training the test set were used to evaluate the model.

The final solution despite all of the efforts to reach a very good model that does not overfit the test set was not possible.

Improvement

There are several improvements that were considered in this project but were not explored due to scope of time or missing knowledge to integrate on this solution.

- More image preprocessing

  Considering this data set are images from cervix a improvement would be to have some preprocessing techniques to get region of interest over this images and discard the noise elements present on the images like metal tools.

- Applying more complex ConvNet Architectures

  During the process I've tried to use pre-trained AlexNet architecture, having almost near results to the LetNet model, however trying even more complex architectures like VGG-16 or ResNet could bring better results.

- Getting more data

  One aspect of this project is that data is not enough to build a robust model, getting more data analyzing with more complex methods like applying clustering or mixtures models to get more insights of data variety will be a very good option to explore. However getting data about cervix screen was found difficult.

I'm sure having more time to study and apply the improvements above can lead to a better and robust solution.

References

1. OMICS International, "Artificial Intelligence Based Semi-automated Screening of Cervical Cancer Using a Primary Training Database", https://www.omicsonline.org/open-access/novel-benchmark-database-of-digitized-and-calibrated-cervical-cells-forartificial-intelligence-based-screening-of-cervical-cancer-ccoa-1000105.php?aid=68453

2. The Scientific World Journal, "Intelligent Screening Systems for Cervical Cancer", http://dx.doi.org/10.1155/2014/810368

3. Kaggle, “Intel & MobileODT Cervical Cancer Screening”, https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening

4. Yann Lecun, “LeNet-5, convolutional neural networks”, http://yann.lecun.com/exdb/lenet/

5. Data set used in the project, https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening/data

6. Data Augmentation Tool, https://github.com/codebox/image_augmentor

7. AlexNet architecture, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf