# Analysis of Machine Learning Algorithms For Histopathological Cancer Detection

**Arpan Vitthalbhai Patel**
apatel58@asu.edu

**Darshan Dhirubhai Malaviya**
dmalaviy@asu.edu

**Harshal Mukeshkumar Trivedi**
htrived2@asu.edu

**Ishani Jigneshkumar Bhatt**
ibhatt1@asu.edu

**Jaykumar Keshavbhai Vaghasiya**
jvaghasi@asu.edu

**Shreerang Hegde**
srhegde1@asu.edu

**Venkateswara Prasad Kaikala**
vkaikala@asu.edu

**Vishal Kumar**
vkumar47@asu.edu

## Abstract

One of the many great things about Machine learning research is that due to its inherent general nature, its probable applications is very extensive. One of the possible directions in which we can invest is Medicine. Being able to program the detection of metastasized cancer in pathological images with machine learning is an area of medical imaging and diagnostics with promising potential for clinical utility[22]. Early diagnosis can increase the chance of successful treatment and survival. However, it is a very challenging and time-consuming task that relies on the experience of pathologists. The automatic diagnosis of cancer by analyzing histopathological images plays a substantial role for patients and their prognosis. Moreover, the ability of ML to perceive key features from complex datasets reveals their significance. So, we developed ML algorithms to identify metastatic cancer in small image patches taken from larger digital pathology scans and conducted a comparative study between traditional machine learning technique and Deep Learning.

## 1 Introduction

Over past few years, a consistent evolution regarding to cancer research has been achieved [18]. Scientists applied different methods, such as screening in early stage, in order to find types of cancer before they cause symptoms. Moreover, they have developed new strategies for the early prediction of cancer treatment result. With the arrival of new technologies in the field of medicine, large amounts of cancer data have been collected and are available to the medical research community. However, the accurate prediction of a disease outcome is one of the most interesting and challenging tasks for physicians. As a result, ML methods have become a popular tool for medical researchers. These techniques can discover and identify patterns and relationships between them, from complex data-sets, while they are able to effectively predict future outcomes of a cancer type. Application of Machine learning algorithms to whole-slide pathology images can potentially improve diagnostic accuracy and efficiency[22].

While achieving a decent classification is possible without domain knowledge, it's always valuable to have some basic understanding of the domain. However, if we decide to strive for a state-of-the-art

performance, we should consider using the domain knowledge and applying heuristics to create a model that's well-fitting to the problem we are trying to solve.

### 1.1 Domain Knowledge

Metastatic Cancer: Metastasis is the spread of cancer cells to new areas of the body (often by way of the lymph system or bloodstream)[22]. A metastatic cancer, or metastatic tumor, is one which has spread from the primary site of origin (where it started) into different area(s) of the body. lymph node metastases can have the following features:

- Foreign cell population: key feature
- Cells with cytologic features of malignancy
- Cells in architectural arrangements seen in malignancy[22] highly variable - dependent on tumor type and differentiation.

### 1.2 Transfer Learning

Transfer learning (Pan and Yang, 2010)[12] emerges from deep learning. It is well-known that it is typically impossible to train a complex deep network from scratch with only a small dataset. Furthermore, there are not any existing principles to design a network structure for a specific task. What we can do is adopt the model and the parameters obtained by other researchers via time-consuming and computationally intensive training on the very large image dataset of ImageNet[22] and use the knowledge it has gained as pre-training for our specific research task. Then, we can retrain the last defined fully-connected layer of the model using only a relatively small amount of data to achieve good results for our target task. Given the significance of personalized medicine and the growing trend on the application of ML techniques, we here present our work that make use of SVM and deep learning methods like CNN, VGG-16, VGG-19, Xception, ResNET and DenseNET for cancer detection. In addition, we do a complete analysis of the types of ML methods being used and compare the overall performance of each proposed scheme while we also discuss their Merits and Demerits.

## 2 Related Work

Breast cancer diagnosis based on image analysis has been studied for more than 40 years, and there have been several notable research achievements in the area. These studies can be divided into two categories according to their methods: one is based on traditional machine learning methods, and the other is based on deep learning methods. The former category is mainly focused on small datasets of breast cancer images and is based on labor intensive and comparatively low-performing, abstract features. The latter category can deal with big data and can also extract much more abstract features from data automatically. Although traditional machine learning methods have made great achievements in analyzing histopathological images of breast cancer and even in dealing with relatively large datasets, their performance is heavily dependent on the choice of data representation (or features) for the task they are trained to perform. Furthermore, they are unable to extract and organize discriminative information from data (Bengio et al., 2013)[8]. Deep learning methods typically are neural network-based learning machines with much more layers than the usual neural network. They have been widely used in the medical field since they can automatically yield more abstract—and ultimately more useful—representations (Bengio et al., 2013)[9] That is, they can extract the discriminative information or features from data without requiring the manual design of features by a domain expert (Spanhol et al., 2016b)[10]. One common method for performing transfer learning (Pan and Yang, 2010)[11] involves obtaining the basic parameters for training a deep learning model by pre-training on large data sets, such as ImageNet, and then using the data set of the new target task to retrain the last fully-connected layer of the model. This process can achieve good results even on small data sets.

## 3 Methods

**Dataset (PCam)**: The PatchCamelyon benchmark is a new and challenging image classification dataset. It consists of 220,025 color images (96px x 96px) extracted from histopathologic scans of

lymph node sections. Each image is annoted with a binary label indicating the presence of metastatic tissue[20]. Without cancer: 130908 and With Cancer 89117. This is an unbalanced dataset.so we have considered only 160000 to transform into the balanced dataset. Then the dataset is divided into a training set, a validation and test set. There is no overlap in WSIs between the splits, and all splits have a 50/50 balance between positive and negative examples.PCam offers a new benchmark for machine learning models: larger than CIFAR10, smaller than imagenet, trainable on a single GPU[21].

### 3.1 Support Vector Machines (SVM)

Support vector machines (SVMs) is a supervised machine learning algorithm which can be used for both classification and regression, but it is mainly used in classification. SVM was first heard in 1992, introduced by Boser, Guyon, and Vapnik in COLT-92, and became famous when, using pixel maps as input; it gives accuracy comparable to sophisticated neural networks with elaborated features in a handwriting recognition task [4].

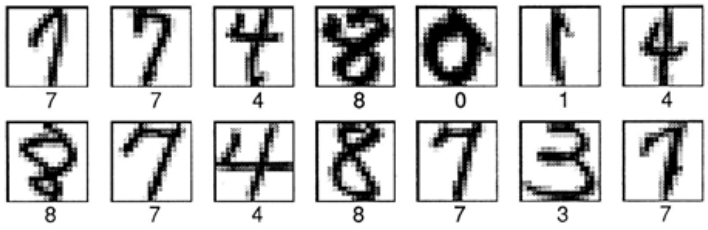


Figure 1: Example of images used in the above mentioned task[5]

SVMs classify data by finding the optimal hyperplane in the given space such that points in the space are as far as possible from the found hyperplane i.e. the hyperplane which has maximum margin, the hyperplane of a n-dimensional space is given by n-1 dimensions. For complex problems where

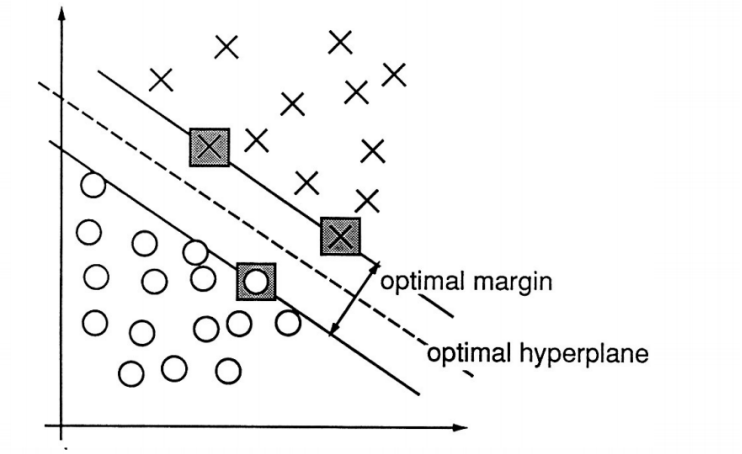


Figure 2: An optimal hyper-plane in a 2D-space[5]

given data is not linearly separable in given space, Kernel techniques are used. Kernel techniques basically transform the given space into some other space where the datapoints maybe are separable by a hyperplane.

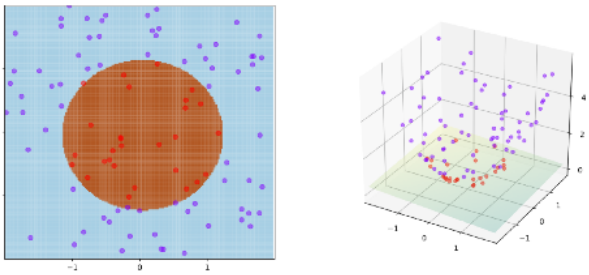

Figure 3: Use of Kernel Technique to separate data-point.[6]

Our problem data-set is the 'Histopathological Cancer Detection' data-set which only has two classes, this makes our problem a binary classification problem and thus easier for an SVM to work. Other than this, the final objective function in SVM is a convex function so only one optimal solution is present, while in case of neural networks it has been observed that there can be a local optimum also. Also, SVM scales well to high dimensional data, which gave us the freedom to choose from a wide range of feature space containing features from a range of 20, 30 to few thousand features. And even with each feature space we can make different possible spaces using kernel technique, which is the real strength of SVM.

### 3.1.1 SVM Implementation

For our experiment we implemented SVM from scratch, we implemented two different approaches:

- Dual SVM formulation for non-separable case
  Our first implementation required resolving the quadratic equation in the 'Dual SVM formulation for non-separable case' solving that equation gave us the values of Lagrange multipliers with which we calculated our w and b[7]. The required Quadratic equation was solved using 'CVXOPT library'[8]. We kept the implementation related to soft-margin SVM for flexibility of over/under-training.
  During the training, the following equation is solved:

$$max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(X_i X_j) \tag{1}$$

  Constraints:

$$K(X_i, X_j) = \phi(X_i).\phi(X_j) \tag{2}$$

$$\sum_i \alpha_i y_i = 0 and C \geq \alpha_i \geq 0 \tag{3}$$

  Using the result, w and b were calculated as:

$$w = \sum_i \alpha_i y_i \phi(X_i) \tag{4}$$

$$b = y_k - \sum_i \alpha_i y_i K(X_i, X_k) \tag{5}$$

  for k where $c > \alpha_k > 0$

  Results were calculated as:

$$sign(w.\phi(X) + b) \tag{6}$$

  In the above equations K represents our kernel function, $\phi(X_i)$ represents the feature vector $X_i$ in the space of the kernel function.

- Gradient Descent SVM
  Support Vector Machine is a supervised machine learning algorithm which works best for linearly separable data. The separating margin for SVM can be represented by:

$$y = w.x + b \tag{7}$$

The objective of the SVM is to maximize the margin that separates the 2 classes of samples i.e. positive and negative. The constraint that needs to be kept into consideration while maximizing the margin is:

$$y(w.x + b) \geq 1 \tag{8}$$

However, this constraint only works when the data is completely linearly separable. The data that we are working with, has overlapping distribution. So the constraint that we need to take care of is as follows:

$$y(w.x + b) \geq 1 - \xi \tag{9}$$

Therefore, our optimization problem becomes[11]:

$$argmin\bigg(C\sum_{n=1}^{N}\xi_n + 12||w^2||\bigg) \tag{10}$$

subject to the constraint:

$$y_i(w.x_i + b) \geq 1 - \xi_i \tag{11}$$

which can be written as:

$$y_i.f(x_i) \geq 1 - \xi_i \tag{12}$$

This constraint when combined with $\xi_i \geq 0$ we get,

$$\xi_i = max(0, 1 - y_i.f(x_i)) \tag{13}$$

So when $y_i.f(x_i) \geq 1$ the sample is correctly classified and on the correct side of the margin and $\xi = 0$.
When $0 \leq y_i.f(x_i) \leq 1$ the sample is correctly classified and between the decision boundary and margin and hence will contribute to the loss because $0 < \xi < 1$.
When $y_i.f(x_i) \leq 0$ the sample is incorrectly classified and will contribute to the loss and $\xi > 1$.

So finally, our optimization function or the cost function becomes:

$$argmin\bigg(C\sum_{i}^{N}max(0, 1 - y_i.f(x_i)) + 12||w^2||\bigg) \tag{14}$$

In layman terms,

$$argmin(lossfunction + regularizationterm) \tag{15}$$

Since the loss function is a convex function, the local minimum will also be the global minimum.
For mathematical convenience we can write the cost function as:

$$argmin\bigg(\sum_{i}^{N}\lambda 2||w^2|| + max(0, 1 - y_i.f(x_i))\bigg) \tag{16}$$

where $\lambda = 1C$

So according to the gradient descent update rule, we have:

$$w^{t+1} = w^t - \eta \times \partial(costfunction)\partial w^t \tag{17}$$

Therefore, the update step will involve 2 cases:

1. When the sample is correctly classified and on the incorrect side of the margin or if the sample is incorrectly classified, the update will be:

$$w^{t+1} = w^t - \eta(\lambda w^t - y_i.x_i) \tag{18}$$

2. When the sample is correctly classified and on the correct side of the margin the update will be:

$$w^{t+1} = w^t - \eta\lambda w^t \tag{19}$$

### 3.2 Convolution Neural Networks (CNN)

Convolution Neural Network or CNN is a type of deep learning neural networks. This class of neural networks have a broad usage in the field of image classification. These networks are inspired by the structure of neural networks inside nervous system of humans. CNNs require very little amount of pre-processing when compared to other image classification algorithms [28].

The architecture of basic CNNs can be seen from the Figure 4:

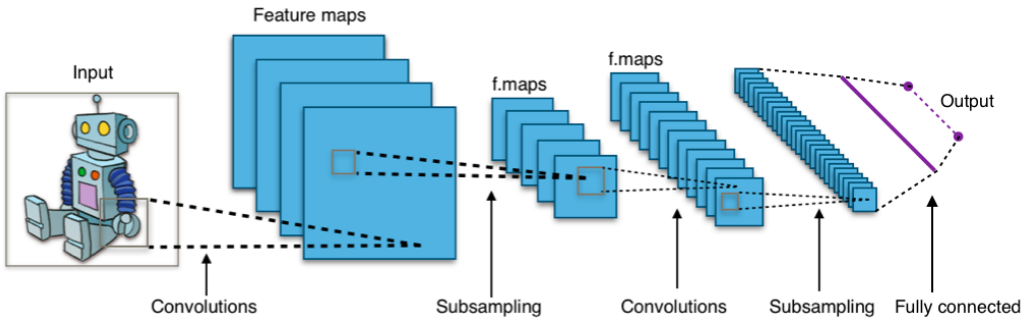


Figure 4: CNN Architecture

A typical CNN architecture consists of an input and output layer along with many hidden layers. The hidden layers typically are the Convolutional layers, Pooling layers, Fully Connected layers and normalization layers. Convolution layer is the basic unit of a CNN. The parameters of this layer are a collection of filters that learn by themselves, i.e., learnable filters, which have a small receptive field, but extend through the full depth of the input volume. Pooling layer has a function of "pooling", which is a form of non-linear "down-sampling". Pooling in a CNN replaces its input with an output at a location with a summary statistics of nearby outputs. Max pooling, Average pooling, L2 norm are some of the pooling methods that are implemented in pooling layers of a CNN. Fully Connected layers have the function of doing high-level reasoning in the neural network. Neurons or nodes in this layer have connections to all activations in the previous layer, as goes by their name. Normalization layer, is the final layer after which the output of the CNN is provided and this layer is based on what the CNN is being used for, for example, if the CNN is used for binary image classification, the output parameter of this layer will have value 2 showing that the output labels are only 2, along with the activation function, like "softmax", used in the layer[28].

CNNs are widely used in the field of image recognition. CNNs require very less amount of pre-processing when compared to other machine learning algorithms that are used for image classification problems. Also, as CNNs replicate the structure of neurons inside a Human brain, they are able to extract features and learn from them in a very similar approach to that of a human brain and so they learn pretty fast. Also, the "weight sharing" feature of CNNs reduces the number of computations compared to those required by many machine learning algorithms and so they require comparatively less computation power. These advantages and many more led to great success of CNNs. In 2012 an error rate of 0.23 percent on the widely used "MNIST database" was reported [29]. Also, it was observed that the performance of CNNs on the ImageNet dataset, which was the dataset of ImageNet Large Scale Visual Recognition Challenge was close to that of humans [30]. Such accurate results

obtained from the CNNs, made them a very successful algorithm in the field of image classification and that was the chief reason why we used CNN is our analysis.

### 3.3 VGG 16 and VGG 19

VGG is a convolutional neural network which is a type of transfer learning technique having deeper networks and smaller filters. The model consists of the convolutional layer with pooling layer, the output of which is then given to the fully connected layer. The last layer consists of Softmax activation function, which is used for the classification. Here to take advantage of the weights of the Imagenet, we froze several layers of the VGGNet while training it. Moreover, in the fully connected layer also we used dropout to avoid the issue of overfitting.

The architecture of VGG-16 and VGG-19 is shown in the Figure 5.

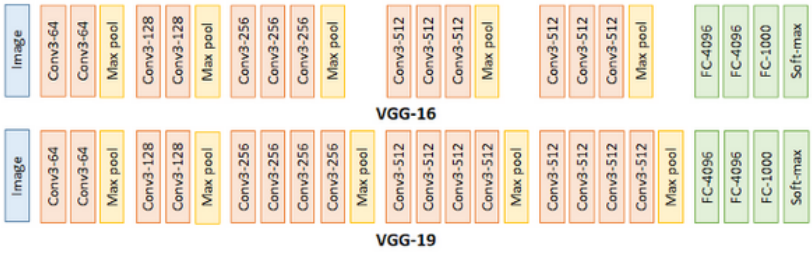


Figure 5: VGG 16 and VGG 19 Architecture

As we can see from the figure that the difference between the architecture of VGG-16 and VGG-19 is that after two blocks of convolutional layers, VGG-16 has 3 convolutional layers whereas VGG-19 has 4 convolutional layers followed by a max pool layer. Hence, VGG-19 is a deeper network as compared to VGG-16.

The reason for using VGG is to utilize the benefit of transfer learning. In real world machine learning problem we don't always have millions of training data with label. We also don't have enough computing power and time to train large model. Large models sometimes take weeks to train properly. The other problem is the number of training data. In deep learning models usually we have to train lots of parameter, sometimes even in millions. Lack of enough data is the biggest limitation of deep learning techniques. By using transfer learning technique we can overcome most of these constraints.

### 3.4 Residual Networks (ResNet-50)

ResNet is a type of deep neural network which was introduced to overcome the limitations of sequential models[4]. Generally, ResNet consists of several subsequent residual modules which are introduced for avoiding the problem of vanishing gradients. It does so by introducing the concept of skipping/shortcut connections. Skipping is, stacking convolutional layers one over another along with adding original input to the output of each convolutional layer. This allows to use fewer layers at initial stages of training, hence, simplifying the network. Skipping allows the model to learn an identity function which allows us to ensure that deeper layer performs at least as good as the shallower layer and not worse[16].

The Figure 6 is a single ResNet block which shows the shortcut connection.

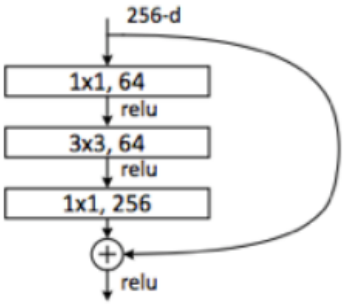


Figure 6: ResNet50 Architecture

The main reason for using ResNet is based on the benefits of using the residual modules. The residual modules makes it possible to effectively develop deeper networks without making it too complex which also helps to accomplish improved performance of the model. Furthermore, it also makes it easy to optimize the deeper networks on larger parameter space.

### 3.5 DenseNet

DenseNet is an extended version of ResNet and an effort to keep increasing the depth of deep CNNs. The DenseNets solves the problem of information loss ensuring maximum information and gradient flow from input to output by connecting each layer directly with the others. DenseNet merges the features by concatenation. The Lth layer has L inputs, which consists of the feature-maps of all convolutional layers. This gives L(L+1)/ 2 connections in an L-layer network, instead of just L in classical networks. Because of this dense connection pattern, it is called Dense Convolutional Network (DenseNet)[8]. Figure 7 describes the layout of DenseNet.

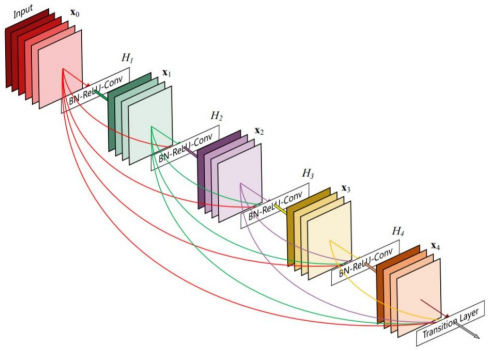


Figure 7: DenseNet Architecture

The reason for using DenseNet is that it can exploit the potential of the network through feature reuse and because of this DenseNets require less parameters than a classical CNN [8], as it is not necessary to learn redundant feature maps. Moreover, DenseNet layers are thin and they just add a small set of new feature-maps. Another problem with increasing the depth in the networks is training, because of the flow of information and gradients. DenseNets solve this problem as each layer has direct access to the gradients from the loss function and the original input image [9].

### 3.6 Xception

Xception is an extreme version of Inception net. It has a modified depth wise separable convolution. Originally the depth-wise separable convolution is the depth-wise convolution followed by a

point-wise convolution. Whereas, the modified depth-wise separable convolution is the point-wise convolution followed by a depth-wise convolution. The network is 71 layers deep and can classify images into 1000 object categories. By default, the image input size 299 by 299. The Xception architecture has the same number of parameters as Inception. So, the performance gains aren't due to the increase in capacity rather it is due to efficient use of model parameters.

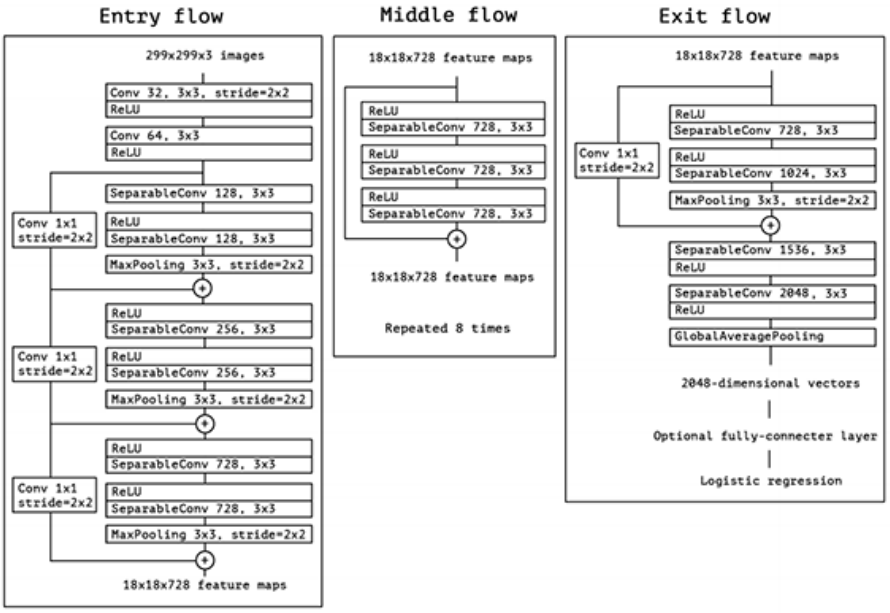


Figure 8: Xception Architecture[20]

Here in the modified depth-wise separable convolution a 1x1 convolution is performed first and then channel-wise spatial convolution. There is no intermediate ReLU non-linearity whereas in the original module there is a non-linearity after first operation. In the Entry flow, the architecture consists of 3 convolution layers each with stride of 2x2. Each of these convolutions also performs two ReLUs and one Max Pooling. In the Middle flow, three ReLU operations are performed eight times. Finally, in the Exit flow, there is one convolution layer followed by separable conv and global average pooling.

Xception net beat the almost similar Inception V3 which was the 1st runner up in ILSVRC 2015. It has already more than 300 citations. Xception without any intermediate activation has the highest accuracy compared with the ones using either ELU or ReLU. It is better than Inception-V3 for both ImageNet, ILSVRC and JFT datasets. Xception net outperforms VGGNet, ResNet, and Inception-V3 when it was tested by using ImageNet dataset which consists of over 15 million labelled high-resolution images with around 22000 categories.

## 4 Experiments and Results

Here we are listing different experiments that we conducted and the results that we obtained from them and their analysis:

### 4.1 SVM

#### 4.1.1 Feature extraction

The data set that we are working with has raw images. In order to classify the images using SVM we need to extract features from these images. The features that are used for learning in the SVM are very important because we need to identify the key elements of an image in order to correctly classify it. Another utility of feature extraction is that it increases the efficiency of the algorithm by providing

only the required information and removing the extraneous or unimportant information from the samples.
When it comes to feature extraction from raw images, there are numerous algorithms that can be used. However, it may so happen that the features extracted using a particular algorithm may not yield desired results.
For our data set we are primarily making use of Flattened image, KAZE descriptor and Hu Moments. All the images are converted to gray-scale before they undergo feature extraction. Lets take a look at the description of the algorithms:

1. Flatten: The simplest and most intuitive of techniques is flatten which converts the 2D grayscale image to 1D vector each element of which can be treated as a feature.
2. KAZE: The main objective of the KAZE algorithm is to detect features in a non linear scale space using anistropic or non linear diffusion[13]. This results in blurring of the image, however, the edge lines and boundaries remain intact thus allowing important features to be extracted[12].
3. Hu Moments: The main objective of Hu moments is to successfully conduct a translation, rotation and scale invariant shape matching for images. This is done by calculating 7 values or features using central moments which will be invariant to image transformations[15].

#### 4.1.2 Dual SVM formulation for non-separable case:

Experiments:

We converted raw images into feature vectors and then trained our model for that feature space over different combination of data-size, c-value, kernel function.

The additional feature space that we explored for images are:

- AKAZE (Zero features extracted)
- ORB (Zero features extracted)

We settled with features extracted using 'Hu moments' as it was giving us the maximum accuracy among all feature spaces that we tested. We were certain that using kernel technique over that feature space may gave us an even better space for training.

Experiments using the above defined implementation:

We tried around 560 different combination of Data-Size, c-value(regularization term) and kernel function depending on the following values:

- Data-Size: Due to the absence of 'CVXOPT' library in Agave system at Arizona State University, we had to train the model on small detest in our own computer. The data-set size ranges from 1000-7000 samples (7 cases). The test-set had a size of 500 data-samples which were not included in the training set.
- Kernel Function: Gaussian kernel with sigma = 5.0, Linear kernel, Polynomial kernel (degree 2), Polynomial kernel (degree 3).
- C-value: Range from 0-1 with an increment on 0.05 (20 different values).

Results:

The following observations were made from our experiments, the inferences from these results will be discussed in the later section.

1. This first set of graphs is showing the variation in accuracy with respect to the data-size used for a specific Kernel Function and C value. This C value is the one for which using the mentioned kernel we got the maximum accuracy irrespective of the data-size.

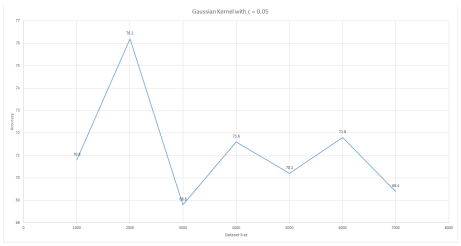


Figure 9: Gaussian kernel

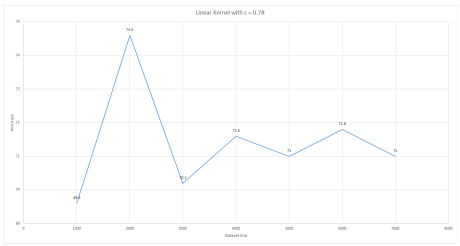


Figure 10: Linear Kernel

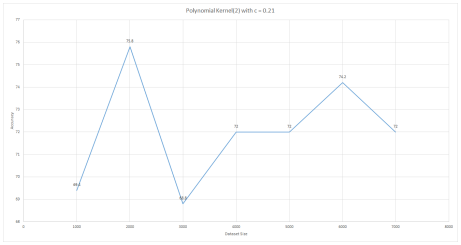


Figure 11: Polynomial Kernel (degree 2)

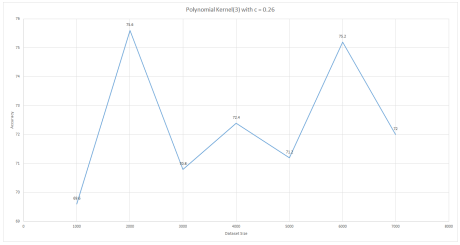


Figure 12: Polynomial Kernel (degree 3)

From the above set of graphs it was observed that every kernel was giving its best accuracy at a data-set of size 2000.

This second set of graphs is showing the variation of accuracy with c-value (Regularization term) with respect to the mentioned Kernel Function and the data-set size of 2000. For this plot we choose data-set size to b 2000 as it was giving the highest accuracy for every kernel.

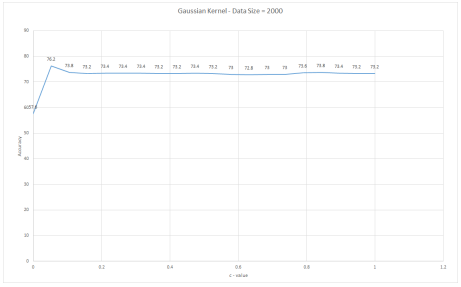


Figure 13: Gaussian kernel

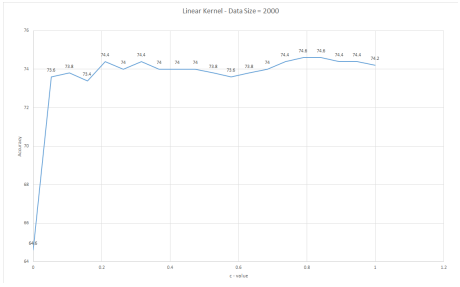


Figure 14: Linear Kernel

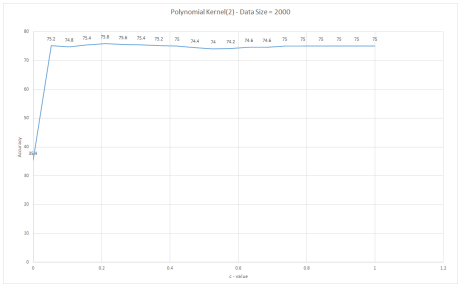


Figure 15: Polynomial Kernel (degree 2)

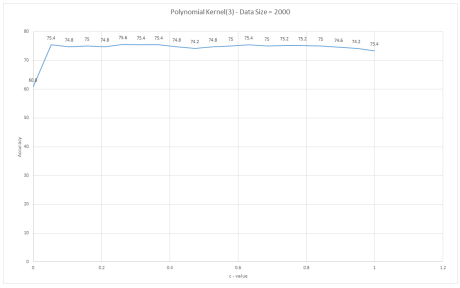


Figure 16: Polynomial Kernel (degree 3)

Following observations can be made:

- For every kernel function, the accuracy is not changing much after a certain value of ‘c’ i.e. the regularization constant.
- All kernel functions are giving the accuracy in the same range.

Insights:

- For histopathological or similar images, Hu moments  histogram are the best features to work with as Hu Moments can be used to describe, characterize, and quantify the shape of an object(cancer cell in our case) in an image.
- SVM does not work well with imbalanced data-set. The ratio of number of negative-to-positive labelled images is 3:2 i.e. 60% percent of the images belongs to the negative class. The reason why SVM is not working is that the separating hyper-plane is shifted towards the class having less number of data-points[9], this happens due to the fact that the less number of data-points from one class makes themselves appear further from the ideal class boundary as a result to lower the misclassification the separating hyper-plane is sifted towards the minority class[10].

### 4.1.3 Feature extraction

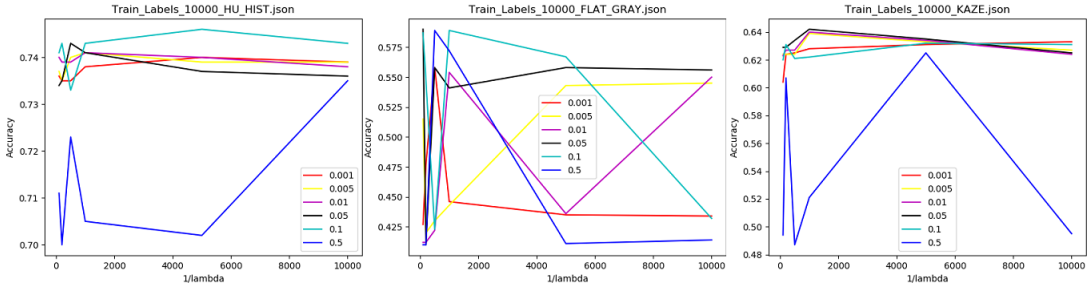


Figure 17: The different learning rates per graph for different algorithm

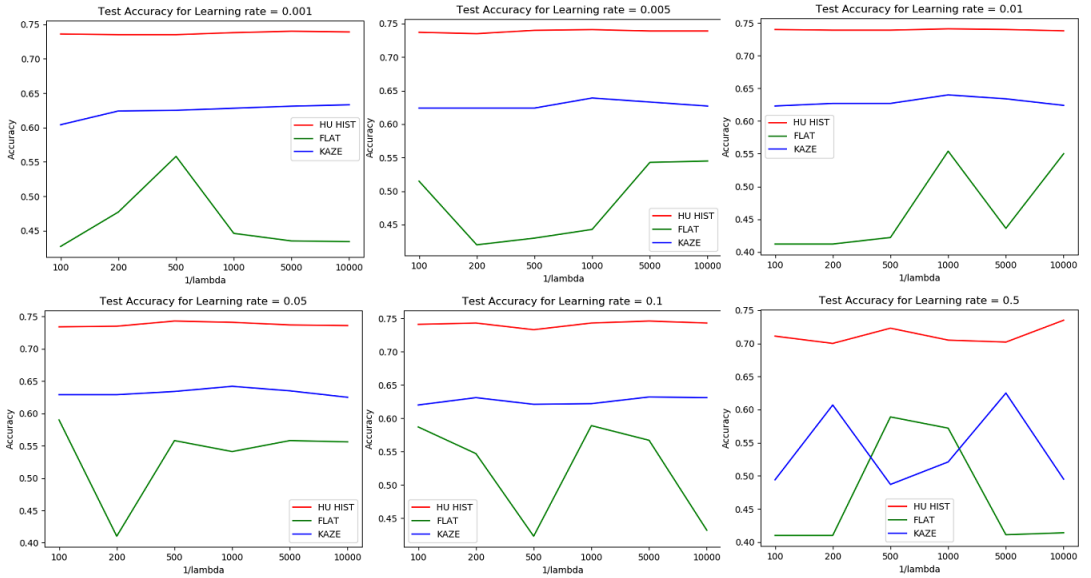


Figure 18: The test accuracy for different feature extraction algorithms for different learning rates

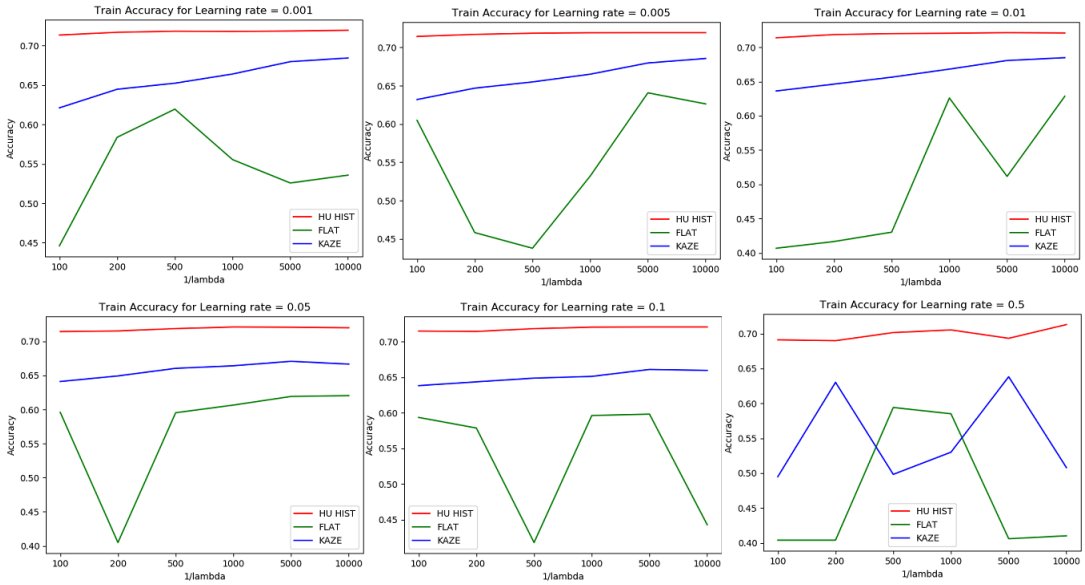


Figure 19: The train accuracy for different feature extraction algorithms for different learning rates

In our experimentation phase, the SVM was executed for different values of learning rate, regularization constant, accuracy and the different feature extraction algorithms that have been used.

1. The figure 17 shows the test accuracy vs 1/lambda graph for different learning rates in each graph for different feature extraction algorithms. The main observations for this figure is to compare how the accuracy changes for different learning rates for a particular feature space.
2. The figure 18 shows the test accuracy vs 1/lambda graph for different feature extraction algorithms for a particular learning rate. The main observation for this graph is to compare the accuracy of different feature extraction algorithms for a particular learning rate.
3. The figure 19 shows the train accuracy vs 1/lambda graph for different feature extraction algorithms for a particular learning rate. The main observation for this graph is to compare the train accuracy of different feature extraction algorithms for a particular learning rate.

Let us look at the insights that can be gained from the experiments that have been run.

1. From the figure 17 you can observe that the graph is more jagged for higher learning rates like 0.1 and 0.5. This is happening because the function overshoots the minima of the convex function multiple times before converging on it.
2. From the figure 18 you can observe that the accuracy for features extracted using HU moments and Histogram is the highest, better than KAZE and image flattening.
3. From the figure 19 you will observe that the training accuracy is less than the test accuracy i.e. the train error is greater than test error, for each respective feature extraction algorithm. One of the reasons for this could be the test set size is much less than train set size as we divided the data in 90% for training and 10% for testing. Another insight that can be derived from figure 18 and figure 19 is that as you keep on increasing the 1/lambda constant the general trend is that the training error is decreasing and the test error is increasing which leads to the belief that if 1/lambda term is very high then it may lead to overfitting of the model. It should be noted, that these graphs are working only with linear kernel. We know that for SVM the problem of overfitting may be resolved using different kernels.

The most noteworthy insight for SVM when working with our dataset is that feature extraction step plays a crucial role in determining the accuracy of the model. Not all image feature extraction

algorithms will guarantee high accuracy i.e. there is no single feature extraction algorithm that can guarantee optimal results for SVM for all kinds of datasets.

### 4.2 CNN

Here, we used two configurations for the Fully Connected Layers in the architecture of this approach:

- Config 1: 4096-2048-1024-512-2 (Fully Connected Layers)
- Config 2: 2048-1024-512-2 (Fully Connected Layers)

Table 1: Experiments Results CNN

| Learning Rate | Freezed Layers | Dropout | FC Layers | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|---|---|
| 0.001 | 0.2 | Config 1 | Max | 49.96% | 50% | 0.6932 | 0.6932 |
| 0.001 | 0.2 | Config 2 | Max | 49.98% | 50% | 0.6932 | 0.6932 |
| 0.001 | 0.2 | Config 1 | Avg | 50.07% | 50% | 0.6932 | 0.6932 |
| 0.001 | 0.2 | Config 2 | Avg | 50.02% | 50% | 0.6932 | 0.6932 |
| 0.001 | 0.5 | Config 1 | Max | 50.08% | 50% | 0.6932 | 0.6934 |
| 0.001 | 0.5 | Config 2 | Max | 50.09% | 50% | 0.6932 | 0.6932 |
| 0.001 | 0.5 | Config 1 | Avg | 49.84% | 50% | 0.6932 | 0.6931 |
| 0.001 | 0.5 | Config 2 | Avg | 50.09% | 50% | 0.6932 | 0.6933 |
| 0.00001 | 0.2 | Config 1 | Max | 92.39% | 88.56% | 0.1905 | 0.3017 |
| 0.00001 | 0.2 | Config 2 | Max | 91.69% | 89.77% | 0.2074 | 0.258 |
| 0.00001 | 0.2 | Config 1 | Avg | 92.32% | 88.18% | 0.1918 | 0.3285 |
| 0.00001 | 0.2 | Config 2 | Avg | 91.76% | 88.62% | 0.2054 | 0.2928 |
| 0.00001 | 0.5 | Config 1 | Max | 69.87% | 65.93% | 0.5451 | 0.8916 |
| 0.00001 | 0.5 | Config 2 | Max | 70.78% | 75.07% | 0.5442 | 0.5133 |
| 0.00001 | 0.5 | Config 1 | Avg | 74.75% | 69.29% | 0.5068 | 0.7503 |
| 0.00001 | 0.5 | Config 2 | Avg | 75.07% | 73.56% | 0.5064 | 0.6177 |

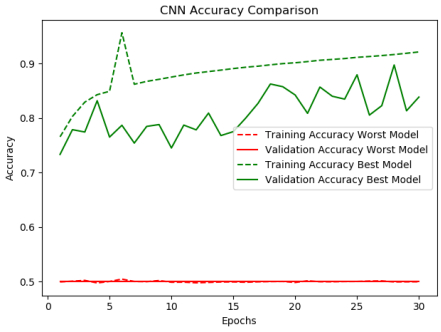


Figure 20: Comparison of best and worst model based on Accuracy

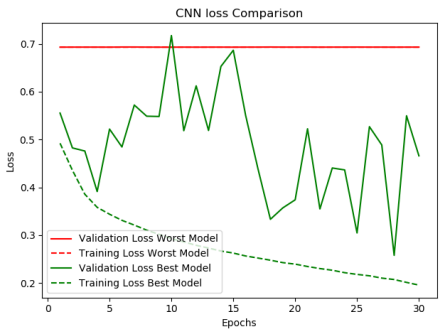


Figure 21: Comparison of best and worst model based on Loss

### 4.3 VGG - 16

Here, we used two configurations for the Fully Connected Layers in the architecture of this approach:

- Config 1: 4096-4096-512-2 (Fully Connected Layers)
- Config 2: 2048-1024-512-2 (Fully Connected Layers)

Table 2: Experiments Results VGG 16

| Learning Rate | Freezed Layers | Dropout | FC Layers | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|---|---|
| 0.00001 | No Freezing | 0.5 | Config 1 | 97.74% | 96.79% | 0.0631 | 0.0981 |
| 0.00001 | No Freezing | 0.5 | Config 2 | 98.06% | 96.73% | 0.0544 | 0.1055 |
| 0.00001 | No Freezing | 0.2 | Config 1 | 97.85% | 96.67% | 0.06 | 0.0955 |
| 0.00001 | No Freezing | 0.2 | Config 2 | 98.02% | 96.65% | 0.0554 | 0.1043 |
| 0.001 | No Freezing | 0.5 | Config 1 | 49.84% | 50.08% | 0.6932 | 0.6931 |
| 0.001 | No Freezing | 0.5 | Config 2 | 49.91% | 50.17% | 8.0292 | 7.9878 |
| 0.001 | No Freezing | 0.2 | Config 1 | 50.02% | 50.15% | 8.0112 | 7.9918 |
| 0.001 | No Freezing | 0.2 | Config 2 | 49.95% | 50.10% | 8.0235 | 7.9998 |
| 0.00001 | 4 layers | 0.5 | Config 1 | 97.62% | 96.06% | 0.0673 | 0.1127 |
| 0.00001 | 4 layers | 0.5 | Config 2 | 97.71% | 96.24% | 0.0628 | 0.1181 |
| 0.00001 | 4 layers | 0.2 | Config 1 | 97.86% | 96.32% | 0.0588 | 0.1133 |
| 0.00001 | 4 layers | 0.2 | Config 2 | 98.00% | 96.30% | 0.0548 | 0.1201 |
| 0.001 | 4 layers | 0.5 | Config 1 | 49.57% | 50.14% | 0.6932 | 0.9631 |
| 0.001 | 4 layers | 0.5 | Config 2 | 97.40% | 96.11% | 0.071 | 0.1144 |
| 0.001 | 4 layers | 0.2 | Config 1 | 50.09% | 50.20% | 8.0001 | 7.9827 |
| 0.001 | 4 layers | 0.2 | Config 2 | 97.61% | 95.76% | 0.0658 | 0.1329 |
| 0.00001 | 2 layers | 0.5 | Config 1 | 94.30% | 94.33% | 0.1554 | 0.1518 |
| 0.00001 | 2 layers | 0.5 | Config 2 | 94.36% | 94.46% | 0.1525 | 0.153 |
| 0.00001 | 2 layers | 0.2 | Config 1 | 94.61% | 94.29% | 0.1448 | 0.162 |
| 0.00001 | 2 layers | 0.2 | Config 2 | 94.70% | 94.65% | 0.1418 | 0.1433 |
| 0.001 | 2 layers | 0.5 | Config 1 | 97.00% | 96.58% | 0.0843 | 0.0987 |
| 0.001 | 2 layers | 0.5 | Config 2 | 97.12% | 96.82% | 0.081 | 0.0964 |
| 0.001 | 2 layers | 0.2 | Config 1 | 97.27% | 96.84% | 0.0745 | 0.0957 |
| 0.001 | 2 layers | 0.2 | Config 2 | 97.24% | 96.85% | 0.076 | 0.0972 |

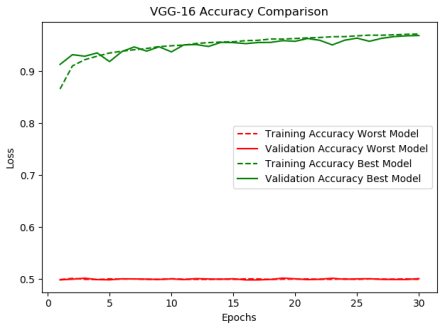


Figure 22: Comparison of best and worst model based on Accuracy

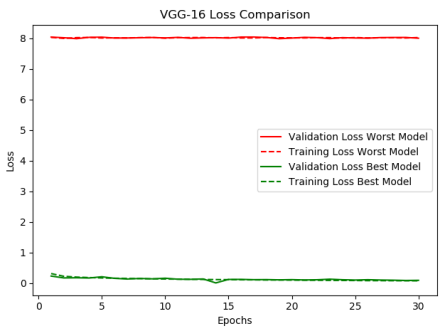


Figure 23: Comparison of best and worst model based on Loss

### 4.4 VGG - 19

Here, we used two configurations for the Fully Connected Layers in the architecture of this approach:

- Config 1: 4096-4096-1024-512-2 (Fully Connected Layers)
- Config 2: 2048-1024-512-2 (Fully Connected Layers)

Table 3: Experiments Results VGG 19

| Learning Rate | Freezed Layers | Dropout | FC Layers | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|---|---|
| 0.00001 | No Freezing | 0.5 | Config 1 | 99.65% | 96.25% | 0.0106 | 0.1875 |
| 0.00001 | No Freezing | 0.5 | Config 2 | 99.67% | 96.44% | 0.0104 | 0.1958 |
| 0.00001 | No Freezing | 0.2 | Config 1 | 99.62% | 96.63% | 0.0115 | 0.19 |
| 0.00001 | No Freezing | 0.2 | Config 2 | 99.69% | 96.44% | 0.0089 | 0.1684 |
| 0.001 | No Freezing | 0.5 | Config 1 | 50% | 50% | 8.01 | 0.6932 |
| 0.001 | No Freezing | 0.5 | Config 2 | 50.06% | 50% | 8.005 | 8.0151 |
| 0.001 | No Freezing | 0.2 | Config 1 | 50% | 50% | 8.0149 | 8.0151 |
| 0.001 | No Freezing | 0.2 | Config 2 | 50% | 50% | 8.0149 | 8.0151 |
| 0.00001 | 4 layers | 0.5 | Config 1 | 99.69% | 96.16% | 0.0105 | 0.1887 |
| 0.00001 | 4 layers | 0.5 | Config 2 | 99.68% | 96.13% | 0.011 | 0.2367 |
| 0.00001 | 4 layers | 0.2 | Config 1 | 99.67% | 95.84% | 0.0107 | 0.245 |
| 0.00001 | 4 layers | 0.2 | Config 2 | 99.37% | 96.08% | 0.0181 | 0.189 |
| 0.001 | 4 layers | 0.5 | Config 1 | 50% | 50% | 8.0146 | 8.0151 |
| 0.001 | 4 layers | 0.5 | Config 2 | 50% | 50% | 8.0148 | 8.0151 |
| 0.001 | 4 layers | 0.2 | Config 1 | 50.15% | 50% | 0.6935 | 0.6932 |
| 0.001 | 4 layers | 0.2 | Config 2 | 50.01% | 49.94% | 8.015 | 8.0151 |
| 0.00001 | 2 layers | 0.5 | Config 1 | 99.66% | 96.61% | 0.01 | 0.2 |
| 0.00001 | 2 layers | 0.5 | Config 2 | 99.68% | 96.51% | 0.0094 | 0.233 |
| 0.00001 | 2 layers | 0.2 | Config 1 | 99.65% | 96.16% | 0.0107 | 0.22 |
| 0.00001 | 2 layers | 0.2 | Config 2 | 99.57% | 96.50% | 0.0126 | 0.1607 |
| 0.001 | 2 layers | 0.5 | Config 1 | 50% | 50% | 8.0151 | 8.0151 |
| 0.001 | 2 layers | 0.5 | Config 2 | 50% | 50% | 8.0149 | 8.0151 |
| 0.001 | 2 layers | 0.2 | Config 1 | 50% | 50% | 8.0151 | 8.0151 |
| 0.001 | 2 layers | 0.2 | Config 2 | 50% | 50% | 8.0149 | 8.0151 |

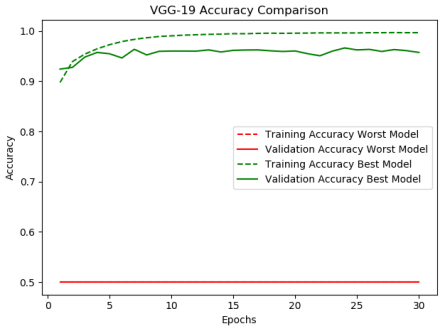


Figure 24: Comparison of best and worst model based on Accuracy

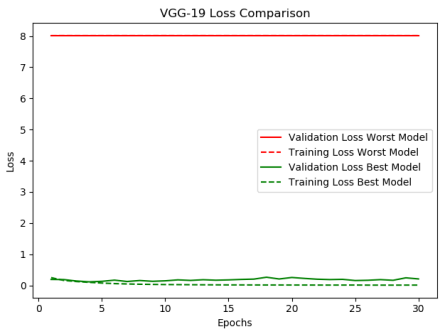


Figure 25: Comparison of best and worst model based on Loss

### 4.5 ResNet

Here, we used two configurations for the Fully Connected Layers in the architecture of this approach:

- Config 1: 4096-2048-1024-512-2 (Fully Connected Layers)
- Config 2: 2048-1024-512-2 (Fully Connected Layers)

Table 4: Experiments Results ResNet

| Learning Rate | Freezed Layers | Dropout | FC Layers | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|---|---|
| 0.00001 | No Freezing | 0.5 | Config 1 | 97.97% | 97.07% | 0.0585 | 0.095 |
| 0.00001 | No Freezing | 0.5 | Config 2 | 97.71% | 97.22% | 0.0648 | 0.0847 |
| 0.00001 | No Freezing | 0.2 | Config 1 | 94.02% | 94.82% | 0.1598 | 0.1391 |
| 0.00001 | No Freezing | 0.2 | Config 2 | 98.58% | 97.31% | 0.0396 | 0.0943 |
| 0.001 | No Freezing | 0.5 | Config 1 | 98.81% | 97.68% | 0.0327 | 0.0817 |
| 0.001 | No Freezing | 0.5 | Config 2 | 97.97% | 97.34% | 0.0566 | 0.0835 |
| 0.001 | No Freezing | 0.2 | Config 1 | 96.89% | 97.05% | 0.0868 | 0.0871 |
| 0.001 | No Freezing | 0.2 | Config 2 | 98.41% | 97.55% | 0.0452 | 0.0823 |

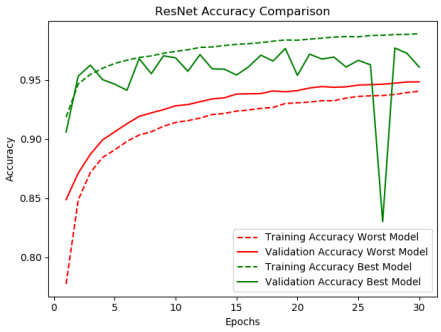


Figure 26: Comparison of best and worst model based on Accuracy

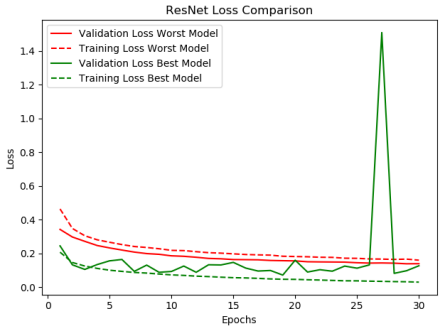


Figure 27: Comparison of best and worst model based on Loss

### 4.6 DenseNet

Here, we used two configurations for the Fully Connected Layers in the architecture of this approach:

- Config 1: 4096-2048-1024-512-2 (Fully Connected Layers)
- Config 2: 2048-1024-512-2 (Fully Connected Layers)

Table 5: Experiments Results DenseNet

| Learning Rate | Freezed Layers | Dropout | FC Layers | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|---|---|
| 0.00001 | No Freezing | 0.5 | Config 1 | 98.34% | 97.66% | 0.0469 | 0.0978 |
| 0.00001 | No Freezing | 0.5 | Config 2 | 98.42% | 97.75% | 0.045 | 0.0758 |
| 0.00001 | No Freezing | 0.2 | Config 1 | 98.49% | 97.73% | 0.0412 | 0.0726 |
| 0.00001 | No Freezing | 0.2 | Config 2 | 98.51% | 97.72% | 0.0411 | 0.0692 |
| 0.001 | No Freezing | 0.5 | Config 1 | 89.76% | 88.27% | 0.5443 | 0.3301 |
| 0.001 | No Freezing | 0.5 | Config 2 | 92.30% | 93.36% | 0.2187 | 0.2228 |
| 0.001 | No Freezing | 0.2 | Config 1 | 98.53% | 97.77% | 0.41 | 0.0649 |
| 0.001 | No Freezing | 0.2 | Config 2 | 96.08% | 95.97% | 0.1134 | 0.1214 |

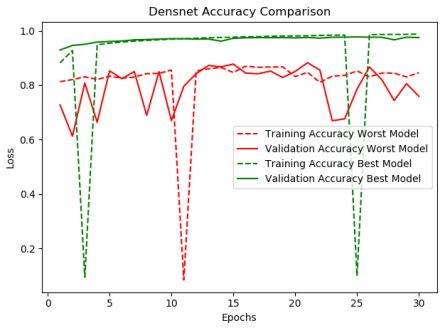


Figure 28: Comparison of best and worst model based on Accuracy

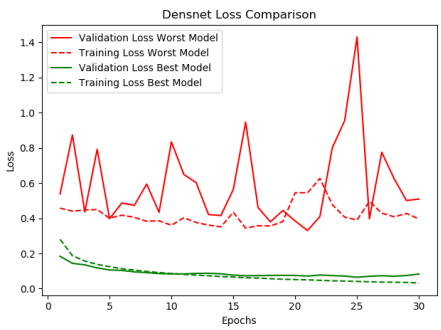


Figure 29: Comparison of best and worst model based on Loss

### 4.7 Xception

Here, we used two configurations for the Fully Connected Layers in the architecture of this approach:

- Config 1: 4096-2048-1024-512-2 (Fully Connected Layers)
- Config 2: 2048-1024-512-2 (Fully Connected Layers)

Table 6: Experiments Results Xception

| Learning Rate | Freezed Layers | Dropout | FC Layers | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|---|---|
| 0.00001 | No Freezing | 0.5 | Config 1 | 94.74% | 94.60% | 0.2378 | 0.1636 |
| 0.00001 | No Freezing | 0.5 | Config 2 | 99.68% | 94.42% | 0.0108 | 0.3338 |
| 0.00001 | No Freezing | 0.2 | Config 1 | 99.73% | 94.53% | 0.0093 | 0.3149 |
| 0.00001 | No Freezing | 0.2 | Config 2 | 94.09% | 94.60% | 0.1609 | 0.1636 |
| 0.001 | No Freezing | 0.5 | Config 1 | 52.21% | 59.79% | 0.8444 | 0.8513 |
| 0.001 | No Freezing | 0.5 | Config 2 | 49.83% | 50% | 0.6932 | 0.6932 |
| 0.001 | No Freezing | 0.2 | Config 1 | 94.88% | 94.61% | 0.1107 | 0.1636 |
| 0.001 | No Freezing | 0.2 | Config 2 | 95.62% | 95.01% | 0.1218 | 0.1416 |

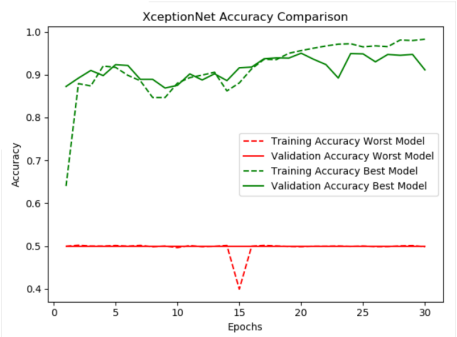


Figure 30: Comparison of best and worst model based on Accuracy

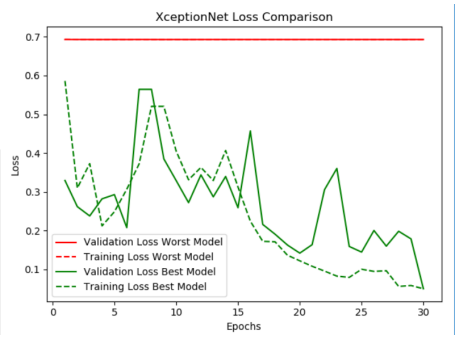


Figure 31: Comparison of best and worst model based on Loss

# 5 Analysis and Insights

## 5.1 CNN

- Best accuracy was achieved for the architecture with 13 layers in which 9 convolutional layers were there and 4 fully connected layers with 2048 nodes, 1024 nodes, 512 nodes and lastly two nodes, learning rate 0.00001 and used max-pooling and dropout 0.2.
- Higher learning rate resulted in taking the model with accuracy of 89.97% to almost 50%, we think that overshooting was the cause of this problem.

## 5.2 VGG - 16

- Best accuracy was achieved for the architecture where the fully connected layers were 2048 nodes, 1024 nodes, 512 nodes and lastly two nodes, learning rate 0.001, 2 freezed layers and dropout 0.2.
- Took less time to train the model because transfer learning was used.

## 5.3 VGG - 19

- Best accuracy was achieved for the architecture where the fully connected layers were 4096 nodes, 4096 nodes, 1024 nodes, 512 nodes and lastly two nodes, learning rate 0.00001, no freezed layers and dropout 0.2. Here we observed that the best accuracy of 96.63% was achieved for the textbook architecture of VGG-19.
- Increasing the accuracy takes the accuracy from 96.63% to almost random guessing.
- Freezing the layers also reduces the accuracy.

## 5.4 ResNet - 50

- Best accuracy for this model was achieved when the learning rate was 0.001, with no freezing layers in the original ResNet50 model and dropout rate 0.5. The number of nodes in fully connected layers were, 4096, 2048, 1024, 412 and 2.
- When the first Fully connected layer had 2048 nodes, the accuracy of the model decreased because it is not as per the traditional ResNet50 architecture.

## 5.5 DenseNet - 121

- Among all the models that we implemented, DenseNet gave us the highest validation accuracy. The validation accuracy that we achieved was 97.73%.
- It was achieved for 0.001 learning rate with no freezing layers in the original model, dropout of 0.2 and there are 5 fully connected layers having 4096 nodes, 2048 nodes, 1024 nodes, 512 nodes and the last one with 2 nodes.
- The reason behind its high performance is that it doesn't learn redundant feature maps, this resulted in lesser parameters for this model.

## 5.6 Xception

- Best accuracy was achieved for the architecture where the fully connected layers were 2048 nodes, 1024 nodes, 512 nodes and lastly two nodes, learning rate 0.001, no freezed layers and dropout 0.2. Here we observed that the best accuracy of 95.01%.

### 5.7 Comparative study of Deep learning models

Table 7: Experiments Results Xception

| | Layers | Training % Accuracy | Validation % Accuracy | Training Loss | Validation Loss |
|---|---|---|---|---|---|
| CNN | 13 | 91.69% | 89.77% | 0.2074 | 0.258 |
| VGG 16 | 17 | 97.24% | 96.85% | 0.076 | 0.0972 |
| VGG 19 | 21 | 99.62% | 96.63% | 0.0115 | 0.1944 |
| Resnet 50 | 50 | 98.81% | 97.68% | 0.0327 | 0.0817 |
| Densenet 121 | 121 | 98.53% | 97.77% | 0.41 | 0.0649 |
| XceptionNet | 71 | 95.62% | 95.01% | 0.1218 | 0.1416 |

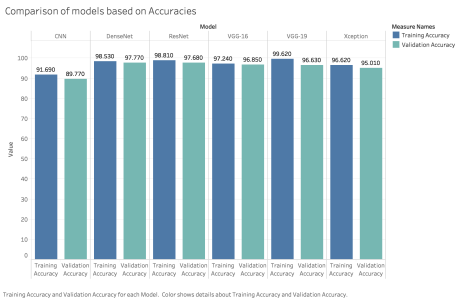


Figure 32: Highest Training and Validation Accuracy of each model

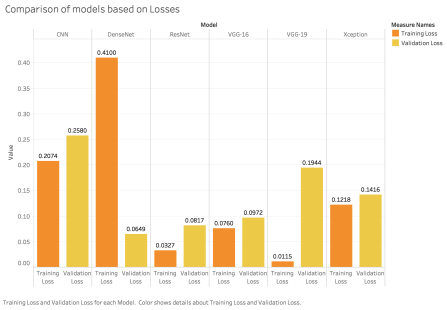


Figure 33: Highest Training and Validation Loss of each model

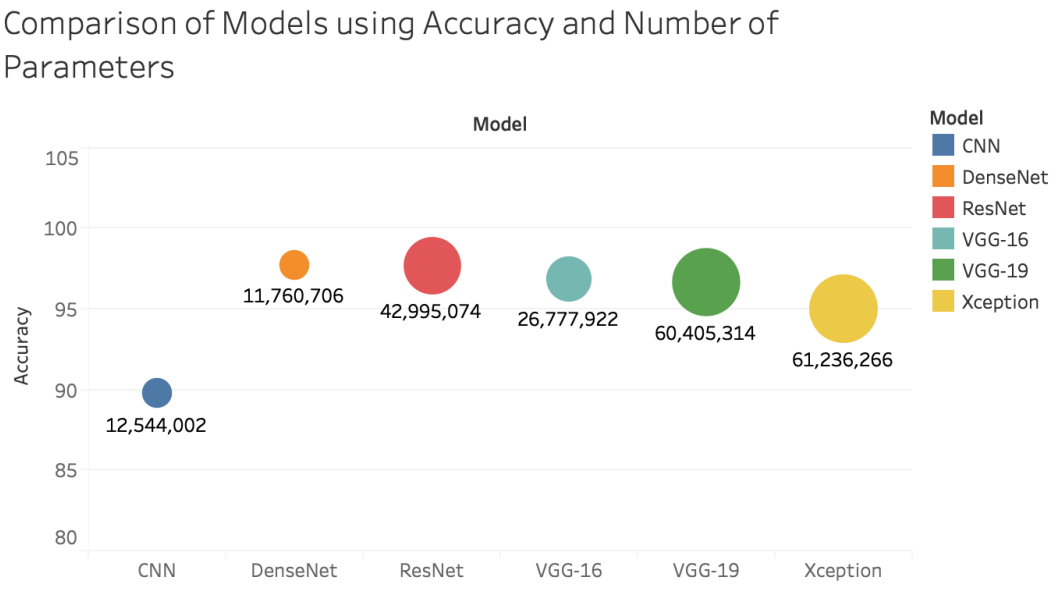


Figure 34: Comparison of models based on highest Validation Accuracy and Number of Parameters

## 6 Conclusion

We successfully completed our objective of this project i.e. a successful comparative study of deep neural networks and support vector machines. We gained deep insights for the histopathological cancer data-set that can be used in future while working with any data-set comprising similar images.

## 7 Future Work

- If we had more time we would like to tweak some more parameters to improve accuracy of model.
- We would also like to test this dataset on some more deep learning architecture.

## 8 Division of Work

| Team Member | Contribution |
|---|---|
| Ishani Bhatt | ResNet Architecture Implementation and analysis |
| Arpan Vitthalbhai Patel | DenseNet Architecture implementation and analysis |
| Darshan Dhirubhai Malaviya | VGG-16 Architecture implementation and analysis |
| Harshal Mukeshkumar Trivedi | VGG-19 Architecture implementation and analysis |
| Jaykumar Keshavbhai Vaghasiya | CNN Architecture implementation and analysis |
| Venkateswara Prasad Kaikala | XceptionNet Architecture implementation and analysis |
| Shreerang Hegde | Feature extraction, Gradient Descent SVM implementation and analysis |
| Vishal Kumar | Feature extraction, Lagrange dual formulation SVM implementation and analysis |

## 9 Acknowledgement

We would like to thank Dr. Jingrui He and Teaching Assistants Jun Wu and Dawei Zhou for giving this wonderful opportunity to learn more about the classical and state-of-the-art Machine Learning algorithms and understand the working of those algorithms. We are also thankful to Professor for giving us permission and access to Agave cluster for training and testing our machine learning models which helped us a lot in reducing our computation time by aiding us with high computational capabilities.

## References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Tutorial slides by Andrew Moore.(n.d.) Retrieved April 20, 2019, from http://www.cs.cmu.edu/ awm

[5] Figure 1, 2: Cortes, C. and Vapnik, V. (1995) Support-Vector Networks. Machine Learning, 20, 273-297.

[6] Figure 3: Kernel method (June 27 2017) Retrieved April 20, 2019, from: https://en.wikipedia.org/wiki/Kernel_method copyright Shiyu Ji

[7] Mathieu's log (n.d.) Retrieved April 20, 2019 , from: https://web.archive.org/web/20140429090836/http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/

[8] Quadratic Programming with Python and CVXOPT (n.d.) Retrieved April 21, 2019, from: https://courses.csail.mit.edu/6.867/wiki/images/a/a7/Qp-cvxopt.pdf

[9] G. Wu and E. Chang, “Adaptive feature-space conformal transformation for imbalanced-data learning,” in Proceedings of the 20th International Conference on Machine Learning, pp. 816–823, 2003.

[10] V. Palade, “Class imbalance learning methods for support vector machines,” 2013.

[11] Lecture 2: The SVM classifier (n.d.) Retrieved April 20, 2019, from: http://www.robots.ox.ac.uk/ az/lectures/ml/lect2.pdf

[12] KAZE Features (n.d.) Retrieved April 20, 2019, from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.716.3013rep=rep1type=pdf

[13] Anisotropic diffusion (n.d.) Retrieved April 20, 2019, from: https://en.wikipedia.org/wiki/Anisotropic_diffusion

[14] Anisotropic diffusion (n.d.) Retrieved April 20, 2019, from: https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/

[15] Shape Matching using Hu Moments (n.d.) Retrieved April 20, 2019, from: https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/

[16]Understanding and Coding Resnet in Keras (n.d) Retrieved April 21, 2019, from:https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

[17] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[18] Veta, M., Pluim, J. P., van Diest, P. J., and Viergever, M. A. (2014). Breast cancer histopathology image analysis: a review. IEEE Transac. Biomed. Eng. 61, 1400–1411. doi: 10.1109/TBME.2014.2303852

[19] Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recog. 30, 1145–1159. doi: 10.1016/S0031-3203(96)00142-2

[20] Figure 8: ImageNet: Xception with Keras by Adrian Rosebrock from: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

[21] PatchCamelyon (PCam) (n.d) retrieved April 26, 2019 from https://github.com/basveeling/pcam

[22] Histopathologic Cancer Detector - Machine Learning in Medicine (n.d) Retrieved April 26, 2019 from https://towardsdatascience.com/histopathologic-cancer-detector-finding-cancer-cells-with-machine-learning-b77ce1ee9b0a

[23] Figure 6: Common architectures in convolutional neural networks by Jeremy Jordan from: https://www.jeremyjordan.me/convnet-architectures/

[24] Figure 7: Understanding and visualizing DenseNets by Pablo Ruiz Ruiz from: https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a

[25] Figure 4: Convolutional neural network from: https://en.wikipedia.org/wiki/Convolutional_neural_network

[26] Figure 5: Review: VGGNet—1st Runner-Up (Image Classification), Winner (Localization) in ILSVRC 2014 by Sik-Ho Tsang from: https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11

[27] Xceptionnet by Mathworks from: https://www.mathworks.com/help/deeplearning/ref/xception.html

[28] Convolutional Neural Networks(n.d.) retrieved April 26, 2019 from: https://en.wikipedia.org/wiki/Convolutional_neural_network

[29] Ciresan, D., U. Meier, and J. Schmidhuber. "Multi-column Deep Neural Networks for Image Classification." 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012): 3642-649. Web.

[30] Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." (2014). Web.