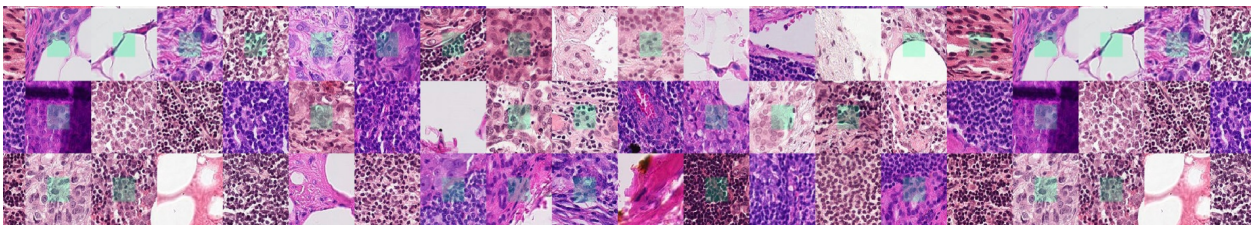
Histopathologic Cancer Detection

Identifying Metastatic Tissue in Histopathologic Scans of Lymph Node Sections.



— — —

A Project By:

Kartikey Sarode, Sudhanshu Kulkarni, Sanskriti Kapoor, Purva Zinjarde

Instructions on running the program

Dependencies:
Language: Python 3.8
Libraries: Scikit Learn, Keras, Tensorflow, Tensorflow-GPU, Pandas, Numpy, Matplotlib, OpenCV

KNN & SVM.ipynb:
Write the path of the data content folder in the `'pd.read_csv'` and `'imagePath'`.

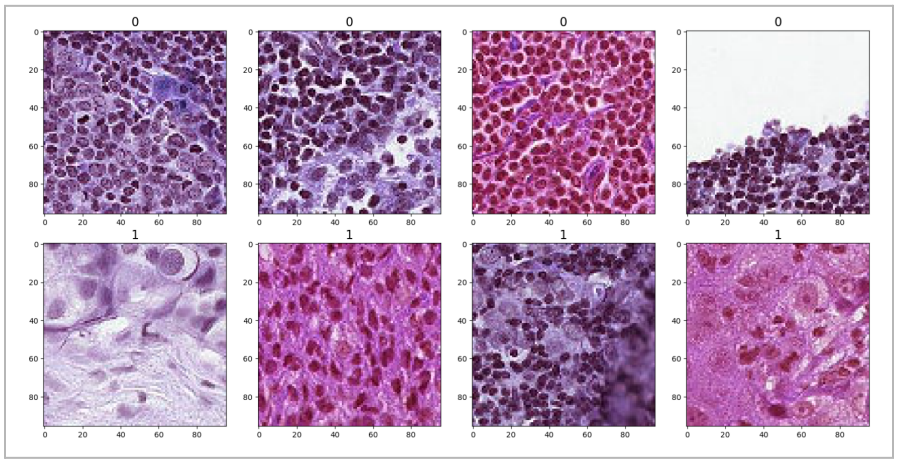
CNN:
Update all paths for input data by replacing "`../input`" with the folder which contains train and test images.

Note: It might take the programs a long time to run (more than 2 hours).

Problem Definition:

Training an optimum model to detect if metastatic cancer exists in images extracted from histopathologic scans of lymph node sections.

A large dataset of images as given below is provided for training and testing the classifiers. The goal is to identify metastatic cancer in small image patches taken from larger digital pathology scans.



A sample of histopathologic scans of lymph node sections

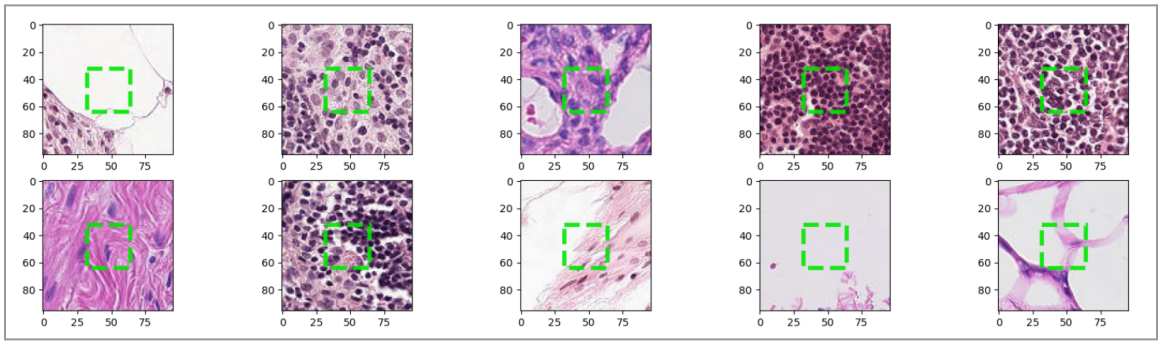
Dataset Description:

The dataset has been obtained from a Kaggle competition which can be accessed with this link.
The dataset is a modified version of the original PCam benchmark dataset. All the images were extracted from histopathological scans of lymph node sections. All the images in this dataset are preprocessed, resized, and unique.

Dataset Characteristics:

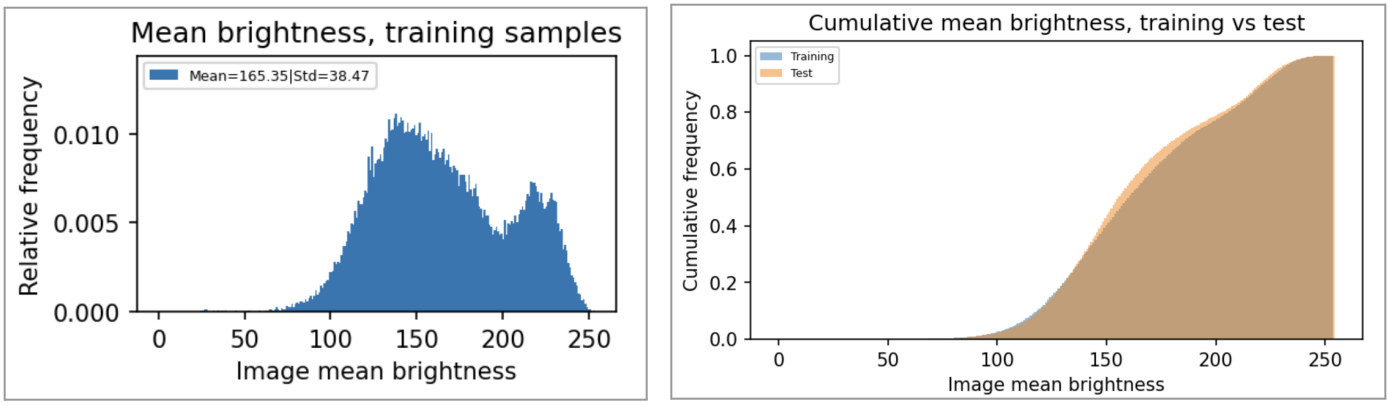
- Size: 220,025 images
- Attributes: 96 x 96 size.
- Missing data: N/A
- Data distribution: The images in the training dataset have a 60 - 40 distribution, with 60% being healthy cells and 40% cancerous.

Since the testing dataset does not have any target labels, the training dataset has been divided into 3 parts - training, validation, and testing. Each image in the training dataset is annotated with a binary label indicating the presence of metastatic tissue. A positive label indicates that the center 32 x 32px region of a patch contains at least one pixel of tumor tissue. Tumor tissue in the outer region of the patch does not influence the label.



Samples showing the 32x32 *px region used to annotate samples*

Exploratory Data Analysis (EDA) on the images showed that all the images in the training and testing dataset have almost identical values for mean image brightness distributions, cumulative mean brightness, and distribution of pixel values w.r.t. RGB values.



Graphical representation of mean image brightness distributions and Cumulative mean brightness distribution

Main Strategies:

Our objective was to implement different classifiers and compare the accuracy in order to find the best possible solution.
3 different types of classifiers have been implemented:

1. SVM
2. CNN
3. KNN

A comparison study has been performed in order to bring out the most optimized version of the classifiers. Different parameter combinations and training dataset sizes have been tested, and the most optimal solution for each classifier has been found.

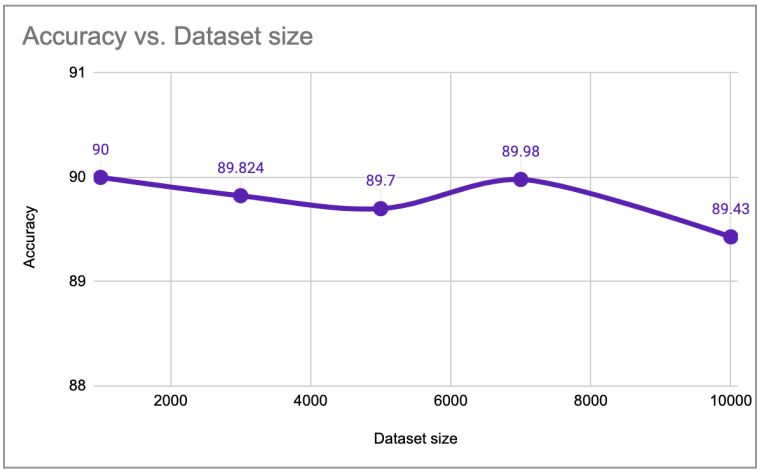
Evaluation Strategy:

1. SVM:

SVM was implemented for the dataset as one of the classifiers. SVM worked really well for a smaller chunk of data as compared to the other two classifiers, providing an accuracy of 89.43% for the testing dataset. Benchmarking of all three algorithms was performed for 10,000 images, and SVM turned out to be the more promising of all three.

The only drawback with SVM was that it took a lot of processing time, even for small chunks of data.

Different Kernel combinations were tried along with different hyperparameters, but the most optimal solution was a combination of RBF and Poly kernel. SVM gave very consistent results for varying sizes of datasets.



Accuracy of classifier vs Dataset size

Results:

Dataset size	10,000 images
Training Accuracy	89.4375 %
Testing Accuracy	88.3125%
Confusion Matrix (for testing dataset)	[[833 109] [78 580]]
F1 Score	0.86%
Recall Score	0.88%

2. CNN:

In this part, a CNN (Convolutional Neural Network) model was trained by using the Keras API on top of the Tensorflow framework. Using TensorFlow allowed us to leverage GPUs for training the model without any specific modifications to the code. Furthermore, using the ***ImageDataGenerators*** feature of TensorFlow lets us pass images in small batches as opposed to passing all of them while the model is training.

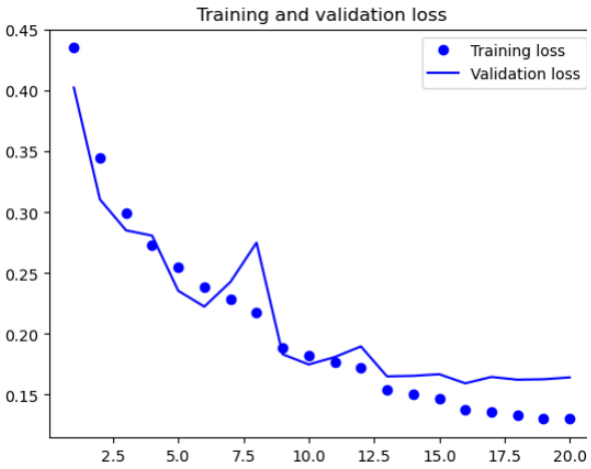
The model uses a simple neural network with three phases in the hidden layer. Each phase consists of three convolutions (*Conv2D*) layers followed by a MaxPooling layer and a Dropout layer. At the end of these three phases, the model has the *Fully Connected layer* where we Flatten the input and use the *SoftMax* activation function to calculate the output label.

Using this approach, we were able to get 94.08% accuracy and an AUC (Area Under the Curve) score of 0.98 when using 144,000 images for training and 16,000 images for evaluation (split equally between both the output classes).

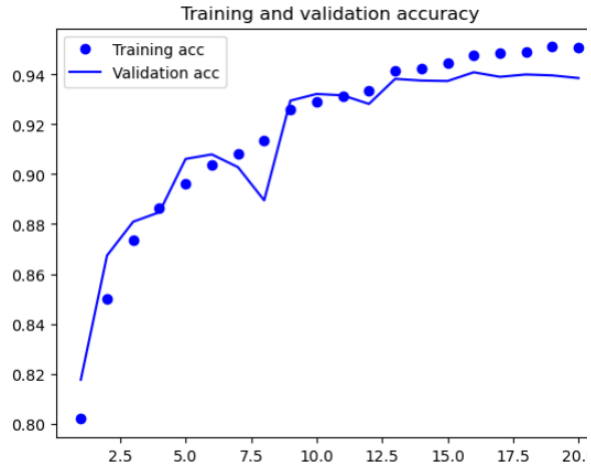
Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 94, 94, 32)	896
conv2d_10 (Conv2D)	(None, 92, 92, 32)	9248
conv2d_11 (Conv2D)	(None, 90, 90, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 45, 45, 32)	0
dropout_4 (Dropout)	(None, 45, 45, 32)	0
conv2d_12 (Conv2D)	(None, 43, 43, 64)	18496
conv2d_13 (Conv2D)	(None, 41, 41, 64)	36928
conv2d_14 (Conv2D)	(None, 39, 39, 64)	36928
max_pooling2d_4 (MaxPooling2	(None, 19, 19, 64)	0
dropout_5 (Dropout)	(None, 19, 19, 64)	0
conv2d_15 (Conv2D)	(None, 17, 17, 128)	73856
conv2d_16 (Conv2D)	(None, 15, 15, 128)	147584
conv2d_17 (Conv2D)	(None, 13, 13, 128)	147584
max_pooling2d_5 (MaxPooling2	(None, 6, 6, 128)	0
dropout_6 (Dropout)	(None, 6, 6, 128)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_2 (Dense)	(None, 256)	1179904
dropout_7 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 2)	514

Total params: 1,661,186
Trainable params: 1,661,186
Non-trainable params: 0

Model Architecture



Training & Validation Loss per epoch



Training & Validation Accuracy per Epoch

3. KNN:

KNN was implemented for the dataset as one of the classifiers. KNN worked really well for a substantial chunk of data, providing an accuracy of 90.34% for the testing dataset.

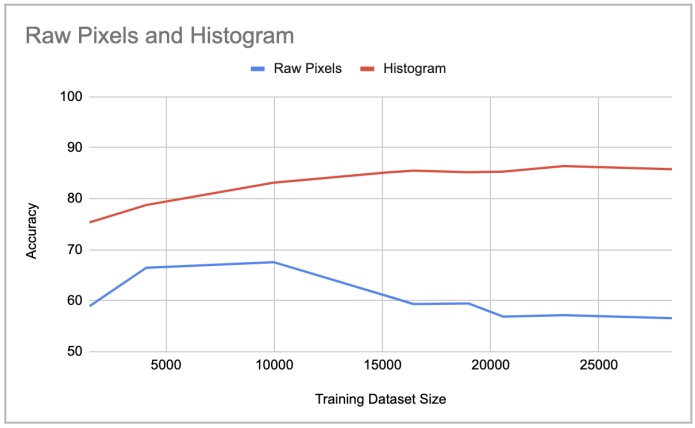
The input data for KNN is of two types:

- Raw pixel data: Convert images to a feature vector, or a list of numbers that quantify the contents of an image. The RGB pixel intensities are flattened into a single list of numbers.
- Histogram data: The input images are converted and a color histogram is constructed to characterize the color distribution of the image. The computed histogram is then normalized.

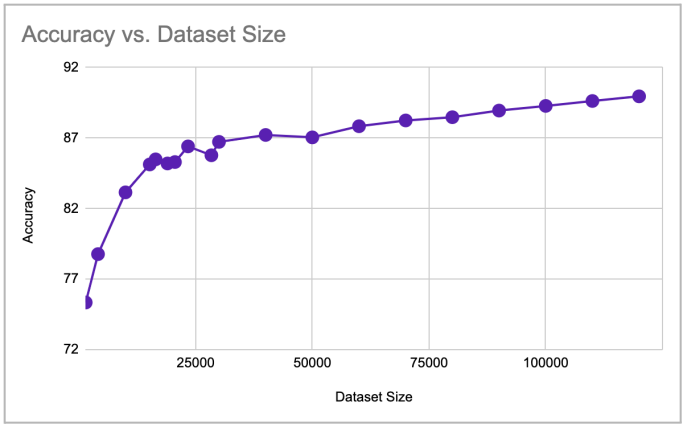
Raw pixel input can be processed only for 1400 - 20,000 images. Further than that, the RAM turns out to be insufficient and the system crashes. Also, as compared to Histogram, utilizing raw pixel intensities as inputs to machine learning algorithms tends to yield poor results as even small changes in rotation, translation, viewpoint, scale, etc., can dramatically influence the image itself, and thus the output feature representation.

Hence, Histogram data has been used for the classifier, since it yields better accuracy and can process up to 120,000 images.

The accuracy for histogram data as inputs shows a steady increase as the dataset size increases.

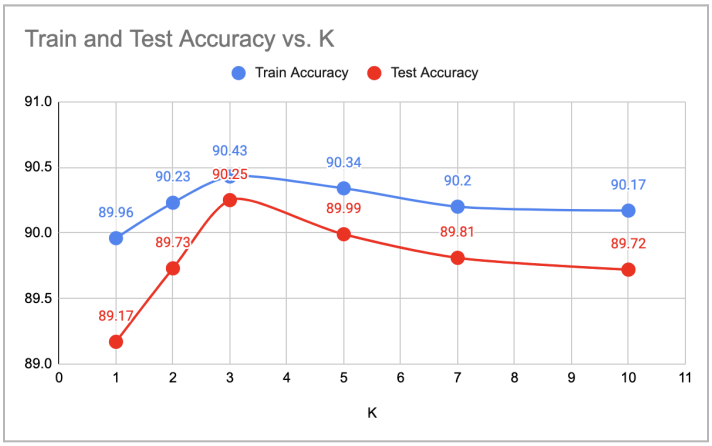


Accuracy of Raw Pixels and Histogram vs Dataset size



Accuracy of classifier vs Dataset size

Different values of K have been tested for training and testing dataset, and the optimal value of K has been deemed to be 3. K values 1, 2, 3, 5, 7, and 10 have been tested.



Training and Testing Accuracy of classifier vs K value

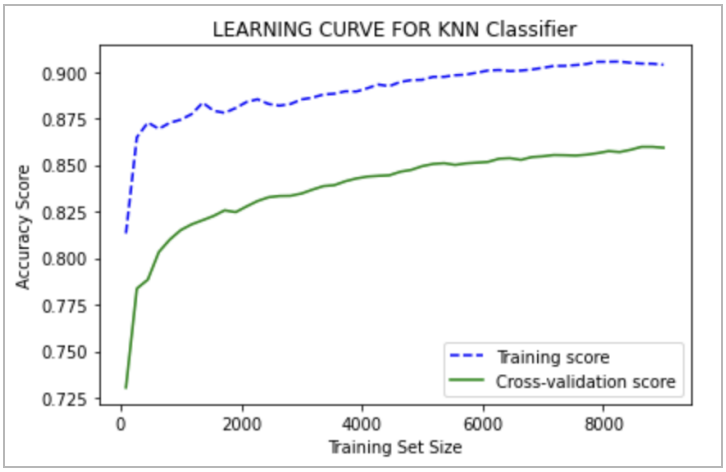
Results:

Dataset size	120,000 images
Training Accuracy	90.33%
Testing Accuracy	90.343%
Confusion Matrix (for testing dataset)	[[10294 732] [1122 7052]]
F1 Score	0.88
Recall Score	0.896

Historically, KNN does not work well for large datasets, which was true for raw pixels as an input, however, it worked very well with histogram input in providing an adequate accuracy and precision.

Learning curve for KNN:

Learning curve was implemented for KNN for Accuracy vs Dataset size. Ideally, the learning curve should be implemented for the entire data set. We were able to process a substantial chunk of images for KNN, however, since the function couldn't handle the amount of data, hence the learning curve has been implemented for 20,000 images.



Learning Curve of Accuracy score vs Dataset size

Pros and Cons of the Strategies:

SVM

Pros:

1. SVM is a great strategy when the dataset has higher dimensions.
2. Outliers have very little impact on SVM.
3. Works well with the imbalance and smaller size training dataset.

Cons:

1. SVM is slow when it comes to computing time especially when there are a large number of features involved, and the dataset size is large.
2. It performs poorly when there is an overlap between the classes.

KNN

Pros:

1. It is very simple to understand and implement.
2. There are no assumptions made about the data.
3. It has only one hyperparameter K.

Cons:

1. It does not work very well with higher dimensions or features.
2. It is sensitive to outliers.

CNN

Pros:

1. CNN learns filters automatically by extracting the right and relevant features i.e. it is not dependent on having domain knowledge.
2. CNN captures the spatial features from an image.

Cons:

1. It requires a large dataset to train which is why for 10000 images SVM performed better than CNN.
2. A low number of hidden layers results in poor accuracy.
3. Need GPU to train faster.

Conclusion and Future Enhancements:

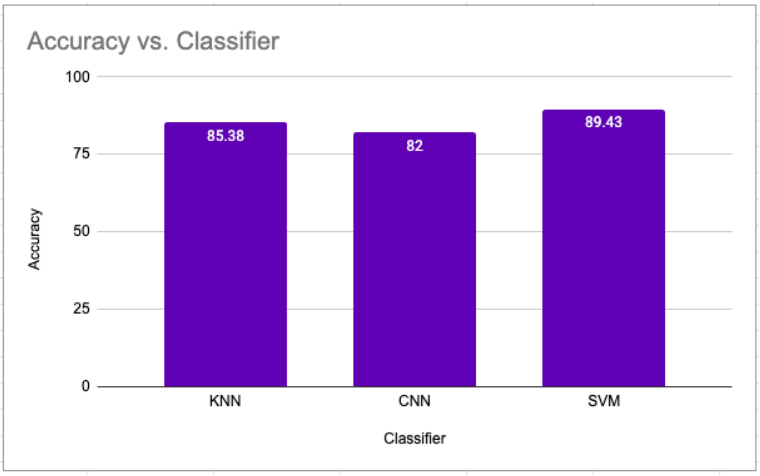
An evaluation of 10,000 images was performed and a comparison of results for the 3 classifiers is as follows:

	CNN	KNN	SVM
Accuracy	82%	85.38%	89.43%

Additionally, these were the results for the individual classifiers.

- KNN: 90% accuracy for 120,000 images.
- CNN: 93% accuracy for 144,000 images.

By looking at the accuracy of the models, we can say that with proper image pre-processing and noise reduction, CNN is the best choice of classifier if we have a large-size training dataset. It is both more precise and computationally efficient. But if we do not have a large dataset to train the CNN model, SVM is a fairly good performing model. This easy and efficient binary classification model helps detect cancer at an early stage and helps identify which cells have a cancerous tumor from metastatic cancer. Originally radiologists assessed the images by looking at the medical images and monitoring the characteristics of the diseases.
This resulted in a lot of false positives resulting in further, unnecessary, and more invasive investigation. But with classification models like these, we can automatically detect complex patterns in the images. Although a false negative could be detrimental to the patient's life. Image classification problems have a scope beyond cancer detection like disease grading and fine-grained lesion segmentation. Also can be used to detect infection, cancers, traumatic injuries, and abnormalities in blood vessels and organs.



Comparative analysis of all three classifiers benchmarked on 10,000 images

Future Enhancements would be the following:

- Explore libraries that support GPU training for SVM and KNN algorithms.
- Run the training jobs on Cloud Platforms like GCP/AWS since they provide machines with higher configurations.
- Exploring different variations of the CNN and benchmarking performance.

Team Contributions:

Kartikey and Sudhanshu:

- Part of the EDA process.
- Creating the CNN model architecture.
- Implementing the CNN model (including hyperparameter tuning and running the model on GPU).

Purva:

- Part of the EDA process.
- Implemented the SVM classifier.
- Implemented the KNN classifier.

Sanskriti:

- Contributed with documentation.
- Helped in initial stages of SVM.