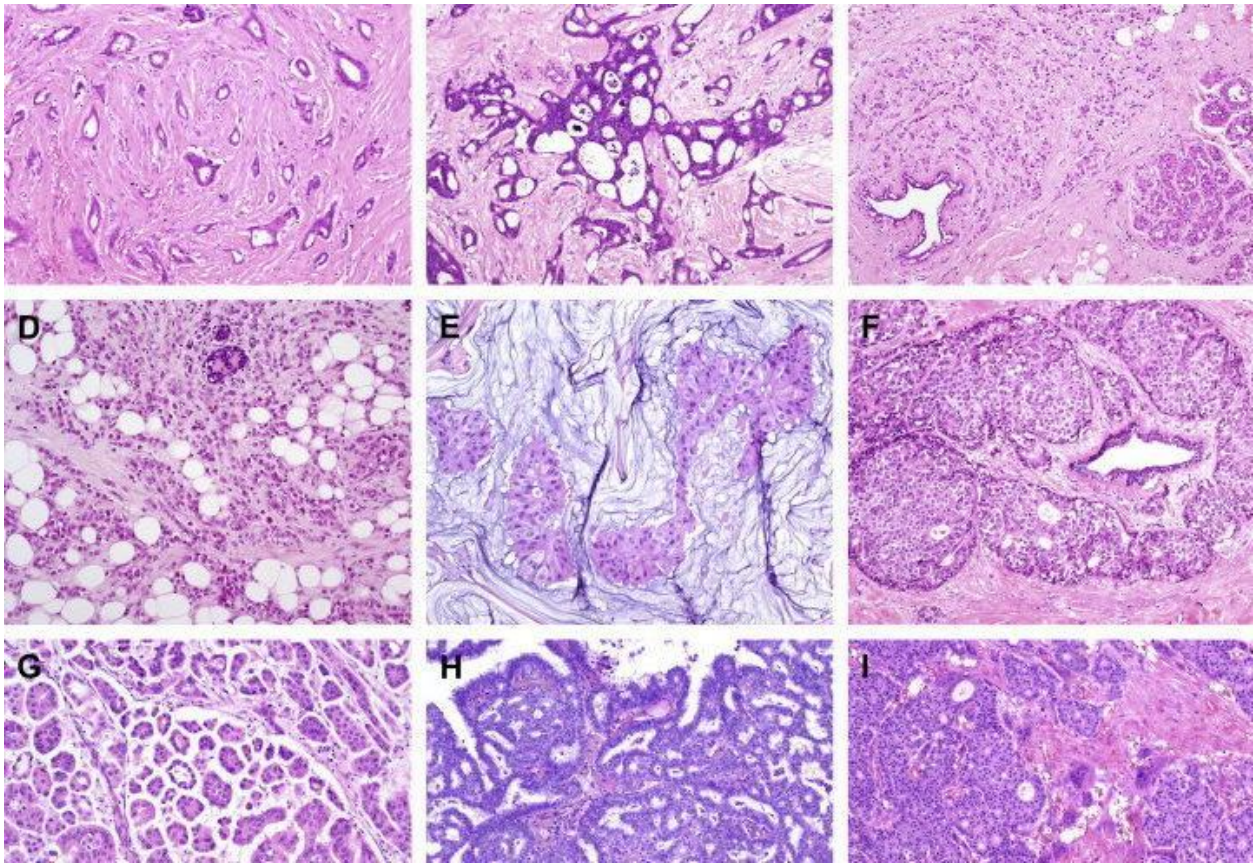
Histopathologic Cancer Detection

Problem:

The aim is to create an algorithm to identify metastatic cancer in small image patches taken from larger digital pathology scans. We detect cancer by identifying metastatic tissue in histopathologic scans of lymph nodes using Deep Learning.

What is Histopathology?

Histopathology is the diagnosis and study of diseases of the tissues, and involves examining tissues and/or cells under a microscope. It is the study of the signs of the disease using the microscopic examination of a biopsy or surgical specimen that is processed and fixed onto glass slides. To visualize different components of the tissue under a microscope, the sections are dyed with one or more stains.



Lymph nodes are small glands that filter the fluid in the lymphatic system and they are the first place a breast cancer is likely to spread. Histological assessment of lymph node metastases is part of determining the stage of breast cancer in TNM classification which is a globally recognized standard for classifying the extent of spread of cancer. The diagnostic procedure for pathologists is tedious and time-consuming as a large area of tissue has to be examined and small metastases can be easily missed. Hence using Deep Learning and Machine Learning Models provide an efficient alternative.

Dataset:

The dataset is a slightly modified version of the PatchCamelyon benchmark dataset. The original PCam dataset contains duplicate images due to its probabilistic sampling, however, this does not contain duplicates.

It can be downloaded here:
https://www.kaggle.com/c/histopathologic-cancer-detection/data

The PatchCamelyon benchmark is a new and challenging image classification dataset. It consists of 327.680 color images (96 x 96px) extracted from histopathologic scans of lymph node sections. Each image is annotated with a binary label indicating presence of metastatic tissue. PCam provides a new benchmark for machine learning models: bigger than CIFAR10, smaller than imagenet, trainable on a single GPU.

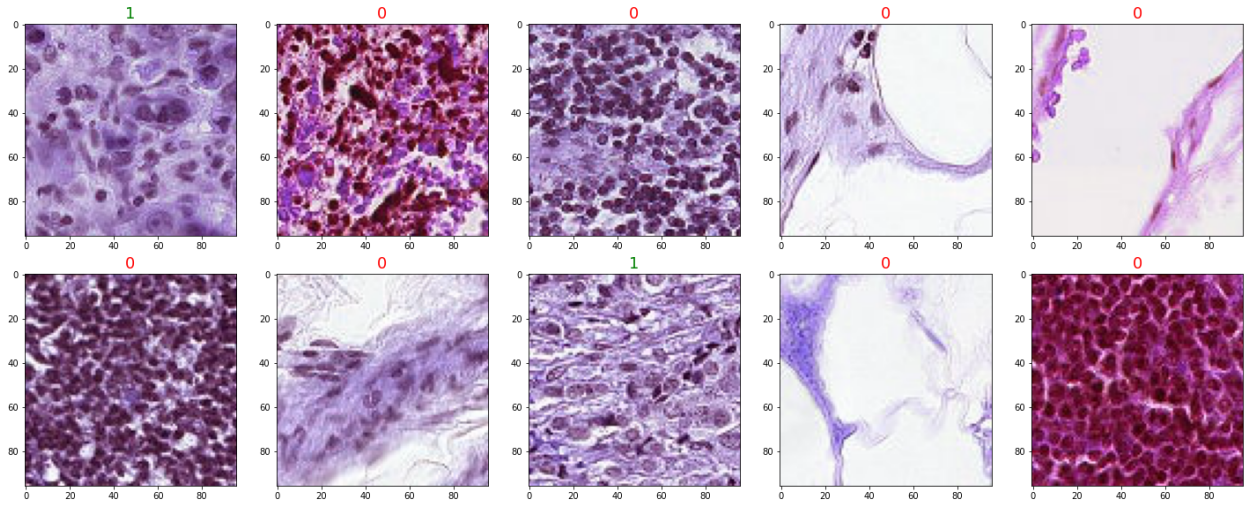
PCam packs the clinically-relevant task of metastasis detection into a straight-forward binary image classification task, akin to CIFAR-10 and MNIST. Models can easily be trained on a single GPU in a couple hours, and achieve competitive scores in the Camelyon16 tasks of tumor detection and whole-slide image diagnosis. Furthermore, the balance between task-difficulty and tractability makes it a prime suspect for fundamental machine learning research on topics as active learning, model uncertainty, and explainability.

The data has 2 folders of training and testing images and a file of training labels.

	label
count	220025.000000
mean	0.405031
std	0.490899
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

The above Image is the description of Dataset

Sample Data Representation



Data Preprocessing:

Splitting the Data into and training and validation, creating a custom data frame which has the image data, the whole folder has image, we can use the image location for the training and testing the model

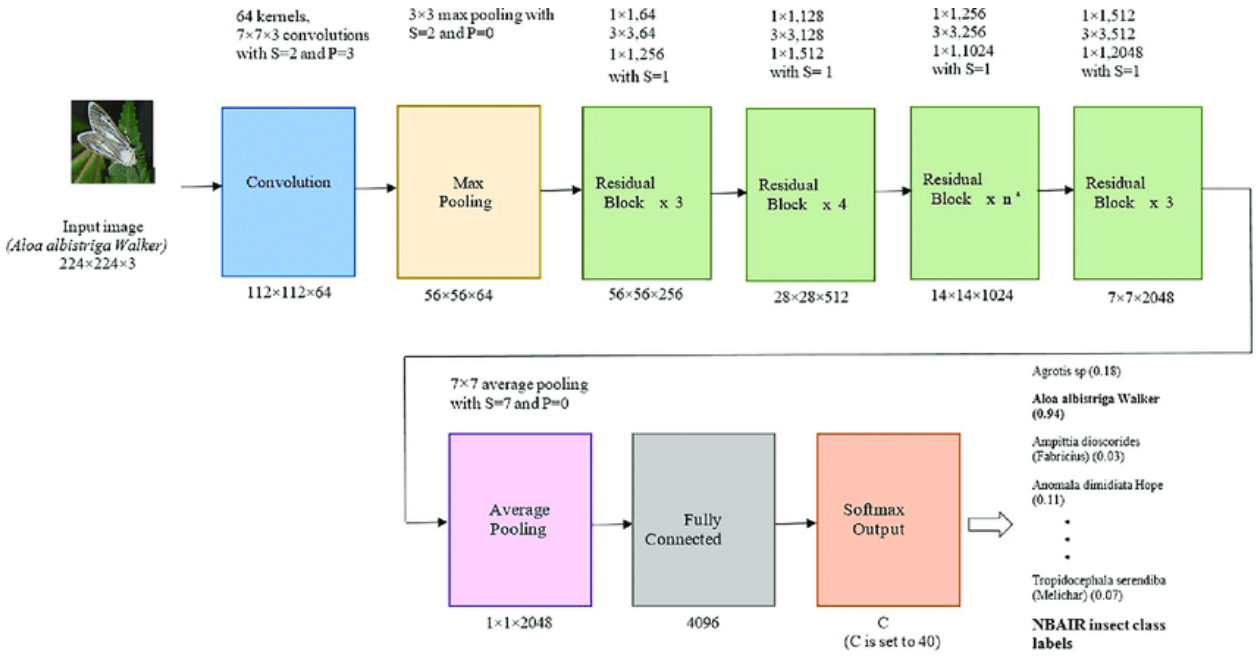
Using keras image processing module we are generalizing the image

In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better. In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class

Model:

We are using Resnet50 as Model here are we doing Transfer learning,

It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved. This is the fundamental concept of ResNet. ResNet50 is a 50 **layer Residual Network**.

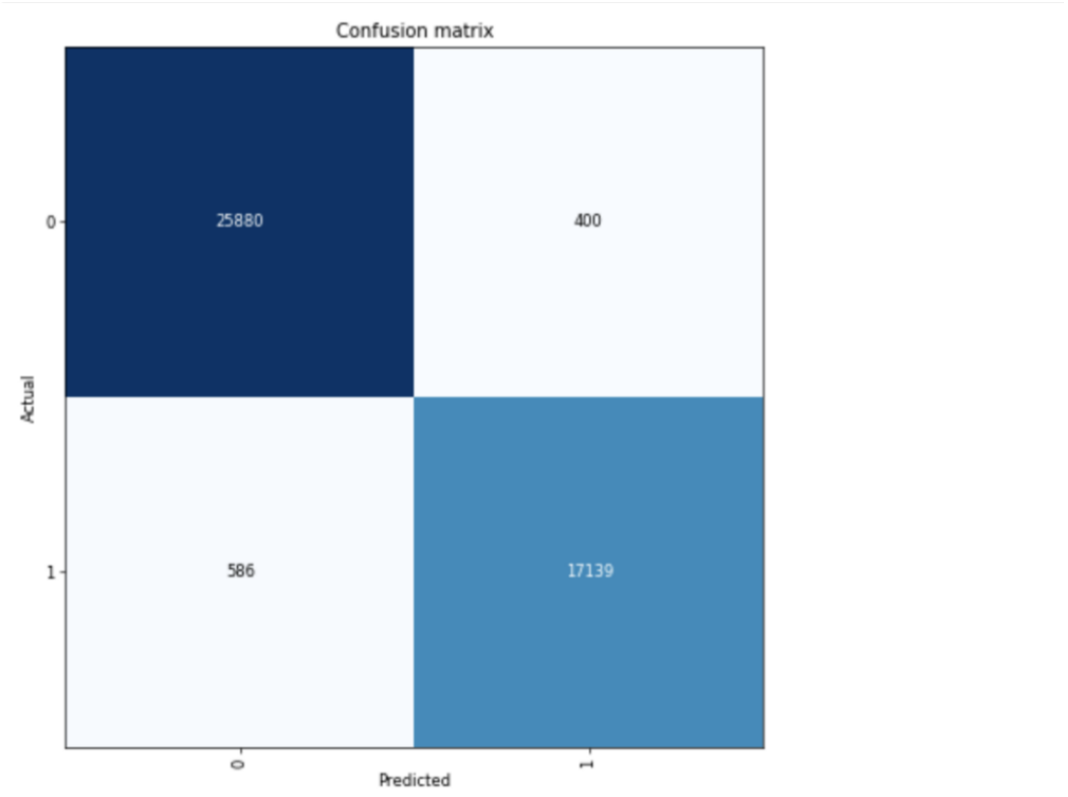


- A convolution with a kernel size of 7 * 7 and 64 different kernels all with a stride of size 2 giving us **1 layer**.
- Next we see max pooling with also a stride size of 2.
- In the next convolution there is a 1 * 1,64 kernel following this a 3 * 3,64 kernel and at last a 1 * 1,256 kernel, These three layers are repeated in total 3 time so giving us **9 layers** in this step.
- Next we see kernel of 1 * 1,128 after that a kernel of 3 * 3,128 and at last a kernel of 1 * 1,512 this step was repeated 4 time so giving us **12 layers** in this step.
- After that there is a kernal of 1 * 1,256 and two more kernels with 3 * 3,256 and 1 * 1,1024 and this is repeated 6 time giving us a total of **18 layers**.
- And then again a 1 * 1,512 kernel with two more of 3 * 3,512 and 1 * 1,2048 and this was repeated 3 times giving us a total of **9 layers**.
- After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function so this gives us **1 layer**.

Analysis:

Analysing the graph of the initial training run, we can see that the training loss and validation loss both steadily decrease and begin to converge while the training progresses.

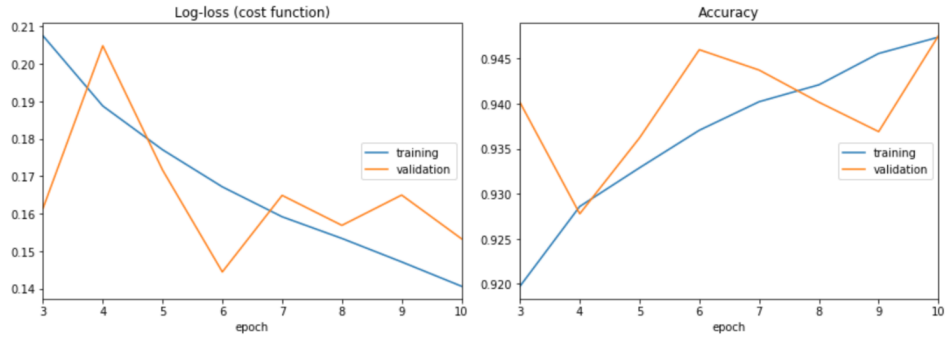
Accuracy at the moment is 94.76%.



The confusion matrix is a handy tool to help us obtain more detail on the effectiveness of the training so far. Specifically, we get some clarity on the amount of false positives and false negatives predicted by our neural net.

Plotting our top losses allows us to examine specific images in more detail. We can generates a heatmap of images that we predicted incorrectly. The heatmap allows us to examine areas of images which confused our network. Its useful to do this so we obtain better context around how our model is behaving on each test run, and direct us to clues as to how to improve it.

Result:




```
Log-loss (cost function):
training   (min:    0.141, max:    0.307, cur:    0.141)
validation (min:    0.144, max:    0.217, cur:    0.153)

Accuracy:
training   (min:    0.873, max:    0.947, cur:    0.947)
validation (min:    0.919, max:    0.948, cur:    0.948)

Epoch 00010: val_acc improved from 0.94600 to 0.94755, saving model to model.h5
```

The accuracy of model is around 94 on training and testing for 10 epoch

Conclusion:

In the final fine-tuning training run, we can see that our training loss and validation loss begin to diverge from each other now mid training, and that the training loss is progressively improving at a much faster rate than validation loss, steadily decreasing until stabilising to a steady range of values in the final epochs of the run.

Any further increases in our validation loss, in the presence of a continually decreasing training loss, would result in overfitting, failing to generalise well to new examples.

Finalising the at this point in our training yields a fine-tuned accuracy of 94.6% over our stage 1 training run result