Machine Learning Engineer Nanodegree

Capstone Project

Chenxin Wang
May 4th, 2019

I. Definition

Project Overview

Histopathologic cancer detection has been performed by human pathologists reviewing stained specimen on slide glasses with microscopes. This process is time-consuming and error-prone. Recent advances in slide imaging facilitate image analysis with machine learning algorithms to assist diagnostic tasks. Among these machine learning algorithms, deep convolutional neural networks (CNNs) have shown great prospect in the cancer detection [1, 2].

To advance computer-aided algorithms on cancer detection, Kaggle set up a competition [3] with the PatchCamelyon dataset, which includes hundreds of thousands of histopathological images. I take this competition as my project and develop a deep learning algorithm to detect tumors in histopathological images.

Problem Statement

This is a binary image classification problem where we identify the presence of tumors in histopathological images. The strategy to attack this problem is in the following:

- Explore the training dataset, understanding the distribution of training labels
- Implement data augmentation techniques like image rotation and color perturbation to improve the dataset
- Train a convolutional neural network from scratch
- Transfer learning with ResNet50 and DenseNet169
- Finetuning

The final model predicts the tumor probability given a histopathological image.

Metrics

In this Kaggle competition, solution models are evaluated on area under the ROC curve between the predicted probability and the observed target on the test set. The ROC curve is a plot of True Positive Rate against False Positive Rate as the threshold probability dividing the positive and negative varies. The area under the ROC curve (AUC-ROC) represents the performance of a solution model. An excellent model has the area under the ROC curve close to 1. When the area under the ROC curve is 0.5, the model has no classification capability. When the area under the ROC curve is close to 0, the model predicts the opposite of the ground truth.

This metric is suitable for this binary image classification problem. A trained algorithm predicts the tumor probability in histopathological images. As the threshold probability dividing the positive and negative changes, the predicted label for a certain image may change. The area under the ROC curve, however, is independent of the threshold probability.

II. Analysis

Data Exploration

The implementation for the following data exploration is in "exploratory_data_analysis.ipynb" notebook.

1. Sample images

The training and test images are of size 96x96px. Each of the training images is labeled according to whether there is at least one pixel of tumor tissue in the center region (32x32px). Each of the boxes in Fig. 1 denotes the center region. Tumor tissue outside this center region does not influence the label. Therefore, a negatively labeled image may contain tumor pixels in the outer region. This may make our training hard.

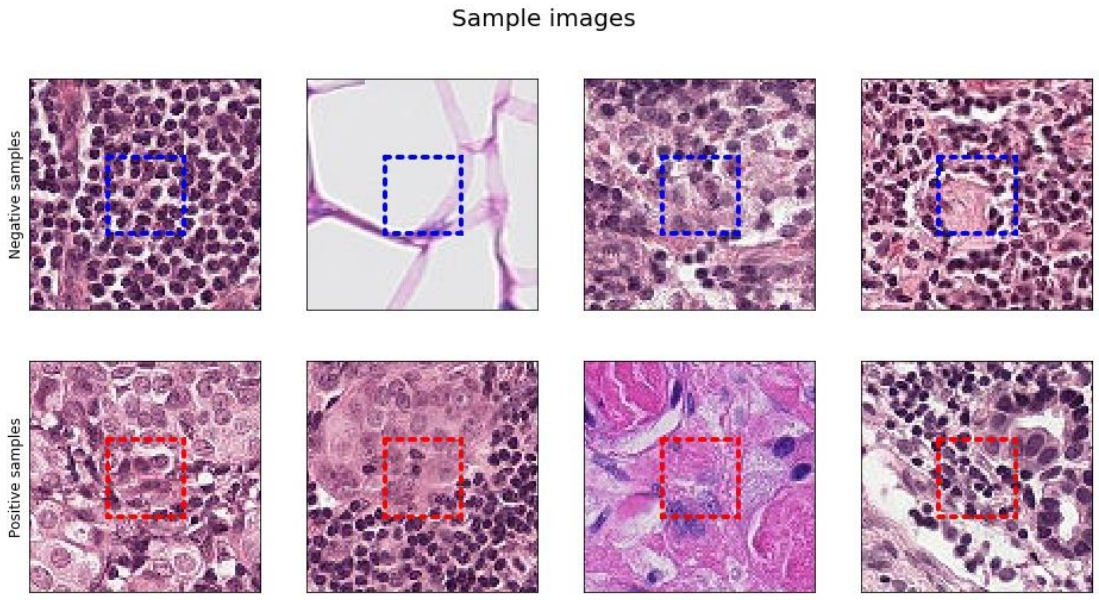

Figure 1: Negative (first row) and positive (second row) samples

From these samples shown in Fig. 1, one can tell it is hard to distinguish a positive sample from a negative one for an untrained person.

2. **Data statistics**

Figure 2 shows the data distribution for the training set. It has about 130k negatively and 90k positively labeled images. The ratio is closer to 60/40 meaning that there are 1.5 more negatives than positives. The mean pixel value (RGB) of the training images (normalized to 0∼1) is [0.7024, 0.5462, 0.6965], and the standard deviation is [0.2389, 0.2821, 0.2163].

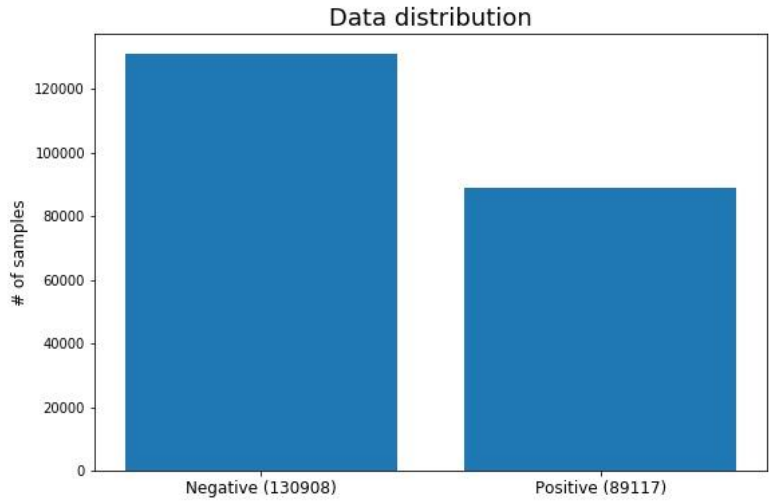

Figure 2: Data distribution

3. **Outliers**

There is one extremely dark image with all pixel values less than 10 (Fig. 3) and six extremely bright images with all pixel values greater than 245 (Fig. 4). These images are all labelled negative. There may be no tissue in these images.

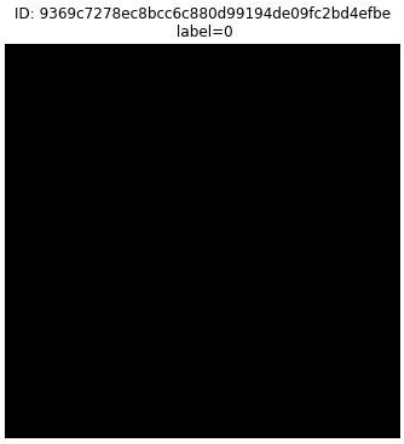

Figure 3: Extremely dark image

Figure 4: Extremely bright images

Exploratory Visualization

Due to limited RAM, a subset of training images (20k samples) is used for the exploratory visualization. Figure 5 shows the mean image brightness distribution for positive and negative samples, separately. For the positive samples, the distribution looks like a normal distribution around a brightness of 150, while the distribution for the negative samples looks like a bimodal distribution with peaks around 135 and 225. This difference may be implicitly used in our following deep learning model to identify positive and negative images.

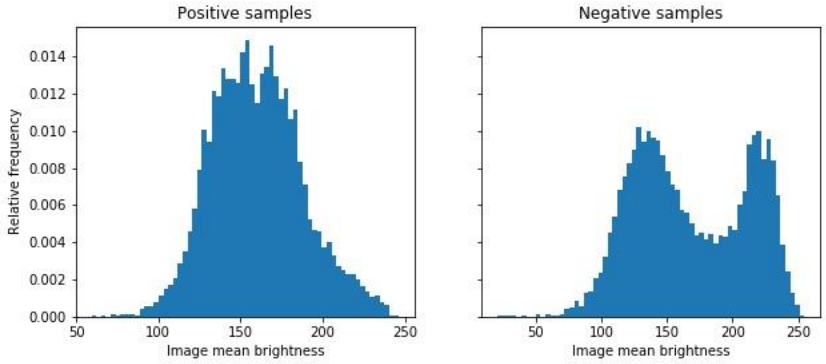

Figure 5: Image mean brightness distribution for positive and negative samples

Algorithms and Techniques

Convolutional neural networks are well suited for this binary image classification problem. A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. For this problem, the input layer handles the input image, and the output layer uses a sigmoid function to give the tumor probability of the input image. The hidden layers consist of a cascade of convolutional layers, pooling layers, fully connected layers and dropout layers. The convolutional and full connected layers identify features from the input images and do the classification. The pooling and dropout layers combat overfitting.

An example illustrates how a convolutional layer works. Consider a 5x5 image in Fig. 6 whose pixel values are only 0 and 1 and a 3x3 kernel matrix (or filter) in Fig. 7. Figure 8 shows the convolution of the 5x5 image and 3x3 kernel matrix. We stride the 3x3 kernel matrix over the image by 1 pixel per step. For each region covered by the kernel matrix, we multiply each element in the 3x3 kernel matrix with the corresponding pixels in that region and add the multiplication results to get a single element in the convolved feature (Fig. 8). One convolutional layer could only identify simple features like lines and edges. A deep convolutional neural network includes multiple convolutional layers to identify complex features like tumor tissues.

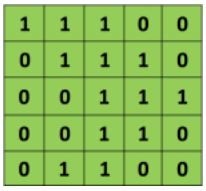

Figure 6: A 5x5 image with pixel values only 0 and 1. Source [4]

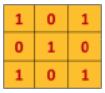

Figure 7: A 3x3 kernel matrix. Source [4]

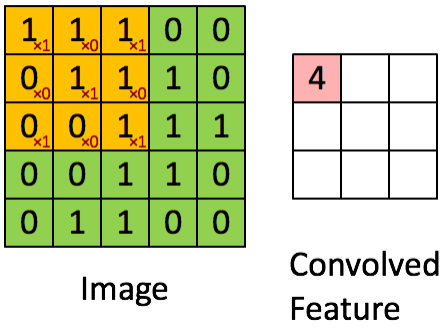

Figure 8: Convolution of a 5x5 image and a 3x3 matrix. Source [4]

Figure 9 shows a maxpooling operation on a 4x4 image. A 2x2 filter runs over the image with a stride of 2. For each region covered by the filter, we will take the max of that region and create a new, output matrix (Fig. 9).

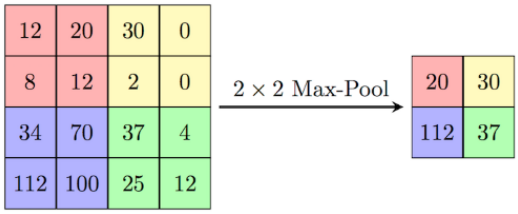

Figure 9: Maxpooling a 4x4 image with 2x2 filters and a stride of 2. Source [5]

Passing the entire dataset to the neural network for training is not feasible. We need to divide our dataset into numbers of batches. Training the neural network with only one epoch may not be enough. We need a number of epochs to combat underfitting. If the number of epochs is too large, we may overfit the data. An optimal number of epochs can be identified when the validation loss stops improving. Learning rate is another parameter that affects training. It controls how much our model learns from a training batch. If the learning rate is low, the training is more reliable, but it takes a lot of time for training. If the learning rate is high, the training may not converge.

Benchmark

The benchmark model is a trained CNN from the Kaggle kernel: https://www.kaggle.com/gomezp/complete-beginner-s-guide-eda-keras-lb-0-93. Kaggle splits the test data set into public and private. Running the benchmark kernel gives an AUC-ROC score of 0.8749 for the public test set and 0.8890 for the private test set.

III. Methodology

Data Preprocessing

Training a convolutional neural network for the image classification does not require any manual feature selection or feature transformation. The neural network itself identifies features for classification. Instead, we do data augmentations to improve the training dataset. The techniques used in this work are random rotation, random translation, random brightness, random contrast, and flip images (horizontally and vertically).

There are seven outliers out of the 220k training images. I tried training with and without these seven outliers. I will report the training with outliers first, and discuss the influence of these outliers later.

The training dataset is split into a training set (80%) and a validation set (20%).

Implementation

Three different deep learning models are implemented. The first one is a CNN from scratch. The implementation is in the "cnn_from_scratch.ipynb" notebook. The CNN has five convolutional layers, one fully-connected layer and one output layer. Each convolutional layer is followed by a batch normalization layer, a relu activation layer, and a pooling layer. The fully-connected layer is followed by a batch normalization layer, a relu activation layer, and a dropout layer. The final output layer uses a sigmoid activation to give the tumor probability. The CNN architecture is in the "cnn_from_scratch_arch.png" file.

Since we have a large number of images, importing all of them into RAM at once is memory-inefficient. Instead, we implement a python generator which allows us to pull pieces of the data and process them on the fly only when needing them.

The second and third models are transfer learning of the Resnet-50 and Densenet-169 architectures, respectively. The implementations of the second and third algorithms are in the "resnet50_tl.ipynb" and "densenet169_tl.ipynb" notebooks, respectively. The FastAI library (https://docs.fast.ai/) is used, and it makes the coding much easier. For example, we can specify a transfer learning model with just one line of code as below.

```
learner = cnn_learner(data, resnet50, pretrained = True, path='.',
                      metrics = [accuracy, auc_score], ps=0.5)
```

The training is easier as well. We do not need to randomly specify a learning rate. Instead, the FastAI library has a function "lr_find", which helps us identify a proper learning rate. Figure 10 shows a plot of the losses against learning rates, which is an output from the FastAI "lr_find" function. The loss reaches a minimum at a learning rate around 7e-2. We pick a learning rate of 2e-2, which is a bit before the minimum.

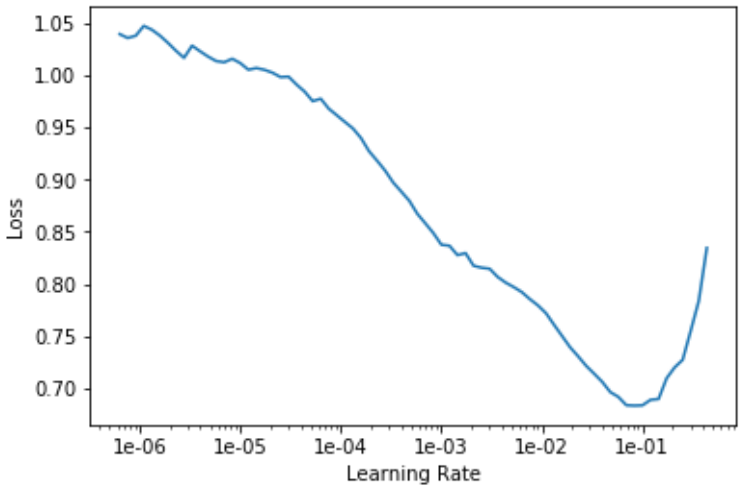

Figure 10: Losses for a range of learning rates. This result is from the FastAI "lr_find" function.

We could train the model with just one line of code as below

```
learner.fit_one_cycle(8, 2e-2)
```

This means 8 epochs are specified for training, and the maximum learning rate is 2e-2. Figure 11 shows the learning rate in a training process. When starting training, a lower learning rate is used to warm up. Then the learning rate goes up to the maximum 2e-2. The high learning rates prevent the model to land in a local minimum. High learning rates, however, may make the validation loss spike (Fig. 12). The learning rates descend to a small value at the end of the training to smooth the validation loss.

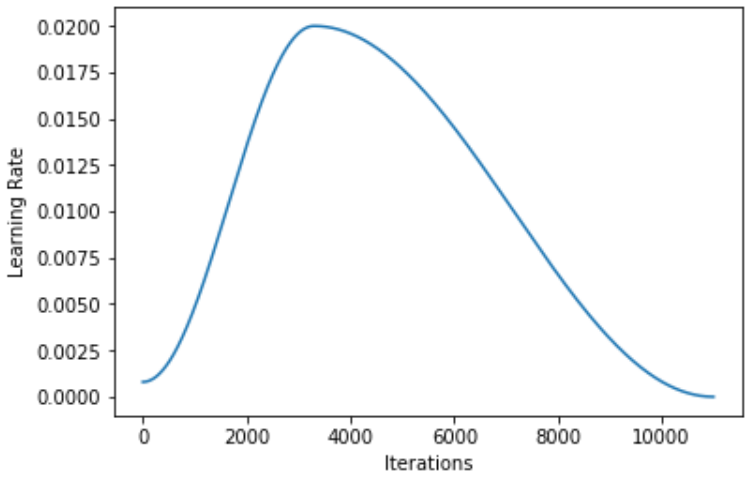

Figure 11: Learning rates in a training process

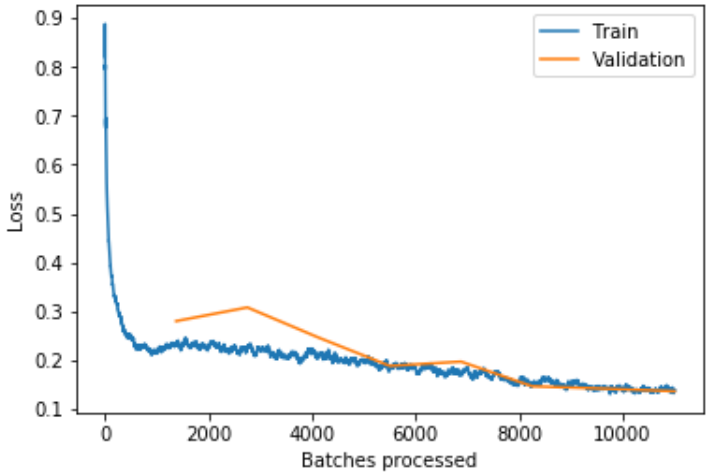

Figure 12: Train and validation loss in a training process

FastAI allows test time augmentation when predicting the labels of new datasets with one line of code as below

```
preds_test,y_test=learner.TTA(ds_type=DatasetType.Test)
```

The data augmentation for training can give a better model. The test time augmentation is believed to improve prediction accuracy because it predicts the tumor probability for the original test image along with several random transforms of the same image and take an average of these predictions to determine the final tumor probability.

Refinement

The training and validation AUC-ROC scores for the CNN model from scratch are 0.9781 and 0.9738, respectively. Kaggle splits the test data set into public and private. This model gives a score of 0.9503 for the public test set and 0.8981 for the private test set.

The Resnet50 transfer learning model gives an AUC-ROC score of 0.9885 for the validation set. The score for the public and private test sets are 0.9649 and 0.9572, respectively.

The Densenet169 transfer learning model has a validation AUC-ROC score of 0.9904. The score for the public and private test sets are 0.9617 and 0.9628, respectively.

Since the CNN model from scratch gives lower test scores than the two transfer learning models, I only refine the two transfer learning models to get higher test scores.

In our first training of the Resnet50 and Densenet169 models, we train only the heads while keeping the rest of the models frozen. After the heads are trained, the models already perform well on the test dataset, as shown above. To refine the models, we unfreeze all the trainable parameters and continue its training. The learning rate in the finetuning is much smaller than that for the first training because we do not want to mess up with the well-trained weights in our first training.

After finetuning, the Resnet50 transfer learning model achieves a higher AUC-ROC score of 0.9930 for the validation set than before, but it gives lower scores of 0.9636 for the public test set and 0.9551 for the private test set.

The Densenet169 transfer learning model, after finetuning, achieves a higher AUC-ROC score of 0.9930 for the validation set than before. Its public test score has a tiny increase from 0.9617 to 0.9621, while its private test score decreases from 0.9628 to 0.9562.

The finetuning does not improve our test scores for either of the Resnet50 and Densenet169 transfer learning models because they already have had a high performance before the finetuning.

Influence of Outliers

The implementations for the CNN model from scratch, Resnet50, and Densenet169 without outliers are in “cnn_from_scratch_without_outliers.ipynb”, “resnet50_tl_without_outliers.ipynb” and “densenet169_tl_without_outliers.ipynb” notebooks, respectively. The final results for the three models with and without outliers are shown in Table 1. The CNN model from scratch has a poorer performance when the outliers are removed, while the two transfer learning models are

insensitive to the outliers. The pre-trained transfer learning models may have learned the features in the outliers, but the model from scratch are not able to learn them when the outliers are removed. In what follows, I only discuss the models trained with outliers included.

Table 1: Final AUC-ROC scores for the three different models with and without outliers

		Validation score	Public test score	Private test score
CNN model from scratch	With outliers	0.9738	0.9503	0.8981
	Without outliers	0.9004	0.8319	0.8964
Resnet50	With outliers	0.9930	0.9636	0.9551
	Without outliers	0.9927	0.9680	0.9487
Densenet169	With outliers	0.9930	0.9621	0.9562
	Without outliers	0.9926	0.9640	0.9594

Results

Model Evaluation and Validation

Both of the Resnet50 and Densenet169 models (before and after finetuning) perform better than the CNN model from scratch on the public and private test sets. The Densenet169 model before the finetuning has the highest score for the private test set, which is harder than the public test set. We use it as our final model.

Deep learning models act like a black box. It is hard to evaluate its parameters or weights. Instead, we evaluate our final deep learning model by its performance on new datasets. The public and private test sets are totally unseen by the final model. The final model gives an AUC-ROC score of 0.9617 for the public test set and 0.9628 for the private test set, indicating that the model generates well to unseen data.

The final model is robust for small perturbations in training data. In the data preprocessing, we do data augmentation, which takes image perturbations into account. Therefore, the final model should be insensitive to image perturbations.

The final model has a high performance, but it is not perfect. It makes mistakes on a small number of validation images, as the confusion matrix in Fig. 13 shows. Therefore, one cannot fully trust the model. Instead, this model can be used to assist a human pathologist for tumor detection.

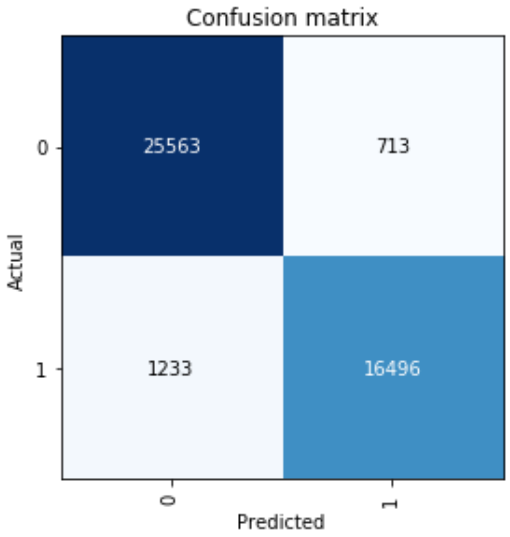

Figure 13: Confusion matrix for the final model on the validation set

Justification

The benchmark model gives an AUC-ROC score of 0.8749 for the public test set and 0.8890 for the private test set. Our final model results in a public score of 0.9617 and a private score of 0.9628. In addition, the benchmark model gives a validation accuracy of 0.8226, while the validation accuracy for our final model is 0.9593. It is obvious that our final results are stronger than the benchmark results.

A plot of the ROC curve for our final model on the validation set is shown in Fig. 14. The area under the ROC curve is close to 1, indicating a high performance of the model on the validation set.

The final model has a high performance on the unseen test sets, indicating that it generates well to new data. Therefore, our final solution has solved the problem.

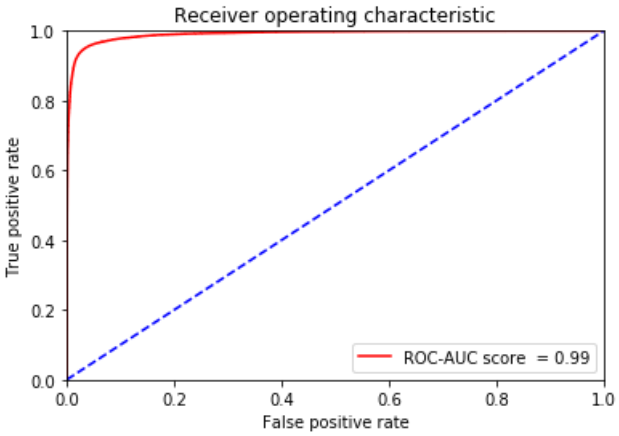

Figure 14: ROC curve for the final model on the validation set

IV. Conclusion

Free-Form Visualization

Figure 15 shows the most incorrect and correct samples from the validation set for the Densenet169 model after finetuning. The most incorrect samples help us understand what the model is struggling with. The most correct samples help us understand what the model is good at. If a human pathologist has a good performance on the most incorrect samples but has a poor performance on the most correct samples, the pathologist can make an improvement on tumor detection with the assistance of the model.

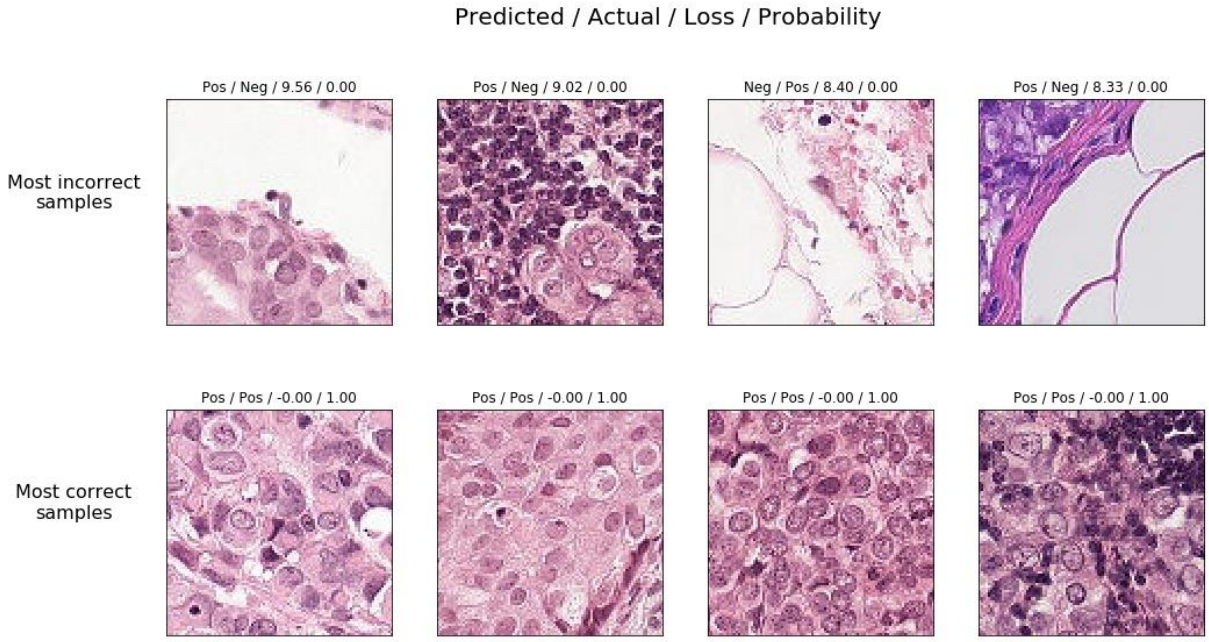

Figure 15: The most incorrect and correct samples from the validation set for the Densenet169 model after finetuning

Reflection

In this project, I explore the training dataset at first, which includes looking at sample images, understanding the distribution of training labels, identifying outliers, and visualizing features of the training data. Second, I do data augmentations such as random rotation, random translation, random brightness, random contrast, and flip images (horizontally and vertically) to improve the training dataset. Third, I train three deep learning models including one CNN model from scratch and two transfer learning models (i.e., Resnet50 and Densenet169). Fourth, I finetune the two transfer learning models. At last, I examined the influence of the outliers on the performance of the three models.

I use FastAI library for the two transfer learning models. The FastAI library makes the programing and training much easier. I would use it in other deep learning projects. I had a difficulty with the CNN model from scratch. In the training, I used "ndimage.imread" from scipy library to read the training images in RGB format. In the prediction of the test set, I used "cv2.imread" to read the test images in BGR format. This gave very low scores for the public and private test sets. I spent a lot of time identifying this mistake.

The final model performs well and fits my expectation for the problem. I would recommend transfer learning instead of training a CNN from scratch to solve these types of problems.

Improvement

I use a fixed learning rate for the training of the CNN model from scratch. A better approach is to specify a varying learning rate along the training as the one implemented by FastAI (Fig. 11).

I would like to plot a heatmap to identify which part of an image the final model uses to make a prediction. I tried "plot_top_losses" with heatmap = True, but it did not work.

Other transfer learning models such as Inception-v3 and VGG-19 may give better solutions than our final Densenet169 model. An ensemble of different models may result in higher AUC-ROC scores on the public and private test sets.

Acknowledgement

The following two Kaggle kernels helps me a lot on this project.

1. https://www.kaggle.com/qitvision/a-complete-ml-pipeline-fast-ai
2. https://www.kaggle.com/gomezp/complete-beginner-s-guide-eda-keras-lb-0-93

Reference

[1] Wang, D., et al.: Deep learning for identifying metastatic breast cancer. (2016)

[2] Liu, Y., et al.: Detecting cancer metastases on gigapixel pathology images. (2017)

[3] Kaggle competition. https://www.kaggle.com/c/histopathologic-cancer-detection.

[4] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[5] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling