Machine Learning Engineer Nanodegree

Capstone Project

Adrian Voicu
November 11, 2020

Digital Pathology - Mitosis Detection using Transfer Learning in PyTorch

Definition

Project Overview

Mitosis Detection is a task in the field of Digital Pathology. Its application is within diagnosing breast cancer and its development stage. The problem tackled in this project is automating the diagnosing task and it was inspired by the TUPAC2016 challenge[1] and the papers published by some of the participant teams.[2]
Another great source of help and inspiration was Andrew Janowczyk's blog[3], which helped me set a more clear path in proceeding with my task and also provided a complete dataset[4] that I used in training and evaluating my model.
A big part of the task is constituted by the pre-processing of the images found in the dataset. Since the original images are high resolution, thus very large, the actual datapoints that are used in training the model are small patches extracted from these original images, which contain cellular nuclei (with or without mitotic centers). Also provided in the dataset, are the so called Blue Ratio Segmentation masks, for each individual image – these are used in extracting the negative class patches.

Problem Statement

The goal of the project is training a model that can predict whether a sample image contains a mitotic center or not. Thus, the problem comes down to **image binary classification**. The platform used for implementing the solution is PyTorch and the programming language is Python. I chose a pre-trained AlexNet model for working on this problem.

The steps taken are as follows:
- download the Mitosis Detection dataset
- data pre-processing: extract positive and negative class patches from the original images and separate a training dataset and a testing one
- handle class imbalance: augment size of positive class by using image rotation
- eliminate positive centers that are found on the image's margins, since they cannot be extracted into a patch
- configure the pre-trained AlexNet model, to be suitable for a binary classification problem
- train the model using the training dataset

1 Tumor Proliferation Assessment Challenge 2016. [http://tupac.tue-image.nl]
2 [https://www.researchgate.net/publication/333633080_Transfer_learning_based_deep_CNN_for_segmentation_and_detection_of_mitoses_in_breast_cancer_histopathological_images]
3 [http://www.andrewjanowczyk.com]
4 [http://www.andrewjanowczyk.com/deep-learning/]

- handle fragmented training by saving and loading the model
- test model on test dataset

Metrics

The metric used for evaluating the performance of the model during training is accuracy:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ no.\ of\ samples}$$

However, since the dataset is not well balanced (most of the patches contain no mitotic centers), observing precision is also useful:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Intuitively, precision is the ability of the model not to label as positive a sample that is negative[5].

Furthermore, considering that of higher importance the need to find *all* positive samples, recall will also be measured and used as an evaluation metric for the model:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Intuitively, recall is the ability of the model to find all positive samples[6]. Having in mind the context of the prediction problem, this seems to be the best metric for evaluation.

F1 score is also evaluated, as a weighted average of precision and recall.

5 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
6 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score

Analysis

Data Exploration

The Mitosis Detection Dataset used for the project consists of the publicly available set of images found on Andrew Janowczyk's blog[7] which includes a list of 311 images (.tif format), with 550 mitotic centers expertly annotated (coordinates of the centers included in the .csv file). These are high resolution scans of microscopic images containing biopsy samples from different patients.
An example of an original image, containing a original biopsy scan (ie. without annotations or filters) can be seen in Figure 1.

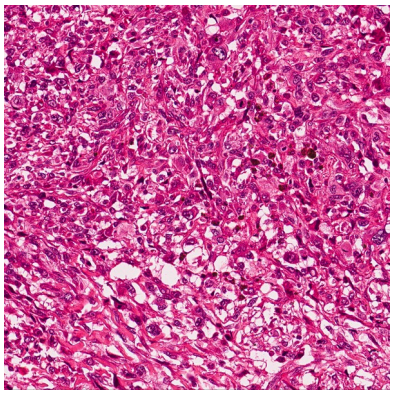

Figure 1: Original Image

Attached to each .tif image, the dataset provides 3 additional helper images:

- first, containing the Blue Ratio Segmentation[8] (BRS) mask of the original (*_brmask.png) as can be seen in Figure 2. This image will not be actually used in the data processing, but it's displayed for better understanding of the concept.

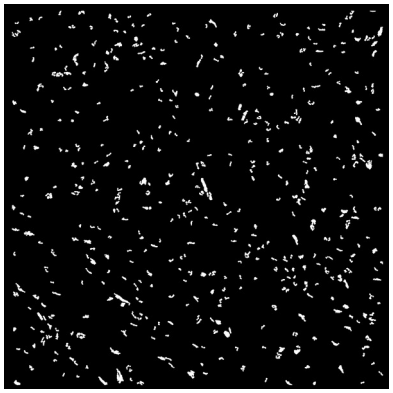

Figure 2: Blue Ratio Segmented Image

7 Mitosis Detection Dataset - http://www.andrewjanowczyk.com/deep-learning/

8 Using the BRS technique is useful to isolate the patches where mitotic centers are most likely to be found. This eliminates the patches that the classifier can easily predict as negative, leaving for training the ones that are more dificult to identify. The BR centers are dilated, in order to include the centers context in the processing. For more details on BRS technique, see [6]

- the second, containing the dilated Blue Ratio Segmentation (BRS) mask of the original (*_cmask.png
– further referenced as “type 1” input image) as can be seen in Figure 3. This image will be used in data processing for extracting **negative** patches containing potential mitotic centers that are harder to classify.

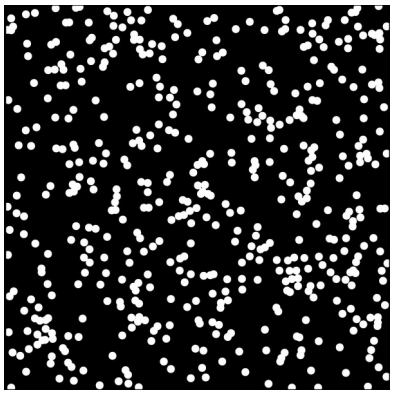

Figure 3: Dilated Blue Ratio Segmented Image

- the third, containing the anotated mitotic circles with a 4 pixel radius drawn around them (*_pc.png – further referenced as “type 2” input image) as can be seen in Figure 4. This type of image will be used in data processing for extracting the **positive** patches.

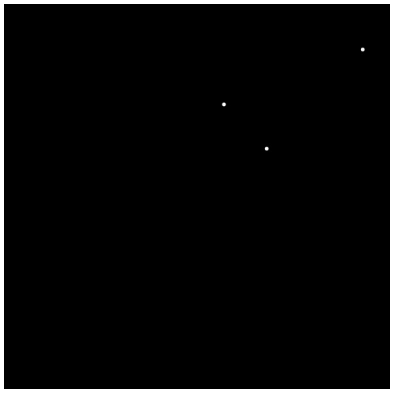

Figure 4: Annotated mitotic centers Image

Algorithms and Techniques

Data preprocessing is a major part of the implementation. Having to deal with high resolution images that contain raw data (non annotated scans in RGB), one challenge was to break down each original image into small patches that will constitute the training and testing dataset.

For dealing with this binary image classification problem, a pretrained AlexNet model was chosen.

Transfer learning as opposed to learning “from scratch” was used for 2 major reasons:
- firstly, it reduces training time and comes with a smaller resource expenditure
- secondly, it was shown that detection rates are improved[9], when compared to a non trained model.

Benchmark

Due to having used Andrew Janowczyk’s work as my main source of inspiration, the results provided by his model will be used as a benchmark for my attempt. However, I must acknowledge the fact that my implementation is a simplified one, that omits some important steps. Hence an F-score of 0.50 will be taken as reference.
Another source of inspiration came from one of the winning teams of the TUPAC2016 challenge. In their paper[10], they present a pre-trained AlexNet model which achieved a Precision score of 0.57, Recall score of 0.533 and F-score of 0.551. This will also be taken as a reference in evaluating my classifier.

Methodology

Data Preprocessing

The preprocessing stage is done in the MakePatches notebook. The following steps are applied:

- The original image, the type 1 image (dilated BRS mask) and type 2 image (mask containing annotated centers in BRS) are resized to 1000x1000 px (from an initial size of 2000x2000); this reduction is made to ensure that the entire mitotic center will be contained in a patch.

- For the positive class:
  - Patches are extracted according to the positions of each annotated center found in the type 2 image.
  - Addressing class imbalance (the negative samples are a lot more common than positive samples) the extracted positive patches are rotated according to a rotation array (eg. {0º, 45º, 90º}).
  - example of a positive patch, with 3 applied rotations:

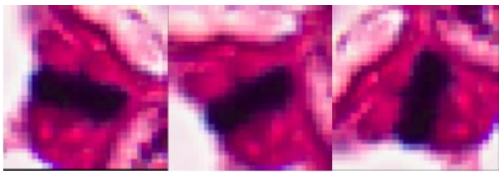

Figure 5: Positive patch: 0, 45, 90 deg rotation

- For the negative class:
  - Patches are extracted according to the coordinates of each center found in the type 1 image; these are considered centers that are more likely to be incorrectly classified, thus there is a

9 https://www.researchgate.net/publication/333633080_Transfer_learning_based_deep_CNN_for_segmentation_and_detection_of_mitoses_in_breast_cancer_histopathological_images
10 Idem

higher added value when using them for training the model, as compared to using random patches from the original image.

- For the purpose of dataset augmentation, a rotation is also applied to each extracted patch, but the rotation array has a smaller dimension (eg. {0º, 45º}).
- the centers found at the edge of the image are discarded, in order to prevent incomplete patches to be generated
- example of different negative patches:

Figure 6: Negative patches

- All the extracted patches, both negative and positive, are saved on a local drive and labeled according to the following naming convention:

- **u_v_w_x**.png, where:
  - **u** represents the original image name
  - **v** represents the sample class (0 or 1)
  - **w** represents the center number
  - **x** represents the rotation applied
  - eg. 01_02_class0_center0_rot0.png

- The generated patch labels are stored in a dictionary, together with the assigned class

- The dataset was separated (manually) into 3 parts: train, validation and test and stored to separate folders on the drive

- The sizes of the resulting datasets:
  - train set: 151293 samples
  - validation set: 35876 samples
  - test set: 35659 samples

The ***extract_patches*** function that executes the patch extraction operations is found and documented in the helper_functions.py file.
For loading the images as well as visualization tasks, the **Open CV** library was used.
For resizing the images, **imutils** library was used. For plotting and graphs, **matplotlib** was used.

Implementation

For implementing the model, the Pytorch framework was chosen. Within it, the **torchvision** module was used for handling dataset operations. One particular coding challenge was understanding the

specific data types and methods used in the Pytorch environment and adapting the code to properly work with these concepts (eg. Tensor, DataLoader).

The following steps were implemented:

- A series of transformations and conversions (resizing, transforming to tensors, normalization) were applied to the set of extracted patches. This was needed in order to ensure compatibility with the Pytorch implementation of the chosen model

- The folders containing the data were uploaded into DataLoader objects and their contents were shuffled

- The Pytorch implementation of an Alexnet pretrained model was loaded. The initial training was performed on the ImageNet dataset that contains over 15 million labeled high resolution images. By using a pretrained model, the classifier is able to detect main features in the images. That leaves the final part of the training to be done on relevant datapoints, which will allow the model to understand which classes it needs to predict.

- Since the AlexNet is pretrained on a dataset that contains around 22000 categories and our problem is a binary classification one, it is mandatory that the final layer of the model (the classification layer) is modified in order to output only 2 categories.

- The loss function and optimizer used for the model are Cross Entropy Loss and Adam. These are generally considered good starting points for decent results with AlexNet. The goal is to minimize the loss, ie. to reduce the measure of entropy between the predicted outputs and the ground truth. Entropy can be generally understood as the measure of uncertainty of a value.

- Due to the fact that no GPU is present and all operations are only running on CPU, the training times are expected to be very long. Thus, a method for saving a model checkpoint and resuming the training was implemented, using the Pytorch framework, which offers a straightforward solution for this problem (ie. torch.save and torch.load)

Results

Model Evaluation and Validation

During training, a validation set was used. The validation was carried out in each epoch, immediately after the training loop. Thus an epoch-scope loss and accuracy were calculated and stored for further evaluation and visualization.

At the end of the training and validation stage, a separate test dataset was used for evaluating the model using “fresh” samples. Based on these results, the evaluation metrics were computed at the end of the testing stage: Precision, Recall, F1 Score and Accuracy. For calculating these, the **scikit-learn** library was used, which provides methods for all the metric scores.

After training for 20 epochs, it was noticed that the loss stabilizes after approximately 10 epochs, as can be seen in the graphic below.

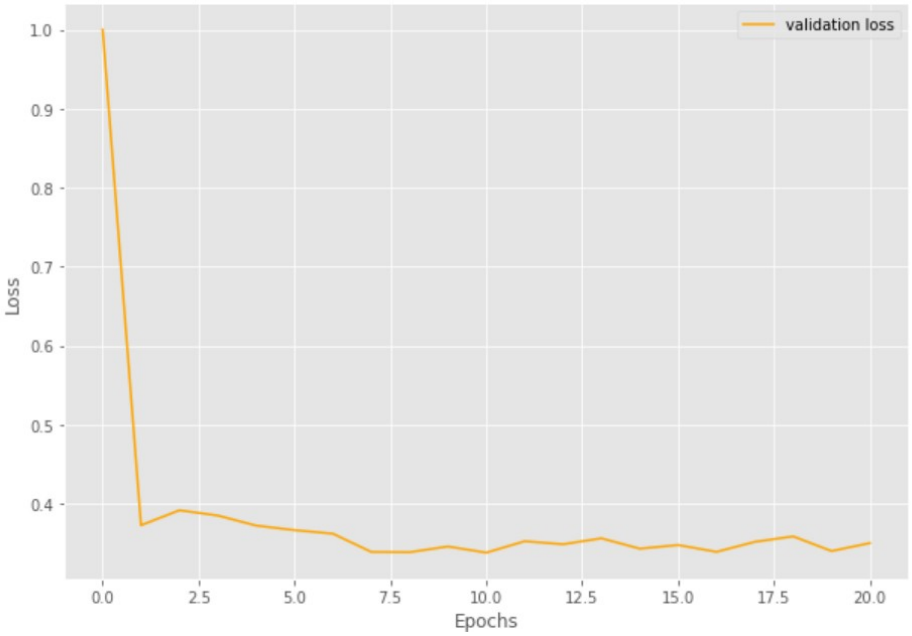

Figure 7: Plotting the validation loss during training

The final results obtained on the test dataset are the following:

- accuracy = 84.87 % - very good result, but not the best metric for our problem, since most of the samples are negative (thus simply predicting all samples as negative could lead to a high value of accuracy)
- precision = 90.3 %
- recall = 67.5% - this is a good evaluation metric, since the main focus of the problem is detecting the positive samples. The score achieved with default parameters is satisfactory.
- f1 score = 77.3 %

Justification

The overall results are relatively good and the training time (on CPU) was relatively short: approximately 8.5 hours. Hence, a strong case can be made for the efficiency of transfer learning, which seems to yield good results, even though the original training dataset (ImageNet) has no common features with the problem to be solved.

An interesting point is the comparison with the results obtained by a pretrained DenseNet model, but trained on only 3 epochs. According to the PyTorch implementation, the model was pretrained on the same ImageNet dataset. A separate notebook was used for this experiment – Train_Model_DenseNet, which contains same code as the original notebook, with small adaptations.
With significantly higher training times (around 6 hours per epoch), the model reaches a loss of 0.31 on the validation set after just 2 training epochs. The complete evaluation metrics for the pretrained DenseNet161 are:

- accuracy = 84 %
- precision = 89.6 %
- recall = 65.9 %

- f1 score = 75.9 %

Again, promising results, that are very similar to AlexNet, albeit with significantly longer training times.

Conclusion

Reflection

The problem of identifying mitotic centers in biopsy scans using Machine Learning seems to have high potential for assisting pathologists in setting patient diagnostics. As can be seen from my relatively simple and basic approach, good results are possible.

However, two main problems remain:

One is the lack of high volume data made available for training. An interesting idea[11] for solving this problem is providing incentives to clinics and patients for sharing their anonymized medical documents to AI researchers that could benefit them. Of course, a strong emphasis must be placed on the topic of data security and anonymity.

The second is the noise introduced in the labeling process, since this is a human activity which is highly prone to error. This makes the ground truth potentially unreliable and decreases the chances of an automated diagnosing process to have consistent results (since the labeling is not always consistent between different experts and between themselves at different times).

Improvement

There is certainly room for developing methods of improvement and expansion to the work described above. Some of these are:

- using a hybrid system: a model is used for selecting difficult to classify mitotic centers; then, these “tricky” samples would serve as negative samples for training the classifier. This would create more balanced classes and would refine the final results, as the prediction model would have to deal only with the most difficult cases.

- using a GPU or cloud based computation (eg. AWS), for decreasing training times and thus being able to test more model types or parameters – especially useful for training more epochs on DenseNet

- deploying the predictor to a mobile app (to potentially facilitate the usage in laboratory environments)

- integrate the **albumentations**[12] library for obtaining a better training dataset, higher performance and cleaner code

11 The Gradient- Why Skin Lesions are Peanuts and Brain Tumors Harder Nuts [https://thegradient.pub/why-skin-lesions-are-peanuts-and-brain-tumors-harder-nuts]

12 Albumentations - [https://github.com/albumentations-team/albumentations]