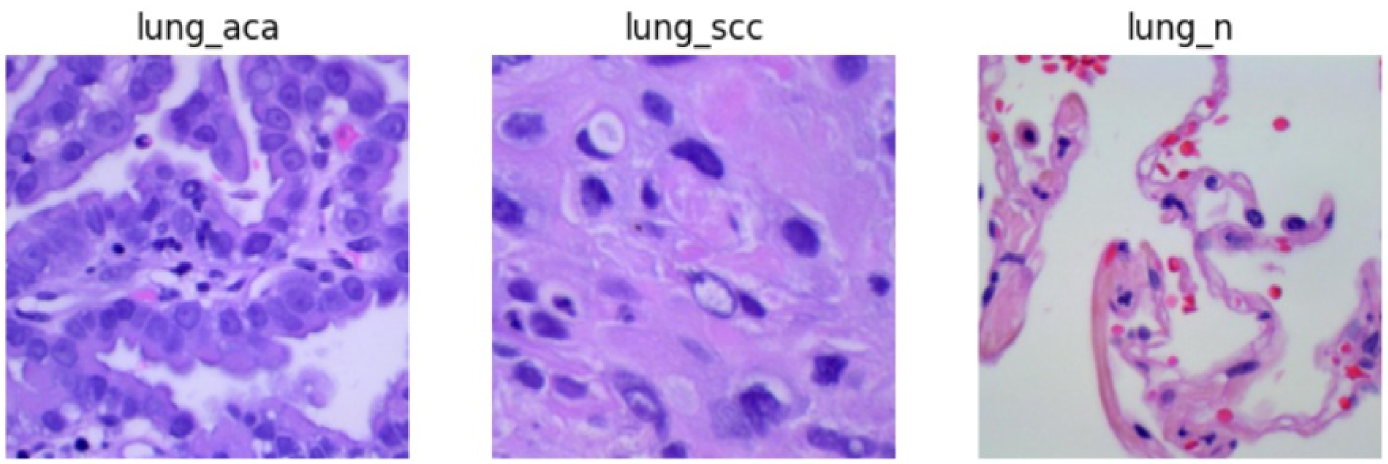
Lung Cancer Detection with Histopathological Images

Authors: Abuduaini Niyazi, Derek Kielty, Rishat Dilmurat, Sujoy Upadhyay, Ronak Desai

Submitted as a final project for the Erdos Institute Deep Learning (Summer 2024) Boot Camp

Introduction

One of the most important applications of deep neural networks in the healthcare industry is the classification of various types of diseases from image data. Particularly, one life-saving application is to detect cancerous cells at an early stage. In this project, we will train convolutional neural networks (CNNs) with a Kaggle Dataset of 15,000 images to distinguish normal lung tissues from cancerous ones.



The above images are (from left to right) adenocarcinoma (ACA), squamous cell carcinoma (SCC), and normal lung tissue (N). They are samples of images found in the data set. Not only do we need to differentiate cancer from no cancer, but we need to differentiate between the two types of lung cancer.

Some others have done an analysis on this with varying levels of accuracy. The analysis with the most upvotes (78 upvotes with a gold medal) used TensorFlow to train a CNN model with 94.67% accuracy on a validation set. Furthermore, using a pre-trained EfficientNetB3 model (pre-trained with ImageNet data), they achieved 100 percent accuracy on the validation set.

In this project, our group aims to match or exceed the validation set accuracy of the other groups that have worked with this dataset previously. Additionally, we use the frameworm of PyTorch to perform our machine learning analysis.

Pre-processing

The dataset that we are using has a total of 15,000 RGB images of size 768 by 768 pixels. The images are already evenly split into 5,000 images each of ACA, SCC, and N. Each of the 5,000 images was originally composed of only 250 images and through data augmentation (rotations, shearing, mirroring, etc.) a more robust set of 5,000 was obtained. To standardize the comparisons, we scale the image size down to 256 by 256, use a batch size of 64, and use a training-validation split of 80-20.

Methods

In carrying out this project, we studied CNN architectures and deep learning techniques that have been used successfully in the past. We started by constructing a basic architecture composed of a few convolutional blocks with max pooling followed by a few dense (fully connected) layers to the three outputs. We experimented with various techniques to improve the model accuracy such as increasing the width (number of channels), depth (number of layers), dropout layers, L2 regularization, and batch normalization. In general, we found that using methods like dropout and batch normalization, we can help prevent overfitting obtain higher accuracies as a result.

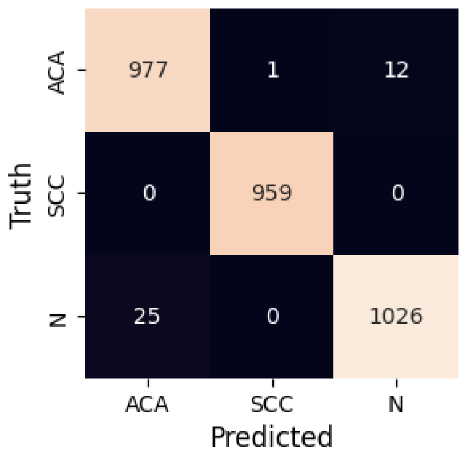
Next, we studied some more advanced CNN architectures. We first tried the Channel Boosted CNN. Then, we tried the Residual Neural Network (ResNet) with differing amounts of blocks.

Then, we tried the AlexNet with moderate success. We improved upon this by following the ZFNet architecture which changes, among a few other things, the filter size and stride in the first two layers. Their implementation saw an improvement over the AlexNet and we see an improvement in performance on this dataset as well.

Additionally, we used some pre-trained models as a comparison for what can be achieved in the best case. We tried Inception_V3 using just PyTorch and ResNet-18 with fastai.

Conclusion

Ultimately, we were able to find success with the ZFNet, achieving 99.5% training accuracy and 98.8% validation accuracy which is an improvement over the aforementioned gold medal winner on this dataset.



Above is a **confusion matrix** on the validation set (of 3,000 total images) which summarizes the nature of the misclassifications using this ZFNet model. Additionally, the pre-trained models were able to obtain a 100 percent training and validation accuracy, just like the gold medal winners.

Along the way, we learned a lot about convolutional neural networks, training deep learning models, and how to search for parameters that are important. We also learned how to build a model from just basic PyTorch components as well as how to implement pre-trained models that can be transfer learned with the dataset at hand.

If this project were to be extended, we could do the following

- Use the deconvolutional layers of the ZFNet to visualize the activations of the convolutional layers
- Train on a more complicated dataset that cannot achieve 100% performance on a pre-trained model.
- Make optimizations to the training that reduce the training time or number of epochs needed to train.

Acknowledgements

We made extensive use of Kaggle's free 30 hours per week of GPU compute time for our analysis. Additionally, we thank the Erdos Institute for providing us the opportunity to learn about deep learning and work on this project.

Other Resources

- PyTorch implementation of ZFNet by *CellEight*
- PyTorch implementation of Local Contrast Norm by *dibyadas*
    - Neat Article on the implementation of LCN