Image classification indicating presence of metastatic tissue using the PatchCamelyon dataset

Kaleb Sverdrup, Arya Kashani, Shreyas Devaraju
Henry Wu, Eric Hsueh
University of California, Irvine
Irvine, CA 92697

Abstract

In our report, we look into using a variety of CNN models and transfer learning methods to identify images of cancerous tissue in the Patch Camelyon dataset. We used a variety of prepossessing methods such as taking Sobel edges, rotating and flipping, and grayscaling images. We found that our implemented convolutional models outperformed the pre-trained DenseNet121 model and performing Sobel edge processing on the images helped to improve performance in most of our models.

1 Introduction

Patch Camelyon is a dataset consisting of histopathologic scans of lymph nodes. These images have binary labels for whether cancer tissue is present or not. The importance of this project is that if we are able to produce a model that is highly accurate, we would impact people's lives by detecting cancerous tissue with higher accuracy and hopefully save lives by detecting the cancer earlier in the patient's life. Currently the best model in the field was by MIT and the Harvard Medical School and got an AUC score of 99.35%.

2 Data

2.1 General Overview

The dataset overall contained 327,680 images, about 8 GB of data. We chose to do an 80/10/10 split so the training data size was 262,144 images while the validation and test sets had 32,768 images. Each image was using the RGB color space and had a size of 96x96, though only the center 32x32 of the images were used in determining the true labels.

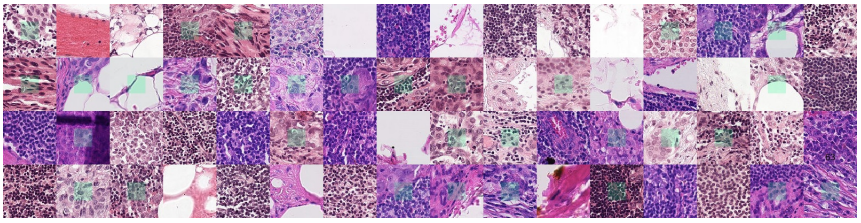

Figure 1: Green boxes indicate tumor tissue in the center region, which dictates a positive label.

2.2 Preprocessing

We tried various different preprocessing and augmentation methods: random rotation and flipping, gaussian noise, randomly reducing the jpeg quality, cropping the outside of the image, grayscaling the images, taking the Sobel edges or getting the edge gradients, and various combination of these methods. We used the rotation and flipping of the image to make our models invariant to these transformations and hopefully create a more robust model overall. The random jpeg quality and the addition of gaussian noise were used to help our model better deal with the noise and artifacts contained in some of the images of the dataset. The cropping of the outside of the images was used in order to decrease the size of our images which would speed up training, and to focus our model on the center of the image which is what really matters in determining the label. Finally, grayscaling and taking the Sobel edges of the images were used to force our models to focus more on the shapes and edges of the objects in the images as they are much more important than the colors in determining the labels of the images.

3 Architecture

3.1 Convolutional Neural Networks

A typical convolutional neural network has two parts, a convolutional base and a classifier. A convolutional base is composed of a stack of convolutional and pooling layers. The main goal of the convolutional base is to generate features from the image. A classifier is usually composed of fully connected layers. The main goal of the classifier is to classify the image based on the detected features. During our project, we created four models based on the convolutional neural network architecture, tested a pre-trained model, and put together three different ensemble models.

3.1.1 Model 1

Model 1 is a convolutional neural network composed of a stack of convolutional and max pooling layers with a ReLU classifier. The model had four sets of three convolutional layers followed by a max pooling layer. After the four sets there is a flattening layer, followed by a 128 node dense layer utilizing relu ending in a 2 node softmaxed dense layer for the predictions. All the convolutional layers have a kernel size of 3x3, a stride of 1x1, same padding, and use the relu activation function. The number of filters in each of the convolutional layers starts at 16 but increases by a factor of 2 with each subsequent set of 3 layers. The max pooling layers all use a pool size of 3x3 and a stride of 2x2 to condense the image between sets.

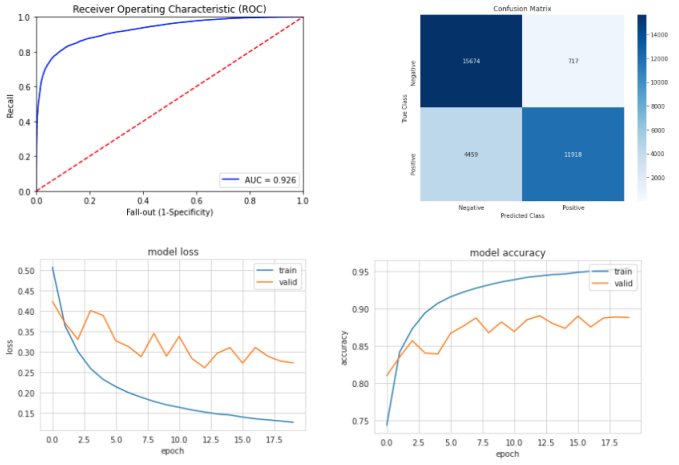

Figure 2: Model 1's ROC, Confusion Matrix, Loss Graph, and Accuracy Graph during training.

3.1.2 Model 2

Model 2 is a convolutional neural network on Sobel edges that is quite similar in structure to Model 1. The only changes in the network were to allow the new image size of 96x96x3x2 or the Sobel edges of the image. The first of two changes needed to support this were to go from a 2d convolutional layer to a 3d one, change the kernel size to 3x3x3 and to make the stride 1x1x1. The second and final change was to transition to max pool 3d, to make the pooling size 3x3x1 due to a lack of depth pooling in tensorflow, and finally to make the stride 2x2x2.

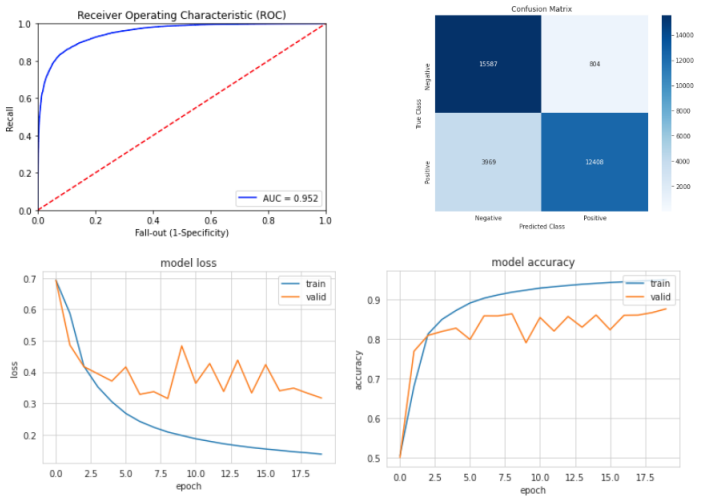

Figure 3: Model 2's ROC, Confusion Matrix, Loss Graph, and Accuracy Graph during training.

3.1.3 Model 3

Model 3 has a very similar structure to Scheme 1, but has an input size of 96, 96, 2 due to taking the sobel edges of grayscaled images. Model 3 uses SELU activation function instead of ReLU, enabling self-normalization and faster convergence compared to just using batch normalization and uses global average pooling at the end and batch normalization along with max pooling at the end of each convolutional block.

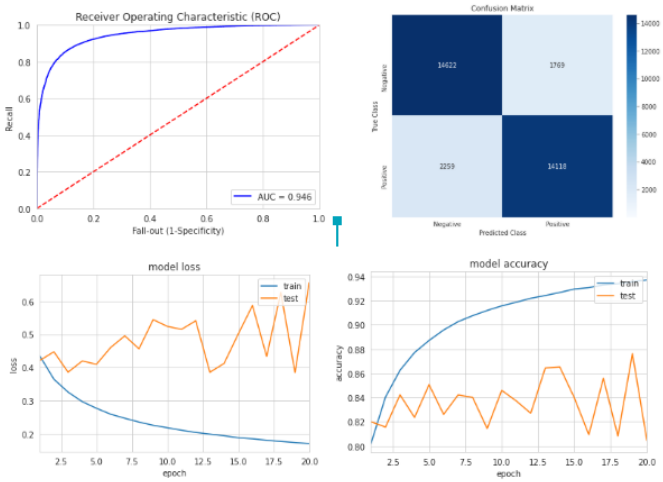

Figure 4: Model 3's ROC, Confusion Matrix, Loss Graph, and Accuracy Graph during training.

3.1.4 Model 4

Model 4 has a very similar structure to Scheme 2. Model 4 uses SELU activation function instead of ReLU, enabling self-normalization and faster convergence compared to just using batch normalization and uses global average pooling at the end and batch normalization along with max pooling at the end of each convolutional block.

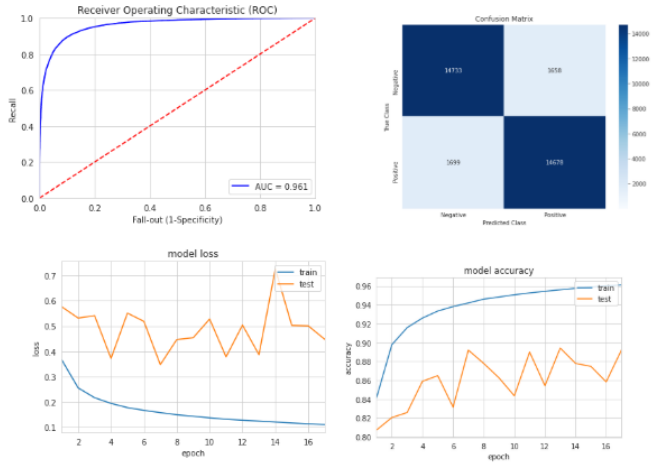

Figure 5: Model 4's ROC, Confusion Matrix, Loss Graph, and Accuracy Graph during training.

3.2 Transfer Learning

Transfer learning is a popular method that allows us to build accurate models in a timesaving way. With transfer learning, instead of starting the learning process from scratch, you start from patterns that have been learned when solving a different problem.

3.2.1 Densenet121 with Imagenet Weights

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. In our project, the pre-trained model we used was DenseNet-121 with Imagenet weights. One of the main features of DenseNet is that for each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. Thus, DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

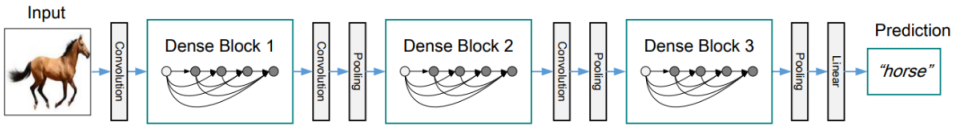

Figure 6: A simplified DenseNet architecture.

A pre-trained model can be repurposed by either training both the base and classifier layers, part of the base and the complete classifier or only the classifier. For large datasets, but different from the pre-trained model's dataset , both the base and classifier layers need to be trained. Whereas, if we have a small dataset, similar to the pre-trained model's dataset, only the classifier layers need to be trained.

In our case, we have a large dataset of medical images and these medical images were different from the Imagenet dataset used by the pretrained DenseNet model. We trained our densenet model in two different ways. First, we froze the base model and only train the top classification layers; we'll call

this the Frozen base model. Second, we unfroze the base model because we trained all the layers, both the base and the top classifier; we'll call this the Unfrozen base model.

In order to use the DenseNet121 model for our binary image classification application, we chopped off the top classification layers, to obtain 1024 features points after global average pooling and added a fully connected dense layer utilizing relu, batch normalization and dropoutof 0.5, ending in a 2 node softmaxed dense layer on top.

3.2.2 Frozen Base Model

For the Frozen Base model, we freeze the weights of the base layers of the DenseNet121 and only train the weights of the top classification layers using the pre-processed RGB images(only random rotation, flipping was used for image pre-processing).

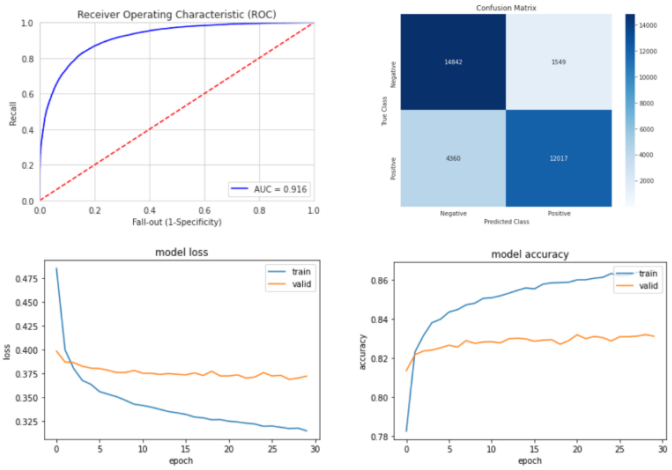

Figure 7: Frozen Base model's ROC, Confusion Matrix, Loss Graph, and Accuracy Graph during training.

Using a SGD optimizer with a learning rate of 0.001, we found that this model achieved a training accuracy of 0.8648 and test accuracy is 0.8194 when trained for 30 epochs. The training loss tends to decrease very slowly and is suggested to run for more epochs in the future.

3.2.3 Unfrozen Base model

For the Unfrozen Base model, we unfreeze the weights of the base layers and train all the weights of the base and classification layers using the pre-processed RGB images (only random rotation, flipping was used for image pre-processing).

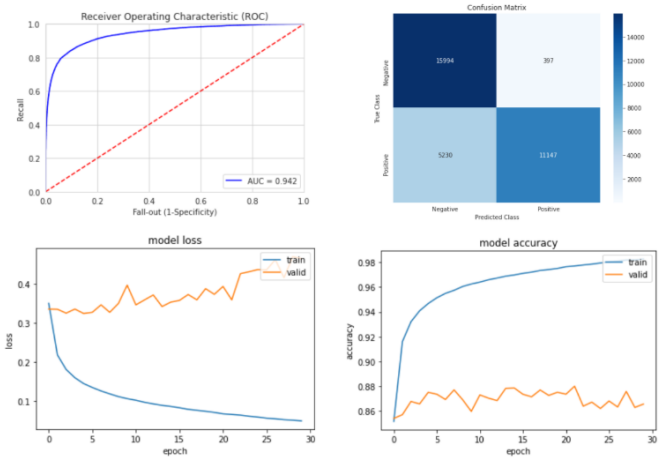

Figure 8: Unfrozen Base model's ROC, Confusion Matrix, Loss Graph, and Accuracy Graph during training..

Again, using a SGD optimizer with a learning rate of 0.001, we found that this model achieved a training accuracy of 0.9827 and test accuracy is 0.8286 when trained for 30 epochs. Here the model tends to overfit a lot. We suggest to use more preprocessing of input images or use regularization to alleviate this problem. Furthermore, we can reduce the overfitting by reducing model complexity, by training only a few layers of the model base. It was found that the learning rate needs to be small so as to not drastically change the initialized weights at the very beginning of the training.

3.3 Ensemble Models

We have three ensemble models that we put together. Our first is an ensemble of the DenseNet-121 model and model 1. The second is an ensemble of model 1 and model 2. The third is an ensemble of the DenseNet-121 model, model 1, and model 2. Our ensemble models were a good thought experiment to run. To make these ensembles, we took the outputs of the various models and passed them into a 8 node dense layer using relu, and finally had a 2 node softmax after that. We used this method to try and give the model more room to utilize the different models in its possesion. However, all three of our ensemble models ran into overfitting, as a result of the structure and the models. Due to the similarities between models 1 and 2, and the tendency for DenseNet-121 to overfit in general, an ensemble using any combination of these three models results in an ensemble with tendencies to overfit.

4 Comparison of Results

Table 1: Comparison of Model Results

	Training	Validation	Test	AUC
Model 1	0.9634	0.8853	0.8423	0.926
Model 2	0.9501	0.8766	0.8547	0.952
Model 3	0.9357	0.8763	0.8774	0.946
Model 4	0.9611	0.8922	0.8980	0.961
Densenet-121 (Frozen base model)	0.8648	0.8311	0.8194	0.921
Densenet-121 (Unfrozen base model)	0.9827	0.8655	0.8266	0.942
Ensemble Densenet-121 and Model 1	0.9817	0.8795	0.8403	0.930
Ensemble Model 1 and Model 2	0.9520	0.8763	0.8315	0.957
Ensemble of Denenet-121, Model 1, and Model 2	0.9411	0.8819	0.8531	0.955

5 Conclusion and Discussion

5.1 General Results

Our model 4 yielded the best results. If we had more time we would try more methods to improve that model and then create ensembles with that model included. Model 3 has an AUC score of 0.946, which was slightly worse than model 2's AUC score of 0.952, but it has an improved confusion matrix which has 1769 false positives and 2259 false negatives whereas model 2 has a confusion matrix with 804 false positives and 3969 false negatives. While the number of false positives has gone up somewhat compared to model 2, the number of false negatives has gone down substantially, which is a major benefit when it comes to detecting cancer. This will help patients get accurate diagnoses and the appropriate treatments they need before the cancer spreads all over the body. Model 4's AUC score is 0.961, which is a substantial improvement over the AUC scores of models 2 and 3. Its confusion matrix is better than model 3 overall and is the most balanced compared to the confusion matrices of all other models, and has 1658 false positives and 1699 false negatives. Transfer learning using the Densenet121 Frozen base and Unfrozen base models performed decent, getting a test accuracy of about 82 percent, but did not out perform the convolution models due to over fitting issues.The ensembles we created all performed well, but that was only due to them fitting onto the best model each of them contained. They did very little to draw conclusions from the lesser models used to create them. This is due to the aforementioned problems with the models that make them up, the Densenet model overfitting and the other 2 models having very similar structure.

5.2 Preprocessing

Gaussian blur and the random jpeg quality both ended up having a neutral to negative impact on the performance, presumably due to noise created by them erasing features important to labeling the images. The cropping to the centers of the image was another place where a negative impact on performance was seen, this is assumedly due to cancer being seen on the outside being very indicative of there being cancer in the center of the image as well. The cases in which there was cancer outside, but none in the center seem too small to outweigh the benefits of keeping the whole image. The rescaling of the image was a mixed bag in terms of the benefits it gave to the models. Due to changing the image from a 96x96x3 to a 96x96x1 matrix the grayscale images the training time of the model was cut in half, but the process of grayscaling the images also ended up causing the model to fall in ability by about 2%. This 2%, though quite major, can be outweighed by the massively decreased training time if resources for training are in short supply. Finally getting the Sobel edges of the images ended up causing our model to overfit less on our dataset and allowed for higher test scores than previously seen. The Sobel edges did slow down our training by transitioning the image to a 96x96x3x2 matrix, but the results outweigh this downside.

5.3 Architecture

Dropout helps combat overfitting by reducing the number of parameters being trained. However, using dropout too often and with high ratios on small models can cause the model to underfit due to it reducing the number of parameters to the point of being insufficiently complex. With larger datasets and/or larger image sizes, the model needs to be fairly complex to accurately predict classes, so in these cases, the less dropout, the better. Using a dropout of 0.2 right before the softmax layer seems to work the best with small models. Global average pooling works best when there are a very large amount of parameters, as it significantly cuts down on the number of parameters, which can help reduce overfitting.

6 Contributions

Kaleb Sverdrup

I mainly looked at implementing and testing various preprocessing methods such as the flipping, rotating, gaussian noise addition, and the Sobel edges. Along with this I worked on the ensemble model and suggested the expansion of our models from 3 sets of 2 convolutional layers to 4 sets of 3 convolutional layers which helped increase the accuracy of our models overall.

Eric Hsueh

I mostly researched and experimented with different preprocessing methods and applied them to different base models. Different preprocessing methods such as random image rotation, random image flipping (both horizontally and vertically), and greyscaling were tested. Other preprocessing methods were tested by other members of the group as well. I ran a variety of models and got back a wide range of results.

Arya Kashani

I experimented with all the different models presented (geert litjens, kaggle beginner, and densenet) and ran many different variations of the models with differing hyperparameters on google colab. I contributed to a fair portion of the slide set presented and I put together the bibliography. I also helped put this final report together.

Shreyas Devaraju

I mainly looked at the implementation of transfer learning using the DenseNet121 model for our application. Looked into the various transfer learning strategies to train the DenseNet121 model, such as freezing and unfreezing the model base. I assisted in designing the ensemble model. Experimented with various hyperparameters and some preprocessing methods, and their effect on the performance of CNN models. Also, contributed in putting together the presentation and report.

Henry Wu

I experimented with the base models and tried many modifications with the models, such as adding batch normalization after each convolutional layer, using global average pooling after all convolutional layers instead of flattening, using SGD and Adam on the same model to see which one is better, and using SELU activation function on each convolution layer instead of ReLU. I also added and modified code for displaying AUC curve, confusion matrix, model accuracy and model loss trend graphs to work on the models.

References

[1] Basveeling. "Basveeling/Pcam." GitHub, 28 Apr. 2020, github.com/basveeling/pcam.

[2] Geert Litjens. "Getting Started With Camelyon (Part 1)." Geert Litjens, Geert Litjens, 24 Apr. 2019, geertlitjens.nl/post/getting-started-with-camelyon/.

[3] Gomez, Pablo. "Complete Beginner's Guide [EDA, Keras, LB 0.93]." Kaggle, Kaggle, 23 Jan. 2019, www.kaggle.com/gomezp/complete-beginner-s-guide-eda-keras-lb-0-93.

[4] Lau, Suki. "Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning." Medium, Towards Data Science, 1 Aug. 2017, towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1.

[5] "Lymph Node Metastasis." Lymph Node Metastasis - Libre Pathology, librepathology.org/wiki/Lymph_node_metastasis.

[6] Marcelino, Pedro. "Transfer Learning from Pre-Trained Models." Medium, Towards Data Science, 23 Oct. 2018, towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751.

[7] "patch_camelyon : TensorFlow Datasets." TensorFlow, 29 Apr. 2020, www.tensorflow.org/datasets/catalog/patch_camelyon.

[8] "Transfer Learning with a Pretrained ConvNet : TensorFlow Core." TensorFlow, www.tensorflow.org/tutorials/images/transfer_learningadd_a_classification_head arxiv.org/pdf/1909.11870.pdf.