Efficient cancer classification from tissue slide images using Convolutional Neural Networks

Joyce Wu *jmw784*
Eduardo Fierro *eff254*
Raúl Delgado Sánchez *rds491*

Abstract

Visual analysis of solid tissue imaging is one of the primary methods currently used by pathologists for determining the stage, type, and subtypes of cancer. As differentiating between cancerous subtypes can be challenging even to experts, recent research has been focused on developing a automatic prediction system for this multi-class classification problem. With a greatly simplified convolutional neural network (CNN) architecture that is significantly faster during training and prediction time, we obtained results comparable to the state-of-the-art for the lung cancer classification task. Furthermore, we successfully applied the same architecture to two new types of cancer, kidney and breast. Lastly, we compare the results of our network trained on the three types of cancer and provide discussion on the factors that led to increased performance.

1 Introduction

In this paper, we explore implementing and training a convolutional neural network (CNN) architecture for the tasks of both recognizing non-cancerous tissue from cancerous tissue, and differentiating between cancer subtypes using high-resolution tissue slide images obtained from The Cancer Genome Atlas (TCGA). The benefits of distinguishing non-cancerous tissue from cancerous tissue is obvious, but quickly and accurately differentiating between subtypes can also be critical for determining the proper chemotherapy treatment for the patient. For example, in the case of lung cancer, the two major subtypes are squamous cell carcinoma and adenocarcinoma. Certain agents may be less efficient [Deshmukh et al., 2010] or more invasive for patients presenting squamous cell carcinoma [Scagliotti et al., 2009]. On the other hand, adenocarcinoma is more likely to contain genetic mutations which may be treated with targeted therapy, such as EGFR mutations and ALK rearrangement [Snyder, 2016].

Although cancerous subtypes are quite visually different, diagnostic agreement for classifying adenocarcinomas and squamous carcinomas is actually relatively low (κ = 0.41 - 0.46 among community pathologists, κ = 0.64 - 0.69 among pulmonary pathology experts and κ = 0.55 - 0.59 among all pathologists under study) [Grilley-Olson et al., 2012]. The difference between community pathologists and pulmonary pathology experts suggests that automatic diagnostic technology may help local pathologists fill some of the expertise gap. Furthermore, training a model for this task sets the groundwork for applying convolutional neural networks to more difficult tasks such as mutation detection.

For the same task on lung cancer, Yu et al. [2016] used an automatic image-segmentation pipeline to identify the tumour nuclei and tumour cytoplasm from which they extracted image features from. Using several classic machine learning models such as Naive Bayes, Support Vector Machine, and Random Forest, they found that a Support Vector Machine with Gaussian kernels performed the best on this task. Their best model achieved AUC $\sim$0.88 for distinguishing between cancer and non-cancerous tissue, and $\sim$0.85 for distinguishing between the cancerous subtypes.

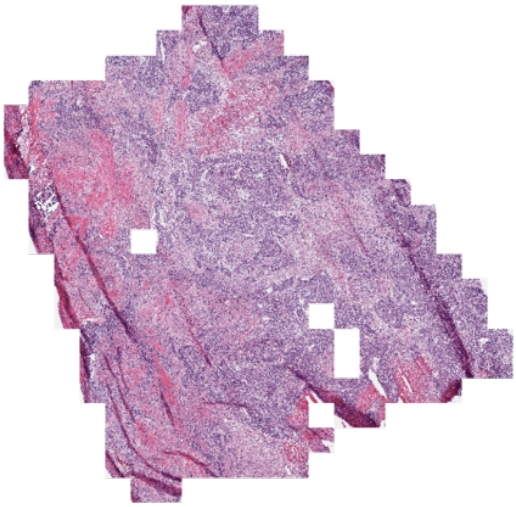


Figure 1: Example tissue slide of TCGA-LUAD

Advances in deep learning have paved the way for artificial neural networks to surpass most traditional machine learning methods in the field of image processing. In particular, convolutional neural networks (CNNs) first proposed by Le Cun and Bengio [1994] have quickly risen to the state-of-the-art on almost all image based tasks. One of the benefits of using a convolutional neural network architecture is that using domain knowledge to handcraft an image feature extraction system is not required as in Yu et al. [2016]. Conveniently, feature extraction is an automatic task for neural models, and is trained end-to-end in a CNN.

Our advisors Aristotelis Tsirigos, Nicolas Coudray, and Narges Razavian trained an implementation of Google's Inception V3, a deep ResNet He et al. [2015a], on the lung cancer classification task and obtained an impressive macro AUC of 0.97 (AUC $\sim$0.993 for distinguishing between cancerous and non-cancerous tissue, and 0.947 for distinguishing between the cancerous subtypes) [Coudray et al., 2017]. We replicated their work with a modified, simpler architecture that is much faster to train and run at prediction time, and achieved a comparable macro AUC of 0.947 on the test set (AUC $\sim$0.99 for distinguishing between cancerous and non-cancerous tissue, and $\sim$0.92 for distinguishing between the cancerous subtypes). Furthermore, we have applied the same architecture and hyper-parameter setting to two new types of cancer, kidney and breast, for which we obtained macro AUCs of 0.995 and 0.996, respectively.

2 Data

High-resolution (20k) tissue slide images were downloaded from the Genomic Data Commons Database from The Cancer Genome Atlas (TCGA) for lung, kidney, and breast cancer [GDC]. All of the slide images were obtained from patients with cancer. Multiple tissue slide images may be from the same patient, some containing cancerous tissue and others not. The slides were labeled by pathologists, which were used as the ground-truth for training. Slides containing no cancerous tissue, although originating from a patient with cancer, were labeled as Solid Tissue Normal.

For lung cancer, we had two cancerous subtypes: Lung Squamous Cell Carcinoma (TCGA-LUSC) and Lung Adenocarcinoma (TCGA-LUAD). For kidney cancer, we had three cancerous subtypes: Kidney Chromophobe (TCGA-KICH), Kidney Renal Clear Cell Carcinoma (TCGA-KIRC), and Kidney Renal Papillary Cell Carcinoma (TCGA-KIRP). Finally, we only had one type of breast cancer: Breast Invasive Carcinoma (TCGA-BRCA).

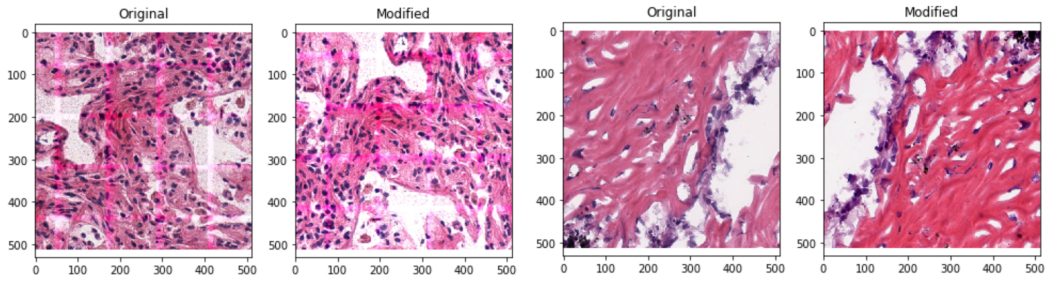


Figure 2: Data Augmentation Example

3 Methodology

3.1 Data pre-processing

3.1.1 Image tiling

The original whole-slide tissue images are extremely high-resolution (sometimes over 100,000 pixels wide), so we employed a method to tile the original images into smaller images for the neural network to process. This tiling strategy is commonly employed for high-resolution medical images, otherwise the resolution would be limited by the GPU memory [Ronneberger et al., 2015]. Furthermore, the size of the training data once tiled is much larger than the original number of training tissue slide images. For example, we only had 2167 lung cancer slides, but over 650k tiles. However, this tiling method operates under the assumption that a label for a particular slide is valid for every individual tile within the slide, which may not necessarily be true. In other other words, tiles corresponding to normal tissue could potentially be labeled as cancerous if any tile from the same image presented a cancerous subtype.

For tiling, we used the open source package `openslide`. The images were tiled into 512×512 pixels tiles using non-overlapping windows. Additionally, tiles with more than 25% of background were filtered out, as they are unlikely to contain informative features. Before feeding the images to the network, we further downsample the tiles to 299×299 pixels to speed up training and to make the network comparable to Google's Inception V3, which requires 299×299 inputs.

3.1.2 Train / validation / test set split

For the three cancer types, the data was split into 70%, 15% and 15% respectively for the train, validation and test datasets. To avoid leakage, we ensured that all slides (and tiles) that came from the same patient were included in the same dataset (train, valid, or test). This ensured that the model could not simply learn histopathological features that are patient specific to achieve good performance at validation and test time.

3.1.3 Parsing and concatenating metadata

Alongside the images, we also had access to clinical data about the patient, including but not limited to: age at diagnosis, the number of cigarettes smoked per day, and gender. Furthermore, we had the x, y coordinates of the tile with respect to the original image. To allow the neural network to take advantage of this information, we encoded and concatenated these features to the input image as extra channels.

3.2 Data augmentation

Data augmentation is a form of regularization, as it helps highly complex neural networks not overfit by creating more training examples for the network to learn from. In this particular application (which may not be true for other applications such as digit classification), we know that the rotational orientation does not change the classification of the tissue. Furthermore, varying dying methods and lighting environments means that the same histological identifying features may be slightly different in color and saturation between slides.

Therefore, using the `Tranforms` library from `Torchvision`, along with self-written functions and classes using the Python `Pillow` library, we augmented the training data by randomly introducing positional transforms such as: a vertical flip, a horizontal flip, or a rotation of 0^o, 90^o, 180^o or 270^o degrees. Additionally, we randomly adjusted the hue, gamma, brightness, contrast, and saturation of the image.

3.3 Weight initialization

As deep neural networks are highly non-convex, it is important to have the proper initialization to achieve the optimal global minimum. Initial weights that are too small may lead to the signal vanishing through the layers. On the other hand, weights that are too large can make the gradient explode. Therefore, we tried the following initialization methods.

3.3.1 Gaussian

Gaussian initialization is the standard method for initializing neural networks, which fills the weights vector with random Gaussian noise centered at 0 with a certain amount of variance.

3.3.2 Xavier

Xavier initialization samples weights from a uniform distribution centered around zero and bounded by a function of the layer size. This initialization maintains variances of the activations and back-propagated gradients through the network, preventing both vanishing and exploding gradients for deep models [Glorot and Bengio, 2010].

3.3.3 He

He initialization properly scales the backward signal by sampling initial weights from a mean-zero Gaussian distribution that has a variance of $\sqrt{2/n_L}$, where n_L denotes the size of the layer. This method allows deeper models to converge [He et al., 2015b].

3.4 Optimizer

Different gradient-based optimizers may also influence the local minimum where the model converges. Along with the standard Stochastic Gradient Descent (SGD) optimizer, we focused on using momentum-based optimizers that can adaptively change the learning rate during training, allowing the model to reach lower minimum.

3.4.1 SGD

Classic stochastic gradient descent is widely used to train neural models. Although it may require more task specific tailoring, using random mini batches to update the descent direction allows for much faster convergence, as only a fraction of the data is used for each update. Additionally, it has been hypothesized that stochasticity allows the model to "jump" out of local minima during training time.

3.4.2 Adam

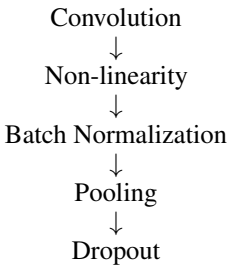
Adam is an algorithm for the first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. It is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. Furthermore, the hyper-parameters have intuitive interpretations and typically require little tuning [Kingma and Ba, 2014].

3.4.3 RMSprop

RMSprop adaptively modifies the learning rate by normalizing the current gradient by dividing by the moving average of squared gradients. This decreases the step for large gradients to avoid exploding, and increases the step for small gradients to avoid vanishing. It is a very robust optimizer which has pseudo curvature information and is equipped to deal with stochastic learning objectives [RMS].

3.5 Model architecture

A typical ResNet type block of layers for convolutional neural networks is:



Convolution layers are the core building blocks of convolutional neural networks. This layer applies a convolution operation to the input using a fixed kernel size. One can select how many feature mappings (output features) for each layer to learn. In general, convolution layers are good for learning hierarchical features that are invariant to rotation and translation.

Adding non-linear function allows the model create non-linear outputs. This distinguishes neural networks from traditional linear models, and increases the complexity that the model is able to represent. This reduces the bias of the model, but increases variance. Rectified linear units (ReLU) and their derivatives have been popular as they are less susceptible to the vanishing gradient problem, and have the ability to generate sparse representations.

Batch normalization normalize layer inputs to address the issue of internal covariate shift. This allows neural networks to use much higher learning rates and decreases the need to carefully tune the initialization. Experiments on ImageNet have shown that applying batch normalization after the non-linearity results in better classification accuracy [BN]. Therefore, we chose to apply batch normalization after the non-linearity.

Pooling combines features together for inference. Two common methods are average pooling and max pooling, which output the average and maximum value of the features within the window, respectively.

Dropout randomly "drops out" features with a certain probability p by setting their values to 0 during training. This is a form of regularization, as it ensures the features are relatively independent and "strong" by themselves without relying on other features.

We built a neural network architecture by stacking these basic convolution blocks. Within each block, we adjusted hyperparameters such as the kernel size, stride, padding, and number of output features. Additionally we tested different nonlinearity functions, evaluated whether adding each pooling type improved results, and tunned the dropout rate p.

The final layer was a fully-connected linear layer that mapped the number of features down to the number of classes. Typically, this layer is followed by a softmax or logsoftmax that transform the class scores into adjusted probabilities. However, we used the cross entropy loss which directly accounts for this.

3.6 Loss function

For this multi-class classification problem, a naturally good loss function is the cross entropy loss:

$$L_{x,y} = -\log\left(\frac{\exp(x[y])}{\sum_j \exp(x[j])}\right)$$

where x is an array that contains the probability predictions for all classes ($\sum_j x[j] = 1$), and y is the correct class.

3.7 Evaluation

3.7.1 Aggregation

While the neural network was trained on a per-tile basis, application of the model must be on a per-slide basis. Thus, the scores from the neural network per tile must be aggregated for the final prediction. We implemented two methods for computing the final aggregated prediction.

Average score

The average score across all the tiles for each class is computed. These are the final scores for the 3 classes.

Proportion of tiles

The class with the maximum score is assigned to each tile. The final score is the proportion of tiles assigned to each class.

3.7.2 Metric

Although cross-entropy loss is a good loss function for multi-class classification, the high level metric that we aimed to maximize is area under the Receiver Operating Characteristic curve (AUC). However, AUC is traditionally defined only for binary classification, but the lung and kidney tasks had more than two classes.

To resolve this, we first calculated the AUC of individual classes using a one-vs-all method. Then, we calculated the micro-average AUC (micro AUC) by considering each element of the label indicator matrix as a binary prediction. We also calculated the macro-average (macro AUC), which is the unweighted mean of the individual class AUCs. We decided to use the macro AUC as the high level metric to optimize, but we provide analysis on the individual class AUCs as well.

3.7.3 Early stopping

Using the macro AUC from the previous section, we implemented an early stopping method to halt training if the performance on the validation set was no longer improving. This is another form of regularization, as it stops training once the model is starting to overfit on the training data. We triggered early stopping if there was no consecutive increase in macro AUC greater than 0.0001 within the last three consecutive checkpoints on the validation set. Since the validation checkpoints also take fair amount of time to be calculated, they were only performed after each epoch.

4 Results

Due to the size of our dataset and limited access to GPU computing, we were unable to run a full grid search on all the hyperparameters and options listed in the methodology section. However, we performed a kind of "coordinate descent" search in which we varied a particular hyperparameter until we found the best performing model, then moved on to optimize the next hyperparameter. With this method, we were still able to draw some insights into the effect of particular hyperparameters on model performance.

4.1 Model architecture

Our first approach was a network that tried to imitate the first few layers of the Google Inception V3 model, by using 3 convolution blocks going from $32 \rightarrow 64 \rightarrow 80$ features. These blocks did not include a pooling function, so this approach required using a final linear layer to map 25,920 features down to the number of classes. We found that without pooling the features together throughout the blocks, the model was unable to properly perform inference. This may be why the Inception model follows these upsampling blocks with several layers of pooling blocks.

Therefore, we tried a 4 convolutional block network going from $16 \rightarrow 32 \rightarrow 64 \rightarrow 32$ output features, using pooling in every block. This was followed by a linear layer that mapped 2592 features

down to the number of classes. With this simpler architecture, the model was able to learn, but we were only able to obtain a macro AUC of $\sim$0.903. We concluded that a deeper network may better represent the complexity of the problem.

The final architecture we developed can be visualized in the Appendix A. We used 6 layers of convolutional blocks followed by a fully connected layer. The first block uses a kernel size of 5, while the following blocks all used a kernel size of 3. Furthermore, the first two blocks did not contain a pooling layer, as we wanted to upsample the number of features before downsampling with max pooling.

4.1.1 Nonlinearity

In our experiments, we found that both ReLU and LeakyReLU worked well, but using LeakyReLU resulted in slightly better performance. Therefore for our final architecture, we used LeakyReLU with negative slope 0.01 as the non-linearity.

4.1.2 Dropout

In our experiments, we found that tuning the dropout rate was very important for model performance. When we first used the default dropout rate of $p = 0.5$, we were only able to obtain an AUC of $\sim$0.857. We realized that the dropout rate may be too high, especially as we were applying dropout after every convolutional block. In comparison, the Inception model only applies one layer of dropout after all of the blocks. When we decreased dropout to 0.1, we were able to consistently reach macro AUC above 0.9. Another explanation may also be due to the use of batch normalization within each convolutional block. Both batch normalization and dropout are regularization methods, so the use of batch normalization reduces the need for dropout. In this case, the higher dropout of $p = 0.5$ was causing our model to underfit.

4.2 Aggregation method

We tried both the average score and proportion of tiles methods for aggregating the tiles, as described in the methodology section 3.7.1. We found that the methods were comparable, though the average method performed between 0.003 and 0.008 AUC better on the three datasets. Intuitively, this makes sense: Assigning each tile a class based on the maximum probability would treat a tile that is assigned a class with 99% probability the same as one with 60% probability. Therefore, taking a "majority vote" method per tile could potentially discard some useful information for the prediction.

Table 1: Validation performance - Macro AUC

Dataset	Lung	Kidney	Breast
Proportion (max)	0.970	0.985	0.996
Average	0.978	0.993	0.999

4.3 Other hyperparameters

One of the first challenges we faced was finding the proper learning rate for the model to learn. For learning rates smaller than 1e-5, the model tended to get stuck with no major decreases in loss after one epoch. On the other hand, when the learning rate was larger than 0.001, the loss function diverged, and the AUC jumped around unstably. After several iterations, we found that the best learning rate was 0.001. We also tried implementing a decaying learning rate approach, which should help the model achieve a better minimum once it is closer to the the optimum. However, this approach did not end up outperforming the best learning rate with an adaptive momentum based optimizer.

As for the initialization, although Gaussian initialization worked well, Xavier initalization helped get the model achieve better performance earlier (AUC 0.94 after 1 epoch vs 0.89). However, we found that the final AUC achieved by Xavier initialization was comparable to the Gaussian initialization at the end of training. Surprisingly, we found that He initialization diverged and did not outperform

the Gaussian initialization as expected. This initialization, originally derived to deal with a novel Parametric Rectifier Linear Unit (PReLU) [He et al., 2015b], perhaps may not apply as well to the LeakyReLU units (a specific case of the proposed PReLU) we ended up using.

For data augmentation, we determined the acceptable limits of variation from the original image by manually visualizing several sample outputs. For example, to augment of brightness for each tile, a value was sampled uniformly at random between 0.75 and 1.25; that is, the brightness would not be changed by more than 25%. Similarly, we chose a maximum contrast difference of $\pm 25\%$, saturation difference of $\pm 25\%$, and hue difference of $\pm 5\%$. We believe that these exact limits could be further tuned, but our heuristic approach already showed a significant boost in performance. Without augmentation, our model was only able to obtain macro AUC around 0.93 on the validation set for the lung cancer task. With data augmentation, we were able to consistently achieve macro AUCs above 0.96.

In terms of the optimizers, both Adam and RMSProp performed similarly. SGD, on the other hand, did not perform as well. This is likely because SGD does not have a momentum-type mechanism to adaptively adjust the learning rate. Use of SGD can require a lot more careful hyperparameter tuning than the adaptive methods such as Adam and RMSProp.

4.4 Final results and Discussion

As shown in Table 2, we were able to obtain a validation AUC of 0.978 on our baseline lung cancer task, which was comparable to the 0.97 result of our advisors with Google's Inception V3. However, unfortunately, the same model evaluated on the test set was only able to achieve around 0.947 macro AUC. As our advisors' performance of 0.97 was on the test set, we conclude that our simpler model is comparable, but slightly worse in performance. However, it is important to note that our model only took 15 hours to reach the desired performance, while their implementation of Google's Inception V3 took around 2 weeks to train. With further hyperparameter tuning, we may be able to achieve even better results.

Table 2: Test performance - Macro AUC

Dataset	Lung	Kidney	Breast
Validation	0.978	0.993	0.999
Test	0.947	0.995	0.996

As seen in the Receiver Operating Characteristic curve in Figure 3, the model does a nearly perfect job when the classification task is only between cancerous tissue and non-cancerous tissue, with AUC of 0.999. But when trying to differentiate between the two subtypes of cancer, TCGA-LUAD and TCGA-LUSC, it doesn't perform as well, with individual one-vs-all AUC of 0.92 for both classes. This lines up with our intuition that non-cancerous tissue should be quite visually distinct from cancerous tissue, while cancerous subtypes are more visually similar. The results from Yu et al. [2016] also follow a similar pattern with a slight difference in AUC between the two tasks.

We also created a visualization for the output prediction of our model, which can be seen in Appendix B. The hue of each tile is adjusted to match the class with the maximum probability score predicted by the network. We used opacity to represent the amount of confidence in that prediction. For example, if the score assigned to the maximum class is 0.9, the tile is shown with 0.9 opacity. Therefore, the darker tiles are the ones that the model is most confident in. Only two of the tiles that compose the original image are classified as Solid Tissue Normal, that is 0.08% of all the tiles. This follows from our observation that the model is quite good at distinguishing between cancerous and non-cancerous tissue; the true label for this particular example is cancerous TCGA-LUAD. Meanwhile, the prediction for TCGA-LUAD and TCGA-LUSC represent 34.85% and 64.32% of the 241 tiles, respectively. This means the maximum method with proportion would score TCGA-LUSC higher than TCGA-LUAD for this particular example. However, you can see some of the blue tiles are fairly light in opacity. When the average method is used, we get a probability distribution of [0.13 0.41 0.57] for Solid Tissue Normal, TCGA-LUAD, and TCGA-LUSC respectively. While our model is still likely to predict the wrong class for this example, it illustrates why the model may perform better when using the average aggregation method.

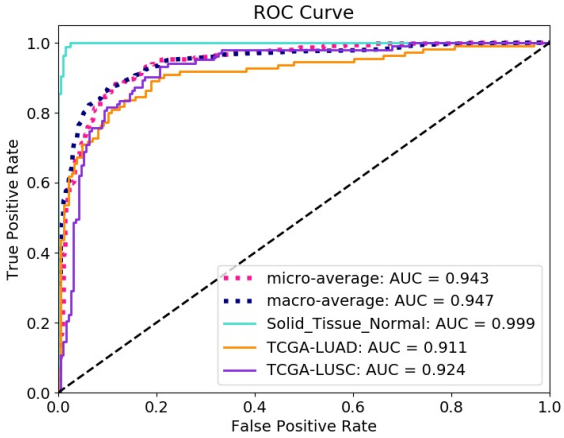
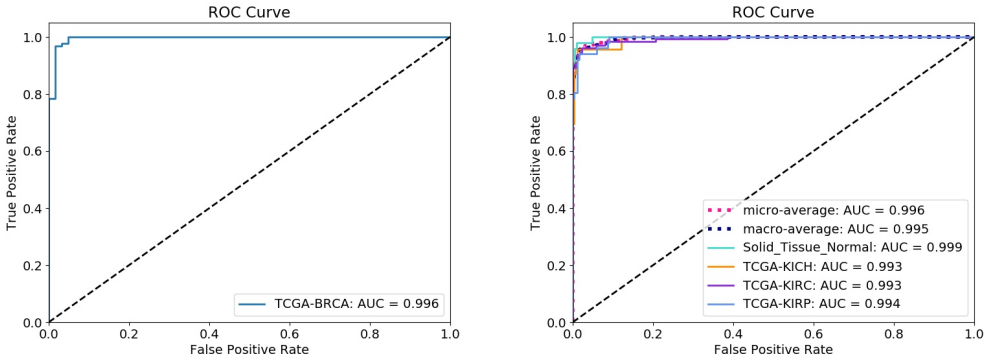


Figure 3: Receiver Operating Characteristic Curve for lung cancer on the test set



(a) Breast cancer

(b) Kidney cancer

Figure 4: ROC curves for breast and kidney cancer on the test set

As seen in Table 2, the best model we found from our work with lung cancer transferred quite well to the breast and kidney classification tasks. While we did retrain the entire network (in contrast to a transfer learning approach), we found that even without retuning the hyperparameters, the performance was quite impressive. In fact, we did not see the same kind of drop in performance between the validation and tests sets, and both were able to achieve a macro AUC around 0.996.

For breast cancer, there were only two possible classes: non-cancerous and cancerous tissue. Just as the case with lung cancer, we found that our model can easily distinguish between these two classes. However, it is still quite impressive considering that we did not further tune the hyperparameters specific to this data set. This suggests that the framework we developed is well suited for the *general* task of distinguishing cancerous tissue from non-cancerous tissue!

For the kidney cancer task, there were more cancerous subtypes (3 compared to 2 for lung). Nevertheless, we found that the model was almost perfectly able to distinguish between them, and achieved a macro AUC better than the lung cancer task by almost 0.05 AUC. On the test set, the individual class AUCs were 0.999, 0.9934, 0.9929, and 0.9936 for Solid Tissue Normal, TCGA-KICH, TCGA-KIRC, TCGA-KIRP, respectively. This may simply be due to the fact that these cancerous subtypes are more visually distinct than the lung cancer subtypes, and therefore the task is easier overall. However, we do not have a baseline to compare to, as the previous work by Yu et al. [2016] and our advisors were focused only on lung cancer. Another possible explanation could be due to the difference in the available training data. While we had 2167 slides for lung cancer, we were able to obtain 2355 slides for kidney cancer - around a 8% increase. However, after tiling, we had around 767k kidney cancer tiles compared to 461k lung cancer tiles in their respective training sets, a 66% increase! This

phenomenon may be explained by the pre-processing step that removed any tile with more than 25% background. The kidney tissue slides may be more dense with richer information.

5 Inception V3 architecture

In parallel to our work with the simpler architecture, we also worked on directly replicating some of the work our advisors previously did on the lung cancer task. To this aim, we used the same training loop and aggregation methods, but with Google's Inception V3 architecture. Using the network publicly available in PyTorch's model library[1] [Szegedy et al., 2015], we transferred the weights from a model pretrained on the ImageNet task. As this is an extremely deep model, we focused on using transfer learning. This was done by by freezing the weights of all the layers except the last layer, and only propagating gradients to update the weights in the linear layer. Additionally, we had to adjust the output dimension of the last layer to match our 3-class lung cancer problem. As the architecture was fixed, we primarily tuned the learning rate and optimizer for this task.

The best performing model, which used RMSprop as the optimizer, was only able to reach an AUC of 0.904 after 15.5 hours of training, when it was stopped by early stopping. Just as with our own architecture, this network struggled with the difference between the two different cancer types (one-vs-all AUC of 0.900 and 0.831 for Lung Squamous Cell Carcinoma and Lung Adenocarcinoma respectively). This is comparable to our advisors' results with transfer learning; they were only able to achieve an AUC of 0.847 for distinguishing between the cancerous subtypes using transfer learning [Coudray et al., 2017].

Our advisors found that training the Inception model fully was necessary to obtain the best performance [Coudray et al., 2017]. However, since we were able to obtain comparable AUC scores using a much simpler model, we decided to invest our GPU resources tuning the hyperparameters of the simpler architecture instead. It is important to note that even though we were only training the last layer with transfer learning, epoch training time using this architecture was still at least 50% longer than the training time with the simpler one.

6 Conclusion

Using our convolutional neural network approach, we have exceeded the performance of Yu et al. [2016] on the lung cancer classification task with a macro AUC of 0.947. While not as high as the result from using a fully trained Inception V3 network, the results are comparable and impressive given the reduced complexity of the model.

With a simpler architecture, we were able to reach comparable results to a very deep network. This difference reflects in the reduced training time of 2 days as compared to 2 weeks for the deep network. This also means it will be significant faster at prediction time; aggregation and prediction for a particular slide image took around 30 minutes. Limited resources in a realistic health care setting means that this speed-up could translate to positive real world outcomes for patients. It also allowed us to train the same network with the same hyperparameter specifications on new cancer types, confirming that the successes gained in lung cancer can be successfully applied to many other different cancers. The final architecture, as pictured in Appendix A, consists of 6 convolutional blocks using LeakyReLU, batch normalization, and dropout. We found that using data augmentation, tuning the learning rate, reducing the drop out rate, and increasing the depth were the most important factors for achieving the best performance.

For lung cancer, the individual class AUCs reflect how the network was better at distinguishing between Solid Tissue Normal (non-cancerous tissue) and the other two types of cancer, with an AUC around 0.999. On the other hand, the network's performance is not as good when trying to differentiate between the two different classes of cancer: Lung Squamous Cell Carcinoma (TCGA-LUSC) and Lung Adenocarcinoma (TCGA-LUAD).

Another positive outcome from our work is that we were able to use the exact same model architecture and hyper-parameter settings to classify kidney cancer and to breast cancer. In fact, the AUC scores improved in comparison to the ones achieved for lung: macro AUC of 0.997 and 0.996 on the test

[1] See http://pytorch.org/docs/master/torchvision/models.html

set respectively. It is important to note that kidney cancer task was a 4 class problem (including Solid Tissue Normal), while the breast cancer task was a binary classification between cancerous and non-cancerous tissue only. As with lung cancer, the model performed significantly better at distinguishing non-cancerous tissue from cancerous tissue for both of these cancer types.

We discuss three possible explanations for the difference in performance on the test set for lung, kidney, and breast cancer classification. First, we consider that it could be naturally more difficult to differentiate between lung cancer types and kidney cancer types. Consultation with a human pathologist or replicating other models on kidney cancer could provide some insight on this. Secondly, the difference in the size of the dataset and the information density of the slides could have helped the model generalize better on the kidney task. Lastly, it could be more common to have more than one cancer type in a single cancerous tissue for lung than for kidney. There exists a type of rare cancer called adenosquamous carcinoma, with mixed TCGA-LUAD and TCGA-LUSC cells, that constitutes 0.4% to 4% of all lung cancer cases [Tochigi et al., 2011].

7 Future work

As we have mentioned already, the limited amount of time for this project meant that we were only able to test a few combinations of hyperparameters. For example, we did not end up trying nonlinearities such as scaled exponential linear units (SELU) or PReLU, and the data augmentation parameters could be further tuned. However, the benefit of setting up the code framework means that additional experiments should be relatively effortless, albeit time-consuming.

One possible area of exploration is the technique used for tile aggregation and prediction. For example, we believe that incorporating a weighted average for tile aggregation is a reasonable strategy since the image background is still present in some tiles. Another method we considered was to train another convolutional neural network on the tile probabilities after the original network was trained on a per-tile basis. This would allow the neural network to take advantage of the local correlations between tiles, such as identifying local clusters of tiles predicted to be a certain class. Unfortunately, we did not have time to implement these strategies, but we believe a simple 1-2 layer network for aggregation could further improve performance.

Another consideration that deserves more attention is the tiling and downsampling method. As the tiling and sorting steps took several days to complete, we only tried one tile size of 512×512 pixels. Furthermore, as mentioned briefly in section 3.1.1., we downsampled our tiles to 299 x 299 pixel before they were used by the network, reducing the resolution by about 40%. However, with more resources and available training time, one could explore whether maintaining the original input resolution would further boost performance. Or, with a GPU with more available memory, one could explore whether the tiling method is necessary at all. Instead of tiling, another approach could be to aggressively downsample within the convolutional neural network with pooling layers with larger kernel sizes. This would allow the network to perform the same reduction in resolution, but only after it has extracted information from the higher-resolution input. Furthermore, it would be better able to make compositional inferences about different regions in the slide as compared to our naive average aggregation method. However, the performance we achieved does suggest that the assumptions we made for using the tiling method were mostly valid.

As mentioned briefly in section 3.1.3, we explored concatenating metadata about the patient as additional information to the tissue slide input. However, as we were focused on obtaining a fair comparison with the results from Yu et al. [2016] and our advisor's work with Google's Inception V3, we did not end up fully exploring this method. Furthermore, it would imbalance the comparison in performance between cancers (for example, cigarettes smoked per day is much more informative for the lung cancer task than for kidney cancer). However, we believe that the best prediction system should be able to take advantage of all the available information a doctor may have, so this method clearly has potential.

We have shown that our method achieves state-of-the-art performance on the task of differentiating between cancerous and non-cancerous tissue and between cancerous subtypes on lung, kidney, and breast cancer. A natural next step is to continue expanding the method on even more types of cancer, such as uterine or liver. Even further, a major next step would be to apply this same method on the task of identifying cancerous mutations. However, as the amount of available data for mutations is much sparser, it may be challenging to reach the same levels of performance.

8 Acknowledgements

We would like to thank our advisors Aristotelis Tsirigos, Nicolas Coudray and Narges Razavian from the NYU School of Medicine, Applied Bioinformatic Laboratories for their constant guidance and support with this project.

This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. We are incredibly grateful for the GPU resources that they have provided to us, without which this project would have been impossible.

Appendices

A Model Architecture

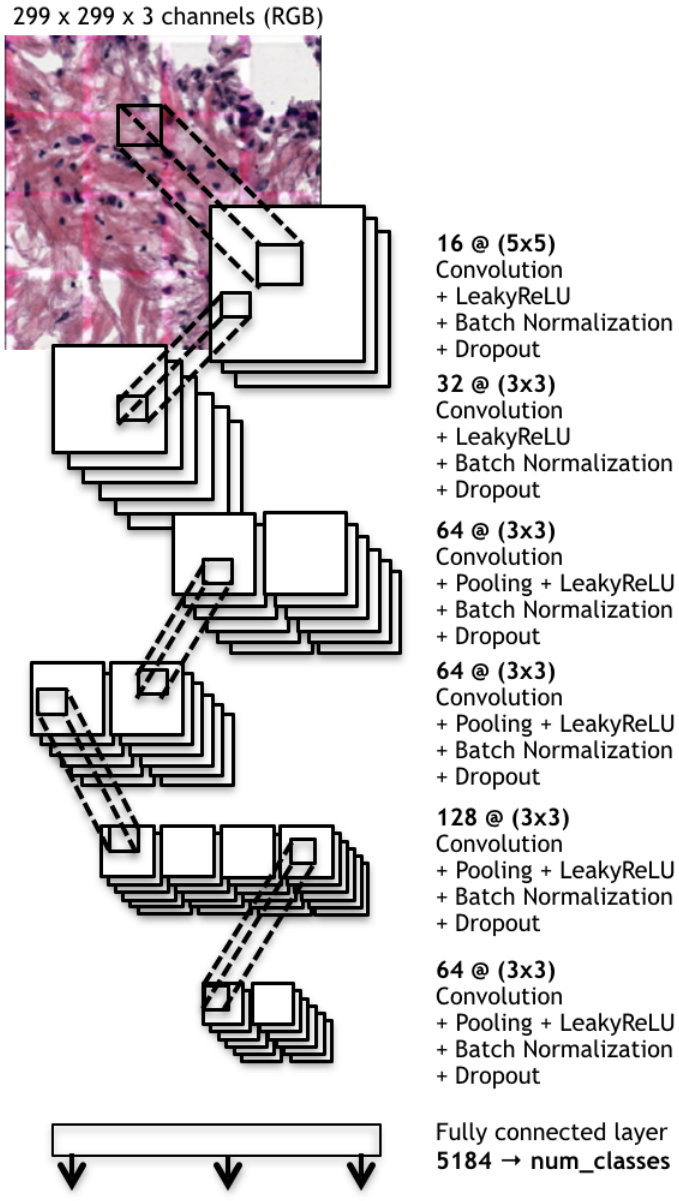
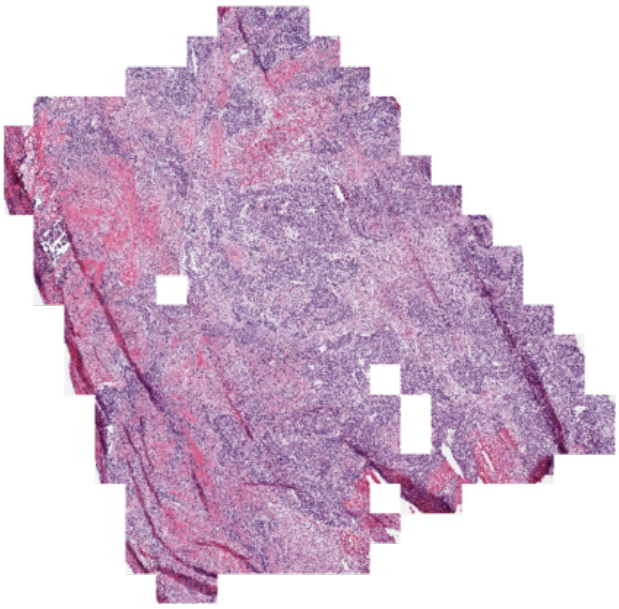
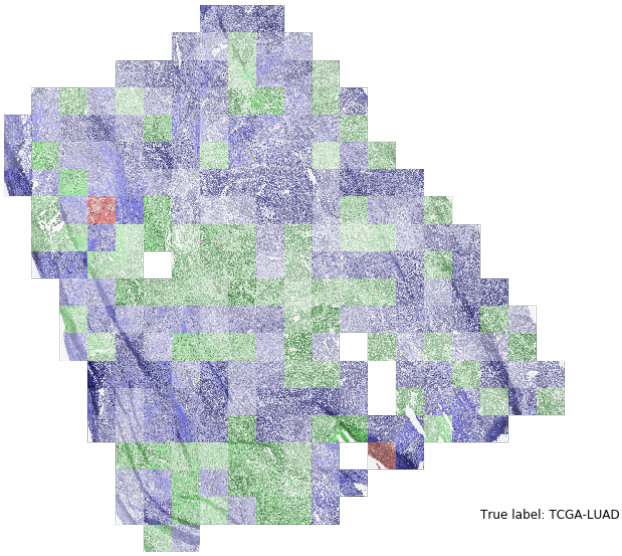


Figure 5: Network architecture

B Tile representation of Lung classification



(a) Original Tiles of a single image



(b) Classified Tiles

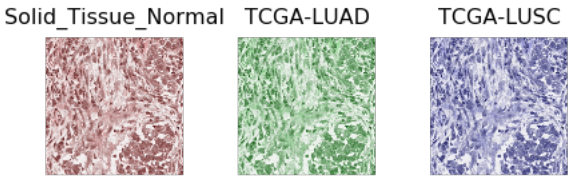


Figure 6: Tile representation of model output predictions

References

Batch normalization experiments on imagenet. URL https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md. Accessed: 2017-12-15.

Nci's genomic data commons. URL https://gdc.cancer.gov/. Accessed: 2017-10-01.

Rmsprop. URL http://climin.readthedocs.io/en/latest/rmsprop.html. Accessed: 2017-12-15.

Nicolas Coudray, Andre L Moreira, Theodore Sakellaropoulos, David Fenyo, Narges Razavian, and Aristotelis Tsirigos. Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning. *bioRxiv*, page 197574, 2017.

M Deshmukh, D Gonzalez, P Shah, M Pynegar, A Rice, S Popat, and A Nicholson. Refining the diagnosis and egfr status of non-small cell carcinoma in biopsy and cytologic material, using a panel of mucin staining, ttf-1, cytokeratin 5/6 and p63, and egfr mutation analysis. *The Journal of Pathology*, 220:S2, 2010.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Juneko E Grilley-Olson, D Neil Hayes, Dominic T Moore, Kevin O Leslie, Matthew D Wilkerson, Bahjat F Qaqish, Michele C Hayward, Christopher R Cabanski, Xiaoying Yin, Mark A Socinski, et al. Validation of interobserver agreement in lung cancer assessment: Hematoxylin-eosin diagnostic reproducibility for non–small cell lung cancer: The 2004 world health organization classification and therapeutically relevant subsets. *Archives of pathology & laboratory medicine*, 137(1):32–40, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015a. URL http://arxiv.org/abs/1512.03385.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015b.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yann Le Cun and Yoshua Bengio. Word-level training of a handwritten word recognizer based on convolutional neural networks. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 88–92. IEEE, 1994.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

Giorgio Scagliotti, Nasser Hanna, Frank Fossella, Katherine Sugarman, Johannes Blatter, Patrick Peterson, Lorinda Simms, and Frances A Shepherd. The differential efficacy of pemetrexed according to nsclc histology: a review of two phase iii studies. *The oncologist*, 14(3):253–263, 2009.

Michael Snyder. *Genomics and Personalized Medicine: What Everyone Needs to Know*. Oxford University Press, 2016.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

Naobumi Tochigi, Sanja Dacic, Marina Nikiforova, Kathleen M Cieply, and Samuel A Yousem. Adenosquamous carcinoma of the lung: a microdissection study of kras and egfr mutational and amplification status in a western patient population. *American journal of clinical pathology*, 135(5):783–789, 2011.

Kun-Hsing Yu, Ce Zhang, Gerald J Berry, Russ B Altman, Christopher Ré, Daniel L Rubin, and Michael Snyder. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. *Nature communications*, 7, 2016.