Intel and MobileODT Cervical Cancer Screening Kaggle Competition: Cervix Type Classification Using Deep Learning and Image Classification

Jack Payette
Stanford University
jpayette@stanford.edu

Jake Rachleff
Stanford University
jakerach@stanford.edu

Cameron Van de Graaf
Stanford University
camvdg@stanford.edu

Abstract

In this project, we attempted to create a deep learning model to classify cervix types in order to help healthcare providers provide better care to women all over the world. The problem is specified by the Kaggle Challenge found at `https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening`. *Classification of medical images is known to be a difficult problem for a number of reasons, but recent advancements in Deep Learning techniques have shown promise for such tasks. We experimented with a number of convolutional architectures before settling on residual neural networks with dropout and batch normalization to produces scores for each class, with loss calculated based on the multi-class logarithmic loss. The dataset, which was provided by Kaggle, consists of 1481 training images, 512 test images, and 4633 additional images that we used for training. Due to the small nature of the dataset, we used a number of data augmentation techniques. Through experimentation, we found that it is indeed very difficult for train a model from scratch that is general enough to solve this problem. While many techniques gave us marginal improvements in our performance, we were unable to reach a truly competitive spot on the leaderboard, likely as a result of overfitting such a limited dataset.*

1. Introduction

Cervical cancer is a deadly but highly treatable disease as long as it's detected in early stages and the correct treatment is administered. Healthcare specialists have broken cervixes down into three types. Women with Type 1 cervixes do not require screening beyond the standard procedure, while women with Type 2 and Type 3 cervixes require more time-consuming screening processes. Many healthcare providers in low resource areas of the world have neither the time nor the expertise to make cervix type classifications, so women all over the world are missing out on potentially lifesaving cancer screenings. For our project, we have entered the Kaggle competition hosted by Intel and MobileODT (found at `https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening`) to create an image classifier for different cervix types. The input to our classifier is a medical image of a cervix, and we use deep residual CNNs to output the probability of the cervix being in each of the three classes. Our hope is that we can create a system that can aid doctors around the world in classifying cervix type and in turn help women get the cervical cancer screening that could potentially save their lives.

2. Related Works

General Deep Learning Background

We saw the relevant body of literature for this work as being divided between three rather distinct subject areas. Firstly, we relied heavily on general literature pertaining to topics we had exposure to in this class - CNNs, dropout, batch normalization, etc. We mostly referenced papers that were suggested to us over the course of the class, Bengio's paper on dropout being a prime example. We would also put several of the external codebases into this section, including Tensorflow and associated documentation as well as the imgaug library. However, we leave most of the detailed explication of how these works influenced our own for the subsequent sections on data and methods.

Deep Learning in Medical Imaging Tasks

The second category of background literature is that pertaining to the application of deep learning in medical image classification, segmentation, etc. For this section, we began with a fairly comprehensive review by Litjens et. al from February 2017 - supplemented by more specific readings in areas of interest. One of the high-level findings was that the use of CNNs and other deep learning architectures in medical imaging tasks has exploded over the past several years, from a negligible number of papers published in the field during 2014, to around 50 in 2015, and reach-

ing around 500 in 2016. Classification for exam purposes, of the type we are attempting to accomplish here, was the third most common task addressed across all papers. However, we saw few papers that attempted classification of normal photographic images, with the vast majority of publications focusing on the common investigative modalities of MRI, microscopy, and CT. In terms of CNN architectures commonly applied to medical tasks, ResNet, VGG, and Inception were the most commonly utilized frameworks (Liu 2016). Transfer learning has often been applied, given the frequently limited quantities of labeled data available, though results are mixed between fine-tuning and feature extraction approaches (Antony 2016). More recent papers have achieved better results training models from scratch on datasets of $>$1000 images (Menegola 2016).

A particularly important point discussed in multiple papers was that it is often elements such as image augmentation and pre-processing that made the ultimate difference in performance more than simply adding more layers. Such domain-specific alterations can improve performance as much as 10%, and can be as specific as stain normalization and as general as elastic transformations (Ronneberger 2015). Though we couldn't find much evidence of CNNs or other deep learning architectures being applied to cervix images, the closest analogue we could find was classification of dermatological conditions by standard photographic images, where authors were able to achieve human-level performance using Inception-v3 (Esteva 2017).

There is also a substantial literature devoted to the unique problems faced in applying deep learning to medical imaging, many of which we encountered over the course of our project. For instance, many papers cite the dangers of falsely labeled images or poor quality data, which influenced our approach to image curation and removal (Armato 2011). Class imbalance - oftentimes resulting from the relative rarity of physiological types or medical conditions in the general population - is another issue that can be severely detrimental to system performance; this has been addressed through rebalancing and augmentation specifically applied to minority classes (Pereira 2016). It is also difficult to discern whether a classification paradigm is always the optimal approach as contrasted with a segmentation or predicted bounding box approach (Sirinukunwattana et al. 2016). Generally Litjens et. al were optimistic about applications of deep learning in medical imaging, with the aforementioned caveats.

Approaches to Cervix Segmentation

While deep learning models have not (so far as we know) been applied to the problem of cervix classification and segmentation prior to the launch of this Kaggle competition, there is some literature describing other statistical methods for accomplishing these tasks. The first paper we could find on this topic was by Gordon and Zimmerman from 2004, which introduced a segmentation paradigm based on statistical properties of the texture and color profile of different parts of cervix images (Gordon 2004). This work was extended slightly by Srinivasan et al. in 2005, where similar techniques were applied to cervix cancer detection specifically. Finally in 2010 Xue et al. developed the most fully fledged suite of cervix analysis tools for detection, segmentation, and classification (Xue 2010). Unfortunately, this toolset was implemented using Java applets more suited to the clinical environment where smaller volumes of images were being processed. With this in mind, we adapted code written by Kaggle user Chattob (link to code in references) that implements the algorithms in the above papers to generate cropping circles and bounding boxes. Chattob's code in turn was based on a follow up to the 2004 Gordon paper (Greenspan 2009).

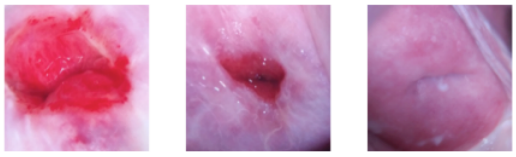
3. Datasets and Features

The dataset that we used for this project was the one provided by Kaggle for this competition. A breakdown of the dataset can be seen in Figure 1.

Dataset	Type 1	Type 2	Type 3
Train	250	781	450
Additional	1189	1558	1886
Test	512 (Unlabeled)		

Figure 1. A breakdown of the Kaggle datatset

To generate our Validation split, we used 50% of the Train images for our Training Set and 50% of our Training images for our Validation Set. We used the additional data as part of our Training Set as well. The reasoning for splitting our data like this is that the Train data consists of high quality images, all from different patients, with no duplicates. The additional data, on the other hand, consists of lower quality images, many of which come from the same patients, and most importantly, contains duplicates. For this reason, we did not use any of the additional images in our validation set, as this would create the possibility of a duplicate image being in both the Training and Validation Set. This would artificially increase the performance of our model on the Validation Set, meaning that the Validation Set would no longer serve as a good proxy for the Test Set.



(a) (b) (c)

Figure 2. Example images of each cervix type. Image (a) is Type 1, image (b) is Type 2, and Image (c) is Type 3

The sizes of the images in this dataset vary greatly, from as small as 480 x 640 to as large as 3096 x 4128. As a result, we used image resizing techniques so that all of our data was 224 x 224 by the time the model received it as input. On the training images, we resized the image so that the shortest side was 256 pixels, and then took a random 224 x 224 crop. This is a common technique for data augmentation. For validation and test images, we resized them so that the side was 224 pixels, and we took the crop from the center of the other axis. We chose this exact algorithm because there are images in both landscape and portrait format. As suggested by TA Ben Poole, we did not subtract the mean image from each image, as this is not considered best practice for medical imaging. We did, however, divide every pixel value by 255 so that every pixel value was in the range of 0 to 1.

Due to the limited nature of our dataset, we employed a number of different data augmentation techniques. The first was that we upsampled from the underrepresented classes so that the data distribution was even among all of the classes. While we believe that this is the "correct" way to train this classifier, we noticed that our models performed worse when we did this. We believe that this is because the underlying distribution of the test set more closely matches that of the training set, rather than that of an even distribution.

We also utilized a number of more substantive data augmentation techniques. The first was an auto-cropping algorithm, which draws a bounding box around the region of interest (i.e. the actual cervix) and crops the image to contain only this region. As noted in the related works section, the code was adapted from Kaggle user Chattob (link to code in references) who in turn made use of algorithms developed in (Greenspan 2009). The general principle underlying this approach is a statistical measure based on the color and texture of cervix tissue. The crop is then made on the basis of an energy minimization algorithm taking in the aforementioned statistical measure for a given region of the image. An example of the input output behavior can be seen in Figure 3.

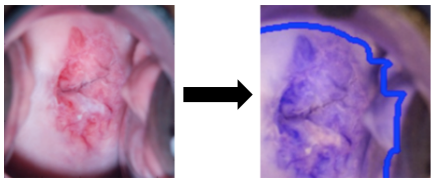


Figure 3. An example of the input-output behavior of the image auto-cropping algorithm. Red-blue transform is an artifact of the cropping procedure that doesn't affect final results.

In addition to the auto-crop functionality, we also experimented with a range of different transforms of the underlying images. This type of augmentation was implemented using the imgaug library (User:aleju, see references for link) and included the following specific transformations: horizontal flip, vertical flip, sharpening, embossing, and edge detection. While the first two are fairly standard in deep learning, we chose the latter four because of the intuitive notion that they emphasize in various ways the cell type boundary that is crucial for distinguishing between cervix types. To put it rather plainly, it is crucial for the system to be able to distinguish between the bright red vs. pink areas visible in Figure 3, as well as to reliably determine the area of the bright red region. Our chosen augmentations intuitively seem to accomplish this objective. Examples of these techniques can be seen in Figure 4. During training, we experimented with a range of hyperparameters (most pertinently the alpha value, which controls the strength of the effect) in order to create fairly dissimilar images that still contained the salient features in the dataset. Overall we found that moderately high alpha values in the range of .6-.8 seem to be most effective for producing qualitatively desirable training images. When we moved to our final testing, we applied the flips with probability .5 each and the remainder of the transformations with probability .3 to every image. We explore the empirical results of different augmentation procedures in the Experiments section.

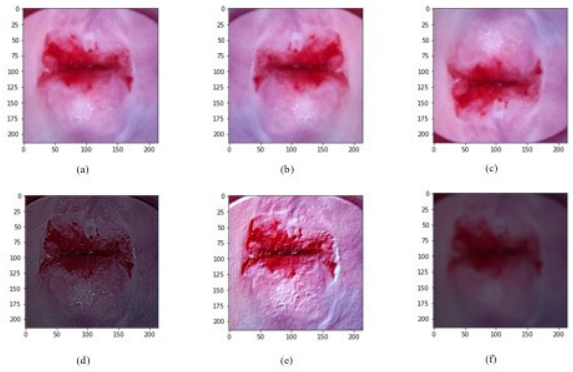


Figure 4. Images illustrating different image transformations. Image (a) has not been altered, image (b) has been flipped left-right, image (c) has been flipped up-down, image (d) has been sharpened, image (e) has been embossed, and image (f) has had edge detection applied

4. Methods

The deep learning framework we used for this project was TensorFlow-GPU version 1.2.0rc0. Our general method for classifying images was to build a deep residual network, employing batch normalization for increased gradient flow, dropout to reduce overfitting, and the Adam optimizer. We used multi-class logarithmic loss as our loss function.

Residual Network

Deep Neural Networks are more expressive than their shallow counterparts and should be able to produce better results, but they are notoriously difficult to optimize. As a result, until 2016, even state of the art Convolutional Neural Nets only went as deep as 19 layers. One of the biggest breakthroughs in the history of image classification came in 2016, when He et al. introduced the residual network. Residual networks are built off the basis of a shortcut connection. The idea behind these shortcut connections is some input i goes through an activation layer, and produces some output o. Rather than the input to the next layer be o, instead we use $i + o$ as the input to the next layer. In a sense, the input shortcuts this layer entirely. An example of a shortcut connection can be seen in figure 5.

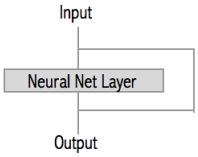


Figure 5. An example of a shortcut connection used in residual networks

Residual networks make heavy use of shortcut connections, and they are hugely important to increasing the depth of neural nets. They allow for layers to become identity mappings (in other words, o is simply zero), so the input is able to flow freely through layers that may be extraneous. Further, because the residual blocks allow gradients to backpropagate more freely, the vanishing backwards gradient flow that was in the past a huge problem for deep networks is far less of an issue. He et al. showed that residual networks up to depth 152 were reasonable to optimize, and less complex than the 16 and 19 layer networks that were considered state of the art at the time. For our project, we decided to implement deep residual networks so that we could take advantage of these properties while generating an expressive model.

We used two different types of residual models for our project. The first, a 32 layer residual CNN makes use of shortcut connections every two layers. After a large 7x7 convolutional layer that increases the input depth using 64 filters and a 3x3 max pooling that reduces the height and width of the input, the network has a number of residual blocks. Each residual block has two 3x3 convolutional layers, with a shortcut connection from the input of the first layer to the output of the second. These residual blocks are interleaved with max pooling layers for downsampling. Whenever the input height and width are reduced by a factor of two during a max pooling layer, the number of filters is doubled in order to keep spatial dimensionality. At the end of the network, there is a global average pooling layer where each activation map is reduced to a single number (the average of all of its units). Finally, there is a single fully connected layer to attain the output size of 3. An example of the complete architecture can be seen in figure 6.

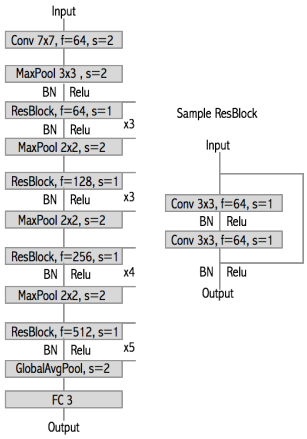


Figure 6. The complete architecture for our 32 layer residual network, along with an example residual block

He et al. also proposed a different type of residual block, which reduces the number of parameters in a block. The architecture is very similar to the one described above, but residual blocks have 3 convolutional layers: a 1x1 convolutional layer with fewer filters (to downsample the input), a 3x3 layer with the same number of filters, and then another 1x1 layer with the number of filters of the original input size to upsample and allow for the shortcut connection. In theory, this requires far fewer parameters than the architecture proposed above, and would allow for even deeper models before memory becomes an issue. Our 53 and 101 layer residual neural nets adopt this type of residual block. The architectures for these models can be seen in figure 7.

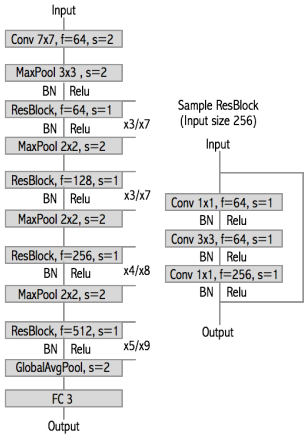


Figure 7. The complete architectures for our 53 and 101 layer residual networks, along with an example residual block

We were interested to see how the downsampling block of the 53 and 101 layer Neural Net would compare to the more memory needy block of the 32 layer Neural Net. Its interesting to note that the only difference between the 32 and 53 layer networks is the type of residual block. Our hypothesis was that, regardless of the depth, each of the neural nets would be able to converge relatively easily because of the shortcut connections. Our belief was that even the 32 layer network was likely too expressive for such a small dataset, but that each of the networks would perform reasonably well because the unnecessary blocks would simply become identity mappings.

Batch Normalization

One of the biggest difficulties with training deep Neural Networks is that the distribution of the inputs to each layer is constantly changing during training time. This is problematic, because many of the nonlinearities used in Neural Nets do not perform well unless the input distribution is in a very specific form. In the past, this has required very precise tuning of learning rates, and very specific initialization of weights, and even then deep networks were incredibly difficult to train. Batch Normalization, as proposed by Ioffe and Szegedy in 2015, is a technique that normalizes inputs to each layer during training time. More specifically, for each batch of inputs, the layer subtracts the batch mean and divides by the batch standard deviation. The layer then scales and shifts the input with a learned parameter, passing this normalized input to the next layer. The layer also keeps track of the running mean and variance, and uses these to normalize input batches at test time. The mathematical formalism for batch normalization can be found in figure 8.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 8. The mathematical formalism for Batch Normalization. Taken from Ioffe, Szegedy 2016

Because we use deep Neural Networks in our project, batch normalization was an absolute necessity. Without batch normalization layers before every activation, even the shallowest of our models was unable to learn, as activations became saturated as a result of bad initialization and imperfect learning rates. After batch normalization was added, even the deepest models were able to learn relatively efficiently with a wide range of learning rates.

Dropout

Dropout is a regularization technique proposed by Srivastava et al. in 2014. It has proven to be a very powerful technique to reduce overfitting, a problem that was plaguing our networks. Despite its power, its implementation is simple. The way that it works is that neurons in hidden layers are dropped (along with their connections) with some probability at training time. Activations and gradients do not flow through dropped neurons, leaving a slightly less expressive model on each pass. While its not obvious at first why this would help reduce overfitting, when we take a closer look at the hidden layers in deep networks, we can see that downstream neurons can learn close dependencies with specific upstream neurons. These specific dependencies generally help to classify train images but do not generalize well to test images. By randomly dropping neurons in each layer for every input, we can smooth out some of these close dependencies, as the direct connection between the neurons may be severed. As a result, dropout has proven to be a very effective technique at reducing overfitting. While He et al. did not use dropout in their proposal of Resnet, our dataset was so small that we were unable to find any relatively complex models (let alone deep residual model) that did not overfit. As a result, we put a dropout layer with a drop probability of 0.4 after every non-linearity in our model. While this slowed our training speed, it did slightly increase generalization, and allowed us to use more expressive models without egregious overfitting.

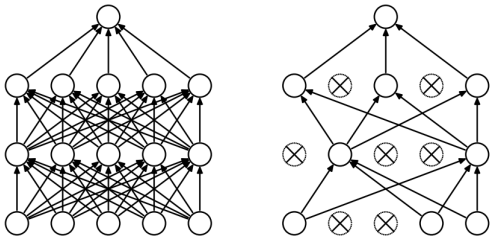


Figure 9. The network on the left is a vanilla Neural Network without dropout. The network on the right is the same network with dropout applied. Taken from Srivastava et al. 2014

Multi-Class Logarithmic Loss

The loss function that the Kaggle competition uses to judge submissions is multi-class logarithmic loss. As a result, we used the same loss function to train our neural networks. The equation for multi-class logarithmic loss is:

$$\text{loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\ln(p_{ij})$$

Where N is the number of images in the test set, M is the number of classes, y_{ij} is an indicator variable for if image i's ground truth label is class j, and p_{ij} is the probability (submitted from our system) that image i is of class j.

Adam Optimizer

For our optimizer, we chose the Adam optimizer proposed by Kingma and Lei Ba in 2015. This optimizer is similar to vanilla stochastic gradient descent in that its a first order, gradient based algorithm used to optimize stochastic objective functions. It is a hybrid of the popular RMSProp and AdaGrad optimizers, and it has the attractive properties of both (it works well with sparse gradients and does not require a stationary objective function). Further, the optimizer has the convenient property of naturally reducing the step size as training proceeds. The algorithm works by keeping track of not only the weight vector, but also biased estimates of the first and second moments and the time step. On each training example, the gradient is calculated and used to update the first and second moment. From these moments, and unbiased estimate of the first and second moments are calculated, and these are used along with the time step and the weight vector to make an update to weight vector. The pseudocode for this update rule can be found in figure 10.

$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 10. Pseudocode for the Adam optimizer. Taken from Kingma and Lei Ba, 2015

We chose Adam as our optimizer for a few reasons. Because of its momentum property, its less likely to get stuck at local optimum like vanilla stochastic gradient descent. Also, because it uses bias correction, correct initialization is less important, as the algorithm naturally takes care of a number of the issues that arise when other update rules are used. Further, because of the natural step size annealing, the Adam optimizer generally requires less tuning, a very attractive property when compute time is so limited and training takes as long as it does (due to the depth of our networks). Finally, (and most importantly) Kingma and Lei Ba showed that Adam has been shown to converge more quickly than other update rules.

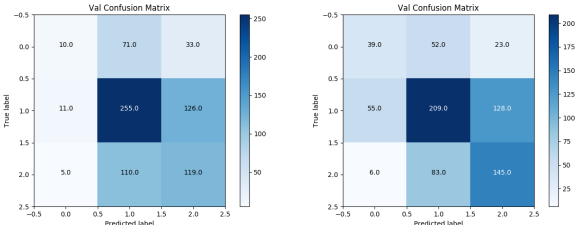
5. Experiments, Results, and Discussion

Baseline Model Selection

In order to find our best model, we ran a few distinct experiments. First, we had to select our top models from a baseline level. To do so, we ran five models: basic softmax, 32 layer CNN without residuals, and residual nets of depth 32, 53, and 102. Here, we found that 32 and 53 layer residual nets performed best, as seen in the chart below. Please note, we also trained a 101 Layer Residual CNN, but it was too deep to be useful, and continually only picked Type 3 as the label for images.

Model	Val Loss	Val Accuracy	Val F1 Score	Test Loss
Softmax	.983	.526	.437	-
32 Layer CNN	.953	.561	.509	0.948
32 Layer Residual CNN	.948	.580	.573	0.923
53 Layer Residual CNN	.930	.554	.514	0.911

At first, these statistics may be confusing. Softmax appears to be nearly as accurate as our basic convolutional network. However, it is important to keep in mind that the distribution of test images is not perfect, so guessing only Class 2 results in an accuracy a bit above 50 percent, since it is the most popular class. For this reason, F1 is a much more valuable metric. Furthermore, the actual competition is evaluated on validation loss, so our most important metric is val loss. To visualize this importance, let us examine the confusion matrix for the Softmax and 32 Layer CNN models, which have similar accuracy but much different F1.



(a) Softmax (b) 32 Layer CNN

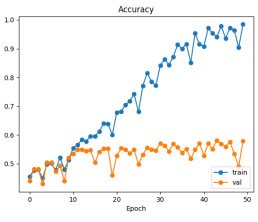
As you can see above, the Softmax model has large success rates classifying Type 2 cervices leading to high accuracy, but also commonly confuses Type 1 and Type 3 for Type 2. This is why accuracy is lower. For the 32 Layer CNN model which has similar accuracy numbers, we see that F1 is so much smaller because it does a better job on both Type 1 and Type 3.

Looking at all recorded statistics, we believe that our 32 Layer CNN was the best baseline model, since it had low val loss and high relative F1. This supports our claim that residual nets would perform better than their non residual counterparts. It is interesting to note that each of the convolutional neural networks initially has poor F1 in training because it only guesses that images are of the class Type 3. This is surprising since the most frequent class is Type 2, yet this result was found consistently across training. We believe that the function landscape has a local minimum here, and it takes a few iterations for the model not to get stuck in said minimum.

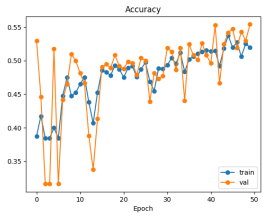
We did not submit Softmax to the competition since its statistics were not good enough to merit a submission, and thus it does not have test data.

Regularization Results

In order to understand the learning process, we looked at the training and validation accuracy graphs for our baseline 32 and 53 layer residual networks:



(c) 32 Layer Residual CNN



(d) 53 Layer Residual CNN

Here, we see a clear difference. The 32 layer network overfit the data, while the 53 layer network seemed to continue to learn. We diagnosed two potential issues here. First, the 32 layer network needed regularization. Second, the 53 layer network needed to train longer. In order to both increase validation results for the 32 layer network, and allow the 53 layer network to train longer without overfitting, we decided to implement dropout. This strategy was not implemented in the original residual network paper, but due to small datasize and powerful models, we wanted to make sure some regularization existed in our models. Thus, we reran these two networks for longer, now with regularization.

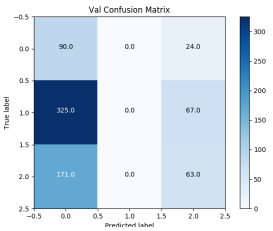
In order to combat this, we must implement some sort of regularization for our model. Here, we decided to use dropout as our form of regularization. The results of our experiment are shown below:

Model	Val Loss	Val Accuracy	Val F1 Score	Test Loss
32 Layer Residual CNN	.948	.580	.573	0.923
53 Layer Residual CNN	.930	.554	.514	0.911
32 Layer Residual CNN with Dropout	.948	.564	.567	-
53 Layer Residual CNN with Dropout	1.072	.499	.467	-

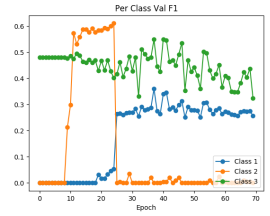
As you can see from the results, dropout had an inconclusive effect on our model. The validation loss, accuracy, and F1 score are all quite similar for the 32 layer residual model both with and without dropout.

The 53 layer model was hindered by dropout - the final confusion matrix and per class F1 scores shown below explain its issues with training:

As the reader can see on the right, the 53 Layer Residual network, like all of our other networks, only can guess Type 3. Then, after around 10 epochs, the model only guesses Type 2 or Type 3. Finally, it stops guessing Type 2, and only guesses Type 1 or Type 3. We plot F1 instead of accuracy here to show the effect of false positives. This divide is easily visualized in the graph to the left. These figures are important since they illustrate a common issue with our



(e) Confusion Matrix at Epoch 70



(f) Per Class F1 over 70 Epochs

dataset - the function landscape appeared to have local optima at guessing either only one type (usually Type 3), or any set of two types. Dropout further pushed our 53 Layer Residual model into these optima, which was an unexpected result seeing as it usually causes better generalization.

Data Augmentation Results

After we found that dropout had somewhat negligible effect on preventing overfitting with our 32 layer model, we realized that the issue could be our small dataset. Thus, we began a new experiment which combined dropout and data augmentation on our 32 layer network. This gave us a larger data set and regularization, which should combine to prevent overfitting. We augmented the data as described in previous sections.

Model	Val Loss	Val Accuracy	Val F1 Score	Test Loss
32 Layer Residual CNN	.948	.580	.573	0.923
32 Layer Residual CNN with Dropout	.948	.564	.567	-
32 Layer Residual CNN with Dropout and Data Augmentation	.936	.576	.551	-
32 Layer Residual CNN with Data Augmentation	.879	.588	.578	.873

As you can see, surprisingly enough the model generalized better when only using data augmentation. This led to our best self trained model with a test accuracy of .873. These results are somewhat unsurprising, since the original residual networks did not use dropout.

Transfer Learning

We also experimented with transfer learning. As stated above, it was our hypothesis (based on our reading of past papers, specifically Menegola 2016) that transfer learning would be difficult to employ given the substantial differences between our dataset and more common sources such as ImageNet. To test this hypothesis, we downloaded pre-trained weights for Inception v3 that had achieved 95% test accuracy on Imagenet. This model is part of the default Tensorflow source code at `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining`. The weights for the final fully-connected layer are deleted and subsequently retrained using SGD. When running 2000 iterations on the un-augmented, non-additional dataset (approx. 15 epochs), we observed a high validation accuracy of .61 and a low validation loss of .87. When we ran

4000 iterations on the full set of auto-cropped images, we observed a low validation loss of .8 and a high validation accuracy of .7. The test accuracy for the latter was around .6.

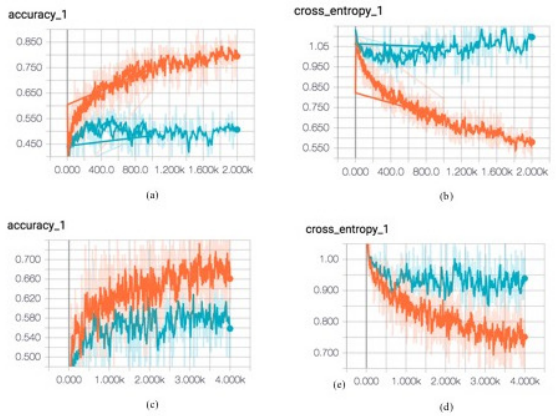


Figure 5. Transfer learning on Inception v3. Plots (a) and (b) are accuracy and loss plots for the limited dataset and (c) and (d) are accuracy and loss plots for all images with auto-crop applied. Orange curves represent train data and cyan represent validation data

6. Conclusion and Future Work

At the beginning of this project, we set out to train our own neural networks to attack the problem of cervix type classification. From our readings of previous papers on the application of deep learning in medical imaging, as well as previous algorithmic attempts at cervix classification, we recognized that this was a difficult task. As we began training models, we came to realize that the relative lack of data and high visual similarity between classes were particularly difficult challenges. In an attempt to combat these, we employed a range of statistical cropping and augmentation techniques.

Over the course of the project, we experimented with a range of different CNN architectures, mainly based on the ResNet model, and found that the 32 Layer Residual Network with Data Augmentation performed the best of self trained models. Though we did not conduct a thorough investigation, we found that transfer learning using an Inception v3 model trained on Imagenet yielded more promising results. In analyzing these findings, we found that we underperformed the top scorers on the Kaggle leaderboard fairly significantly, though we generally used similar techniques. Moreover, in conversations with other students in the course we noticed that even when applying the same models there was a high variance in performance within and between groups. There are several possible explanations for such discrepancies, including a more successful hyperparameter search or a better data curation strategy.

Since the fundamental goal of this project was to learn about training CNNs in the real world, we focused heavily on techniques used to train networks from scratch. However, if we had more time, we would conduct a more thorough comparison between models trained from scratch and those initialized using transfer learning. We would also pursue a more extensive understanding of the flaws in our model beyond the use of classification errors - we could use saliency maps or minimum fooling images for example. In the final analysis, we feel confident that while this is a difficult task, given appropriate time and resources a deep learning model could conceivably approach trained human level performance.

7. References

1. Khan J, Wei1 J, Ringner M, Saal L, Ladanyi M, Westermann F, Berthold F, Schwab M, Antonescu C, Peterson C, Meltzer P. (2001). Classification and Diagnostic Prediction of Cancers Using Gene Expression Profiling and Artificial Neural Networks. http://www.nature.com/nm/journal/v7/n6/abs/nm0601_673.html
2. Kuruvilla J, Gunavathi K. (2014). Lung Cancer Classification Using Neural Networks for CT Images. http://www.sciencedirect.com/science/article/pii/S0169260713003532
3. Geras K, Wolfson S, Kim S, Moy L, Cho K. (2017). High-Resolution Breast Cancer Screening with Multi-View Deep Convolutional Neural Networks. https://arxiv.org/abs/1703.07047
4. He K, Zhang X, Ren S, Sun J. (2016). Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385
5. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
6. Ioffe S, Szegedy C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167
7. Glorot X, Bengio Y. (2010). Xavier Initialization. http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
8. Kingma D, Lei Ba J. (2015). Adam: A Method For Stochastic Optimization. https://arxiv.org/pdf/1412.6980.pdf
9. Xue Z, Long R, Antani S, Neve L, Zhu Y, Thomas G. (2010). A Unified Set of Analysis Tools for Uterine Cervix Image Segmentation. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2955170/
10. Srinivasan Y, et al. (2005). A probabilistic approach to segmentation and classification of neoplasia in uterine cervix

images using color and geometric features. International Society for Optics and Photonics. https://lhncbc.nlm.nih.gov/files/archive/pub2005015.pdf

11. Gordon S, Zimmerman G, Greenspan H. (2004). Image segmentation of uterine cervix images for indexing in PACS. http://ieeexplore.ieee.org/abstract/document/1311731/
12. Litjens, G., Kooi, T., Ehteshami, B., et al. A Survey on Deep Learning in Medical Image Analysis. 2017. https://arxiv.org/abs/1702.05747
13. Liu, Y., Gadepalli, K., Norouzi, M., Dahl, G. E., Kohlberger, T., Boyko, A., Venugopalan, S., Timofeev, A., Nelson, P. Q., Corrado, G. S., Hipp, J. D., Peng, L., Stumpe, M. C., 2017. Detecting cancer metastases on gigapixel pathology images. arXiv:1703.02442
14. Antony, J., McGuinness, K., Connor, N. E. O., Moran, K., 2016. Quantifying radiographic knee osteoarthritis severity using deep convolutional neural networks. arXiv:1609.02469.
15. Menegola, A., Fornaciali, M., Pires, R., Avila, S., Valle, E., 2016. Towards automated melanoma screening: Exploring transfer learning schemes. arXiv:1609.01228.
16. Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., Thrun, S., 2017. Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 115118.
17. Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In: Med Image Comput Comput Assist Interv. Vol. 9351 of Lect Notes Comput Sci. pp. 234241
18. Armato, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, et al. 2011. The lung image database consortium (LIDC) and image database resource initiative (IDRI): a completed reference database of lung nodules on CT scans. Med Phys 38, 915931.
19. Pereira, S., Pinto, A., Alves, V., Silva, C. A., 2016. Brain tumor segmentation using convolutional neural networks in MRI images. IEEE Trans Med Imaging.
20. Sirinukunwattana, K., Raza, S. E. A., Tsang, Y.-W., Snead, D. R., Cree, I. A., Rajpoot, N. M., 2016. Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. IEEE Trans Med Imaging 35 (5), 11961206.
21. Greenspan H, Gordon S, Zimmerman G, et al. 2009. Automatic Detection of Anatomical Landmarks in Uterine Cervix Images. IEEE TRANSACTIONS ON MEDICAL IMAGING. https://www.researchgate.net/publication/24041301_Automatic_Detection_of_Anatomical_Landmarks_in_Uterine_Cervix_Images
22. User:Chattob. 2017. Cervix segmentation (GMM). https://www.kaggle.com/chattob/cervix-segmentation-gmm/notebook
23. User:aleju. 2017. imgaug: Image augmentation for machine learning experiments. https://github.com/aleju/imgaug