DATS 6203 Section 10: Machine Learning II

Instructor: Dr. Amir Jafari

Final Project Report: Acute Lymphoblastic Leukemia Classification (Kaggle Competition) | Group Final Report

Dylan Saez, Christopher Taylor, Stefani Guevara

December 6, 2021

Table of Contents

Introduction

In this project, we experiment with several convolutional neural networks (CNNs) in order to find the model that best distinguishes between normal and leukemia blast (cancer) cells. Building a model to accurately classify which type of cell is shown to the computer is extremely important when it comes to diagnosing patients correctly in the medical field.

An overarching interest in becoming more familiar with CNN networks for image classification and applying machine learning skills to a healthcare-related question was the main motivation for choosing this application. The dataset used to answer this question consists of 15,135 images from 118 patients with two classes: Normal cell and Leukemia blast (cancer) cell. 10,661 of these images are used in the training set, 1,867 on our validation set; 2,586 are used on the test set kept by the hosts of the competition. The number of images translates to over 10 GB of data.

In order to show how the best model was selected, information will be provided on the numerous experiments performed consisting of different image classification models, their results, and how they fell short compared with the best model.

PyTorch, a machine learning framework, is used to implement the networks built since there are pretrained networks that we will use as a baseline comparison. F-1 Score and Cohen will be used to measure the performance of the network.

Models experimented on this dataset include VGGNet16, VGGNet19, EfficientNetB7, EfficientnetB3, EfficientnetB4, EfficientnetB5, EfficientnetB0, GoogleNet, Densenet161, DenseNet121, DenseNet169, and ResNet152.

To obtain sufficient background on applying the chosen networks, papers have been included to describe the original intention of the neural network, compare to baseline scores, and describe any features modified for this specific problem.

Description of the Dataset

As stated above, the dataset consists of 15,135 images from 118 patients with two classes: Normal cell and Leukemia blast (cancer) cell. The number of images translates to over 10GB of data.

According to the section titled, *About this dataset* on the Kaggle competition where this application was introduced, "These cells have been segmented from microscopic images and are representative of images in the real-world because they contain some staining noise and illumination errors, although these errors have largely been fixed in the course of acquisition". Staining noise indicates a better visualization of the different cells by negative staining, an important tool for the study of biological molecules. The paper where negative staining is used in preparation for visualization modification, Ohi, Li, Cheng, and Walz (2004) indicates, "Negative staining, the embedding of a specimen in a layer of dried heavy metal solution, was introduced early on a as a quick and easy specimen preparation technique that significantly increases the specimen contrast." On the other hand, illumination errors are removed to "support colour-based image recognition and stable tracking (in and out of shadows)," Finlayson (2018).

This dataset can be trusted to be an accurate representation of Normal and Leukemia blast (cancer) cells since it was reported in the Kaggle competition that an expert oncologist annotated the ground truth labels of the data.

Experiment

Dylan Saez's Experimentation

Saez Experimentation Overview

Identical to Stefani's statement, "After Chris's finalized initial code base model script, I took to experimenting with different pretrained architectures provided by the torchvision models package."

Before implementing the different VGGNet and EfficientNet models, I gathered some background information on these two networks by reviewing the papers that introduced them.

The CNN model, VGG Network, was proposed by Karen Simonyan and Andrew Zisserman in their paper, “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The main idea from this paper is that adding more (convolution) layers, the model performance increases. Specifically, increasing depth (layers) using an architecture training with 224x224 RBG images and 3x3 kernel (convolution filters), the performance of the model increases. Using non-linear activations in between layers allows the function to become more “discriminate” and decreases the number of parameters by a 3x3 non-linear decomposition. This paper also introduced a range of depth 16-19 for layers.

Another CNN model, EfficientNet, was introduced by Mingxing Tan and Quoc V. Le in their paper, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. Model scaling deals with three main components: depth, width, and resolution to improve the performance of the model.

Depth: the number of layers in a network.

Width: the number of neurons in a layer, filters in a convolutional layer

Resolution: (height)(width) of the input image.

By scaling up on the baseline models MobileNets and ResNet, Tan and V. Le demonstrate the effectiveness using a compound coefficient of depth, width, and resolution.

Model 1: VGGNet16

Inspired by an attempt posted to this Kaggle Competition and previous work using this model in a completely different application, I tested VGGNet (16) and VGGNet(19). I did not attempt the other versions since my scores were low and I wanted to move onto different models. For data augmentation, I applied random horizontal and vertical flips with probability of 0.5. The input shape was the same as that of the notebook’s (128, 128, 3). The model was in the mid-high 50’s. I used 20 epochs, the notebook used 500. Batch size was 512, learning rate was 0.001, and no class imbalance code was written. Accuracy scores ranged from mid-high 50’s for VGGNet16, with a low accuracy score of 54.9% to 58.3%.

Model 2: VGGNet19

With 20 epochs, I reached the high 50's: with a low accuracy score of 57.4% % and a high of 59.2%. The same data preparation from (16) applies here. As stated above, I did not attempt the other VGG versions since my scores were low and I wanted to move onto different models. Surprisingly, for the other application I applied this model to, I received scores in the high 70's. I would hypothesize that my score was so low because of the data augmentation - there may be a lot more to add in this section of the code.

Model 3: Efficientnet_B4

Chris tested the largest EfficientNet Model (B7) and Stefani tested the EfficientNet-B0 (baseline network). I tested two different iterations of the network (B4) and (B5). The main idea of EfficientNet is tuning a model scaling with the three components: depth, width, resolution to achieve maximum model performance. For data augmentation, I applied horizontal flip, vertical flip, brightness, and contrast specifications. The input size of the image was (300,300). Batch size was 32, epochs assigned to 7, and learning rate of 0.001. For class imbalance, I used oversampling for the minority class. Accuracy scores ranged from mid-60's for EfficientNet(B4) with a low accuracy score of 62.3% and a high of 63.8%.

Model 4: Efficientnet_B5

Following the same specifications as B4: With 7 epochs, I reached up to low-60's on the test set, with a low accuracy score of 61.1% and a high of 62.8%.

Christopher Taylor's Experiments

Model 1 : Initial Code Base Model

For the initial code setup, I used my code from exam 2 as the starting point. However, I had to overhaul parts of the code to work for the new project. I mainly overhauled the data preprocessing, dataset class, and the data loader functionality of the code base. I did reference one of the other Kaggle competition code files. They had used the Tensorflow framework, but it was still a good reference for how to best load the data effectively. The model for this file was a one layer convolutional network as I just want to ensure the file would run all the way through. We then decided to focus on pretrained models instead of building our own architecture. This base model had an accuracy of 63.6%.

Model 2: Basic Data Augmentation with DenseNet121

I used the densenet121, we were looking to test various pretrained models. Stefani and I wound up choosing the same pretrained models. DenseNet architecture follows a pattern of dense blocks of four convolution layers and then transition blocks of one convolution and average pooling. The data augmentation used for this model was just random crop and random horizontal flip. No efforts to work on the class imbalance was done for this model. The model's best performance was 65.3% accuracy.

Model 3: Albumentation with DenseNet161

Albumentation is a python library that focuses on data augmentation and gives you more options and control over your data augmentation than torchvision. The main desire to use this was the ease of adding probability of applying augmentation while still being computational efficient. This was the initial testing model for albumentation so I used a similar densenet model and kept my data augmentation to random cropping and flipping. This model performed at 70.8% accuracy.

Model 4: Albumentation with GoogleNet

In looking at different pretrained models I tested googlenet. This network has 22 layers within a CNN model. In addition to testing this model, I began to test different albumentation probabilities, and added having a vertical flip into my augmentation package. This model had an accuracy of 72.3%.

Model 5: Albumentation with Efficientnet_B7

With testing other pretrained models. I tested the largest Efficient model. EfficientNet is a CNN model that focuses on balancing network depth, width and resolution to enhance performance. This had a long run time, and I only ran it for one epoch. It had an accuracy of 52%, this would have been improved with subsequent epochs, however with the run time made that seem like ineffective use of my computational time. After doing some more research, I decided it was best to use a different EfficientNet pretrained model that was more computationally efficient as the problem was only a binary classification.

Model 6: Expanded Albumentation with EfficientNet_B3

I decided to expand my data augmentation to a list of adjusting gamma, brightness, contrast, cropping, and flipping. This was inspired by a Kaggle submission that had a high performance. Additionally, I added class weights to deal with class imbalance. This model was my best performing model at 74.5% accuracy. I additionally increased my learning rate back to 0.001 to allow for larger adjustments in the first few epochs. This was however not our group's best performing model.

Stefani Guevara's Experimentation

After Chris's finalized initial code base model script, I took to experimenting with different pretrained architectures provided by the torchvision models package.

Model 1: DenseNet169

Several experiments were conducted for each model. For densenet169, with the original unbalanced data, 5 epochs, and a batch size of 30 the model obtained an accuracy score of 66.8% and a weighted F1 score of 67.4%. Multiple attempts were done varying the number of epochs and batch size as well as image size under the densenet169 pretrained model, however they did not result in more impressive scores. The only data augmentation step included in this series of runs was an image normalization using transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)).

After doubling the number of instances of the normal hem cells and training with 3 epochs, the densenet169 model did reach as much as 75.0% accuracy and a comparable weighted F1 score of 73.8% on the test set. Here, the image size remained at 300x300 and the augmentation included random vertical and horizontal flips using RandomApply with a 50% probability.

With this Densenet model and with the remaining models, I left the number of instances of the minority class doubled on the training set.

Model 2: ResNet152

The results with resnet152 were consistently underwhelming with low scores ranging below 70s, even with higher epochs. Testing with the resnet152 pretrained model included the same procedure as that with the densenet pretrained network. At best, this model obtained an accuracy score of 68.3%.

Model 3: GoogleNet

The googlenet pretrained network with basic augmentation – normalization and random flips – provided better scores than ResNet using the same number of epochs, and marginally higher than with DenseNet, reaching up to 75.3% accuracy on one of its runs. This network provided promising results, however, after successfully implementing EfficientNet and seeing more impressive scores our team decided to look into experimenting with this latter architecture.

Model 4: EfficientNet_B0

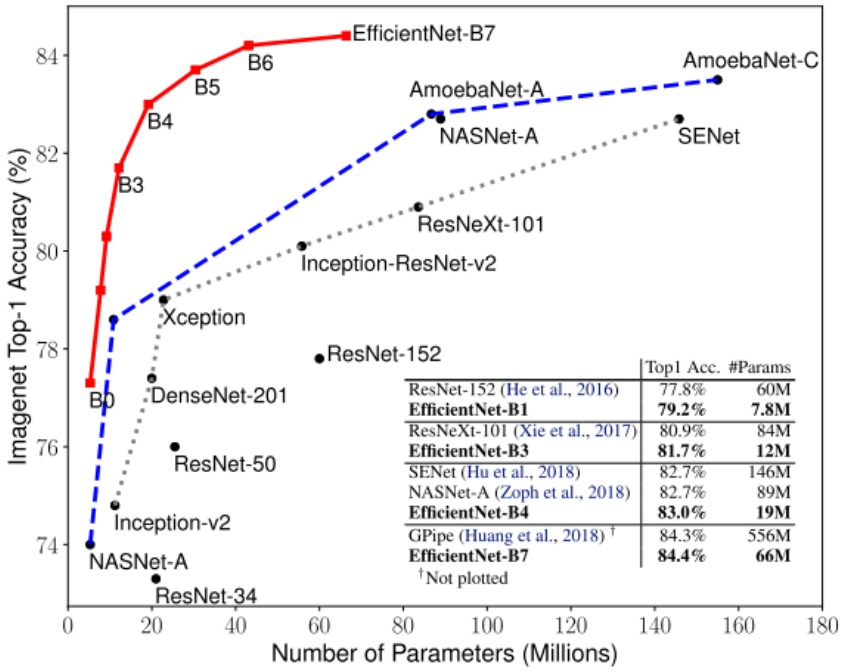
Influenced by the Leukemia-Detection using Efficient-Net B3 Kaggle notebook by Ashish Goswami, we wanted to look at how EfficientNet would perform on this dataset, considering it's a much newer architecture than the previously attempted pretrained networks; moreover, Goswami approached this dataset with Keras while we attempted to use PyTorch.

Each member attempted a different version of torchvision's EfficientNet, thus I implemented efficientnet_b0. On the first attempt with efficientnet_b0 and basic augmentation with 7 epochs our model did better than the previous models, reaching up to high 70's on the test set, specifically achieving an accuracy score of 77.6% and a weighted F1 score of 77.6% on its 2nd epoch, the highest scores obtained by the group on this project. My next best model obtained an accuracy score of 76.96% and a weighted F1 score of 76.98%.

Additional variations on the above model – such as removing the scheduler, changing the learning rate, and increasing the number of epochs by as much as 10 – resulted in comparable though slightly lower performances.

Results

As discussed in the previous section, the results of each of our experiments varied with respect to the type of pretrained network implemented. Out of all the networks we used, however, the EfficientNet baseline architecture resulted in the best overall performance with an accuracy score of 77.6% and a weighted F1 score of 77.6%. Moreover, the EfficientNet baseline had a notably faster runtime – as designed by its authors who looked at maximizing both accuracy and speed when searching for an optimal neural network architecture.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Figure 1. Model Size vs. ImageNet Accuracy

A popular image provided by the authors of EfficientNet, the figure above illustrates both how fast and accurate this architecture is compared to its predecessors, including some of those implemented in this project.

With EfficientNet, our best model used a rather minimal data augmentation strategy: horizontal and vertical flips with a 50% probability of application to each image and compulsory image

normalization. As discussed by Prellberg and Kramer, no brightness or color augmentation was undertaken due to the preprocessing completed on the dataset prior to public release. Moreover, no rescaling or color augmentation was done on the training set as it would lead to unnecessary distribution shifts from the testing set that would lead to poorer performance. In addition to the simple data augmentation strategy, this best model used a batch size of 32, image size of 300x300, an Adam optimizer with its default learning rate and a ReducedLROnPlateau scheduler, which we left at a factor of 0.5 and patience of 0. Our next best model used all else the same with an additional center crop of 200x200 and 9 total epochs. Both of these top models used an oversampling technique that consisted of doubling the number of instances of the minority, hem cell, class.

Summary & Conclusion

Overall our models performed competitively compared to known benchmarks. The majority of submissions on kaggle are between 36%-65%. Our 77.6% accuracy definitely performed better compared to the average submission. With that said there were three submissions that achieved scores of over 88- 93% for predicting if cells were cancerous. The first one built an efficientNet_B3 model, with some key differences to ours, the individual shuffles the validation into the training data then resampled. Additionally, they created a custom learning rate scheduler and most importantly were able to run 30 epochs for one given model. Due to time and resource constraints we were not able to match that computational time. The other two are both off older versions of the dataset, thus making it harder to compare directly, but one made their own 14 layer convolution model, and the other also used a pretrained ResNext50 model. Overall looking at benchmarks and our own testing, we found choosing the right pretrained architecture is key to obtaining a competitive score for your particular image classification problem. For our case we found that efficientNet performed the best for classifying and detecting ALL cells.

References

Acute Lymphoblastic Leukemia Classification from Microscopic Images using Convolutional Neural Networks (2020 paper)
Acute Lymphoblastic Leukemia Detection from Microscopic Images Using Weighted Ensemble of Convolutional Neural Networks (2021 paper)
C_NMC_2019 Dataset: ALL Challenge dataset of ISBI 2019 (C-NMC 2019)
Best deep CNN architectures and their principles: from AlexNet to EfficientNet
Committed Towards Better Future: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (2020)
Very Deep Convolutional Networks for Large-Scale Image Recognition Karen Simonyan, Andrew Zisserman (2014)
Kaggle Competition Link
VGGNet Image Link
EfficientNet Image Link (Slide 7)

Code Appendix

Please visit our git repository here:
https://github.com/guevarsd/Final-Project-Group4