An Exploration on the Pneumonia MNIST Dataset Through Small Sample Learning With and Without External Data

Chelsie Ng Yu Xiang Zhang

Abstract

This report investigates two challenges. First, given only 10 randomly selected class balanced samples of the Pneumonia MNIST data set, can we create a good training algorithm? By how much can we improve its accuracy? How much does the answer depend on not using any external data or any model trained on external data? We show evidence of how ResNet18 with RandAugment augmentation technique outperformed the baseline CNN and hyperparameters by 5%. Second, we further explore the same challenge with the ability to use external data or models not trained on the same data. We provide proof that pre-trained ResNet152 underperformed with an extremely slow training time while there is a tie between VGG19 and the baseline AlexNet.

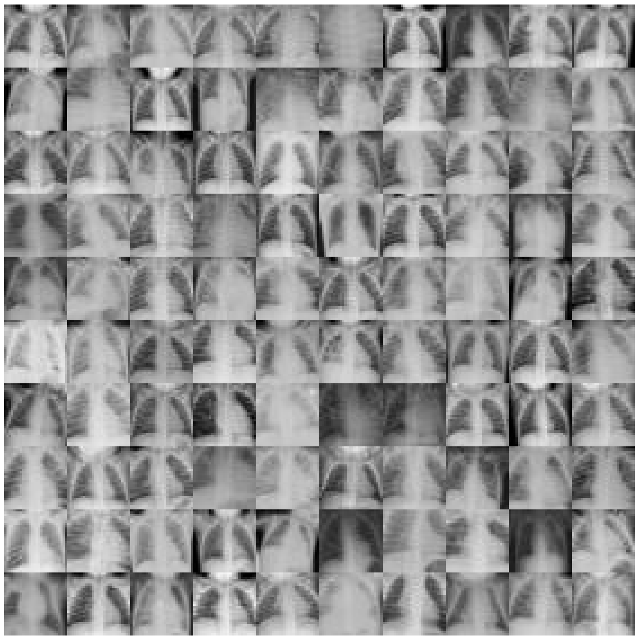


Figure 1. Chest X-ray data

1. Introduction

The project description proposes that we train our models on a random class balanced 10-sample subset of the Pneumonia MNIST training dataset and evaluate them on a 1000-sample subset.

Here is a look at the Chest X-ray data with and without Pneumonia (figure 1). The Pneumonia MNIST Dataset has two classes and consists of classifying chest x-ray images as having pneumonia or not.

1.1. Challenge 1: Learning with limited data without external data

This challenge involves exclusively using the 10 training samples. Since no external data is allowed, we will explore different model architectures, data augmentation techniques and custom libraries. More specifically, we will explore different Residual Neural Networks, automated data augmentation with a reduced search space, tuning-free data augmentation and automated deep learning.

1.2. Challenge 2: Learning with limited data with external data

This challenge is more lenient. Besides the 10 samples, we can use any external data or pre-trained models given that they were not trained on or disclosed information obtained from the same 10 samples. We considered various forms of fine-tuning and transfer learning with pre-trained models.

1.3. Validation Strategy

Due to the very limited training data, the sample of 10 training images is likely representative of the sample of validation images that we test on, but not representative of the real data generating distribution. To alleviate this risk, we always evaluate our models on 50 x 1000 testing samples during the testing phase. Precisely, we present our results using the Pneumonia MNIST test set for 50 random permutations of 10 training samples and 1000 testing samples.

2. Literature Review

2.1. AutoGluon

AutoGluon is a framework for AutoML, which aims at automating feature engineering, model selection and training. While most AutoML frameworks are based on hyperparameter search, AutoGluon avoids it to reduce training time by leveraging stacking and ensembles [8]. It has ranked topmost in at least two Kaggle competitions with minimal training time [12].

The framework minimizes the effort of data scientists and engineers [7] by preprocessing data, performing feature selection, model selection and training with only a few lines of code. Training the baseline model on a small dataset (such as the one in this project) only takes a few minutes.

2.2. Data augmentation

Nowadays, deep learning has performed remarkably well, especially on many computer vision tasks using deep convolutional neural networks. However, one drawback is that these networks tend to overfit if data is sparse[14]. Besides, medical image analysis is one of many application domains that do not have access to big data [14]. Hence the reason why data augmentation comes in handy.

Data augmentation is a popular data-space solution to the problem of limited data [14]. An interesting fact that we observed in Connor Shorten and Taghi M. Khoshgoftaar's survey paper about Image Data Augmentation for Deep Learning, was that the use of GAN image synthesis in medical imaging to classify liver lesions greatly improved its performance from 78.6% sensitivity and 88.4% specificity to 85.7% and 92.4% respectively[14]. We were very interested to apply GAN-based data augmentation in our project but unfortunately, we found it to be too bulky to implement.

We then turned to PyTorch's Automatic Augmentation Transforms library. In this paper, we apply the RandAugment and TrivialAugmentWide data augmentation techniques.

Automatic Augmentation has recently led to state-of-the-art results in image classification and object detection [5]. Instead of manually designing data augmentation policies, automated augmentation finds the best policy as a discrete search problem which consists of a search algorithm and a search space [4].

The downside is that its training complexity increases when applied on a large scale due to its separate search phase and, eventually may escalate computational cost [5]. Its separate search phase also makes it impossible to adjust the regularization strength based on model or data set size.

RandAugment removes both of these obstacles [5]. RandAugment has a significantly reduced search space which allows it to be trained on the target task with no need for a separate proxy task. Furthermore, due to its parameterization, its regularization strength may be tailored to different model and dataset sizes[5].

TrivialAugment outperforms the automatic augmentation methods mentioned above for almost free [11]. TrivialAugment has the most simple baseline. It is parameter-free and only applies a single augmentation to each image. Thus, no need to trade off simplicity, cost and performance [11].

2.3. Residual Neural Network

Deeper neural networks are more difficult to train as they face the vanishing gradient problem. The latter happens during the backpropagation process, when there are too many layers, the gradients become too small to proceed with optimization.

ResNet was developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun who were the winner of ILSVRC 2015 [2]. They were able to design an ultra-deep network free of the vanishing gradient problem [2]. ResNet follows a Convolutional Neural Network architecture but the authors explicitly reformulated the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions [10]. They solved the vanishing problem by skipping multiple layers that are not initially used, reusing activation functions from previous layers, re-training the network again and expanding the residual convolutional layers [13].

In challenge 1, we decided to compare performances between the baseline CNN and ResNet as we believe that ResNet makes it possible to explore additional features which a shallow convolutional network architecture may not capture [13].

In challenge 2, we decided to use the pre-trained ResNet model for our classification problem without the need to train from scratch as it has already been trained on large datasets such as ImageNet for image recognition purposes [2]. We will also focus on freezing the initial layers since the size of the training data is small and the data similarity between Pneumonia MNIST Dataset and ImageNet is very low [9]. This can assist with network generalization and speed up convergence [2].

2.4. VGG Neural Network

Because only the models from the PyTorch repository are permitted for challenge 2, our naive approach for exploring alternative solutions involves navigating the list of models [3]. A natural improvement over the baseline AlexNet model is by going deeper with VGGNet.

VGGNet builds on the success of Convolutional networks (ConvNets) and is particularly inspired by the configuration of AlexNet [15]. VGG generalizes the convolutional layers present in AlexNet and replaces the non-uniform configurations with repeated stacks of VGG blocks. The model has placed first and second in ImageNet Challenge 2014.

2.5. Vision Transformers

Transformer-based models are known to perform well on natural language processing (NLP) and computer vision (CV). We would like to explore this architecture in this project. Hugging Face Hub makes available a list of transformer models and the ViTForImageClassification in particular based on [1].

Vision Transformers have had a huge influence in CV. Its appearance challenges the previously dominant position of convolutional neural networks in the field of CV. Transformer architectures generally require a large amount of data to train; such pre-trained models tend to yield great results in NLP. Applying a similar strategy to computer vision, the ViT pre-trained model is expected to perform well in image classification tasks [6].

3. Challenge 1: AutoGluon

3.1. Methodology

As a framework separate from PyTorch, AutoGluon does not work with the provided starter code but requires a simple setup process. The Pneumonia MedMNIST dataset is saved as PNG files and a CSV file that records their splits (TRAIN, VALIDATION, TEST), image file locations and labels. The images were then loaded as an ImageDataset using AutoGluon API.

Because the MedMNIST dataset was saved with its own splits, the training process went along with this setup. 10 samples were randomly picked from the training set. However, the validation set only contains 135 samples and they were all used during model selection.

Training with AutoGluon requires a large amount of memory. Local runs on a Windows 10 machine with a GeForce RTX 3070 8GB throw memory errors. Google Colab works with Python 3.7 by default which is incompatible with AutoGluon and causes problems while loading PIL images. The baseline version was finally run on a remote Ubuntu 20.04 machine with GeForce RTX 3080 8GB.

3.2. Experimental Results

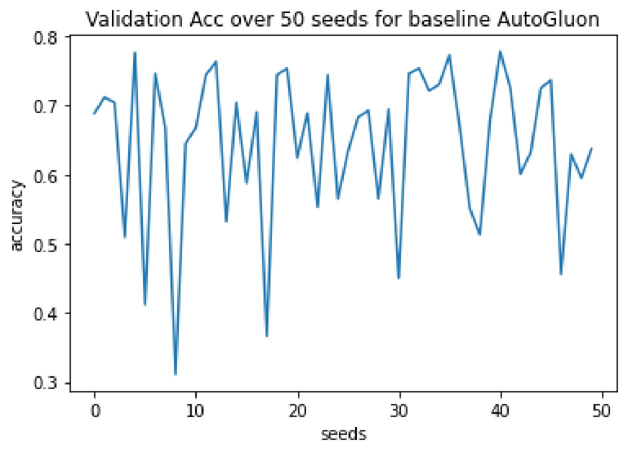


Figure 2. The accuracies of the baseline AutoGluon over 50 seeds.

Based on the results 2 from 50 random seeds, the baseline model selection tends to overfit the training data and perform poorly on validation data. All results were below 80% accuracy and hence unable to beat the baseline ConvNet benchmark 79.88 +- 5.73.

3.3. Discussion

AutoGluon provides automated hyperparameter search and model selection. The training process was completed in 23m 15s and thus failed both in training time and validation accuracy compared with the baseline CNN model (28.2s).

4. Challenge 1: Residual Neural Network

4.1. Methodology

The first methodology was to find a better network than CNN. After doing some research, it is claimed that Residual Neural Networks outperformed CNN due to their residual blocks which not only overcame the vanishing gradient problem but also, explore exclusive features that shallow networks may not.

We experimented between the different variations of ResNet: ResNet18, ResNet34, ResNet50, ResNet101, ResNet152. The latter was used while untrained, due to the limitations of this challenge. We were very careful to not use the pre-trained model weights. They were instead initialized randomly by PyTorch when the `pretrained` parameter is set to `False`. We plotted multiple graphs to help us better determine which one is the best. While the bigger models took more time to train, it was not a deciding factor. We were looking for the one which gave the best accuracy.

4.2. Experimental Results

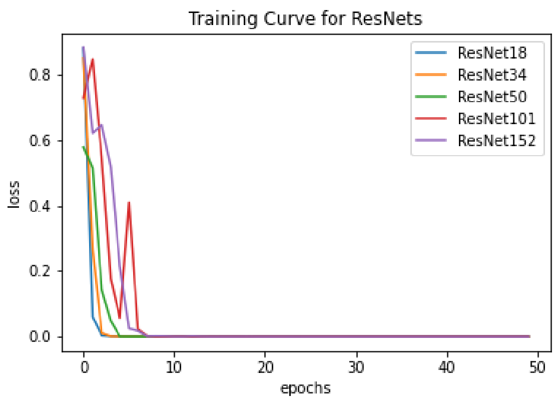


Figure 3. The loss of all ResNets

In each of the models, the loss eventually came down to near zero (figure 3).

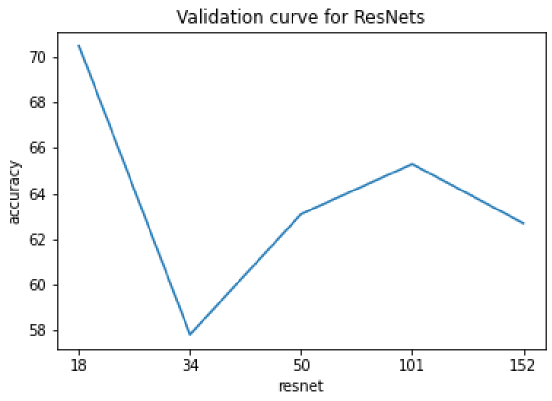


Figure 4. The accuracies of all ResNets

Testing each of the trained models on the validation set, the results were then really different (figure 4). ResNet18 seemed to have a better generalization and gave an accuracy of +70%. On the other hand, ResNet34 performed the worst at an accuracy of around 58%.

This little experiment gave us an insight into which model might have the most potential. To give us a more solid result, we compared how the model did over 50 different seeds.

Base CNN model

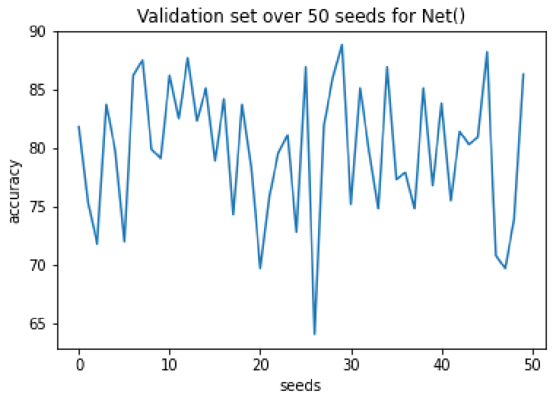


Figure 5. Validation of CNN over 50 seeds

On average the base CNN model gave an accuracy of 79.83% with a standard deviation of 5.71 and a variance of 32.65 (see Figure 5).

ResNet18

On average the ResNet model gave an accuracy of 74.98% with a standard deviation of 6.74 and a variance of 45.40 (figure 6).

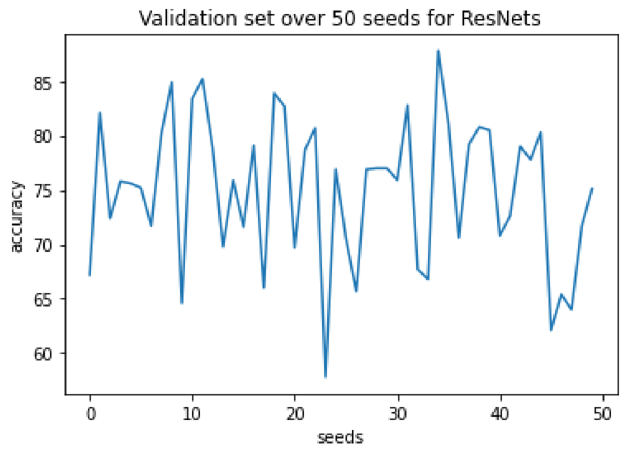


Figure 6. Validation of ResNet18 over 50 seeds

Models	Accuracies	Run times
Base CNN	79.83	41.5s
ResNet18	74.98	13min20s

Table 1. Comparison between CNN & ResNet18 over 50 seeds

As we can see in table 1, the CNN baseline model was still performing better than the ResNet.

4.3. Discussion

At this point, we have not yet exceeded the baseline model. However, we still have more room to make it better. Now that we got the best-performing model, we could improve it further by using data augmentation and hyperparameter tuning.

5. Challenge 1: Data Augmentation

5.1. Methodology

The idea behind data augmentation is to create more data. The images could be cropped, flipped or change in colours based on the available transformations. This will give the training data more variety, make the model more robust and help it generalize better.

We had then experimented with two different data augmentation techniques: RandAugment and TrivialAugmentWide. As discussed in section 2.2, these techniques alleviate the obstacles of automatic augmentation by reducing the search space and by being tuning-free respectively.

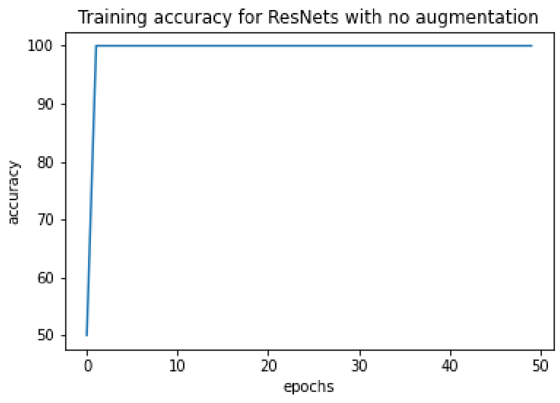


Figure 7. Training with no augmentation

5.2. Experimental Results

As we can see in Figure 7, training without data augmentation quickly reached 100% accuracy. However, with data augmentation, the model did not overfit and varied a lot more throughout the epochs.

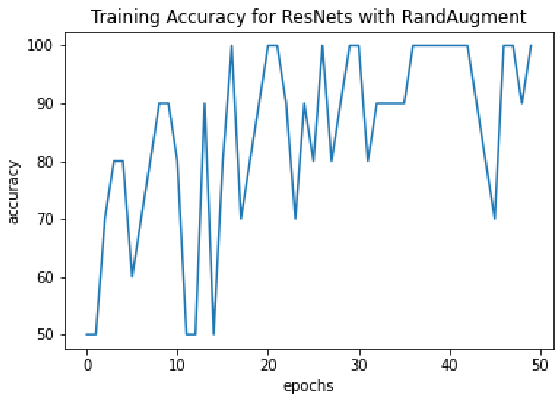


Figure 8. Training with RandAugment

Over 50 seeds, training and validating with RandAugment gave an accuracy of 83.76% with standard deviation of 5.64 and variance of 31.83 (figure 8) while with TrivialAugmentWide we got 82.58%, 6.15 and 37.84 respectively (figure 9).

ResNet18 together with RandAugment performed slightly better than TrivialAugmentWide.

5.3. Discussion

With data augmentation, we went way above the base CNN model. i.e 83.76% over the baseline accuracy of 79.83%. One thing left to do was to hyper-tune the best ResNet with the best data augmentation tech-

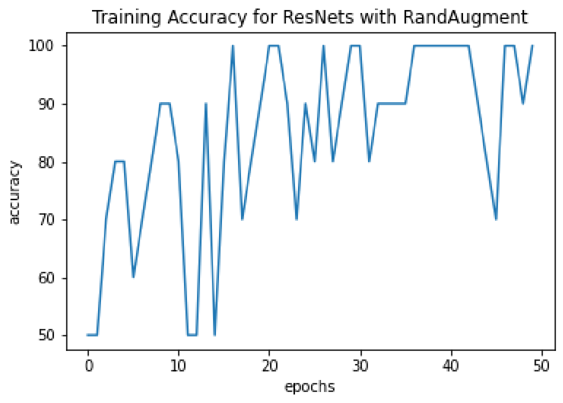


Figure 9. Training with TrivialAugmentWide

nique.

6. Challenge 1: HyperParameter Tuning

6.1. Methodology

Many variables and parameters can help the model optimize better: batch size, learning rate, optimizer and number of epochs. We experiment with them to find the best combination. This technique is replicated to the CNN as well, so we can compare them to the best extent.

6.2. Experimental Results

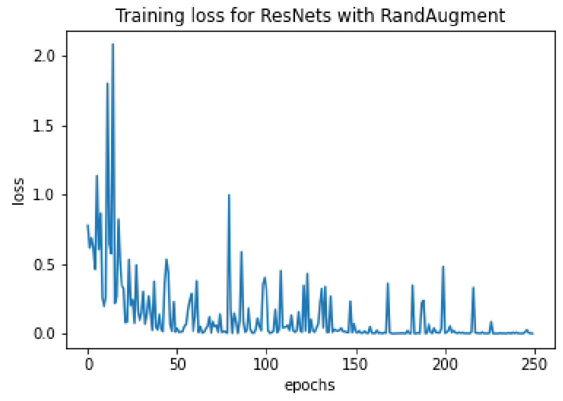


Figure 10. ResNet18: Training loss with hyperparameters

We conclude that the best hyper-parameters are: a batch size of 32, and a learning rate of 1e-3 with AdamW optimizer trained on 250 epochs. As we can see from Figure 10 and 11, over 250 epochs the model was able to converge near zero and reach 100% respectively with a decreasing amount of fluctuations.

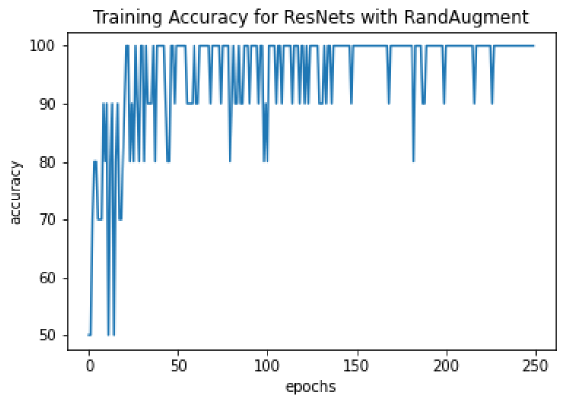


Figure 11. ResNet18: Training accuracy with hyperparameters

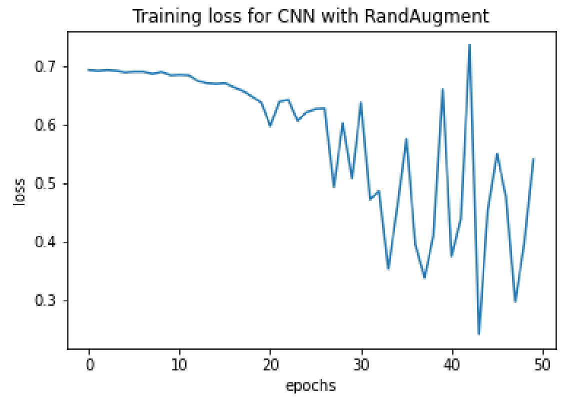


Figure 12. CNN: Training loss with hyperparameters

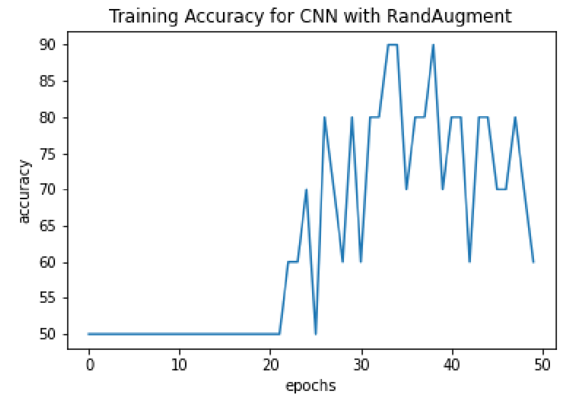


Figure 13. CNN: Training accuracy with hyperparameters

On the other hand for the CNN model, we conclude that the best hyper-parameters are the same as our ResNet model. However, the best epochs to train it on is 50 as we noticed the training accuracy decreased as of 150 epochs. As we can see from Figure 12 and

13, the CNN model was neither able to converge towards zero nor reach an accuracy of 100%.

To confirm that the ResNet model performed way better than the CNN model, we trained and validated them over 50 different seeds.

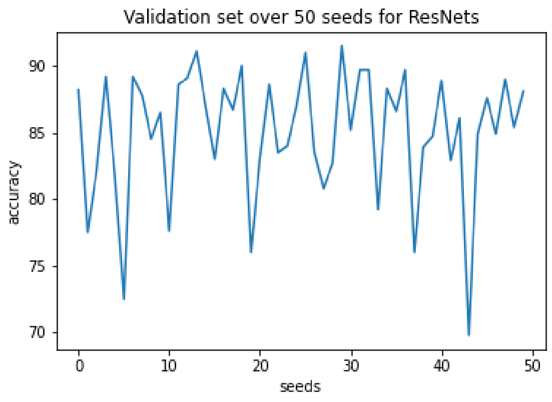


Figure 14. ResNet18 with hyperparameters over 50 seeds

The ResNet18 model gave an accuracy of 85.06% with a standard deviation of 4.78 and variance of 22.86 (figure 14). Its run time is 56min 18s.

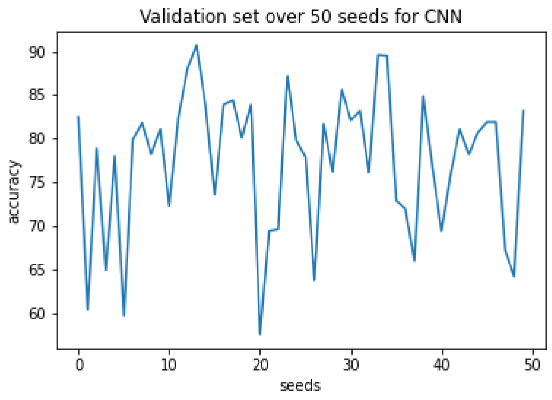


Figure 15. CNN with hyperparameters over 50 seeds

The CNN model gave an accuracy of 77.48% with a standard deviation of 8.11 and a variance of 65.73 (figure 15). Its run time is 40.6s.

6.3. Discussion

Tuning the model gave a minor improvement on the validation set. 85.06% over the baseline parameter's accuracy of 83.76%. We also observed that training ResNet over 50 seeds is 80x slower than the CNN model. Conversely, we noticed that optimizing the CNN's model parameters along with data augmentation, decreased its accuracy from 79.83% to 77.48%. Hence to conclude challenge 1, our final accuracy is 85.06% with our tuned RandAugmented ResNet18 model.

7. Challenge 2: Pre-trained Residual Neural Network

7.1. Methodology

Since we do not have any limitations for challenge 2, this time we will experiment with the ResNet models being pre-trained with their respective pre-trained weights. We will also experiment with retraining all the layers and only the last ones, that is, the fourth layer and the fully connected layer. The latter will help to customize the layers according to the Pneumonia MNIST Dataset as discussed in section 2.3.

7.2. Experimental Results

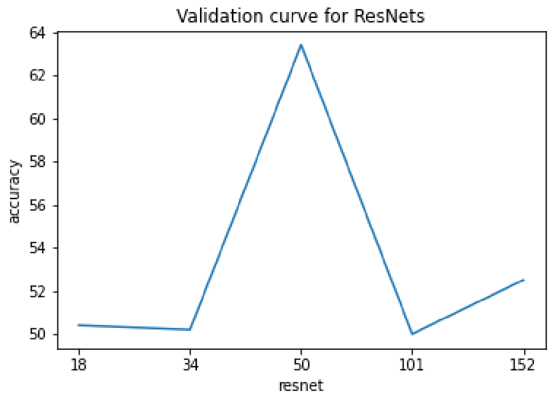


Figure 16. Pre-trained ResNets with all layers optimized

We can observe that when all layers are trained, the best model is ResNet50 and reaches around 64% (figure 16). When all the layers are frozen except the last layers, ResNet152 performs better at around 72% (figure 17).

Models	Accuracies	Run times
AlexNet	85.72	28min 6s
ResNet152	68.75	8h 28min 58s

Table 2. Comparison between AlexNet & ResNet152 over 50 Seeds

Over 50 seeds, the base AlexNet model still did better with an accuracy of 85.72%.

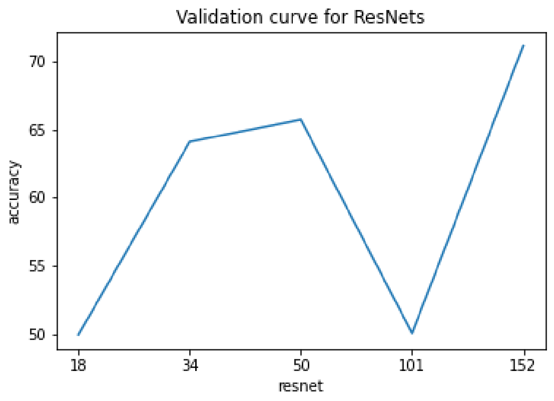


Figure 17. Pre-trained ResNets with only last layers optimized

7.3. Discussion

We can conclude that the ResNet models underperformed compared to the base AlexNet model. Besides, training the ResNet152 model was tremendously slow (see table 2). Hence, we did not go further and tune the model further.

8. Challenge 2: VGG

8.1. Methodology

Inspired by the baseline AlexNet model's performance, we formed the idea of using an improved version of AlexNet. VGGNet extracts a VGG block substructure from AlexNet and expects to perform better than AlexNet. The VGG blocks are stacked together to build a deeper version of AlexNet. It uses more memory as a result and trains significantly slower than AlexNet.

We loaded both VGG16 and VGG19 pre-trained models from the PyTorch repository. A preliminary run shows a boost in validation accuracy from VGG19 so we decide to pursue the deepest model. To fine-tune the model, we set the model to output a single number as output for classification. We adopt SGD optimizer to update the weights on the classification head with an initial learning rate of 1e-3, momentum 0.9 and weight decay 0.005. For each seed, the model trains for 200 epochs. Training for more epochs will only improve training accuracy and may lead to overfitting.

The learning rate scheduler `ReduceLROnPlateau` will decide whether to reduce the learning rate based on the validation loss value every epoch. `StepLR` reduces the learning rate at a fixed rate of 0.99 and will in practice lead to similar performance but faster training. We adopt `StepLR` with a step size of 5 so that it will reduce the learning rate every 5 epochs and will not waste time computing the validation loss.

8.2. Experimental Results

The final validation accuracy was fixed at only 85.48 +- 2.20. This is only on par with AlexNet model.

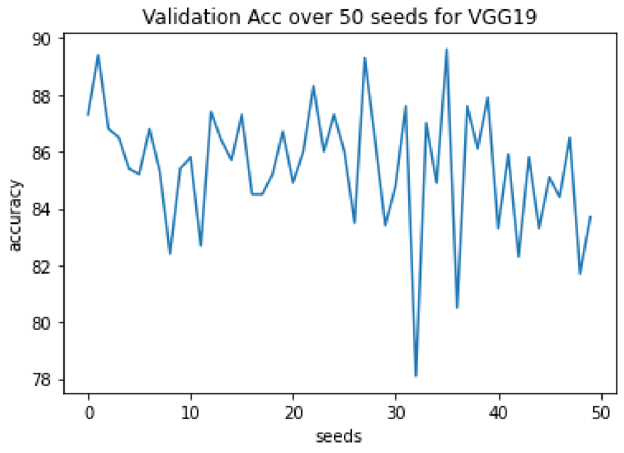


Figure 18. Pre-trained VGG19 over 50 Seeds

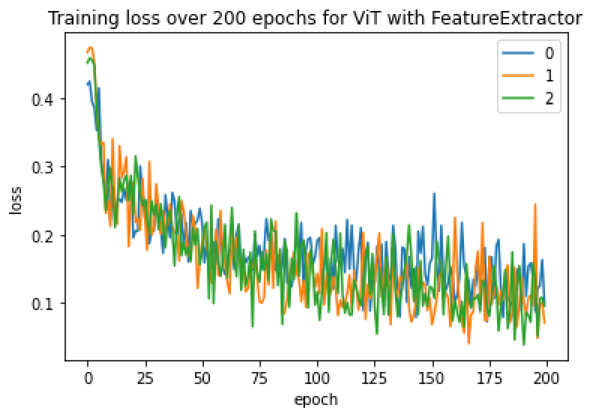
8.3. Discussion

Because of the deeper architecture and the LR scheduler update, training VGG19 has taken 13m 49s. This is about 10 times slower than AlexNet's 1m 16s, but the validation accuracy did not improve significantly.

9. Challenge 2: ViT

9.1. Methodology

One of the problems is how to prepare the images for Tranformer-based models. We leveraged the `FeatureExtractor 'google/vit-base-patch16-224-in21k'` from Hugging Face Hub to apply the correct transformations (cropping, normalizing and turning grayscale to RGB). On top of this, we added data augmentation with random augmentation and flipping. After applying such transformations (compared with the default resizing, normalizing and turning to RGB), we noticed



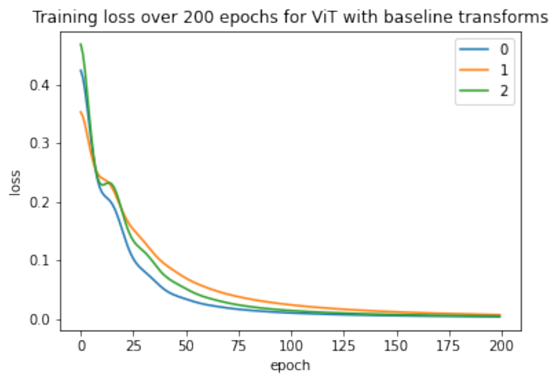


Figure 19. A comparison between the models trained with FeatureExtractor transformations and without. The legend shows different seeds.

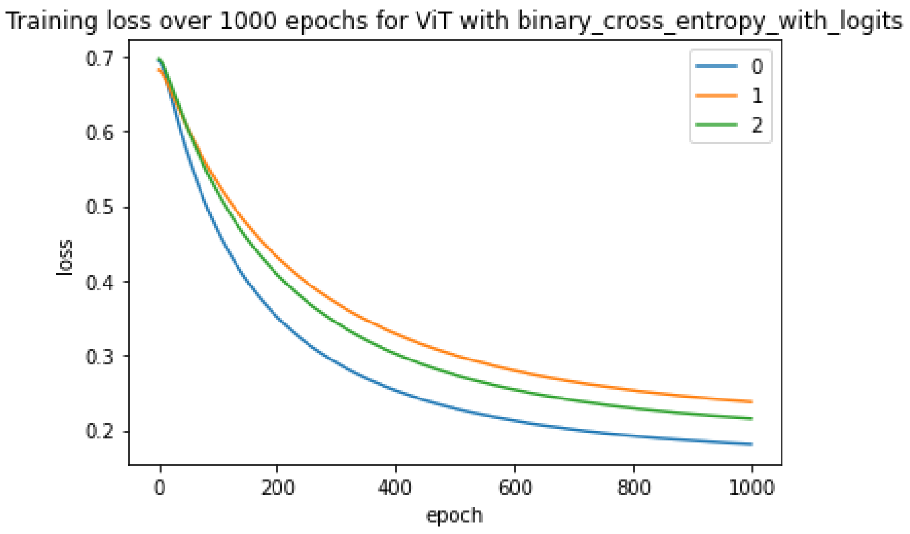


Figure 20. The ViT trained by loss function of binary cross entropy with logits.

that the loss value becomes unable to converge and leads to lower accuracy, see 19.

We load a pre-trained `'google/vit-base-patch16-224-in21k'` model from Hugging Face Hub to initialize a `ViTForImageClassification`. We set the `num_labels` to be 1 so the model will create a classification head that outputs a single label. The Hugging Face model output comes with its own logits and loss value. They can be used to calculate accuracy and perform backpropagation. In comparison with the default `binary_cross_entropy_with_logits`, we did not observe a difference in terms of final validation accuracy. We attempted 'Adam' 'AdamW' and 'SGD' optimizers with and without weight decay. We adopted dynamic learning rate reduction to minimize the loss function. The learning rate scheduler `StepLR` will reduce the learning rate every epoch by a fixed rate gamma=0.99. To allow the training loss value to stabilize, the ViT will train for 200 epochs for each seed value over 3 seeds.

9.2. Experimental Results

Among various combinations of the above hyperparameters, the best validation accuracy that was achieved over 3 seeds is 76.87 +- 2.45, and over 50 seeds is 75.38 +- 5.89. From 20, we can observe a similar graph for the loss computed by binary cross-entropy with logits. Although the loss value range is different, the results show a similar validation accuracy of 74.37 +- 0.29.

9.3. Discussion

Fine-tuning of ViT has significantly slowed down compared with the baseline code. Each instance of the ViT model trains for more epochs; the LR scheduler needs to update the learning rate every step. Total running time of ViT training for 50 seeds extends to 22m 22s compared with 1m 16s for the baseline AlexNet model.

10. Conclusion

In this work, we explored different methods to conduct few-shot learning using MedMNIST Pneumonia dataset. Our results show that untrained ResNet18 with RandAugment outperforms the baseline untrained CNN while pre-trained VGG19 has a similar performance with baseline pre-trained AlexNet. Our goals involve applying the knowledge in deep neural networks and trying various architectures to deepen our understanding of this topic. In that sense, we explored 3 network architectures, 2 model selection techniques and data augmentation.

For future improvements on this project, we would like to leverage stacking techniques to exploit well-performing models. We can experiment with different customized ConvNet configurations to better understand the problem and why the baseline models perform well. We may also apply a combination of data augmentation techniques to assist the model training.

References

[1] Fine-Tune ViT for Image Classification with hugging-face transformers. 3

[2] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, Mar. 2021. 2, 3

[3] Torch Contributors. Models and pre-trained weights — Torchvision 0.12 documentation. 3

[4] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018. 2

[5] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019. 2

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020. 3

[7] Nick Erickson. AutoGluon: Deep Learning AutoML, Oct. 2020. 2

[8] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. 2020. 2

[9] Dishashree Gupta. Transfer learning and the art of using pre-trained models in deep learning. `https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-` 2017. (Accessed Apr. 25, 2022). 3

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 2

[11] Samuel G. Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation. *CoRR*, abs/2103.10158, 2021. 2

[12] Shashank Prasanna. Machine learning with AutoGluon, an open source AutoML library, Mar. 2020. 2

[13] Run:AI. PyTorch ResNet. 2, 3

[14] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, articles/10.1186/s40537-019-0197-0, 2019. 2

[15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014. 3