Extended Essay : Computer Science

The effect of regularization of the pretraining of the source task on the performance of "medical-to-medical" transfer learning to train convolutional neural networks across medical imaging datasets.

Research Question : To what extent can the regularization of the pretraining on the source task can improve the performance of the model on the target task in "medical-to-medical" transfer learning from tuberculosis CT-scans to train a convolutional neural network for COVID-19 CT-scans binary classification ?

Words: 3995

Table Of Contents

Acknowledgments……4

Glossary of terms…….5

1. Introduction……..6

2. Background Information and Theory: ……8

2.1 Computer vision and pattern recognition

2.2 Neural networks

2.2.1 Forward Propagation: How does a neural network “predict”?

2.2.2 Back Propagation: How does a “neural network” learn to “predict”?

2.3 Convolutional Neural Networks (CNNs)

2.4 Overfitting and regularization techniques investigated

2.4.1 Data Augmentation

2.4.2 Dropout

2.4.3 Batch Normalization

2.5 Transfer Learning

3. Methodology …….……... 18

3.1 Implementation of the neural networks

3.2 The datasets

3.3 The model architecture

3.4 Preprocessing

3.5 Metrics used in evaluating the models

3.6 Hyperparameters used in the experiment

4. Experimental Results…….24

5. Discussion……..25

6. Conclusion…….28

I. Bibliography……...29

II. Appendix……..35

Acknowledgments

I want to thank my supervisor and IB coordinator who have allowed me to undertake this rewarding research opportunity, despite computer science not being offered at my school. I would love to thank my family, my friends, especially my lunch-table friends.

Furthermore, I would love to thank everyone who taught me something, whether it was differentiating a function, evaluating macroeconomic policies, doing lab reports, pronouncing a Spanish word, writing a political essay, analysing literary work, or making a dreamcatcher.

Glossary of terms

NN neural network

BN Batch Normalization

CNN Convolutional neural network

TL Transfer learning

GPU Graphical Processing Unit

1 . Introduction

Training convolutional neural networks (CNNs) to learn the patterns found in medical scans for computer-aided diagnosis can benefit medicine tremendously. However, a significant amount of labeled data is needed for training to perform adequately on unseen data. Labeled data scarcity prevails in medical applications of **machine learning** (**ML**) because of a multitude of reasons[1], increasing the likelihood of **overfitting**, the inability of the model to generalize patterns.[2]

One of the techniques investigated in the literature of computer vision to address data scarcity is **transfer learning (TL).**[3] It involves "utilizing knowledge gained while solving one a **source-task** and applying it to a different but related problem (**target-task**)". "Knowledge" in this context refers to the parameters of the **neural network** (**NN**) as a mathematical function, which are the weights and biases for linear operations that the NN function does on the input to perform a task. This technique is usually applied when the source-task has significantly more available data compared to the target-task[4]

One of the applications of **TL** is what was referred to in a paper [5] as a "medical-to-medical" TL strategy, where the source-task dataset is of the same medical domain of the target task. Despite the significant similarity between medical imaging datasets of the same domain, there is a possibility that **TL** may transfer parameters were learned from the residual noises of the source task dataset, rather than parameters learned from its more generalized features and thus limiting the potential of **TL** across medical imaging datasets. Regularization techniques

in **ML** aim to increase the generalizability of trained models, to have a more viable performance on unseen data. Therefore, regularization of pre-training could significantly limit the learning of residual noises during pretraining on the source-task dataset, improving the transferability of generalized features that could be learned from that dataset, which would, in turn, improve the performance of model on target-task on unseen data as it learned those generalized features.

In this paper, the effect of regularization of the pretraining of the source-task on the performance of transfer learning models on target-task across medical imaging datasets is investigated. For the experiment, the source-task is **tuberculosis binary classification**, which is used to fine-tune a model for **COVID-19 binary classification**, where both tasks share the same domain of CT-chest scans. The paper explores three of the widely-used techniques for regularization: **dropout**, **batch normalization (BN),** and **data augmentation**. *To what extent can the regularization of the pretraining on the source task can improve the performance of the model on the target task in "medical-to-medical" transfer learning from tuberculosis CT-scans to train a convolutional neural network for COVID-19 CT-scans binary classification?*

Answering this question could contribute to the literature of **TL** in medical imaging and computer vision in general, as no study, to the best of my knowledge, has investigated the effect of regularization of pretraining on the performance of **TL**. Advancing the performance of medical imaging by investigating improvements to a technique of significant utility such as **TL** can provide a stream of benefits for both medical research and the clinical routine

2. Background Information and Theory

2.1 Computer vision and pattern recognition

Hand-engineered **feature extractors for computer vision tasks** were built based on **domain-specific knowledge** to select the **relevant features needed** from a given image and **implement** their extractors for a given task[6]. **Deep learning** introduced an end-to-end (domain-agnostic) learning approach that capitalizes on training on adequate data to recognize the relevant features underlying a given dataset to perform a given task. [7] This approach is promising for medical imaging as meeting the demand for medical-domain knowledge is non-trivial.[8]

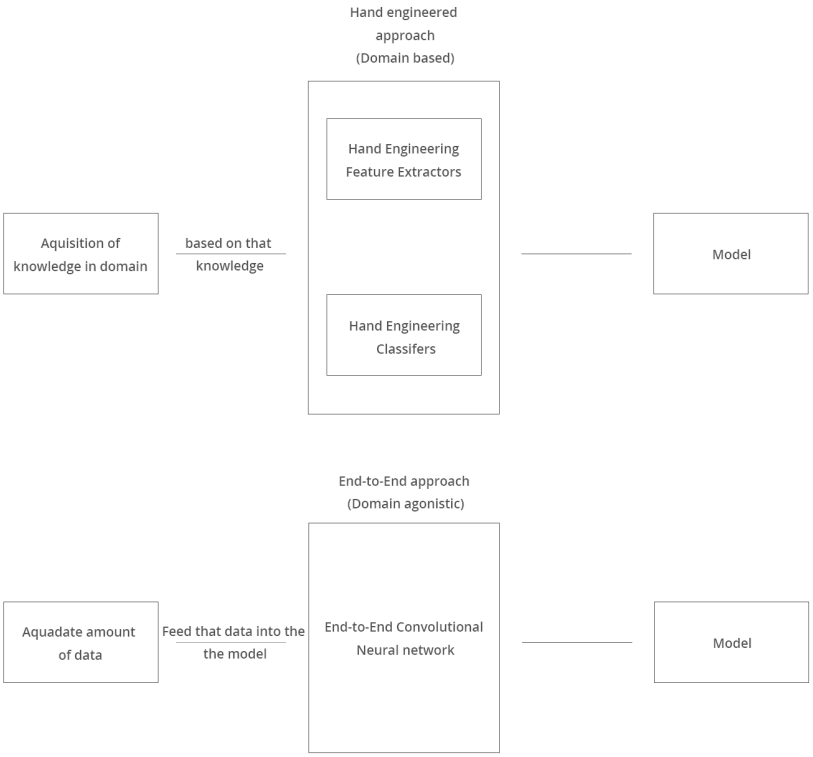


Figure (1): A figure that contrasts the domain-based hand-engineered approach to the domain agonistic (end-to-end) approach in computer vision engineering

2.2 Neural networks (NNs) :

A NN consists of stacked **layers of units (perceptions) that represent a mathematical function of the input**. It consists of **an input layer**, **several hidden layers,** and an **output** layer. [9]

2.2.1 Forward propagation: How does a neural network "predict" ? :

The NNs predict in a process called forward propagation. The input is fed into the input layer, then the neurons apply a linear operation on the data, where the inputs are multiplied by particular **weights** and a **bias** is added. This linear operation is followed by a non-linear operation (**activation function**) that aims to introduce non-linearities to the NN. After that, an output is produced, it becomes the input of the next layer. The input data keeps going "deeper" in the network until the final layer produces its output, in a process of sequential linear algebraic operations[10]

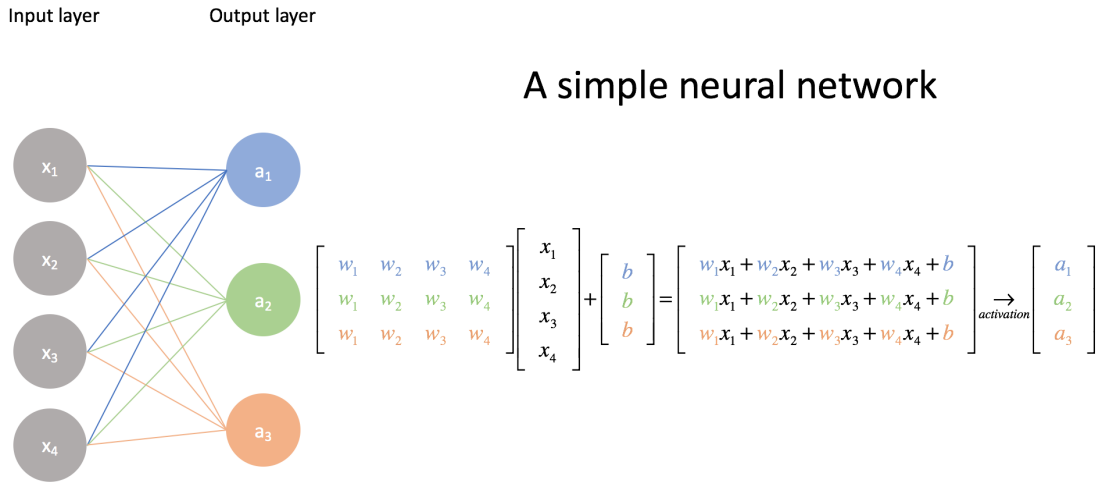


Figure (2): A figure that shows how NNs forward-propagates through matrix multiplication[11]

2.2.2 Backpropagation: How does a neural network “learn” to “predict”? :

The training of the NNs involves an iterative process of **forward propagation** and **backpropagation**. For each iteration, the network **forward-propagates** to produce an output. After that, the output is evaluated by the **loss function** $\mathcal{L}$, a function of the **predicted output** and the **labeled output** provided in the training dataset that assesses the performance of the network for a given iteration (The lower $\mathcal{L}$, the better). After calculating the error, **backpropagation** takes place. The negative gradient of the **loss function** $\mathcal{L}$ with respect to each of the parameters is approximated, and each learnable parameter of the network is updated using **gradient descent algorithm (optimizer)**[12] so that $\mathcal{L}$ is minimized with respect to the parameters of the network. As the value of the gradient becomes increasingly small (i.e approaching a local minimum of $\mathcal{L}$), the model is said to have **converged**. **Equation (1)** represents the basic version of gradient descent , where each parameter of the network is updated by the product of the negative gradient of the loss function $\mathcal{L}$ with respect to that parameter and a constant called **learning rate** η**,** a hyperparameter that governs the size of the updates.[13][14]

$$w_{t+1} = w_t - \eta \nabla w_t$$
$$b_{t+1} = b_t - \eta \nabla b_t$$
$$where, \nabla w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w=w_t,\ b=b_t}, \nabla b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at\ w=w_t,\ b=b_t}$$

Equation (1): The update rule of the gradient descent algorithm (optimizer)[15]

2.3 Convolutional neural networks (CNNs)

In CNNs, which achieved significant performance on computer vision tasks[16], low-level features are extracted by shallower filters, then those low-level ones are abstracted by deeper filters to detect more complex and higher-level features[17]. Filters represent tensors that hold a number of weights that extracts features by applying them to the input through convolution process[18] Consider figure (3) that demonstrates how Sobel-y filter functions as **a vertical edge detection extractor**[19]. In CNNs, those weights are learned analogously to the weights in classical NNs described in **2.2**.

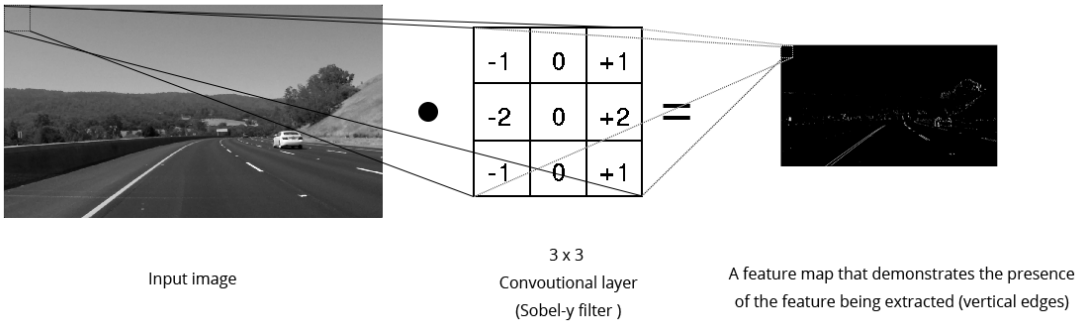
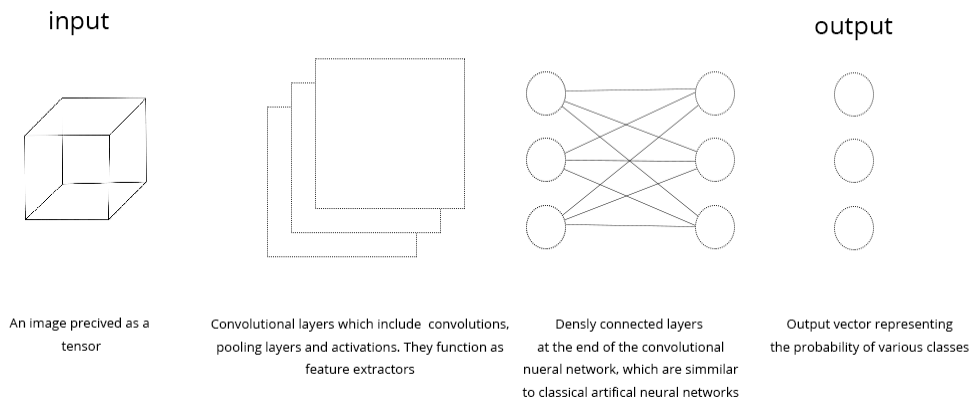


Figure (3): The filter has been "slid" over the image, outputting a "feature map" of the "edge" feature in this given image, where the brightest parts represent the presence of the vertical edges in the input image.

After applying a stack of convolutional processes to the input, the output of those processes is usually then passed to a set of **fully-connected (FC) layers**, like the one described in **2.2**, for processing the highly-abstracted feature maps by the filters to produce the output. The typical architecture of a CNN is demonstrated in Figure (4) below.[20]



Figure(4): The classical components of a convolutional neural network architecture

Convolution layers: A convolutional layer consists of several filters that extract particular features by outputting the feature maps of those features. The parameters (weights) of those filters are learned by CNNs by backpropagation. After convolution is applied, a learnable **bias** is added to the produced feature map and then is passed to the **activation function** that introduces non-linearity to the model.[21][22]. After applying the activation, the output may be passed to a **pooling layer**, which can be thought of as a downsampling layer that preserves the important features extracted.

Fully-connected (FC) layers: FC layers are the same layers found in classical NNs. FC layers are often added at the end of a CNN, where they process the high-level and low-dimensional features extracted and downsampled by stacks of convolutional layers. [23]

2.4- Overfitting, regularization, and regularization techniques investigated:

Overfitting is a significant issue during training machine learning (**ML**) models, where highly-parameterized models fail to generalize features of the datasets, which can be usually observed in a gap between the superior performance of the overfitted model on the training dataset, and its poor performance on unseen data. One of the approaches to address this issue is training on larger datasets or lowering the capacity (i.e number of parameters) of the model. However, this is not always possible, particularly in medical scenarios with data scarcity and non-negotiable need for high performance. **Regularization**, which encompasses the suite of techniques that aim to improve the generalizability of **ML** models, is consequently of significant importance.[24]

2.4.1 Data augmentation:

Data augmentation encompasses a set of techniques that aim to compensate for the limitedness of a training dataset by increasing it artificially. This is done by creating modified data from the existing ones. Data augmentation has a **regularizing effect**, as it increases the generalizability of the model by introducing variations that the model may have not learned from the limited training dataset and could be present on unseen data.[25] This paper explores a **limited class** of data augmentation techniques, which involve several basic **geometric transformations** and **contrast variations**.

2.4.2 Dropout :

Dropout is a regularization technique that combines the predictions of a large number of NNs. This is done by randomly "dropping out" and eliminating some of the perceptions of the neural network during each training iteration. For instance, if dropout is applied to the layer $x$of neural network that consists of perceptions n with a probability of p(hyperparameter of dropout): In each iteration, each unit of layer x has the probability of $p$to be "dropped out ", where that unit and its corresponding weights are removed from the network. This technique ultimately leads to training a "thinned" neural network whose weights are averaged from all of the "noisy and barely-trained" networks that consist of the surviving perceptions from dropout. Dropout has a **regularizing effect**, as it prevents complex co-adaptations between the units of the NN by averaging the parameters that were influenced by the residual noises learned by each noisy network from each mini-batch during training, improving NNs generalizability[26]

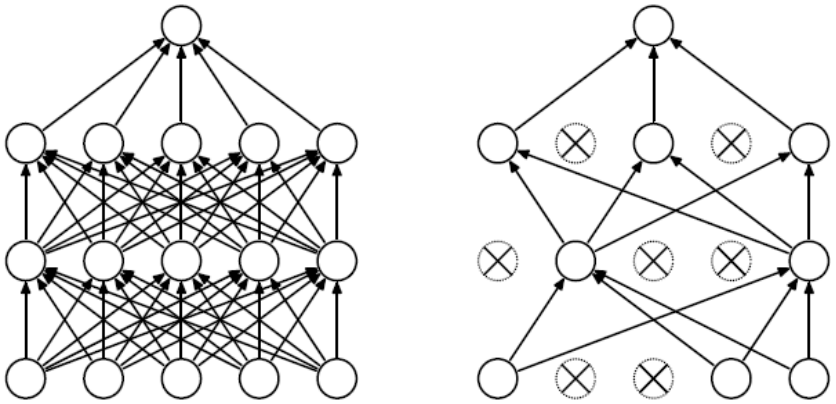


Figure (5): Demonstration of dropout technique from its paper.

2.4.3 Batch Normalization (BN) :

BN is a technique that standardizes the inputs of the layers of NNs, which are also outputs (activations) of the preceding layers. By "standardizing", the original paper meant subtracting the mean of the activations of the preceding layers network and dividing by their standard deviation for each mini-batch, so that they have a mean of 0 and a standard deviation of 1. Standardization is done to input data as it has proven to improve the training speed empirically, but this technique expands this process of standardization to be applied to the inner layers of the network. Those standardized activations means and standard deviations values of different batches are averaged by two learnable parameters for each activation: β (mean value of the mean of the activation) and γ (mean value of the standard deviation of the activation) respectively. After training, those two hyperparameters are used in standardizing the input data that comes from preceding layers.

This technique has a **regularizing effect**, given that the two learnable parameters β **and** γ of each activation are updated by each mini-batch introduced, where each mini-batch has a different mean and standard deviation. Similarly to dropout, BN averages the residual noises from different mini-batches, leading to a better generalization.[27]

2.5 Transfer Learning (TL) :

CNNs have a hierarchical architecture, where the lower-level features extractors are present in the shallower layers of the CNN. Lower-level feature extractors can be shared between classifiers of different tasks.[28] TL aims to transfer some of those feature extractors (filters weights) from the pre-trained model that it has been trained on the more available data of the source-task, to the model of the target task, which may not learn those parameters from its

scarce data. Due to the scarcity of medical data, TL is of significant potential for medical imaging applications [29].

A TL application is a "medical-to-medical" strategy, which involves pretraining a NN on a medical task of the same domain (i.e tuberculosis and COVID-19 classification share the same domain as both datasets use chest CT-scans), and then fine-tuning the NN on the medical target-task. This approach has been implemented in this paper[30] The strategy is illustrated in **Figure (6)**.

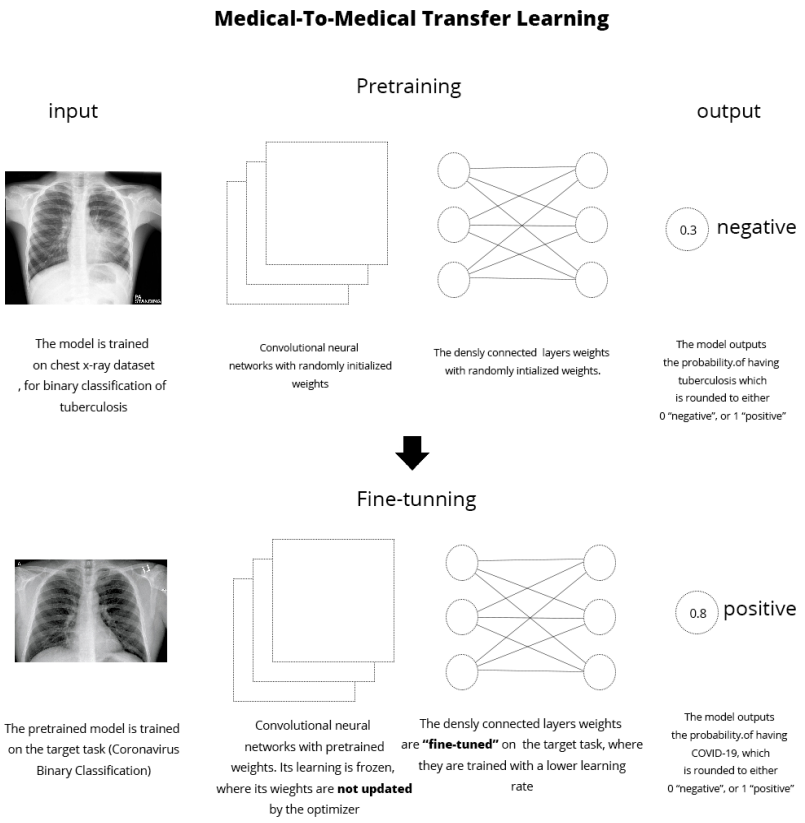


Figure (6): a figure that illustrates the medical-to-medical transfer learning strategy

Despite the significant overlapping between the features of source-task and the target-task datasets in medical imaging datasets, it is possible that the source-task model may transfer some of the residual noises of its training dataset, which could hinder the improvement that TL can bring, as irrelevant and noised parameters are transferred. Regularization of pretraining could limit the learning of residual noises of the source-task dataset and improve the transferability of the more generalized features of the source-task dataset that are likely to be relevant for the target-task, which would, in turn, improve the performance of TL. To examine this effect, an experiment has been designed to examine the effect of regularization of the pretraining for the source-task on the performance of the fine-tuned model on the target-task with unseen data in TL across medical imaging datasets.

3. Methodology

3.1 The implementation of networks :

Models with three techniques of regularization (**data augmentation**, **dropout**, and **BN**) and a **baseline** model (no regularization) will be trained on the binary classification of **tuberculosis CT-scans (**source task**)**. After pretraining, the weights of each pretrained model will be transferred to the same architecture and each one would be "fine-tuned" on **COVID-19 CT-scans binary classification (**target task**)**, where just the last 3 FC layers parameters being updated. This freezing of layers is to avoid **catastrophic forgetting**[31], **where** the NN loses the parameters it has learned from a previous task while learning sequentially. Freezing shallower layers addresses this by preserving most of the parameters learned from **TL** from the source-task (tuberculosis).

After the four models are fine-tuned, their performance will be evaluated against four small datasets of unseen COVID-19 scans to increase certainty about the relative performance of the different models to each other. The networks have been implemented in Python 3 using Google's "Tensorflow 2", a framework for training **ML** models[32]. The experiments were conducted on Kaggle[33], a platform that offers free cloud-based Nvidia K80[34] GPUs to compensate for the limitations of the owned hardware. **GPUs** are vital for training CNNs efficiently because of their performance on linear algebraic processes.

3.2 The Datasets :

3.2.1 Tuberculosis CT-scans dataset: The models were pretrained on the binary classification of tuberculosis CT-scans, which shares the same domain (chest CT-scans) of the target task (COVID-19 binary classification). The pretrained-model has been trained on a public dataset collected by a team of researchers[35] , containing 7000 x-ray JPEG images that are equally balanced between “negative” class and “positive” class. Out of those 7000 available images, 350 images were used for validation and monitoring networks performance as they train, leaving 6650 for pretraining the CNN.

4.2.2 COVID-19 CT-scans dataset: One of the datasets that have been widely used in many research papers was the **COVID-19 Radiography Dataset**[36]. However, the images that are classified as “negative” suffered from a pediatric bias, where all of them were collected from children, while all of the COVID-19 scans were of adults. This may lead the model to learn wrong patterns, which could affect the reliability of the experiment results. To address that issue, I have replaced images of the negative class from the original dataset with ones from the NIH[37] chest x-ray dataset from 2017 that are labeled with “no findings” .

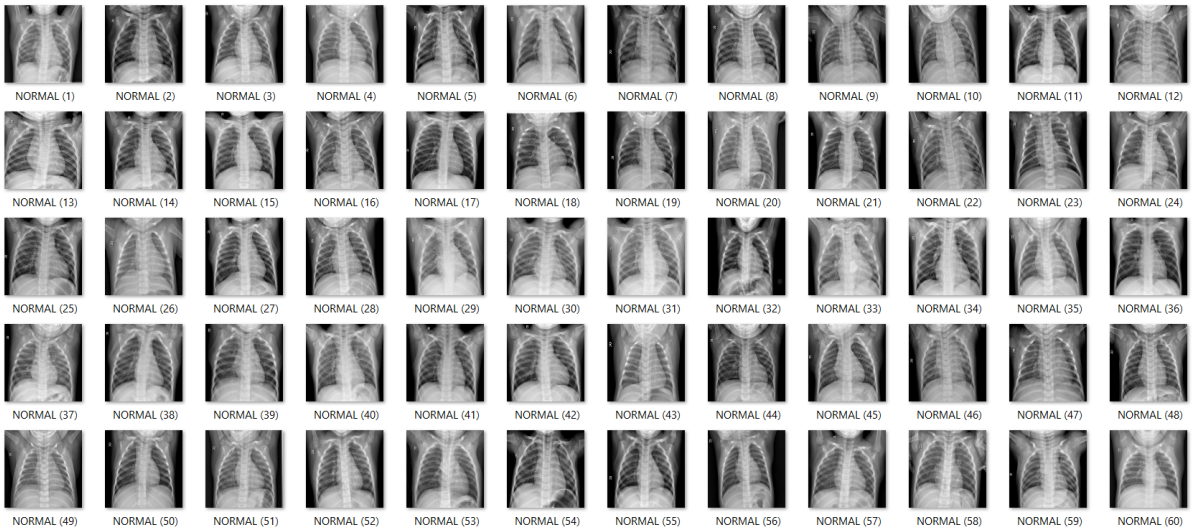


Figure (7): A figure that shows the pediatric bias in the negative class of the COVID-19 dataset. “Negative” images were replaced with ones from the NIH dataset

The dataset that is used for training on binary classification is a dataset that consists of 1134 COVID-19 positive images and 1150 negative images. However, I have only used 300 images from each class i.e 600 for training in total to examine the research question under significant data scarcity conditions, which is the motivation for using **TL** across medical datasets. The performance of the network was evaluated using 4 equally-sized datasets composed of the rest of the labeled images, each one containing 150 images for each class, to increase the certainty about the relative performance of the four models

3.3 The model architecture :

The model’s architecture that was used for the experiment is Krizhevsky’s Alexnet[38], a CNN architecture that consists of 5 convolutional layers followed by 3 FC layers as shown in **Figure (8)**. The reason behind choosing this architecture is its relatively low number of parameters compared to other architectures, given the limitations of available computational resources. Its output layer was adjusted to have one output for binary classification rather than 1000. Furthermore, significant modifications have been made to the architecture for experimenting. Originally, the architecture had both **BN** and **dropout** implemented, however, the architecture was modified to create two other model architectures, where each one has one of the two techniques implemented. Furthermore, the **baseline** and **data augmentation** models were created by removing both of the techniques from the original architecture. This was done to evaluate the impact of each regularization technique on pretraining on its own. The weights of all of the pretrained models were transferred to a unified Alexnet architecture,

to evaluate the generalizability of the learned parameters of each model, particularly by examining their performance on unseen data.

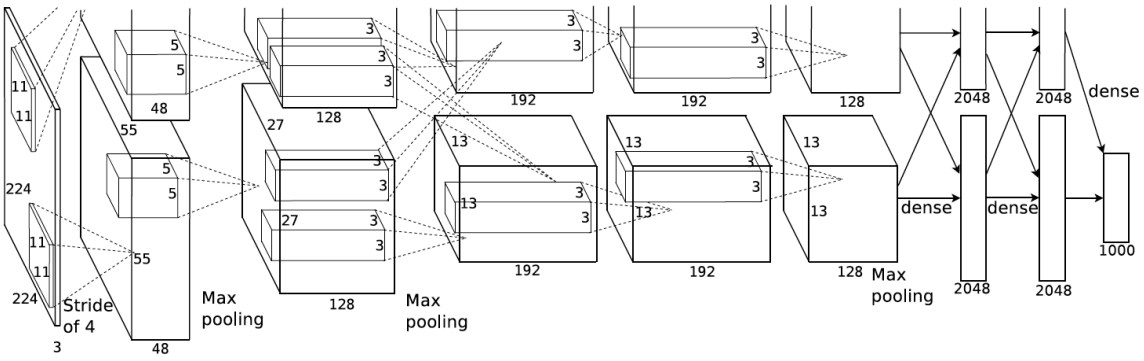


Figure (8): The architecture of the network used for the experiment: "AlexNet"[38]

3.4 Preprocessing :

To feed the data of the training dataset into the model, I have used Keras' **ImageDataGenerator**[39], to automate resizing the images to the fixed input size of AlexNet of 224* 224. Furthermore, all the images were preprocessed to grayscale, as I have noticed that many images in the tuberculosis dataset that are classified to be positive are colored as shown in Figure (9), while all the negative ones and the entire COVID-19 dataset images were grayscale, which could result in wrong generalizations that would affect the results.

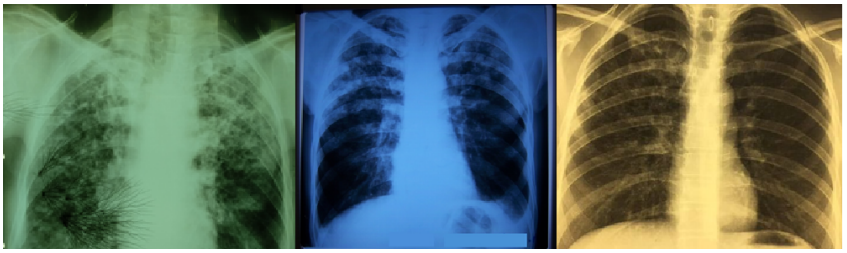
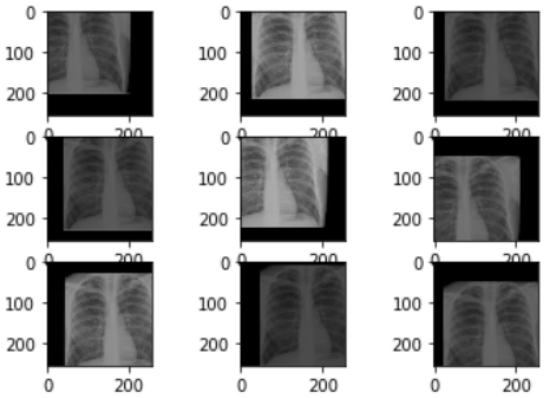


Figure (9): Some of the colored samples that were found in the tuberculosis dataset positive class

.

Besides preprocessing, **ImageDataGenerator** was used for applying **data augmentation** for one of the pretrained models, where several transformations were applied to artificially increase the size of the network. The transformations included: vertical shifts, horizontal shifts, zoom in and out, and contrast variations as shown in Figure (10).



Figure(10): A number of basic geometric transformations and variations of contrast applied to one of the tuberculosis CT-scans.

3.5 Hyperparameters used in the experiment :

Hyperparameter	Description	Hyperparameter Value	Notes
Learning rate	A constant that indicates the amount by which the parameters of the model are updated. [40]	Initial rate for **pre-training** = 10^{-2}. It was reduced by a factor of 10^{-2} when the learning curve plateaued. Learning rate for **fine-tuning** = 10^{-3}. It was kept constant	After experimenting, those were found to be the most suitable learning rate
Batch size	The number of the **images** in one batch, where the parameters are updated after each one.	64	-
Loss function	The objective function that assesses the	Binary cross-entropy	A loss function for binary classification

	performance of the model on each iteration	(CE)[41]	problems
Optimizer	The algorithm that is used in updating the parameters after error calculation	Adagrad[42]	Despite Adam being the state-of-the-art of optimization algorithms[43], Adagrad was found to have a more stable performance. This could be because of the absence of the momentum implementation in the optimizer**,** making Adagrad more stable
Number of epochs	The number of the complete iterations over the whole dataset	Pretraining: 25 Fine-tuning: 10	-

Table (1): The hyperparameters used in training the models

3.6 Metrics used in evaluation:

Given that all the datasets were balanced, there was no need for metrics that deal with unbalanced datasets. Accuracy was used for evaluation.

$$Accuracy(TP, TF, FP, FN) = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation (2): The formula for accuracy, where TP is true positive, TF is true negative, FP is false positive and FN is false negative. $Accuracy \in [0, 1]$, the higher the better[44]

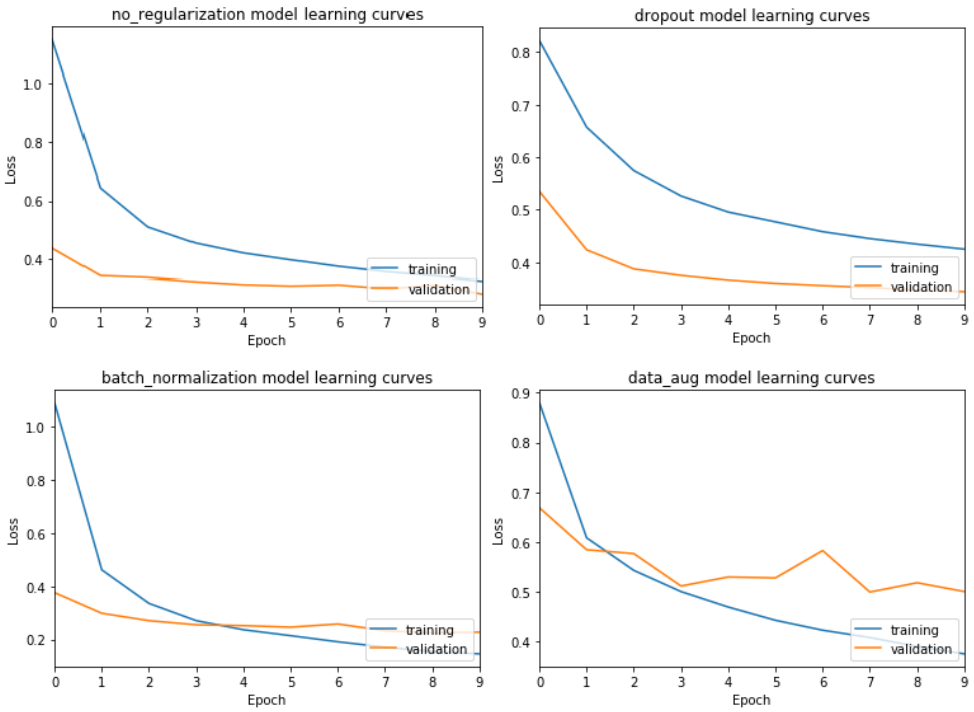
4. Experimental Results

Model	Validation loss on the source task (tuberculosis binary classification)	Validation accuracy on the source task (tuberculosis binary classification)
Without regularization during pre-training (baseline)	0.283	0.962
With data augmentation	0.124	0.974
With Dropout	0.085	0.986
With BN **(best performing**)	0.227	**0.989**

Table (2): The performance of the models after pretraining on tuberculosis binary classification

	Validation accuracy on the target task (COVID-19 binary classification)				
Model	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Average accuracy ± uncertainty
Without regularization (baseline)	0.8	0.8833	0.8933	0.85	0.856 ± 0.042
With data augmentation	0.7567	0.9167	0.9	0.84	0.853 ± 0.072
With Dropout	0.8067	0.8967	0.9	0.8633	0.866 ± 0.043
With BN (best performing)	**0.8633**	**0.9267**	**0.9167**	**0.93**	**0.909 ± 0.031**

Table (3): The performance of the models after fine-tuning on COVID-19 binary classification. Two out of the three regularization techniques outperformed the baseline model



Graph (1): The learning curves during the 10 epochs of fine-tuning on the target task (COVID-19 binary classification)

5. Discussion :

By examining the performances of the models during pretraining on the source-task (tuberculosis) in Table (3), it is noted that the regularization improves the performance of the models on the validation data i.e unseen data compared to the unregularized baseline model, which was only able to achieve an **accuracy** of **0.962 and loss** of **0.283.** The **BN** model had the best performance on the source task (tuberculosis). After the models were fine-tuned, the trend of the regularized models outperforming the unregularized baseline model on validation data almost continued, as we can see in Table (4). **Dropout** and **BN** models outperformed the **baseline,** where the dropout model achieved an average accuracy of **0.866 ± 0.043** and BN

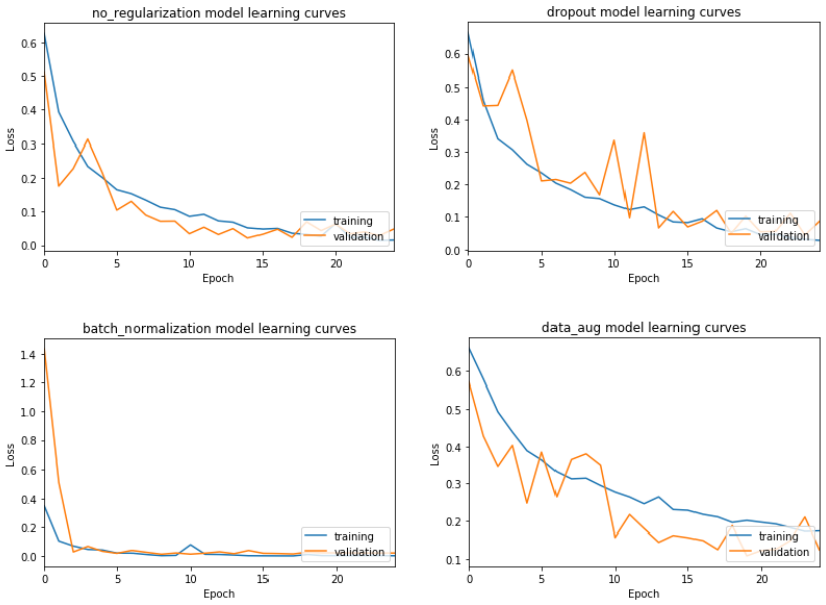
model achieved **0.909 ± 0.031**, compared to the baseline which only achieved an average accuracy of **0.856 ± 0.042** This has supported the presence of the positive effects of regularization of pretraining on TL performance across medical imaging datasets.

However, the data augmentation model has performed slightly worse on average, where it has only achieved an average accuracy of **0.853 ± 0.072** across the 4 datasets. Despite that, there are large variations in the performance of the data augmentation across the test datasets, which resulted in a large uncertainty (**0.072 in accuracy**), as it has performed extremely better on datasets (**2**) (**0.9167**) and (**3**) (**0.9**) compared to (**1**) (**0.757**) and (**4**) (**0.84**). This rendered the data inconclusive on the relative performance of the data augmentation model to the baseline.

Nevertheless, I argue that this experiment did not demonstrate the potential of **data augmentation** in pretraining for improving TL across medical datasets. I believe that this weaker performance on average compared to the baseline model was a result of the irrelevance of the transformations applied, which were to a great extent arbitrary given the absence of my medical-domain knowledge. In turn, this has limited the ability of the data augmentation to introduce variations that would be present in unseen CT-scans to the pretrained model, which has limited the potential of this regularization technique on improving TL performance on validation data of the target-task (COVID-19). More relevant transformations could be learned from medical experts that introduce relevant variations to the model could be implemented in future research. Furthermore, geometric transformations and contrast variations are a narrow class of data augmentation. There are more sophisticated

techniques of data augmentation such as Generative Adversarial Networks (GANs)[45], which could be also investigated in future research.

The best performing fine-tuned model was the BN model, where it has achieved an accuracy of **0.909 ± 0.031** after **10** epochs, outperforming the **other three** models. However, I believe that considering this improvement to be purely based on increased generalizability of the features learned would be an overestimation of the generalizability added to the model by the regularization effect of BN. In addition to regularization effects, BN also significantly fastened and stabilized learning on the source task (tuberculosis), as it makes the optimization landscape significantly smoother, this smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training[46]. This was a clear advantage to the **BN** pretrained model given the limited number of epochs undertaken due to the limitation of the available resources (**only 25**), where other models could have performed better with a larger number of epochs. This can be appreciated by comparing the learning curves of the different models on tuberculosis binary classification (source task). The BN model has converged only after 2 epochs, unlike every other model whose learning curves have not converged as completely. Despite that, the baseline model learning curve has almost converged in Graph(2), which can infer that not a significant improvement would be achieved by further epochs, which could compensate for the wide gap between the baseline and the BN models performance on unseen data (**0.053 in average accuracy**). Therefore, BN model performance remains as empirical support for the potential of regularization in pretraining for improving TL across medical imaging datasets



Graph (2): The learning curves during the 25 epochs of pretraining on the source task (tuberculosis binary classification)

6. Conclusion:

It was concluded that the regularization of the pretraining can contribute to a significant extent to the performance of TL across imaging medical datasets as proposed. Both **dropout** and **BN techniques in pretraining** have resulted in increased generalizability of their fine-tuned models, which was reflected in the superior performances of those models on validation data compared to the baseline one. However, the data for the **data augmentation model** could not support the presence of those positive effects of regularization of pretraining on TL performance, yet the discussion has shown the flaws in implementation may have prevented the experiment from showing those effects. Furthermore, the discussion has demonstrated how BN effects on training speed could have led to an inflated estimation to a certain extent of the improvement in the generalizability of the fine-tuned model, yet it showed how it remains an empirical support positive effects of regularization of the pretraining on the performance of TL across medical imaging datasets.

I . Bibliography

1. Andreas Maier 2020 “Will we ever solve the Shortage of Data in Medical Applications?” web, https://towardsdatascience.com/will-we-ever-solve-the-shortage-of-data-in-medical-applications-70da163e2c2d

2. Everitt B.S., Skrondal A. (2010), Cambridge Dictionary of Statistics, Cambridge University Press.

3. Kaur, T., Gandhi, T.K. Deep convolutional neural networks with transfer learning for automated brain image classification. Machine Vision and Applications 31, 20 (2020). https://doi.org/10.1007/s00138-020-01069-2

4. West, Jeremy; Ventura, Dan; Warnick, Sean (2007). "Spring Research Presentation: A Theoretical Foundation for Inductive Transfer". Brigham Young University, College of Physical and Mathematical Sciences. Archived from the original on 2007-08-01. Retrieved 2007-08-05.

5. Wang, S., Dong, L., Wang, X., & Wang, X. (2020). Classification of pathological types of lung cancer from CT images by deep residual neural networks with transfer learning strategy, *Open Medicine*, *15*(1), 190-197. DOI: https://doi.org/10.1515/med-2020-0028

6. Loris Nanni, Stefano Ghidoni, Sheryl Brahnam,2017 Handcrafted vs. non-handcrafted features for computer vision classification, Pages 158-172, ISSN 0031-3203, https://doi.org/10.1016/j.patcog.2017.05.025.

7. Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9

8. "Artificial Intelligence Can Change the future of Medical Diagnosis | Dr.Shinjini Kundu | TEDxPittsburgh". YouTube. Retrieved 2017-08-12.

9. Géron, Aurélien (2018) “Neural networks and deep learning”

10. Zimbres, Rubens (2016) “Matrix Multiplication in Neural Networks”, web https://www.datasciencecentral.com/profiles/blogs/matrix-multiplication-in-neural-networks

11. Jordan, Jeremy (2017) Neural networks: representation, web, https://www.jeremyjordan.me/intro-to-neural-networks/

12. Géron, Aurélien (2018) “Neural networks and deep learning”

13. Géron, Aurélien (2018) “Neural networks and deep learning”

14. Murphy, Kevin P. (2012). Machine Learning: A Probabilistic Perspective. Cambridge: MIT Press. p. 247. ISBN 978-0-262-01802-9.

15. Ruder, Sebastian(2016) An overview of gradient descent optimization algorithms, web, https://ruder.io/optimizing-gradient-descent/

16. Zewen Li, Wenjie Yang, Shouheng Peng, Fan Liu (2020) A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects https://arxiv.org/abs/2004.02806

17. Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9

18. NO Mahony (2019), Deep Learning vs. Traditional Computer Vision https://arxiv.org/ftp/arxiv/papers/1910/1910.13796.pdf

19. R. Fisher, S. Perkins, A. Walker, and E. Wolfart (2003) Sobel Edge Detector, http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

20. (2017). "CS231n Convolutional Neural Networks for Visual Recognition". *cs231n.github.io*.

21. .Alexander Amini (January 2020), *Convolutional Neural Networks for Computer Vision*. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/. License: Creative Commons BY-NC-SA.

22. “Activation Functions (Linear/Non-linear) in Deep Learning” (2020) https://xzz201920.medium.com/activation-functions-linear-non-linear-in-deep-learning-relu-sigmoid-softmax-swish-leaky-relu-a6333be712ea

23. Géron, Aurélien (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastopol, CA: O'Reilly Media. ISBN 9780226484648., pp. 448

24. Kukačka (2017), “Regularization for Deep Learning: A Taxonomy” https://arxiv.org/abs/1710.10686

25. Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0

26. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov (2014) "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

27. Ioffe, Szegedy (2015), Accelerating Deep Network Training by Reducing Internal Covariate Shift https://arxiv.org/abs/1502.03167

28. Michael Bereket, 2017, How Different Are Cats and Cells Anyway?, Stanford AI for Healthcare, https://medium.com/stanford-ai-for-healthcare/how-different-are-cats-and-cells-anyway-closing-the-gap-for-deep-learning-in-histopathology-14f92d0ddb63

29. Cortés, Ernesto. Sánchez, Sergio (2020) Deep Learning Transfer with AlexNet for chest X-ray COVID-19 recognition

30. WanWang, S., Dong, L., Wang, X., & Wang, X. (2020). Classification of pathological types of lung cancer from CT images by deep residual neural networks with transfer learning strategy, *Open Medicine*, *15*(1), 190-197. DOI: https://doi.org/10.1515/med-2020-0028

31. McCloskey, Michael; Cohen, Neal J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. Psychology of Learning

and Motivation. 24. pp. 109–165. doi:10.1016/S0079-7421(08)60536-8. ISBN 978-0-12-543324-2.

32. https://www.tensorflow.org/

33. https://www.kaggle.com/

34. https://www.nvidia.com/en-gb/data-center/tesla-k80/

35. Tawsifur Rahman, Amith Khandakar, Muhammad A. Kadir, Khandaker R. Islam, Khandaker F. Islam, Zaid B. Mahbub, Mohamed Arselene Ayari, Muhammad E. H. Chowdhury. (2020) "Reliable Tuberculosis Detection using Chest X-ray with Deep Learning, Segmentation and Visualization". IEEE Access, Vol. 8, pp 191586 - 191601. DOI. 10.1109/ACCESS.2020.3031384.https://www.kaggle.com/tawsifurrahman/tuberculosis-tb-chest-xray-dataset

36. M.E.H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al-Emadi, M.B.I. Reaz, M. T. Islam, “Can AI help in screening Viral and COVID-19 pneumonia?” IEEE Access, Vol. 8, 2020, pp. 132665 - 132676.https://www.kaggle.com/tawsifurrahman/covid19-radiography-database

37. Wang X, Peng Y, Lu L, Lu Z, Bagheri M, Summers RM. ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. IEEE CVPR 2017 https://www.kaggle.com/nih-chest-xrays/sample

38. Krizhevsky, Sutskever, Hinton (2012) ImageNet classification with deep convolutional neural networks https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

39. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

40. Murphy, Kevin P. (2012). Machine Learning: A Probabilistic Perspective. Cambridge: MIT Press. p. 247. ISBN 978-0-262-01802-9.

41. Murphy, Kevin (2012). Machine Learning: A Probabilistic Perspective. MIT. ISBN 978-0262018029.

42. Duchi (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

43. Kingma, Jimmy Ba (2014), Adam: A Method for Stochastic Optimization, https://arxiv.org/abs/1412.6980

44. Metz, CE (October 1978). "Basic principles of ROC analysis" (PDF). Semin Nucl Med. 8 (4): 283–98. PMID 112681

45. Sandfort, V., Yan, K., Pickhardt, P.J. *et al.* Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks. *Sci Rep* 9, 16884 (2019). https://doi.org/10.1038/s41598-019-52737-x

46. Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, Aleksander Madry (2018) “How Does Batch Normalization Help Optimization?” https://arxiv.org/abs/1805.11604

II. Appendix

```
1- Code for the experiments :
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D,
MaxPooling2D,MaxPool2D
from keras.layers.normalization import BatchNormalization
import numpy as np
np.random.seed(42)
from keras.models import load_model
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import regularizers
import imageio
import seaborn
import shutil
seaborn.set()

tech_name = "data_aug"
all_dict = "../input/tuberculosis-tb-chest-xray-dataset/Dataset"
pre_data_gen = ImageDataGenerator(rescale=1.0/255,
        shear_range=0.2,
        zoom_range=0.2,
        brightness_range=(0.2, 0.8),
        width_shift_range=0.2,
        height_shift_range=0.2)

pre_train_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "training")

metrics = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        keras.metrics.AUC(name="auc"),

]
```

```
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(256,256,1), kernel_size=(11,11),
strides=(4,4), padding='same'))
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1),
padding='same'))
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(Activation('relu'))
# Add Dropout to prevent overfitting

#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(Activation('relu'))
#Add Dropout

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(Activation('relu'))
#Add Dropout
```

```
#Output Layer
AlexNet.add(Dense(1))
AlexNet.add(Activation('sigmoid'))


pre_val_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "validation")
csv_logger = tf.keras.callbacks.CSVLogger('training_pretrained_' + tech_name
+".csv", append=True)


batch_size = 64
history = tf.keras.callbacks.History()
opt = keras.optimizers.Adagrad(learning_rate= 1e-2)
reduceLR = tf.keras.callbacks.ReduceLROnPlateau(mintor="loss",min_lr=1e-6,
factor=1e-2)


callbacks_list = [csv_logger,history,reduceLR]

AlexNet.compile(optimizer=opt,loss='binary_crossentropy',metrics=metrics)
history = AlexNet.fit(x = pre_train_gen, validation_data = "pre_val_gen",
epochs = 9, batch_size = batch_size,callbacks = callbacks_list,
shuffle=True)

AlexNet.save_weights( "alexnet_pretrainedTB_" + tech_name + ".h5")

tech_name = "dropout"
all_dict = "../input/tuberculosis-tb-chest-xray-dataset/Dataset"
pre_data_gen = ImageDataGenerator(rescale=1.0/255, validation_split = 0.05)

pre_train_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "training")
pre_val_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "validation")
metrics = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        keras.metrics.AUC(name="auc"),

]
```

```
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(256,256,1), kernel_size=(11,11),
strides=(4,4), padding='same'))
AlexNet.add(Activation('relu'))
# AlexNet.add(BatchNormalization())
# AlexNet.add(Dropout(0.4))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5),  strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.3))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))


#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384,  kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.3))


#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384,  kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.3))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.3))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))


#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
```

```
AlexNet.add(Activation('relu'))


#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(Activation('relu'))

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(Activation('relu'))

#Output Layer
AlexNet.add(Dense(1))
AlexNet.add(Activation('sigmoid'))

csv_logger = tf.keras.callbacks.CSVLogger('training_pretrained_' + tech_name
+".csv", append=True)


batch_size = 64
history = tf.keras.callbacks.History()
opt = keras.optimizers.Adagrad(learning_rate= 1e-2)
reduceLR =
tf.keras.callbacks.ReduceLROnPlateau(mintor="val_loss",min_lr=1e-6,
factor=1e-2)


callbacks_list = [csv_logger,history,reduceLR]

AlexNet.compile(optimizer=opt,loss='binary_crossentropy',metrics=metrics)
history = AlexNet.fit(x = pre_train_gen, validation_data =pre_val_gen,
epochs = 25, batch_size = batch_size,callbacks = callbacks_list,
shuffle=True)
AlexNet.save_weights( "alexnet_pretrainedTB_" + tech_name + ".h5")
tech_name = "bn"
all_dict = "../input/tuberculosis-tb-chest-xray-dataset/Dataset"
pre_data_gen = ImageDataGenerator(rescale=1.0/255, validation_split = 0.05)

pre_train_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "training")
pre_val_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "validation")
metrics = [
        'accuracy',
```

```
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        keras.metrics.AUC(name="auc"),

]

AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(256,256,1), kernel_size=(11,11),
strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5),  strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384,  kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384,  kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(Activation('relu'))
```

```
#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(Activation('relu'))

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(Activation('relu'))

#Output Layer
AlexNet.add(Dense(1))
AlexNet.add(Activation('sigmoid'))

AlexNet.summary()

csv_logger = tf.keras.callbacks.CSVLogger('training_pretrained_' + tech_name
+".csv", append=True)


batch_size = 64
history = tf.keras.callbacks.History()
opt = keras.optimizers.Adagrad(learning_rate= 1e-2)
reduceLR =
tf.keras.callbacks.ReduceLROnPlateau(mintor="val_loss",min_lr=1e-6,
factor=1e-2)


callbacks_list = [csv_logger,history,reduceLR]

AlexNet.compile(optimizer=opt,loss='binary_crossentropy',metrics=metrics)
history = AlexNet.fit(x = pre_train_gen,validation_data = pre_val_gen
,epochs = 25, batch_size = batch_size,callbacks = callbacks_list,
shuffle=True)
AlexNet.save_weights( "alexnet_pretrainedTB_" + tech_name + ".h5")

tech_name = "noreg"
all_dict = "../input/tuberculosis-tb-chest-xray-dataset/Dataset"
pre_data_gen = ImageDataGenerator(rescale=1.0/255, validation_split = 0.05)
pre_train_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "training")
pre_val_gen = pre_data_gen.flow_from_directory(directory = all_dict,
classes=["Normal",
"Tuberculosis"],color_mode='grayscale',class_mode="binary",target_size=(256,
256), shuffle = True, subset = "validation")

metrics = [
        'accuracy',
```

```
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        keras.metrics.AUC(name="auc"),

]

AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(256,256,1), kernel_size=(11,11),
strides=(4,4), padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5),  strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384,  kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384,  kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(Activation('relu'))
```

```
#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(Activation('relu'))

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(Activation('relu'))

#Output Layer
AlexNet.add(Dense(1))
AlexNet.add(Activation('sigmoid'))

csv_logger = tf.keras.callbacks.CSVLogger('training_pretrained_' + tech_name
+".csv", append=True)


batch_size = 64
history = tf.keras.callbacks.History()
opt = keras.optimizers.Adagrad(learning_rate= 1e-2)
reduceLR = tf.keras.callbacks.ReduceLROnPlateau(mintor="loss",min_lr=1e-6,
factor=1e-2)


callbacks_list = [csv_logger,history,reduceLR]

AlexNet.compile(optimizer=opt,loss='binary_crossentropy',metrics=metrics)
history = AlexNet.fit(x = pre_train_gen, validation_data = pre_val_gen,
epochs = 25, batch_size = batch_size,callbacks = callbacks_list,
shuffle=True)
AlexNet.save_weights( "alexnet_pretrainedTB_" + tech_name + ".h5")

    batch_size = 64


dataset = "dbcovid30"

train_dict = os.path.join("../input/",dataset)
test_dict = "../input/covidtestsets/COVID-DB-TEST-"

data_gen = ImageDataGenerator(rescale=1.0/255, validation_split = 0.05)

train_gen = data_gen.flow_from_directory(directory = train_dict,classes =
["NORMAL", "COVID-19"], class_mode="binary",target_size=(256, 256),
subset="training")
val_gen = data_gen.flow_from_directory(directory = train_dict ,classes =
["NORMAL", "COVID-19"], class_mode="binary", target_size=(256, 256),
subset="validation")
```

```
    ###################################
    #Instantiation
AlexNet_TL = Sequential()

#1st Convolutional Layer
AlexNet_TL.add(Conv2D(filters=96, input_shape=(256,256,1),
kernel_size=(11,11), strides=(4,4), padding='same'))
# AlexNet_TL.add(BatchNormalization())

AlexNet_TL.add(Activation('relu'))
# AlexNet.add(Dropout(0.2))
AlexNet_TL.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))


#2nd Convolutional Layer
AlexNet_TL.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1),
padding='same'))
# AlexNet_TL.add(BatchNormalization())
AlexNet_TL.add(Activation('relu'))
# AlexNet.add(Dropout(0.3))
AlexNet_TL.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))


#3rd Convolutional Layer
AlexNet_TL.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet_TL.add(BatchNormalization())
AlexNet_TL.add(Activation('relu'))
# AlexNet.add(Dropout(0.3))


#4th Convolutional Layer
AlexNet_TL.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet_TL.add(BatchNormalization())
# AlexNet.add(Dropout(0.3))


AlexNet_TL.add(Activation('relu'))

#5th Convolutional Layer
AlexNet_TL.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
# AlexNet_TL.add(BatchNormalization())
AlexNet_TL.add(Activation('relu'))
# AlexNet.add(Dropout(0.3))
```

```
AlexNet_TL.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))


#Passing it to a Fully Connected layer
AlexNet_TL.add(Flatten())
# 1st Fully Connected Layer
AlexNet_TL.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet_TL.add(Activation('relu'))
# Add Dropout to prevent overfitting

#2nd Fully Connected Layer
AlexNet_TL.add(Dense(4096))
AlexNet_TL.add(Activation('relu'))

#3rd Fully Connected Layer
AlexNet_TL.add(Dense(1000))
AlexNet_TL.add(Activation('relu'))

#Output Layer
AlexNet_TL.add(Dense(1))
AlexNet_TL.add(Activation('sigmoid'))

###################################


AlexNet_TL.load_weights("../input/data-aug/alexnet_pretrainedTB_data.h5")




history_TL = tf.keras.callbacks.History()
csv_logger_TL = tf.keras.callbacks.CSVLogger("model : " + "L2" + ".csv",
append=True)
callbacks_TL = [ csv_logger_TL ,history_TL ]

opt_TL = keras.optimizers.Adagrad(learning_rate=1e-3)

for l in AlexNet_TL.layers[:-9] :
        l.trainable = False


metrics_TL = [
    'accuracy',
tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'),
keras.metrics.AUC(name="auc")]
```

```
AlexNet_TL.compile(optimizer=opt_TL,loss='binary_crossentropy',metrics=metrics_TL)

TL_history = AlexNet_TL.fit(x = train_gen, validation_data = val_gen, epochs = 10 , batch_size = batch_size, callbacks= callbacks_TL)



for k in range(1,5) :

    test_gen = data_gen.flow_from_directory(directory = test_dict + str(k),classes = ["NORMAL", "COVID-19"], class_mode="binary", target_size=(256, 256))


    print("testing for model : " + "data" + " test-dataset : " + str (k))


    AlexNet_TL.evaluate(test_gen)
```

2- Data collected

Pretraining (no regularization “baseline”)

epoch	accuracy	auc	loss	precision	recall	val_accuracy	val_auc	val_loss	val_precision	val_recall
0	0.675	0.737	0.627	0.659	0.726	0.714	0.989	0.512	1.000	0.429
1	0.832	0.904	0.394	0.830	0.836	0.951	0.990	0.176	0.920	0.989
2	0.872	0.942	0.307	0.871	0.873	0.909	0.991	0.228	0.845	1.000
3	0.908	0.967	0.234	0.911	0.904	0.863	0.989	0.313	0.785	1.000
4	0.921	0.975	0.201	0.918	0.924	0.909	0.991	0.216	0.845	1.000
5	0.940	0.983	0.165	0.942	0.938	0.954	0.996	0.105	0.930	0.983
6	0.943	0.985	0.153	0.944	0.942	0.943	0.998	0.131	0.897	1.000
7	0.950	0.989	0.134	0.949	0.951	0.966	0.997	0.090	0.988	0.943
8	0.957	0.992	0.113	0.956	0.958	0.969	0.999	0.071	0.946	0.994
9	0.961	0.993	0.106	0.960	0.962	0.971	0.998	0.072	0.951	0.994
10	0.969	0.995	0.086	0.970	0.969	0.989	0.999	0.034	0.983	0.994
11	0.965	0.995	0.092	0.963	0.966	0.977	0.999	0.054	0.961	0.994
12	0.973	0.997	0.073	0.972	0.974	0.983	1.000	0.032	0.972	0.994

13	0.976	0.997	0.069	0.976	0.975	0.983	1.000	0.050	0.972	0.994
14	0.982	0.998	0.052	0.982	0.982	0.991	1.000	0.022	0.989	0.994
15	0.983	0.998	0.048	0.983	0.982	0.980	1.000	0.033	0.962	1.000
16	0.982	0.998	0.050	0.983	0.981	0.977	0.999	0.048	0.961	0.994
17	0.987	0.999	0.036	0.987	0.987	0.991	1.000	0.024	0.989	0.994
18	0.988	0.999	0.032	0.989	0.987	0.971	1.000	0.070	0.946	1.000
19	0.990	1.000	0.027	0.990	0.989	0.983	1.000	0.044	0.972	0.994
20	0.974	0.997	0.065	0.973	0.975	0.977	0.998	0.063	0.983	0.971
21	0.993	1.000	0.023	0.992	0.993	0.983	1.000	0.034	0.972	0.994
22	0.991	0.999	0.024	0.992	0.990	0.983	1.000	0.040	0.972	0.994
23	0.995	1.000	0.015	0.996	0.995	0.986	1.000	0.029	0.978	0.994
24	0.995	1.000	0.016	0.996	0.995	0.986	1.000	0.048	0.972	1.000

Pretraining (dropout) :

epoch	accuracy	auc	loss	precision	recall	val_accuracy	val_auc	val_loss	val_precision	val_recall
0	0.5967	0.6642	0.6655	0.5909	0.6286	0.8457	0.9683	0.5954	0.7665	0.9943
1	0.8008	0.8702	0.4580	0.8025	0.7979	0.8829	0.9774	0.4415	0.9653	0.7943
2	0.8692	0.9291	0.3400	0.8665	0.8728	0.8343	0.9634	0.4430	0.7511	1.0000
3	0.8771	0.9438	0.3059	0.8777	0.8764	0.7000	0.9567	0.5494	0.6250	1.0000
4	0.8970	0.9588	0.2613	0.9007	0.8923	0.8486	0.9963	0.3972	0.7675	1.0000
5	0.9048	0.9665	0.2344	0.9116	0.8965	0.9371	0.9964	0.2097	0.8923	0.9943
6	0.9206	0.9747	0.2028	0.9237	0.9170	0.9486	0.9951	0.2141	0.9110	0.9943
7	0.9304	0.9793	0.1831	0.9365	0.9233	0.9143	0.9989	0.2026	0.8537	1.0000
8	0.9405	0.9840	0.1593	0.9442	0.9362	0.9029	0.9962	0.2361	0.8373	1.0000
9	0.9432	0.9849	0.1551	0.9448	0.9414	0.9371	0.9993	0.1668	0.8883	1.0000
10	0.9474	0.9882	0.1358	0.9501	0.9444	0.8200	0.9958	0.3357	0.7353	1.0000
11	0.9550	0.9906	0.1212	0.9571	0.9528	0.9657	0.9996	0.0958	0.9358	1.0000
12	0.9496	0.9889	0.1296	0.9517	0.9474	0.8257	0.9978	0.3587	0.7415	1.0000
13	0.9611	0.9927	0.1046	0.9593	0.9630	0.9829	0.9999	0.0644	0.9669	1.0000
14	0.9702	0.9952	0.0830	0.9708	0.9696	0.9457	1.0000	0.1157	0.9021	1.0000

15	0.9699	0.9954	0.0808	0.9705	0.9693	0.9857	0.9999	0.0675	0.9722	1.0000
16	0.9659	0.9945	0.0928	0.9641	0.9678	0.9771	1.0000	0.0850	0.9563	1.0000
17	0.9777	0.9971	0.0642	0.9760	0.9795	0.9371	1.0000	0.1186	0.8883	1.0000
18	0.9808	0.9981	0.0526	0.9810	0.9805	0.9857	1.0000	0.0477	0.9722	1.0000
19	0.9777	0.9970	0.0622	0.9786	0.9768	0.9629	0.9996	0.1011	0.9309	1.0000
20	0.9845	0.9986	0.0438	0.9849	0.9841	0.9771	1.0000	0.0528	0.9563	1.0000
21	0.9839	0.9984	0.0448	0.9832	0.9847	0.9771	1.0000	0.0539	0.9563	1.0000
22	0.9884	0.9990	0.0350	0.9880	0.9889	0.9429	1.0000	0.1101	0.8974	1.0000
23	0.9901	0.9993	0.0326	0.9889	0.9913	0.9857	1.0000	0.0425	0.9722	1.0000
24	0.9883	0.9993	0.0293	0.9871	0.9895	0.9600	1.0000	0.0846	0.9259	1.0000

Pretraining (batch normalization) :

epoch	accuracy	auc	loss	precision	recall	val_accuracy	val_auc	val_loss	val_precision	val_recall
0	0.894	0.954	0.349	0.893	0.897	0.500	0.884	1.433	0.500	1.000
1	0.963	0.993	0.103	0.964	0.963	0.840	0.973	0.512	0.758	1.000
2	0.976	0.997	0.068	0.977	0.974	0.991	1.000	0.028	0.983	1.000
3	0.985	0.998	0.045	0.985	0.985	0.974	0.998	0.066	0.972	0.977
4	0.985	0.999	0.041	0.986	0.983	0.986	1.000	0.032	0.972	1.000
5	0.994	1.000	0.020	0.995	0.992	0.991	1.000	0.018	0.983	1.000
6	0.994	1.000	0.019	0.995	0.993	0.983	1.000	0.038	0.967	1.000
7	0.997	1.000	0.010	0.997	0.996	0.986	1.000	0.025	0.978	0.994
8	1.000	1.000	0.002	1.000	0.999	0.991	1.000	0.012	0.983	1.000
9	0.999	1.000	0.004	0.999	0.999	0.989	1.000	0.021	0.978	1.000
10	0.981	0.994	0.077	0.981	0.980	0.994	1.000	0.012	0.989	1.000
11	0.997	1.000	0.011	0.998	0.995	0.989	1.000	0.019	0.978	1.000
12	0.996	1.000	0.010	0.996	0.997	0.989	1.000	0.028	0.978	1.000
13	0.998	1.000	0.006	0.998	0.998	0.991	1.000	0.016	0.983	1.000
14	1.000	1.000	0.002	1.000	1.000	0.986	1.000	0.038	0.972	1.000
15	1.000	1.000	0.001	1.000	1.000	0.991	1.000	0.018	0.983	1.000
16	1.000	1.000	0.001	1.000	1.000	0.989	1.000	0.017	0.978	1.000
17	1.000	1.000	0.000	1.000	1.000	0.991	1.000	0.014	0.983	1.000
18	0.997	1.000	0.010	0.997	0.998	0.986	1.000	0.026	0.978	0.994
19	0.9995	1.0000	0.0025	1.0000	0.9991	0.9886	0.9998	0.0220	0.9831	0.9943
20	0.9994	1.0000	0.0022	0.9997	0.9991	0.9886	0.9998	0.0210	0.9831	0.9943

21	0.9995	1.0000	0.0020	0.9994	0.9997	0.9886	0.9998	0.0210	0.9831	0.9943
22	0.9988	1.0000	0.0026	0.9985	0.9991	0.9886	0.9998	0.0201	0.9831	0.9943
23	0.9992	1.0000	0.0023	0.9997	0.9988	0.9886	0.9998	0.0204	0.9831	0.9943
24	0.9999	1.0000	0.0014	1.0000	0.9997	0.9886	0.9998	0.0202	0.9831	0.9943

Pretraining (data augmentation) :

epoch	accuracy	auc	loss	precision	recall	val_accuracy	val_auc	val_loss	val_precision	val_recall
0	0.618	0.669	0.661	0.604	0.689	0.649	0.836	0.572	0.894	0.337
1	0.725	0.785	0.579	0.739	0.696	0.823	0.921	0.428	0.742	0.989
2	0.792	0.851	0.493	0.787	0.800	0.897	0.953	0.346	0.897	0.897
3	0.811	0.883	0.438	0.809	0.815	0.809	0.929	0.403	0.897	0.697
4	0.837	0.909	0.388	0.837	0.837	0.931	0.968	0.248	0.936	0.926
5	0.848	0.920	0.364	0.844	0.854	0.820	0.973	0.385	0.737	0.994
6	0.863	0.934	0.332	0.864	0.863	0.926	0.978	0.264	0.941	0.909
7	0.871	0.941	0.313	0.872	0.870	0.829	0.981	0.365	0.745	1.000
8	0.868	0.941	0.314	0.870	0.866	0.803	0.974	0.380	0.717	1.000
9	0.877	0.948	0.295	0.878	0.875	0.849	0.977	0.349	0.770	0.994
10	0.881	0.954	0.277	0.878	0.884	0.949	0.993	0.157	0.911	0.994
11	0.890	0.958	0.264	0.892	0.888	0.909	0.974	0.217	0.870	0.960
12	0.900	0.963	0.246	0.901	0.898	0.929	0.992	0.180	0.883	0.989
13	0.890	0.958	0.264	0.891	0.890	0.949	0.990	0.144	0.920	0.983
14	0.907	0.968	0.231	0.911	0.902	0.931	0.994	0.162	0.879	1.000
15	0.909	0.968	0.229	0.912	0.906	0.951	0.986	0.157	0.934	0.971
16	0.909	0.971	0.218	0.910	0.908	0.954	0.991	0.149	0.934	0.977
17	0.917	0.973	0.211	0.920	0.914	0.957	0.989	0.125	0.960	0.954
18	0.921	0.976	0.196	0.923	0.918	0.929	0.984	0.188	0.887	0.983
19	0.920	0.975	0.202	0.923	0.917	0.974	0.995	0.108	0.977	0.971
20	0.926	0.976	0.197	0.933	0.918	0.974	0.994	0.123	0.961	0.989
21	0.926	0.977	0.192	0.930	0.920	0.954	0.994	0.125	0.925	0.989
22	0.926	0.980	0.182	0.931	0.921	0.940	0.987	0.149	0.938	0.943
23	0.933	0.982	0.173	0.936	0.929	0.909	0.982	0.211	0.967	0.846
24	0.933	0.981	0.174	0.937	0.928	0.951	0.993	0.124	0.929	0.977