EXTENDED ESSAY

Topic: Comparison of Two Deep Learning Models' Data Adaptability in AI Medicine

Research Question:

To what extent is Vision Transformer (ViT) better in terms of robustness compared to ResNet in retinal imaging-based prediction?

Subject Area: Computer Science

Report word count: 3975

Session: May 2023

Contents

INTRODUCTION

Diabetes is a chronic disease characterized by elevated levels of blood glucose caused by overweight, obesity and an inactive lifestyle, which over time would lead to serious damage to our organs. Statistics show that 1.5 million deaths are directly attributed to diabetes each year, out of a total population of 422 million (WHO). While diabetes is incurable, a person can stay in remission by being aware and engaging in a healthy lifestyle. Thus, the ability to diagnose and warn if one has diabetes is highly valuable.

Compared to the conventional way of diagnosis, Artificial Intelligence (AI) driven disease prediction not only frees the doctors from diagnosing patients, so they can focus more on patient care, but also is proven to have greater accuracy in detecting diseases (Mantelakis, et al).

Diabetes is among the dozens of known diseases that can be predicted by AI models based on patients' retinal imaging. However, in real-life, these images might not be ideal for the models to "consume". Noises like overexposure, underexposure, or low-quality camera can influence the results of diagnosis (Belde). A model's ability to adapt to noisy data and maintain accurate result is called robustness.

This research paper aims to explore the robustness of two models, ResNet and Vision Transformer (ViT), in the field of diabetic retinal detection by comparing their performance in different types of noisy data under similar training conditions. The following will cover theoretical knowledge first, followed by experimentation.

BACKGROUND

AI, ML and DL

AI describes efforts to train computers to imitate a human's ability to solve problems. A subset of AI, machine learning (ML) means computers' learning from data using algorithms to perform tasks without being explicitly programmed. Deep learning (DL) is a branch of ML that uses neural networks. See Figure 1 for a summary.

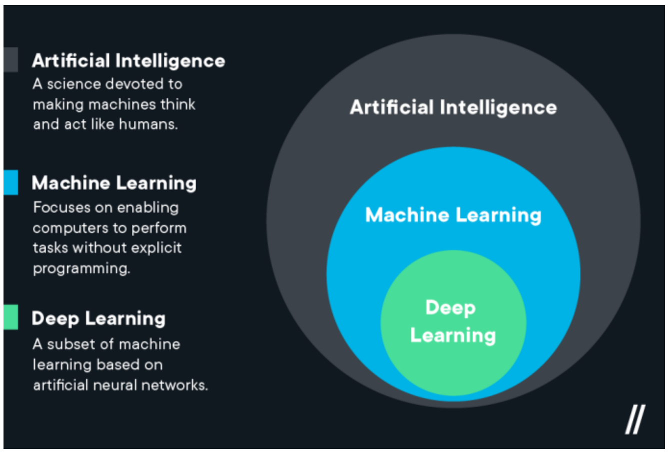
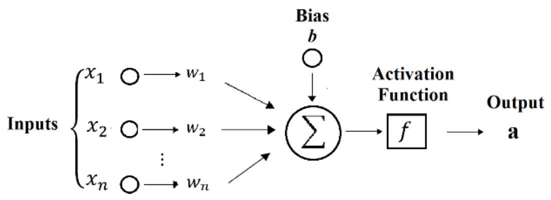

Artificial Intelligence
A science devoted to making machines think and act like humans.
Machine Learning
Focuses on enabling computers to perform tasks without explicit programming.
Deep Learning
A subset of machine learning based on artificial neural networks.
Artificial Intelligence
Machine Learning
Deep Learning

Figure 1. Deep learning, machine learning and Artificial Intelligence (Middleton)

Deep Learning is extensively used today in applications such as image classification, natural language processing, autonomous driving, healthcare, and fraud detection (IBM).

Neural Networks

Neural networks (NNs) are at the heart of deep learning algorithms. Designed to mimic human neural networks, a NN consists of connected neural nodes.



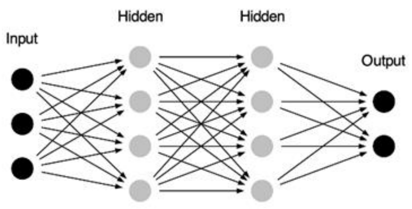


Fig 2a. A typical neural node in a NN

Fig 2b. An example NN

A neural node is illustrated in Figure 2a, it takes inputs from vector $X\ (x_1, x_2, x_3, \ldots\ x_n)$, has a node specific bias value (b) and maintains a weight value vector $\vec{w}$ storing w_i for each input node (x_i). The bias (b) decides the threshold value of the node's activation, while the weight value decides the level of importance (weight) of an input. A neural node does two functions:

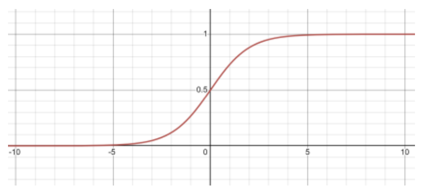
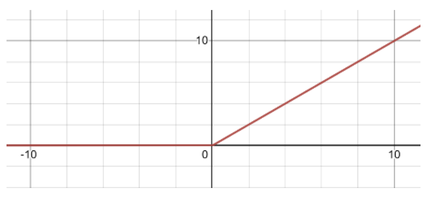
1. **Summation** (z) of the input values (x_i) with respective weights (w_i) and bias (b),

$$z = x_1 * w_1 + x_2 * w_2 + \ldots + x_n * w_n + b$$

2. **Activation** of the input signal using function $f(z)$, with a as the output, and the summation value z as input,

$$a = f(z)$$

The activation function $f(z)$ is essential to the function of the neural node because it adds non-linearity to the signal, just like the selective activation mechanism of a human neuron. There are a variety of such functions, some of the common ones include Sigmoid, ReLU, Softmax, etc (Table 1).

Name	$\boldsymbol{f(z)}$	**Graph**	**Range**
Sigmoid	$f(z) = \dfrac{1}{1+e^{-z}}$		(0,1)
ReLU	$f(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$		$(0, \infty)$

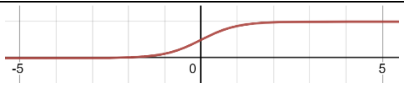
Softmax	$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}$		(0,1)

Table 1. Summary of some common activation functions.

Figure 2b illustrates a NN consisting of one input layer, two hidden layers and one output layer. The **input layer** takes raw input from and passes on the information (features) to the first hidden layer. As their name suggest, the hidden layers are not exposed. A **hidden layer** performs computations on the features entered through the previous layer (input layer or hidden layer) and transfers the result to the next layer (a hidden layer or output layer). The **output layer** brings the information learned through the hidden layer and delivers the final value as a result.

Data flows through each layer, goes through each node's summation and activation, with previous layer's output as input for next layer, until the final output. This process of passing data from one layer to the next layer defines this NN as a **feedforward** network.

After the feedforward is complete, a NN calculates the **Loss/Cost** C using the outputs against the training example's labelled results (target vector y). A common way of computing loss is **Categorical-Cross-Entropy** (**CCE**), defined as

$$C = -\sum_{i=1}^{N} (y_i \log_e a_i^{(L)})$$

where y_i is the truth label (taking a value 0 or 1) from the target vector y, and L stands for the output layer consisting of N nodes.

A NN's behavior is determined by its weights and biases and achieves learning by:

1. (for a new NN) initialize each node's weights with random values , biases with 0s
2. feed forward the training example data through all layers of the NN to compute outputs
3. calculate the loss/error C, which is a function of weights and biases
4. use a mechanism called **gradient descent** and **back propagation** to adjust each node's weights and bias values to minimize the loss and improve precision of outputs
5. keep doing steps 1 to 4 until achieving maximum precision and minimal loss. This usually takes many epochs. An **epoch** means training the neural network with all the training data for one cycle.

Gradient descent

Gradient descent starts at a random input and figures out which direction to step to yield a lower function value; specifically, we take the derivative/slope of the function at the point, if the slope is negative, shift a "learning step" to the right. If the slope is positive, shift a "learning step" to the left. At the new position we check the slope and repeat the above until the derivative is 0 or nearly 0, meaning we are at the minimum or near it, like a ball rolling down a hill. Figure 3a illustrates this.

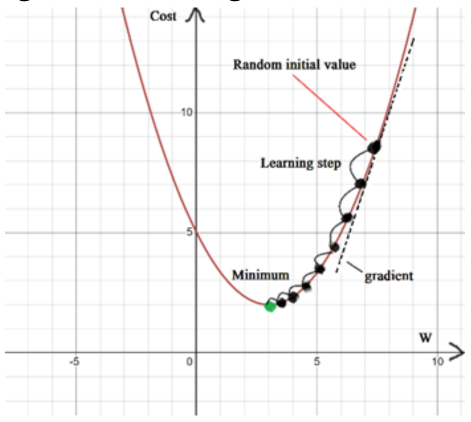


Figure 3a. Illustration of gradient descent.

In multivariable calculus, a gradient vector ∇C containing partial derivatives is used to represent the direction of steepest ascent; the negative of the gradient, $-\nabla C$, gives the downhill direction. So for weight updates, we have:

$$W_{new} = W - \eta \nabla_\theta C(W)$$

where

- η is the learning rate, which can be a configurable constant like 0.01
- W is the current weight matrix, W_{new} is the new weight matrix
- $\nabla_\theta C(W)$ is the partial derivative of the cost function w.r.t weights W

Similarly,

$$B_{new} = B - \eta \nabla_\theta C(B)$$

where

- η is the learning rate
- B is the current Bias vector, B_{new} is the new Bias vector.
- $\nabla_\theta C(B)$ is the partial derivative of the cost function w.r.t biases B

There are basically three types of gradient descent: Batch Gradient Descent (GD), Minibatch Gradient Descent (Minibatch) and Stochastic Gradient Descent (SGD). GD computes accumulative gradients on the whole dataset per iteration and then update the weights and biases; Minibatch divides the dataset into mini-batches, computes accumulative gradients on a mini-batch and then updates; SGD takes one random example each time, computes the gradient and then updates the weights and biases. GD is thorough but is slow to converge and requires large memory, SGD updates frequently and converges fast with small memory fingerprints but could be choppy, Minibatch has vectorization processing capabilities but is rigid in execution order (lacks randomness) when running through the batches. On top of them the AI community has built different optimization extensions, such as AdaGrad, RmsProp, Adam, SGD, etc.

Back Propagation

According to the chain rule of calculus, conceptually, we have

$$\frac{\partial C}{\partial W^{(L)}} = \frac{\partial Z^{(L)}}{\partial W^{(L)}} \frac{\partial A^{(L)}}{\partial Z^{(L)}} \frac{\partial C}{\partial A^{(L)}}$$

Through chaining, a nudge in $W^{(L)}$ could have rippling effects: it first leads to changes to $Z^{(L)}$, further leading to changes in $A^{(L)}$, and finally changing C. Gradient computation takes the route of **Back Propagation**, from C all the way back to the interested weight or bias.

To generalize, to calculate the gradient for weights of layer l, the chain rule works:

$$\frac{\partial C}{\partial W^{(l)}} = \frac{\partial Z^{(l)}}{\partial W^{(l)}} \frac{\partial A^{(l)}}{\partial Z^{(l)}} \frac{\partial C}{\partial A^{(l)}}$$

where $l \in [1, L], L$ is the output layer. (Kostadinov)

CNN

Convolutional neural network (CNN) is a class of NNs commonly applied to analyze visual imagery. It contains four types of layers: an input layer of 2D array of image pixels, one or more convolutional layers, followed by one or more pooling layers, and a final fully connected (FC) layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. A CNN starts to recognize larger elements or shapes of the object until it finally identifies the intended object as the image data progresses through its layers (Ji, et al).

Convolutional Layer

The convolutional layer is the core building block of a CNN. It includes input data, a filter, and a feature map. The input is typically a color image, made up of a matrix of pixels in three dimensions: a height, width, and depth, in RGB format. A feature detector, also known as a kernel/filter, moves across the receptive fields of the image, checking if the feature is present. This process is known as convolution.

The feature detector is a two-dimensional (2-D) array of weights, typically a 3x3 filter. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter and then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output is known as a feature map.

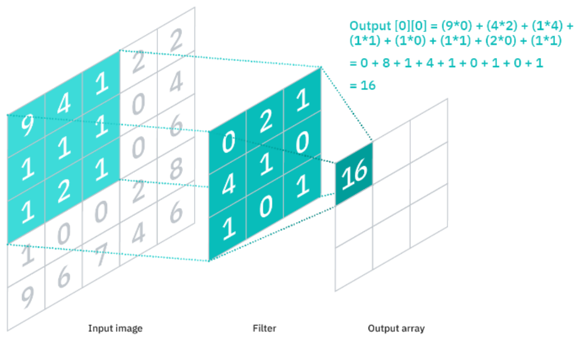

Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
16
Input image
Filter
Output array

Figure 3b. How convolution works (IBM)

After each convolution operation, a ReLU transformation is perfomred to the feature map, introducing nonlinearity to the model.

When another convolution layer follows the initial convolution layer, the structure of the CNN becomes hierarchical as the later layers can see the pixels within the receptive fields of prior layers. For example, the task is to determine if an image contains a bicycle. A CNN treats the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the NN, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.

Pooling Layer

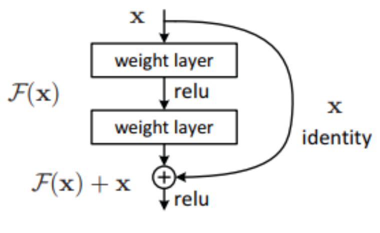
A pooling layer reduces dimensionality. Like the convolutional layer, the pooling operation sweeps a filter across the entire input, except that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- Max pooling: the pixel with the maximum value is chosen
- Average pooling: the average value within the receptive field is used

Though some information is lost in the pooling layer, it helps to reduce complexity and improve efficiency.

ResNet

One issue with CNN is the vanishing gradient problem: after some depth, the performance degrades because when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero. This results in the weights never get updated and therefore, no learning is being performed (Ruiz). To fix this, researchers implemented residual layers:



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4a. Residual Layer (He, et al)

In the above figure, there is a direct connection that skips some layers of the model. This is called ’skip connection’ and is the heart of residual blocks. Without the skip connection, input ‘X gets multiplied by the weights of the layer followed by adding a bias term.

$$H(x) = F(x)$$

Now with the introduction of a new skip connection technique, the output is H(x) is changed to

$$H(x) = F(x) + x$$

This technique solves the vanishing gradient problem in deep CNNs by allowing alternate shortcut paths for the gradient to flow through (Ruiz). A NN containing residual layers is a **ResNet** (He, et al). ResNet50 is a typical ResNet that is 50 layers deep.

Figure 4b illustrates a ResNet50 model containing 6 parts: Input pre-processing, 4 control-flow-group(cfg) blocks, and a fully connected layer. Each cfg block contains a convolution block and an identity block. Both blocks have 3 convolution layers.

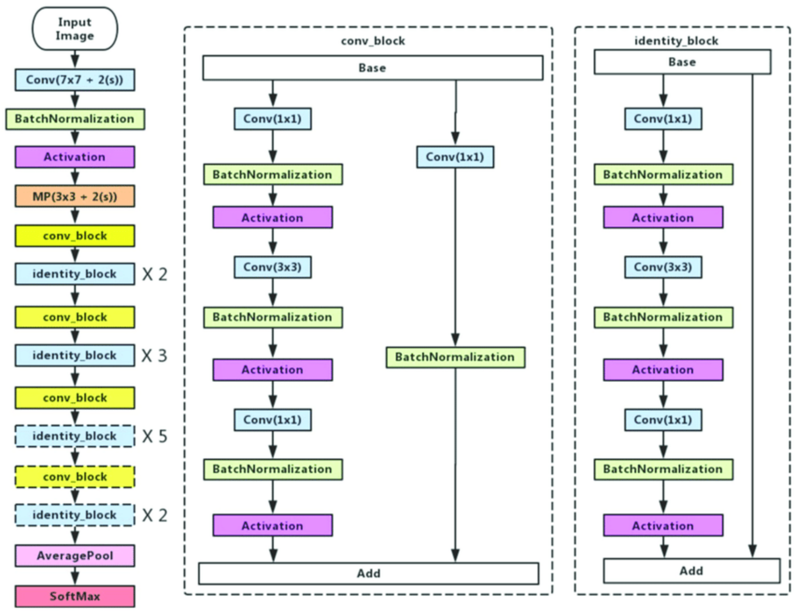


Figure 4b. Architecture of ResNet50 model (Ji, et al)

Vision Transformer (ViT)

ViT is a visual model based on the architecture of a transformer originally designed for text-based tasks. It represents an input image as a series of image patches, like the series of word embeddings used when using transformers to text, and directly predicts class labels for the image (Boesch). ViT-B/32 is a typical ViT that uses 32x32 patches.

ViT uses multi-head self-attention in Computer Vision. The model (Figure 5) splits the images into a series of positional embedding patches, which are processed by the transformer encoder. It does so to understand the local and global features that the image possesses. First, ViT divides the image into N “patches” of such as 32x32, then flattens them and makes a linear projection to convert them into 2D data. This way each patch can be treated as a token, which can be input to the Transformer Encoder which works together with a NN called Multi-Layer Perceptron (MLP) to classify the image.

Training ViT requires a large amount of data. Transformers become more accurate with more data (Paul, et al).

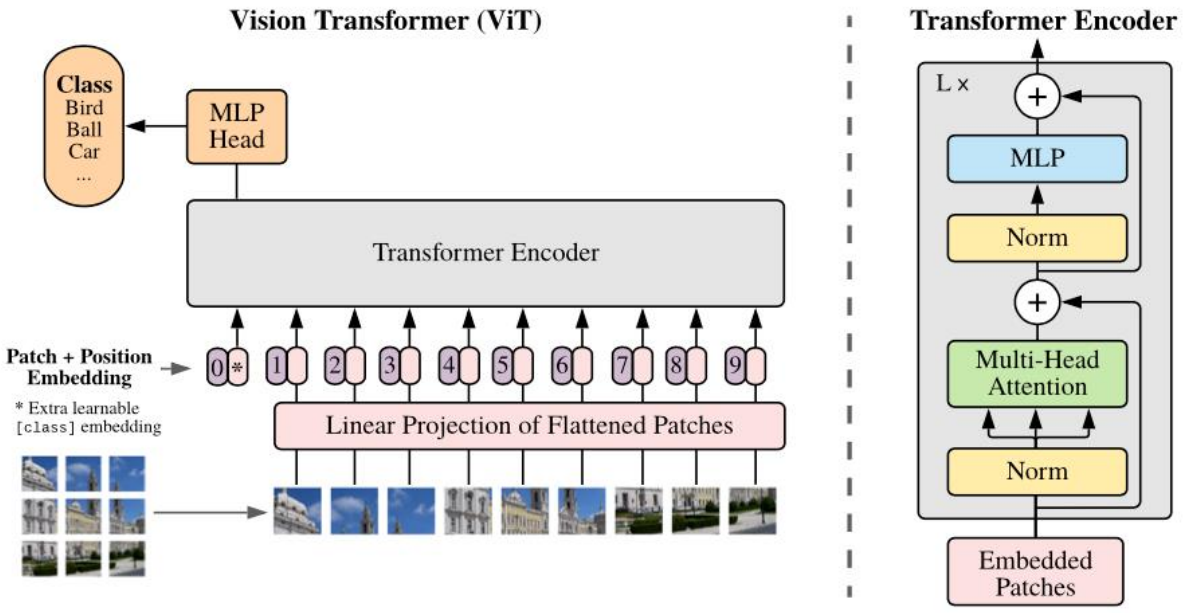


Figure 5. Architecture of Vision Transformer ViT-B/32 (Boesch)

The performance of a vision transformer model depends on decisions such as that of the optimizer, network depth, and dataset-specific hyperparameters.

ROC and AUC

An ROC curve (Receiver Operating Characteristic Curve) is a graph showing the performance of a classification model. This curve plots two parameters:

- True Positive Rate (TPR) - the probability that a True Positive (TP) will test positive. FN=False Negative

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- False Positive Rate (FPR) - the probability that an actual negative will test positive. FP = False Positive; TN = True Negative.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

Below is a typical ROC curve.

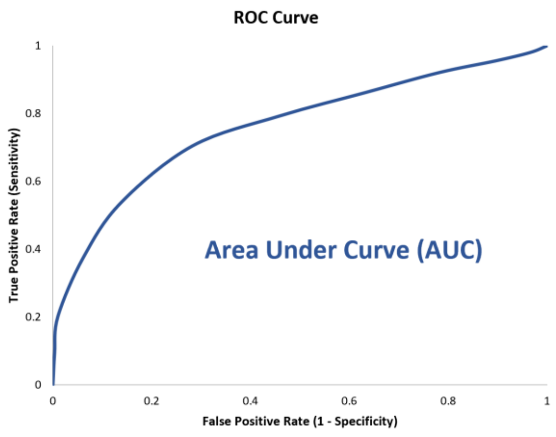


Figure 6. an example Receiver Operating Characteristic Curve (ROC) (Narkhede)

AUC stands for "Area under the ROC Curve." It measures the two-dimensional area underneath the ROC curve (think integral calculus) from (0,0) to (1,1). An excellent model has AUC near 1, meaning it has a good measure of separability. When AUC is 0.5, it means the model has no class separation capacity (Narkhede). AUC values are excellent indicators of NNs' adaptability/robustness.

Figure 7 shows a ROC and its distributions of TN and TP probabilities:

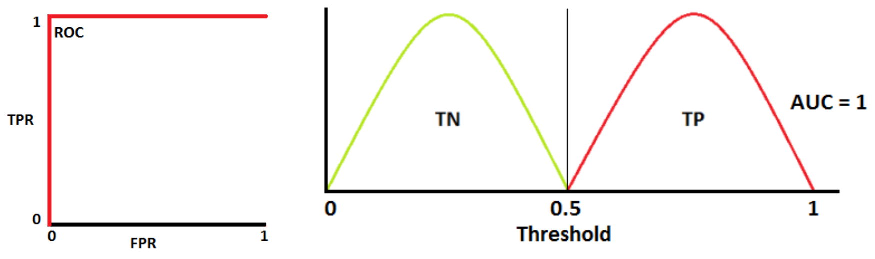


Figure 7. Left: ROC curve with AUC=1; Right: Distribution of Probability. Red: positive class; Green: negative class (Narkhede)

When AUC is approximately 0.5 (see Figure 8), the mode has 0 capacity to distinguish between positive class and negative class.

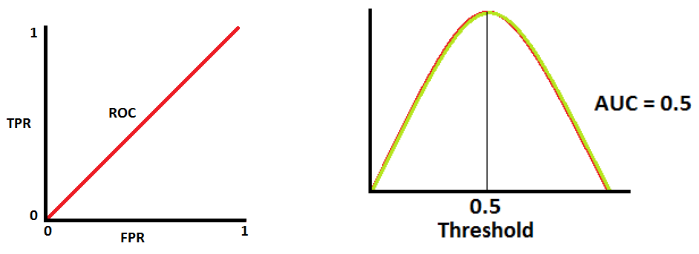


Figure 8. ROC curve with AUC = 0.5 and its distribution of probability (Narkhede).

Usually when a model's AUC on its test dataset is around 90% (see Figure 9), it would be considered as a well-trained model.

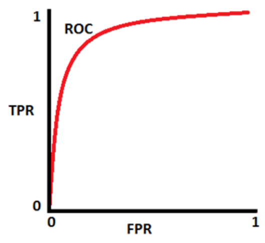


Figure 9. ROC curve with AUC =0.9 (Narkhede)

HYPOTHESIS

ResNet uses a pooling approach that gains speed but incurs information loss; ViT's multi-head attention mechanism keeps global and spatial information. We hypothesize that ViT-B has better adaptability on noisy data while ResNet converges faster.

METHODOLOGY

This project aims to compare the robustness & precision of two representative models: ResNet50 and ViT-B/32, on diabetic retinal imaging. Precision is measured by comparing the diseases predicted by the models and the actual diseases annotated by medical staff. Adaptivity is measured by AUC.

Data

Due to limited access on patient data and ethical concerns, the experiment uses the Kaggle Diabetic Retinopathy Dataset(DRD) at https://www.kaggle.com/c/diabetic-retinopathy-detection/data, a secondary set of high-resolution retina images. A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4, according to the following scale: 0 - No DR; 1 – Mild; 2 – Moderate; 3 – Severe; 4 - Proliferative DR.

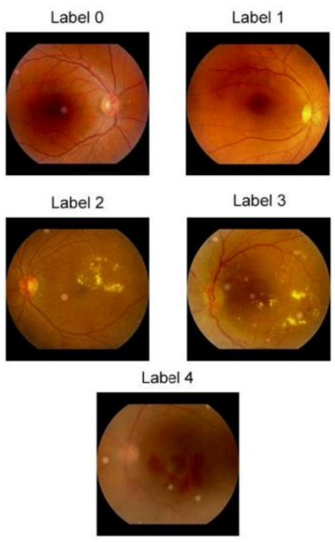


Figure 10. The five DR stages with medical annotative labels. (Sikder)

Development Machine

Apple M1 Macbook Pro, with VS Code; Python3.8

GPU Server:

The GPU server is rented from Tencent Cloud, running Ubuntu18.04 with python 3.8 and tensorflow 2.4.0. It has two nVidia Tesla V100 (16GB each) GPU.

```
/mnt/huangtianyou/DDR$ uname -a
Linux GPU1 5.4.0-1064-azure #67~18.04.1-Ubuntu SMP Wed Nov 10 11:38:21 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux

/mnt/huangtianyou/DDR$ nvidia-smi
Mon Feb 21 18:01:20 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  On   | 00000001:00:00.0 Off |                    0 |
| N/A   25C    P0    25W / 250W |      0MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-PCIE...  On   | 00000002:00:00.0 Off |                    0 |
| N/A   25C    P0    24W / 250W |      0MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 11. GPU server’s configs.

Remote Development

sFTP and scp are used to transfer files and data between local Mac and remote GPU Server.

Visual Studio Code’s remote SSH execution and debugging capabilities are leveraged. Below is an illustration of how it works:

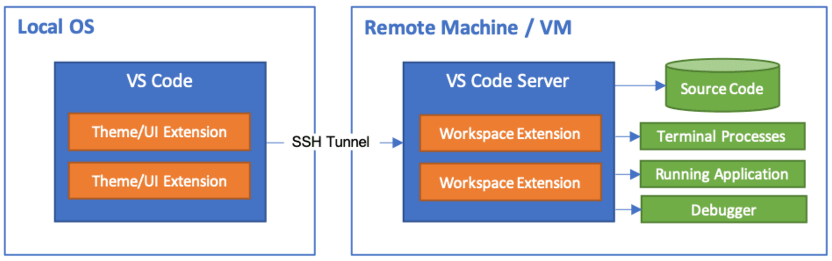


Figure 12. How VS Code Remote works

Experimental Procedure

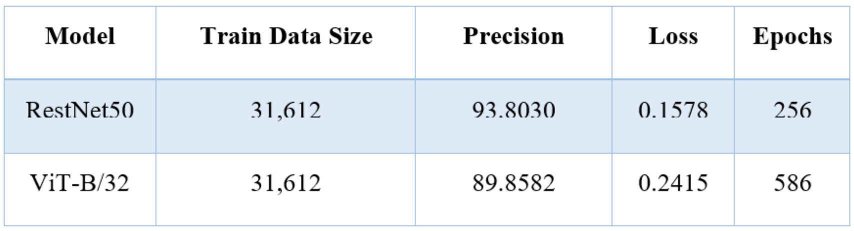
1. Download the Kaggle DRD dataset
2. Load the dataset onto GPU server
3. Generate a txt file that contains the index, file name, and label for the dataset.
4. Separate the dataset into 8(train):1(validate):1(test)
5. Set the Load_Image.py configs
6. Use Python to generate 3 test groups:
    i. Normal test group
    ii. Blurredness test group (using *cv2.blur()* from cv2 library on test dataset to generate various blurred sets with pixel ratio of 1:1x1, 1:3x3, …, 1:30x30, at a step of 3)
    iii. Brightness test group (using *ImageEnhance.Brightness().enhance()* from cv2 library on test dataset to generate various brightened sets: enhancement factor from 0.2 to 2.9, at a step of 0.3; max enhancement factor is 3.0; 1.0 is the same as the original image)
7. Set up the models ResNet-50 and ViT-B/32 with PyTorch
8. Train the models for 1000 epochs
9. Save the trained models as an PyTorch file
10. Use the top ResNet50 model and ViT-B/32 model with the highest AUC values to predict against the test datasets (Normal, Blurredness, Brightness), record the AUC scores into a csv file.
11. Analyze the AUC scores and compare the two models’ robustness.

Model Optimizer

For training of the models, the SGD optimizer is chosen because of its fast convergence and small memory fingerprints (Patrikar).

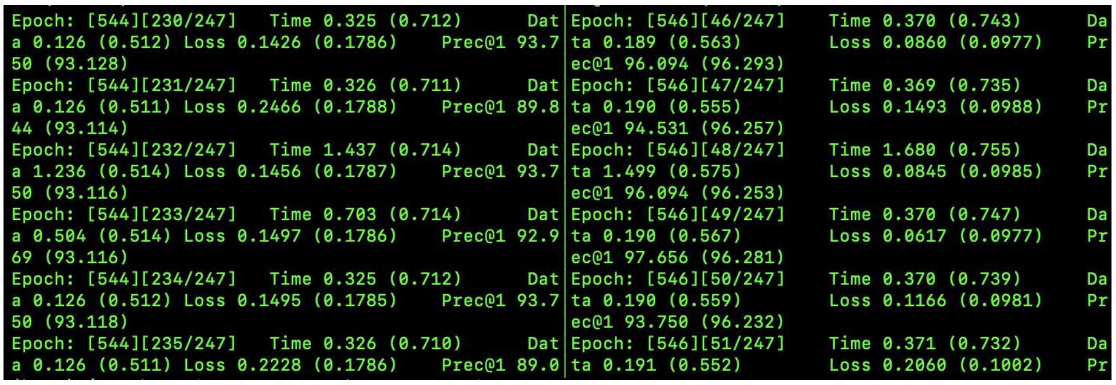
Experiment Results

A. Training of the models



Model	Train Data Size	Precision	Loss	Epochs
RestNet50	31,612	93.8030	0.1578	256
ViT-B/32	31,612	89.8582	0.2415	586

Table 2. Primary data on the training summary.



```
Epoch: [544][230/247]   Time 0.325 (0.712)      Dat|Epoch: [546][46/247]     Time 0.370 (0.743)      Da
a 0.126 (0.512) Loss 0.1426 (0.1786)    Prec@1 93.7|ta 0.189 (0.563)         Loss 0.0860 (0.0977)    Pr
50 (93.128)                                        |ec@1 96.094 (96.293)
Epoch: [544][231/247]   Time 0.326 (0.711)      Dat|Epoch: [546][47/247]     Time 0.369 (0.735)      Da
a 0.126 (0.511) Loss 0.2466 (0.1788)    Prec@1 89.8|ta 0.190 (0.555)         Loss 0.1493 (0.0988)    Pr
44 (93.114)                                        |ec@1 94.531 (96.257)
Epoch: [544][232/247]   Time 1.437 (0.714)      Dat|Epoch: [546][48/247]     Time 1.680 (0.755)      Da
a 1.236 (0.514) Loss 0.1456 (0.1787)    Prec@1 93.7|ta 1.499 (0.575)         Loss 0.0845 (0.0985)    Pr
50 (93.116)                                        |ec@1 96.094 (96.253)
Epoch: [544][233/247]   Time 0.703 (0.714)      Dat|Epoch: [546][49/247]     Time 0.370 (0.747)      Da
a 0.504 (0.514) Loss 0.1497 (0.1786)    Prec@1 92.9|ta 0.190 (0.567)         Loss 0.0617 (0.0977)    Pr
69 (93.116)                                        |ec@1 97.656 (96.281)
Epoch: [544][234/247]   Time 0.325 (0.712)      Dat|Epoch: [546][50/247]     Time 0.370 (0.739)      Da
a 0.126 (0.512) Loss 0.1495 (0.1785)    Prec@1 93.7|ta 0.190 (0.559)         Loss 0.1166 (0.0981)    Pr
50 (93.118)                                        |ec@1 93.750 (96.232)
Epoch: [544][235/247]   Time 0.326 (0.710)      Dat|Epoch: [546][51/247]     Time 0.371 (0.732)      Da
a 0.126 (0.511) Loss 0.2228 (0.1786)    Prec@1 89.0|ta 0.191 (0.552)         Loss 0.2060 (0.1002)    Pr
```

Figure 13. The two models' training speed are comparable (Left: ResNet50; Right: ViT-B/32).

B. Selection of each model's optimal state: the highest AUC scoring model state on the validation (Val) dataset was chosen in each case, whose corresponding AUC values on the Test dataset were subsequently used to calibrate in C.

Val Data Size	**Val ViT AUC**	**Val Data Size**	**Val ResNet50 AUC**
1756	0.8008	1756	0.8730
Test Data Size	**Test ViT AUC**	**Test Data Size**	**Test ResNet50 AUC**
1758	**0.7817**	1758	**0.8565**

Table 3. Optimal models' AUC values on validation data and test data.

C. Measuring the optimal models' AUC against blurred and brightened Test Datasets

Blurriness	ViT AUC	Ratio1(%)*	ResNet50 AUC	Ratio2(%)
1	0.7811	-0.0719	0.8434	-1.5314
3	0.7786	-0.4006	0.8429	-1.5807
6	0.7746	-0.9053	0.8338	-2.6521
9	0.7608	-2.6766	0.7993	-6.6767
12	0.7429	-4.9696	0.7600	-11.2674
15	0.7246	-7.3054	0.7306	-14.6922
18	0.7085	-9.3635	0.7052	-17.6684
21	0.6919	-11.4935	0.6867	-19.8196
24	0.6795	-13.0735	0.6627	-22.6223
27	0.6671	-14.6648	0.6416	-25.0921
30	0.6556	-16.1346	0.6252	-27.0064

Table 4a. Optimal models' AUC values on blurred dataset

Brightness	ViT AUC	Ratio1(%)*	ResNet50 AUC	Ratio2(%)
0.2	0.5988	-23.4023	0.6652	-22.3387
0.5	0.7184	-8.0928	0.7872	-8.0931
0.8	0.7570	-3.1583	0.8000	-6.5909
1.1	0.7777	-0.5071	0.8173	-4.5721
1.4	0.7478	-4.3362	0.8010	-6.4749
1.7	0.7039	-9.9547	0.7631	-10.9004
2.0	0.6791	-13.1236	0.7333	-14.3774
2.3	0.6424	-17.8188	0.7029	-17.9251
2.6	0.5991	-23.3576	0.6788	-20.7418
2.9	0.5641	-27.8349	0.6423	-25.0107

Table 4b. Optimal models' AUC values on brightened dataset

*Ratio1(%) = (ViT AUC – Test ViT AUC)/Test ViT AUC * 100%

*Ratio2(%) = (ResNet50 AUC – Test ResNet50 AUC)/Test ResNet50 AUC * 100%

ANALYSIS & DISCUSSION

A. ResNet50 converges faster

Both models were trained for 1000 epochs against the same training dataset of 31,612 samples, both with SGD as optimizer and CCE as loss option. Figures 14 and 15 show their precision/loss trends.

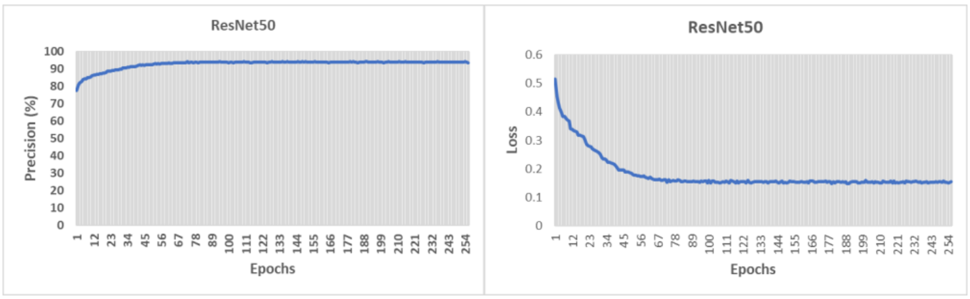


Figure 14. ResNet50 approaches maximum precision and minimum loss after ~80 epochs.

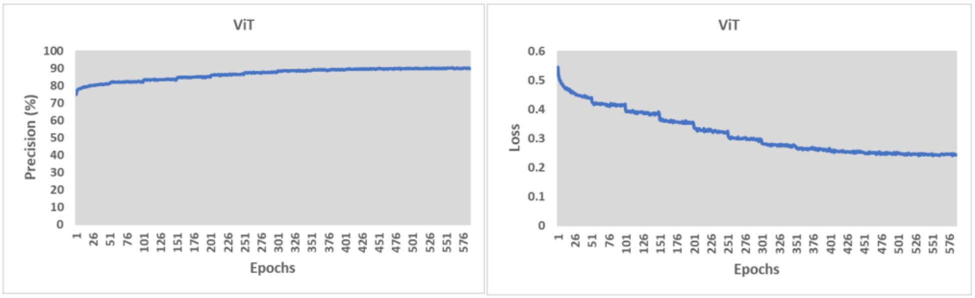
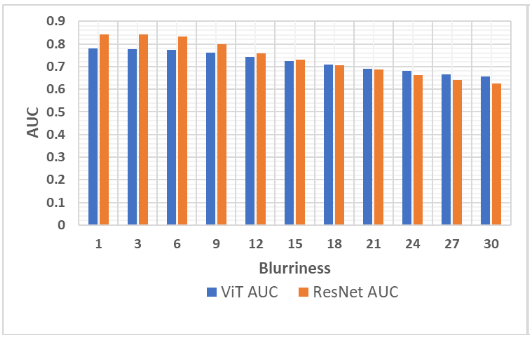


Figure 15. ViT-B/32 approaches maximum precision and minimum loss after ~550 epochs.

After about 80 epochs, ResNet50 stabilizes around 93.8% precision and loss of 0.15. In comparison, it takes ViT-B/32 about 550 epochs to approach ~90.0% precision and loss of 0.24. Given that both models spend comparable time per epoch (Figure 13), the results suggest that ResNet50 converges faster than ViT-B/32, verifying our hypothesis on the converging speeds of the two models.

This is understandable because the ViT-B/32 first cuts the image into many layers (patch embedding) before entering the fully connected layers, thus it takes more iterations to train and converge. It is also notable that ViT's precision is slightly lower, as transformers require real large dataset to be more precise (Raghu, et al).

B. ViT-B/32 is substantially more robust on blurred data.



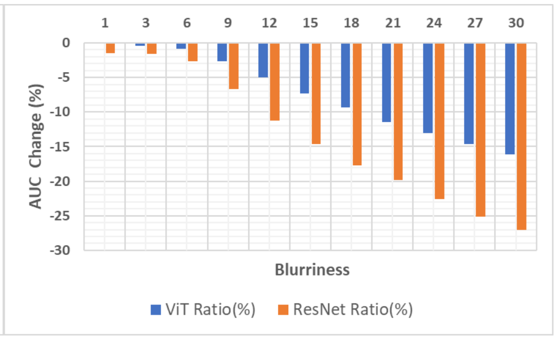


Figure 16a. Blurriness vs AUC

Figure 16b. Blurriness vs AUC change %

Figure 16a shows the comparison of AUC values of the two models at different levels of blurriness (described in step 6 of experimental procedures). Figure 16b shows the AUC change % vs the same models' AUC values on the test dataset. These results suggest that on blurred image, ViT-B/32's (AUC change: -16.1%) is substantially more robust than ResNet50 (AUC change: -27.0%), verifying the hypothesis that ViT is more robust on noisy data.

To rationalize this, I examined the fundamental architecture difference between the two models. ResNet50 uses pixel arrays, whereas ViT-B/32 divides an image into fixed-size patches. ViT-B/32's self-attention mechanism encompasses a much longer range (global image), unlike ResNet kernels, which only encompasses a local 3x3 or 5x5 kernel. Blurring a retinal image is basically "meaning/summing" up the pixels within certain range (weakening distinctive identity) of the data. Thus ResNet50's performance was severely affected since its kernels encompasses a small range of data compared to ViT-B/32 where it encompasses the entire image. Figure 17 shows their differences.

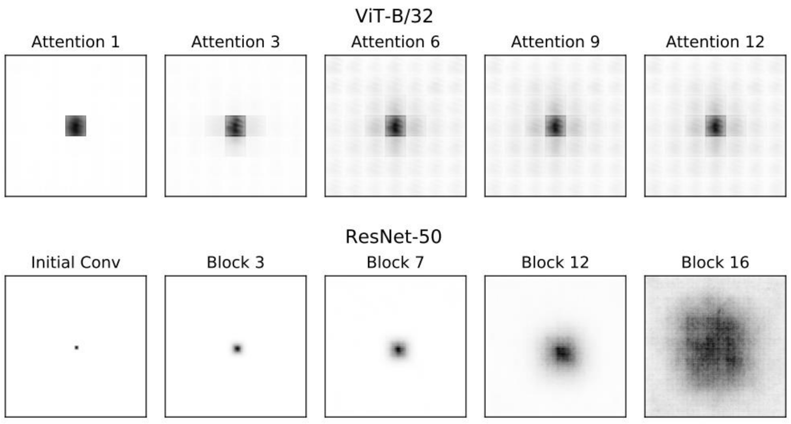
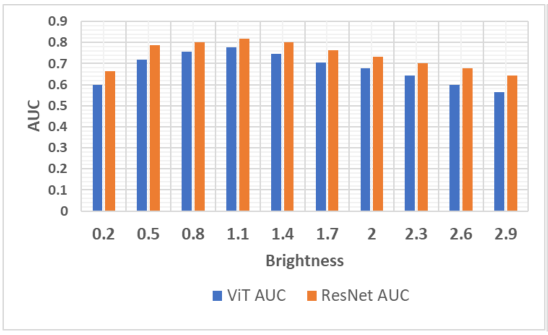


Figure 17. ResNet effective receptive fields are highly local and grow gradually; ViT effective receptive fields shift from local to global (Raghu, et al)

C. ViT-B/32 performs substantially better when the brightness factor is around 1.0 (original image).



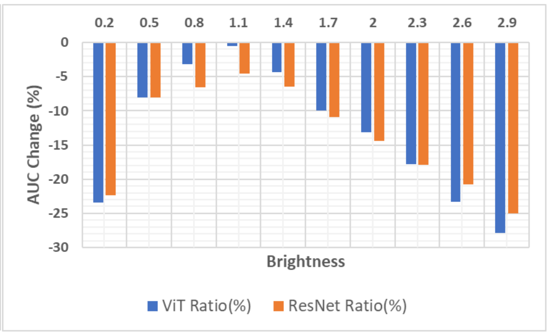


Figure 18a. Brightness vs AUC

Figure 18b. Brightness vs AUC change %

Comparison of the two models' AUC vs brightness (described in step 6 of experimental procedures) is shown in figure 18a. Figure 18b shows the AUC change % vs the same models' AUC values on the test dataset. The comparison suggests that ViT-B/32 performs substantially better when the brightness factor is within 0.8-1.4, which verifies our hypothesis that ViT has higher adaptability on noisy data. When the images' brightness is less than 0.5 or greater than 1.7, both models show comparable robustness, which is a bit different than the hypothesis.

These observations could be explained by ViT's uniform representations across all layers. Its self-attention mechanism enables early aggregation of global information and residual connections, which strongly propagate features from lower to higher layers. Thus, ViT preserves input spatial information of the colored pixels better (shown in Figure 19) and exhibits more robustness when an image's brightness is within a normal range (0.8-1.4) to the original image (1.0). When an image's brightness is < 0.5 or >1.7, the colored pixels are either so overshadowed or under shadowed by neighboring pixels that ViT's robustness degrades to the comparable level of ResNet50.

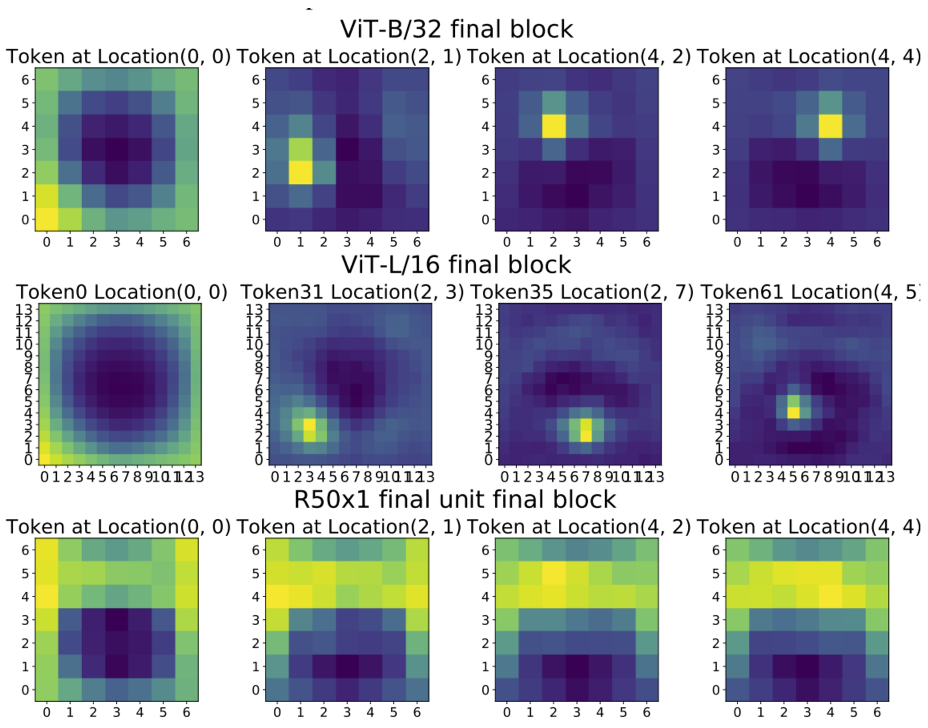


Figure 19. Higher layers of ViT models maintain spatial location information more faithfully than ResNets. (Raghu, et al)

LIMITATIONS

One limitation of the experiment is the size of the training retinal dataset. If a larger dataset of 200 K samples is used, the models' precisions can be much improved. It was hard to find retinal imaging datasets with the right labelling.

A second limitation is the memory size (16GBx2) of the GPU server. To reduce the memory pressure on the servers, I had to resize the raw images from 5000x6000 pixels to 1000x1000, resulting in loss of information and quality. Access to more powerful GPU servers can help achieve higher precisions and better results overall.

A third limitation is the optimizer used. SGD is a good at fast convergence on large data sets and memory efficiency, but its convergence could be choppy because it employs a fixed learning rate. The Adam optimizer uses adaptive learning rate, should help get a smoother convergence and thus is the logical next step for the experiment.

A fourth limitation is lack of tools in the AI industry to examine exactly what is happening during the training of an AI model. Though it is possible to reproduce by hand how a simple NN works, it is literally impossible to examine a specific step when training a model without proper tools. Thus, conceptual understanding carries more weight on such occasions.

CONCLUSION

Through comparing the representative models ViT-B/32 and ResNet50 using the Kaggle DRD dataset, we have been able to verify the hypothesis that ResNet converges much faster than ViT and that ViT is generally substantially more robust on the noisy dataset containing blurred or brightened images. We have also found that ViT and ResNet show comparable robustness on brightened images whose brightness factors are <0.5 or >1.7. Further analyses suggest that such differences and parity are due to the architectural characteristics of the models. Understanding these models' differences in robustness can be highly meaningful in the field of AI diagnosis of diabetes based on retinal imaging.

REFERENCES

WHO (World Health Organization). (n.d.). “Diabetes”. 2 May, 2022, https://www.who.int/health-topics/diabetes#tab=tab_1

Belde, sunil. “Noise Removal in Images Using Deep Learning Models.” 25 Apr. 2021, https://medium.com/analytics-vidhya/noise-removal-in-images-using-deep-learning-models-3972544372d2

Mantelakis, A., Assael, Y., Sorooshian, P., & Khajuria, A. “Machine learning demonstrates high accuracy for disease diagnosis and prognosis in plastic surgery”. 24 June, 2021. Plastic and reconstructive surgery. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8225366/

Middleton, Michael. “Deep Learning vs. Machine Learning - What's the Difference?” 9 Aug. 2022, Flatiron School, https://flatironschool.com/blog/deep-learning-vs-machine-learning/

Ji, Qinge, et al. “Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images”. Feb 2019, Algorithms 12(3):51, https://www.researchgate.net/publication/331364877

IBM Cloud Education. (n.d.). “What are neural networks?” IBM. Retrieved May 2, 2022, https://www.ibm.com/cloud/learn/neural-networks

Ruiz, Pablo, “Understanding and visualizing ResNets”. Retrieved May 2, 2022, https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

Shakhadri, S. A. G, “What is resnet: Build resnet from scratch with python”. 8 June, 2021, Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/build-resnet-from-scratch-with-python/

Boesch, G. “Vision transformers (VIT) in Image Recognition” 2022 guide. viso.ai. Retrieved May 2, 2022, from https://viso.ai/deep-learning/vision-transformer-vit/

Kostadinov, Simeon. “Understanding Backpropagation Algorithm.” 12 Aug, 2019, https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

Sikder, Niloy. “Determing the Severity of Diabetic Retinopathy through Analyzing Retinal Images Using Machine Learning algorithm”. December 2020, DOI: 10.13140/RG.2.2.23178.39364

Narkhede, Sarang. “Understanding AUC - roc curve”. 27 June, 2018, https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

He, Kaiming, et al. “Deep Residual Learning for Image Recognition”. 10 Dec, 2015, arXiv:1512.03385. https://arxiv.org/abs/1512.03385

Patrikar, Sushant. “Batch, Mini Batch & Stochastic Gradient Descent”. 1 Oct, 2019. *https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a*

Paul, Sayak, Chen, Pin-Yu. “Vision Transformers are Robust Learners”. https://www.aaai.org/AAAI22Papers/AAAI-2044.PaulS.pdf

Raghu, Maithra, et al. “Do Vision Transformers See Like Convolutional Neural Networks?” 3 Mar 2022, arXiv:2108.08810v2, https://arxiv.org/pdf/2108.08810.pdf

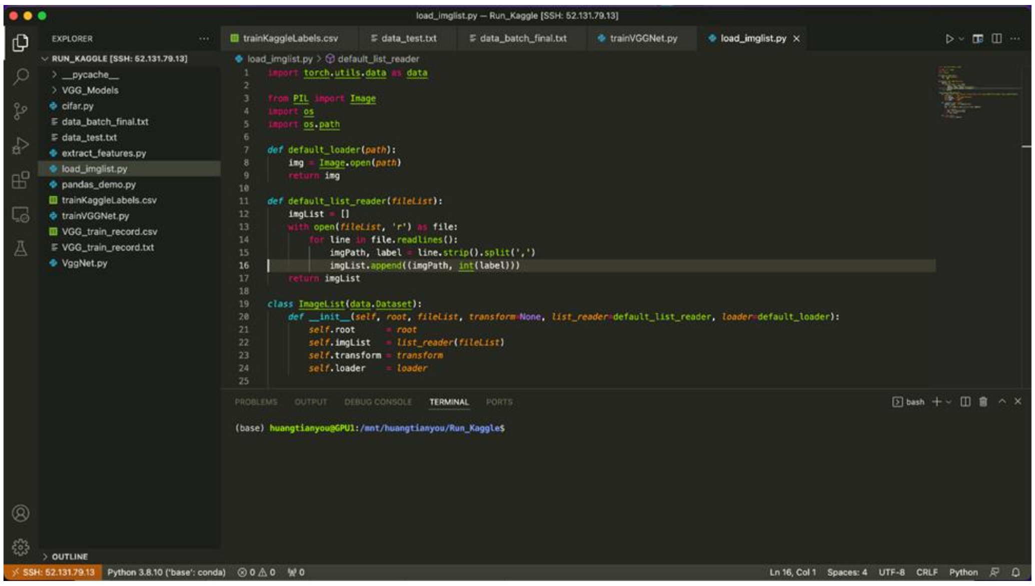
APPENDIX:

Source code

Stored at: https://github.com/JamesHuang2004

Local Development and Remote Execution

The screenshot below shows how I was running vscode on the local macbook pro and running the code on the remote GPU server. Note the left-bottom-corner shows the SSH remote connection.



```python
import torch.utils.data as data

from PIL import Image
import os
import os.path

def default_loader(path):
    img = Image.open(path)
    return img

def default_list_reader(fileList):
    imgList = []
    with open(fileList, 'r') as file:
        for line in file.readlines():
            imgPath, label = line.strip().split(',')
            imgList.append((imgPath, int(label)))
    return imgList

class ImageList(data.Dataset):
    def __init__(self, root, fileList, transform=None, list_reader=default_list_reader, loader=default_loader):
        self.root      = root
        self.imgList   = list_reader(fileList)
        self.transform = transform
        self.loader    = loader

```

```
(base) huangtianyou@GPU1:/mnt/huangtianyou/Run_Kaggle$
```

Figure Appendix-A. Local development and remote execution in action.