# A Comparative Study of TensorFlow and PyTorch in Diabetic Retinopathy Detection

Nikhitha Chandana Chinthapalli,
Department of Computer Science,
University of Houston, Houston, TX, USA.
nchintha@cougarnet.uh

Sai Sri Charishma Vaddi,
Department of Computer Science,
University of Houston, Houston, TX, USA.
svaddi2@cougarnet.uh.edu

***Abstract* – The primary objective of this study is to compare the performance of TensorFlow and PyTorch frameworks in detecting Diabetic Retinopathy from retina images. Build a Convolutional Neural Network Model to detect the severity level of Diabetic Retinopathy disease in the retina images using two different deep learning frameworks, TensorFlow and PyTorch. Deploy both the trained models in AWS EC2. Conduct performance analysis between two versions.**

***Index Terms*** – TensorFlow, PyTorch, ResNet50

## I. INTRODUCTION

Diabetic retinopathy (DR) is a leading cause of vision loss globally, primarily affecting individuals with diabetes. Early detection and diagnosis of DR can significantly improve the prognosis and prevent severe vision impairment or blindness. Deep learning techniques, particularly convolutional neural networks (CNNs), have shown promising results in automated DR detection from retinal images.

Deep learning frameworks like TensorFlow and PyTorch have become popular choices among researchers and developers in various machine learning and artificial intelligence tasks. Both frameworks offer unique features, ease of use, and performance capabilities. The choice between these two frameworks often depends on the specific problem, the nature of the dataset, and the researcher's preferences.

## II. DATA

### *A. Gathering Data*

The Diabetic Retinopathy Detection dataset is a comprehensive dataset available on Kaggle (https://www.kaggle.com/c/diabeticretinopathy-detection), which contains a large number of high resolution retina images taken under various imaging conditions. This dataset was originally collected to support the development of machine learning models. For detecting diabetic retinopathy, a severe eye disease that can lead to blindness if not detected and treated early.

Diabetic retinopathy affects blood vessels in the retina, causing them to leak blood or fluid, which may eventually lead to vision loss. Early detection is crucial to prevent irreversible damage and to initiate proper treatment. As part of the dataset, retina images are labeled with one of five severity levels: 0 (No DR), 1 (Mild), 2 (Moderate), 3 (Severe), and 4 (Proliferative DR).

### *B. Data visualization*

Initial data visualization is important for understanding the dataset's structure and gaining insights into the distribution of various classes. In the case of the Diabetic Retinopathy Detection dataset, we can visualize the distribution of different severity levels (0-4) and display sample images from each class.

Graph for distribution of severity levels in the dataset:

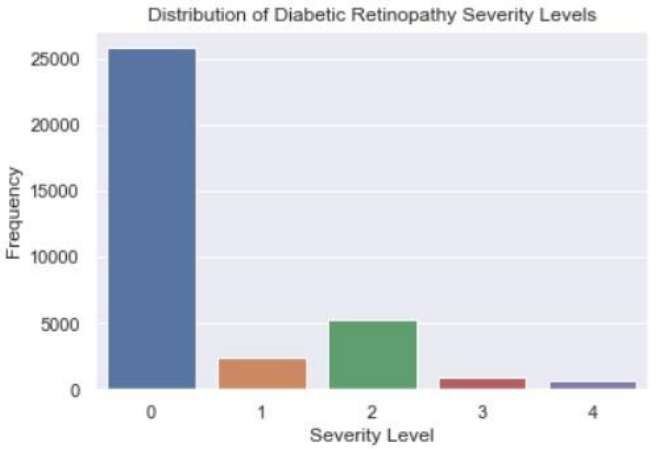


Fig. 1. Visualization of the severity levels of the dataset.

The above visualization can help you understand the variety of images in the dataset and observe any visible patterns or features associated with different severity levels of diabetic retinopathy.

From the above visualization it is evident that the dataset contains more images that belong to ‘No DR’ category than any other category and ‘Proliferative DR’ category has least number of images.

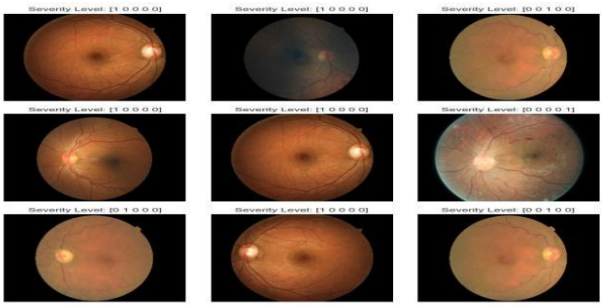

Fig. 2 . Visualization of the images in the dataset.

*C . Data Preprocessing*

Data cleaning is a critical step before performing any analysis on the data. several preprocessing steps are applied to prepare the images and labels for training and validation. The following steps are followed for cleaning the data:

*1) Load and resize images*

```
fixed_size = (300, 300)
resized_image = cv2.resize(image, fixed_size)
```

Fig.3 . Resize Images

*2) Data augmentation*

Data augmentation is performed to increase the size and diversity of the dataset, improving the model's generalization capabilities. In the TensorFlow code, the dataset is augmented by duplicating the images and labels, effectively doubling their size.

```
X = x + x
Y = y + y
```

Fig.4. Data Augmentation

*3) One-hot encoding*

For the TensorFlow code, the labels are one-hot encoded to represent them as categorical variables [6]. This transformation is necessary for using the categorical cross-entropy loss function during model training.

```
one_hot_labels = pd.Series(dataset['level'])
one_hot_labels = pd.get_dummies(one_hot_labels, sparse = True)
one_hot_labels = np.asarray(one_hot_labels)
```

Fig. 5. One-hot encoding

*4) Train-validation split*

The dataset is split into training and validation sets using an 80-20 ratio. This ensures that a separate set of images is used for evaluating the model's performance during training and for comparing the two frameworks.

```
x_train, x_valid, y_train, y_valid = train_test_split(X, Y, test_size = 0.2, train
```

Fig.6. Split train and test data

*5) Data normalization and conversion:*

For the PyTorch code, the images are normalized using predefined mean and standard deviation values for the three RGB channels. This normalization is performed using torchvision's transforms - Normalize () function. Additionally, images are converted to PyTorch tensors using transforms ToTensor().

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Fig.7. Data normalization

These preprocessing steps are essential for ensuring that the data is in a suitable format for training, allowing the models to learn effectively from the available information and improving their ability to detect diabetic retinopathy in new, unseen images.

## III. INCORPORATED PACKAGES

*A. Pandas*

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. It has functions for analyzing, cleaning, exploring and manipulating data.

*B. NumPy*

NumPy is a Python library used for working with arrays. NumPy stands for Numerical Python It also has functions for working in domain of linear algebra, fourier transform, and matrices. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This is the main reason why NumPy is faster than lists. Also, it is optimized to work with latest CPU architectures.

*C. Sklearn*

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

### D. TensorFlow

TensorFlow, an interface for expressing machine learning algorithms, is utilized for implementing ML systems into fabrication over a bunch of areas of computer science, including sentiment analysis, voice recognition, geographic information extraction, computer vision, text summarization, information retrieval, computational drug discovery and flaw detection to pursue research. In the proposed model, the whole Sequential CNN architecture (consists of several layers) uses TensorFlow at backend. It is also used to reshape the data (image) in the data processing.

### E. Keras

Keras gives fundamental reflections and building units for creation and transportation of ML arrangements with high iteration velocity. It takes full advantage of the scalability and cross-platform capabilities of TensorFlow. The core data structures of Keras are layers and models. All the layers used in the CNN model are implemented using Keras [5]. Along with the conversion of the class vector to the binary class matrix in data processing, it helps to compile the overall model.

### F. Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing features to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.

### G. PyTorch

PyTorch is an open-source machine learning framework that allows developers and researchers to build and train various artificial intelligence models, including neural networks. It was developed primarily by Facebook's artificial intelligence research team and is widely used in academia and industry.

PyTorch is built on top of the Torch library, which is a scientific computing framework written in Lua programming language[3]. However, PyTorch is designed to be more user-friendly and flexible than Torch, with a focus on Python programming language. It provides a wide range of features such as automatic differentiation, GPU acceleration, distributed training, and various utilities for building and training machine learning models.

In this study, we use the ResNet50 architecture as the backbone for our deep learning model in both TensorFlow and PyTorch. ResNet50 is a popular deep learning model known for its residual connections, which help in alleviating the vanishing gradient problem and allow for deeper networks. The model has been pre-trained on the ImageNet dataset to extract useful features from the retinal images.

## IV. PROPOSED METHOD

**Resent 50 Architecture**

ResNet-50 is a convolutional neural network architecture that was introduced by Microsoft Research in 2015 as a part of their Deep Residual Learning for Image Recognition paper [4]. It is one of the most used pre-trained models for various computer vision tasks, such as object detection and image classification.

The architecture of ResNet-50 is based on residual learning, which means that instead of learning the actual mapping between the input and output, it learns the difference between the input and output. This is achieved by adding skip connections or shortcuts to the network, which allow the information to bypass one or more layers and flow directly to the next layer.

The ResNet-50 architecture has a total of 50 layers, hence the name. The use of skip connections helps to solve the vanishing gradient problem, which is a common issue in deep neural networks. By allowing information to flow directly through the network, ResNet-50 can learn more complex features and achieve higher accuracy on image recognition tasks [9].

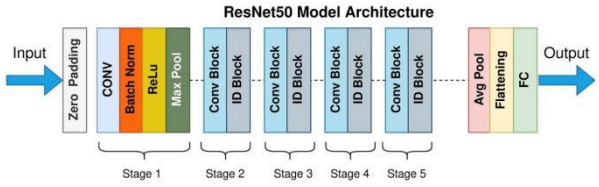


Fig. 8. Visualization of the Resnet50 architecture.

The following steps were performed to adapt the ResNet50 architecture for our Diabetic Retinopathy detection task:

### A. Preprocessing and Data Augmentation:

The retinal images were resized to a fixed size of 300x300 pixels. Data augmentation was performed by duplicating the dataset, which helps in increasing the size of the dataset and reducing overfitting. For PyTorch, additional normalization was applied using the mean and standard deviation values of the ImageNet dataset.

### B. Customization of the Model:

In both TensorFlow and PyTorch, the final fully connected layer of the pre-trained ResNet50 model was replaced with

custom layers. These layers consist of several dense layers with ReLU activation functions followed by a final dense layer with a SoftMax activation function to output probabilities for the 5 severity levels of DR.

### *C. Model Training:*

The models were trained using a batch size of 32 and a maximum of 15 epochs. Early stopping was employed with a patience of 15 epochs to prevent overfitting and save the best model based on the validation loss. The Adam optimizer was used for optimization, and the categorical cross-entropy loss function was employed as the objective function.

### *D. Model Evaluation:*

The models were evaluated on a separate validation set to measure their performance. The primary evaluation metric was the accuracy of the predictions, indicating the proportion of correct classifications. Additionally, training time was measured for both TensorFlow and PyTorch models, providing insights into the efficiency of each framework during the training process.

## V. MODEL ARCHITECTURE, TRAINING AND EVALUATION

### *A. Building model using TensorFlow*

#### *1) Model architecture for TensorFlow*

In the TensorFlow model, the pre-trained ResNet50 architecture was used as the backbone. The final fully connected layer was replaced with custom dense layers with ReLU activation functions, followed by a final dense layer with a SoftMax activation function to output probabilities for the 5 DR severity levels.

#### *2) Model training for TensorFlow*

The TensorFlow model was trained using a batch size of 32 and a maximum of 15 epochs. The Adam optimizer was employed for optimization, and the categorical cross-entropy loss function was used as the objective function.

#### *3) Model Evaluation:*

The TensorFlow model was evaluated on the validation set after training. The performance was measured in terms of validation loss and validation accuracy. The results provided insights into the model's ability to generalize unseen data.

### *B. Building model using PyTorch*

#### *1) Model architecture for PyTorch*

In the PyTorch model, the pre-trained ResNet50 architecture was used as the backbone. The final fully connected layer was replaced with custom dense layers with ReLU activation functions, followed by a final dense layer with a softmax activation function to output probabilities for the 5 DR severity levels.

#### *2) Model training for TensorFlow*

The PyTorch model was trained using a batch size of 32 and a maximum of 15 epochs. The Adam optimizer was employed for optimization, and the categorical cross-entropy loss function was used as the objective function.

#### *3) Model Evaluation:*

The PyTorch model was evaluated on the validation set after training. The performance was measured using validation loss and validation accuracy. The results helped to understand the model's generalization capability on unseen data.

### *C. Storing model object*

In this study, we used the Keras library to train our model, so we exported the model in the h5 format using the model.save() method. This saved the trained weights and architecture of the model to a single file that could be loaded into a Flask application.

## VI. CREATING A WEBSITE

### *A. Creating a Website Template*

Now we need to create a website template with a form consisting of 1 input field to upload retina image and a predict button. It should be named home.html. On clicking the predict button, the result is displayed which is handled by result.html file. Both files need to be placed in the templates folder. All the other files like images, CSS files, JS files should be separately placed in a folder named static.

### *B. Create a Flask Backend API*

Now we need to create a python file (app.py) for collecting the request and to predict the severity of Diabetic Retinopathy. Create a new Flask application (app.py) and define the necessary routes that your website will use to interact with the user. In flask application define a home route with @app.route('/') and create a function to render a simple HTML page to display on the home page. Load the machine learning model using the appropriate library, such as pickle for a serialized Python object, or tensorflow for a saved model. This can be done either in a separate script or inside the Flask app. Define a new route to handle model predictions, with @app.route('/predict', methods=['POST']). This route should accept input data, preprocess it if necessary, and pass it to the model for prediction. Finally, return the prediction as a response. Create an HTML form that allows users to input data to make a prediction. This form should submit data to the prediction route using the POST method. Finally, run the Flask app with app.run() to start the server and serve the website [1].

## VII. DEPLOYMENT TO AWS CLOUD

### *A. Create an Ec2 instance*

#### *1) Choose an Amazon Machine Image (AMI)*

When creating an EC2 instance, you can select an Amazon Machine Image (AMI) to use as a template for your instance. The AMI includes an operating system and any additional software required for your application.

In the "Choose an Amazon Machine Image (AMI)" step, you can either choose an AMI from the "AWS Marketplace", "My AMIs", or "Public images" [1]. The available AMIs are sorted by different categories, such as "Amazon Linux 2", "Ubuntu", "Windows", and more. You can select an AMI based on your operating system and application requirements.

We have selected Ubuntu 20.04 as per our requirement.

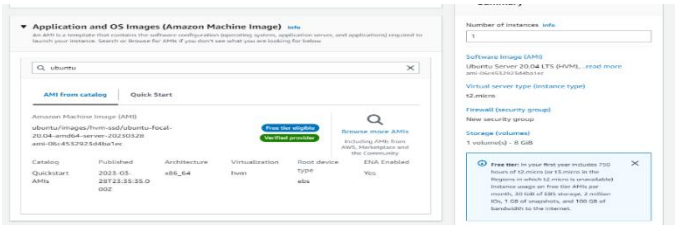


Fig.9. Select Amazon Machine Image

2) *Select Instance type*

Amazon EC2 offers a variety of instance types optimized for different workloads, such as computer, memory, storage, GPU, and networking. Each instance type has its own hardware specifications, including CPU, memory, storage, and network performance. The instance type you choose should match the requirements of your application, such as CPU and memory usage, I/O operations, and network bandwidth. You can use the Amazon CloudWatch service to monitor and optimize your instance performance.

Here we have selected t2.medium. The t2.medium instance type has 2 vCPUs, which are burstable up to 3.3 GHz using the CPU Credits system. The t2.medium instance type has 4 GiB of memory, which is suitable for moderate workloads that require a balance of computer and memory resources. The t2.medium instance type supports Amazon Elastic Block Store (EBS) volumes for persistent storage, and can also use instance store volumes for temporary storage.

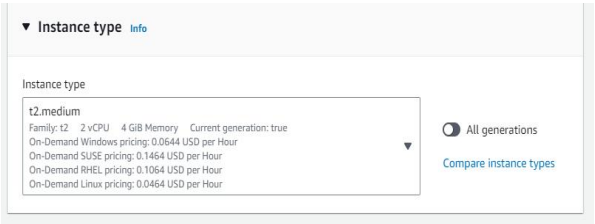


Fig. 10. Select Instance Type t2.medium

3) *Edit the security group.*

After creating the EC2 instance, we edited the security group to allow incoming traffic to the web application. We added a new inbound rule that allowed traffic on port 80 (HTTP) or port 443 (HTTPS) to reach the instance. This ensured that users could access the web application over the internet.

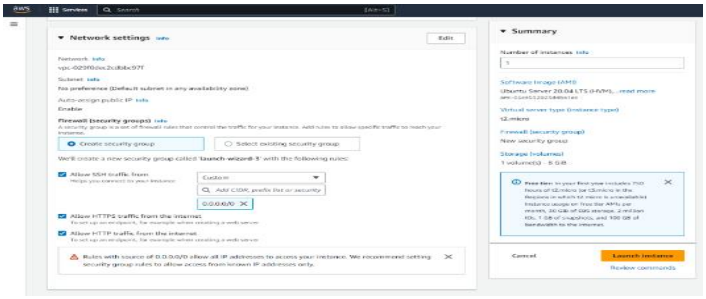


Fig. 11. Security group

4) Configure Storage

When creating an Amazon EC2 instance, the instance is launched with a single Elastic Block Store (EBS) root volume that serves as the main storage device for the instance. The size of the root volume is 8 GiB.

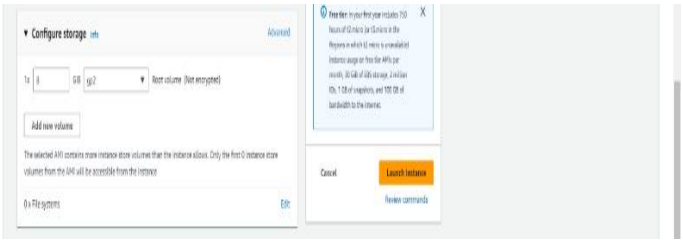


Fig.12. Configure storage

5) Download keygen (pem file)

To authenticate when logging in to the EC2 instance, we downloaded a key pair (pem file). We created a new key pair in the EC2 console and downloaded the private key file (.pem) to our local machine. This key pair allowed us to securely log in to the EC2 instance using SSH.

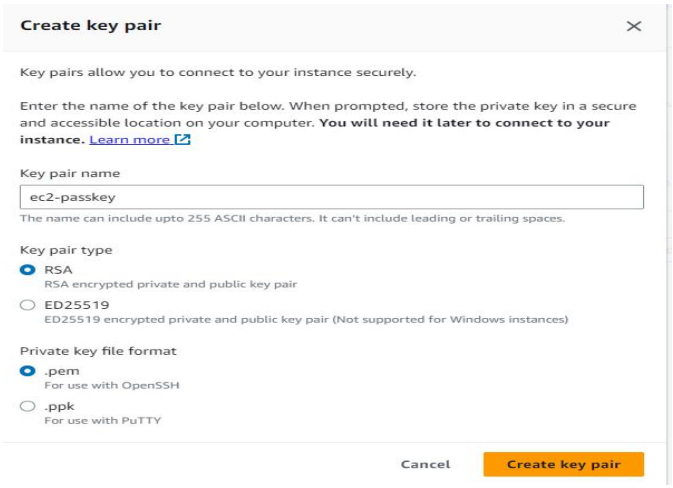


Fig. 13.Create Key - value pair

6) Create Instance

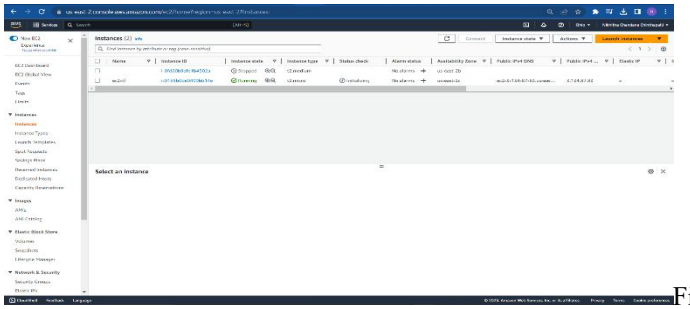

Fig. 14. Instances running

## B. *Connect to EC2 Environment*

### 1) *Download and install Putty and WinSCP and connect to ec2 environment using private key*

To connect to the EC2 instance from our local machine, we downloaded and installed two tools: Putty and WinSCP. Putty is a client for SSH that allows us to connect to the EC2 instance, and WinSCP is a graphical SFTP client that allows us to transfer files between our local machine and the EC2 instance. Use security key ec2-passkey (downloaded while creating ec2 instance) to connect to ec2 environment.

### 2) *Upload Flask website to EC2 using WinSCP*

After installing WinSCP, we used it to upload the Flask web application files to the EC2 instance. We connected to the instance using the key pair and the public DNS address of the instance. Once connected, we transferred the files to the appropriate directory on the instance.

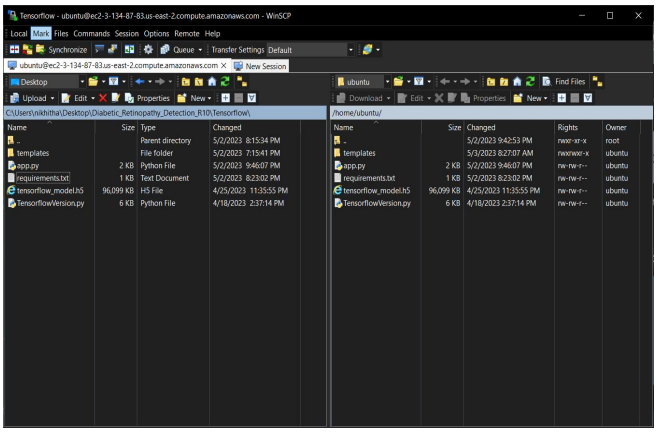

Fig.15. Migrate files from local to ec2 environment

### 3) *Install packages on EC2 using Putty*

Finally, we connected to the EC2 instance using Putty and installed any required packages for the Flask web application. The required packages are listed in the requirements.txt file. All the packages are installed at once using the command Pip3 install -r requirements.txt. command**.**

## C. *Run flask application on ec2*

Run the Flask application using the command "python app.py". This assumes that the name of your Flask application file is "app.py".

Once the application is running, we open a web browser and navigate to the URL shown in the output, such as http://172.31.17.107:8080, to view your Flask application in action. Similarly, we can deploy flask files for both PyTorch version and TensorFlow version applications.

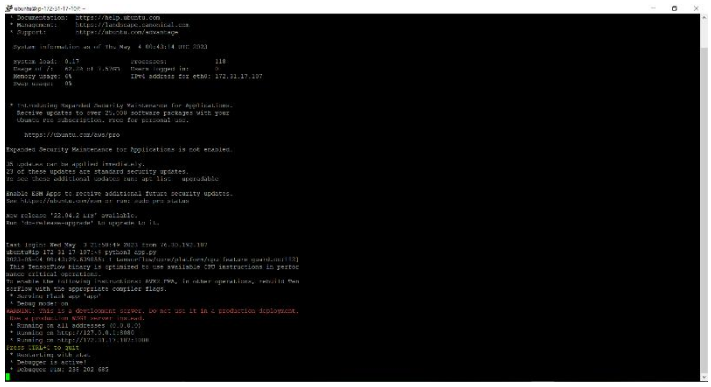

Fig.16. Run app.py flask file

## D. *Test the Flask application*

On navigating to the public URL, http://ec2-3-138-198-118.us-east-2.compute.amazonaws.com:8080/ , we can access the home page of the application. We can upload the retina image and click the predict button to predict the severity of Diabetic Retinopathy.

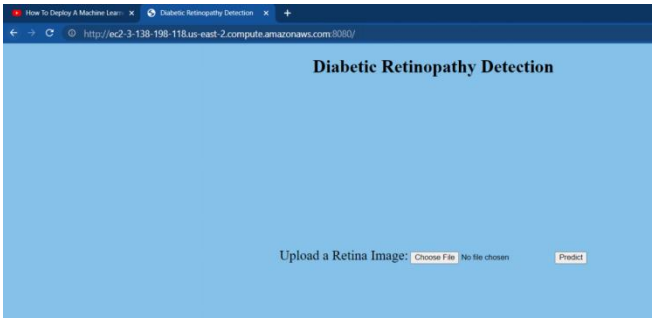


Fig.17. Test application deployed on aws ec2 using public url

On clicking predict button, it navigates to the result page which displays the Severity level.

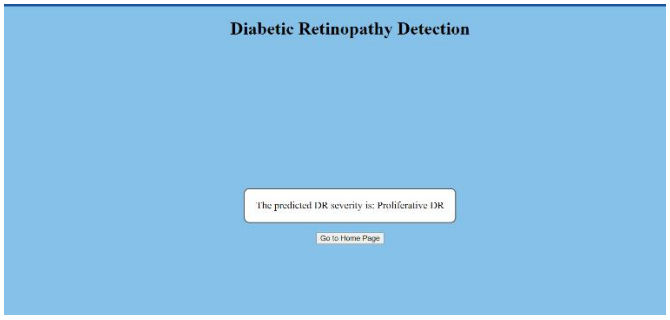


Fig.18. Result Page

Similarly, we can deploy both versions on applications and access from anywhere using public URL.

## VIII. RESULTS AND ANALYSIS

### A. *Model Comparison between TensorFlow and PyTorch*

*1) Accuracy comparison:*

The validation accuracies of both TensorFlow and PyTorch models can be compared to determine which model performed better in terms of correctly classifying the DR severity levels. A higher accuracy indicates better performance.

Here are the screenshots of accuracy for both models:

*TensorFlow:*

```
Final validation accuracy for TensorFlow: 0.7500
```

Fig. 19. Accuracy of TensorFlow model

This means that accuracy percentage is 75.

*PyTorch:*

```
Final validation accuracy for PyTorch: 0.5000
```

Fig. 20. Accuracy of pyTorch model

This means that the accuracy percentage is 50.

From the above comparison, TensorFlow shows better validation accuracy.

*2) Training time comparison:*

The time taken to train the TensorFlow and PyTorch models can be compared to evaluate their efficiency. Faster training times may be advantageous in certain scenarios, especially when working with large datasets or under time constraints.

Here are the screenshots for training time of both the models:

*TensorFlow:*

```
Training time: 38.11 seconds
```

Fig. 21. Training Time of TensorFlow model

*PyTorch:*

```
Training time: 113.67 seconds
```

Fig. 22. Training Time of PyTorch model

Clearly TensorFlow shows lesser time compared to PyTorch.

*3) Training and validation accuracy:*

The training and validation accuracy graph can help us in the judgement of better model from TensorFlow or PyTorch.

TensorFlow graph:

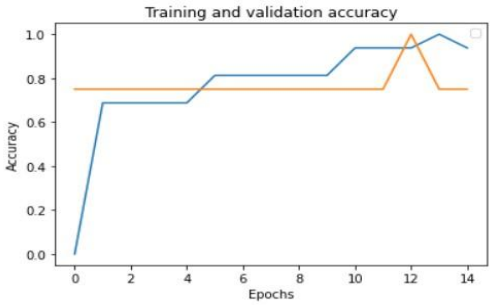


Fig. 23. Graph for training and validation accuracy in TensorFlow.

*PyTorchgraph:*

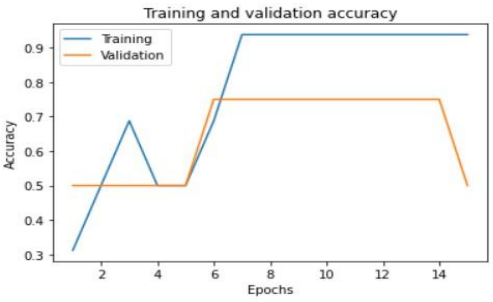


Fig. 24 . Graph for training and validation accuracy in PyTorch.

From the above two graphs, degradation in PyTorch accuracy can be seen while TensorFlow stands as better model.

*4) Training and validation loss*

Similarly, training and validation loss graph can help us in the judgement of better model from TensorFlow or PyTorch.

*TensorFlow graph:*

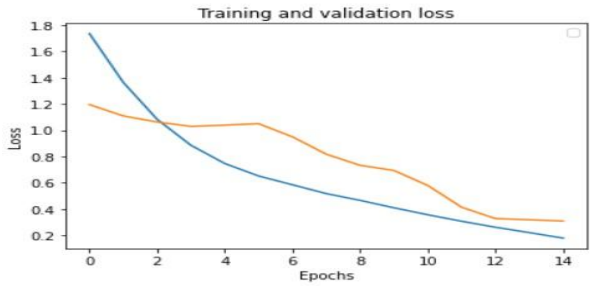


Fig. 25. Graph for training and validation loss in TensorFlow.

*PyTorch graph:*

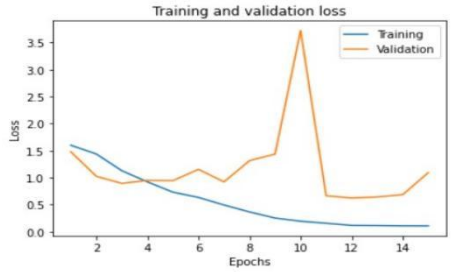


Fig. 26. Graph for training and validation loss in PyTorch.

Above both graph shows that in TensorFlow validation loss decreases, but in PyTorch it takes peaks, which means TensorFlow can handle this type of data more accurately and is giving less validation loss.

## IX. CONCLUSION

Based on the results obtained from the model evaluations, we can conclude the following:

1) The final validation accuracy for the PyTorch model is 0.5000 which means 50%, while the TensorFlow model has a higher validation accuracy of 0.7500 which means 75%. This suggests that the TensorFlow model performed better in terms of correctly classifying the DR severity levels.

2) The training time for the PyTorch model is 113.67 seconds, whereas the TensorFlow model took significantly less time at 38.11 seconds. The shorter training time for the TensorFlow model indicates that it is more time-efficient compared to the PyTorch model in this specific case.

3) Comparing the two deep learning frameworks, TensorFlow and PyTorch, for the task of diabetic retinopathy detection, the TensorFlow model exhibited better performance in terms of validation accuracy and training time. However, it is crucial to consider other factors, such as ease of implementation, flexibility, and community support, when selecting a deep learning framework for a specific task.

4) Further improvements can be made to both models by employing advanced data augmentation techniques, experimenting with different model architectures, tuning hyperparameters, and implementing regularization techniques.

## *IX.* CHALLENGES:

### A. *Imbalanced Dataset:*

The distribution of the classes in the dataset is often imbalanced, with a significant proportion of the data belonging to one class. In the case of the Diabetic Retinopathy Detection dataset, there are more samples of the 'No DR' class than the other classes.

### B. *Large and Complex Dataset:*

The Diabetic Retinopathy Detection dataset is large and complex, with over 35,000 images and five different classes. Processing and analyzing such a large dataset can be computationally intensive, requiring significant hardware resources.

### C. *Noise and Variability:*

Medical imaging datasets such as this one can contain a lot of noise and variability in terms of image quality, lighting conditions, and imaging equipment. This can make it challenging to develop accurate models that can generalize well to new, unseen data.

### *D. Lack of Expertise:*

Developing accurate models for medical imaging datasets often requires expertise in both machine learning and medical imaging. Without proper domain expertise, it can be challenging to understand the nuances of the dataset and develop accurate models.

### *E. Ethical Considerations:*

Working with medical imaging datasets comes with ethical considerations, such as ensuring patient privacy and confidentiality, obtaining proper consent for data use, and ensuring that the models developed are safe and effective for use in clinical settings.

## X. REFERENCES


[1] "Deploying Deep Learning Models on AWS EC2 and S3 with Flask" by K. Shetty and S. Suresh: This paper describes a method for deploying deep learning models using Flask, Docker, and AWS EC2 and S3. The authors demonstrate how to train a deep learning model, save it to an S3 bucket, and deploy it to an EC2 instance using Flask.

[2] “TensorFlow: A System for Large-Scale Machine Learning" by Martín Abadi et al., 12th USENIX Symposium on Operating Systems Design and Implementation in 2016.

[3] "PyTorch: An Imperative Style, High-Performance Deep Learning Library" by Adam Paszke et al. (2019)

[4] "Deep Residual Learning for Image Recognition" by Kaiming He et al. (2016)

[5] K. Team, “Keras documentation: About Keras”, Keras.io, 2020. [Online]. Available: https://keras.io/about. 2020.

[6] Chang, Hong-Yi; Liu, Pei-Lin; Wang, Wei-Chun.Chen, Tung-Ching (2019). [IEEE 2019 IEEE International Conference on Consumer Electronics -Taiwan (ICCE-TW) - YILAN, Taiwan (2019.5.20-2019.5.22)] 2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW) – English learning tool for elders- design of instant fruit identification system with TensorFlow., (), 1–2. doi:10.1109/icce-tw46550.2019.8991988

[7] https://github.com/pannous/tensorflow-speech-recognition

[8] https://github.com/pytorch/vision/tree/main/torchvision/models

[9] "Deep Residual Learning for Image Recognition" by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016): This is the original ResNet paper that introduced the residual learning approach, which involves adding shortcut connections between layers to improve the flow of information in deep neural networks.


*Github repo:*
https://github.com/ChinthapalliNikhithaChandana/DiabeticRetinopathyDetection