Infyma Hackathon

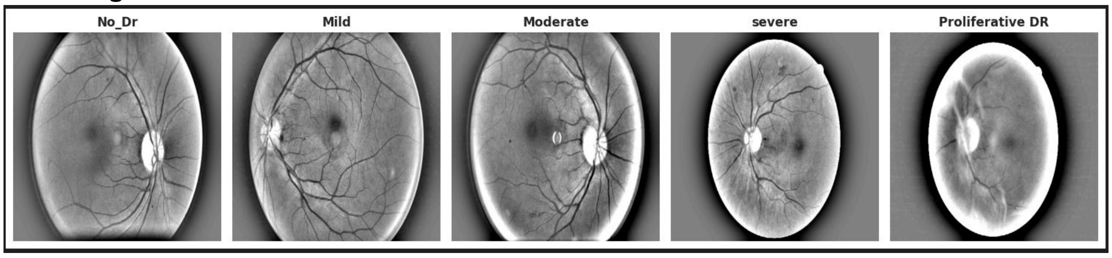
Team Name : Little Learner
Team Lead : Zaraar Malik
Total Members : 1

Problem Statement

The problem statement was very simple to understand. We had to perform image classification on a provided dataset which belonged to the Field of Doctors specifically to Optometry.

Dataset Details

The dataset provided to us has the following name :
'Diabetic_Retinopathy_Balanced'.
This dataset is the derivative of another dataset which is named as following :
'Diabetic Retinopathy (resized)'
The Dataset Contains Images from 5 different Classes which are mentioned as following :



As show above these are the Five different images belonging to 5 different classed provided in the dataset.

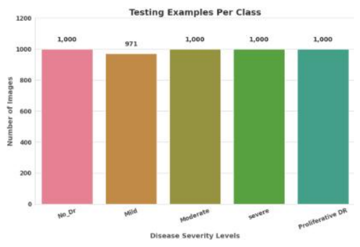
Steps Taken to Solve the Problem

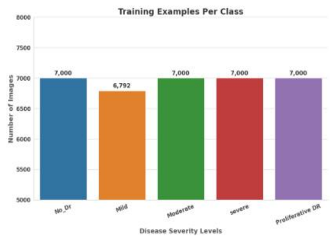
Hidden Insights from Dataset:

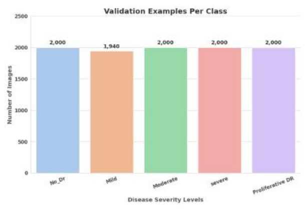
The first step I did was to find out the data distribution inside the dataset . Main Goal was to find out if there is any imbalanced class.
Reason : If any class is imablanced , it can highly effect the model during training.

- If the class has very less samples, the model may not be able to capture the image features.
- If the class has samples a lot more than all the othe classes, the model may become biased because its weights will more likely be effected by a class which has a lot of samples as comapred to the other class which as a lot less samples.

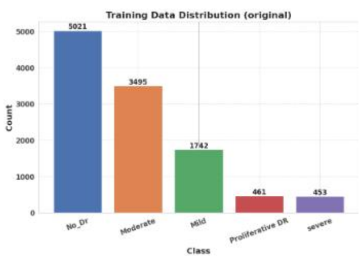
Following is data Distribution for all the 3 folders (training, testing and validation):

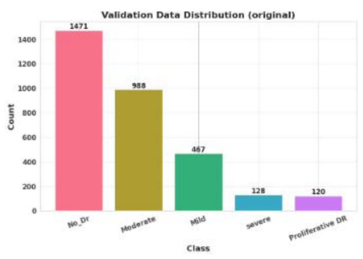


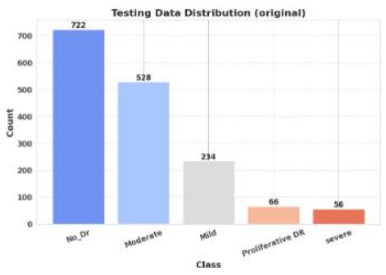




Looking at these graphs, one can easily figure out that the dataset is almost balanced with a few less samples from the 2nd class. However, after observing closely and re-reading the provided dataset card from Kaggle, I realized that these graphs are not true. Actually, **Augmentation Techniques** have been implemented on a **highly-imbalanced data** inorder to make it slightly better so that performance of **DL models** can be increased. Upon Investigation, Following are the actual figures of the dataset provided:







As we can see from these images, I removed all the augmented files from the dataset and just explored the distribution of the base images. From this we can clearly see how much imbalanced the dataset is.

Binarization Problem ?

Another Important thing to notice in the images was that during the creation of the Dataset, Binary Thresh-Holding was implemented on the Dataset. Now, In this case where we were supposed to identify a specific Disease Stage from Retina Images, I think Binary Thresh-Holding resulted in a negative impact on the performance of the model. The main purpose of Thresh-Holding in the simplest terms is to separate the background from the foreground. This is shown below :

Original Image

Thresholded and segmented Image

In our case, this posed as a negative image processing technique as instead of Binary Thresholding, we should have implemented a multi-level image segmentation. This could have resulted in a better per-forming model as the disease areas could have been highlighted with a specific color which could further guide the model to understanthe effected areas in the Retina.

Augmentation's Benefit ?

Now, another reason why image augmentation was a bit dangerous in this case was that there were barely any images for the Last 3 classes as compared to first 2 classes as shown in the above bar plots. Now, from the Initial graphs we had 7000 training samples for the class "Proliferative_DR" and then originally we had 461 training samples for the Class "Proliferative_DR". That means roughly every Image had roughly 15 copies of itself with augmentation in the training data. What this means is that augmenting would lead to 80-87% similar image with a little bit difference via e.g Contrast or RandoFlip or Zoom etc. Mostly, the image features remain the same. This is one of the major factor why every model I trained (which I will discuss further) starts to overfit after reaching 70-75% validation accuracy. Yes, image augmentation was absolutely necessary but this how it has effected the DL Models.

Finally, 1 key factor to note is that this data was derived from a previous data which was already highly imbalanced. Moreover, Following are some of the reults of people who have tried solving this problem:

1. https://www.kaggle.com/code/surajbansi28/diabetes-detection-through-retinopathy
   a) **Best Validation Accuracy: 0.5668 | Testing Accuracy : 0.5423**
2. https://www.kaggle.com/code/lalumutawalli/cnn-retdbm-75-accuracy
   a) **Val Loss: 23.53 | Val Accuracy: 0.6804**

Desciding the Best Model for the Dataset :

The is one of the most crucial part of this Hackathon as due to time contraint , Training every model With a good performance score was nearly impossible in just 48hrs and using google colab and kaggle with limited resources. So, using what I already learnt about from the Dataset, I decided to remove Some of the models from my list which I was going to train. Following are the models I removed :

- **VGG-19 and VGG-16:**
  - **a.** This model was actually trained on ImageNet Dataset which did not contain me dical images. It contained Natural Images
  - **b.** No Skip Connections. Vanishing Gradient Problem would occur for me in this case.
- **Transformer-Based Models:**
  - **a.** The only reason I did not use these models was that, they split image into patch es. In this case "Spatial Information" or "Spatial Hierarchies" was very important which could be easily lost once the images are splitted-up into patches
- **UNET Model:**
  - **a.** The image were already in a Binarized Format. I could not understand how or w hat wa s the point of Segmenting a binarized Image. Had the Images been in their original format, I would have selected the UNET model.
  - **b.** UNET Model is a pretty computationally expensive model so even if I trained th is model optimizing it and deploying would still pose an issue in a resource-restric ted envrionment.
- **Ensemble Model :**
  - **a.** My main plan was to exhaust Single Models and if none of them worked, then I would go to this approach . Since, I was (I think based on results available on the web) slightly successful, I did not use this approach and then again time was also an issue. However, again in a resource-restricted environment, this would still po se an issue .

With all of these models out of the equation, I was left with the following models :

- Resnet
- EfficientNet
- DenseNet
- InceptionV3

Out of these 4 , I Trained the first 3 models along with a Vision Transformer(I was just curious whethe r my statement regarding them was correct or was I just solely saying words which have no meaning). I didn't Train InceptionV3 since I worked with it sometime ago and I just wanted to learn a new Archit ecture and In this case it was the EfficientNet Architecture.

Results of Resnet Model

There are multiple models of resent such as resnet-18, resnet-34, resnet-101 etc. I decided to use the Resnet-50 Model. The main reason was trying to extract useful features using a powerful pre-trained model and then pass those features to Dense Layers to perform Classification. I attacted a Custom Dense Network after this model to perform the Classification for the dataset. Following are the scores of the model :

Train Loss: 0.1115 | Val Loss: 0.7196 | Val Acc: 74.67% | Testing Loss: 0.7196 | Testing Acc: 74.39%
`(Results are also available in the notbook in Rough_Notebooks Folder)`

Results of DenseNet Model

After the Resnet Model. I saved that and then tried to train this model. Unfortunately, this model started to overfit on the training data. My deduction for this model was that this was previously not trained on medical images. Moreover, The Architecture Complexity of Resnet Model is far greater than this model. Despite, its performance trade-off, I had a very difficult dataset due to which this model was unable to perform. I didn't event checked the Test Acc since this model overfitted here.
Train Loss: 0.3278 | Val Loss: 0.7092 | Val Acc: 73.87% |
`(Results are also available in the notbook in Rough_Notebooks Folder)`

Results of Vision Transformer Model

What I mentioned during the earlier stage was proved true during my implementation. I was not planning to implement this model(not from scratch :)) but I just wanted to make sure if what I was thinking and what I stated in the documentation could hold true or not. I trained this model and just as I said, this model performed the worst out of all the 4 models I trained. The procedure was the same as for the prevously mentioned models. I used the following ViT model :

"vit_base_patch16_224"

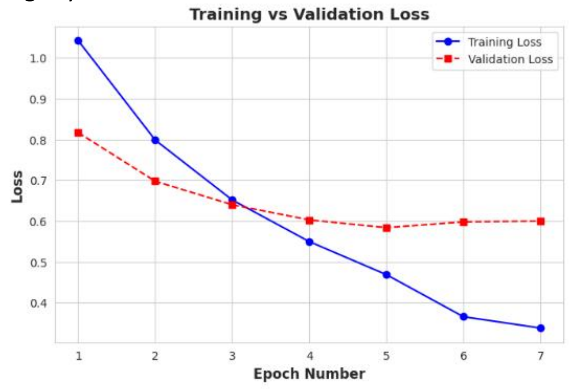
Train Loss: 0.4066 | Train Accuracy: 83.05% | Val Loss: 1.2174 | Val Accuracy: 60.38% |

`(Results are also available in the notbook in Rough_Notebooks Folder)`

So, as the results are shown, my statements were correct that transformer-based models would not be able to perform in this case with such type of dataset.

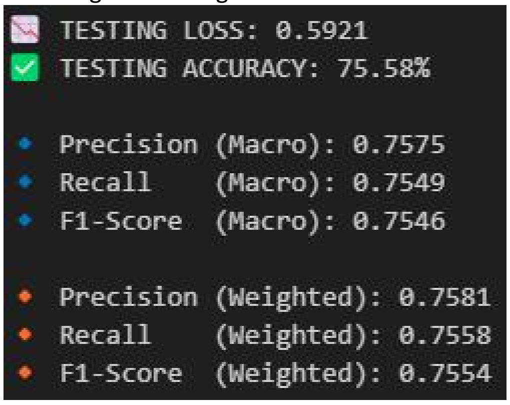
Results of EfficientNet-B4 Model

Finally, atlast we are here on the best model. Best things are saved for last. This model performed slightly better than the Resnet-Architecture. Following are the results :



This was the best model, So I made the final graph only for this model (:) I was just tired even though I just had to copy paste the code here and there).

Following the Testing Result:



```
📉 TESTING LOSS: 0.5921
✅ TESTING ACCURACY: 75.58%

🔹 Precision (Macro): 0.7575
🔹 Recall    (Macro): 0.7549
🔹 F1-Score  (Macro): 0.7546

🔸 Precision (Weighted): 0.7581
🔸 Recall    (Weighted): 0.7558
🔸 F1-Score  (Weighted): 0.7554
```

I used the same approach. Dense Layers infront of the pretrained model. However, I experimented wi th a few more parameters which are mentioned as following :

- **AdamW Optimizer:** This basically is a modified version of Adam Optimizer which typically e veryone uses. Moreover, 1 major reason to use this was that it has decouples weight decay which help in better generalization. I could explain more but I think I already wrote a lot. O ptimizers are a very interesting feature in DL.
- **Mixed Precision:** This is more of an optimization thing. This basically tries to save memory d uring training while maintaining performance of the model.
- **StepLR:** This decays the learning rate as training goes on. First Explore the Feature space an d then slowly generalize on it

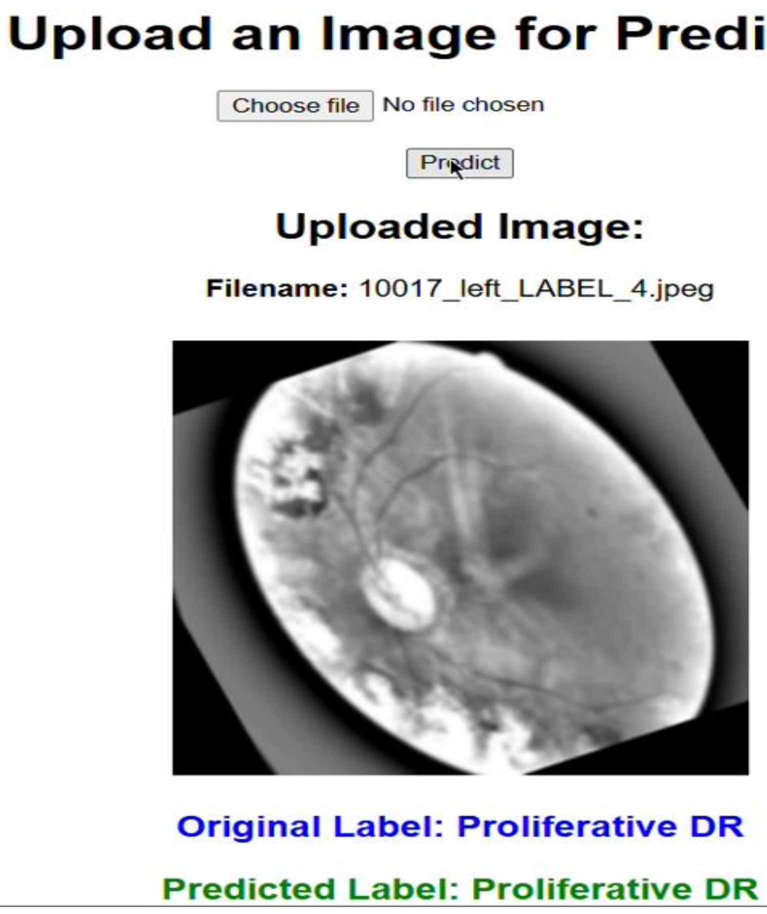
Optimization of EfficientNet B4

There are multiple optimization techniques available on the internet.
Since lack of time, I only implemented 2 techniques out of which each has its own Benefit.
Quantization was not possible as Efficient Net has a different architecture which does not support this.
I still tried to do it but faced with errors mostly stating that something is wrong with the architecture.
So, I further investigated and found out that

- **Just-In-TIme(JIT) Compilation :** This basically removes the python overhead in simple terms. So, essentially making the code run faster. But thde code on GPU was faster but not on the CPU. The Results are clearly mentioned in the JupyterNotebook.
- **ONNX(Open Neural Network Exchange) Format:** This is quite famous and it has been implemented in a lot of places. I also used this sometime ago. And believe me if someone wants to make a DL model run faster on CPU this is pretty important for them. After converting the model to ONNX format, the inferrence on CPU went from 14seconds - 4.4seconds. Pretty amazing right. Here is the best part. There was no Accuracy Drop in the Testing Dataset.

Deployment of Optimized Model

The model was first optimized and then deployedd on my local machine. I used a simple Flask Code to make a simple webpage where I could upload an image. Click the predict button and the Image would be classified by the deployed model.
Following is the picture of the deployed model:



Upload an Image for Prediction

Uploaded Image:

Filename: 10017_left_LABEL_4.jpeg

Original Label: Proliferative DR

Predicted Label: Proliferative DR

Conclusion:
This Hackathon has been a great journey for me. I cant believe how much I have learned in such a short period of time. I am really happy that I was able to work on a real world problem and express myself as well. I am truly honoured to have completed this Hackathon in due time.
I would love to participate in the upcoming hackathon's as well.

Thank you Team Infyma For such a lovely and enjoyful journey