Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	AML 2404
Section	Section 2
Project Title	Skin Diseases Classification using Deep Learning
Group Name	G
Student names/Student IDs	Tomson George (C0857730) Praveen Mahaulpatha (C0860583) Thulana Abeywardana (C0861333) Jaskaran Singh Moti (C0860026)
Reporting Week	Week 6 (17 October 2022 to 23 October 2022)
Faculty Supervisor	William Pourmajidi

1. Tasks Outlined in Previous Weekly Progress Report

Task 01: Debugging GitHub workflow to access the dataset

Responsible: (Praveen)

This task is an extension of the previous week's task of implementing MLOps on the project. The objective was to tackle an obstacle encountered when trying to implement and run a git workflow where the dataset was sitting in the local repository, and the workflow could not run due to that.

Task 02: Structuring the flask code in a proper format and implement web templates

Responsible: (Tomson)

As the flask code was not in a structured way and the final user interface was not in a proper manner, the code was restructured and custom made html files were used to show the home page in a more user friendly manner.

Task 03: Augmenting image to improve the performance

Responsible: (Thulana)

As the model has not improved its performances, the images were resized as a step of data preprocessing. First the images were resized to a size of 474 x 720 and it took longer time to run the model and even after that, model has not shown any significant improvement of its performances. Then images were rescaled to a size of 224 x 224, but the performance was not improved.

Task 04: Get familiar with Jupyter dash and plot graphs

Responsible: (Jaskaran)

The objective of the task was to create interactive graphs and get familiarize with JupyterDash. From a sample dataset, some interactive graphs were plotted and learned the basics of Jupyter Dash.

2. Progress Made in Reporting Week

Task 1: Praveen: Debugging GitHub workflow to access the dataset

Below options were identified to overcoming this issue;

1. Hosting the dataset on Cloud and training a local model
2. Host the dataset along with the model and automate CI process on the cloud itself
3. Create a toy dataset on Git (This will only validate the code but not the model accuracy)

2.1.1. Hosting the dataset on Cloud and training a local model

In this solution we will be using Azure Cloud computing service offered by Microsoft.

The pre-requisites are as follows,

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning today.
- An Azure Machine Learning workspace.
- The Azure Machine Learning SDK for Python installed (>= 1.13.0), which includes the azureml-datasets package.

The basic solution is to create an unregistered TabularDataset from a web URL anywhere. It could be hosted and accessed the same in the model training. The documentation offers an example of a dataset with CSV file format which is easily translatable to a tabular form but our solution needs a different method as it contains images. Either we will have to convert the images to vectors based on the dimensions and take it to a tabular form but for this week, we have not tried that solution given the time limitations.

Also, for that matter, we experimented with DVC, which is a " free, open-source VS Code Extension and command line tool. DVC works on top of Git repositories and has a similar command line interface and flow as Git. DVC can also work stand-alone, but without versioning capabilities."

The architecture to implement CI/CD using DVC is as below.

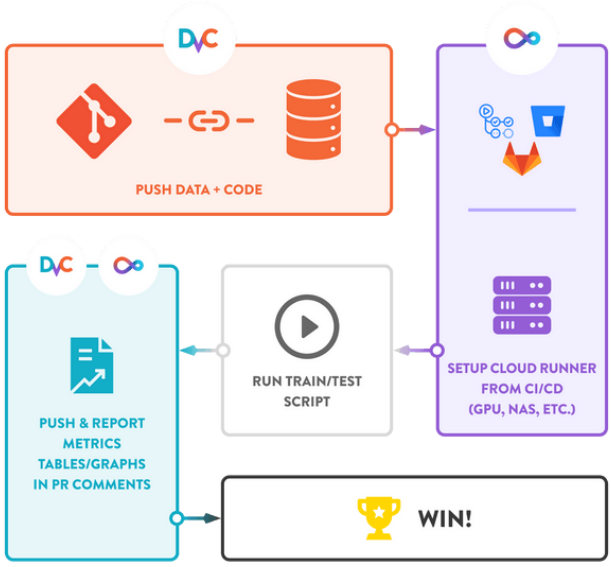

Figure 2.1

Models, Data, and Metrics as Code: DVC removes the need to create versioning databases, use special file/folder structures, or write bespoke interfacing code. Instead, DVC stores meta-information in Git ("codifying" data and ML models) while pushing the actual data content to cloud storage. DVC also provides metrics-driven navigation in Git repositories — tabulating and plotting model metrics changes across commits.

Data Validation: It is common practice for tests to be triggered each time a code change is pushed to a repository branch. DVC can be used in a similar manner to checkout different data versions for the purposes of testing and running sanity checks. Mistakes can be caught automatically without requiring contributors to set up complicated tests locally.

Metrics (Model Validation): Whenever a change is committed, DVC can check that the pipeline (including data, parameters, code, and metrics) is up to date, thereby ensuring that Git commits and model artifacts are in sync.

Refine in the Cloud: Rather than frequently updating models locally (e.g. based on new data from regular feeds), DVC and CML let you retrain/refine in the cloud. For example, CI providers allow scheduling of regular jobs. Every day, a job could use CML to provision a GPU server on which DVC will pull in data from a regularly updated source, checkout a pre-existing model, and then deploy an updated model refined on the additional data.

Similarly, other options were studied and identified higher technical capabilities and scope is required if we are to proceed with the implementation of MLOps. Also, since we are finding difficulties in the core processes of the project of model training and cloud hosting, we have decided to pause the implementation of MLOps temporarily and focus the effort on the former tasks.

Task 2: Tomson: Structuring the flask code in a proper format and implement web templates

1.1 Rewrite the flask application in a more structured way

Earlier the code was written in more unstructured way like importing libraries in the middle and coding according to the flow of learning. Folder and file naming were not how it was done conventionally. The following is the structure we are trying to achieve,

```
/yourapp
    /run.py
    /config.py
    /app
        /__init__.py
        /views.py
        /models.py
        /static/
            /main.css
        /templates/
            /base.html
    /requirements.txt
    /yourappenv
```

`run.py` - contains the actual python code that will import the app and start the development server.
`config.py` - stores configurations for your app.
`__init__.py` - initializes your application creating a Flask app instance.
`views.py` - this is where `routes` are defined.
`models.py` - this is where you define models for your application.
`static` - contains static files i.e. CSS, Javascript, images
`templates` - this is where you store your `html` templates i.e. `index.html`, `layout.html`
`requirements.txt` - this is where you store your package dependancies, you can use `pip`
`yourappenv` - your virtual environment for development

Figure 2.2

So the code was rewritten, files and folders were renamed to follow the standard way.

1.2 Learn more about creating web templates for uploading and displaying the image

A sample HTML template for uploading images was created and rendered using the render_template function of the flask library. Also learned more about passing values while calling API.

Task 3 : Thulana: Augmenting image to improve the performance

Step 1:

Due to the poor performance of the model, data was augmented in this stage. Initially, Inges were flipped vertically and horizontally. Then images were rotated randomly. The model was run for 5 epochs and inclined accuracy and decline were observed for both training and validation data. After this step, a better performance was observed but the accuracy level was insufficient.

The corresponding parameters of the nural network model are as follows:

Layer	Kernels	Filter Size	Activation Function	Shape / Other parameters
Input Layer				(224,224,3)
Random flip				**horizontal_and_vertical**

Random rotation				**0.2**
Convo2D	32	3*3	“relu”	
MaxPool2D				
Convo2D	32	3*3	“relu”	
MaxPool2D				
Convo2D	32	3*3	“relu”	
MaxPool2D				
Flatten				
Dense				128
Dropout				0.5
Dense				128
Dropout				0.5
Dense				128
Dropout				0.5
Output Layer				4

Table: 2.1

Parameter	Value
Learning Rate	0.01
Epochs	5
Loss Function	SparseCategoricalCrossentropy (logits = False)

Table: 2.2

Following results were observed:

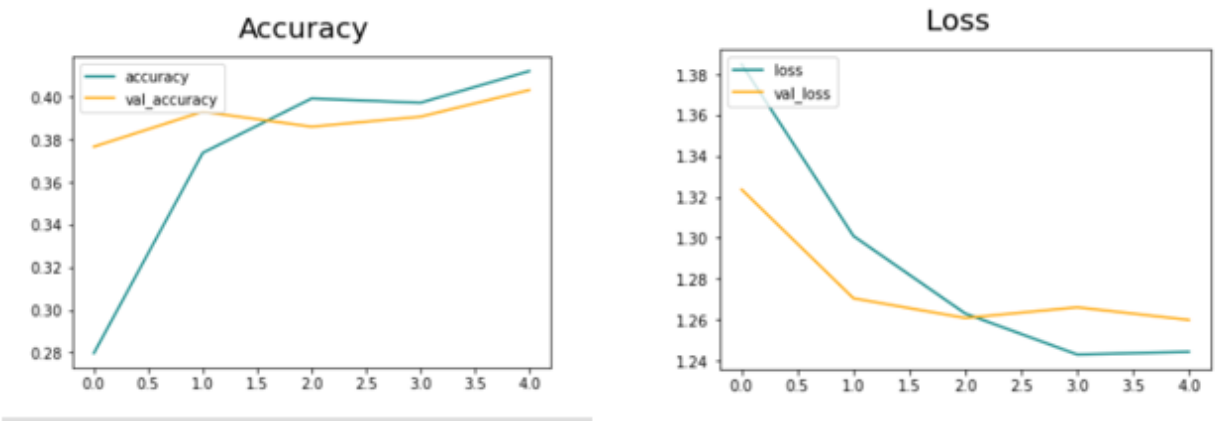

Figure: 2.3

Step 2:

To obtain a better performance, the model was run for 10 epochs; but the expected model performance was not observed. The corresponding parameters of the nural network model are as follows:

Layer	Kernels	Filter Size	Activation Function	Shape / Other parameters
Input Layer				(224,224,3)
Random flip				**horizontal_and_vertical**
Random rotation				**0.2**
Convo2D	32	3*3	“relu”	
MaxPool2D				
Convo2D	32	3*3	“relu”	
MaxPool2D				
Convo2D	32	3*3	“relu”	
MaxPool2D				
Flatten				
Dense				128
Dropout				0.5
Dense				128
Dropout				0.5

Dense				128
Dropout				0.5
Output Layer				4

Table: 2.3

Parameter	Value
Learning Rate	0.01
Epochs	10
Loss Function	SparseCategoricalCrossentropy (logits = False)

Table: 2.4

Following results were observed:

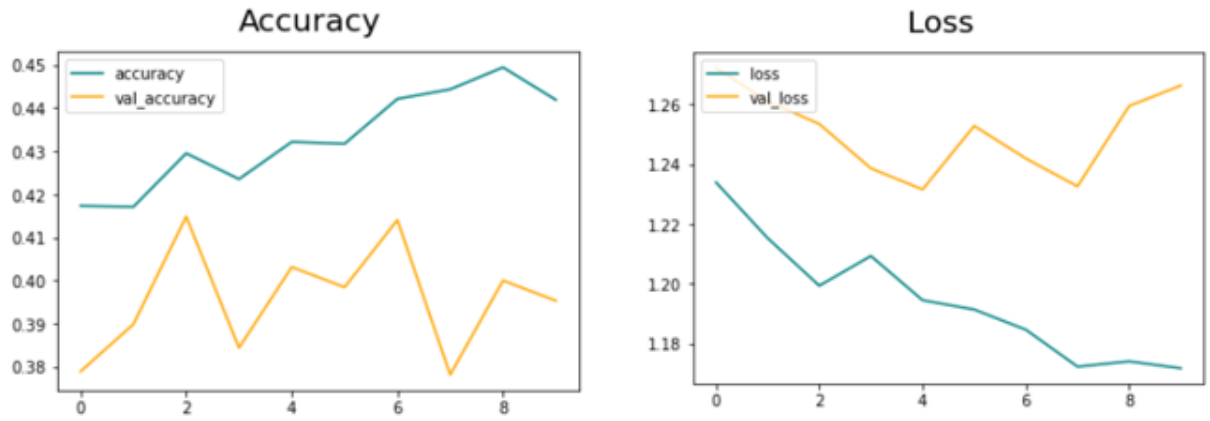

Figure: 2.4

Step 3:

In this step, the order of selecting training, test and validation data was changed as follows (data size is same).

Previous order:

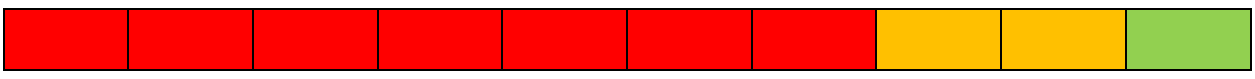

Figure: 2.5

Present order:

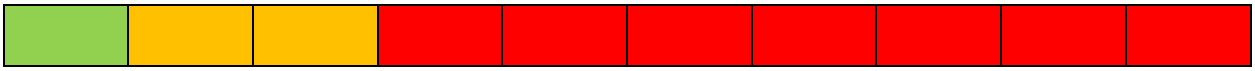

Figure: 2.6

	Training
	Test
	Validation

Table: 2.5

Then the model was run without changing the parameters in the previous step (Table 3.3 and 3.4). The following results were observed:

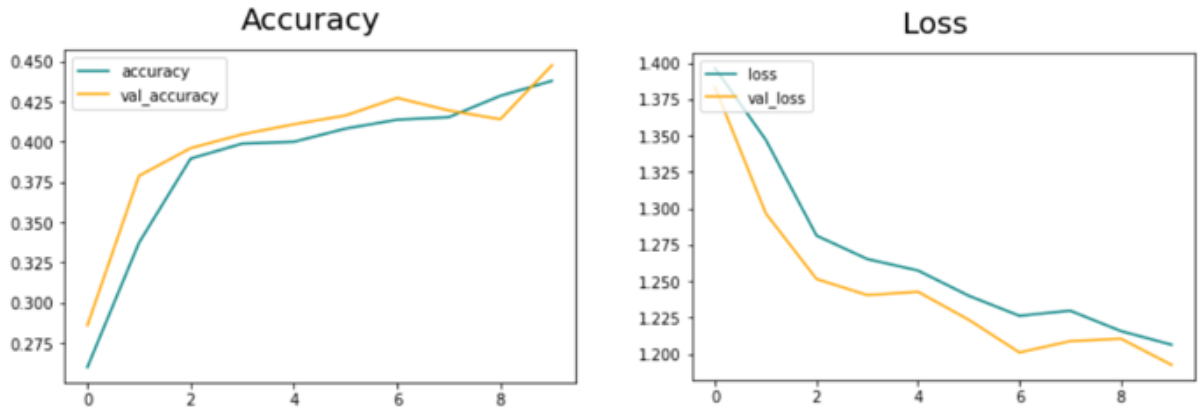

Figure: 2.7

Step 4:

Again the order of selecting training, test and validation data was changed as follows.

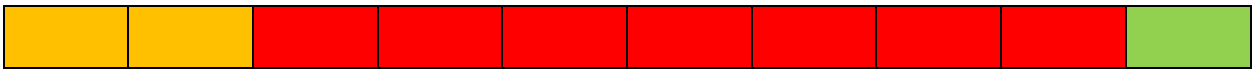

Figure: 2.8

Then the model was run without changing parameters in table 3.3, for 20 epochs. The following results were observed:

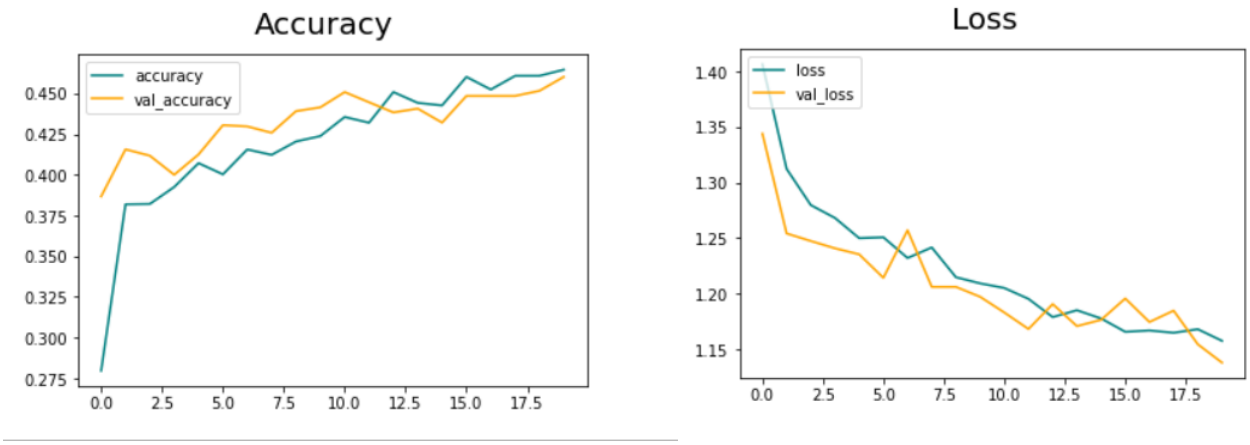

Figure: 2.9

Step 5:

The model was run without changing parameters in table 3.3, for 60 and 100 epochs. The following results were observed:

60 Epochs

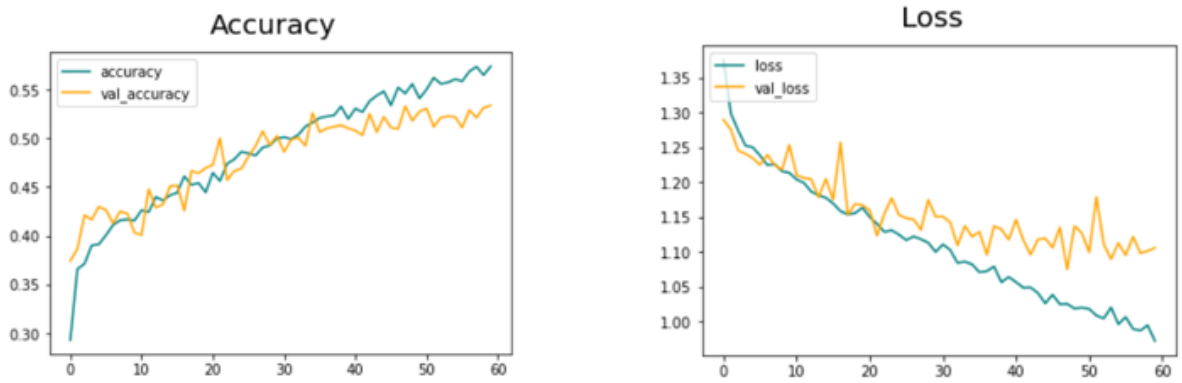

Figure: 2.10

100 Epochs

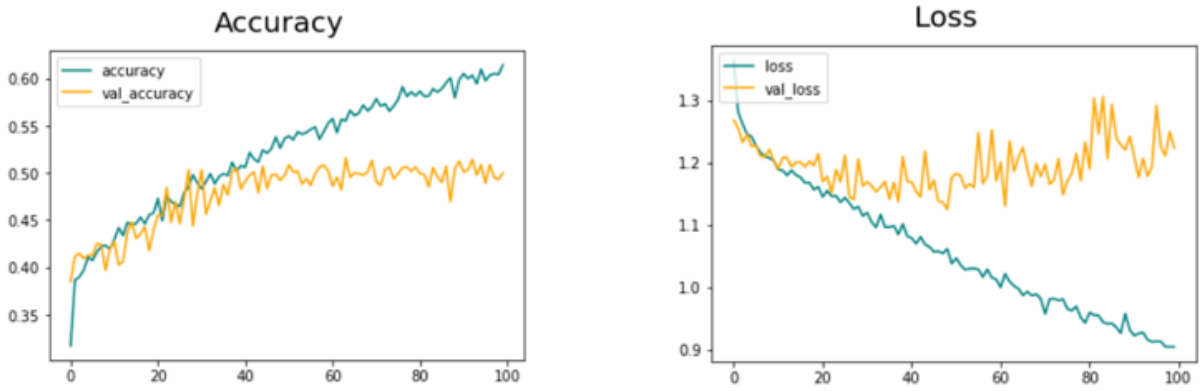

Figure: 2.11

After data augmentation, comparatively a better result was observed. But the accuracy and loss were not up to the expected level. Therefore, more data augmentation is required.

Task 4: Jaskaran: Visualize the model performance using plotly

Interactive graphs were created using JupyterDash and got familiarized with it.

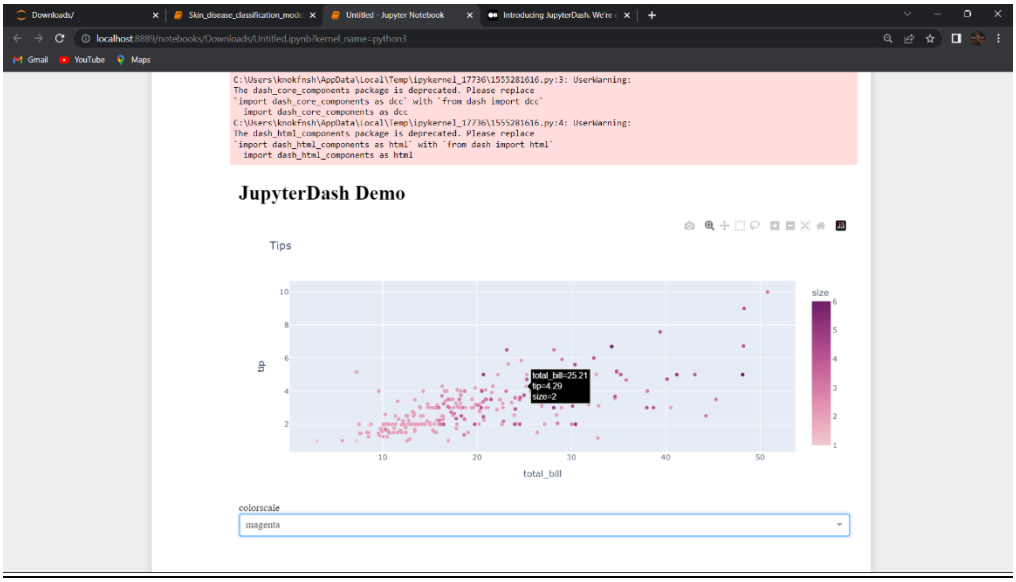

Figure 2.12

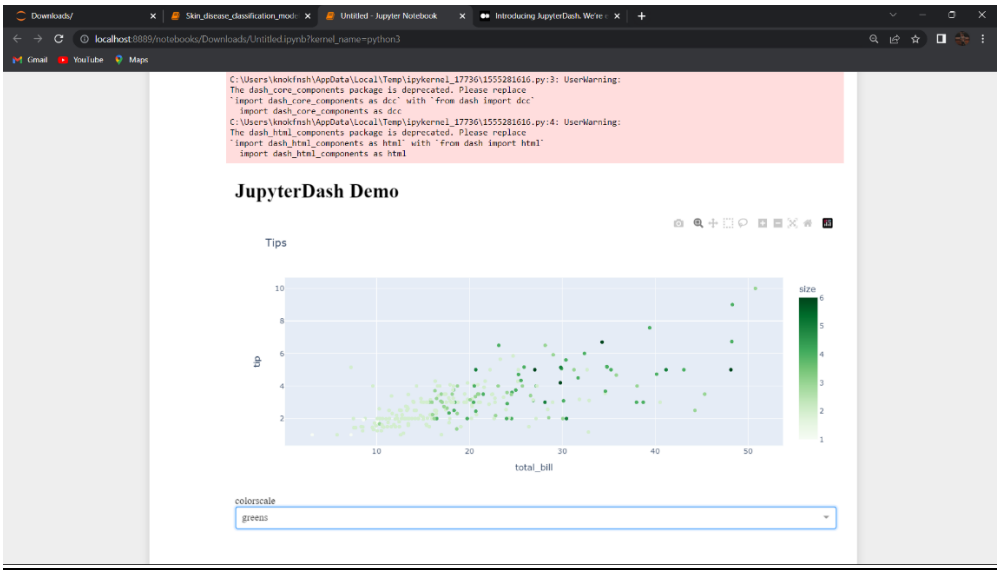

Figure 2.13

3. **Difficulties Encountered in Reporting Week**

a. The technical competency for implementing MLOps requires time, and the team's current expertise is insufficient to expedite the process.
b. While experimenting with GET and POST calls, errors like key mismatch, parameter missing etc were occurred. Some time was spent on identifying the fault in code.
c. Training the model took a lot of time since the dataset has more images.

4. **Tasks to Be Completed Next Week**

Tasks	Responsible
Create a dummy python file on the cloud and access the image uploaded by the user through it.	Tomson
Creating a python file to load the saved model, predict and output the prediction.	Praveen
Optimize the model performance by further augmenting the data	Thulana
Create a graphical representation of model's performance using JupyterDash	Jaskaran