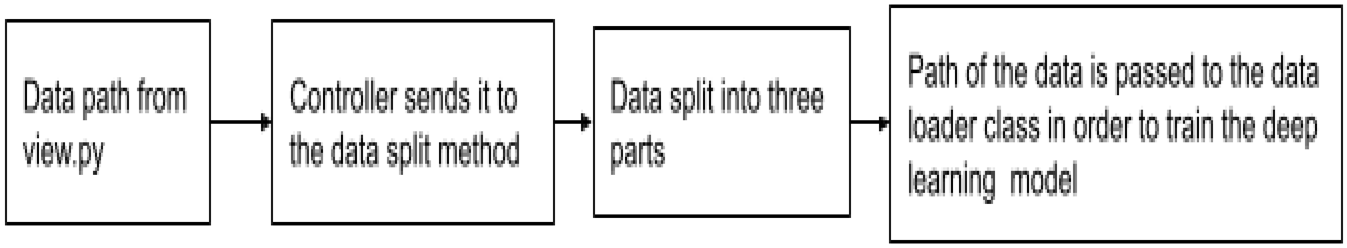
# *REPORT*

## *1.Code walk through:*

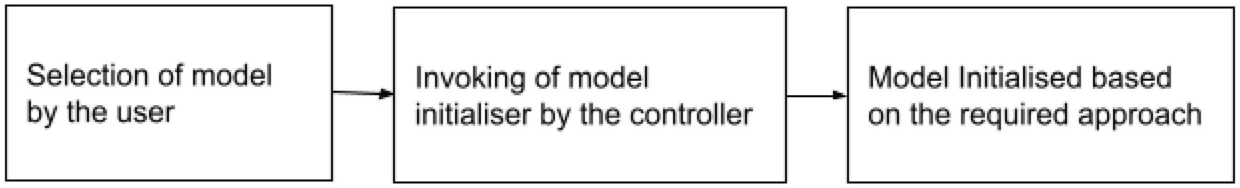
- Data.py:

The user enters the location of the data from the hard drive and is passed to the LoadDataset class in order to be split into three parts: train,valid and test data . Since MNIST already has been split into train and test , the training data path will be split into train and validation set. The method split_train_test returns the paths of the split dataset. The class “data loader” inherits from the base class “Loaddataset” class . It is a generic data loader class which returns data based on the task: classification and segmentation and the mode of deep learning exercise. The methods in this class also embody functionalities to transform and augment the data based on the task and mode.


Data path from view.py
Controller sends it to the data split method
Data split into three parts
Path of the data is passed to the data loader class in order to train the deep learning model

- Model.py:

A choice of five deep learning models is specified to the users in the form of buttons.Here a single class is used to invoke a deep learning model for both the tasks. It makes use of the in-built models available within the torchvision module. It takes in arguments such as model name , the approach to be used for training that is transfer learning , fine-tuning and training from scratch and the option to freeze or unfreeze the feature extraction layers. It returns the model after applying changes with the given options and the input size required by the model.


Selection of model by the user
Invoking of model initialiser by the controller
Model Initialised based on the required approach

- Optimizer.py:

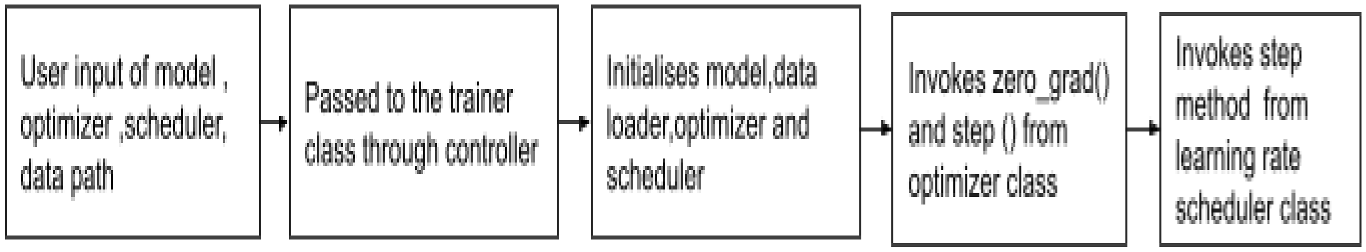
A choice of two optimizers is given to the user after selection and it is passed to the trainer class by the controller . A single class is used for both the optimizers as both involve common methods that are setting zero gradients and stepping it down.

- Learning_rate_scheduler.py:

A choice of three learning rates is provided to the user. A base class includes the step method which is commonly used by all the three schedulers. Three inherited classes for each of the schedulers with their respective parameters are provided.

- Train.py:

A generic training class is used to train the model for both the tasks . The user's choice of data-path, model, optimizer ,scheduler is fed to this class through the controller. The trainer class further invokes the feed of data from data loader to the model , optimizer and learning rate scheduler. This class returns the trained model.


User input of model , optimizer ,scheduler, data path
Passed to the trainer class through controller
Initialises model,data loader,optimizer and scheduler
Invokes zero_grad() and step () from optimizer class
Invokes step method from learning rate scheduler class

- Validation.py:

Once the training process is accomplished, the system proceeds to the validation part which takes in the data path, model as the input and returns validation accuracy as the metrics.

- Test.py:

Once the data is validated, the system moves on to the testing part where the data is tested and metrics such as accuracy, f1 score, precision, recall, dice loss, Iou-loss are returned.

- Inference.py:

Once the user passes an image , the inference method is invoked which in turn provides the recognized characters in the image, the count of different characters in the image and returns a segmented image.

- Core.py :

The data passed by the user is initialised by passing to the respective methods and is in turn passed to the train and inference interfaces.The data from the core is then passed to the controller in order to be displayed to the user.

- Controller.py:

It receives data from the user through the view class and is initialised through the core to the respective methods. It checks the validity of the data and also updates the data to the view class for the user to view the results.

- View.py:

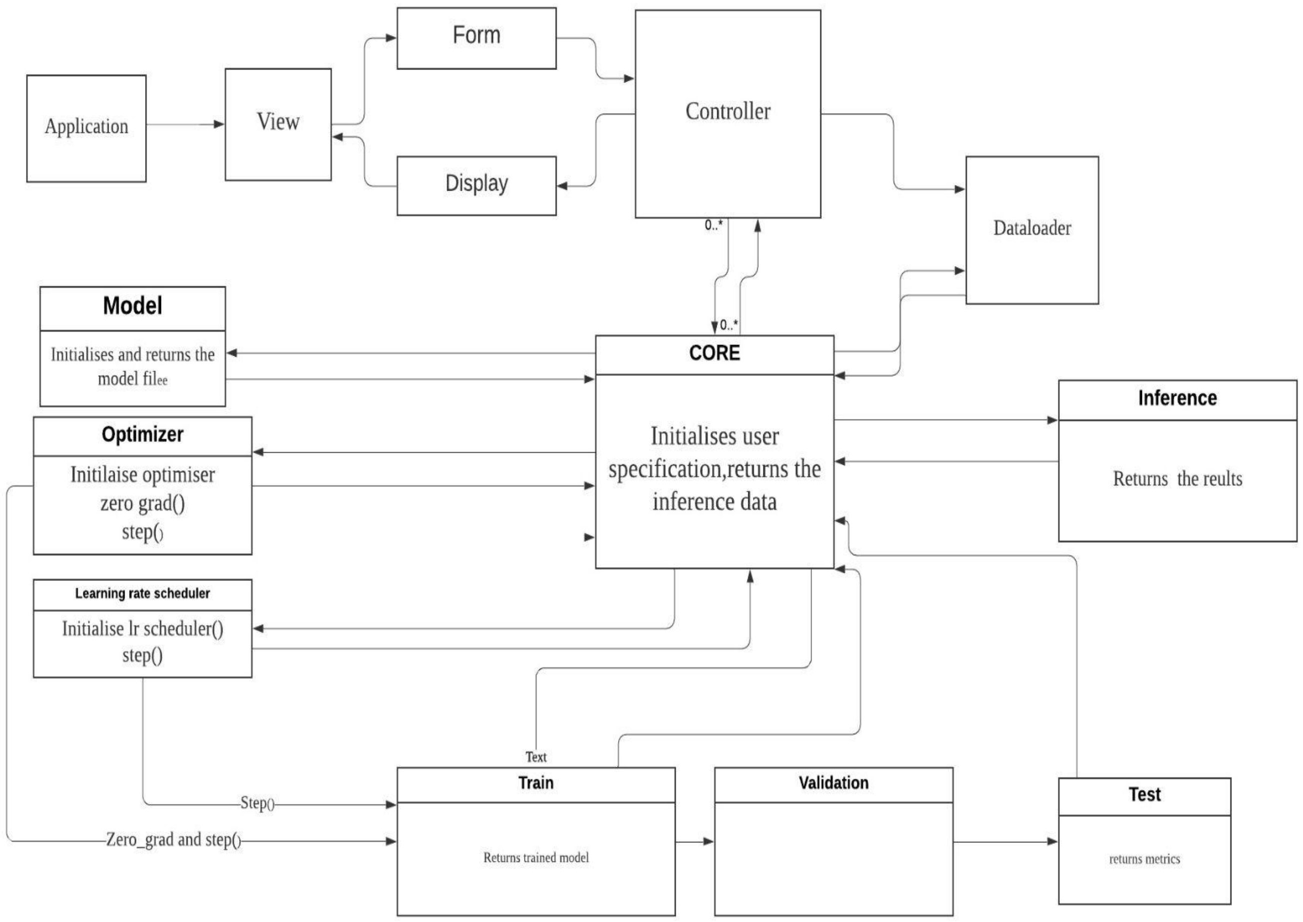
It is the GUI that requests data from the user and displays the results back to them . It accepts entries in the form of strings and button clicks. Button method is used to interact with the user for them to select the desired option.

- App.py:

This is the main application which in turn invokes the GUI and controller.

## *2.UML DIAGRAM:*

The following UML diagram describes the overall architecture of the system:


Application
View
Form
Display
Controller
Dataloader
0..*
0..*
Model
Initialises and returns the model filee
CORE
Initialises user specification,returns the inference data
Inference
Returns the reults
Optimizer
Initilaise optimiser
zero grad()
step()
Learning rate scheduler
Initialise lr scheduler()
step()
Text
Step()
Zero_grad and step()
Train
Returns trained model
Validation
Test
returns metrics