MODLBOX

COM S 402 Senior Design Document

Team Number: 13
Chandan Kumar
Eshita Zaman
Joao Lira, Full stack developer
Bhavesh Dewan, Full Stack developer
Chad Monmany, Backend developer
Jay Tiwari, Backend developer
Bao, Full Stack developer

Team Website: https://seniord.cs.iastate.edu/2024-Jan-13/

Revised: May 8th, 2024

EXECUTIVE SUMMARY

Development Standards & Practices Used

Pyinstaller

PySide6

Python

Tensorboard

Ultralytics

PyTorch (Torch and Torchvision)

Summary of Requirements

- Creating/Arranging the folder structure as per the model.
- Ability to generate graphs.
- Integrating terminal in the application.
- Option to select between classification (labeling an image) and object detection (identifying an object within an image).
- Dataset Selector(Standard dataset), different dataset selector for classification and detection.
- Option to select different models. I.e Alexnet, Yolo V8, Yolo V5 etc.
- Option to select GPU. In order to review the configuration's effectiveness with and without GPU constraints.
- Option to set parameters, i.e., number of Epochs to train, Batch size.

New Skills/Knowledge acquired that was not taught in courses

Python Packaging

Machine Learning

PySide 6

Tensorboard

PyTorch

Table of Contents

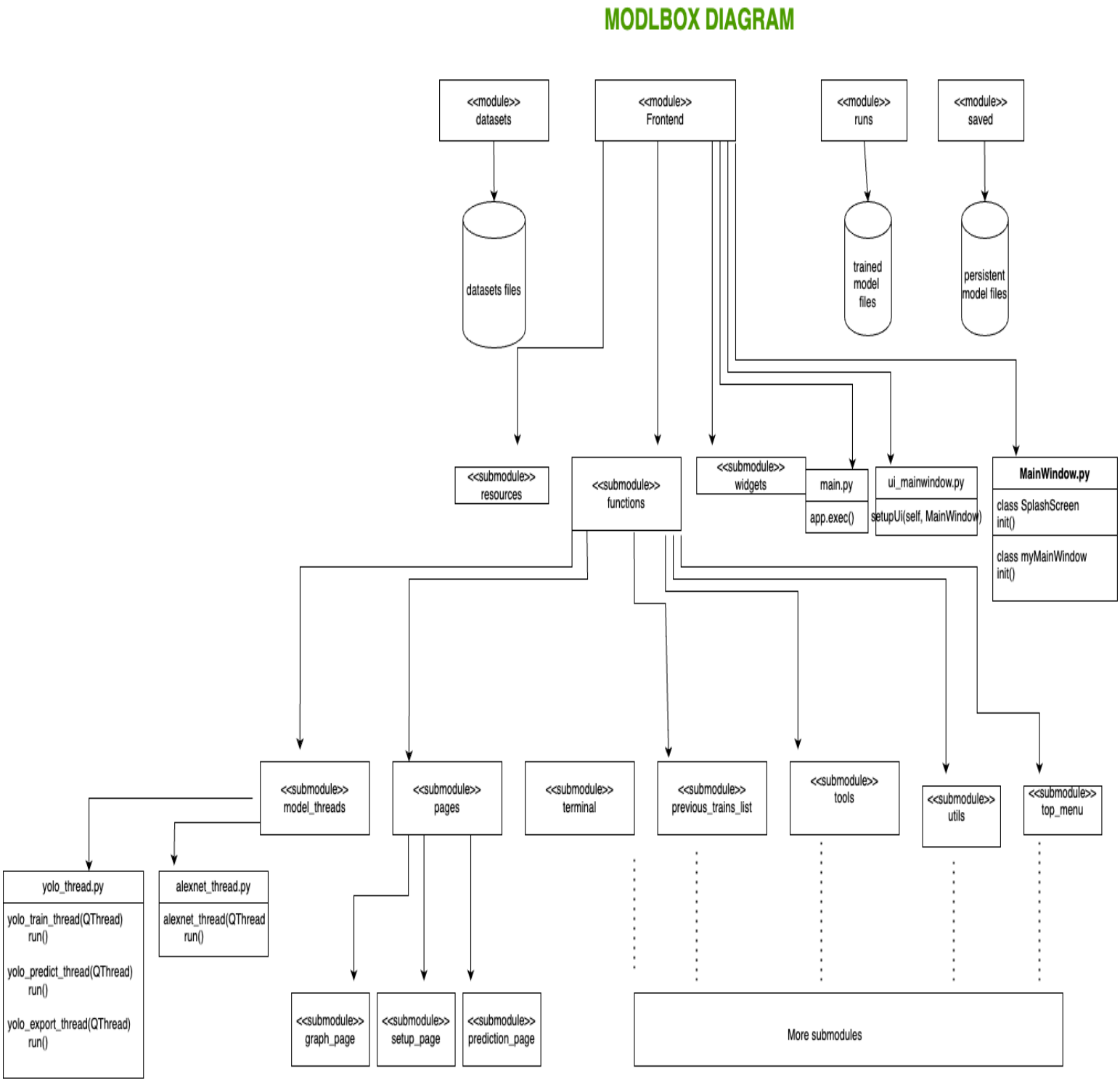
1 Requirements

- Design and create an intuitive frontend.
- Option to select different models.
- Have the option to select between classification (labeling an image) and object detection (identifying an object within an image) models.
- Options to select GPU hardware. In order to review configurations effectiveness with and without GPU constraints.
- Have the option to set parameters, i.e, number of epochs to train and batch size.
- Dataset selector, different datasets available for classification and detection.
- Ability to generate real-time graphical data.
- Integrate the terminal into the application.
- Create/Arrange the folder structure as per the model-generated data.

1.1 Problem and Project Statement

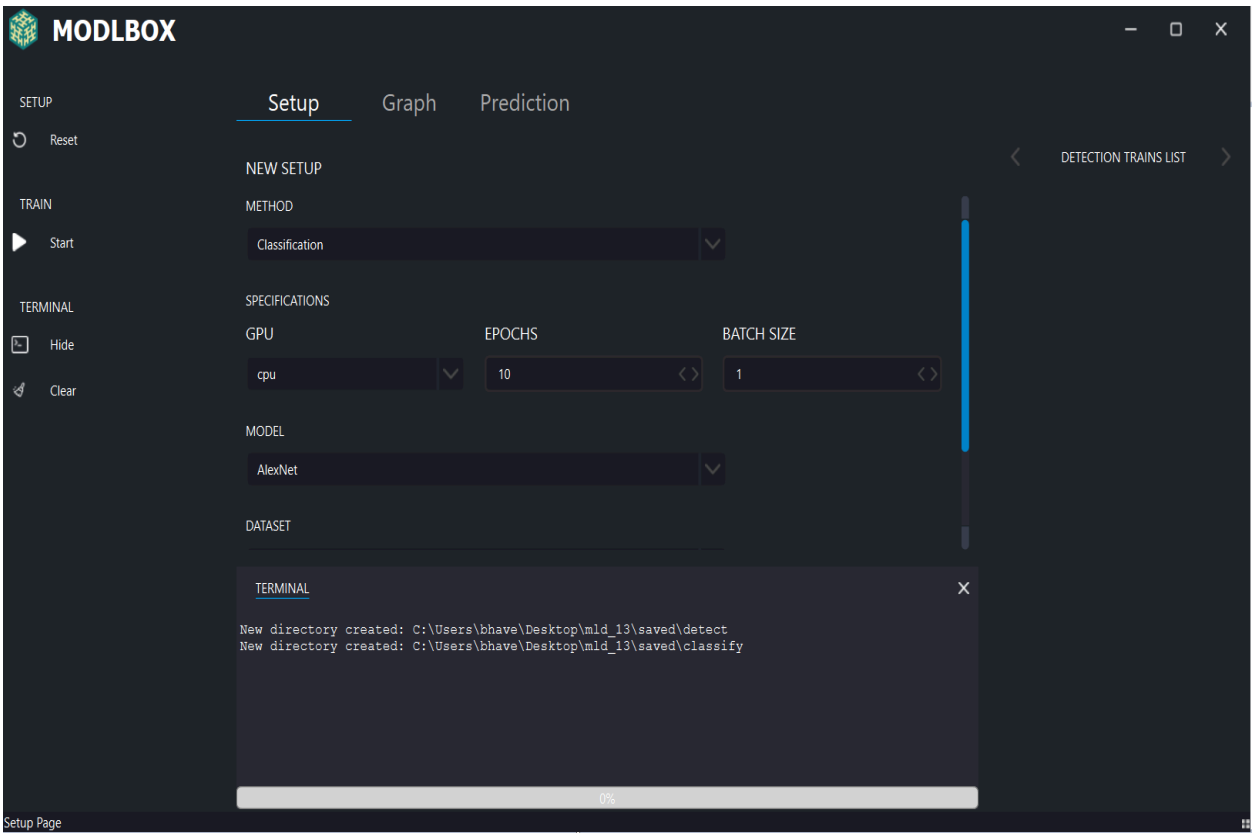
- MODLBOX is an application to speed up the Machine learning/deep learning development lifecycle.
- Aim to provide an intuitive user interface so one can navigate through the software with ease.
- Useful for those who have limited/do not have coding experience
- Integrates various machine learning libraries, thus enhancing flexibility and performance in model training.
- Users can effortlessly select different components of the DL pipeline, including databases and models, to create end-to-end models.

2 Design

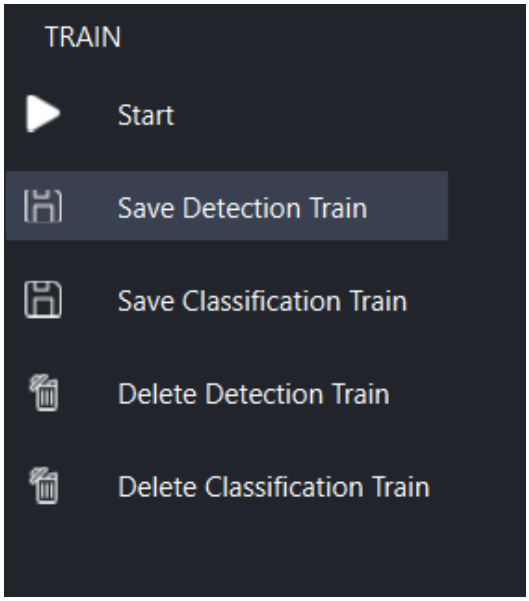

MODLBOX DIAGRAM
<<module>> datasets
<<module>> Frontend
<<module>> runs
<<module>> saved
datasets files
trained model files
persistent model files
<<submodule>> resources
<<submodule>> functions
<<submodule>> widgets
main.py
app.exec()
ui_mainwindow.py
setupUi(self, MainWindow)
MainWindow.py
class SplashScreen
init()
class myMainWindow
init()
<<submodule>> model_threads
<<submodule>> pages
<<submodule>> terminal
<<submodule>> previous_trains_list
<<submodule>> tools
<<submodule>> utils
<<submodule>> top_menu
yolo_thread.py
yolo_train_thread(QThread)
run()
yolo_predict_thread(QThread)
run()
yolo_export_thread(QThread)
run()
alexnet_thread.py
alexnet_thread(QThread
run()
<<submodule>> graph_page
<<submodule>> setup_page
<<submodule>> prediction_page
More submodules

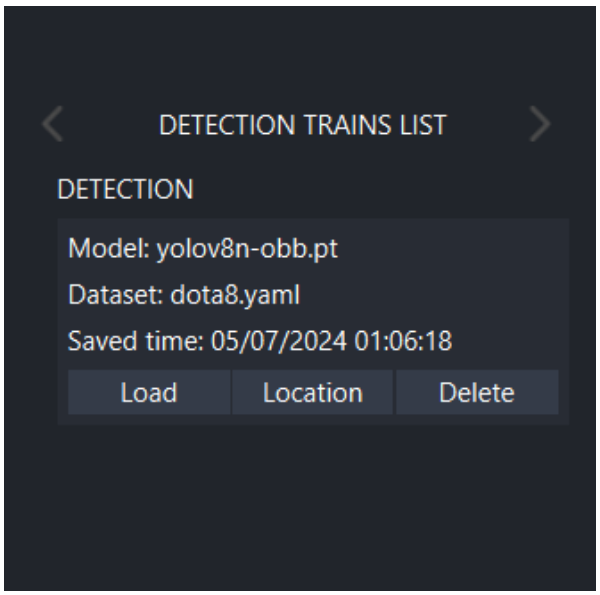
2.1 Splash Screen

2.2 Setup Page, Parameter setting and sidebars

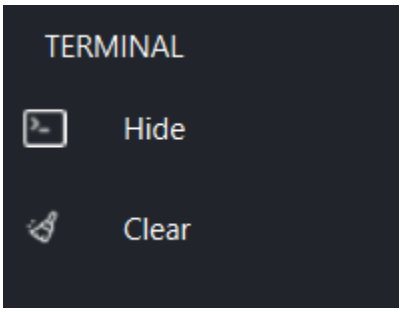


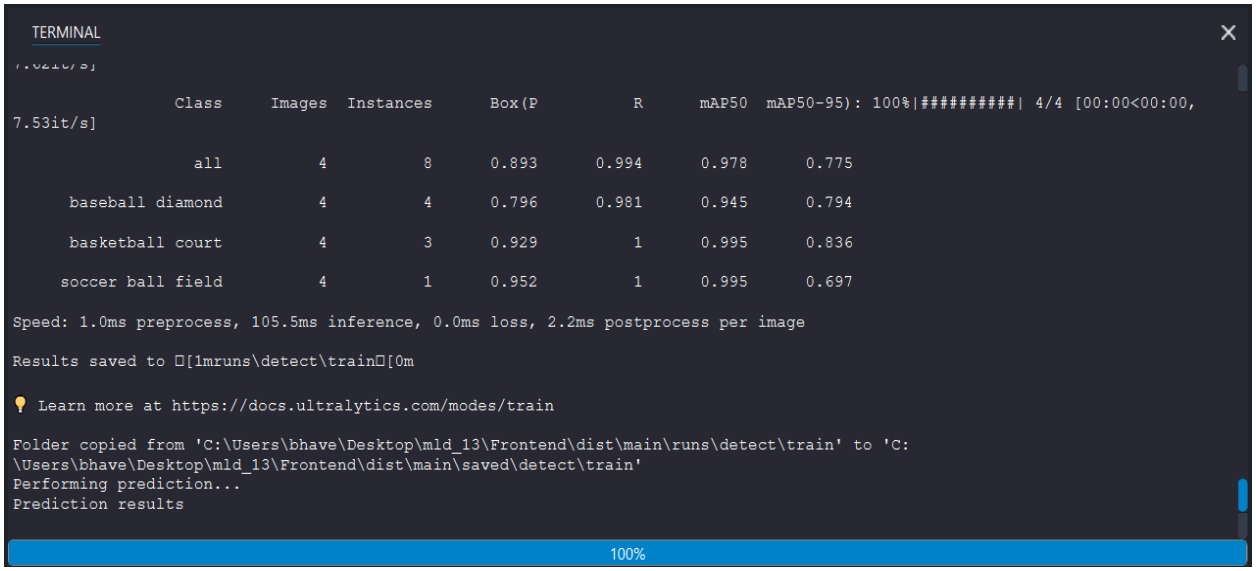
2.2.1 Sidebar after training complete and Saved Project List



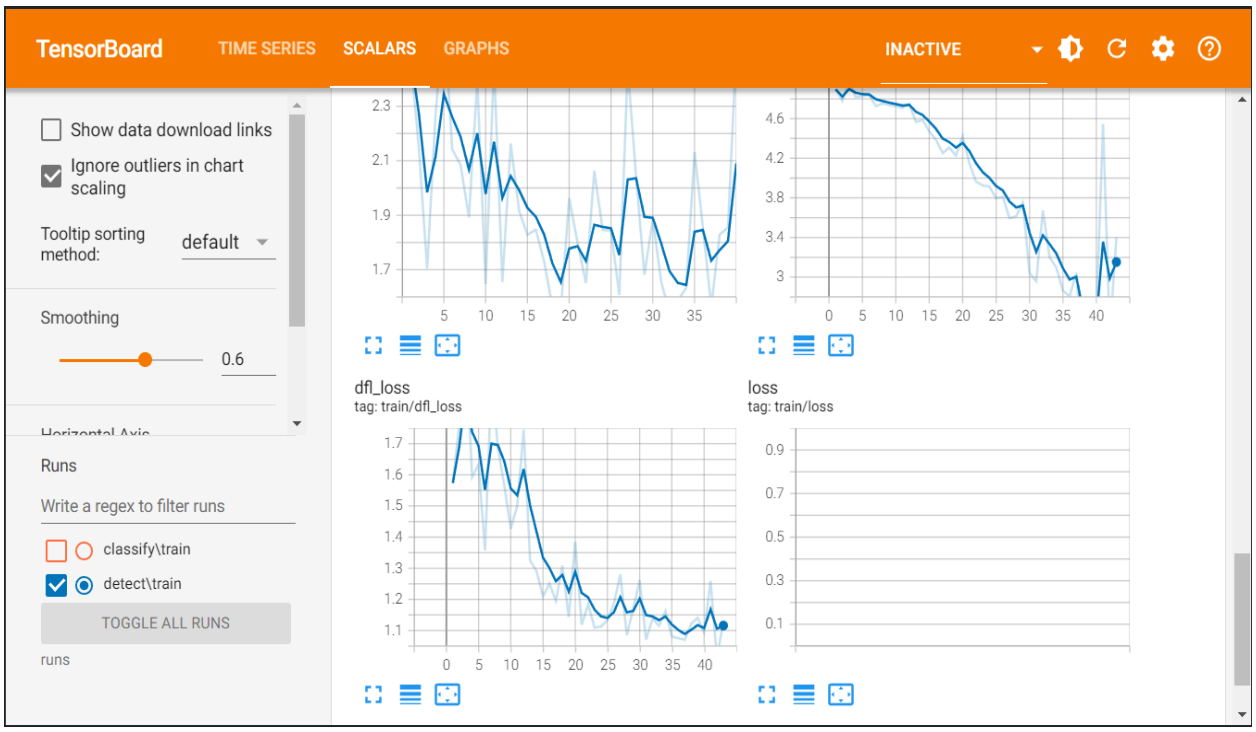


2.3 Terminal and Progress Bar

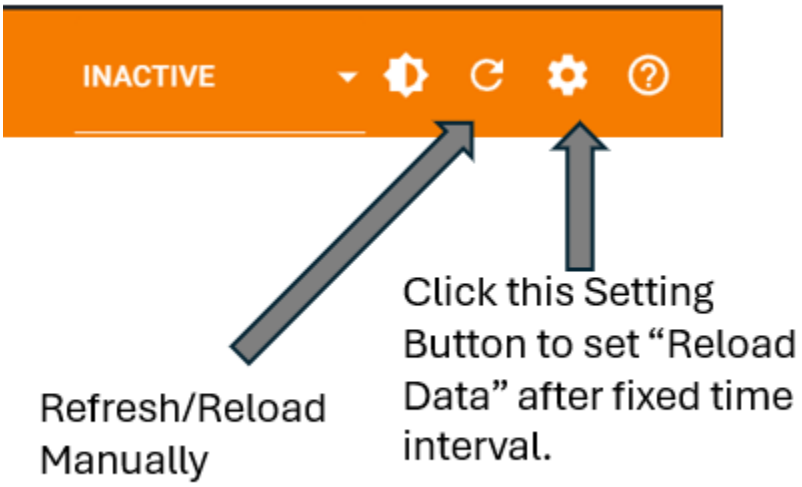




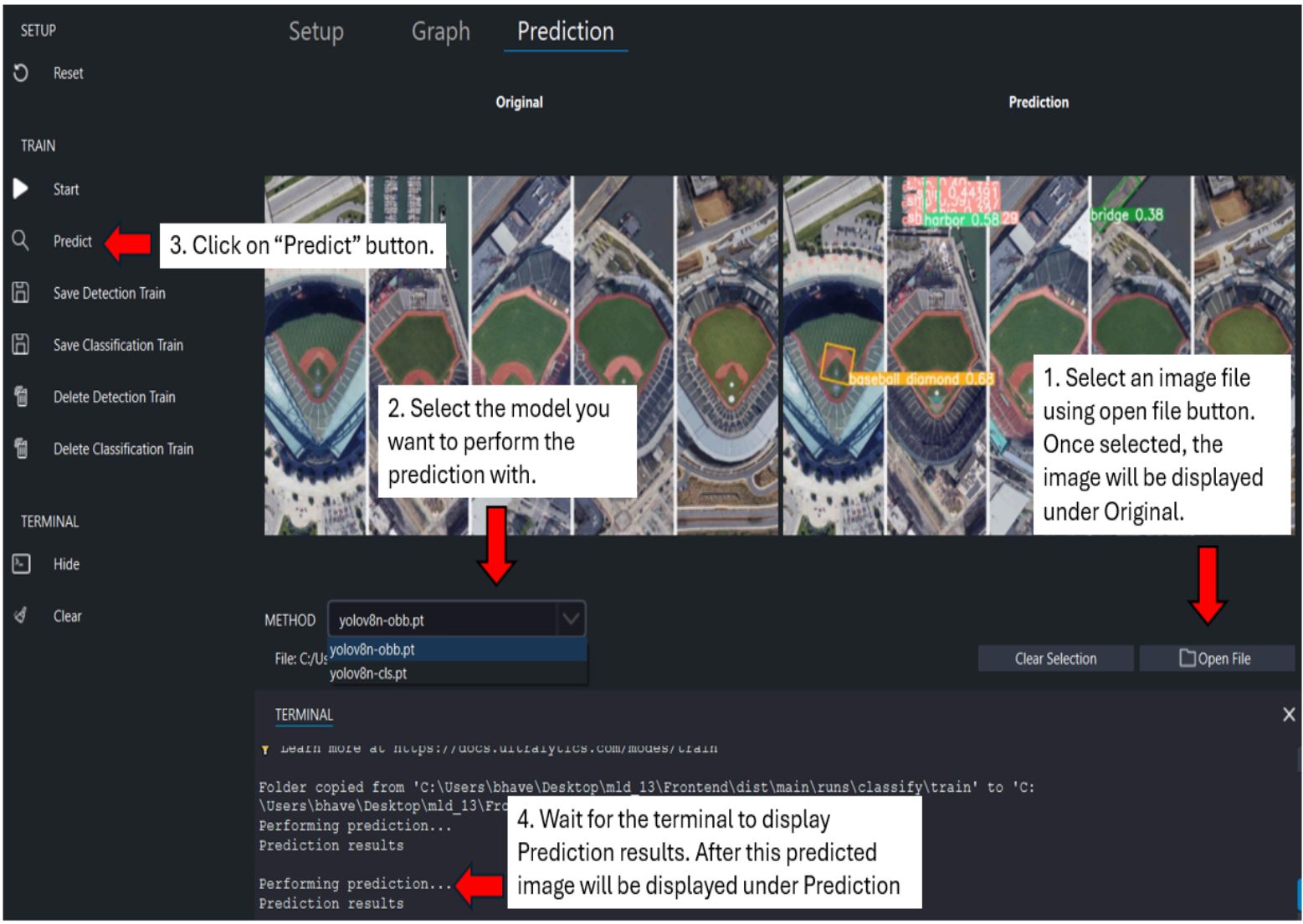
2.4 Graph Page



2.4.1 Graph Settings



2.5 Prediction Page



3 Work Done

3.1 Jay Tiwari

- Setting up backend
- Integration of all models and a couple of the datasets used in the application.
- Setting up requirement notes during client meetings.

3.2 Joao Lira

- Logic for training and predicting with Yolo models and five datasets.
- Multi-threading the application.
- Adding UI options of specifications for training.
- Displaying detailed training information on the terminal with a progress bar.
- Implemented a prediction pane for displaying prediction results.

3.3 Bao Nguyen

- Design overall main window UI
- Design setup page UI and prediction page UI and terminal UI
- Implement support features design
- Implement printing feature for every operation
- Implement training progress bar, implement page changing menu
- Design left side toolbox UI, and saved list UI
- Implemented save/delete feature,
- Architect project code structure.

3.4 Bhavesh Dewan

- Setting up Splash Screen
- Frontend development
- Fixing terminal output
- Packaging of software
- User guide document

- Parameter functionality
- Code organization
- Tensorboard integration
- Design graph page UI
- Senior design website
- Communicating with the client through emails.

3.5 Chad Monmany

- Sprint planning
- Backend development
- Model and datasets
- Research and experimentation
- Tensorboard integration
- Building and packaging software,
- Testing software package and debugging
- Documentation.

4 Results Achieved

- Completion of all requirements
- MODLBOX Software Application

Appendix

MODLBOX

Group 13

Bao, Bhavesh, Chad, Jay, Joao

Overview:

- What is Deep Learning and Machine Learning?
- Objective
- Requirements
  - Stage 1
    - Frontend
    - Backend
  - Stage 2
- Challenges during development
- Planning and Timeline

What is Machine Learning?

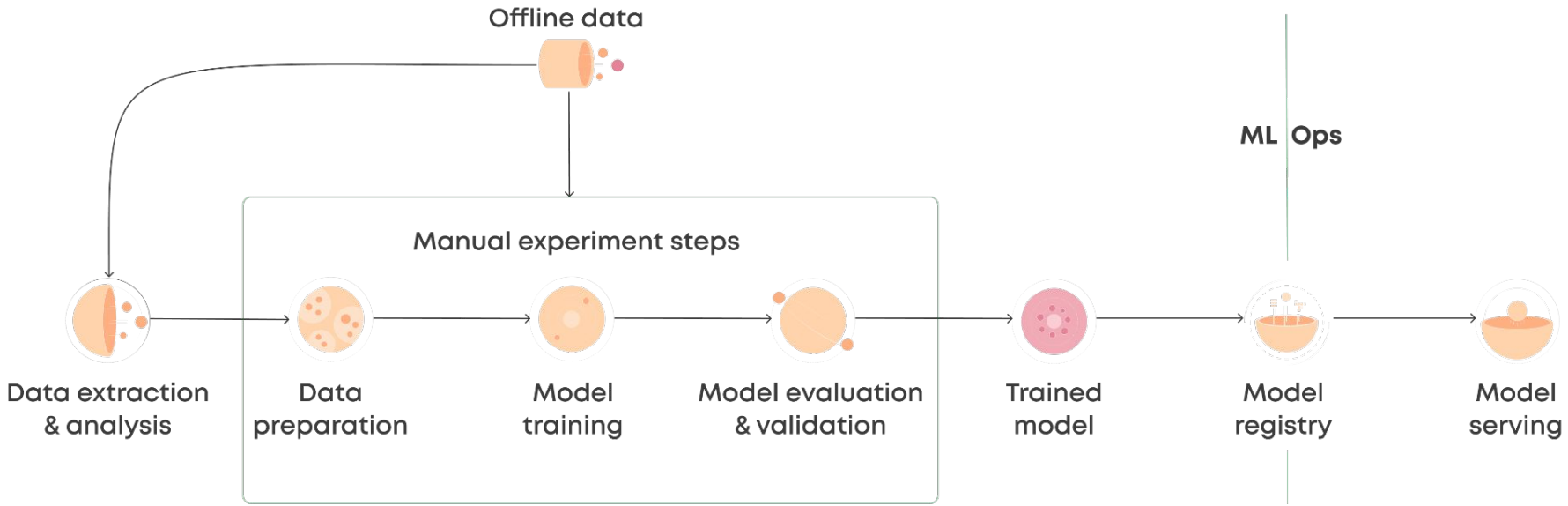
- Branch of Artificial Intelligence.
- Gives computers the ability to learn without being programmed by going using data to make predictions.

- Common algorithms to achieve these prediction models:
  - Neural Networks: Simulating how the human brain works to recognize patterns.
  - Decision Trees: branching sequence of linked decisions that can be represented with a tree diagram. Used for regression and classifying data.
- Common Applications: Chatbots, recommendation algorithms, image analysis and object detection.

What is Deep Learning?

- Subset of machine learning.
- Involves mainly training of artificial neural networks with large datasets.
- Consists of many layers of interconnected nodes to optimize prediction and categorizations.
- Layers include:
  - Visible layers: input and output layers
  - Hidden layers: Layers between the visible layers in order to further optimize and refine the accuracy of the output.
- To correct and find errors in predictions, use backpropagation which is moving backwards through the layers to train the model.

The Machine learning pipeline



Objective:

DEVELOP A SOFTWARE FOR SPEEDING UP THE ML/DL DEVELOPMENT LIFECYCLE.

UI FOR SELECTION OF DIFFERENT MODELS.

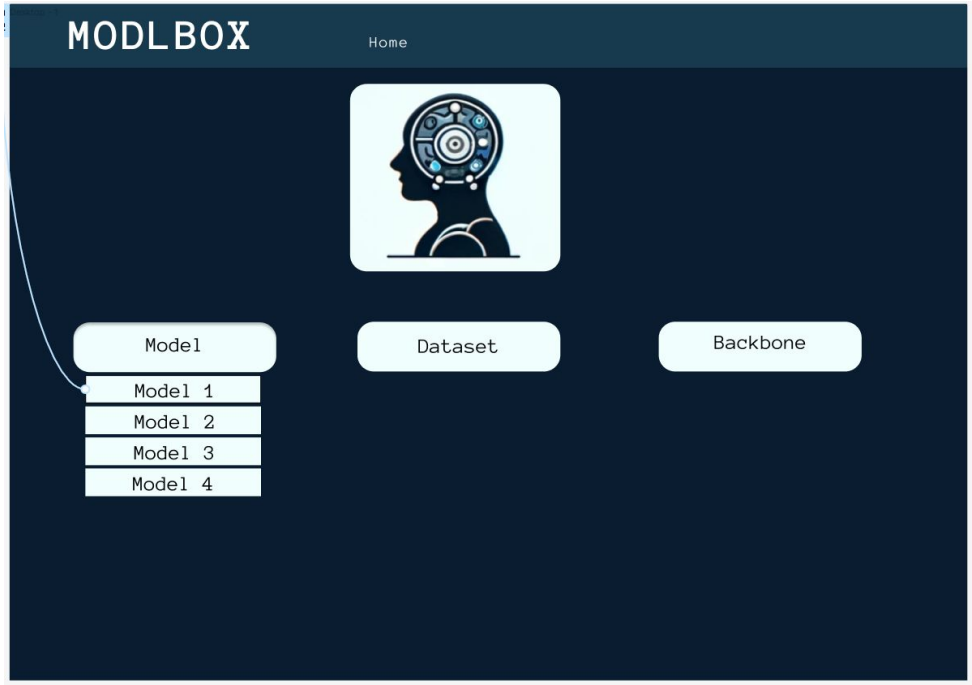
Requirements (Stage 1):

- Option to select between classification (labeling an image) and object detection (identifying an object within an image).
- If detection, then he can choose option to select 1 stage (as in Yolo, direct 1-phase but may be less precise) and 2 stage detectors (as in R-CNN, 2-phases and but more precise).
- Dataset Selector(Standard dataset), different dataset selector for classification and detection.
- Option to select different models. I.e CNN, RNN, DAE, LSTM
- Option to select GPU (None, 1, 2, 4, 8, 16, all). In order to review configurations effectiveness with and without GPU constraints
- Option to set parameters, i.e as in flags in Linux commands, in DL, may be number of Epochs to train

UI Objective (Stage1):

- UI design will reflect the main philosophy and objective of the software
- It will be intuitive and accessible for the end user who is not familiar with Machine Learning as a whole
- While not compromising on features for customizability and testing

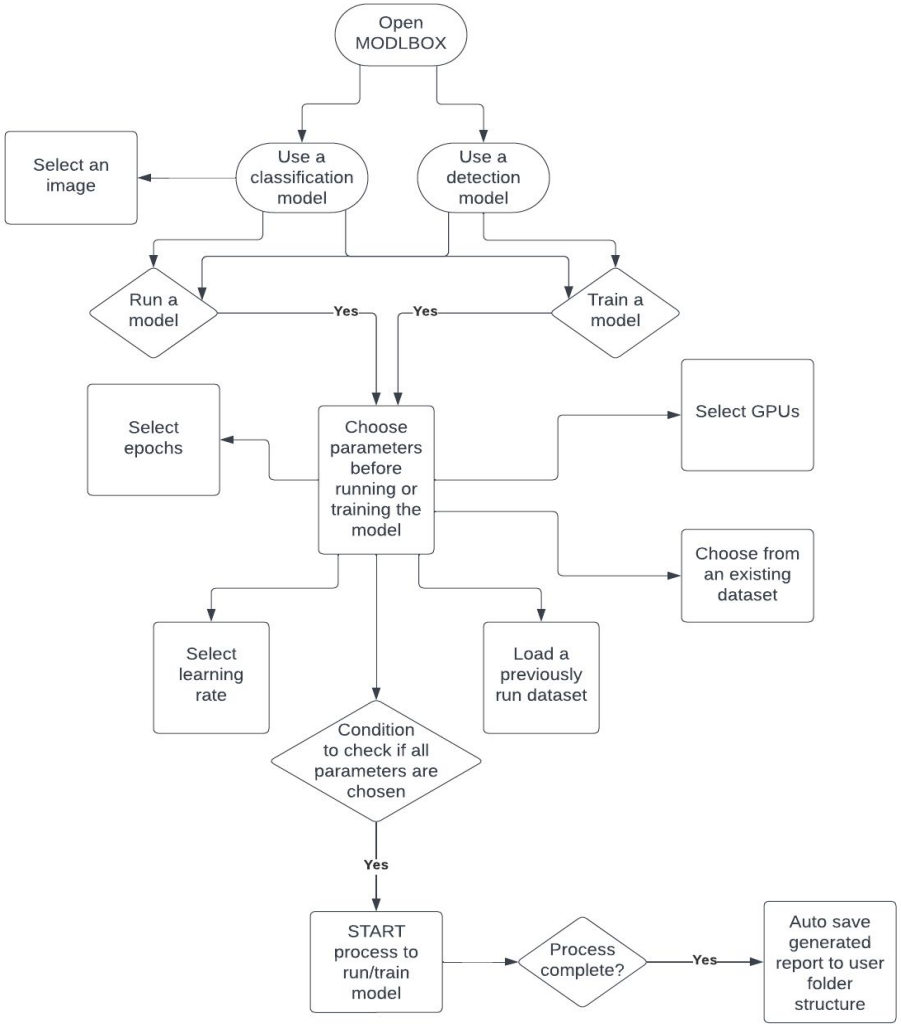
Screen Sketch



Backend components

- Once the user has selected and set all their flags in the UI, this is where the **magic** happens.
- Choosing a **modular design framewor**k for the backend component is essential since setting flags and parameters happens in layers and allowing this flexibility is necessary for future users and developers.
- The i**nsights that we get from running and training the models are generated here** to eventually save it locally on the user's computer.
- **PyTorch**: An open-source ML library based on the Torch framework, known for being exceptionally fast at executing large-scale ML models.
- Key features:
  - Save models periodically
  - Prevent **overfitting** by setting correct parameters and choosing the right dataset

Control-flow diagram



Additional Requirements (Stage 2):

Option to integrate custom datasets.

The software should automatically create/arrange the folder structure as per the model.

Ability to change the backbones.

Ability to generate graphs.

Ability to be invoked from Jupyter notebook

Additional Requirements (Stage 2):

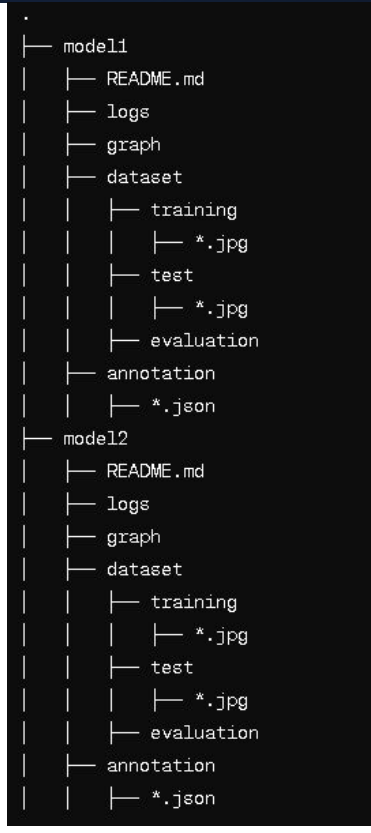
Option to integrate
custom datasets.

- Alongside enabling users to utilize standard datasets such as COCO, ImageNet, LUIS, the software should include an option allowing users to provide custom datasets.
- A valid custom dataset consists of a collection of data files (images, numerical data) and annotation files, with the quantity potentially unlimited.
- Currently, we expect that the input data provided is entirely valid.

Additional Requirements (Stage 2):

The software should automatically create/arrange the folder structure as per the model.

- There are clear naming conventions as well as default organizational structures to adhere to.
- User-provided datasets can be divided into training data and test data for performance evaluation.



Additional Requirements (Stage 2):

Ability to change the backbones.

- The software will provide suitable backbones after the user selects a model.
- For example, when the Yolo model is chosen, the software will list the compatible backbones for the provided data: ResNet-50, ResNet-18, and so on.
- The software is required to allow for changing the backbone structure mid-training. However, this may result in the training process being reset, and the outcome of the learning process may also differ after changing the backbone.

Additional Requirements (Stage 2):

Ability to generate graphs.

- The two main graphs are the loss graph and the accuracy graph.
- The software utilizes tools provided by TensorBoard to track and visualize metrics such as loss and accuracy in real time.
- The software connects to the TensorBoard server and initializes a writer object using the `SummaryWriter()` function. As a result, whenever users need, they can press the request button on the UI to display the graph.
- Since TensorBoard cannot capture graphs, the software also needs to have a function to save it into the model's graph folder.

Development Tools:

Pyside6

- Allows to create cross-platform desktop applications
- enables development of GUIs using Python
- offers seamless integration with other Python libraries and frameworks
- compatible with various operating systems, including Windows, macOS, and Linux.

Development Tools:

Pytorch

- open-source machine learning framework .
- supports dynamic computation graphs, allowing for easier debugging and more natural coding style.

Development Tools:

Tensorboard

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph
- Viewing histograms of weights, biases, or other tensors as they change over time

TensorBoard SCALARS IMAGES AUDIO GRAPHS DISTRIBUTIONS HISTOGRAMS EMBEDDINGS TEXT

Write a regex to create a tag group

Show data download links

Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing

0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

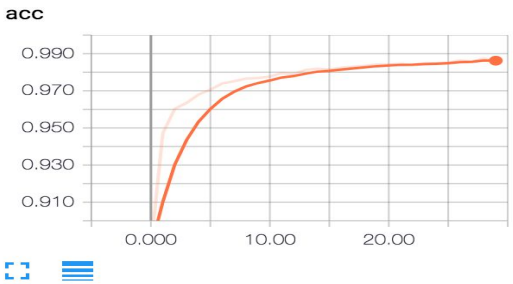
Write a regex to filter runs

logs/run_a

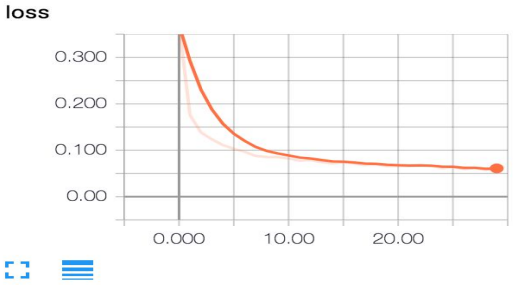
TOGGLE ALL RUNS

/Users/jjallaire/packages/keras/vignettes/examples

acc 1



loss 1



val_acc 1

val_loss 1

Timeline (Current Progress)

DEMO 1

- Draw screen sketches and control flow of the application.
- Meet with the client to gather additional requirements.
- Set up development environment/tools.
- Learn about the development tools being used.
- Start working on Pyside6 to create UI for the frontend.

Timeline (Remaining Work)

DEMO 2

- Ability to integrate custom datasets for training machine learning models.
- Flexibility to change backbone architectures for two-stage detectors.
- Capability to generate graphs and visualize model performance metrics.
- Ability to save and load trained models from a designated folder.
- Support for handling large datasets and long training times, including mechanisms for resuming training from saved checkpoints.
- Incorporating flags or parameters in the codebase to enable configurable settings and runtime options for users.
- Integration of backend with frontend components

Challenges

- The code is implemented correctly, and all functionalities are working as expected.
- All the expectations of client are complete.
- UI is intuitive and can provide excellent UX.
- Documentation is clear enough that any person is able to perform all functionalities without needing help.

DEMO

Questions?

MODLBOX

Group 13

Bao, Bhavesh, Chad, Jay, Joao

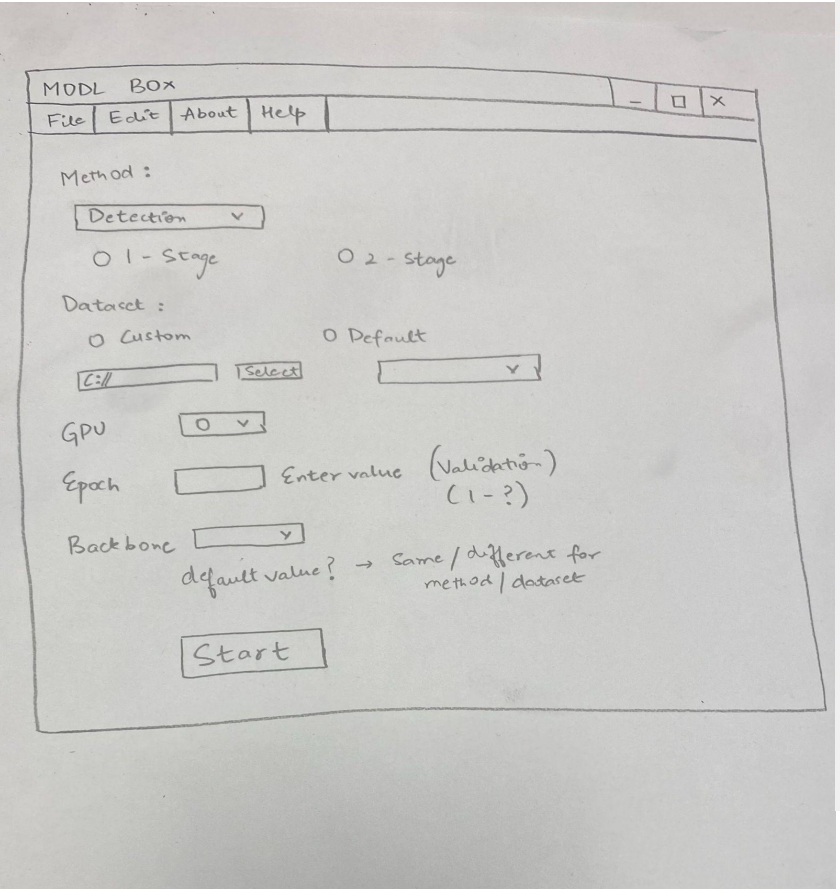
Overview:

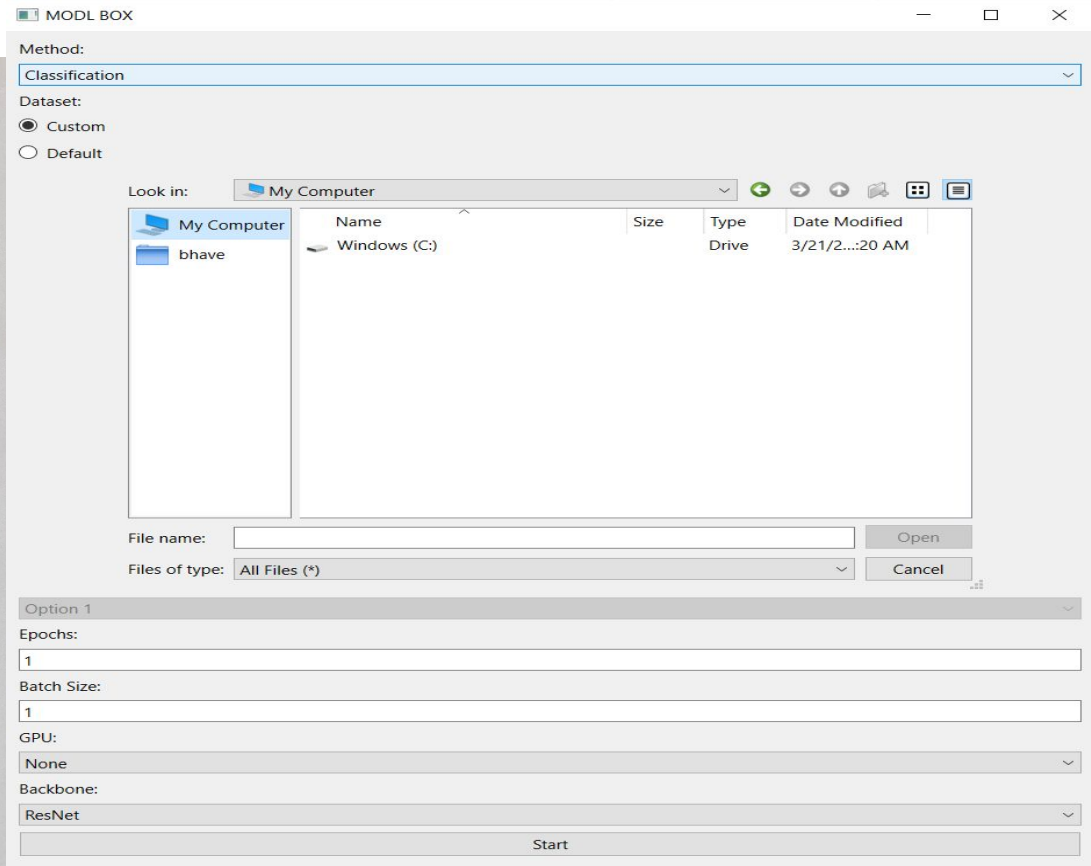
- Project Recap
- Design & Features
  - Setup Page
  - Analyze Page and its Components
  - Multithreading
  - Model Implementation
- Remaining Work Timeline
- Challenges during development
- Demo

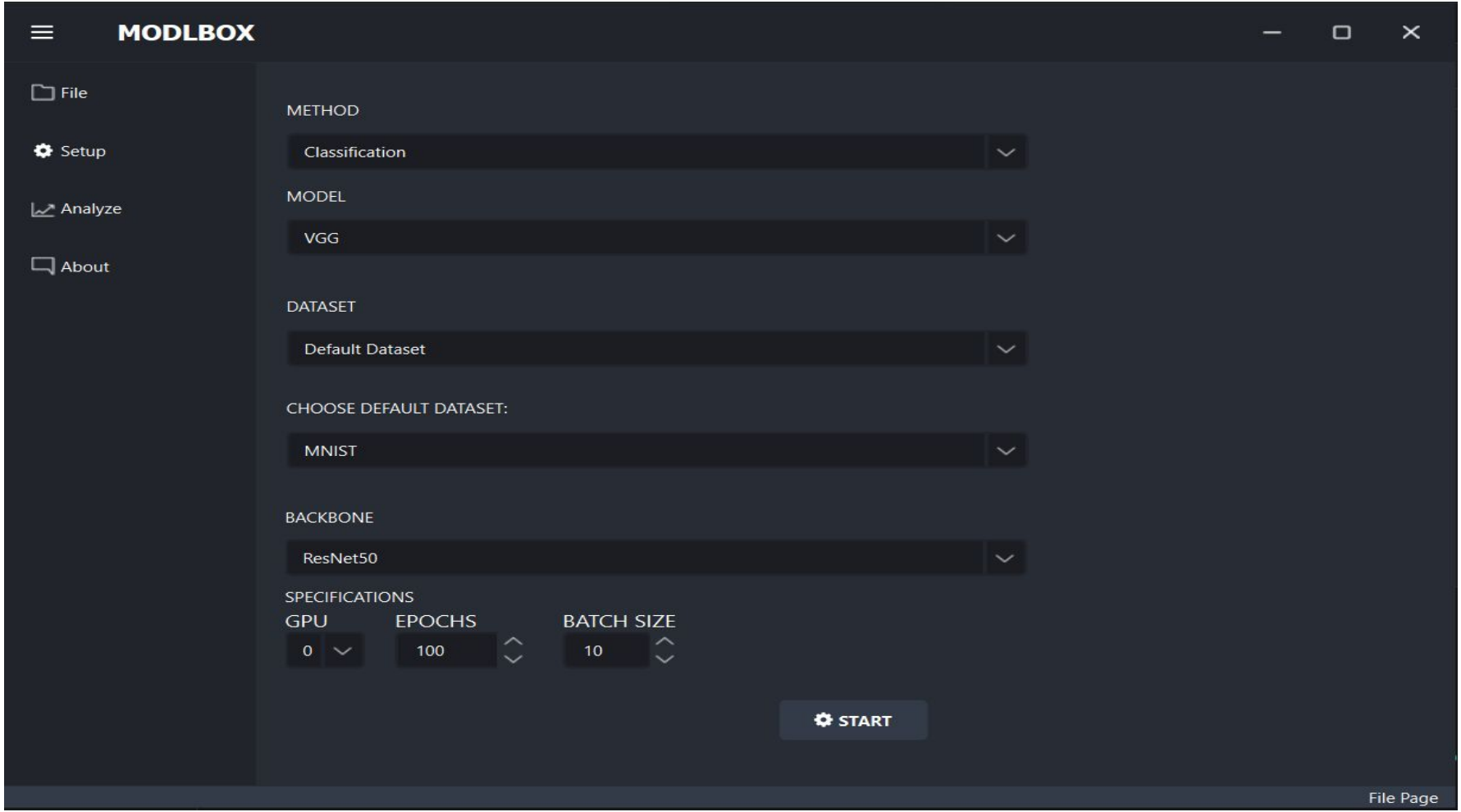
Project Recap

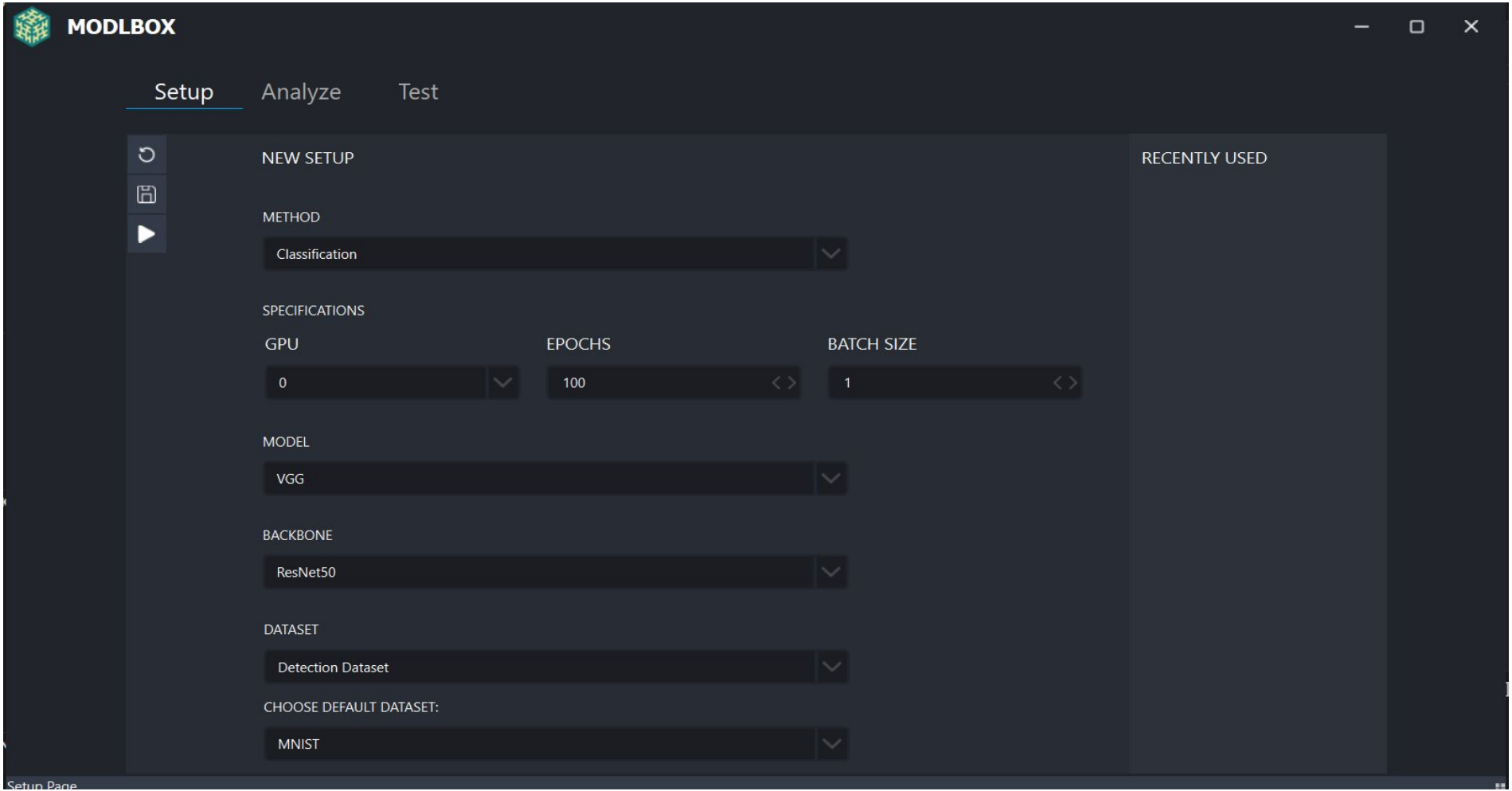
- Develop a software for speeding up the Machine learning/deep learning development lifecycle.
- Aim to provide an intuitive user interface, one can navigate through the software with ease.
- Useful for those who have limited/do not have coding experience
- Integrates various machine learning libraries, thus enhancing flexibility and performance in model training.
- Users can effortlessly select different components of the DL pipeline, including databases, backbones, and models, to create customized end-to-end models tailored to their specific requirements. train/run custom datasets.
- Goal is to reduce deployment time and minimize debugging requirements.

Frontend Design Iterations






MODLBOX
File
Setup
Analyze
About
METHOD
Classification
MODEL
VGG
DATASET
Default Dataset
CHOOSE DEFAULT DATASET:
MNIST
BACKBONE
ResNet50
SPECIFICATIONS
GPU
EPOCHS
BATCH SIZE
0
100
10
START
File Page

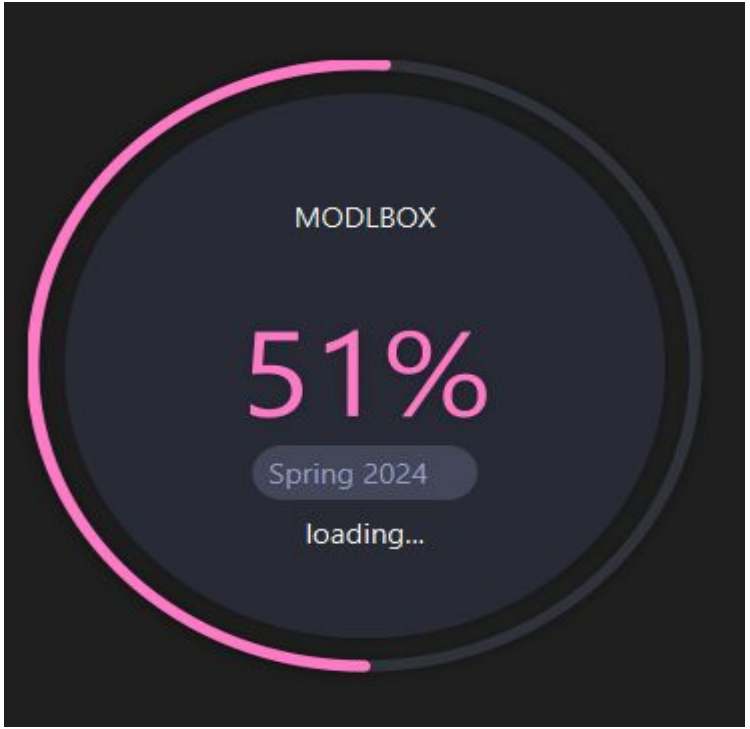

MODLBOX
Setup
Analyze
Test
NEW SETUP
RECENTLY USED
METHOD
Classification
SPECIFICATIONS
GPU
EPOCHS
BATCH SIZE
0
100
1
MODEL
VGG
BACKBONE
ResNet50
DATASET
Detection Dataset
CHOOSE DEFAULT DATASET:
MNIST
Setup Page

Setup Page Features

- Made using Qt designer and Pyside
- Functionality (buttons tied to each other)
- Default values
- Easy to understand UI/ Organized the code
- When start button is pressed, user is redirected to analyze page
- Improved UI and UX based on feedback from the client

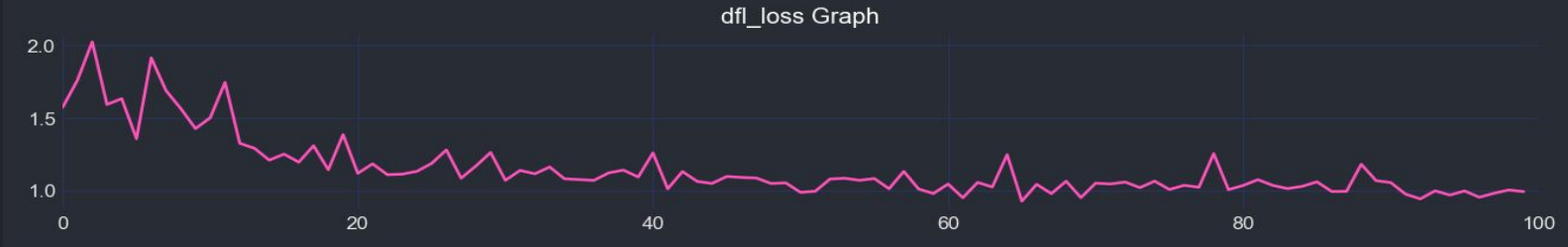
Splash Screen

- Displays when starting the software
- Placeholder for loading period
- Enhancing the UI and generating the first impression of our software
- It is also developed using Qt designer



MODLBOX

Setup Analyze Test



Terminal Frame (Temp Label)

basketball court 4 3 0 0 0 0

soccer ball field 4 1 0.0161 1 0.0474 0.0332

Speed: 2.5ms preprocess, 95.0ms inference, 0.0ms loss, 1.5ms postprocess per image

Results saved to ←[1mruns\detect\train←[0m

Learn more at https://docs.ultralytics.com/modes/train

Analyze Page

Analyze Page

- Library used-
  Matplotlib's animation (Funcanimation)
  Pandas for data handling
  mplcyberpunk for styling

- generateLossMatplot Method: This method sets up the initial state of the graph. It sets the style to 'cyberpunk', creates a Figure and Axes object, embeds the Figure into a canvas, adds the canvas to a layout, and sets the background color and title for the graph.

```
class LossMatplot():

    @staticmethod
    def generateLossMatplot(main_window):
        plt.style.use("cyberpunk")

        main_window.figure = Figure()
        main_window.axes = main_window.figure.add_subplot(111)
        main_window.canvas = FigureCanvas(main_window.figure)
        main_window.graphFrameVerticalLayout.addWidget(main_window.canvas)
        main_window.figure.set_tight_layout(True)
        main_window.figure.set_facecolor((40/255, 44/255, 52/255))
        main_window.axes.set_facecolor((40/255, 44/255, 52/255))
        main_window.axes.set_title('dfl_loss Graph', color='white')
        main_window.canvas.draw()
```

- updateLossMatPlot Method: This method updates the graph with new data. It reads data from a CSV file, initializes variables to store data and control animation, plots an initial empty line, sets the x-axis limit based on a value from a spin box widget, and sets up a FuncAnimation object to update the graph at a specified interval.

```python
@staticmethod
def updateLossMatPlot(main_window):
    main_window.df = pd.read_csv("runs/detect/train/results.csv")

    #generate data, count and limit
    main_window.x_data = []
    main_window.y_data = []
    main_window.count = 0
    main_window.limit = 0

    #draw line from data
    main_window.line, = main_window.axes.plot(main_window.x_data, main_window.y_data)
    main_window.axes.set_xlim(0,  main_window.epochs_spinBox.value())

    #generate FuncAnimation object to update graph
    main_window.ani = FuncAnimation(main_window.figure, LossMatplot.update, fargs=(main_window,)
                                    , interval=500, blit=True, cache_frame_data=False)
```

- update Method: This method is called by the FuncAnimation object to update the graph. It reads data from a CSV file, checks if the animation should stop based on a limit, updates the graph data if there is new data available, adjusts the axes limits, and redraws the canvas.

```python
@staticmethod
def update(frame, main_window):
    try:
        main_window.df = pd.read_csv("runs/detect/train/results.csv")

        if main_window.limit == 5:
            main_window.ani.event_source.stop()
            return main_window.line,

        if main_window.count < len(main_window.df):
            row = main_window.df.iloc[main_window.count]
            y_value = row.iloc[3]

            main_window.x_data.append(main_window.count)
            main_window.y_data.append(y_value)

            main_window.line.set_xdata(main_window.x_data)
            main_window.line.set_ydata(main_window.y_data)

            main_window.axes.relim()
            main_window.axes.autoscale_view()

            main_window.count += 1
            main_window.limit = 0
        else:
            main_window.limit += 1

        main_window.canvas.draw()
    except Exception as e:
        pass

    main_window.canvas.draw()

    return main_window.line,
```

Providing train-time feedback to users

- As with graph representation, we also added an extra layer of feedback in train time by leveraging the terminal output. This more low-level approach, we felt, added an improved sense of robustness to the operation and a contrast to the more high-level view of the graph
- Some of the informations that it provides is the Epoch, GPU_mem (hardware load), loss functions (prediction), instances (data processing), size (batch size)

Fig. Terminal frame

Concurrent Execution

- Deep Learning model creation (training, operating, evaluation) can take a pretty long time to complete, the user would have to wait for these operations to finish before proceeding to interact with the app, in the case where this was not addressed, it would heavily hinder the user experience. So we choose to make our application Multithreaded

```
You, 2 hours ago | 2 authors (You and others)
class yolo_thread(QThread):
    started = Signal()
    finished = Signal()
    stdout = Signal(str)
    linescount = Signal(int)

    def __init__(self, model_name, **kwargs):
        super().__init__()
        self.model_name = model_name
        self.kwargs = kwargs

    def run(self):
        # This is being USED
        print("Starting the model training process...")
            You, 2 hours ago • Uncommitted changes
        self.started.emit()
        # ------------------------------------
        #filters output

        proc = subprocess.Popen(['yolo', 'detect', 'train', 'data=dota8.yaml','model
        line = ""
        counter = 0
        while True:
            line = proc.stdout.readline()
            if not line:
                break
            counter += 1
            line = line.decode("utf-8")
            self.stdout.emit(line)
            self.linescount.emit(counter)
```

Fig. threaded code

Fig. concurrent training and UI

Model Implementation: Alexnet

Model Type: Classification

- After selecting classification model, it trains the model based on choices.
- Training per epoch generates the running loss.

```
print("Performing classification...")

# Step 2: Load and Preprocess Dataset
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Load your small dataset
trainset = torchvision.datasets.Flowers102(path_to_folder, download=True,
                                           transform=transform)
# trainset = torchvision.datasets.YourDataset(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True)

# Step 3: Import AlexNet Model
# if true then it uses the imagenet dataset, set to false b
print("Inside s1")
alexnet = models.alexnet(weights=None)

# Step 4: Modify AlexNet for Transfer Learning
num_classes = 102  # Modify this according to your dataset
print("Inside s2")
alexnet.classifier[6] = nn.Linear(4096, num_classes)
print("Inside s3")
# Step 5: Define Loss Function and Optimizer
criterion = nn.CrossEntropyLoss()
print("Inside s4")
optimizer = optim.SGD(alexnet.parameters(), lr=0.001, momentum=0.9)

# Step 6: Training Loop

epochs = epochs  # You can adjust this
for epoch in range(epochs):
    print("Inside the loop")
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()

        outputs = alexnet(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 1000 == 999:  # Print every 1000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 1000))
        running_loss = 0.0

    print('Finished Training')
```

Model Implementation: YOLOv8

Model Type: 1 stage detection

- YOLO models are part of a different package (Ultralytics) causing implementation to drastically differ from the latter.
- Output per epoch is also loss value, processed to display to terminal output.

```
proc = subprocess.Popen(['yolo', 'detect', 'train', 'data=dota8.yaml','model=yolov8n.pt', 'epochs=100','imgsz=640'],stdout=subprocess.PIPE)
line = ""
counter = 0
while True:
    line = proc.stdout.readline()
    if not line:
    | break
    counter += 1
    line = line.decode("utf-8")
    self.stdout.emit(line)
    self.linescount.emit(counter)
#the real code does filtering here
```

train/box_loss,	train/cls_loss,	train/dfl_loss,
2.4886,	4.8962,	1.5735,
2.1474,	4.7782,	1.7605,
1.7044,	4.9719,	2.0242,
2.2663,	4.8182,	1.5929,
2.647,	4.8284,	1.6329,
2.1422,	4.8394,	1.357,
2.0844,	4.7263,	1.9132,
1.8927,	4.7524,	1.6884,
2.397,	4.7369,	1.5649,
1.6473,	4.7255,	1.427,
2.4555,	4.7087,	1.5016,
1.6551,	4.7494,	1.7449,
2.1631,	4.567,	1.3255,
1.9167,	4.5862,	1.2906,
1.8288,	4.4732,	1.2092,
1.846,	4.3879,	1.2508,
1.7308,	4.2544,	1.1963,
1.5651,	4.3134,	1.3095,
1.5538,	4.2258,	1.1438,
1.9627,	4.4265,	1.384,
1.7976,	4.1497,	1.1181,
1.6588,	3.9643,	1.1888,
2.0586,	3.9248,	1.1068,
1.8451,	3.911,	1.1119,
1.8415,	3.7974,	1.1248,
1.6107,	3.8084,	1.1871,
2.4386,	3.5992,	1.2747,
2.045,	3.6166,	1.0848,
1.6835,	3.7445,	1.1675,
1.9153,	3.0316,	1.2629,

Remaining Work

- Set up virtual environment
- Package software with correct project structure
- Create custom dataset functionality
- Add ability to save trained models
- Continue documentation
- Create user guide

Challenges

- Setting up an unrequired initial API
- Changing design from OO to functionally designed code
- Concurrency and threading was unexpected
- Implementing the CONDA environment
- Concurrent generation of graphs and fetching terminal data
- Training our models (limited by hardware resources)
- Testing
- Setting up a design for ease of future implementation of additional models and features

DEMO

https://iowastate.instructuremedia.com/embed/1d0ccc71-ed6b-4ec2-89fb-51dcb0280042

Questions?

MODLBOX

Group 13

Bao, Bhavesh, Chad, Jay, Joao

MODLBOX

- MODLBOX is an application to speed up the Machine learning/deep learning development lifecycle.
- Aim to provide an intuitive user interface, one can navigate through the software with ease.
- Useful for those who have limited/do not have coding experience
- Integrates various machine learning libraries, thus enhancing flexibility and performance in model training.
- Users can effortlessly select different components of the DL pipeline, including databases, and models, to create end-to-end models.

Overview

- Features from previous demo:
  - Splash screen
  - Collecting user inputs for training model
  - Preliminary terminal
  - Thread and subprocess
  - YOLO model

Overview

- New Features:
  - New UI
  - Tensorboard for analyzing training process
  - Upgraded terminal
  - New model options (Alexnet, other YOLO versions)
  - New dataset options (e.g. VOC, Dota8, ImageWoof320)
  - Save feature
  - Predict after train feature

Requirements

- Creating/Arranging the folder structure as per the model.
- Ability to generate graphs.
- Integrating terminal in the application.
- Option to select between classification (labeling an image) and object detection (identifying an object within an image).
- Dataset Selector(Standard dataset), different dataset selector for classification and detection.
- Option to select different models. I.e Alexnet, Yolo V8, Yolo V5 etc.
- Option to select GPU. In order to review configurations effectiveness with and without GPU constraints
- Option to set parameters, i.e number of Epochs to train, Batch size.

Softwares Used- Bhavesh

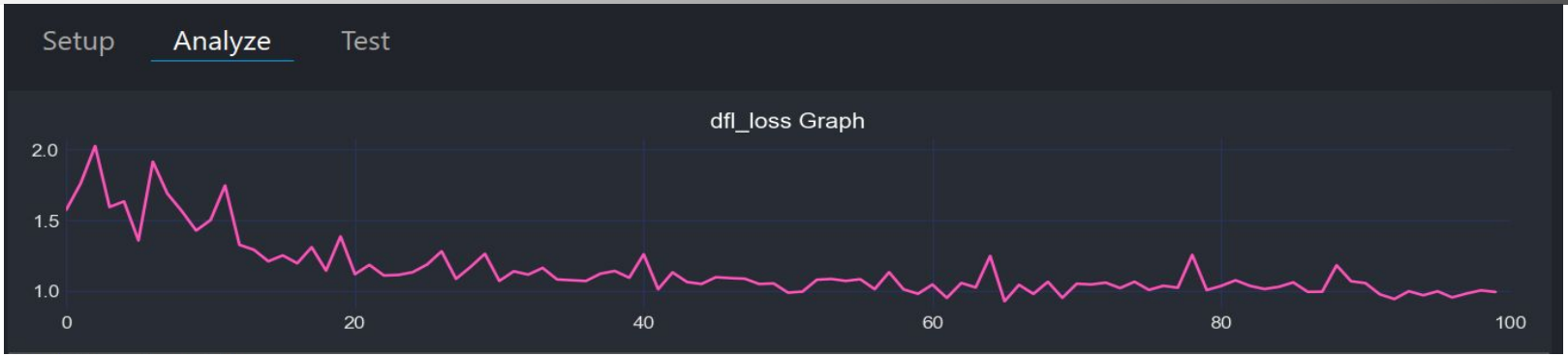
PySide6

PyTorch

TensorBoard

ultralytics

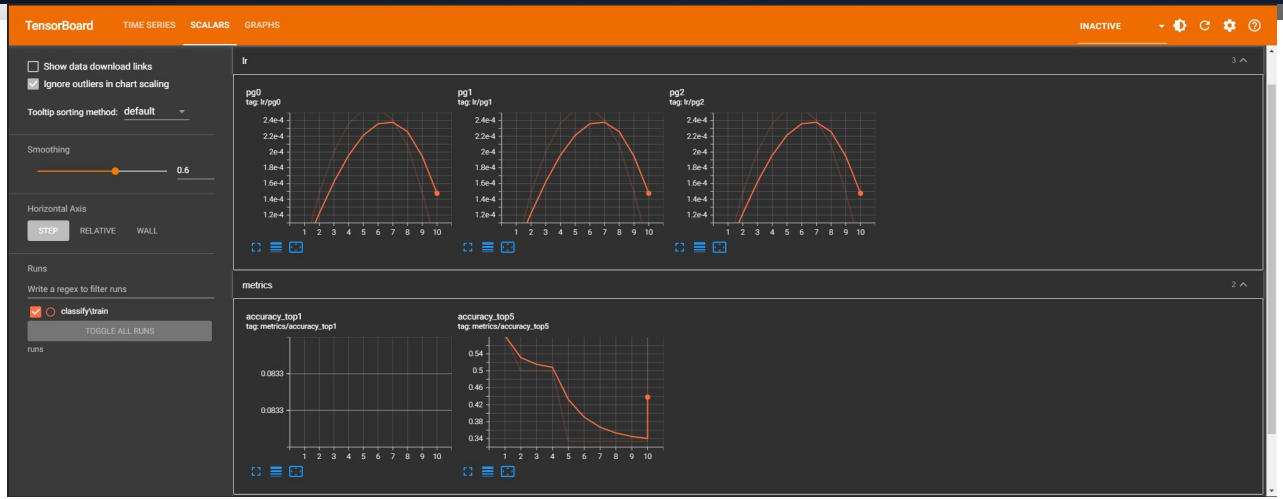
Graph Page



Problems:

- Only one graph being generated
- Graph being generated after the completion of model training
- Initial values on x-axis and y-axis were shown incorrectly
- Not able to download the graphs

Graph Page



Solution:

- Implementation of Tensorboard embedded in our application
- Allows for real time generation of graphs
- Correctly labels axes of the graph
- Added the ability to download the different graph metrics into .PNGs

Packaging

- Created python virtual environment (.venv) to keep only packages relevant to the project
- Used pyinstaller to package the application based on our specifications i.e. with/without terminal, add windows icon, add hidden dependencies, etc.
- Final executable ends up being ~2GBs zipped.

Roadblocks and Challenges faced:

- Packaging process initially was still done mostly manually as some dependencies were not found through pyinstaller packaging.
- Due to large size of dependencies i.e. Pytorch, the application and packaging process on our devices tend to take up to 10-20 minutes.
- Application still needs outside installation (need verify) i.e. needs Tensorboard and python installed on the users device.

Interface

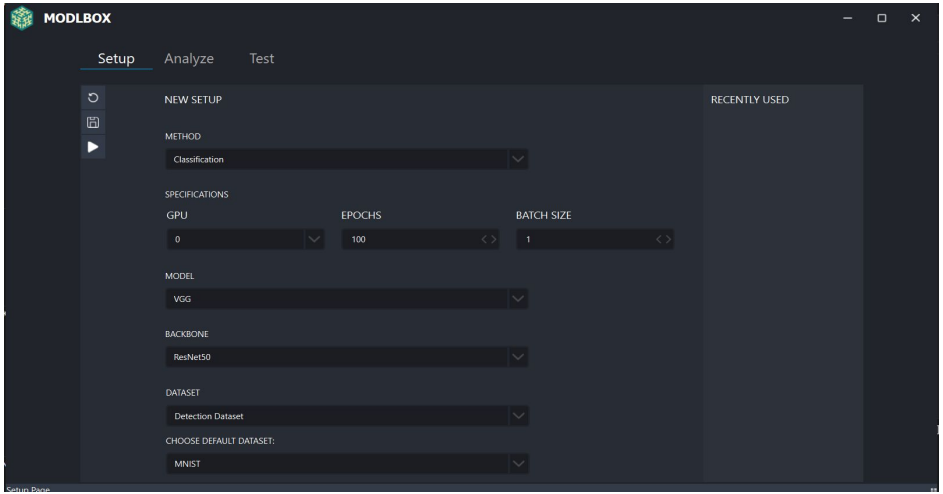


Figure 1. Previous user interface

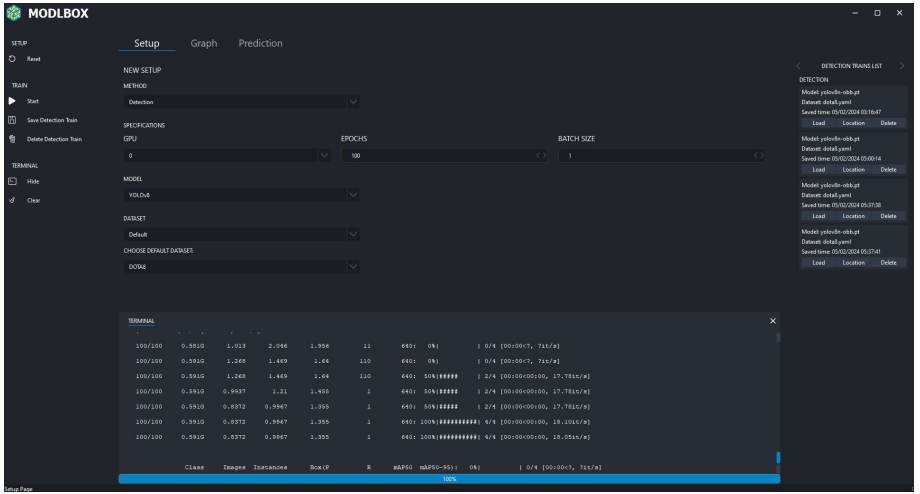
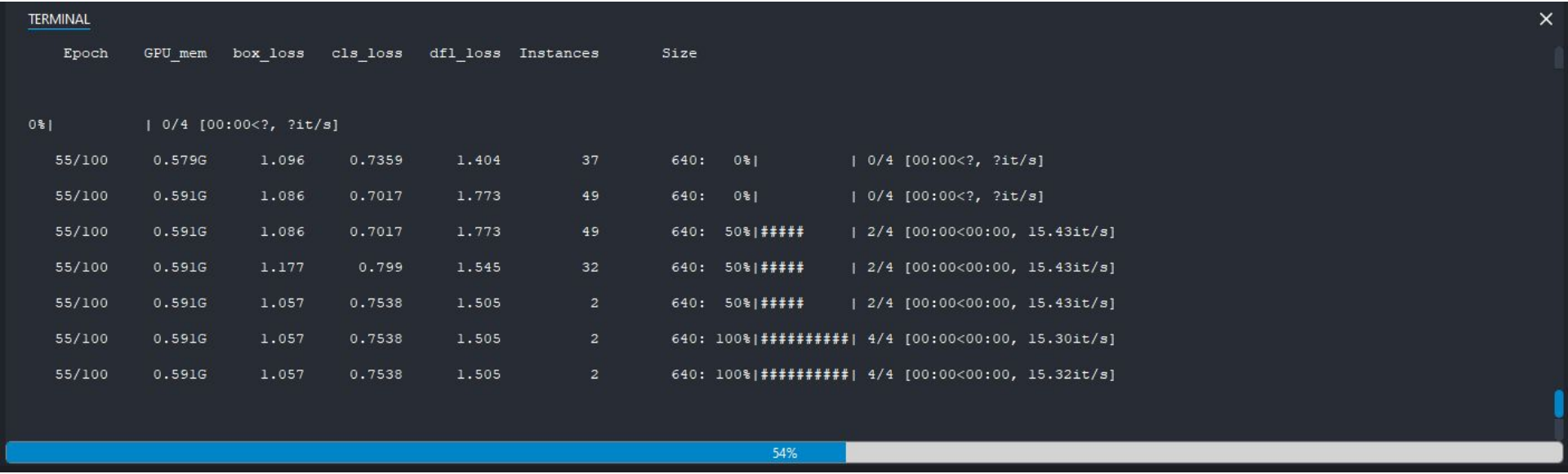


Figure 2. Current user interface

Terminal

- Terminal shows internal training statistics (e.g. epoch, loss, GPU usage)
- Embedded progress bar shows overall progress of the training
- Terminal pane is resizable and minimizable (Hide & Show Terminal)



Training on new datasets - Joao

- ImageWoof320 is a classification dataset, with categories for dog breeds.
- DOTA8: Image segmentation dataset, with satellite imagery. Bounding boxes are rotation-invariant. Useful in aerial recognition for military and civil use cases.

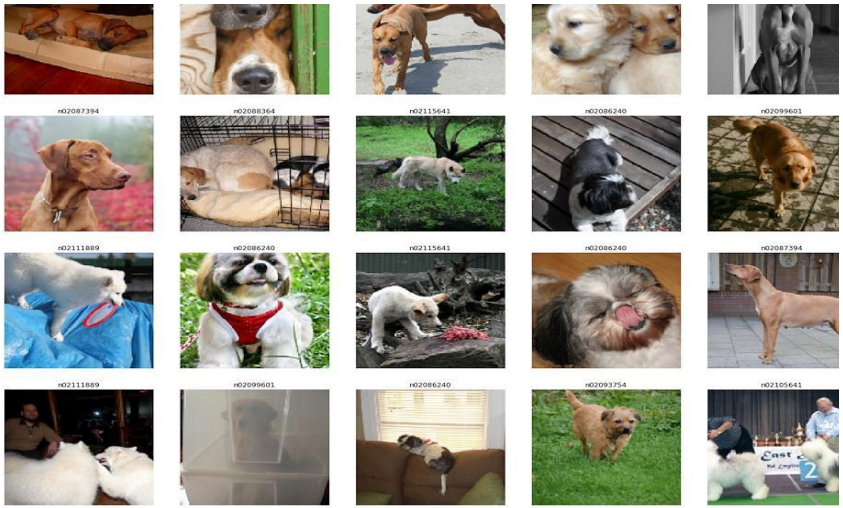


Figure 1. ImageWoof320 dataset

Figure 2. Dota8 dataset

Training on new datasets - Joao

- VisDrone is a detection dataset It contains carefully annotated ground truth data for various computer vision tasks related to drone-based image and video analysis.
- PASCAL VOC is a well-known object detection, segmentation, and classification dataset. It is designed to encourage research on a wide variety of object categories and is commonly used for benchmarking

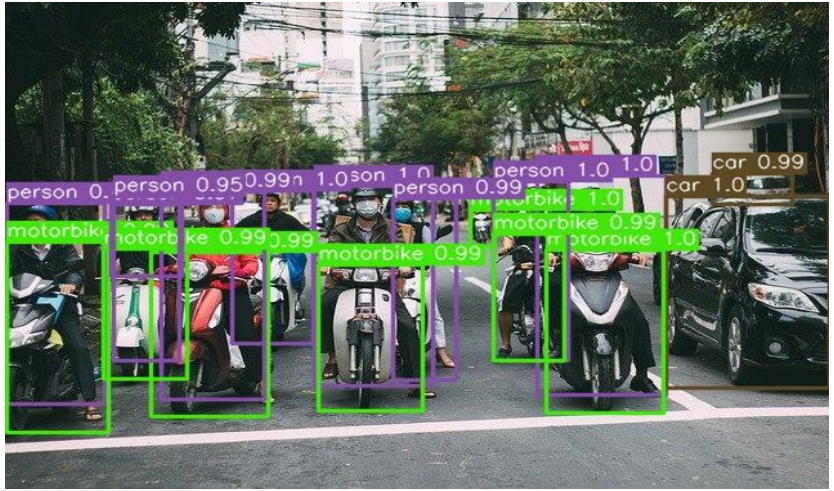


Figure 1. PASCAL VOC dataset

Figure 2. Visdrone dataset

Prediction pane - Joao

- Left pane is the original image
- Right pane is the inference output
- User can also switch between available models.
- User can upload image from the device.
- Button for prediction is available when users have model trained and image uploaded.

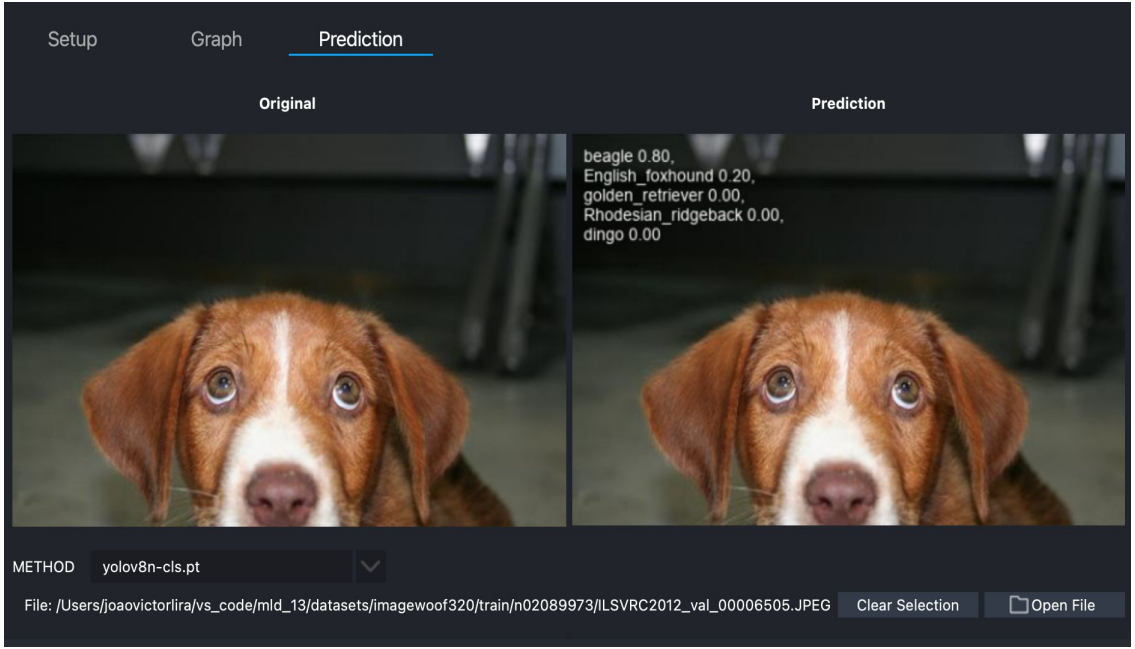


Figure 1. Prediction pane

Prediction Interface - Bao

- Left pane is the original image
- Right pane is the inference output
- User can also switch between available models.
- User can upload image from the device.
- Button for prediction is available when users have model trained and image uploaded.

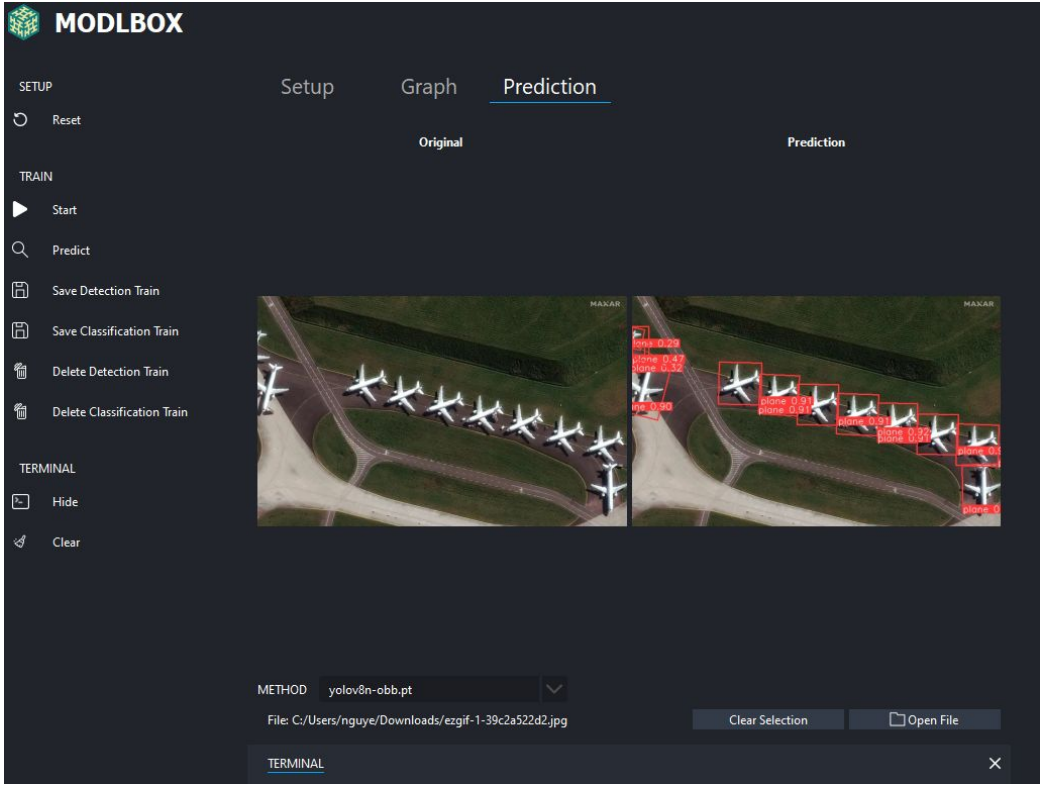


Figure 1. Prediction pane

Saving

- Option for saving output would appear after training
- Saving the output will create a new entry on the right sidebar
- Each entry holds:
  - Model used
  - Dataset used
  - Timestamp
  - Utility buttons

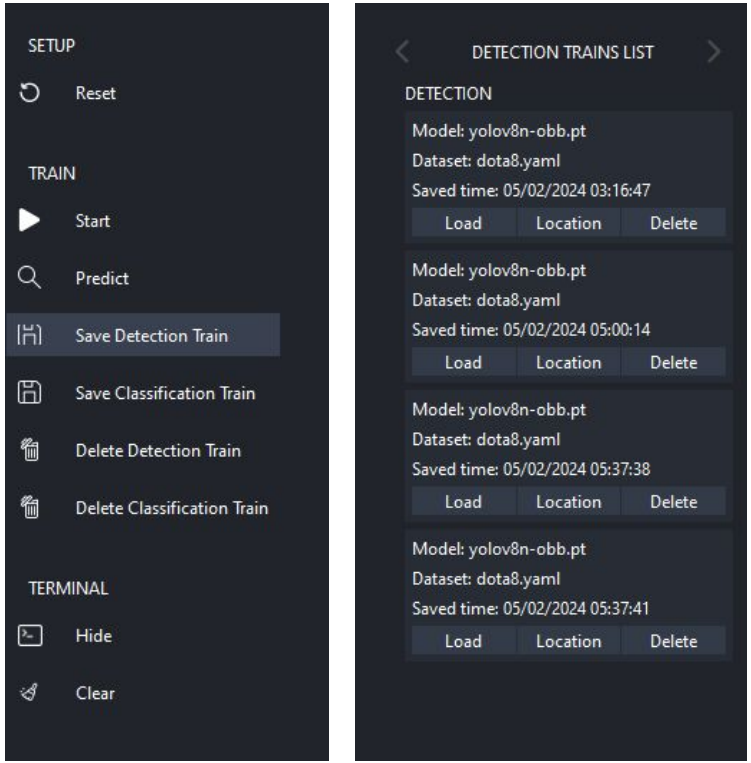


Figure 1. Sidebar Interface after training

Safety features

- When users request a new train while the previous training output remains unsaved or deleted, an information box will appear.
- The start, save and delete data buttons will be temporarily hidden during training
- If a user requests to delete a folder, a warning box appears to confirm the decision.

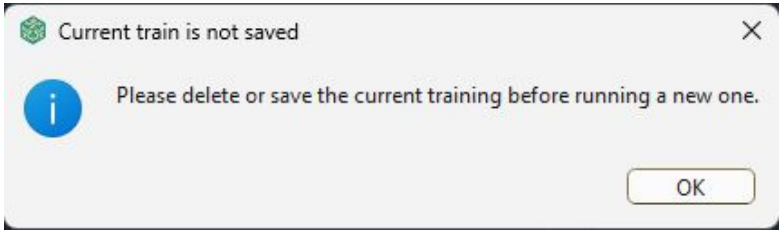


Figure 1. Information box

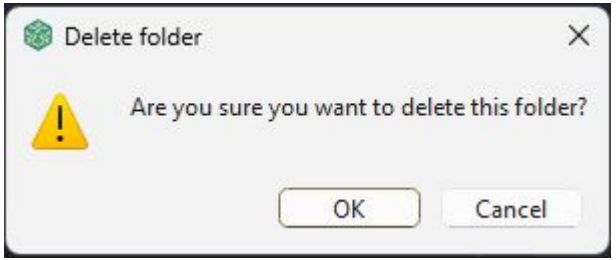


Figure 1. Warning box

Integration of Models: - Jay

- Setting up a design to allow the ease of addition for other models. (Following an AbstractFactory design)
- Displaying valuable information to the user during every run, including CPU loss, Memory loss, and total memory being used on each epoch.
- Setting up the training and prediction for the AlexNet model on multiple datasets involved understanding and implementing transformation, data loaders and normalization of data.

Challenges

- Threading continues to be a challenge during training and prediction
- Application sometimes freezes for inconsistent durations (possible relation with subprocesses and devices)
- Enormous dependencies, given the varied nature of our data sources and models
- GPU utilization requires extensive hardware and software set up (CUDA installation on every computer)
- Testing every new dataset for prediction takes a while, since models can take a while to train.
- Outdated documentation (e.g. inaccessible URLs)
- Communication problems, especially with AI/ML terminology

Contributions

- Jay Tiwari: Setting up backend, integration of all models and a couple of the datasets being used in the application. Setting up requirement notes during client meetings.

- Joao Lira: logic for training and predicting with yolo models and 5 datasets. Multi-threading the application, adding UI options of specifications for training along along with displaying detailed training information on the terminal with a progress bar. Implemented prediction pane for displaying prediction results.

- Bao Nguyen: design overall main window UI, design setup page UI, design prediction page UI and implement support features, design terminal UI and implement printing feature for every operation, implement training progress bar, implement page changing menu, design left side toolbox UI, work on saving feature, deleting feature , design saved list UI, architect project code structure.

Contributions

- Bhavesh Dewan: Setting up Splash Screen, frontend development, fixing terminal output, packaging of software, user guide document, parameter functionality, organizing code, Tensorboard integration, design graph page UI, senior design website, communicating with the client through emails.

- Chad Monmany: Sprint planning, backend development, model integration and some datasets, research and experimentation, Tensorboard integration, building and packaging, Testing software package and debugging, documentation.


DEMO

QUESTIONS?