Live Deployment of the UI is present at: https://boschtsr.ml/

To install the app locally: User Interface

Data Creation

- We started with GTSRB Dataset (`dataset.tar.gz`) as the base.
- After some literature survey, we found TSRD Dataset and took 5 new classes from it. Images from the common classes of TSRD and GTSRB were also added to the base. Thus, we obtained an augmented dataset (`datasetaug.tar.gz`)
- Several transformations were applied to Training, Validation as well as the Testing images of DatasetAug to get DatasetDiff (`datasetdiff.tar.gz`)
- Finally, to increase difficulty even more, we applies transformations with even higher probabilities and added ~4000 new transformed images as well to the Test set to get DatasetTesting (`datasettesting.tar.gz`)

Dataset Types

- `dataset.tar.gz` : Vanilla GTSRB [43 Classes]
- `datasetaug.tar.gz` : GTSRB augmented with TSRD [48 Classes]
- `datasetdiff.tar.gz` : Difficult Dataset [Transformed Test and Train Images]
- `datasettesting.tar.gz` : Very Difficult Dataset [Even more transformed Test], more transformed images added to the Test set as well

Experiments and Evaluation

This section describes our entire development process in chronological order.

[Row X] refers to the Xth row in the model statistics table given at the end.

- We start with `dataset.tar.gz` and obtain benchmark scores. [Row 19]
- After literature survey, we added 5 classes from TSRD dataset. Moreover, images that were common to both the Datasets were also added to the base dataset to get `datasetaug.tar.gz`
- UI was created with various features (see UI features and guide).

- In order to make the dataset more difficult, we transform the test images using several transforms with varying probabilities. Around 4000 images were also added with similar transforms and even higher probabilities to increase the number of augmented images.
- Now we test the benchmark model on this dataset. [Row 20]
- Clearly, the F1 score obtained is very poor. The scores obtained were visualised using the features of the UI.
- From the data obtained, there were three tyes of issues found in the misclassified images:
    - Presence of images from the extra 5 classes (out of distribution)
    - Many images were transformed (blurred, shifted, rotated, data loss, etc.)
    - Some images are fine in the sense that they can be correctly classified by a human but the model failed to perform well on those.
- This is how we draw the above conclusions using the UI:
    - We can see all the misclassified images and the metrics in the UI itself and thus, can manually identify the reasons of failure as well.
    - The UI has the ability to visualise classes (cluster graphs) and tell the user if such is the case. This helps in dealing determining the presence of images from new classes. Additionally, we try to further extend this functionality to detecting new classes based on cluster entropy.
    - Firstly, t-SNE embeddings of the new images are generated, and the existing model is tested on the new images. Clusters with high entropy are predicted as new classes with reasonable confidence. However, this approach is only intended to help the user speedup the addition of these new images into the existing dataset.
    - In order to increase interpretability of the results, the prediction values and the image itself is used to generate an anchor which is basically a mask that, if applied to any image will generate the same prediction as it did right now. If the image is classified correctly, the anchor completely captures our region of interest. In misclassified images, we find that the anchor captures irrelevant parts of the image.
    - We also generate Integrated Gradients that help us in identifying the contributions of each pixel of the image to the recommendation generated by the model.
- Once we find the reasons/types of failures, we can think of steps to counter those problems:
    - In order to deal with presence of new classes, we have 2 approaches:
        - Train on all classes from scratch [Row 1, 2 and 9]: A very obvious but un-scalable solution. As the dataset increases, the time taken per epoch will greatly increase. Training a model on the entire dataset takes a lot of time and thus, we would like a better approach for the same. This is also a very wasteful approach as all the time spent on training a model for 43 classes (the benchmark model) is wasted.
        - Incremental Learning: This is what we feel is the better option. In this case, we take the model trained on 43 classes, replace the last layer (softmax classifier with 43 weights) with

a new softmax classifier with 48 weights, having same first 43 weights. We freeze all the layers of this new model except the last 5 weights of the last layer and train it on examples from the new classes for a few epochs (1 to 2). After this, the model is trained for about 10 epochs on the train dataset consisting of all 48 classes.

    - An important thing to notice here is, the incremental learning model trained for about 26 epochs [Row 16] is able to achieve performance close to a model trained from scratch for about 83 epochs [Row 7]. The ability to build on previous knowledge is a very important requirement for scalability. [Row 14 & 17, Row 15 & 18] show that the trade-off is worth it.

  - In order to improve the model to deal with images that reflect real life scenarios, we have 2 options:

    - Change the training dataset by applying transforms to it and create a new dataset, but as one can see from `datasetdiff.tar.gz`, this approach is not that appealing.
    - Another option is to integrate the ability of applying transforms to the training images in the model itself. This is different from the previous approach as at each epoch, random transformations are applied to the entire training and validation set and as a result, the model almost never sees the same image twice as well as is able to capture a large number of image-transform permutations. The superiority of this approach is visible from [Row 1 & 4, Row 2 & 5].

- If we observe the statistics, we can conclude that:

  - Using random transforms as a part of the model itself improves the performance not only on the dataset that has transformed images but also, the Vanilla/Augmented dataset which has images that have not been transformed. Thus, we integrated it in the model itself.

Additional Experiments

This section details some additional approaches that we explored.

- Initially, we experimented with LIME to provide some interpretable functionality to the model. However, Integrated Gradients and Anchors were found to be much more informative. Thus, this is what we settled with.

- In order to extend the functionality of adding classes to try to accomodate unlabeled images, we first attempted to train out-of-distribution detectors, with the aim of trying to distinguish between traffic signs that the model had been trained on and signs that belonged to no class. Two papers were implemented:

  1. Outlier Exposure - The method tunes the mode again by minimizing a custom loss function which essentialy teaches the model the differentiating factors between the in-distribution and out-distribution. The loss function used doesn't affect the original classification much. However, here the Out distribution used is TSRD, also a traffic dataset which we think is too close to the original database. OOD works better in differentiating between say a traffic sign and a dog, not between two traffic signs. Hence, satisfactory results were not obtained.

2. ODIN - Applies pre-processing in terms of adding adversarial noise to the input and temperature scaling to softmax function. The ideal result should be poorer confidence for the OOD images and therefore a way to threshold for OOD. Its also done for calibration of model so that the model is not overconfident. The confidence for OOD increased and for in-distribution images decreased. This was the opposite of the desired output. We believe that the implementation worked more on the caibration part rather than OOD.

Since the above approaches didn't pan out, we resorted to the cluster entropy based approach. The user can create an "Extra" class - a folder to store images that have currently not been labeled confidently. Once a decent amount of images accumulate in this class, the user can choose to use the cluster visualizations for these images and determine the presence of new classes with reasonable confidence.

- As detailed in the above section, new classes are integrated with incremental learning, and most of the augmented images are handled by in-model augmentations. However, as can be seen in the UI, some misclassified images remain, of which almost all are either heavily blurred or have large proportion of pixel dropouts. The misclassified images that are not human-interpretable are not valuable to the model and hence we consider deleting these images from the dataset entirely. In order to handle the remaining mistakes and move further towards human-level performance, we considered the following approaches:

    - From a perspective of feedback learning, we tested the following approach for artificial image generation for misclassified images

        - We trained a Conditional VAE for generating images on the training data set of 43 classes. The images generated in such manner were clearly distinguishable from the realistic images, the quality was poor as suggested by high FID value. Hence, a GAN based approach was more suitable because the Discriminator does exactly the work of forcing the generator to generate high quality realistic images.

        - We used a class conditioned StyleGAN2 with Adaptive Discriminator Augmentation. The motivation for this was the non availability of a very large training corpus which GANs generally require. ADA helps the GAN to generate good quality images with lesser data. Images generated using StyleGAN2 had considerably higher quality with low FID value.

      Our idea is basically to generate similar images around a misclassified input and train our classification model on these images. This can be done using projecting the target misclassified image into the latent vector space of the Generator and then sampling latent vectors close to the projected vector. This allows us to get multiple images similar to the original misclassified images on which we can further train the classifier. This led to a individual class based increase in f1 score. This approach is independent of the classification model used and can be in a manner of feedback learning on the model.

    - To determine the reasons for misclassification, we use the anchor algorithm to generate superpixels resulting in the high probability of the wrong class. Anchors provide local explanations for a particular prediction by generating perturbations around the given image. The explanations generated by anchors were image-dependent and did not provide much insight into the network working. But these explanations provided information about the image segments, which were similar to other classes and ultimately led to misclassification.

  - To generate images close to misclassified images using the class representative, we tried to generate images using activation maximization approach. This method involves generating images that led to high probability of a particular class. This is achieved by modifying the pixel values until it has highest activation for that particular class. Using this method, we generated masks that led to high probabilities for each class. We superimposed this mask over an image from another class which was correctly classified with high confidence originally. So this method cannot be used to generate images around misclassified images, but the generated images could be used to improve the dataset to increase the model's robustness.

- Traffic Sign Detection - A pre-trained YOLO model is to be used for traffic sign detection. The car will capture images from camera feed in fixed time intervals and pass the image through the model and extract signs throughout the day. This dataset generated will be used to improve model. This additional feature has been implemented but not integrated currently in the UI. With the real-life images, we can build the dataset using the above described techniques to improve model performance.

Guide to run the UI locally

Download the required file: interiit-backend.tar.gz

The documentation assumes that the following:

- User has python3.8 installed and python scripts can be excuted by running `python3 /path/to/script/`

- The pip is of the lastest version

- CUDA is set up properly (if gpu is to be used)

Install Redis:

```
sudo apt install redis-server
```

Install MongoDB:

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-
org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
sudo apt-get update
sudo apt-get install -y mongodb-org
sudo systemctl start mongod
sudo systemctl enable mongod
```

Install python dependencies by running (in the `interiit-backend` folder):

```
sudo apt-get install python3.8-dev
python3 -m pip install -U pip
python3 -m pip install -r requirements.txt
```

Run the backend server by running (in the `interiit-backend` folder):

```
./start.sh
```

Now navigate to:

```
http://localhost:5000/
```

- To run the app, the user only needs the code present in `interiit-backend` but the code for app's frontend (`interiit-frontend`) has also been provided.

Code Structure

Directory or File Name	Description
`generated/`	Stores all the trained models and their metadata.
`static/`	Stores datasets and static files for the frontend.
`app.py`	Contains all the code for the app's backend.
`benchmark.pth`	The weights for Benchmark model.
`heatmap.py`	Functions to get Anchors and Integrated Gradients.
`model.py`	Contains model's architecture.
`segragate.py`	Contains code logic for smart segregation.
`requirements.txt`	A requirements file without any packages that depend on other packages in the file.
`start.sh`	Script to start the app.
`utility.py`	Utility functions to upload, apply transforms, get model stats, etc.

UI Features and Guide

There are following datasets available in the UI:

- **Main Dataset** : Comprises of all images from `DatasetTesting` and the images added using the UI
- **GTSRB Dataset** : Vanilla GTSRB Dataset
- **GTSRB_48 Dataset** : `DatasetAug`
- **Difficult Dataset** : `DatasetTesting`

The UI is divided into 7 segments which are as follows:

1. Explore Dataset:

- Used to visualise the entire dataset
- Can see multiple images of the classes at once, similar to File Explorer

2. Visualize Classes:

- Generates t-SNE feature embeddings for uploaded images and displays a cluster map
- Colours points on the cluster map based on predictions generated for uploaded images by a model selected by the user.

3. Add New Classes:

- Used to add new classes
- Adds new label to the set of existing ones

4. Add New Images to Dataset:

- Used to add new images to existing classes
- While adding, the user has the ability to select one or more (combine and permute) of the following augmentations, and generate more images:
  - Brightness & Contrast
  - Shift & Rotate
  - Blur (Gaussian, Median and Motion) & Optical Distortion
  - Noise (ISO, Gaussian and Multiplicative)
  - Hue, Saturation & Color Jitter
  - Dropout & Cutout
  - Affine & Perspective Transforms
- The user has the ability to either add the selected images to test set or use smart segregation to split the train set into train and validation set. Smart segregation is done using the model's output upto the second last layer as feature vector to the input images, followed by clustering and then splitting each of the cluster's into the ratio specified by the user.

- The UI allows uploading multiple images at once

5. Additional Images Added:

- Here, we can see the new images added to the dataset
- The user has an option to move images between test, train and validation sets
- The user can also edit the images (crop, rotate, etc.)
- Unwanted images can also be deleted

6. Evaluate Models

- Selecting a model displays it's training statistics
- We can select any of the datasets and run evaluation for any model trained by us
- Upto 5 evaluations can be run simultaneously, this helps in getting more info about the model in lesser time
- Once an evaluation is complete, a report is generated that presents the user with class-wise as well as overall metrics of the following kind:
    - F1-score
    - Accuracy
    - Precision
    - Recall / Sensitivity / True Positive Rate
    - Specificity / True Negative Rate
    - Positive Likelihood
    - Negative Likelihood
    - Balanced Classification Rate
    - Balance Error Rate / Half Total Error Rate
    - Matthew's Correlation
- The user is also presented with charts for these metrics to aid visualisation
- Apart from this, the user can see the misclassified images from each of the classes
- Upon clicking on any of these images, we get Integrated Gradients and also have the option to get Anchors
- These are useful to make deductions about the model and aid in improvement as mentioned in `Experiments and Evaluation`

7. Improve Model

- This pane has two features: Incremental Learning and Transfer Learning
- Incremental Learning is to be used when we increase the number of classes and want to upgrade the previous model to incorporate new data without training a new model from scratch

- Transfer Learning can be used to either replace only the classifier (`freeze weights`) or resume training of a previously trained model
- As described in `Experiments and Evaluation`, we suggest running incremental learning for 1 or 2 epoch on the Benchmark model followed by Transfer Learning for about 10 to 15 epochs to get good results in a scalable fashion
- The user also has the option to enable the use of transforms while training the model (recommended)

8. Make Prediction

- We can view the statistics of a trained model by selecting it
- Upload images to be tested and select a model to make the prediction
- We get the labels and the confidence of the model while predicting such labels

Model Architecture

Type/Stride/Pad	Filter Shape	Input Size
Conv/s1/p0	1 x 1 x 1	48 x 48
Conv/s1/p0	5 x 5 x 29	48 x 48
Pool/s2/p0	3 x 3	maxpool
Conv/s1/p0	3 x 3 x 59	22 x 22
Pool/s2/p0	3 x 3	maxpool
Conv/s1/p0	3 x 3 x 74	10 x 10
Pool/s2/p0	3 x 3	maxpool
FC/s1	1 x 300	1 x 1184
FC/s1	1 x 300	1 x 300
Softmax/s1	Classifier	1 x Number_of_Classes

Top Models

General Naming Convention: `model`_*epochs*`.pth`

Vanilla43

- Trained on the 43 classes of `dataset.tar.gz`

Base43

- Trained on first 43 classes of `datasetaug.tar.gz`

Aug43

- Trained on first 43 classes of `datasetaug.tar.gz` with random transformations applied at each epoch.

Base48

- Trained on all 48 classes of `datasetaug.tar.gz`

Aug48

- Trained on all 48 classes of `datasetaug.tar.gz` with random transformations applied at each epoch.

Diff48

- Trained on all 48 classes of `datasetdiff.tar.gz`

Inc48

- `Aug43_2` was retrained on last 5 classes of `datasetaug.tar.gz` with random transformations applied at each epoch.
- All layers except the last one were frozen, and the last layer was expanded to 48 units. Only the last 5 units were trained.

Incf48

- `Inc48_1` was retrained on all 48 classes of `datasetaug.tar.gz` with random transformations applied at each epoch.

Model Statistics

	Model Type	**Model Name**	**Train Dataset**	**Test Dataset**	**Accuracy**	**F1 Score**
1	Base48	model_136.pth	DatasetAug	DatasetDiff	93.166	0.905
2	Base48	model_136.pth	DatasetAug	DatasetAug	97.393	0.954
3	Diff48	model_136.pth	DatasetDiff	DatasetDiff	95.322	0.931
4	Aug48	model_83.pth	DatasetAug + In-model Transforms	DatasetDiff	97.118	0.957
5	Aug48	model_83.pth	DatasetAug + In-model Transforms	DatasetAug	98.701	0.976
6	Base43	model_114.pth	DatasetAug	DatasetDiff	91.033	0.812

	Model Type	Model Name	Train Dataset	Test Dataset	Accuracy	F1 Score
7	Aug48	model_83.pth	DatasetAug + In-model Transforms	DatasetTesting	91.067	0.893
8	Diff48	model_136.pth	DatasetDiff	DatasetTesting	86.679	0.840
9	Base48	model_136.pth	DatasetAug	DatasetTesting	79.628	0.757
10	Base43	model_114.pth	DatasetAug	DatasetTesting	78.229	0.685
11	Inc48	model_1.pth	DatasetAug + In-model Transforms	DatasetTesting	63.743	0.661
12	Inc48	model_1.pth	DatasetAug + In-model Transforms	DatasetAug	80.683	0.817
13	Incf48	model_10.pth	DatasetAug + In-model Transforms	DatasetAug	98.387	0.973
14	Incf48	model_10.pth	DatasetAug + In-model Transforms	DatasetTesting	87.179	0.844
15	Incf48	model_16.pth	DatasetAug + In-model Transforms	DatasetTesting	88.398	0.859
16	Incf48	model_26.pth	DatasetAug + In-model Transforms	DatasetTesting	89.179	0.870
17	Aug48	model_12.pth	DatasetAug + In-model Transforms	DatasetTesting	83.142	0.800
18	Aug48	model_32.pth	DatasetAug + In-model Transforms	DatasetTesting	88.254	0.855
19	Vanilla43	model_193.pth	Dataset	Dataset	97.736	0.958
20	Vanilla43	model_193.pth	Dataset	DatasetTesting	78.229	0.685

References

Albumentations : Library used for generating randomized augmentations

External Datasets Used:

Mapillary Dataset

TSRD Dataset

Model Interpretability:

- Alibi : Library used for Anchor Explanations and Integrated Gradients
- LIME
- Activation Maximization

Incremental Learning:

- Adding New Classes Without Access to the Original Training Data with Applications to Language Identification
- Learning without Forgetting

Out-of-distribution Detection:

- ODIN
- Outlier Exposure

Improving Performance on Misclassified Images:

- Conditional VAE : Paper, Code
- StyleGAN2-ADA : Paper, Code