AutoML

AutoML - Initial Experiments

The initial experiments on AutoML involved testing the potential of Edge Impulse EON Tuner in identifying suitable ML models for the data collected during Phase 1 of SSUP. The data included the images of tomatoes clipped to 64x64 resolution for computational requirements. Five statistical information (*mean, min, max, standard deviation and variance*), each from two color spaces viz. HSV and LAB for each channel along with the temperature and humidity information formed the input data. In fact, the dataset had 34 columns and 221 rows.

To train the models in Edge Impulse EON Tuner, the categorical column *Image_Name* had to be removed from the dataset and the 221 rows in the original dataset were split into CSV files with 1 row each. The top 3 models, respectively with test accuracies 66%, 40% and 34%, returned by the EON Tuner are shown in Figure 1, sorted by accuracy.

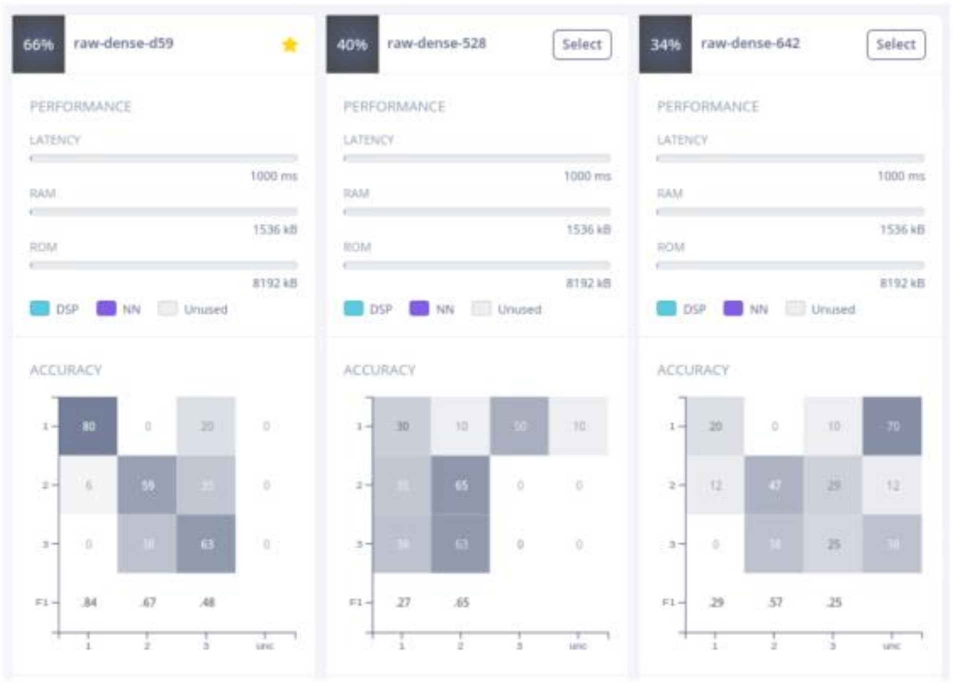


Figure 1: EON-Tuner Models for Data from Phase-1

As the test accuracies are poor, a heatmap to visualize the correlation between the data has been plotted in Figure 2. As can be seen in the figure, high correlation exists between the data. Subsequently, the correlation has been removed and RGB color channel information (*mean, min, max, standard deviation, and variance*) have also been added as part of the data (now comprising 47 columns). Figure 3 shows the EON Tuner suggested model for the augmented data. The augmented data facilitated relatively higher accuracy. The models suggested had test accuracies 74%, 68% and 62%.

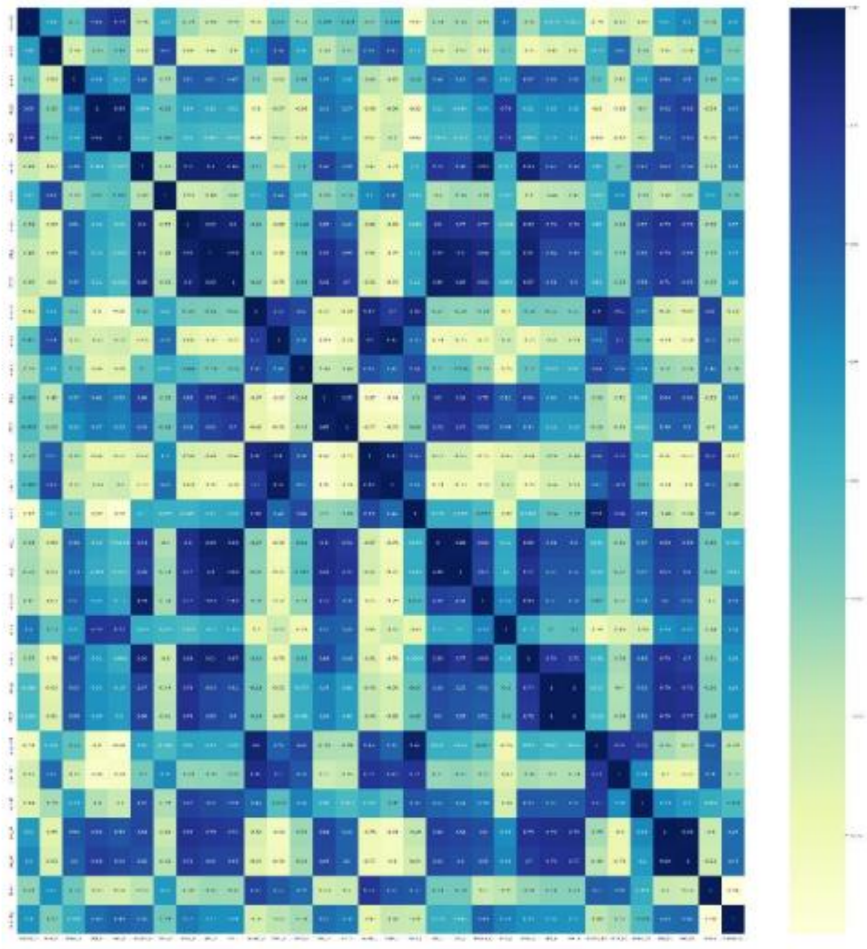


Figure 2: Heatmap for the dataset collected during Phase 1

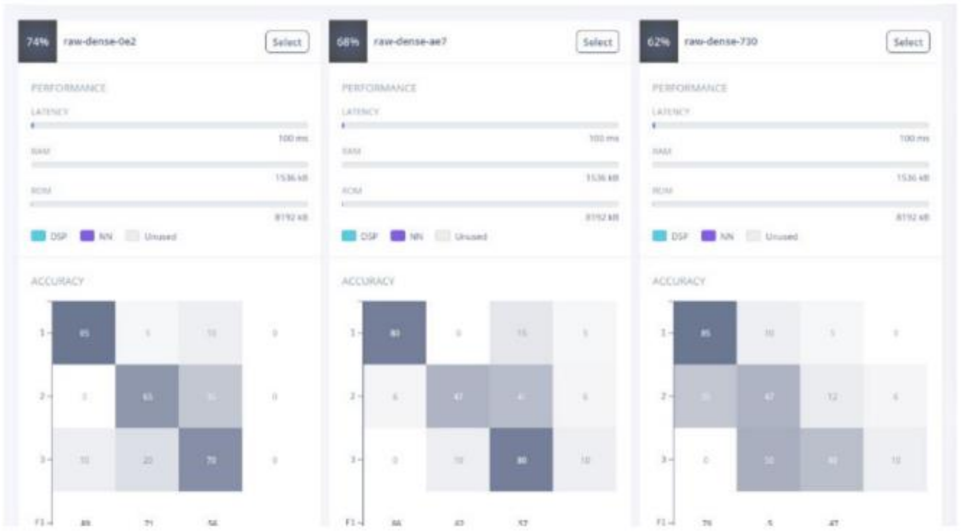
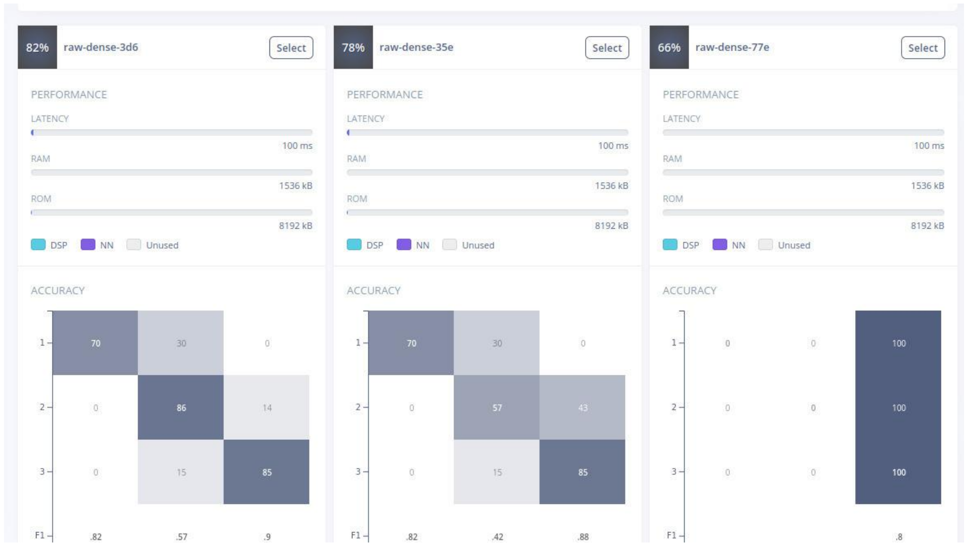


Figure 3: EON-Tuner models for the data augmented with RGB information

Further the data for compute reasons in Phase-1 comprises 64x64 images with definite loss of information. The original images however were of 320x240 resolution. In addition, it has been observed that the data collected during day 3 and day 4 had been poorly classified by CNN (Convolutional Neural Networks) during phase 1. So, we used the images with original resolution and combined day 3 and day 4 data. This change was introduced as the tomatoes did not show significant changes in their color and texture after day 3 which limits the amount of inferencing, we could do via the image alone. Further, we used only the RGB information in the data. Figure 4 shows the top 3 ML models identified by the EON-Tuner. It is worth observing that the test accuracies achieved were 82%, 78% and 66%.



raw-dense-3d6 (82%)

1	70	30	0
2	0	86	14
3	0	15	85
F1	.82	.57	.9

raw-dense-35e (78%)

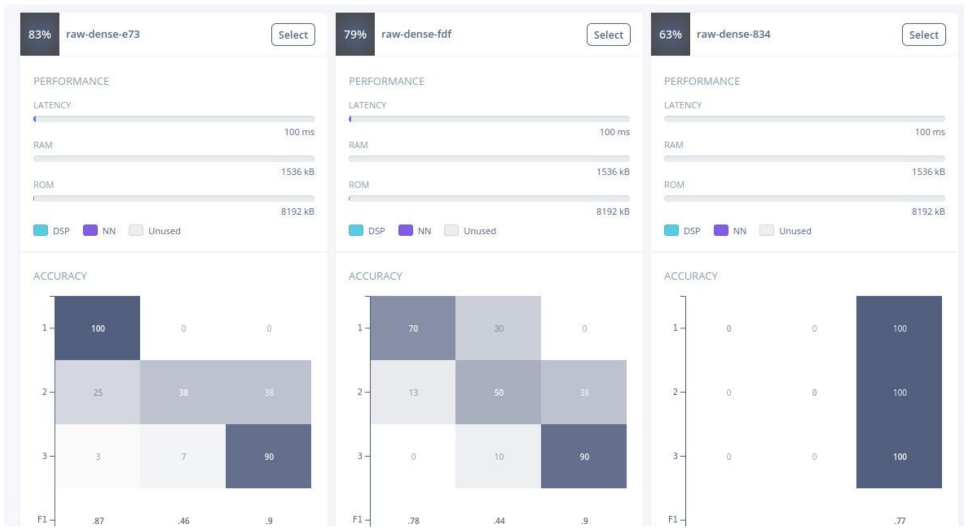
1	70	30	0
2	0	57	43
3	0	15	85
F1	.82	.42	.88

raw-dense-77e (66%)

1	0	0	100
2	0	0	100
3	0	0	100
F1			.8

Figure 4: Models selected for data with original image resolution

Finally, after adding the HSV and LAB color spaces' statistical information along with that of RGB, test accuracies did not improve much (Refer Figure 5 below). This is not surprising, as we observed that changing color spaces as we deduced was simply increasing redundancy in the dataset.


83% raw-dense-e73 Select
79% raw-dense-fdf Select
63% raw-dense-834 Select
PERFORMANCE
LATENCY 100 ms
RAM 1536 kB
ROM 8192 kB
DSP NN Unused
ACCURACY
1: 100 0 0 | 2: 25 38 38 | 3: 3 7 90 | F1: .87 .46 .9
1: 70 30 0 | 2: 13 50 38 | 3: 0 10 90 | F1: .78 .44 .9
1: 0 0 100 | 2: 0 0 100 | 3: 0 0 100 | F1: .77
Figure 5: Models suggested for data with original resolution and RGB+HSV color space information

The preliminary experiments with AutoML model (EON-Tuner) in Edge Impulse gave us both interesting insights and research directions.

- The best performing AutoML models could be imported for Spresense as the target device
- The ML models, selected by EON-Tuner, per se have not been investigated as part of the initial experiments. Further experiments will investigate the selected models.
- The statistical information regarding the color spaces is lossy.
- Data engineering is very crucial for this Tiny AutoML task and needs considerable care and effort.

AutoML – From Edge Impulse to Spresense

The most accurate model suggested by EonTuner achieved 78% test accuracy. The chosen model has two dense layers – one layer with 40 neurons and the other with 20 neurons. The final layer is a dropout layer with a dropout rate of 0.25. The model has a learning rate of 0.0005 and trained for 30 epochs.

Subsequently, the chosen ML model has to be ported to the Spresense edge device. This will help to investigate the benchmarking of the chosen model in terms of performance both in edge device as well as in EonTuner.

Though, AutoML experiments in EonTuner has been primarily run for Sony Spresense as the target board, after the model has been selected as the primary block for the classifier the option to build the model for Spresense has not been available in the deployment page. Consequently, efforts have been made to download the model through Edge Impulse API in 'ipynb' format which could later be converted to a tensor flow lite model. However, due to GET error, help was sought from the Edge Impulse Forums. Eventually, the classifier blocks have been downloaded as TensorFlow lite quantized model. Issues with the driver code were also faced. Eventually, a TensorFlow Lite (int8 quantized) model and is of 5KB size was downloaded from Edge Impulse. The other formats of the model are TensorFlow Lite (float32), Keras h5 model and TensorFlow SavedModel. The float32 model is 13KB in size, Keras h5 is 12KB and the SavedModel is 18KB in size.

While troubleshooting the issues with the downloading of EonTuner model, other AutoML platforms have also been explored to facilitate granular control over the architecture of chosen ML models. The following are the two popular alternatives.

AdaNet: AdaNet is a lightweight TensorFlow based framework for automatically learning high-quality models with minimal expert intervention. It can operate in two ways -- a collection of candidate subnetworks providing the best accuracy is taken up for the ensemble model; all the candidate subnetworks are included in the ensemble model. Users also have control over choosing which method to be employed. As the architecture is built around ensemble of linear and dense estimators, the performance is bound to be higher than a normal model and AdaNet also guarantees higher speed.

AutoKeras: AutoKeras is an AutoML system based on Keras framework. It uses Neural Architecture Search (NAS) which automates construction of Artificial Neural Networks (ANNs). Due to this feature, an efficient Neural Network can be formed without any prior knowledge of the various hyper-parameters involved.

In addition to the AutoML paradigm, we also propose AI (Artificial Intelligence) acceleration techniques that focus on compressing large ML models for effective deployment in edge devices. *Quantization* and *Pruning* are the two predominant techniques used for compressing ML/DL models. Quantization helps condensing huge TensorFlow models which are to be deployed on the edge devices by converting input values from large set to output values in a smaller set. There are two main techniques:

- Post training quantization
- Quantization Aware training

Pruning helps to make models smaller (with minimal loss in accuracy or performance) and faster by removing redundant or non-significant parameters such as connectors, neurons, channels, layers, filters etc. Pruning can also be done by removing the weights. Pruning can be done while the model is being trained or post the training.

In the context of constrained applications like food drying, we also considered *incremental learning.* Incremental learning does not require a large amount of data to be collected before the model is being trained, instead it will start with a basic model typically predicting the average value seen so far. When new data arrives, the model is trained to learn more complex patterns (The model grows its knowledge and accuracy slowly as more data is observed).

Modelling the Neural Network using TensorFlow*:* The model suggested by Edge Impulse EON Tuner (the AutoML engine used in the experiments) was reconstructed in python notebook. EON Tuner suggested a model with two dense layers, with 40 and 20 neurons respectively and a dropout rate of 0.25. The output layer contains 3 classes (Fresh, semi-dried and dried). This model was trained for higher epochs on normalized dataset to increase the performance of the model.

The model was trained on dataset comprising statistical values of RGB, HSV and LAB color spaces along with temperature and humidity values. These values are normalized to a range of 0-1. Upon training the model for 150 epochs, it gave an accuracy of 95%. The architecture of the model is shown in Figure 4.

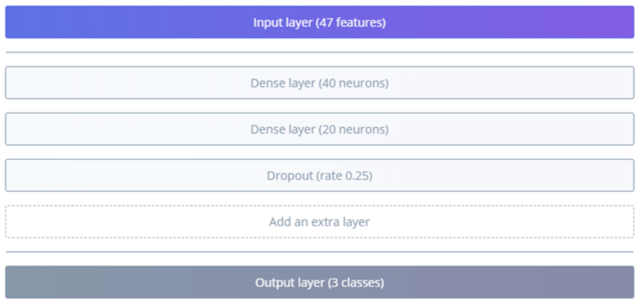


Figure 4: Architecture suggested by EON Tuner

Test for Overfit: A model typically shows decrease in training loss and validation loss as we train it for more epochs. A model is said to be overfit when the training loss decreases, but the validation loss of the model increases. We use this criterion to check if the suggested model has been overfit.

The training loss and validation loss of the model was plotted after training it for 700 epochs as shown in Figure 5. The Figure shows that the validation loss decreases till around 180 epochs and then increases, whereas the training loss keeps decreasing. Since we train the model for 150 epochs, we could conclude that the model is not overfit.

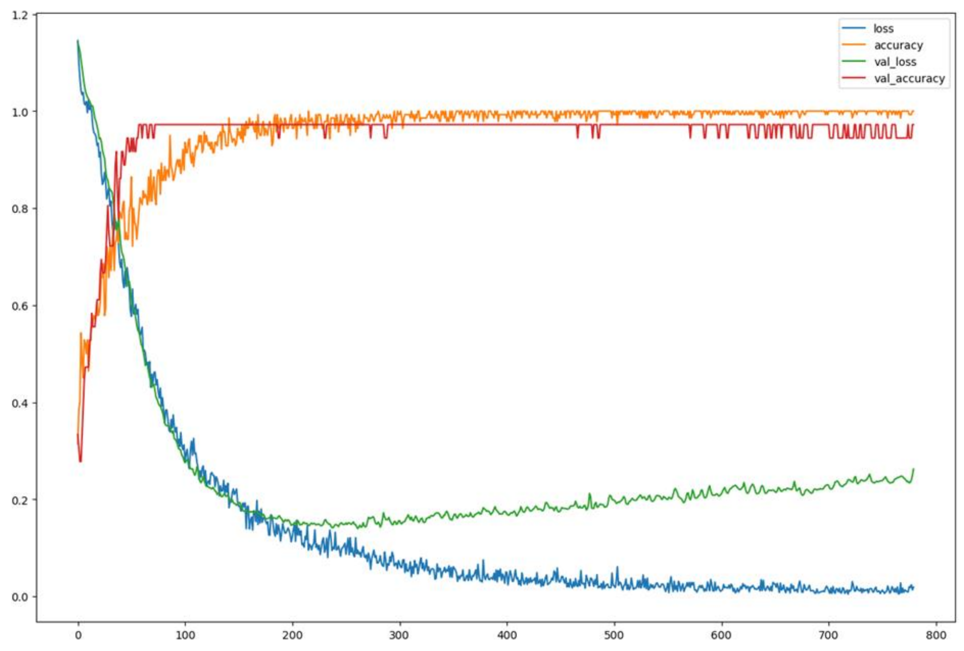


Figure 5: Training and Validation loss of the EON Tuner suggested model

Burning the model from EON Tuner on board: Various models have been trained on Edge Impulse EON Tuner with a statistical dataset comprising HSV, LAB, RGB color space intensity values, humidity, and temperature. From the top models suggested by the Eon Tuner, the best model with 78% accuracy was selected and downloaded. The selected model was then loaded on the Sony Spresense board. But the model gave only 64% accuracy on board when it was tested with the same test dataset that was used earlier in the software.

From the analysis, we hypothesized the reason to be EON Tuner's limit on the maximum number of epochs during the training of the model. To get a better understanding and more flexibility, the current model was recreated in Python Notebook using TensorFlow. The model was trained with the same train and test dataset that was used earlier. The Model was trained with more epochs now and with modified constraints for better accuracy. During the testing in the Python Notebook, the model reached 95% of test accuracy. The newly trained model was then burnt on Sony Spresense board. However, the model performance again decreased from 95% to 64% on board. This debunked our hypothesis and we found out the issue was in the quantization of weights.

Quantization Aware Training: The experiments thus conducted proved one thing, either implementing the same model in the board or converting a trained model to a *tf-lite* format heavily affects the performance. This can be attributed to the fact that during quantization there is a lot of internal compression and loss of precision. Because of that, the model performs poorly when implemented on board. The drop in performance was as drastic as a 20+% drop in accuracy on the board. Quantization-aware training was implemented expecting no drop or lesser drop in performance while executing the model on the Sony Spresense board.

Quantization aware training was implemented using the high-level library provided by TensorFlow. The main steps involved in this process are as follows.

1. Define the model
2. Fit the training data to the model
3. Train the model using the training dataset and validate using the validation dataset
4. Using *tensorflow_model_optimization*, quantize the trained model
5. Recompile the quantized model
6. Train and evaluate the quantized model
7. The resulting quantized model can then be converted to the *tf-lite* model, which can be deployed in the Sony Spresense board

With respect to the performance, either there will be no drop in the baseline model and the quantization aware training model (idea scenario) or there will be a minor drop in the accuracy of the quantization aware training model. The simulation experiments showed that both the baseline and quant models give the same accuracy.

Experimenting with a new dataset: We created a new dataset merging all the available ppm files. Used RGB, LAB, and HSV color spaces and extracted relevant statistical values. The data collection process spanned for 4 days hence logically added 4 classes to the dataset.

Without normalization, the model gave a 43% accuracy trained over 65 epochs. After normalization, the model gave a 95% accuracy over 65 epochs. After quantization-aware training, the normalized model on quantization aware training and converting to *tf-lite* model gave 96%. This was suspected because of class imbalance. Using oversampling of under-represented classes, the class imbalance issue was fixed, and the resultant model gave an accuracy of 87.5%. Hence the difference in performance between the model in computer and on board was reduced from ~20% to ~7%.

AutoML Experiments Setup:

Data Preprocessing Steps

1. The image dataset used to create the dataset contains images of tomatoes over a period of 4 days. The size of each image is 64*64.
2. Each image from the dataset is read and its pixel wise RGB information is stored in a Numpy array. This is done for every pixel of every image in each class of the 3 classes.
3. This Numpy array is used to get the statistical values such as mean, min, max, standard deviation and variance of R, G and B of each image.

4. These statistical values are appended to the already existing dataset used previously for KNN model training, which already contains the HSV and LAB statistical values and the temperature and humidity.
5. We finally have a dataset with RGB, HSV and LAB statistical values with temperature and humidity.
6. This dataset is used to train the model for further classification with 3 classes:
   - Class 1 – Day 1
   - Class 2 – Day 2
   - Class 3 – Day 3 and Day 4 combined

AutoML Experiments – New dataest and Background Subtraction

AutoML suggested Deep Learning Model on New Polyhouse Data: Based on the devised data collection standards, new data on Tomato drying has been collected. The architecture, as detailed in Figure 4, suggested by the AutoML prior experiments has been run on the new drying dataset.

As has been pre-processed earlier, the drying images of tomato have been converted into HSV and LAB colour spaces and their statistical features viz. mean, standard deviation, min, max and variance have been collected.

After having run the above architecture on the dataset for 150 epochs, the test loss and accuracy were respectively 0.3119595050 and 0.8999999. After quantization aware training (i.e., 150 epochs of normal training and 1 epoch of quantization aware training) to enable the porting of architecture into edge devices for faster inference, the test loss and accuracy were 0.32073074579 and 0.8999999. It is worth noting that the accuracy is retained yet the architecture by virtue of the quantization is light-weight amenable for porting to hardware for faster inferencing.

Effect of Background Subtraction on the Model Performance Using Old Polyhouse Data: To investigate the effect of background subtraction of drying images on the model accuracy, the architecture chosen by AutoML in Figure 1 has been applied on background subtracted images from the old Polyhouse data collected during Phase 1.

Figure 6 shows the original down sampled image. After 150 epochs, the model achieved an accuracy of 93.33%. Like the previous experiment, the model has been subjected to 2 epochs of quantization aware training apart from the normal 150 epochs, achieving the same accuracy on the input images.

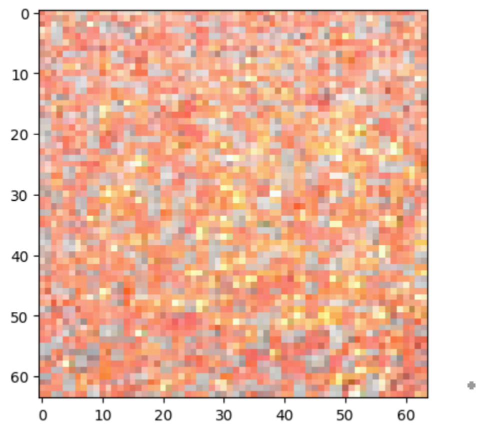


Figure 6: Original down sampled image

Using New Polyhouse Data: The same model has been employed with 150 epochs of training on the new polyhouse data. The model achieved a loss of 0.34995353221 and accuracy of 0.889. However, after 2 more extra epochs of quantization aware training the loss became 0.3363 and the accuracy dipped slightly as 0.8775. The loss of accuracy though very marginal is agreeable by virtue of the quantization.

AutoML on the New Polyhouse Data: Rather than validating the already selected architecture on the newly collected data, AutoML has been employed directly on the new data to choose the appropriate model architecture for the data. Figure 7 shows the top three models chosen by EON Tuner. Figure 8 shows the architecture of the top performing model on the new drying data.

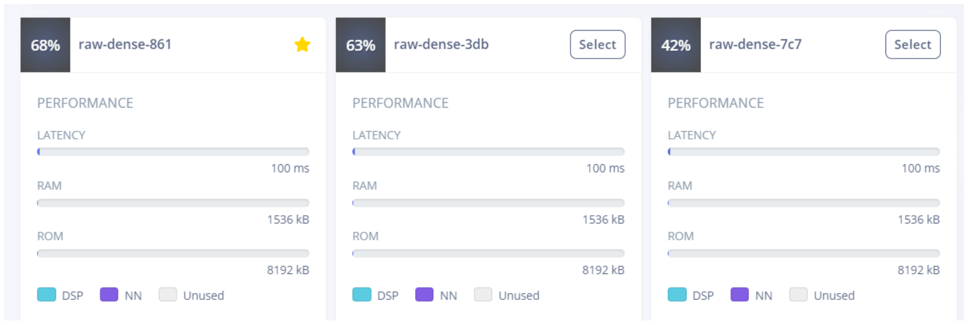


Figure 7: Top three model architecture chosen by Eon Tuner for the new data

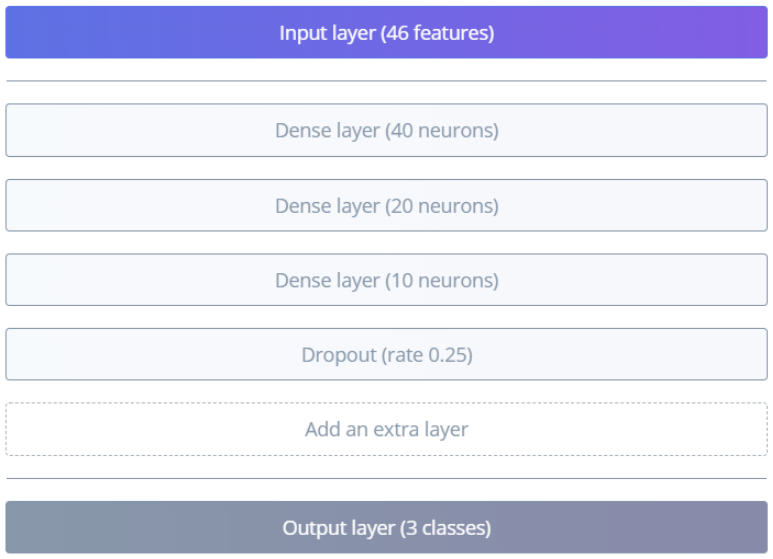
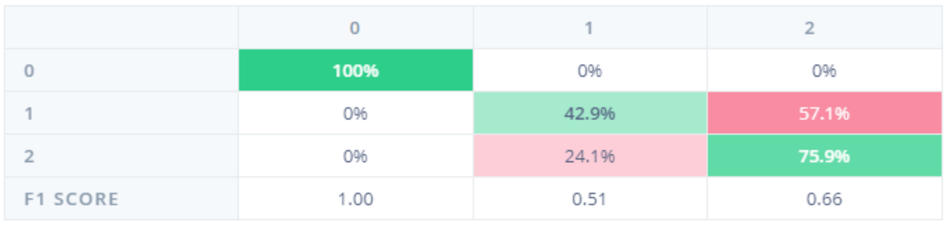


Figure 8: Architecture of the Top Model chosen by Eon Tuner

Figures 9 and 10 respectively show the confusion matrix and the data exploration graph for the top model. As you can see from Figure 9, the challenge primarily lies in classifying the semi-dried and fully dried images as not only the coloration is similar but the down sampling of images cause loss of information.



	0	1	2
0	100%	0%	0%
1	0%	42.9%	57.1%
2	0%	24.1%	75.9%
F1 SCORE	1.00	0.51	0.66

Figure 9: Confusion Matrix for the Top Performing Model

Figure 10 shows the details of classification through the Data Exploration Graph. As can be seen, Day 1 images are easily separated from the rest of the images by the model while both Day 2 and Day 3 images are not very well distinguished for the reasons cited above.

It is worth noting that henceforth all the previous experiments focussed on Deep Learning models for the AutoML experiments. However, as we plan to deploy these models in Federated Learning framework, we explored the possibilities of AutoML libraries which employ only non-deep learning ML algorithms.

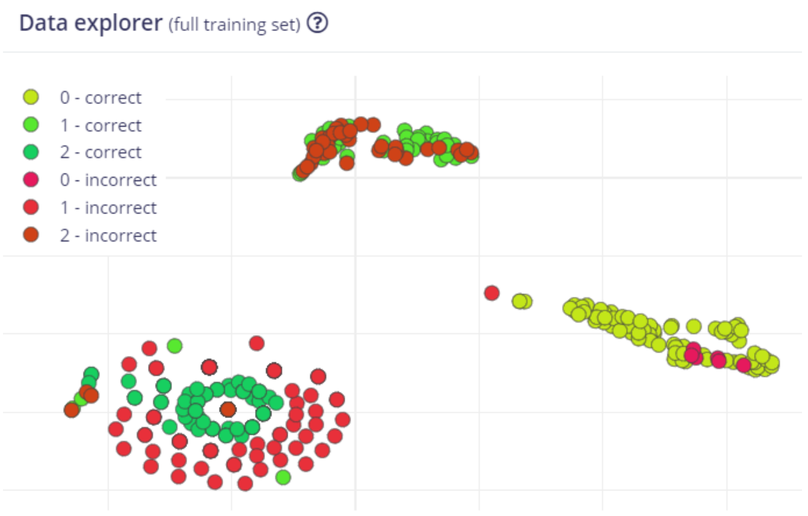
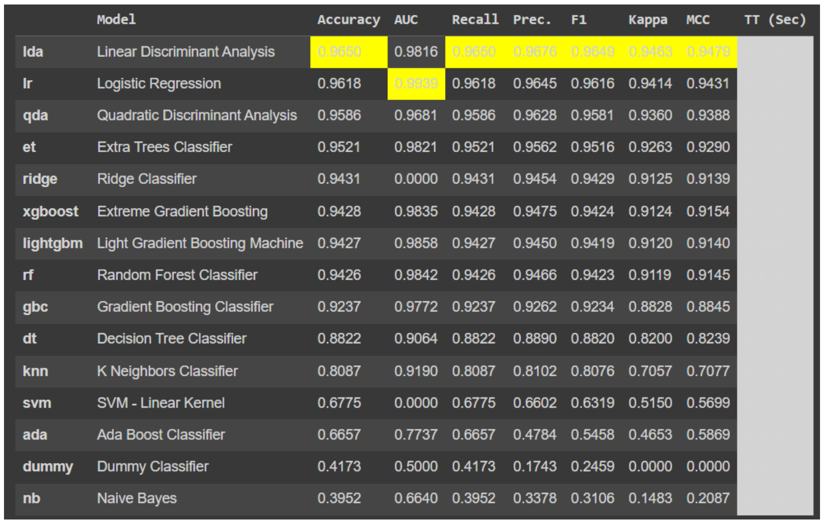


Figure 10: Data Exploration Graph showing the details of classification.

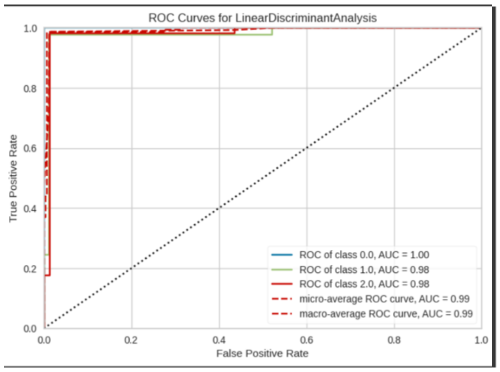
AutoML Libraries with ML Algorithms as Constituents

Various AutoML libraries which use only (non-Deep Learning) ML algorithms have been experimented using the new dataset. PyCaret library uses various models including scikit-learn, XGBoost, LightGBM, CatBoost, spaCy, Optuna, Hyperopt, Ray etc. Figure 11 shows the performance of various constituent models on the new data. Linear Discriminant Analysis (LDA) algorithm exhibited superior performance on the new data set. Figure 12a shows the ROC curve obtained by LDA and Figure 12b shows the feature importance plot. As you can see, standard deviation and variance are the most dominating features that facilitate the classification.



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lda	Linear Discriminant Analysis	0.9650	0.9816	0.9650	0.9676	0.9649	0.9463	0.9479	
lr	Logistic Regression	0.9618	0.9939	0.9618	0.9645	0.9616	0.9414	0.9431	
qda	Quadratic Discriminant Analysis	0.9586	0.9681	0.9586	0.9628	0.9581	0.9360	0.9388	
et	Extra Trees Classifier	0.9521	0.9821	0.9521	0.9562	0.9516	0.9263	0.9290	
ridge	Ridge Classifier	0.9431	0.0000	0.9431	0.9454	0.9429	0.9125	0.9139	
xgboost	Extreme Gradient Boosting	0.9428	0.9835	0.9428	0.9475	0.9424	0.9124	0.9154	
lightgbm	Light Gradient Boosting Machine	0.9427	0.9858	0.9427	0.9450	0.9419	0.9120	0.9140	
rf	Random Forest Classifier	0.9426	0.9842	0.9426	0.9466	0.9423	0.9119	0.9145	
gbc	Gradient Boosting Classifier	0.9237	0.9772	0.9237	0.9262	0.9234	0.8828	0.8845	
dt	Decision Tree Classifier	0.8822	0.9064	0.8822	0.8890	0.8820	0.8200	0.8239	
knn	K Neighbors Classifier	0.8087	0.9190	0.8087	0.8102	0.8076	0.7057	0.7077	
svm	SVM - Linear Kernel	0.6775	0.0000	0.6775	0.6602	0.6319	0.5150	0.5699	
ada	Ada Boost Classifier	0.6657	0.7737	0.6657	0.4784	0.5458	0.4653	0.5869	
dummy	Dummy Classifier	0.4173	0.5000	0.4173	0.1743	0.2459	0.0000	0.0000	
nb	Naive Bayes	0.3952	0.6640	0.3952	0.3378	0.3106	0.1483	0.2087	

Figure 11: Performance of Constituent Models in PyCaret



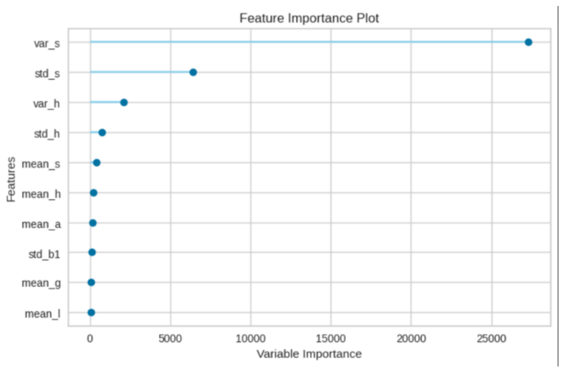
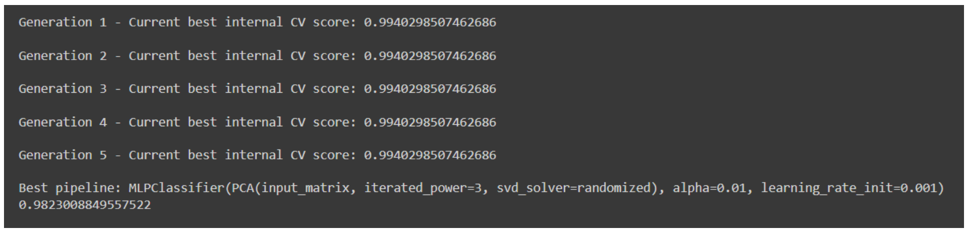


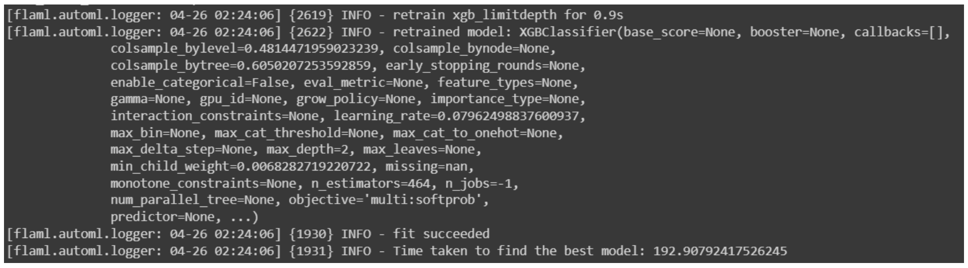
Figure 12: (a) ROC Curve for LDA (b) Feature Importance Plot

TPOT and FLAML are the other two AutoML libraries that were also experimented with. Figure 13 shows a sample run of 5 generations with TPOT. It is worth noting that the selected MLP classifier is still a connectionist algorithm. Figure 14 shows a sample run being executed with FLAML.



```
Generation 1 - Current best internal CV score: 0.9940298507462686

Generation 2 - Current best internal CV score: 0.9940298507462686

Generation 3 - Current best internal CV score: 0.9940298507462686

Generation 4 - Current best internal CV score: 0.9940298507462686

Generation 5 - Current best internal CV score: 0.9940298507462686

Best pipeline: MLPClassifier(PCA(input_matrix, iterated_power=3, svd_solver=randomized), alpha=0.01, learning_rate_init=0.001)
0.9823008849557522
```

Figure 13: Sample 5 generations of TPOT with best pipeline



```
[flaml.automl.logger: 04-26 02:24:06] {2619} INFO - retrain xgb_limitdepth for 0.9s
[flaml.automl.logger: 04-26 02:24:06] {2622} INFO - retrained model: XGBClassifier(base_score=None, booster=None, callbacks=[],
              colsample_bylevel=0.4814471959023239, colsample_bynode=None,
              colsample_bytree=0.6050207253592859, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.07962498837600937,
              max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaves=None,
              min_child_weight=0.0068282719220722, missing=nan,
              monotone_constraints=None, n_estimators=464, n_jobs=-1,
              num_parallel_tree=None, objective='multi:softprob',
              predictor=None, ...)
[flaml.automl.logger: 04-26 02:24:06] {1930} INFO - fit succeeded
[flaml.automl.logger: 04-26 02:24:06] {1931} INFO - Time taken to find the best model: 192.90792417526245
```

Figure 14: Sample FLAML run

AutoML- Inferencing on the live image

Dataset Creation

- A new dataset has been created using over 550 images from the latest data collection.
- The ppm images were subjected to Background Subtraction which helps isolate the tomatoes in the image from the background as shown in Figure 15.

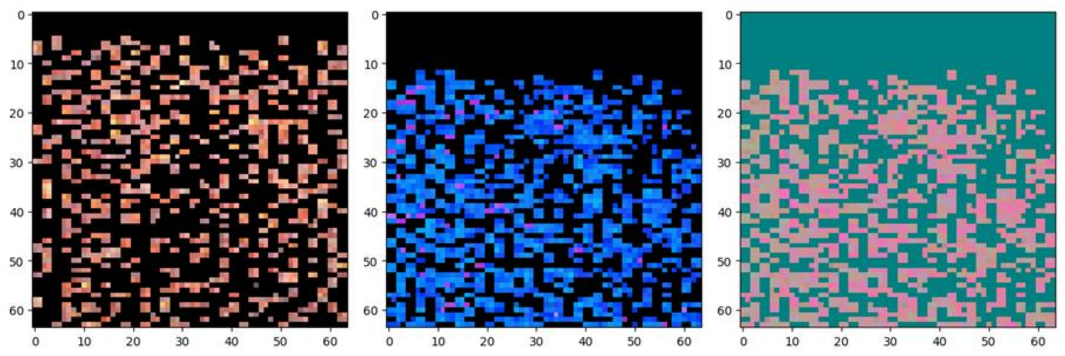


Figure 15: Sample images from the dataset created

Lab Inferencing (After Quantization Aware Training): The model selected by EON Tuner was subjected to 150 epochs of normal training and 2 epochs of quantization aware training. The model achieved an accuracy of 88.2% on the new data set reiterating the efficacy of the AutoML method.

Porting Model into Spresense

- The model was flashed into Sony Spresense and deployed in the polyhouse for live inferencing.
- The model predicted the dryness of the tomatoes based on the image captured by Spresense which is then converted into RGB, LAB and HSV statistical values and fed to the model with temperature and humidity of the polyhouse at that moment.

Regression task

The regression task was chosen for food drying as many stakeholders are interested in having a model that gives "time left to dry". To make a regression model, several changes need to be made to the dataset generation process and the modelling.

Data generation for regression:

1. The dataset contains some mismatching values between images and other inferences due to the variation in time stamp. It was cleaned by matching the closest pairs among them.

2. Principal Component Analysis was done with the inferences of color values of the image. Created 10 PCA (Principal Component Analysis) attributes, which were then visualized and analyzed based on their Explained Variance Ratios.

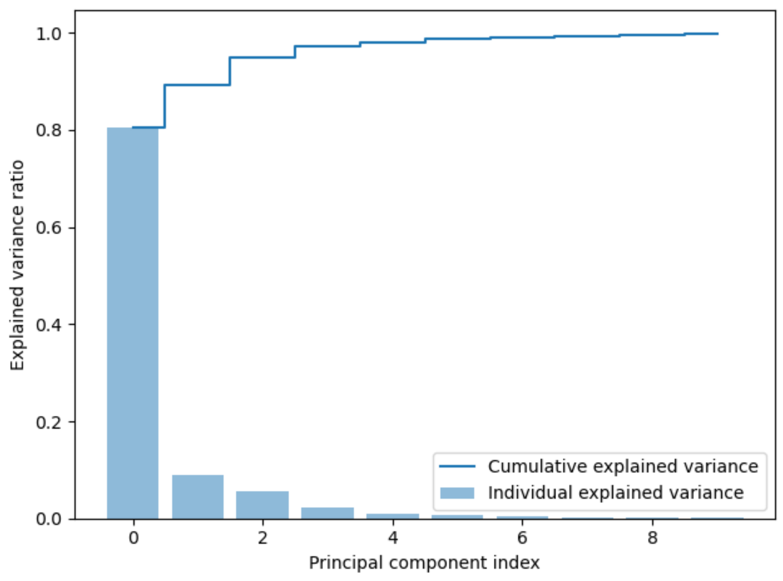


Figure 16: Principal Components for regression analysis

As shown in Figure 16, the first 3 PCs were able to explain over 90% variance. Those components were selected and added to the dataset so that they could improve the accuracy of the model.

Fitting Traditional Model on Dataset:

Linear Regression: Linear Regression was implemented with the final dataset. That resulted in a linear model with the Mean squared error of **1045614567.875** and R-squared of **0.168**. Linear regression is unable to fit the dataset, and this can be attributed to the non-linearity in the dataset. Figure 17 shows the heatmap.

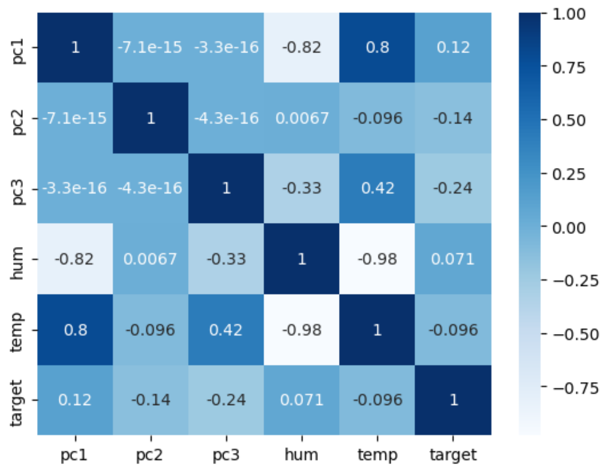


Figure 17: Heat map for Linear Regression

Decision Tree Regressor: A decision tree regressor is applied on the dataset as it is more robust to non-linearity in the dataset. That resulting model results in a Root Mean Squared (RMS) error of **10364.814** and R-squared score of **0.92.**

Light Gradient Boosting Regressor: Light Gradient Boosting Model was applied on the dataset. Upon running the model for 30 rounds, the model resulted in an RMSE value of **10939.8**. From the regression plots, figure 18, it can be inferred that the model predicted most of the values closer to the ground truth values.

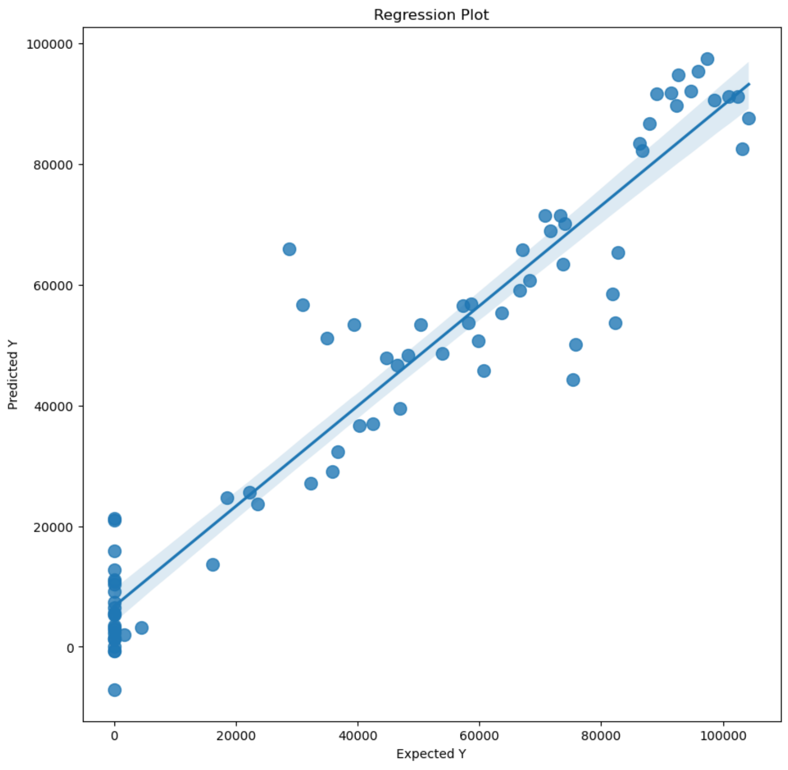


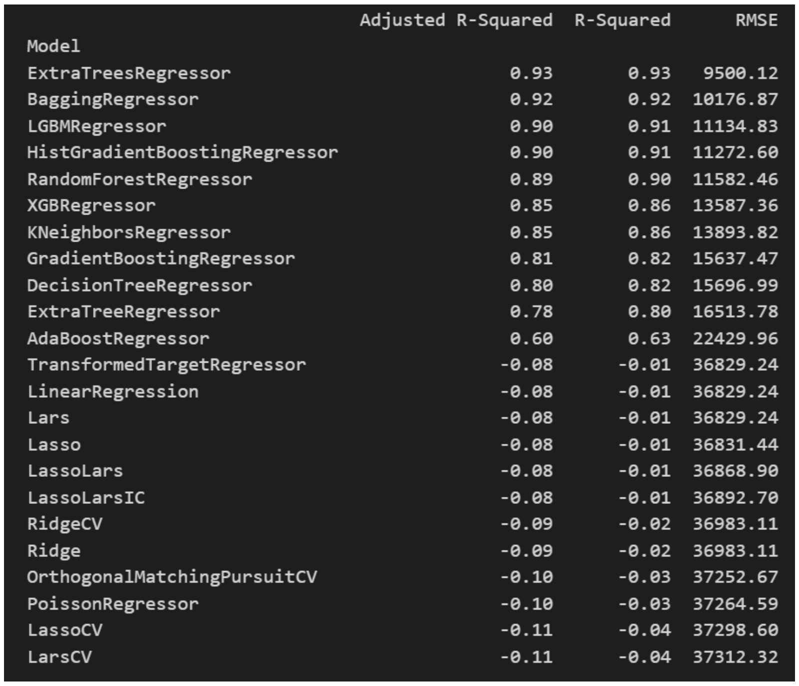
Figure 18: Regression Plot

XGB Regressor: Finally, eXtreme Gradient Boosting Regressor was implemented on the dataset. The resulting regression model provided Mean Absolute Error (MAE) and R-squared values of **6258.66** and **0.85**, respectively.

AutoML Solutions:

The traditional regression techniques were not trained with the most optimal set of parameters, and it is time consuming to manually search for the best regressor model by implementing all of them. AutoML simplifies this process and provides the best set of models with the semi-optimized or fully optimized set of hyperparameters.

LazyPredict: LazyPredict is a low code python library that trains different regressors simultaneously with the same dataset and provides the top models in sorted performance order as shown in Figure 19.



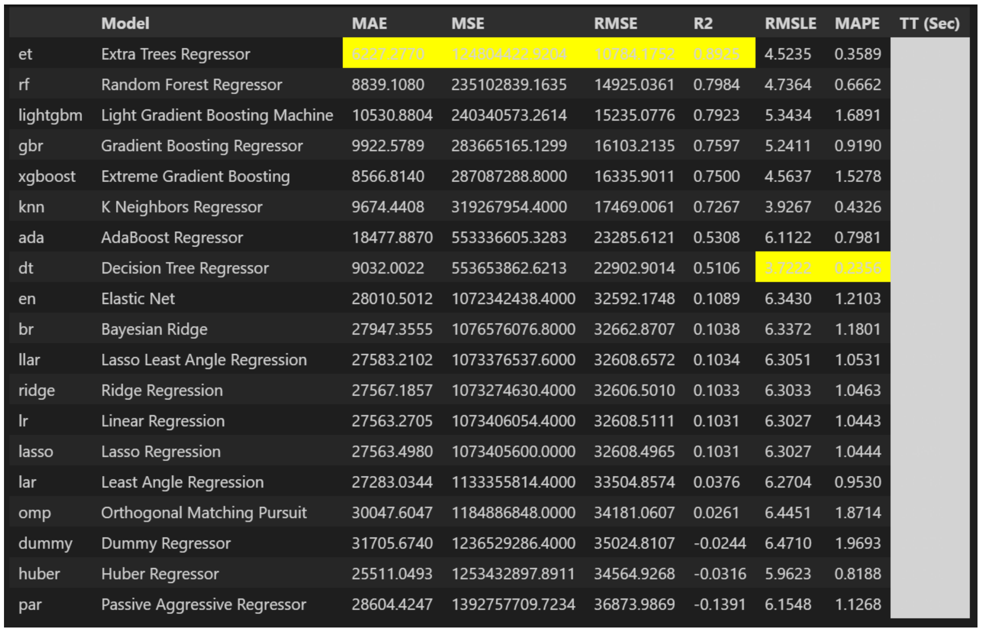
```
                               Adjusted R-Squared  R-Squared      RMSE
Model
ExtraTreesRegressor                          0.93       0.93   9500.12
BaggingRegressor                             0.92       0.92  10176.87
LGBMRegressor                                0.90       0.91  11134.83
HistGradientBoostingRegressor                0.90       0.91  11272.60
RandomForestRegressor                        0.89       0.90  11582.46
XGBRegressor                                 0.85       0.86  13587.36
KNeighborsRegressor                          0.85       0.86  13893.82
GradientBoostingRegressor                    0.81       0.82  15637.47
DecisionTreeRegressor                        0.80       0.82  15696.99
ExtraTreeRegressor                           0.78       0.80  16513.78
AdaBoostRegressor                            0.60       0.63  22429.96
TransformedTargetRegressor                  -0.08      -0.01  36829.24
LinearRegression                            -0.08      -0.01  36829.24
Lars                                        -0.08      -0.01  36829.24
Lasso                                       -0.08      -0.01  36831.44
LassoLars                                   -0.08      -0.01  36868.90
LassoLarsIC                                 -0.08      -0.01  36892.70
RidgeCV                                     -0.09      -0.02  36983.11
Ridge                                       -0.09      -0.02  36983.11
OrthogonalMatchingPursuitCV                 -0.10      -0.03  37252.67
PoissonRegressor                            -0.10      -0.03  37264.59
LassoCV                                     -0.11      -0.04  37298.60
LarsCV                                      -0.11      -0.04  37312.32
```

Figure 19: LazyPredict sample output

It can be inferred from Figure 19 that the best regression models resulted by the LazyPredict is ExtraTreesRegressor. There are models with R-Squared values less than 0 which means that their performance is poorer than the constant function. Although LazyPredict provides the best set of regressors, it does not provide us with the pipeline to get inferencing from the model.

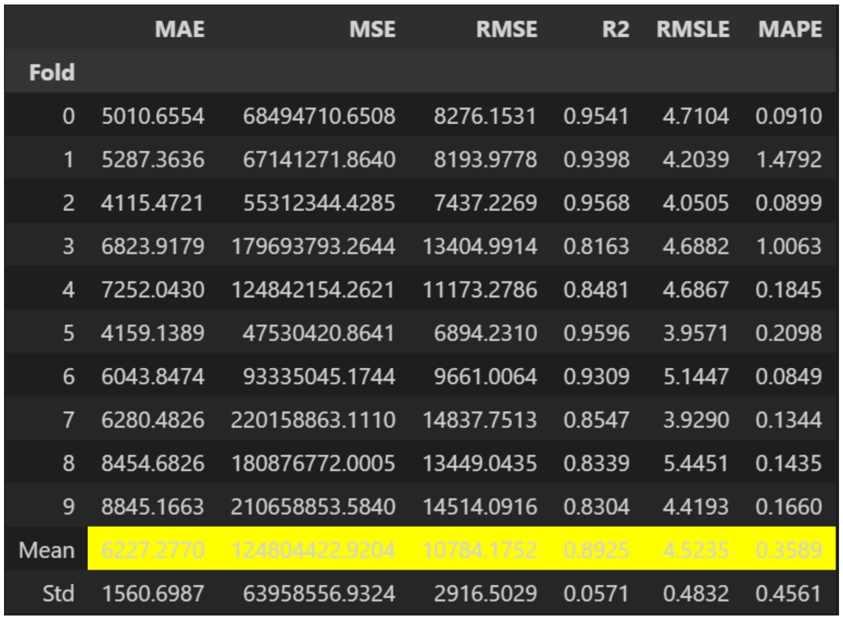
PyCaret: PyCaret is an another AutoML model that performs multiple simultaneous regressions and compares them to provide highly efficient models. Figure 20 shows the sample output of PyCaret. It is observed that PyCaret also suggests Extra Trees Regressor as the top model. And many trained models have R-Squared more than 0 which means PyCaret trained those models more efficiently than LazyPredict.

Extra Tree Regression: From the suggestions of LazyPredict and PyCaret, Extra Trees Regression was found to be the most suited model for the dataset. Hence the ET regression model was implemented for 10 folds using PyCaret. Figure 21 highlights the metrics received by training the model for 10 folds using the default set of hyperparameters from PyCaret. Although the results are satisfactory, the performance of the original model could be improved by fine tuning the hyperparameters.



	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	6227.2770	124804422.9204	10784.1752	0.8925	4.5235	0.3589	
rf	Random Forest Regressor	8839.1080	235102839.1635	14925.0361	0.7984	4.7364	0.6662	
lightgbm	Light Gradient Boosting Machine	10530.8804	240340573.2614	15235.0776	0.7923	5.3434	1.6891	
gbr	Gradient Boosting Regressor	9922.5789	283665165.1299	16103.2135	0.7597	5.2411	0.9190	
xgboost	Extreme Gradient Boosting	8566.8140	287087288.8000	16335.9011	0.7500	4.5637	1.5278	
knn	K Neighbors Regressor	9674.4408	319267954.4000	17469.0061	0.7267	3.9267	0.4326	
ada	AdaBoost Regressor	18477.8870	553336605.3283	23285.6121	0.5308	6.1122	0.7981	
dt	Decision Tree Regressor	9032.0022	553653862.6213	22902.9014	0.5106	3.7222	0.2356	
en	Elastic Net	28010.5012	1072342438.4000	32592.1748	0.1089	6.3430	1.2103	
br	Bayesian Ridge	27947.3555	1076576076.8000	32662.8707	0.1038	6.3372	1.1801	
llar	Lasso Least Angle Regression	27583.2102	1073376537.6000	32608.6572	0.1034	6.3051	1.0531	
ridge	Ridge Regression	27567.1857	1073274630.4000	32606.5010	0.1033	6.3033	1.0463	
lr	Linear Regression	27563.2705	1073406054.4000	32608.5111	0.1031	6.3027	1.0443	
lasso	Lasso Regression	27563.4980	1073405600.0000	32608.4965	0.1031	6.3027	1.0444	
lar	Least Angle Regression	27283.0344	1133355814.4000	33504.8574	0.0376	6.2704	0.9530	
omp	Orthogonal Matching Pursuit	30047.6047	1184886848.0000	34181.0607	0.0261	6.4451	1.8714	
dummy	Dummy Regressor	31705.6740	1236529286.4000	35024.8107	-0.0244	6.4710	1.9693	
huber	Huber Regressor	25511.0493	1253432897.8911	34564.9268	-0.0316	5.9623	0.8188	
par	Passive Aggressive Regressor	28604.4247	1392757709.7234	36873.9869	-0.1391	6.1548	1.1268	

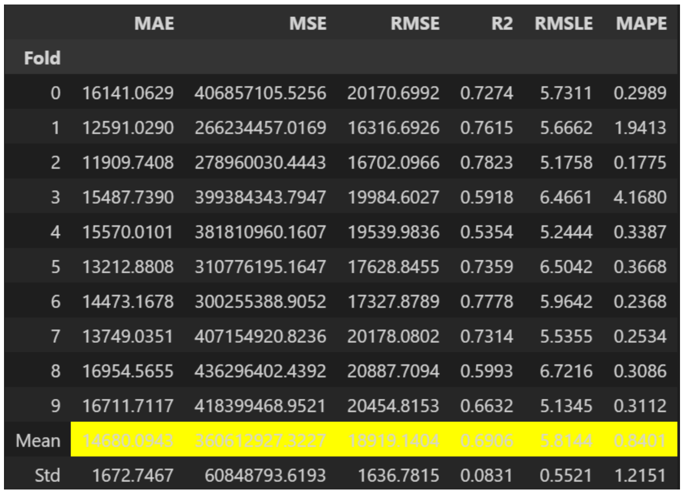
Figure 20: Sample output of PyCaret



	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	5010.6554	68494710.6508	8276.1531	0.9541	4.7104	0.0910
1	5287.3636	67141271.8640	8193.9778	0.9398	4.2039	1.4792
2	4115.4721	55312344.4285	7437.2269	0.9568	4.0505	0.0899
3	6823.9179	179693793.2644	13404.9914	0.8163	4.6882	1.0063
4	7252.0430	124842154.2621	11173.2786	0.8481	4.6867	0.1845
5	4159.1389	47530420.8641	6894.2310	0.9596	3.9571	0.2098
6	6043.8474	93335045.1744	9661.0064	0.9309	5.1447	0.0849
7	6280.4826	220158863.1110	14837.7513	0.8547	3.9290	0.1344
8	8454.6826	180876772.0005	13449.0435	0.8339	5.4451	0.1435
9	8845.1663	210658853.5840	14514.0916	0.8304	4.4193	0.1660
Mean	6227.2770	124804422.9204	10784.1752	0.8925	4.5235	0.3589
Std	1560.6987	63958556.9324	2916.5029	0.0571	0.4832	0.4561

Figure 21: ET Regressor Performance

Figure 22 shows the resultant metrics by fitting 10 folds for each of the 10 candidates with a distinct set of hyperparameters, totaling 100 fits. From the performance metric of both models, the original model was fine-tuned to provide even more superior performance. The residual plot, in Figure 23, suggests that the training data fits seamlessly with the fine-tuned set of hyper parameters, but residual values are high for certain data points in the test-set. This could be due to few outliers in the temperature or humidity readings in the dataset. The prediction error plot, in Figure 24, highlights how accurately the regression curve fits the dataset. The closer the prediction error plot to the identity function, the better the model.



Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	16141.0629	406857105.5256	20170.6992	0.7274	5.7311	0.2989
1	12591.0290	266234457.0169	16316.6926	0.7615	5.6662	1.9413
2	11909.7408	278960030.4443	16702.0966	0.7823	5.1758	0.1775
3	15487.7390	399384343.7947	19984.6027	0.5918	6.4661	4.1680
4	15570.0101	381810960.1607	19539.9836	0.5354	5.2444	0.3387
5	13212.8808	310776195.1647	17628.8455	0.7359	6.5042	0.3668
6	14473.1678	300255388.9052	17327.8789	0.7778	5.9642	0.2368
7	13749.0351	407154920.8236	20178.0802	0.7314	5.5355	0.2534
8	16954.5655	436296402.4392	20887.7094	0.5993	6.7216	0.3086
9	16711.7117	418399468.9521	20454.8153	0.6632	5.1345	0.3112
Mean	14680.0943	360612927.3227	18919.1404	0.6906	5.8144	0.8401
Std	1672.7467	60848793.6193	1636.7815	0.0831	0.5521	1.2151

Figure 22: ET Regressor Performance with 10 fitting with distinct hyperparameters

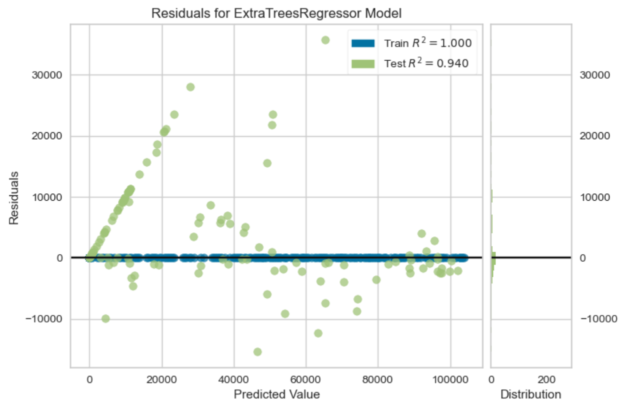


Figure 23: Residual Plot

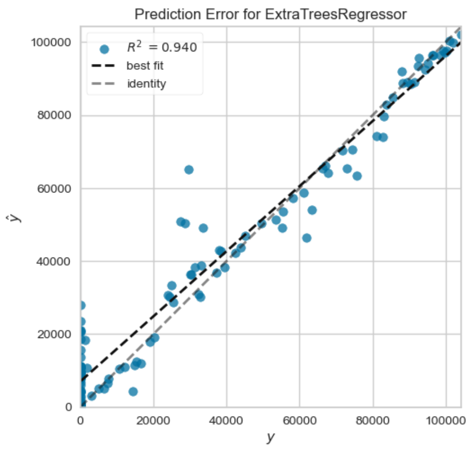


Figure 24: Prediction Error Plot

Conclusion

The AutoML experiments have been carried out to automate the model selection for live inferencing, in Spresense, from drying of agricultural food products. It has been realized that the AutoML was indeed effective in identifying appropriate ML models. However, efforts (like quantization aware training) need to be put to tune the models considering the hardware limitations in terms of model inferencing. AutoML has been extended from classification tasks to regression tasks with good results. The outcome of the experiments will also facilitate the federated learning tasks by providing optimal ML model for the task.