Plant leaf classification using neural networks

Václav Kobera

Project overview

The project focuses on classification of plants with different kinds of defects and diseases using open-source dataset available at https://data.mendeley.com/datasets/tywbtsjrjv/1. Since the dataset contains large collection (over 1000 samples for each category) of labeled images project focuses on solving this topic using CNN (Convolution neural networks), ViT (Vison Transformers) and Transfer Learning. The goal is to compare the performance and trade-offs with different approaches

Methodology

Dataset data selection

The dataset contains classes that I didn't find interesting to classify or some classes had images that will force network to optimise on not important traits of the image such as background.

- removed plants with only one class
- removed background class images
- removed all classes for corn

After filtration dataset had 29 classes to classify (from original 39) more info is in Jupyter notebook DatasetOverview.ipynb

Data Preprocessing and augmentation

The dataset images were already augmented so I didn’t augment them further each class has been divided into training (first 800) and validation (from 800 to 1000) if the class was bigger than that the images were ignored to keep each class same size for simpler evaluation later. Each image was preprocessed as follows:

- Resize to 128x128 or 256x256 depending on model input size
- Image normalisation to value range of [0, 1] by dividing each pixel value by 255
- Change array order from [W H C] to [C W H] and making tensor from it
- For each image adding Hot end vector with value 1 on index of classID

Each NN was trained over 20 epochs with different learning rate parameters, Adam optimiser and CrossEnropy loss.

Custom CNN

The custom architecture consists of two main stages: a feature extractor and a classification head.

Feature extractor:

- **Conv2d with 3x3 kernel** performing convolution operation with flipped kernel (Cross-Correlation) doubling the number of channels (on first iteration goes from 3 to 32 channels instead)
- **BetchNorm2d -** standardise layer inputs, stabilising the training process.
- **ReLU** - activation function introduces non-linearity to the model.
- **MaxPool with 2x2 kernel** downsampling the channel size (width and height)

This was repeated until the channel size was 16x16 pixels

Classification Head:

- **Flatten** - converting 3d feature maps to 1d vector
- **Linear fully connected layer** - 512 hidden neurons
- **BatchNorm1d** - batch normalisation
- **ReLU** - applies not linearity
- **(Optional) Dropout** - ignore some of the embeddings for final classification
- **Output Linear** layer from 512 units to **29 class logits**

CNN was trained with and without Dropout with input size of 128 and 256

Transfer Learning ResNet

For transfer learning I used ResNet50 from HuggingFace https://huggingface.co/microsoft/resnet-50

I froze feature extractor and used pretrained states.

I replaced classifier since the original has 1000 distinct classes but our dataset has only 29 and trained only the classifier layers in ResNet.

Custom Visual Transformer

The custom architecture contains patch embedding, positional embedding, classification token, attention and forward layers with skip connections

- **Patch embedding** - converts img to embeddings of 8x8 pixel size
- **Positiona embadding** - adding information of patch position in the image
- **Classification token** - token added to carry classification information
- **5 layers of:**
  - **MultiheadAttention** - allows the model to weigh the importance of different image parts
  - **SkipConnections from before MultiheadAttention** (adding values from before applying Attention)
  - **Forward**
    - **Normalisation**
    - **Linear full connected layer to 4x the size**
    - **ReLu**
    - **Dropout**
    - **Linear full connected layer from 4x the size back to original**
    - **Dropout**
  - **SiipConnections over Forward**

Implementation

Training and evaulation:

KeyParts:

- Dataset folder mapping to internal class representation
- Splitting dataset for training (fist 800 images) and validation (images indexed from 800 to 1000)
- Loading data from dataset normalisation of an image and adding class labels
- Training loop with loss and back propagation
- Evaulation of each training session
  - graphs of loss progressions
  - confusion matrix of best epoch
  - cases where model failed to classify most often
  - Comparing all training sessions

Each training session had own prepared script that runs for 20 epochs and collected data of Training and Validation loss and time elapsed

Tools:

- Python
  - Jupyter Notebook - for detailed reports
  - PyTorch, Einops - own model creation data loaders and training
  - OpenCV - image loading and resizing
  - Numpy - simple vector operation such as image normalisation etc
  - Pandas, Mathplotlib, Seaborn - Training evaluation and visualisation

GUI:
Gui is separated to 2 different parts

1. Where user inputs image and page ll load all modes results and user can compare different models

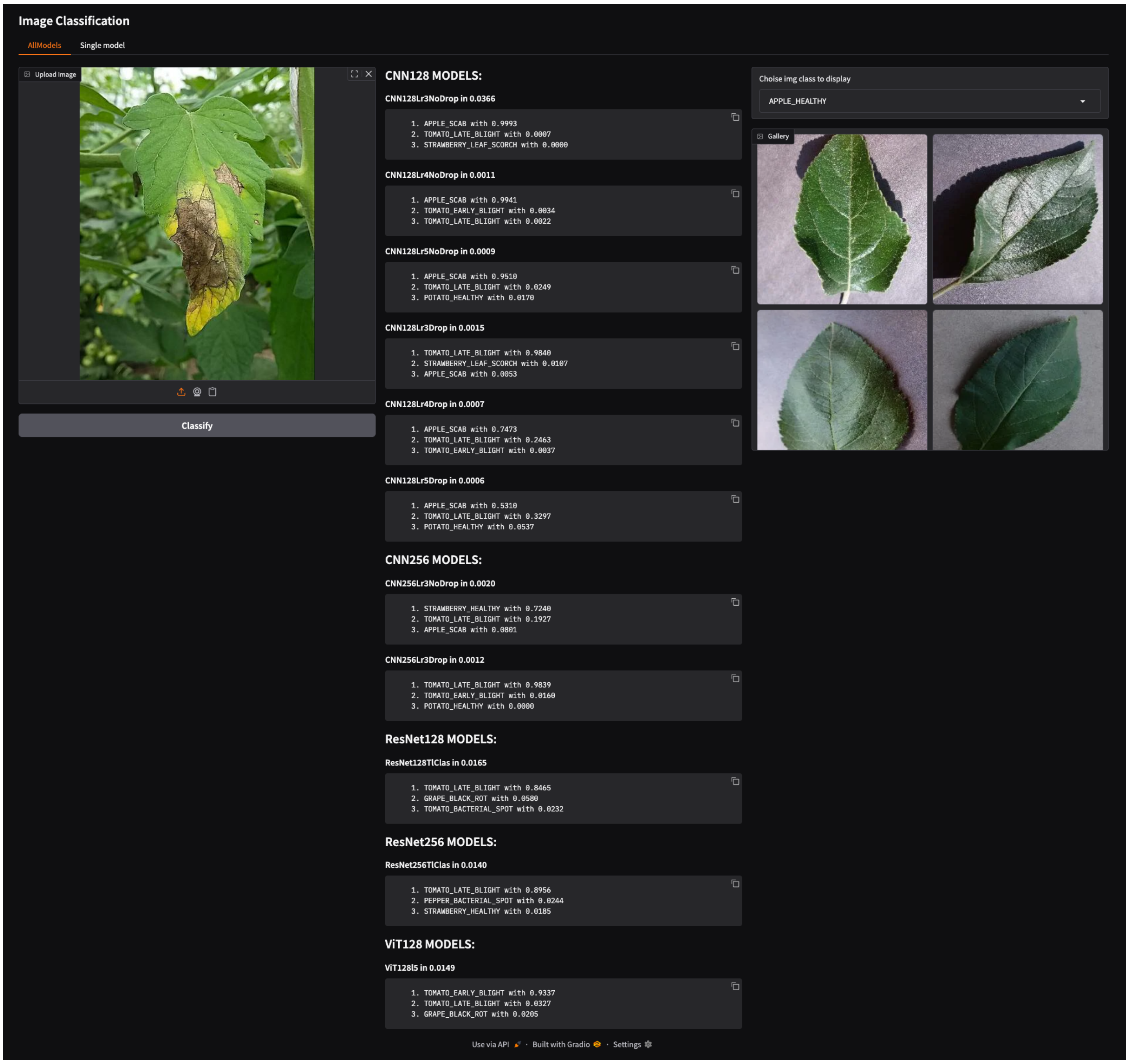
2. User input image and selects model and model shows top3 categories as well as shows user why the model selected this category (iterpretability)

Tools:

- Python
  - Gradio - Http UI framework
  - PIL - Images on UI
  - OpenCV - image resizing
  - Captum - Interpretability for CNN
  - https://github.com/jacobgil/vit-explain - ViT interpretability

GUI Showcase

All Models



CNN128 MODELS:

CNN128Lr3NoDrop in 0.0366

```
1. APPLE_SCAB with 0.9993
2. TOMATO_LATE_BLIGHT with 0.0007
3. STRAWBERRY_LEAF_SCORCH with 0.0000
```

CNN128Lr4NoDrop in 0.0011

```
1. APPLE_SCAB with 0.9941
2. TOMATO_EARLY_BLIGHT with 0.0034
3. TOMATO_LATE_BLIGHT with 0.0022
```

CNN128Lr5NoDrop in 0.0009

```
1. APPLE_SCAB with 0.9510
2. TOMATO_LATE_BLIGHT with 0.0249
3. POTATO_HEALTHY with 0.0170
```

CNN128Lr3Drop in 0.0015

```
1. TOMATO_LATE_BLIGHT with 0.9840
2. STRAWBERRY_LEAF_SCORCH with 0.0107
3. APPLE_SCAB with 0.0053
```

CNN128Lr4Drop in 0.0007

```
1. APPLE_SCAB with 0.7473
2. TOMATO_LATE_BLIGHT with 0.2463
3. TOMATO_EARLY_BLIGHT with 0.0037
```

CNN128Lr5Drop in 0.0006

```
1. APPLE_SCAB with 0.5310
2. TOMATO_LATE_BLIGHT with 0.3297
3. POTATO_HEALTHY with 0.0537
```

CNN256 MODELS:

CNN256Lr3NoDrop in 0.0020

```
1. STRAWBERRY_HEALTHY with 0.7240
2. TOMATO_LATE_BLIGHT with 0.1927
3. APPLE_SCAB with 0.0801
```

CNN256Lr3Drop in 0.0012

```
1. TOMATO_LATE_BLIGHT with 0.9839
2. TOMATO_EARLY_BLIGHT with 0.0160
3. POTATO_HEALTHY with 0.0000
```

ResNet128 MODELS:

ResNet128TlClas in 0.0165

```
1. TOMATO_LATE_BLIGHT with 0.8465
2. GRAPE_BLACK_ROT with 0.0580
3. TOMATO_BACTERIAL_SPOT with 0.0232
```

ResNet256 MODELS:

ResNet256TlClas in 0.0140

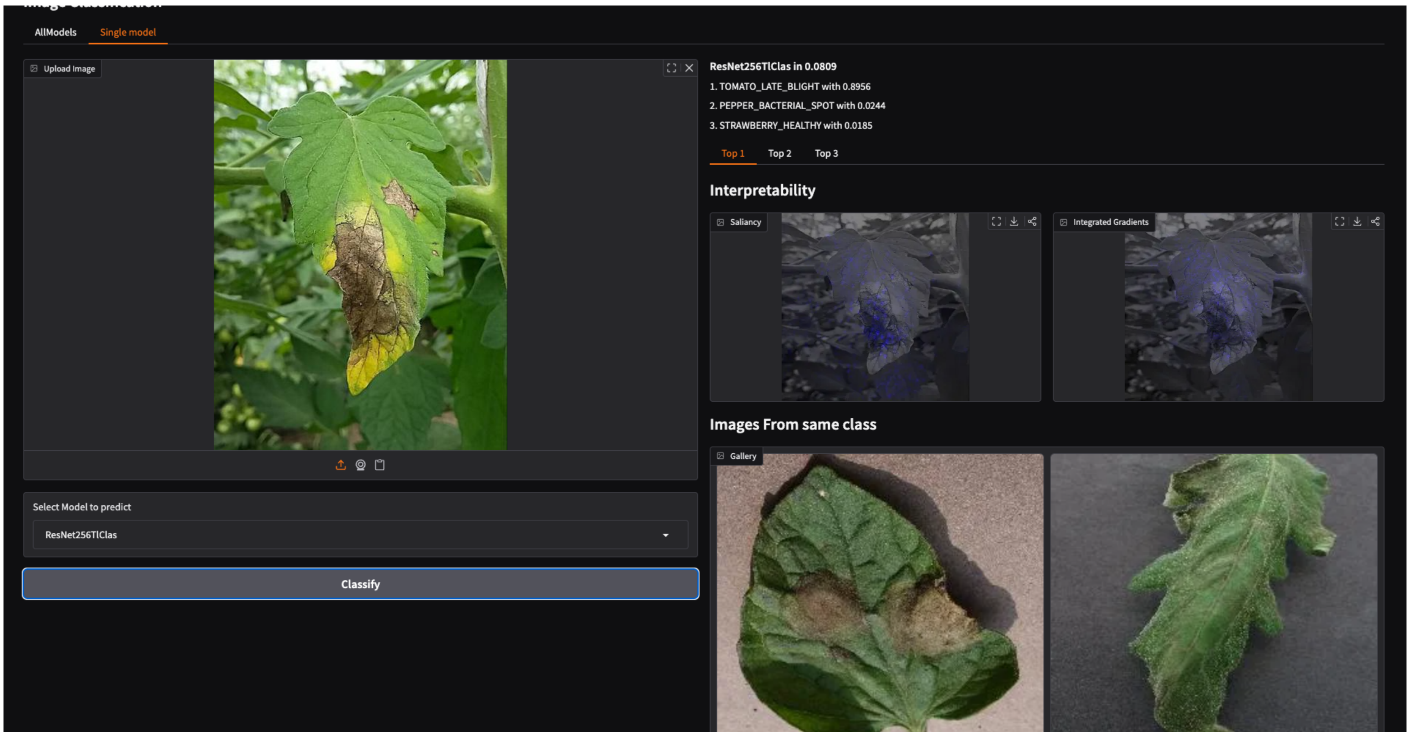
```
1. TOMATO_LATE_BLIGHT with 0.8956
2. PEPPER_BACTERIAL_SPOT with 0.0244
3. STRAWBERRY_HEALTHY with 0.0185
```

ViT128 MODELS:

ViT128l5 in 0.0149

```
1. TOMATO_EARLY_BLIGHT with 0.9337
2. TOMATO_LATE_BLIGHT with 0.0327
3. GRAPE_BLACK_ROT with 0.0205
```

Single Model



ResNet256TlClas in 0.0809

1. TOMATO_LATE_BLIGHT with 0.8956
2. PEPPER_BACTERIAL_SPOT with 0.0244
3. STRAWBERRY_HEALTHY with 0.0185

Interpretability

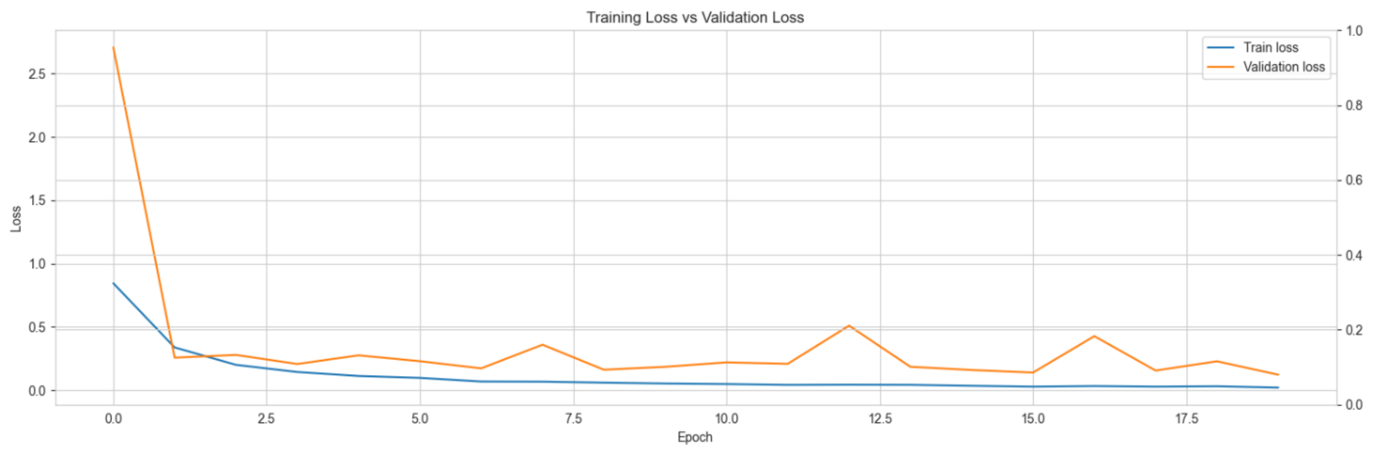
Images From same class

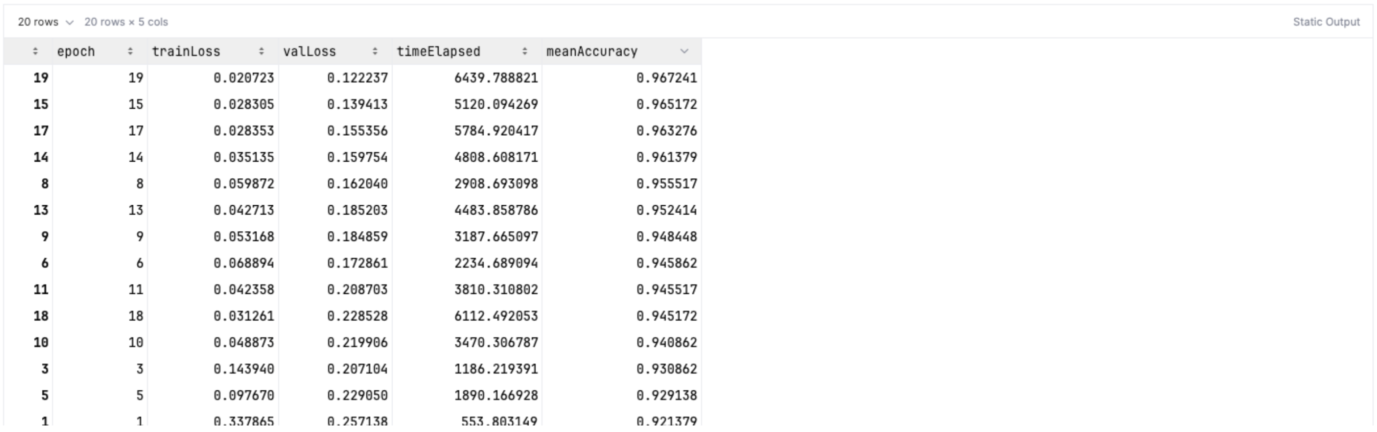
Evaluation

For each model I calculated mean accuracy for all classes in validation data for each epoch of training. Selected best mean accuracy across epochs and showed confusion matrix as well as worst classification classes for given model. Here I show such output for model CNN256Lr3Drop. But you can find all models evaluated in `modelEvulation.ipynb` notebook.

LossProgress and accuracy

```
max accuracy is in epoch 19
```



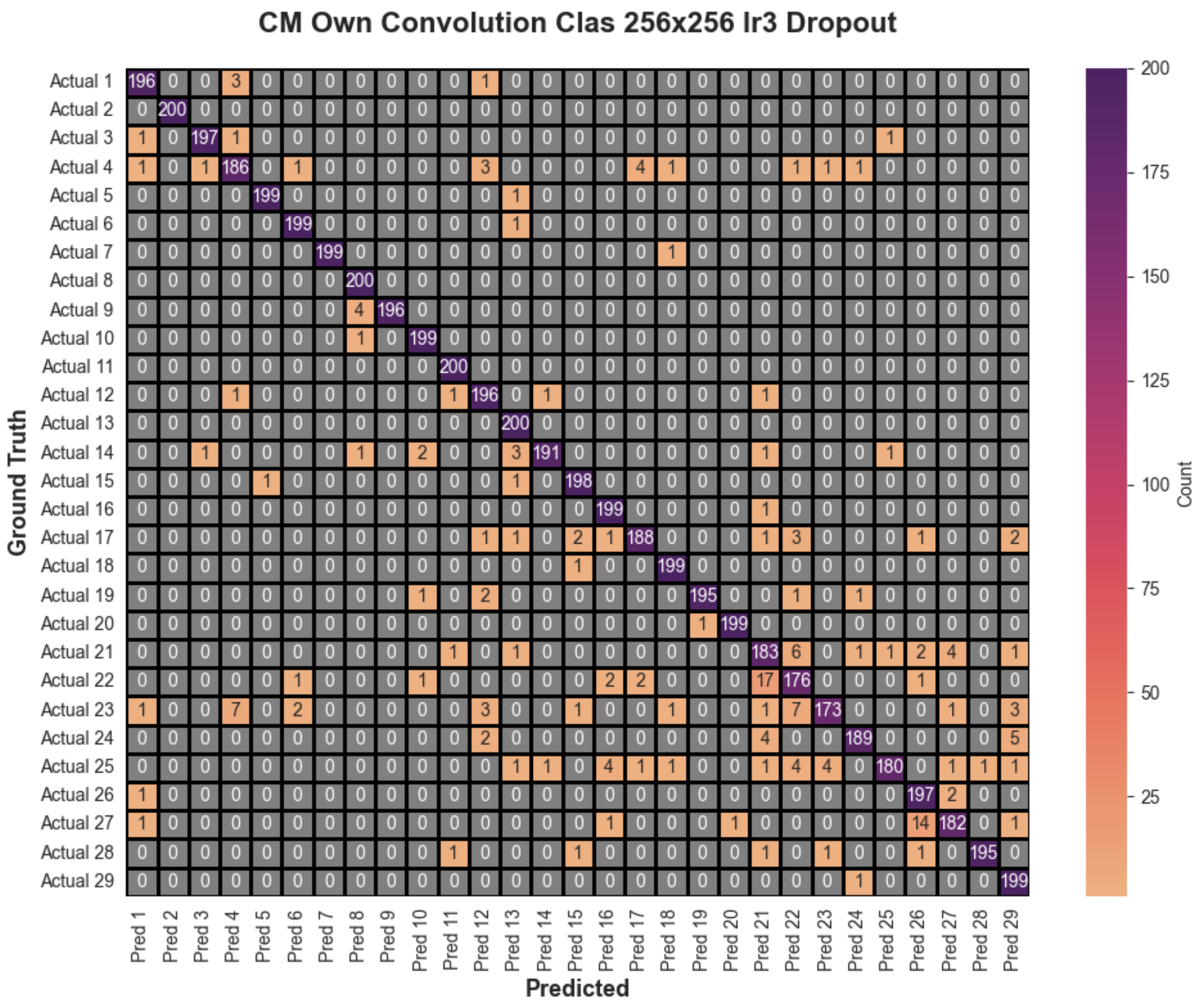


20 rows 20 rows × 5 cols Static Output

	epoch	trainLoss	valLoss	timeElapsed	meanAccuracy
19	19	0.020723	0.122237	6439.788821	0.967241
15	15	0.028305	0.139413	5120.094269	0.965172
17	17	0.028353	0.155356	5784.920417	0.963276
14	14	0.035135	0.159754	4808.608171	0.961379
8	8	0.059872	0.162040	2908.693098	0.955517
13	13	0.042713	0.185203	4483.858786	0.952414
9	9	0.053168	0.184859	3187.665097	0.948448
6	6	0.068894	0.172861	2234.689094	0.945862
11	11	0.042358	0.208703	3810.310802	0.945517
18	18	0.031261	0.228528	6112.492053	0.945172
10	10	0.048873	0.219906	3470.306787	0.940862
3	3	0.143940	0.207104	1186.219391	0.930862
5	5	0.097670	0.229050	1890.166928	0.929138
1	1	0.337865	0.257138	553.803149	0.921379

Confusion Matrix

ℹ️ `for labels see LeadDataset.py and its Claasses enum`



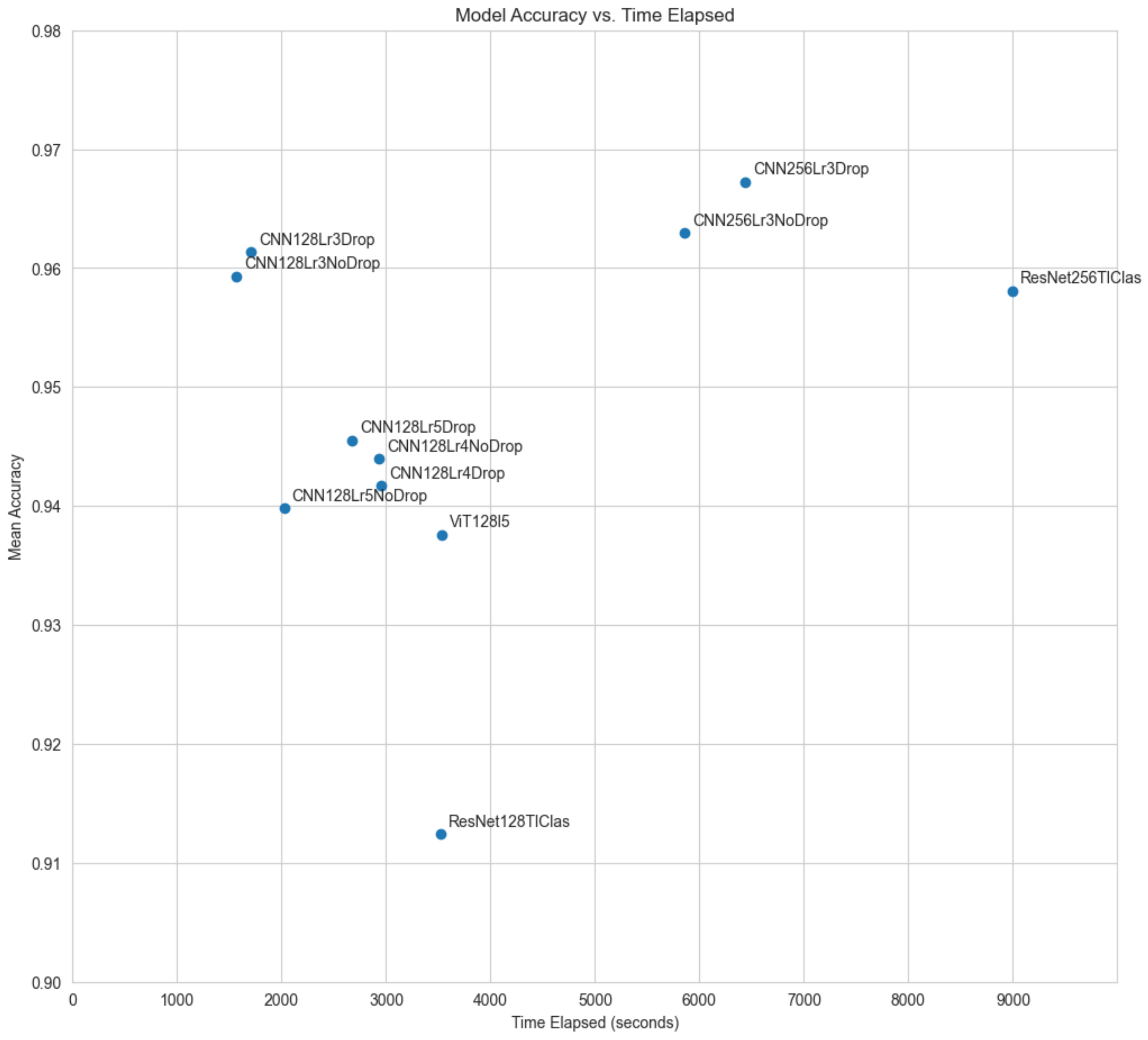
	Pred 1	Pred 2	Pred 3	Pred 4	Pred 5	Pred 6	Pred 7	Pred 8	Pred 9	Pred 10	Pred 11	Pred 12	Pred 13	Pred 14	Pred 15	Pred 16	Pred 17	Pred 18	Pred 19	Pred 20	Pred 21	Pred 22	Pred 23	Pred 24	Pred 25	Pred 26	Pred 27	Pred 28	Pred 29
Actual 1	196	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 2	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 3	1	0	197	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Actual 4	1	0	1	186	0	1	0	0	0	0	0	3	0	0	0	0	4	1	0	0	0	1	1	1	0	0	0	0	0
Actual 5	0	0	0	0	199	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 6	0	0	0	0	0	199	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 7	0	0	0	0	0	0	199	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Actual 8	0	0	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 9	0	0	0	0	0	0	0	4	196	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 10	0	0	0	0	0	0	0	1	0	199	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 11	0	0	0	0	0	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 12	0	0	0	1	0	0	0	0	0	0	1	196	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Actual 13	0	0	0	0	0	0	0	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 14	0	0	1	0	0	0	0	1	0	2	0	0	3	191	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
Actual 15	0	0	0	0	1	0	0	0	0	0	0	0	1	0	198	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	199	0	0	0	0	1	0	0	0	0	0	0	0	0
Actual 17	0	0	0	0	0	0	0	0	0	0	0	1	1	0	2	1	188	0	0	0	1	3	0	0	0	1	0	0	2
Actual 18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	199	0	0	0	0	0	0	0	0	0	0	0
Actual 19	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	195	0	0	1	0	1	0	0	0	0	0
Actual 20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	199	0	0	0	0	0	0	0	0	0
Actual 21	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	183	6	0	1	1	2	4	0	1
Actual 22	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	2	2	0	0	0	17	176	0	0	0	1	0	0	0
Actual 23	1	0	0	7	0	2	0	0	0	0	0	3	0	0	1	0	0	1	0	0	1	7	173	0	0	0	1	0	3
Actual 24	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	4	0	0	189	0	0	0	0	5
Actual 25	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	4	1	1	0	0	1	4	4	0	180	0	1	1	1
Actual 26	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	197	2	0	0
Actual 27	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	14	182	0	1
Actual 28	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0	1	0	195	0
Actual 29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	199

WorstClasses to classify:

```
worst 5 classes are:
[<Classes.TOMATO_LEAF_MOLD: 23>, 'accuracy: 0.865']
[<Classes.TOMATO_LATE_BLIGHT: 22>, 'accuracy: 0.88']
[<Classes.TOMATO_SEPTORIA_LEAF_SPOT: 25>, 'accuracy: 0.9']
[<Classes.TOMATO_TARGET_SPOT: 27>, 'accuracy: 0.91']
[<Classes.TOMATO_EARLY_BLIGHT: 21>, 'accuracy: 0.915']


GT: TOMATO_LEAF_MOLD but model predicated:
   - APPLE_SCAB in 3.5% cases
   - TOMATO_LATE_BLIGHT in 3.5% cases
   - TOMATO_YELLOW_LEAF_CURL_VIRUS in 1.5% cases
   - PEACH_BACTERIAL_SPOT in 1.5% cases
   - CHERRY_POWDERY_MILDEW in 1.0% cases

GT: TOMATO_LATE_BLIGHT but model predicated:
   - TOMATO_EARLY_BLIGHT in 8.5% cases
   - POTATO_EARLY_BLIGHT in 1.0% cases
   - POTATO_LATE_BLIGHT in 1.0% cases
   - GRAPE_LEAF_BLIGHT in 0.5% cases
   - TOMATO_SPIDER_MITE in 0.5% cases

GT: TOMATO_SEPTORIA_LEAF_SPOT but model predicated:
   - TOMATO_LEAF_MOLD in 2.0% cases
   - TOMATO_LATE_BLIGHT in 2.0% cases
   - POTATO_EARLY_BLIGHT in 2.0% cases
   - TOMATO_YELLOW_LEAF_CURL_VIRUS in 0.5% cases
   - TOMATO_MOSAIC_VIRUS in 0.5% cases

GT: TOMATO_TARGET_SPOT but model predicated:
   - TOMATO_SPIDER_MITE in 7.0% cases
   - TOMATO_YELLOW_LEAF_CURL_VIRUS in 0.5% cases
   - TOMATO_HEALTHY in 0.5% cases
   - POTATO_EARLY_BLIGHT in 0.5% cases
   - APPLE_HEALTHY in 0.5% cases

GT: TOMATO_EARLY_BLIGHT but model predicated:
   - TOMATO_LATE_BLIGHT in 3.0% cases
   - TOMATO_TARGET_SPOT in 2.0% cases
   - TOMATO_SPIDER_MITE in 1.0% cases
   - TOMATO_YELLOW_LEAF_CURL_VIRUS in 0.5% cases
   - TOMATO_SEPTORIA_LEAF_SPOT in 0.5% cases
```

Then I compared all the models together in single graph comparing time to train and achieved accuracy.

Note: that all CNN128 models should have simular times to train but I had to train some of them on different device since I run out of memory while training (**all CNN128 should be around 1600 seconds**). All other models are trained on same device.



Discussion

This project was just evaluated on the given dataset. Interesting will be to test the models on real user input data and see how it compares. Because the dataset it kinda limited and maybe some classes are classified correctly but not with the real features that experts will use to classify (maybe the background of glass etc.).

Another interesting point would be training the models for longer period than just 20 epochs with early stopping or other techniques to get full potential of the models. We used 20 epochs hard limit mainly to reduce training time to test more settings and ideas.

Also we could try to transfere learn ResNet or other open model with training some of the extractor layers (but not on ResNet50 on smaller model like ResNet18).

I was surprised how good my Visual Transformer performed since these models usually need way more data since they are missing the locality focus that CNN have.

Interesting is also that the own CNN are performing well on the dataset while being not very deep. And outperforming ResNet50 transfer learned freezing feature extractor.

Takeaways

In this project we trained multiple visual models. We evaluated them and compared them in time to train and accuracy on the dataset. We found that own simple CNN can perform better than transfer learning big model.

We also created GUI with interpretability so user can see what were main focus points of image classification for given model “why the model thinks its this class”.