T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

DETECTION AND CLASSIFICATION OF POLLEN GRAIN MICROSCOPE IMAGES

FATIH KAAN SALGIR

SUPERVISOR
ERCHAN APTOULA

GEBZE
2022

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

DETECTION AND CLASSIFICATION OF POLLEN GRAIN MICROSCOPE IMAGES

FATIH KAAN SALGIR

SUPERVISOR
ERCHAN APTOULA

2022
GEBZE

GRADUATION PROJECT
JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 16/06/2022 by the following jury.

JURY

Member
(Supervisor) : Erchan Aptoula

Member : Yakup Genc

Member : Erkan Zergeroglu

ABSTRACT

Pollen recognition has been shown to be important for a number of aresas. However these palynology studies rely on highly qualified professionals to analyze pollen grains, which have become costly.

The objective is to develop a deep computer vision system that given the image of a polen acquired through an optical miscroscope, will recognize and detect the polens. The larger aim is to be able to brand honey through the polens present in it. The classification of pollens will be according to both their species and the family.

Deep learning has proven to be the ultimate technique in computer vision, but requires large dataset to train deep networks from scratch. This project focuses the classification and localization of pollen images through convolutional neural networks, transfer learning, and data augmentation.

ACKNOWLEDGEMENT

For his important suggestions and his lead I am greateful to my supervisor Prof. Dr. Erchan Aptoula. Also, I thank the researchers in Bingol University for the pollen dataset and pollen videos they have provided.

Fatih Kaan SALGIR

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
CNN	:	Convolutional Neural Network
mAP	:	Mean Average Precision
IoU	:	Intersection over Union
SSD	:	Single Shot Detector
R-CNN	:	Region Based Convolutional Neural Networks
TF	:	Tensorflow

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In this project, it is aimed to develop a deep computer vision system that given the image of pollen acquired through an optical microscope, will recognize and detect the pollens. The classification of pollens will be according to both their species and the family. The larger aim is to be able to brand honey through the pollens present in it. Besides certification of honey production, pollen grain recognition is used to trace relations between different groups of plants and their evolutionary lines, to establish the location or seasonal time frame for crime scenes, to determine agricultural practices occurring at archaeological sites.

1.1. Related Works

Most of the studies uses self-collected datasets. One of the common dataset used in other works is Pollen23E dataset [1]. It consists of 23 types of pollen and 805 images.

In Deep Convolutional Neural Network for Pollen Grains Classification [2], the proposed method was feature extraction using ReLU non linearities convolutional calculation, where first 5 are convolutional neurons responsible for image representations. Then the dimensionality of those obtained features were reduced using pooling method to get vectors of size 2048 element, which each vector represents an image in the dataset. After getting the vectors, they were used as input to 3 fully connected layers for image classification. The accuracy was about 85% in Pollen23E dataset.

The authors in [3] use Local Binary Pattern which is more robust to noise and rotation of pollen images. The dataset they have used consists of 389 images with 26 classes.

The study in [4] is not about classifying pollen types but their development cycles into 5 groups, such as well-developed pollen grain or anomalous pollen grain. They also propose a segmentation pipeline, which aims to partition an image by maximizing the pollen grains detection alongside reducing the detection of dust and artifacts. The segmentation process starts with, the pre-processing by reducing the background noise. In the segmentation pipeline, the image is transformed to grayscale, several morphological operators applied, flood fill algorithm applied, and connected components smaller than a threshold are removed.

In the Large-scale Pollen Recognition with Deep Learning [5], it is experimented on three types of convolutional neural network training which are, training from scratch, fine-tuning of pre-trained network, and feature extraction. In transfer learning they have experimented with InceptionV3, ResNet and DenseNet architectures trained on the 2012

ImageNet image data set which consists of 1000 classes. The best result is achieved with DensNet with 96% of accuracy. In the datset they have proposed, there are 134 types where each class consists of about 30 samples. The results show that feature learning produced far more accurate results than pre-designed features. Training from scratch produced inferior results.

1.2. Success Criteria

- The system will reach at least 80 % accuracy in terms of classifiying pollen images.
- Given an image with pollens, bounding boxes should be drawn, outlining the object at least 0.7 mAP at 0.5 IoU.
- Pollen recognition system should be able to run on live video stream at 15 FPS on Nvidia RTX2080TI.

2. POLLEN GRAIN CLASSIFICATION

2.1. Dataset

Most of the results are obtained by self-collected datasets, and many authors used images from a scanning electron microscope (SEM). So they are quite different than the dataset I have provided.

During the project multiple datasets are provided. First experiments are performed using first version of the dataset. Second version of the datset includes new images as well as all of the images from the first version. The result shown in the report are obtained using second version of the dataset. In the dataset there are 74 classes and 1258 images in total. The resolution of an image is 2456×1842 px. The partition ratio used in experiments are 70% to training, 15% to validation and 15% to test set. The distribution of the sample counts per class and partition ratio can be seen in the figure 2.1.

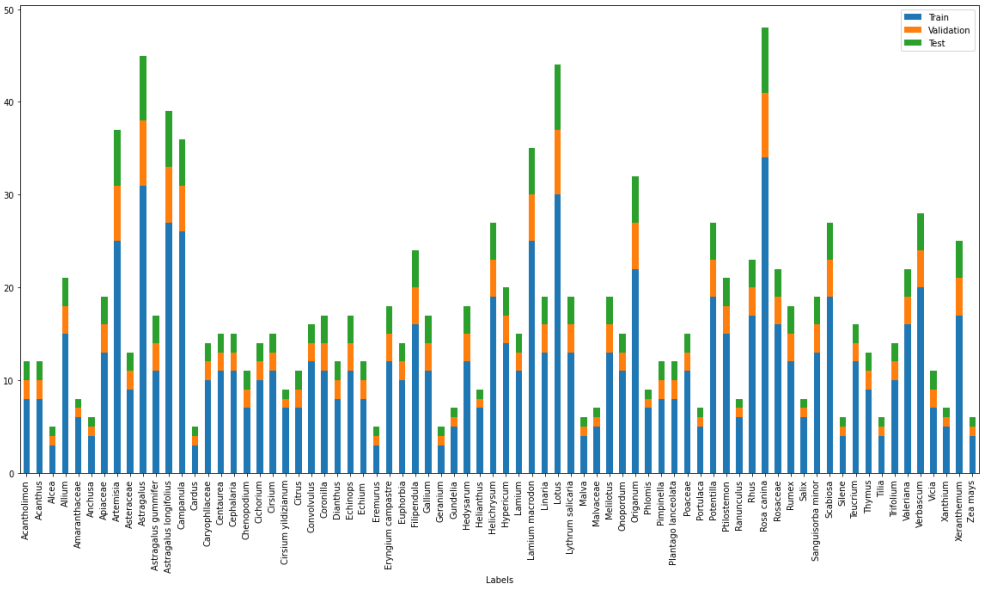


Figure 2.1: Dataset V2 - Partitioned

The dataset is limited, and has high intra-class variance and inter-class similarity. To deal with this problem data augmentation techniques are used.

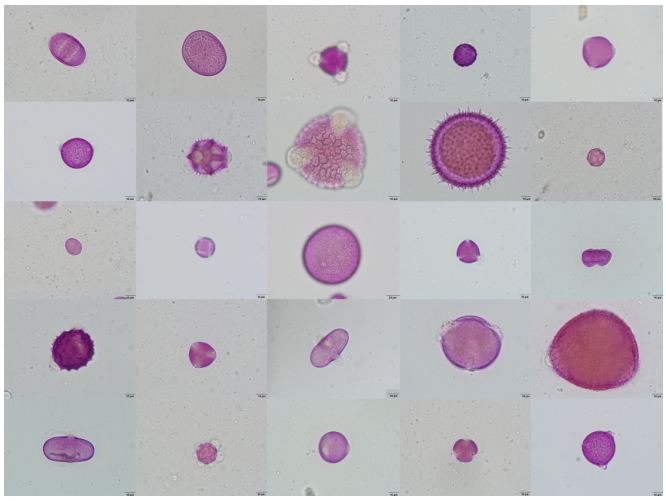


Figure 2.2: Some of the microscope images from the dataset

2.2. Working Environment

For the classification, I have mostly experimented with Google Colab connected to Google machines. In training and evaluating object detection models, I have used Google Colab connected to local Nvidia RTX2080TI environment with Tensorflow-2.3 and CUDA 10.1 installed. The experiments are mostly performed using Tensorflow and Keras in Python.

2.3. Data Augmentation

Image augmentation is a way to randomly apply transformations to input images, introducing extra variance in the training dataset. Extra variance, in turn, leads to better model generalization, which is essential for good performance. According to problem domain augmentation methods such as rotation, translation, change in the brightness and contrast are helpful. Most of the images in dataset is obtained by using a fixed microscope scale. To be able to use the size information of pollens, augmentation methods that result in changes in scaling are avoided.

2.4. Classification

Due to the highly imbalanced dataset accuracy itself is not a good performance metric. First experiments with default configurations show that the model is biased to the majority sampled classes. To solve class imbalance problem class weights are calculated using `n_samples/(n_classes*np.bincount(y))`. The balanced class

weight implementation is taken from sklearn's compute class weight function which is inspired by Logistic Regression in Rare Events Data [6].

Class weights change the range of the loss, this may affect the stability of the training depending on the optimizer. The preferred optimizer (Adam), in experiments, is unaffected by the scaling change.

The custom split method is implemented to partition the dataset into the train, validation, and test sets, since Tensorflow split API does not perform well for classes with a small number of samples.

2.4.1. Training from Scratch

In order to compare different methods and architectures, a baseline model is implemented. The baseline CNN model consists of 3 convolution blocks with a max-pooling layer in each of them, and a fully connected layer with 128 units, that is activated by ReLU activation function. Before feeding images to the network I have rescaled them to 400 by 300 pixels, keeping a fixed aspect ratio. The model is trained for 200 epochs with Adam optimizer with a learning rate 10^{-3} According to test results, the model reaches about 74% classification accuracy.

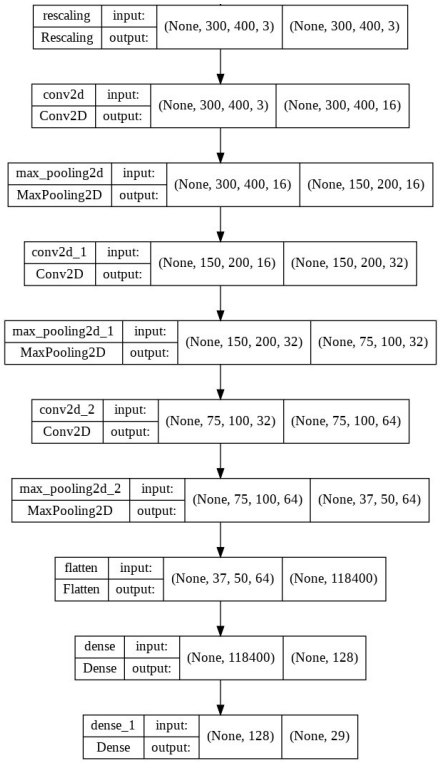


Figure 2.3: Basic CNN Architecture

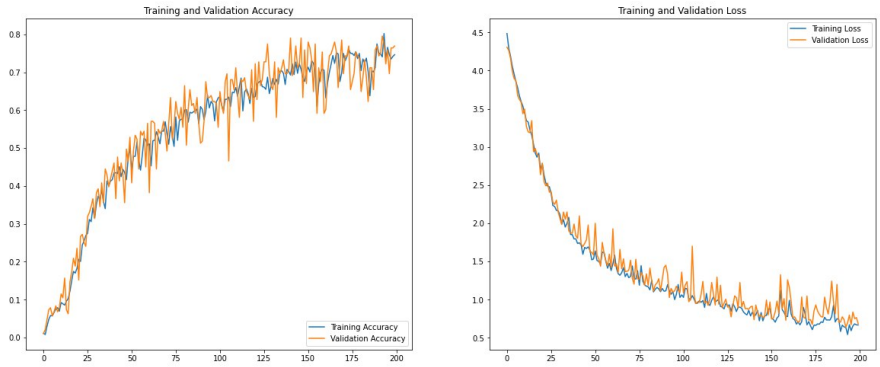


Figure 2.4: Basic CNN: Training and Validation Accuracy

2.4.2. Transfer Learning & Fine Tuning

In transfer learning, I have used MobileNet V2 as the base model. It is pre-trained on ImageNet, a large dataset consisting of 1.4 million images and 1000 classes. I have frozen the convolutional base and added a regular densely-connected classification head. In this setup, there are 2.3 million parameters in the base model frozen and 95 thousand trainable parameters in the dense layer. After training for 150 epochs with 0.0001 base learning rate and Adam optimizer, the model reached up to 72% validation accuracy. In the figure 2.5 this step corresponds to the left side of the learning curve. Starting of fine-tuning step is marked with a green line. To increase performance even further and to fine-tune the weights, the top 56 layers of the pre-trained model unfroze and trained for another 150 epochs alongside the classifier. In fine-tuning step, RMSprop optimizer is used with 10 times decreased learning rate to avoid overfitting. After fine-tuning the model has reached 90% classification accuracy on the test set.

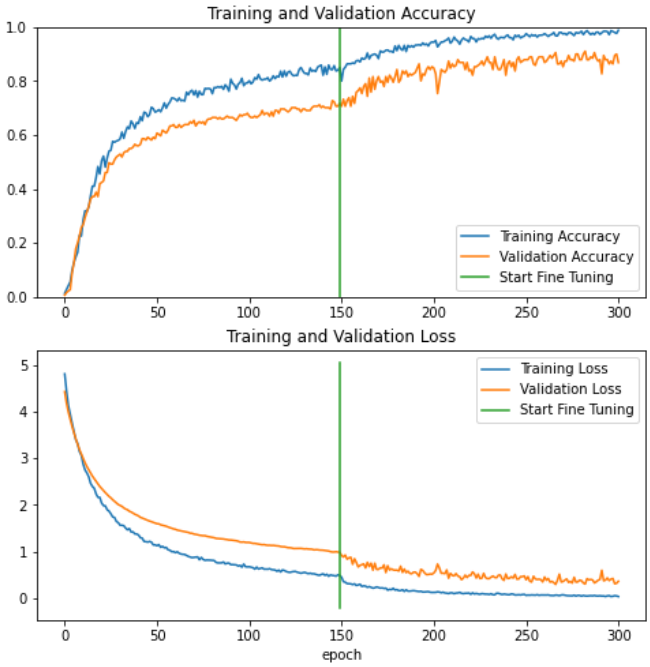


Figure 2.5: Transfer Learning: Training and Validation Accuracy

2.5. Object Detection

In object detection, the aim is to be able to draw bounding boxes surrounding detected pollen grains. To train such a model from scratch would take a long time and require huge amounts of input data. Instead, transfer learning methods experiment with Tensorflow Object Detection API.

There are two different object detection task; for the individual species, and the family they belong to. The pre-trained models are useful for initializing models when training on novel datasets. For the detection of individual species, the following model architectures are used.

- SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
- Faster R-CNN ResNet50 V1 1024x1024
- EfficientDet D1 640x640

The models are pre-trained on the COCO 2017 dataset. After experimenting on these pre-trained networks, the best result is obtained by EfficientDet D1. For the model that detects families, the training job is initialized with boty EfficientDet D1 and SSD ResNet50.

2.5.1. Data Preparation

To train object detection models Dataset V2 is used. The pollen images in the dataset are annotated, and bounding boxes and ground truth labels are set using LabelImg tool. Since we can identify families by looking at the species, pollen labels are set using only species. Annotations are saved as XML files in PASCAL VOC format. Models based on the TensorFlow object detection API need a special format for all input data, called TFRecord. TensorFlow also requires a label map pbtxt file, which maps each of the used labels to integer values. This label map is used both in the training and detection processes. For the first model, there are 74 entries in the label map, each corresponding to a species. For the second one, there are 31 entries on the label map, each corresponding to a family of species. For 2 different models, we need two label maps and two TF record's generated using those label maps. In order to know which species belong to which family, a yaml file is created where keys are the family names and values are the species belonging to that family. This file is used to map each species to a family and generate family record files from pollen images and annotation files.

2.5.2. TF Record

The 70% of the dataset is used to train the model, 15% to validate, and 15% to test. The partition of the dataset is done by a custom script that iterates dataset directories and copies the specified amount of pollen images along with the xml annotations to separate directories. For two different tasks, the same split is used. Therefore their relative performance doesn't depend on the dataset. Separate train, validation, and test record files are generated using the partitioned dataset, label maps, and the file that maps species to families.

Until this point, the process is common for separate models. The steps after this point are automated and saved in notebook files, which include downloading the model, setting up the file structure for the training environment, configuring the number of classes and input paths, and testing the model.

2.5.3. Training Job & Model Configuration

The TensorFlow Object Detection API uses protobuf files to configure the training and evaluation process. The pre-trained models allow model configuration via pipeline.config protobuf file. The config file is split into 5 parts.

- The `model` configuration that defines what type of model will be trained (ie.

meta-architecture, feature extractor).

- The `train_config`, which decides what parameters should be used to train model parameters (ie. SGD parameters, input preprocessing and feature extractor initialization values).
- The `eval_config`, which determines what set of metrics will be reported for evaluation.
- The `train_input_config`, which defines what dataset the model should be trained on.
- The `eval_input_config`, which defines what dataset the model will be evaluated on.

2.5.4. Training

Basic configurations is the the same for all of the models.

- `num_classes`: Exact number of class that the model is going to detect. For the first models this will be 74 and 31 for the second.
- `batch_size`: Batch size set depending on how much memory available. This set to 4 in training.
- `fine_tune_checkpoint`: Checkpoint of the pre-trained model. Set to checkpoint to corresponding pre-trained model.
- `label_map_path`: Generated label map pbtxt file. There are two different label maps. Path is set according to related task (species or families).
- `train_input_reader.input_path`: Train record file. Configured according to related task.
- `eval_input_reader.input_path`: Validation record file. Configured according to related task.

To use mean average precision and recall, the metrics set parameter in the evaluation config set to `coco_detection_metrics`. mAP is calculated for different IOU values.

The validation process is configured to run in CPU parallel to model training. The job waits for checkpoints that the training job generates during model training and uses them to validate the model on a separate dataset.

2.5.4.1. Baseline Model

The first model (SSD ResNet50 V1 FPN) is configured with the minimum required set of parameters to get a baseline result. The minimum required parameters are dataset and label paths, number of classes, and batch size. The classification scores are placed in table 2.1 with the baseline tag, to show how effective the configurations are in other models.

2.5.4.2. Parameter Tuning

In pre-processing, the following augmentation methods from Tensorflow API are used in training:

- `random_horizontal_flip`
- `random_vertical_flip`
- `random_adjust_hue`
- `random_adjust_contrast`
- `random_adjust_saturation`
- `random_adjust_brightness`

To better avoid overfitting cosine learning rate decay is used that controls the learning rate over time. Starting with a low learning rate gives control over gradients at the beginning of training. It is important to save the weights of the original model. The model is fine-tuned on a custom dataset, there's no need to change low-level features that the neural net has already learned. An initial increase in learning rate helps the model to have enough capacity not to get stuck in a local minimum. Smooth learning rate decay over time will lead to stable training, and will also let the model find the best possible fit for the data.

```
learning_rate {
cosine_decay_learning_rate {
  learning_rate_base: 0.04
  total_steps: 25000
  warmup_learning_rate: 0.013333
  warmup_steps: 2000
}
```

Listing 2.1: SSD ResNet50 Learning Rate Configuration

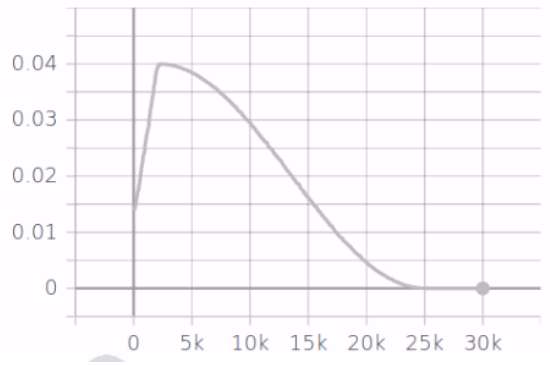


Figure 2.6: SSD Cosine Decay Learning Rate over 30k iterations

According to results obtained by the baseline SSD ResNet50 model, the model is good at localizing objects but performs quite poorly in classification. So in order to get a better classification, the weight of the classification loss is increased.

```
second_stage_localization_loss_weight: 1.0
second_stage_classification_loss_weight: 1.2
```

Listing 2.2: Faster R-CNN Localization & Classification Loss Weights

Increasing the classification loss helped balance localization and classification in Faster R-CNN, however, it is required to get better classification results. Thus classification weight is even prioritized over localization weight in EfficentDet D1.

```
classification_weight: 1.3
localization_weight: 0.8
```

Listing 2.3: EfficentDet D1 Localization & Classification Loss Weights

EfficentDet D1's default pipeline configuration uses focal loss as classification loss function. According to RetinaNet paper [7], focal loss is very useful for classification when the dataset is highly imbalanced. It down-weights well-classified examples and focuses on hard examples. The loss value is much higher for a sample that is misclassified by the classifier as compared to the loss value corresponding to a well-classified example.

```
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid_focal {
      gamma: 1.5
```

```
      alpha: 0.25
    }
  }
}
```

Listing 2.4: EfficentDet D1 Loss Function

Another parameter to configure is the anchor generation. In anchor generation configuration list of multiple aspect ratios of bounding boxes can be defined. Adjusting the anchors to be specific to the dataset can both improve model accuracy and reduce training time. Pollen images are usually circular shaped, therefore their bounding boxes are close to square, so the only aspect ratio registered is 1.0. Depending on the model there are different anchor generators. Faster R-CNN uses grid anchor generator of first stage anchor generator. SSD models use multiscale anchor generator.

```
first_stage_anchor_generator {
  grid_anchor_generator {
    height_stride: 16
    width_stride: 16
    scales: 0.25
    scales: 0.5
    scales: 1.0
    scales: 2.0
    aspect_ratios: 1.0
  }
}
```

Listing 2.5: Faster R-CNN Anchor Generator

```
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    scales_per_octave: 3
  }
}
```

Listing 2.6: SSD Anchor Generator

Post-processing configuration uses non-maximum suppression (NMS). `iou_threshold` is a parameter that lets NMS make proper filtering for overlapping boxes. Since the dataset doesn't have a dense distribution of objects on an image, set to default. `max_detections_per_class` sets how many objects expected for every single class. In the training set, there is only single class for each image however, in the videos, there

are about 6-7 images in a frame. `max_total_detections` defines how many objects expected.

```
post_processing {
  batch_non_max_suppression {
    score_threshold: 1e-08
    iou_threshold: 0.5
    max_detections_per_class: 20
    max_total_detections: 100
  }
  score_converter: SIGMOID
}
```

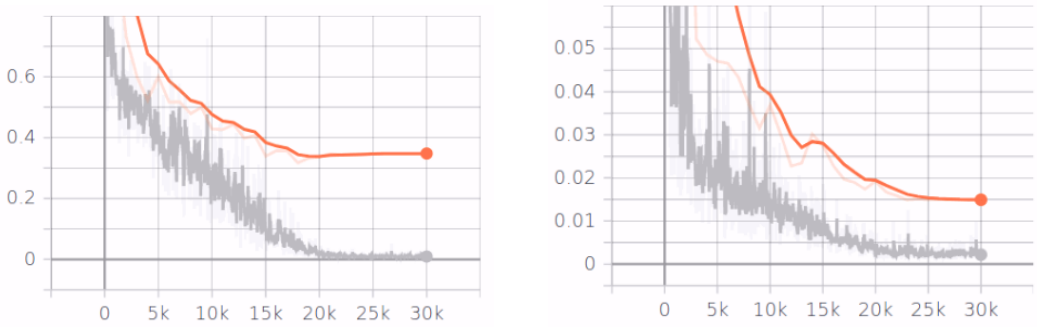
Listing 2.7: SSD Non Max Suppression Configuration

2.5.4.3. Training & Evaluation Results

Classification result in the test set is calculated by comparing maximum scored detection and real class name. The values in table 2.1 is weighted averages of class distrubutions.

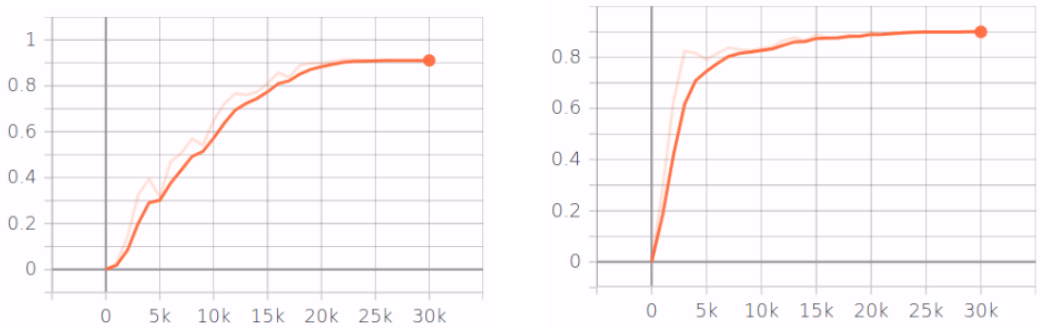
Table 2.1: Classification scores of object detection models

	Accuracy	Precision	Recall	F1 Score
SSD ResNet50 V1 FPN (Baseline)	0.46	0.39	0.46	0.38
Faster R-CNN ResNet50	0.63	0.65	0.64	0.59
EfficentDet D1	0.82	0.84	0.82	0.80
SSD ResNet50 V1 FPN	0.86	0.87	0.86	0.85
SSD ResNet50 V1 FPN (Family)	0.62	0.63	0.62	0.61
EfficentDet D1 (Family)	0.88	0.87	0.88	0.87



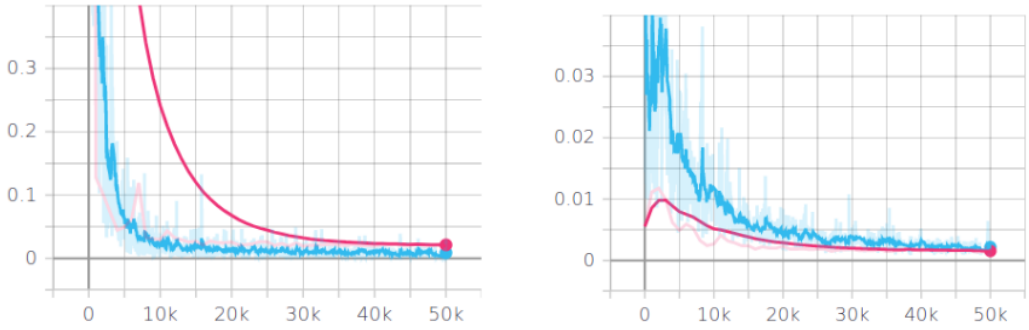
Classification Loss Localization Loss

Figure 2.7: SSD ResNet 50 V1 Loss



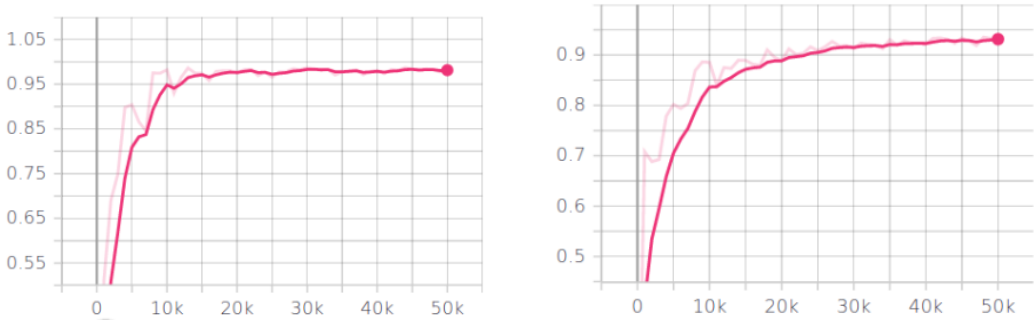
Precision - mAP@.75IOU Average Recall with 1 Detection

Figure 2.8: SSD ResNet50 V1 Detection Boxes Precision/Recall



Classification Loss Localization Loss

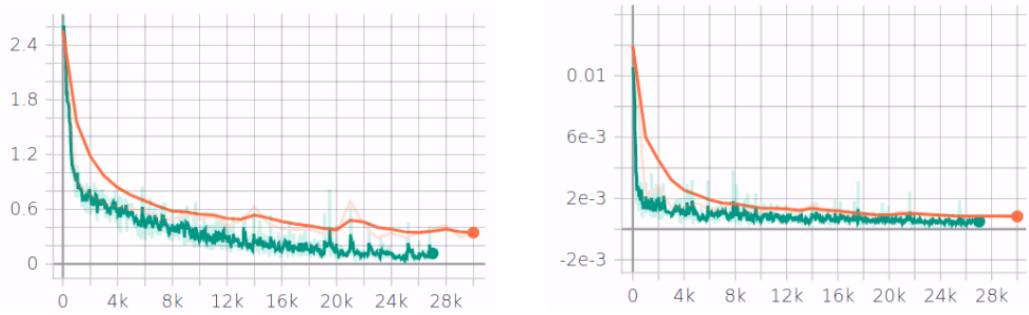
Figure 2.9: Faster R-CNN Loss



Precision - mAP@.75IOU

Average Recall with 1 Detection

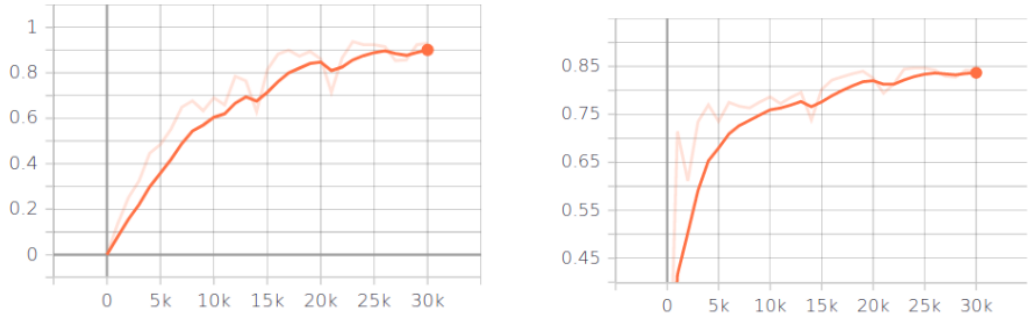
Figure 2.10: Faster R-CNN Detection Boxes Precision/Recall



Classification Loss

Localization Loss

Figure 2.11: EfficentDet D1 Loss



Precision - mAP@.75IOU

Average Recall with 1 Detection

Figure 2.12: EfficentDet D1 Detection Boxes Precision/Recall

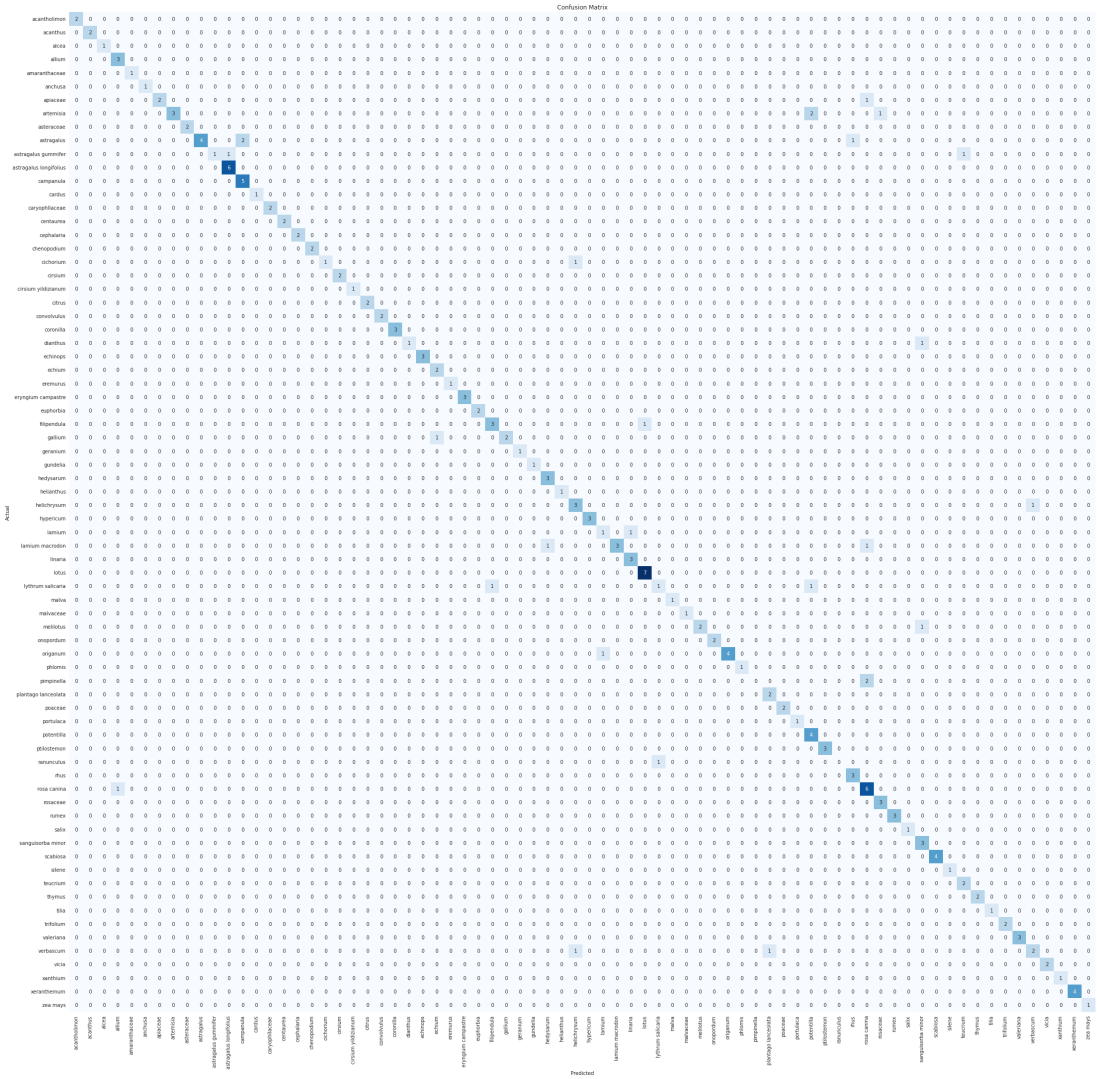


Figure 2.13: SSD ResNet50 V1 Confusion Matrix - Species Level Detection

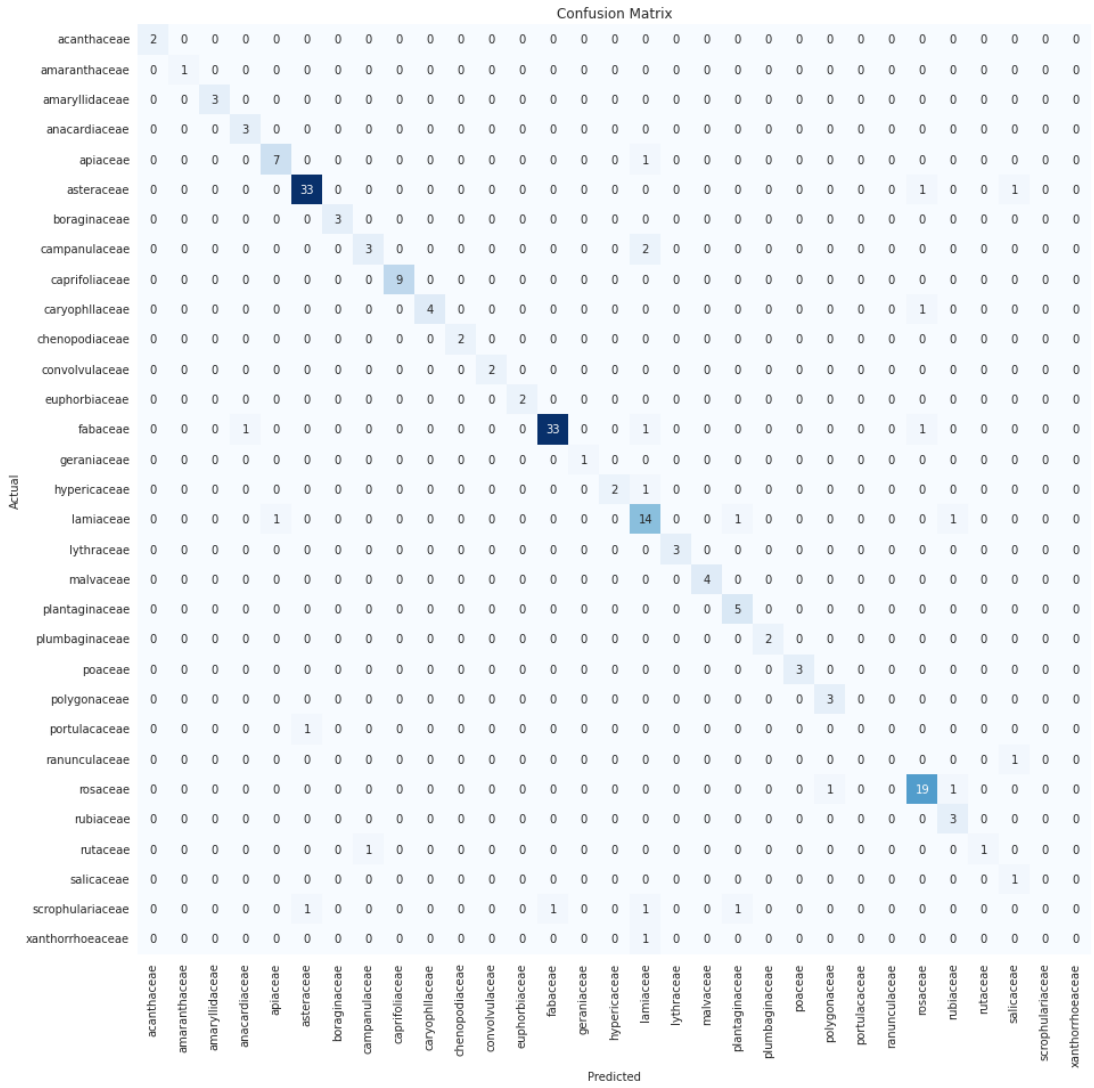


Figure 2.14: EfficentDet D1 Confusion Matrix - Family Level Detection

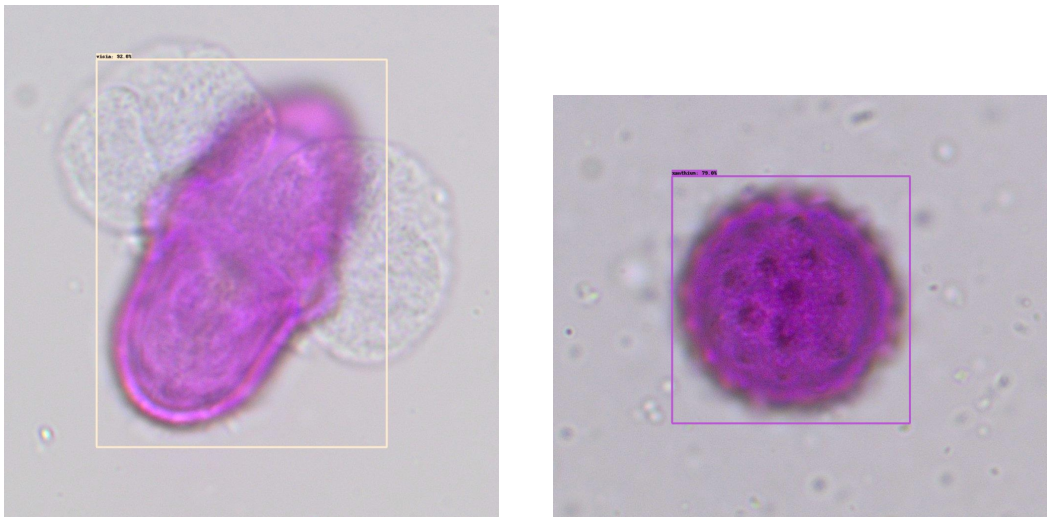


Figure 2.15: Object Detection Demonstration

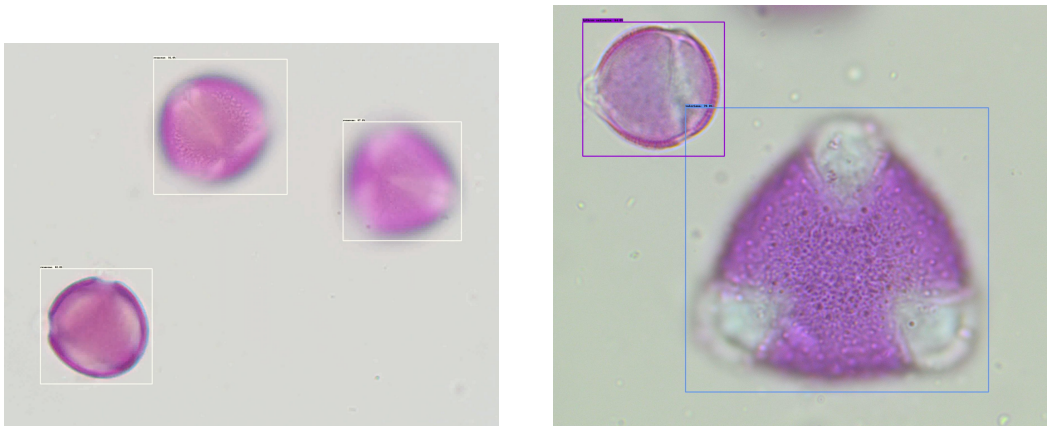


Figure 2.16: Object Detection Demonstration - Multiple Pollen Grains

2.5.5. Evaluating Success Criteria

According to MobileNetV2 results, the model is achieved up to 90% classification accuracy. The localization with classification results reached about 0.8 mAP at 0.75 IoU. Object detection in the video stream is tested with a custom setup that captures 360 frames and calculates estimated frames per second. The estimated FPS of SSD ResNet50 V1 architecture is 8.6 FPS on Nvidia RTX2080Ti with power limit set to 125W/250W. Therefore the first two success criteria are met, and the third is partially met.

3. CONCLUSIONS

In this project I have experimented on classification of pollen images, different data augmentation techniques, and investigated two ways of training CNNs: from scratch and transfer learning.

In classification, training from scratch produced about 74% accuracy, while transfer learning and fine-tuning with MobileNetV2 reached up to 90% accuracy.

In object detection three different architectures are via transfer learning methods. Considering the CNN architectures I found the better result with SSD ResNet50 in species level classification achieving up to 86% accuracy, and EfficentDet D1 in family level classification, achieving up to 88% of accuracy.

BIBLIOGRAPHY

[1] A. B. Gonçalves, J. S. Souza, G. G. da Silva, *et al.*, "Polen23e: Image dataset for the brazilian savannah pollen types," [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0157044.

[2] H. Menad, F. Ben-Naoum, and A. Amine, "Deep convolutional neural network for pollen grains classification," in *JERI*, 2019.

[3] H. Yin, Y. Chen, J. Xiong, R. Xia, J. Xie, and K. Yang, "An improved local binary pattern method for pollen image classification and recognition," *Computers Electrical Engineering*, vol. 90, p. 106 983, 2021, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2021.106983. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621000161.

[4] S. Battiato, A. Ortis, F. Trenta, L. Ascari, M. Politi, and C. Siniscalco, "Detection and classification of pollen grain microscope images," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 4220–4227. DOI: 10.1109/CVPRW50498.2020.00498.

[5] A. R. d. Geus, C. A. Barcelos, M. A. Batista, and S. F. d. Silva, "Large-scale pollen recognition with deep learning," in *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019, pp. 1–5. DOI: 10.23919/EUSIPCO.2019.8902735.

[6] G. King and L. Zeng, "Logistic regression in rare events data," *Political Analysis*, vol. 9, pp. 137–163, 2001.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.