UNIVERSITEIT
STELLENBOSCH
UNIVERSITY

Deep Learning for Food Security?

Automatic Plant Disease Recognition

Undergraduate Thesis

Lydia de Lange

18350070

Study leader: Dr. HA Engelbrecht

Overview

Overview

- Problem
- Solution Criteria
- Vision
- Design Choices
- Hyperparameter tuner
- Experiments
- Conclusion and Recommendations

Problem & Solution

Problem

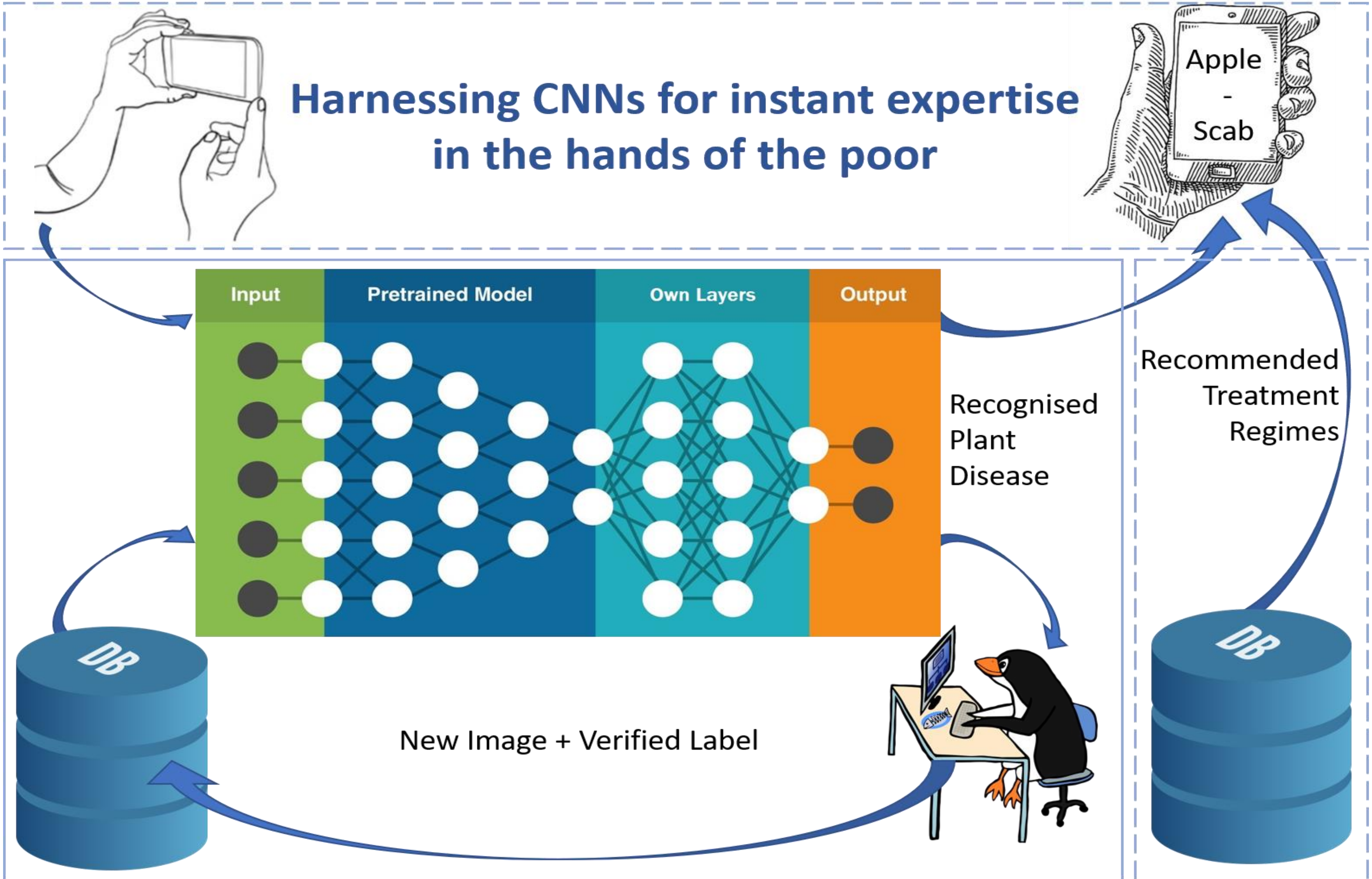
Investigate automatic classification to identify plant diseases from images.

- correct and timely treatments
- reduce chance of
  - financial losses,
  - crop losses and
  - environmental damage.
- Future work - smartphone application
  - commercial farmers
  - rural households

Solution Criteria

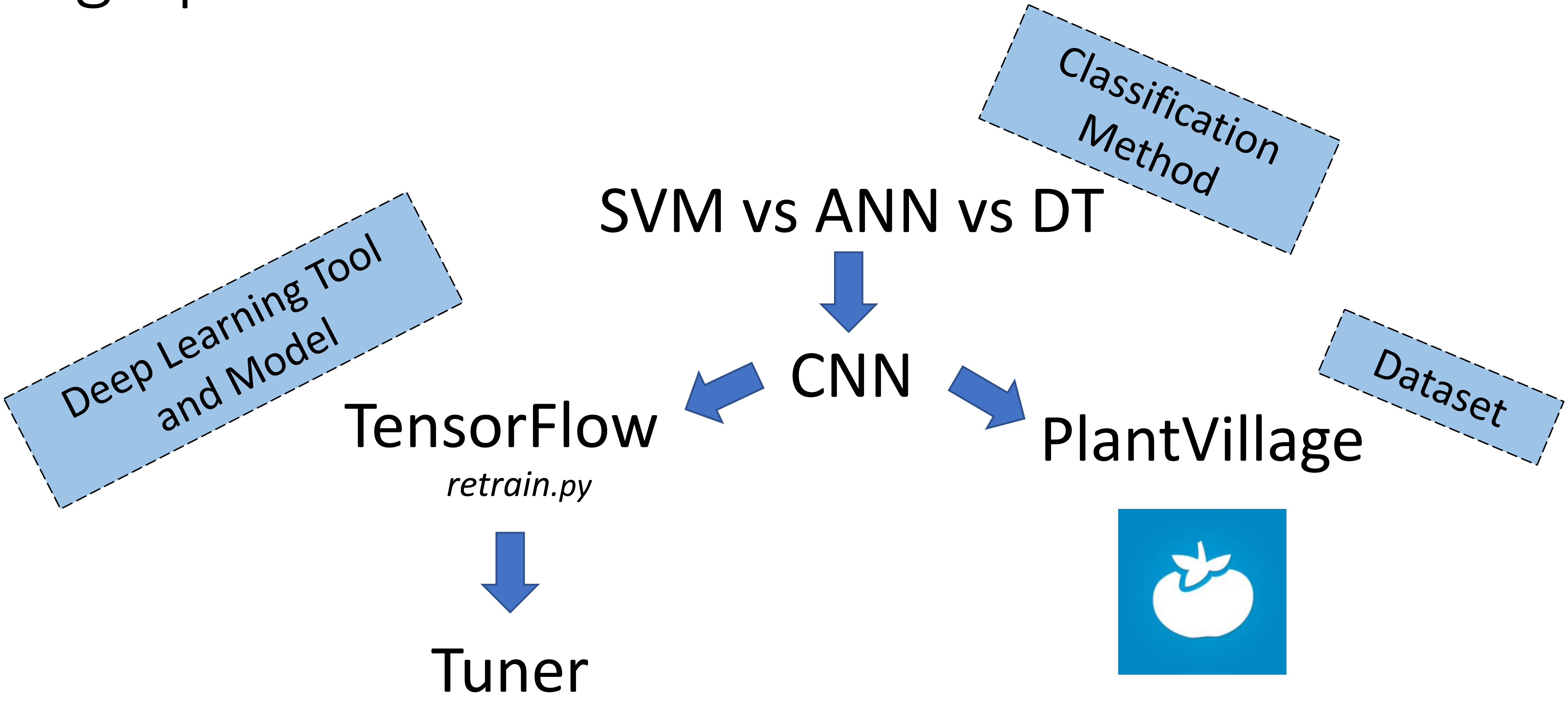
- Minimise classification error
- Comprehensive (Include many crop types and diseases)
- Expandable (add crop types and diseases)
- Suitable for smartphone app
- Identify diseases from plant leaf images

Vision:
Deep Learning for Food Security


Harnessing CNNs for instant expertise
in the hands of the poor
Apple
-
Scab
Input
Pretrained Model
Own Layers
Output
Recognised
Plant
Disease
Recommended
Treatment
Regimes
DB
New Image + Verified Label
DB

Design Choices

Design process



Dataset

- PlantVillage dataset
- 54 306 images
- 14 crop species
- 26 diseases

➔ 38 classes

Tuner

Optimise accuracy of a model by tuning specified hyperparameter values

Objectives:

- Tuning algorithm,
- Change program flow:
  - Intercept hyperparameter values,
  - Access final accuracy of the trained model,
- Set ranges, and
- Stop tuner at any time.

Model

TensorFlow's *retrain.py:*

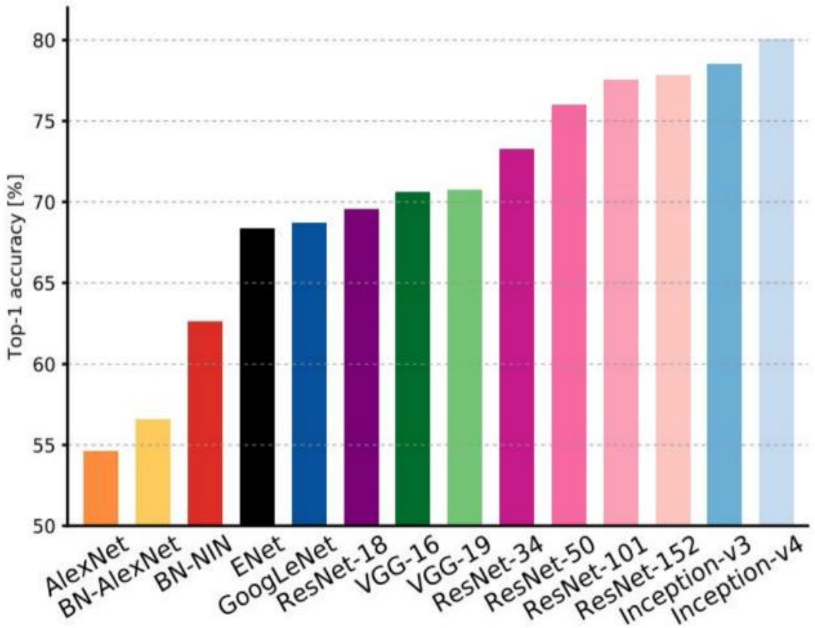
✓Transfer learning

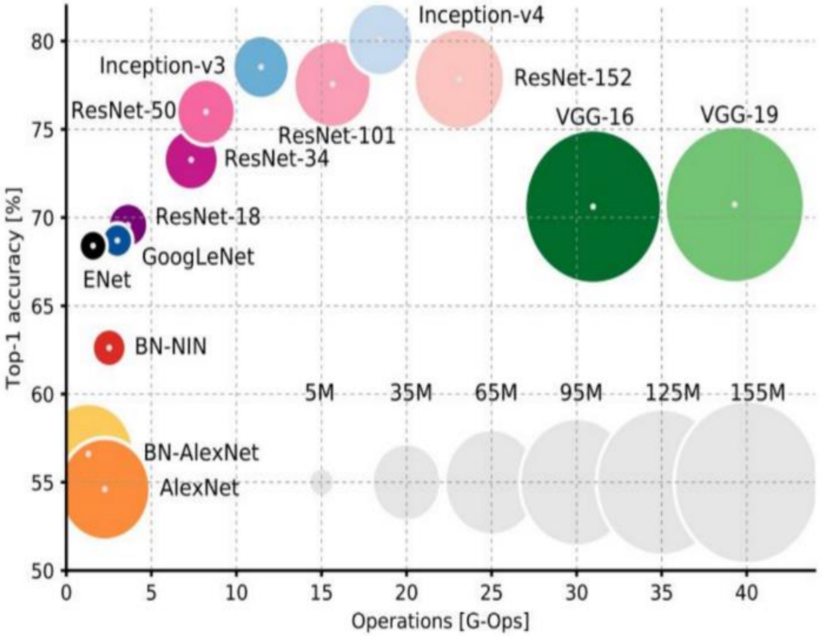
✓Augmentations

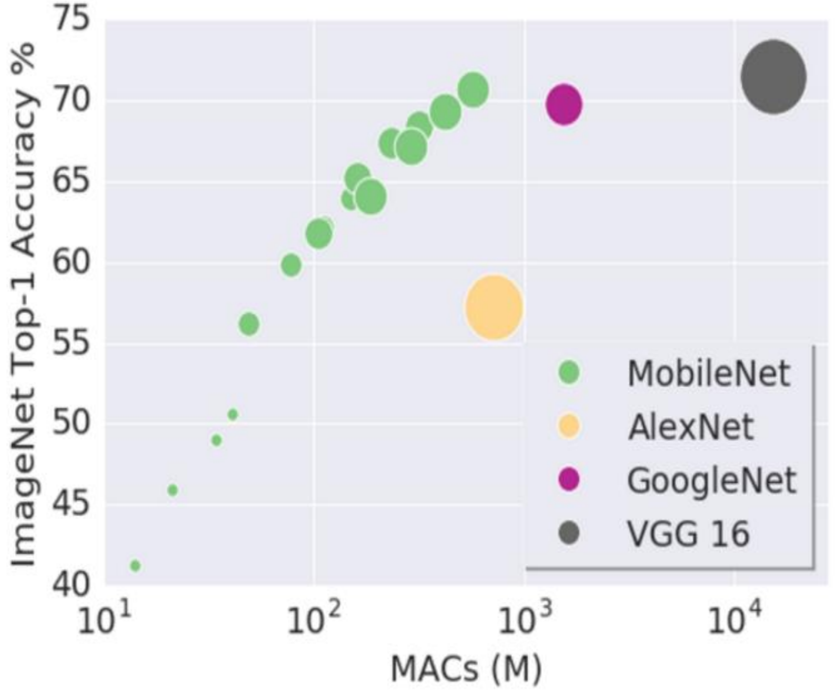
✓bottlenecks

Architectures:

- Inception v3, or
- One of 32 MobileNets







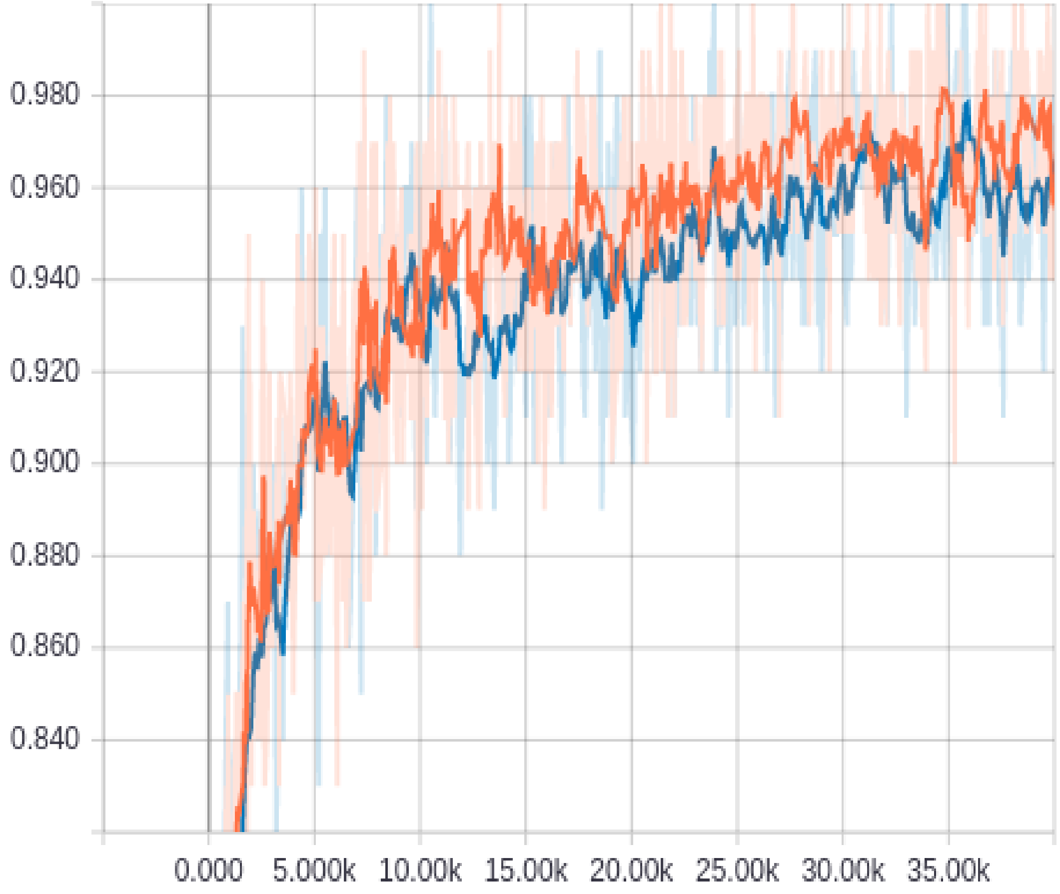
Experiments

Experiments - Objectives

- Evaluate default model;
- Viability of using hyperparameter tuner;
- Effect of augmentations;
- Inter-dependency between hyperparameters;
- Best hyperparameter combination;
- MobileNet vs Inception V3 architecture;
- Effects of MobileNet settings:
  - parameter size,
  - image size and
  - quantized; and
- Present optimised solution.

Default model

- Objectives
  - Provide baseline
  - Evaluation criteria
  - Dataset properties
- Conclusions
  - High accuracies
  - Room for improvement (Mohanty et al achieved 99.35%)
  - Trained fast
  - Reproducible
  - No overfitting



Steps	**600**	**1 500**	**4 000**	**10 000**	**20 000**	**50 000**
Accuracy	77,3	83,5	89,3	92,5	94,0	95,6
#	7	12	1	2	4	5

Tune individual parameters

- Conclusions
  - Hyperparameter tuner is successful, finds reliable results.
  - Augmentations on this dataset aren’t recommended, only increases training time.

#	Name	Training Steps	Best Found Value	Best Tuner Accuracy	Validated Accuracy	Improvement on default (%)
26	learning_rate	1500	0.33314	95.466	94.6	12.534 %
25	train_batch_size	1500	2231	84.2	84.0	0.831 %
24	flip_left_right	600	False	77.1	76.9	-0.259 %
13	random_crop	1500	5	84.0	84.0	0.595 %
15	random_scale	600	10	77.9	77.2	0.770 %
17	random_brightness	600	48	78.3	77.9	1.277 %

Tune parameters together

- Inter-dependency
  - hyperparameters are interdependent,
  - achieve better accuracies when trained together
- Accuracy and scalability
  - Configuration 1 = best accuracy
  - Both configurations show signs of overtraining at 4000 steps
- Unexpectedly large values
  - Too perfect training data

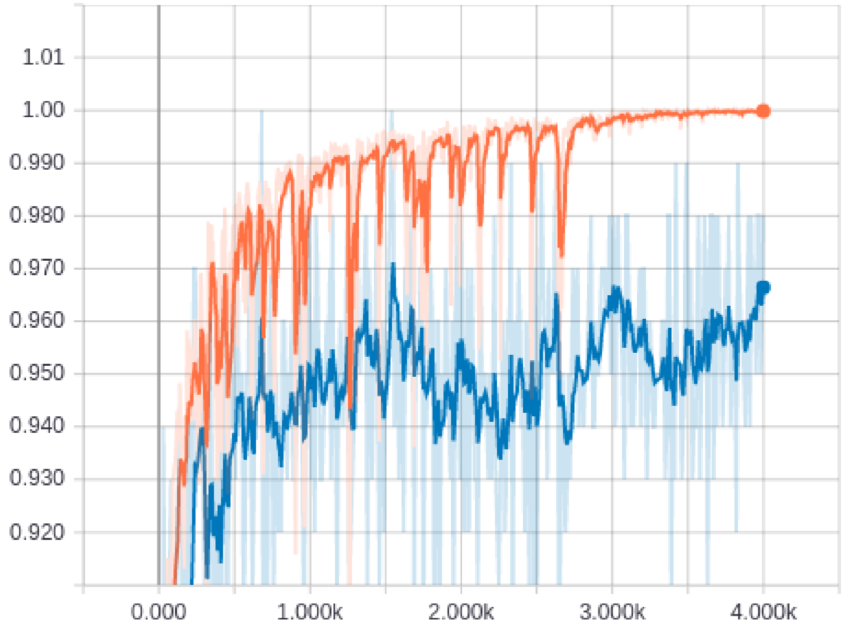
#	Configuration	Name	Value	Accuracy	Improvement on default (%)
-	1: Trained together	learning_rate	0.57758	96.098	13.110
		train_batch_size	4124		
26	2: Trained individually	learning_rate	0.33314	95.466	12.534
25		train_batch_size	2231	84.2	0.831

#	Configuration	Accuracy	Improvement on default (%)
27	Configuration 1	96.6	7.557
28	Configuration 2	92.6	3.564

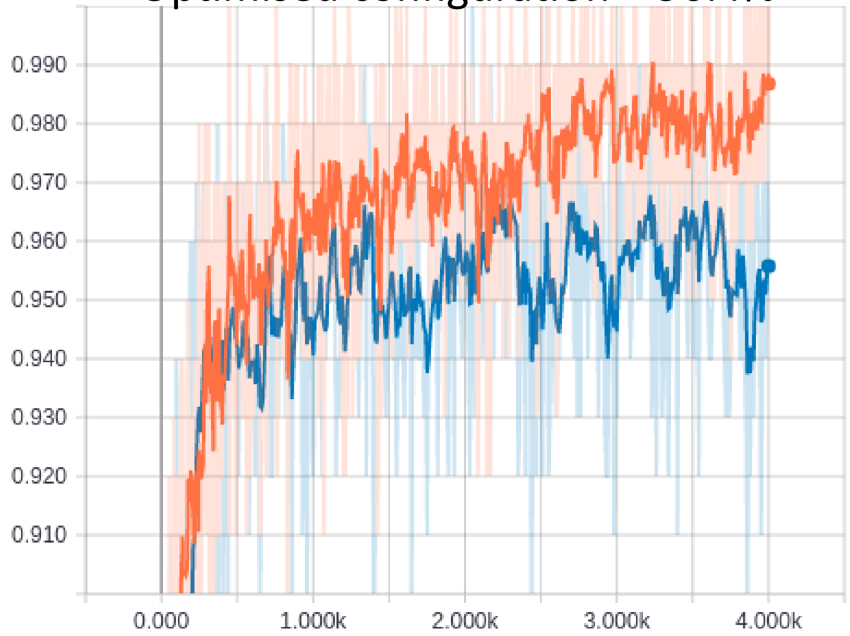
MobileNets

- Compare MobileNet to Inception v3, 4000 training steps
  - MobileNet trained much faster
  - hyperparameter configuration optimised for Inception v3 did not transfer well
  - Both overtrained
  - MobileNet has less parameters to train than Inception v3, so an optimal configuration might be easier to reach in fewer training steps

Default configuration – 96.6%



Optimised configuration – 96.4%



Effect of MobileNet settings

- Any decrease in settings results in faster training time but worse accuracy
- **Parameter size**: 0.828% linear decrease in accuracy for each 0.25 step down
- **Resolution of the input images**: 0.4% decrease in accuracy for each 32 pixels step down
- **Quantized**: 0.5% decrease in accuracy

#	Parameter Size	Input Size	Quantized	Accuracy	Decrease in accuracy (%)
29	1.0	224	No	96.6	0.000
31	0.75	224	No	95.8	0.828
32	0.50	224	No	95.0	1.656
33	0.25	224	No	93.4	3.313
36	1.0	192	No	96.3	0.311
34	1.0	128	No	95.4	1.242
37	1.0	224	Yes	96.1	0.518

Conclusion

Conclusion

- recommend mobilenet_1.0_244 architecture with the default model parameters and 4000 training steps.
- 96.6 % accuracy on held out test set.
- most accurate,
- trained much faster,
- used far less parameters and computational resources,
- suitable for smartphone application,
- easy to improve or alter using the MobileNet architecture settings.

Challenges

- comprehensiveness and expandability of the dataset solution criteria not yet satisfied
- 38 classes of crop and disease types is still too limited to be truly useful
- Data still “too perfect”
- Data exists, but it is not freely available to the public

PlantVillage Dataset – “too perfect”

Future Work

- Vision and larger project
  - Use tuner on MobileNets
  - Use for smartphone application or add to existing local software such as FarmBoek
  - Alliances with agricultural companies to acquire more data
- Hyperparameter tuner
  - Design to minimise training time, validation loss and accuracy
  - To attempt to solve problem of overfitting

The End

Additional Information

Shortcomings

- too perfect data,
- too little data and too few classes;
- unclear way to find good hyperparameters; and
- no free way for the public to identify a plant’s disease and find a treatment.

Dataset

DATASET ACQUISITION METHOD	SIZE	PRICE	ACQUISITION EFFORT	TIME INVESTED
COMPILE OWN	Limited set	Free	Coding and expertise	3 months or more
BUY	Extensive set	Expensive	Minimal	Minimal
FIND ONLINE	Reasonable set	Free	Browsing	Unknown

Image Classification Method

Methods

- Support Vector Models (SVM)
- Artificial Neural Networks (ANN)
- Decision Trees (DT)

CRITERIA	SVM	ANN	DT
ACCURACY	Excellent	Excellent	Excellent
MODEL TRAINING TIME	Poor	Poor	Good
REPRODUCIBILITY	Excellent	Excellent	Good
ROBUSTNESS TO NOISE IN TRAINING DATA	Poor	Good	Terrible
USE OF COMPUTATIONAL RESOURCES	Poor	Poor	Good
SOLUTION CRITERIA	Terrible	Excellent	Terrible

Deep Learning Tool

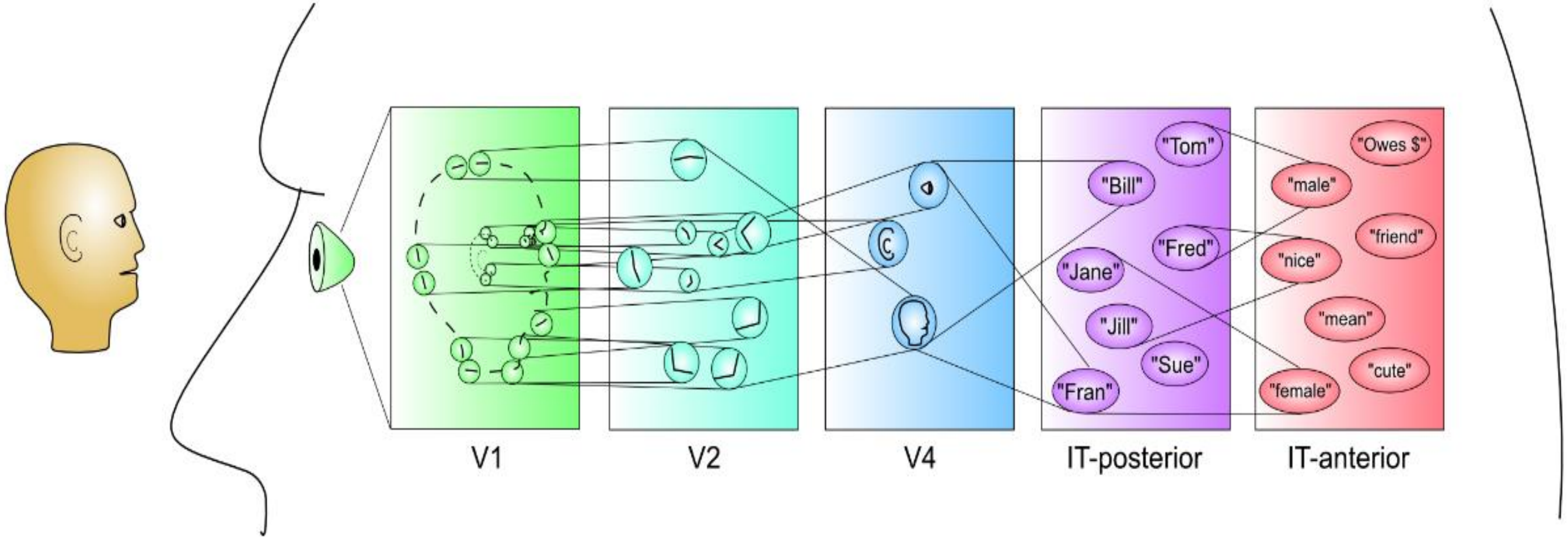
- Tensorflow
- Caffe
- H2O
- MXNet

Tensorflow:

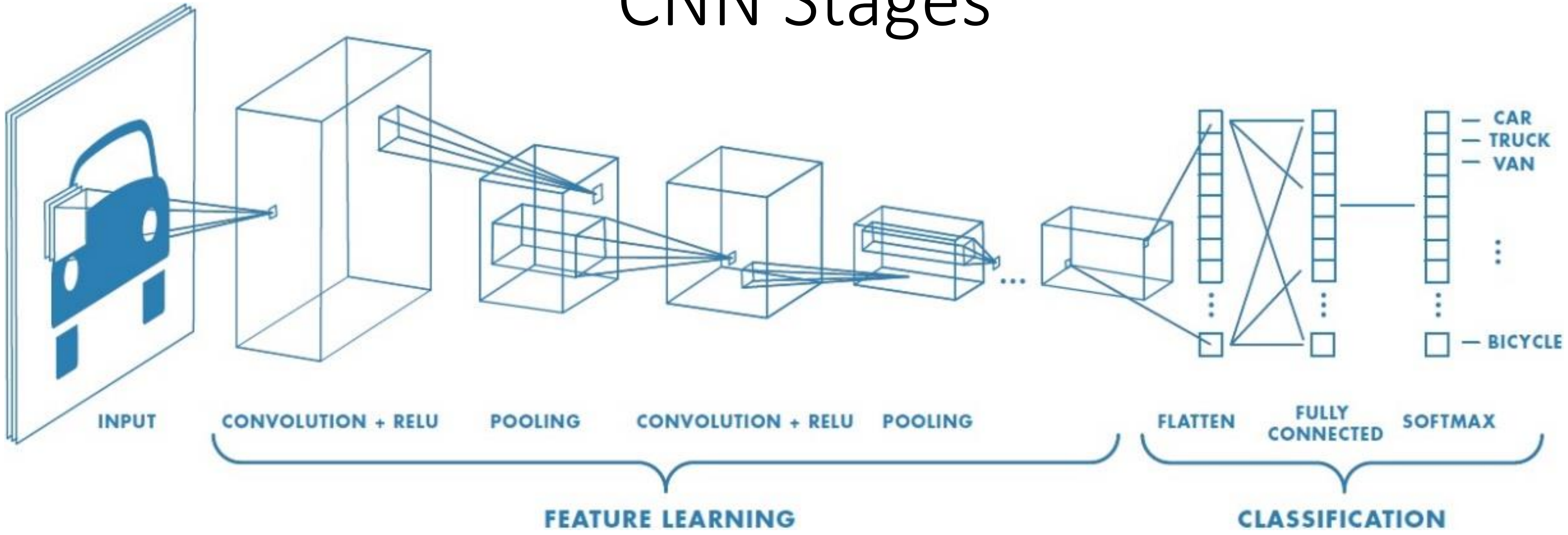
- Wide user base
- Community support
- Provides monitoring and debugging tools
- Automatically discovers and uses GPU
- Still developing rapidly
- Well documented examples
- Used at Deep Learning Indaba

What is a CNN

- Mimic mammalian visual vortex
- Neurons activate to edges in specific orientation
- Layered architecture
- Construct complex abstract features from features in the previous layer



CNN Stages



Three operations is repeated several times

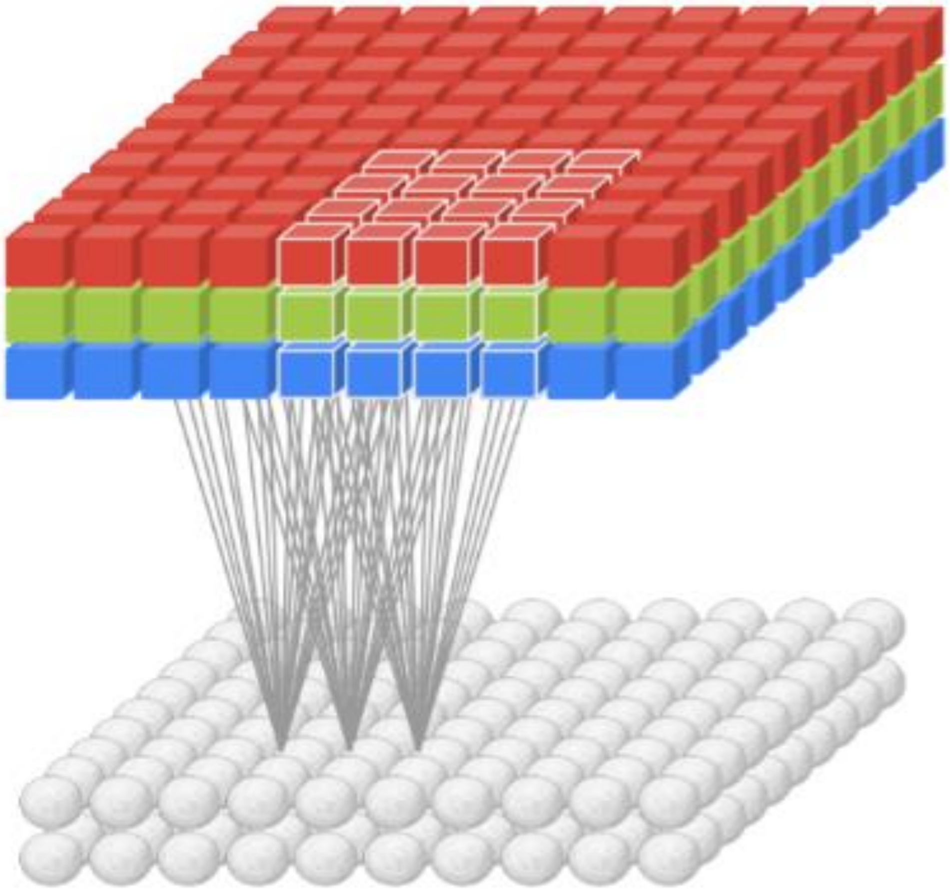
Convolution = learnable filters

ReLU = saturate nonlinearities, set negative values to zero.

Maxpooling = downsamples input, reduces computational complexity

Theoretical Work – Convolutional Layer

- Learnable filters stored in weight tensor
- Receptive field slided across the width and height of input image
- Dot product between filter and selected part of image
- = scalar value + learnable bias
- Produces two-dimensional feature map for each filter
- These maps indicate the similarity of the image to that filter at every spatial position
- Output volume = maps stacked along the depth dimension



Theoretical Work – Training process

A CNN is trained by continuously looping through four (4) steps:

1. Sample a batch of data as
   a) the input volume of size $[batchsize, w, h, C]$ and
   b) one-hot encoded label matrix Y';
2. Forward propagate the input volume through the network to calculate the loss between the predicted labels and actual labels;

$$L(W) = \frac{1}{N}\sum_{i}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

3. Backpropagate using the chain rule to calculate the gradient of the loss function with regards to each updatable parameter (eg. $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial b}$ and $\frac{\partial L}{\partial I}$); and
4. Update the parameters (weights and biases) using the gradient.

Cool resources

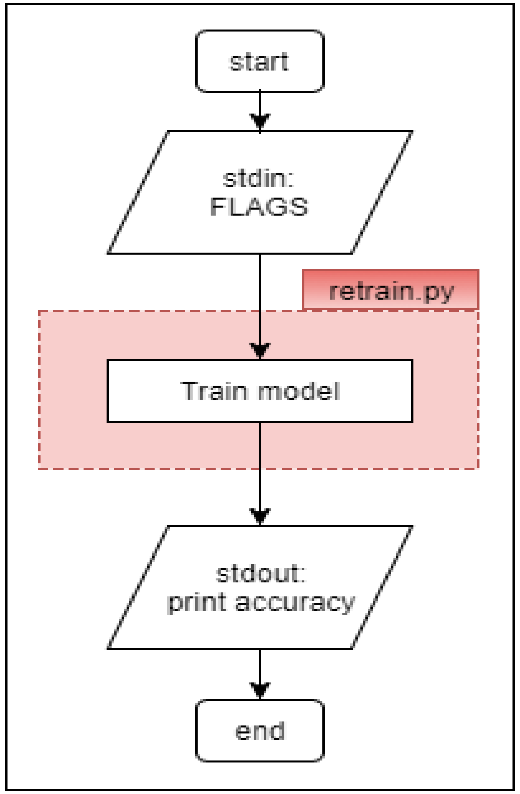
- Easy to follow Youtube videos:

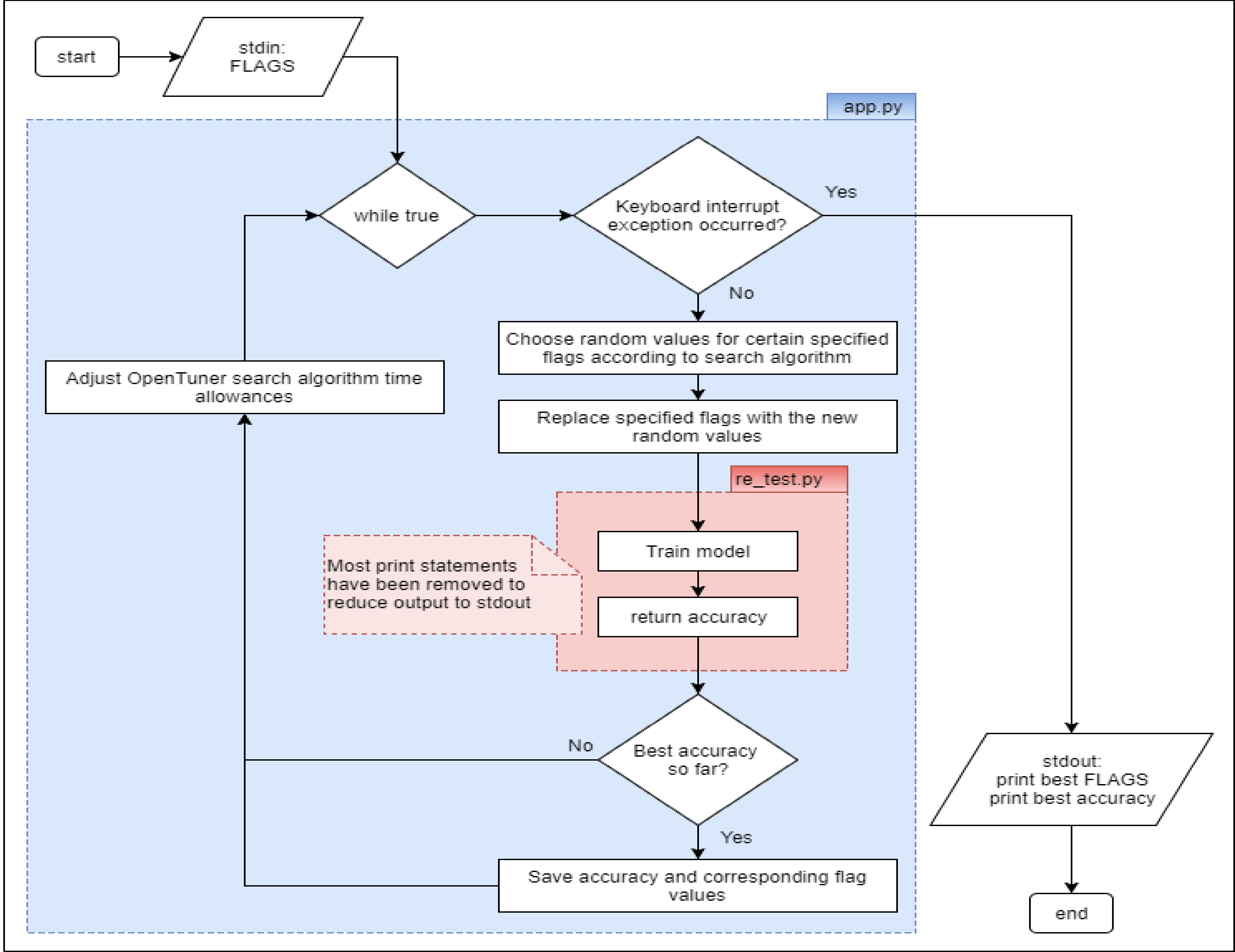
  Siraj Raval (in this case his CNN video): https://www.youtube.com/watch?v=FTr3n7uBIuE&t=2018s

- FarmBoek:

  https://www.farmboek.com/Home/Home

Tuner


start
stdin:
FLAGS
retrain.py
Train model
stdout:
print accuracy
end


start
stdin:
FLAGS
app.py
while true
Keyboard interrupt exception occurred?
Yes
No
Choose random values for certain specified flags according to search algorithm
Replace specified flags with the new random values
re_test.py
Train model
return accuracy
Most print statements have been removed to reduce output to stdout
Adjust OpenTuner search algorithm time allowances
Best accuracy so far?
No
Yes
Save accuracy and corresponding flag values
stdout:
print best FLAGS
print best accuracy
end