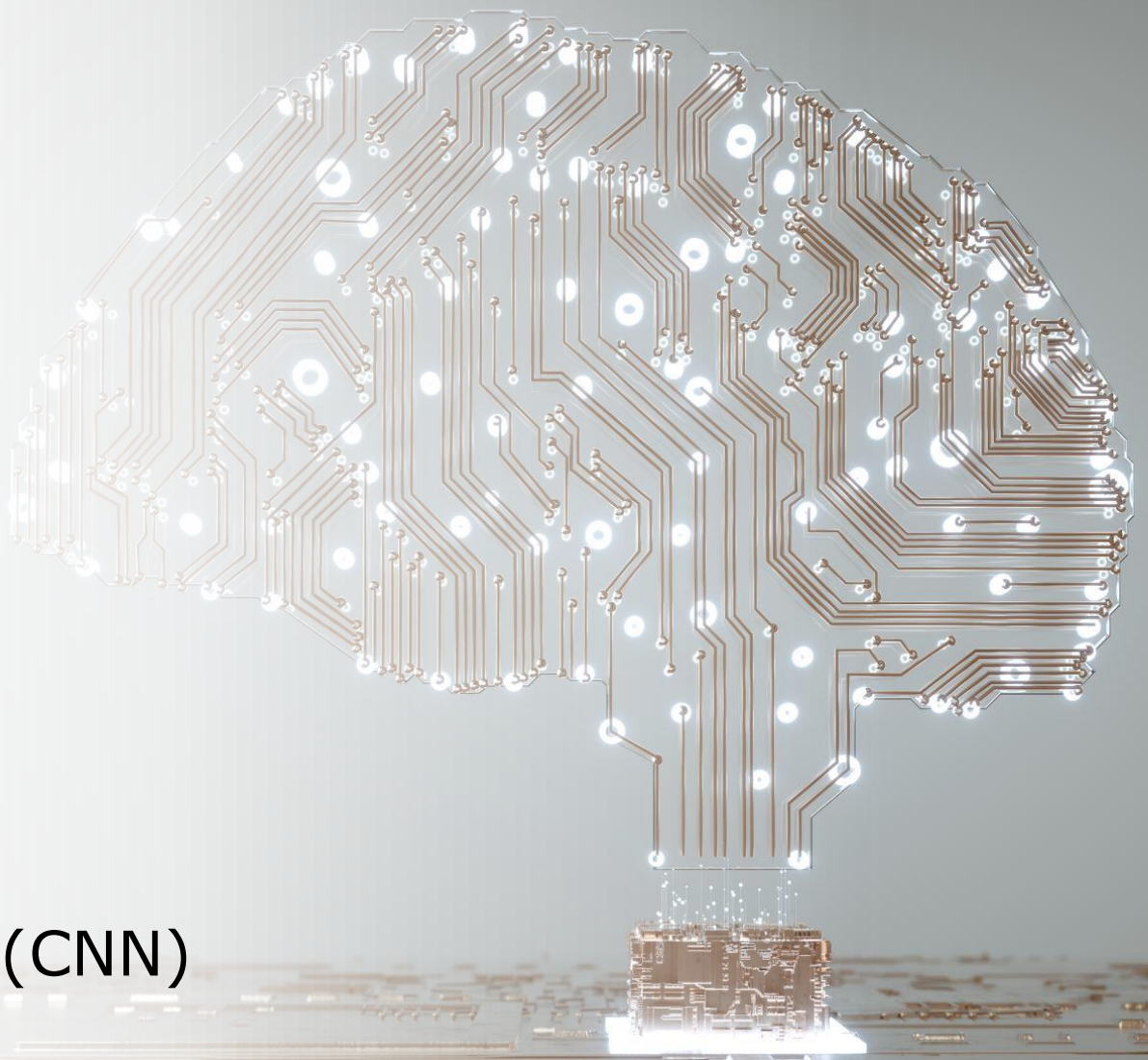
# Artificial intelligence

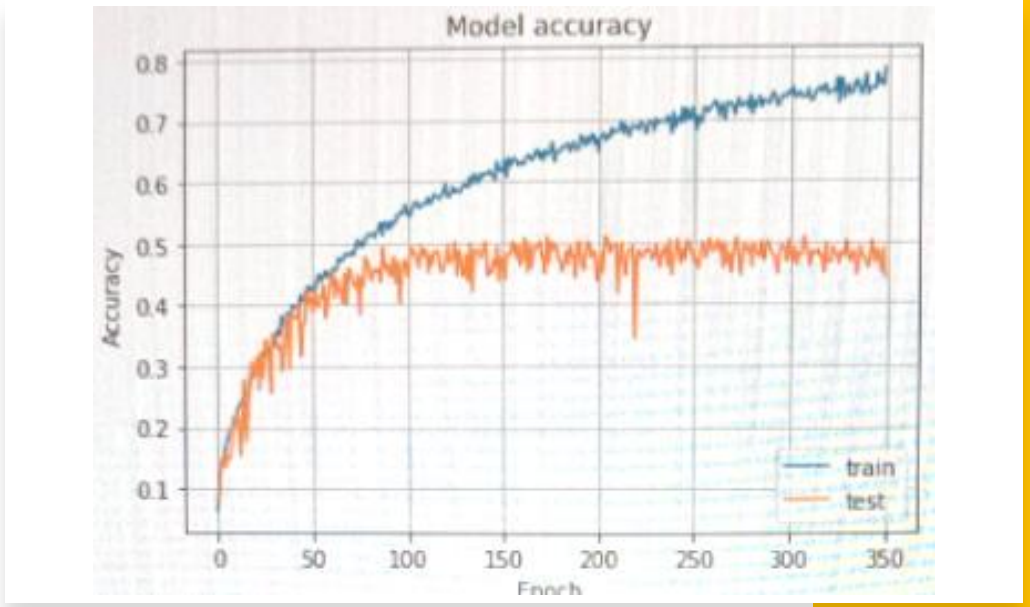
Image recognition using
Convolutional Neural Networks (CNN)



Baris Firat Arslan | Edward Gregory | Patrick Zimmermann | Sascha Marte | Wilhelm Kuchernig

# Trials & Findings

- Initially check the impact of several parameters
- Refine parameters where accuracy increased

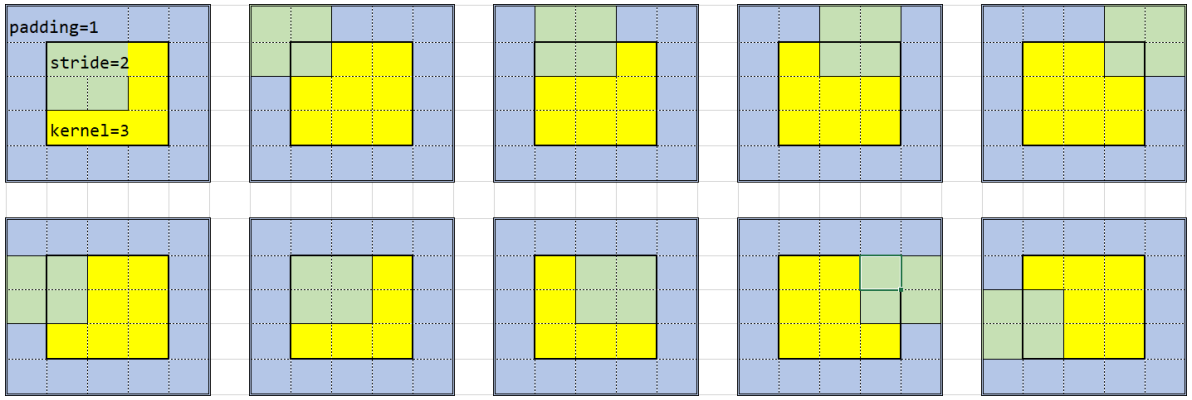


| conv_count | dense_algo | target_size | filters | padding | batch_size | dense | epochs | learning_rate | aug | bias | batch_norm | flatten | avg_pool | Run | compare_to | paramaeters | acc | loss | val_acc | val_loss | t_epoch |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 0 | 0 | 0 | 1 | 0 | 1 | | 496614 | 0,0743 | 4,2048 | 0,0289 | 5,2127 | 41 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 0 | 0 | 1 | 0 | 1 | 2 | 1 | 5222 | 0,3806 | 2,2551 | 0,3800 | 2,3841 | 43 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 1 | 0 | 1 | 0 | 1 | 3 | 2 | 5222 | 0,3154 | 2,5809 | 0,2941 | 2,7442 | 67 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 0 | 1 | 1 | 0 | 1 | 4 | 2 | 5286 | 0,3898 | 2,2411 | 0,2978 | 2,6682 | 42 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 1 | 1 | 1 | 0 | 1 | 5 | 2 | 5286 | 0,3614 | 2,4047 | 0,2837 | 2,9009 | 63 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 1 | 1 | 0 | 1 | 0 | 6 | 2 | 496678 | 0,0282 | 4,5004 | 0,0281 | 4,5302 | 60 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,01 | 0 | 0 | 1 | 1 | 1 | 7 | 2 | 5286 | 0,3689 | 2,3395 | 0,3496 | 2,6694 | 66 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 10 | 0,001 | 0 | 0 | 1 | 0 | 1 | 8 | 2 | 5222 | 0,2677 | 2,8866 | 0,3007 | 2,8789 | 45 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 9 | 8 | 5222 | 0,4204 | 2,1502 | 0,4000 | 2,4079 | 43 |
| 2 | softmax | 128 | 16 | none | n/100 | 32 | 100 | 0,001 | 0 | 0 | 1 | 0 | 1 | 10 | 9 | 5222 | 0,5138 | 1,7637 | 0,4037 | 2,3401 | 45 |
| 2 | softmax | 256 | 16 | none | n/100 | 32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 11 | 9 | 5222 | 0,3964 | 2,2416 | 0,3585 | 2,5857 | 107 |
| 2 | softmax | 64 | 16 | none | n/100 | 32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 12 | 9 | 5222 | 0,4435 | 2,0499 | 0,4030 | 2,3588 | 28 |
| 2 | softmax | 32 | 16 | none | n/100 | 32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 13 | 9 | 5222 | 0,4382 | 2,0284 | 0,4148 | 2,3039 | 25 |
| 2 | softmax | 64 | 32 | none | n/100 | 32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 14 | 12 | 9126 | 0,5488 | 1,6164 | 0,4644 | 2,0828 | 32 |
| 2 | softmax | 64 | 32/16 | none | n/100 | 32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 15 | 14 | 6502 | 0,4752 | 1,9108 | 0,4148 | 2,3326 | 31 |
| 2 | softmax | 64 | 32/16 | none | n/100 | 32/64 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 16 | 12 | 11878 | 0,4886 | 1,7951 | 0,4267 | | xx |
| 2 | softmax | 64 | 32/16 | none | n/100 | 16/32 | 36 | 0,001 | 0 | 0 | 0/1 | 0 | 1 | 17 | 12 | 6646 | 0,3764 | 2,3469 | 0,3348 | 2,5420 | 32 |
| 3 | softmax | 64 | 32/16/8 | none | n/100 | 128/64/32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 18 | 12 | 17996 | 0,4494 | 1,9199 | 0,4022 | 2,2293 | 34 |
| 2 * 2 | softmax | 64 | 32/16 | none | n/100 | 16/32 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 17 | 16 | 13142 | 0,5152 | 1,6146 | 0,3630 | 2,5115 | 33 |
| 3 * 2 | softmax | 64 | 32/16/8 | none | n/100 | 32/64/128 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 18 | 17 | 32198/224 | 0,5479 | 1,4959 | 0,4052 | 2,5187 | 60 |
| 3 * 2 | sigmoid | 64 | 32/16/8 | none | n/100 | 32/64/128 | 50 | 0,001 | 0 | 0 | 1 | 0 | 1 | 19 | 17 | 32198/224 | 0,5134 | 1,6010 | 0,3919 | 2,4017 | 67 |

# Parametrization

- Use overlapping areas (padding) around the input data (kernel)

→ Allows use of more filter w/o running into sizing error disaster

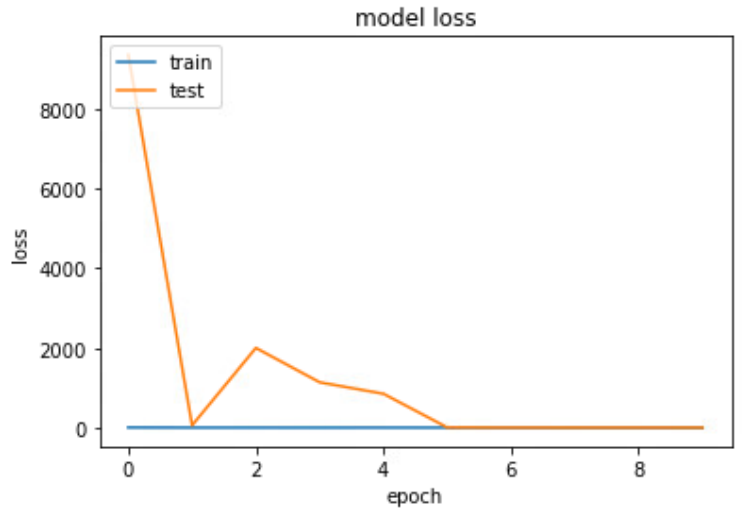


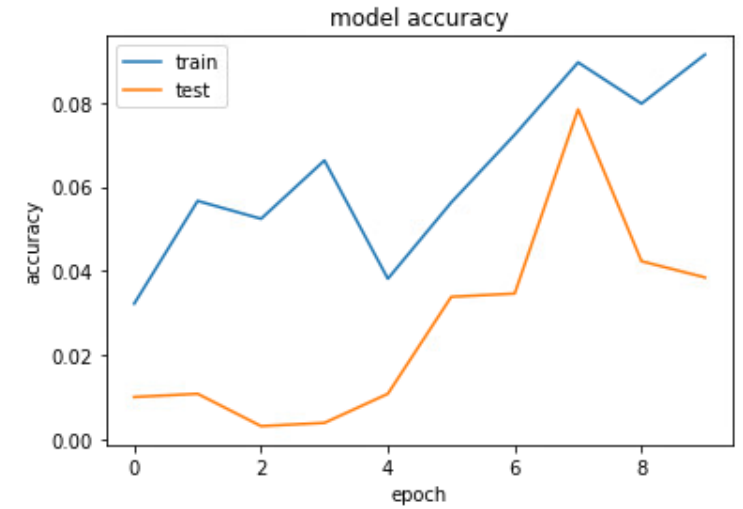
# Trials & Findings

- Lower learning rate = lower distance btw. train/test
- Changing target size only has no effect (waste of time)
- AveragePooling2D() >> Flatten()

```
Epoch 1/10
54/54 [==============================] - 636s 12s/step - loss: 8.3678 - accuracy: 0.0322 - val_loss: 9351.1035 - val_accuracy: 0.0100
Epoch 2/10
54/54 [==============================] - 605s 11s/step - loss: 4.8615 - accuracy: 0.0567 - val_loss: 61.7247 - val_accuracy: 0.0108
Epoch 3/10
54/54 [==============================] - 591s 11s/step - loss: 4.8345 - accuracy: 0.0524 - val_loss: 2002.5267 - val_accuracy: 0.0031
Epoch 4/10
54/54 [==============================] - 575s 11s/step - loss: 4.3010 - accuracy: 0.0663 - val_loss: 1140.8383 - val_accuracy: 0.0038
Epoch 5/10
54/54 [==============================] - 590s 11s/step - loss: 4.7794 - accuracy: 0.0381 - val_loss: 852.8815 - val_accuracy: 0.0108
Epoch 6/10
54/54 [==============================] - 575s 11s/step - loss: 4.3914 - accuracy: 0.0563 - val_loss: 8.5325 - val_accuracy: 0.0338
Epoch 7/10
54/54 [==============================] - 591s 11s/step - loss: 4.2114 - accuracy: 0.0724 - val_loss: 5.2387 - val_accuracy: 0.0346
Epoch 8/10
54/54 [==============================] - 592s 11s/step - loss: 4.0803 - accuracy: 0.0896 - val_loss: 4.8771 - val_accuracy: 0.0785
Epoch 9/10
54/54 [==============================] - 579s 11s/step - loss: 4.1038 - accuracy: 0.0798 - val_loss: 4.6874 - val_accuracy: 0.0423
Epoch 10/10
54/54 [==============================] - 577s 11s/step - loss: 4.0881 - accuracy: 0.0915 - val_loss: 4.4181 - val_accuracy: 0.0385
```

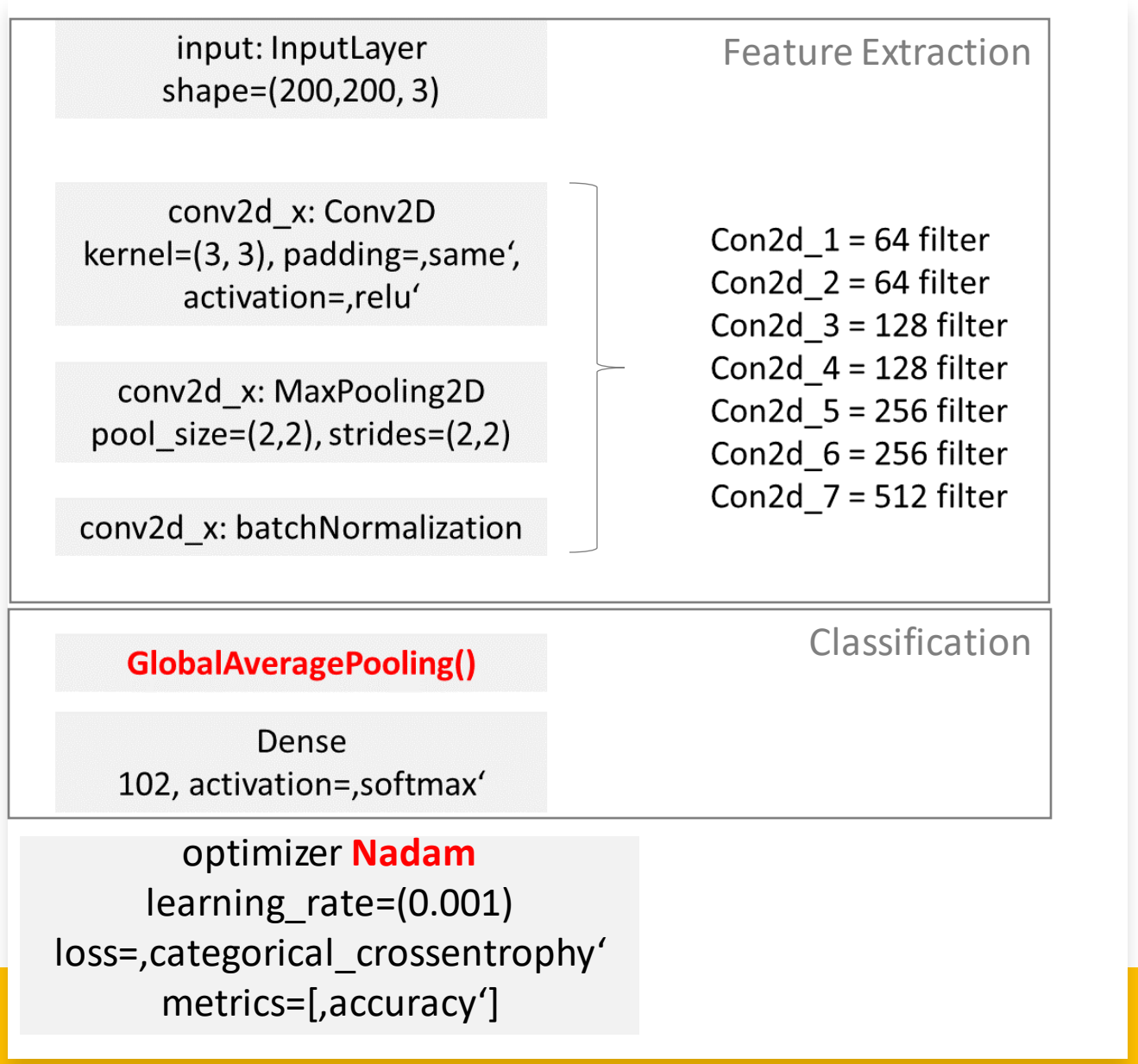
```python
# Defining the Optimizer
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam
opt = Adam(learning_rate=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```



# Optimized Model

- 7 Conv2D Layer
- ReLU
- Filters (64/128/256/512)
- GlobalAveragePooling2D()
- Optimizers: Nadam



# Results & Conclusions

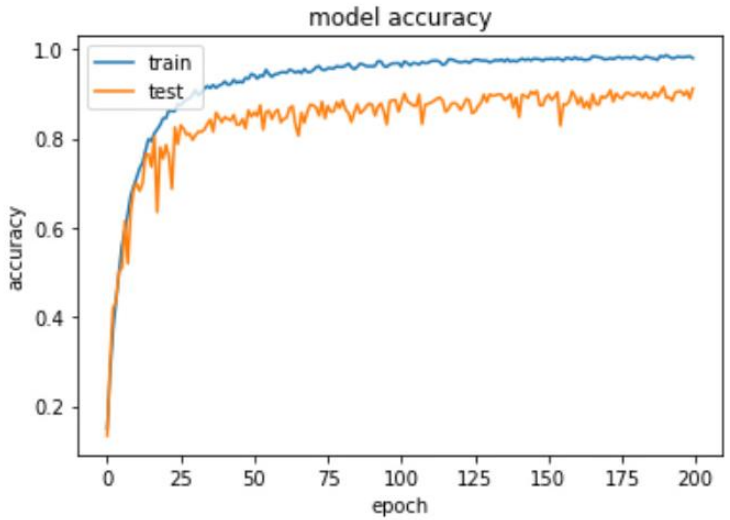
*accuracy*

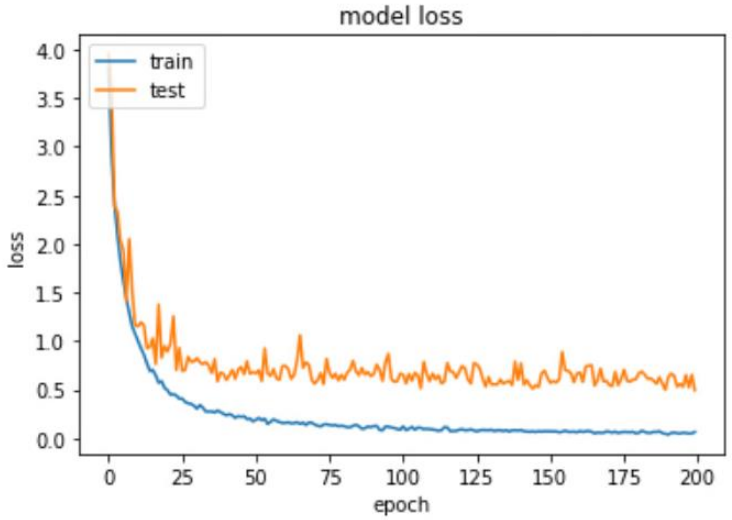
**98,03%**

*val_accuracy*

**91,19%**

```
Epoch 195/200
422/422 [==============================] - 494s 1s/step - loss: 0.0496 - accuracy: 0.9836 - val_loss: 0.5592 - val_accuracy: 0.9052
Epoch 196/200
422/422 [==============================] - 490s 1s/step - loss: 0.0578 - accuracy: 0.9823 - val_loss: 0.5300 - val_accuracy: 0.9059
Epoch 197/200
422/422 [==============================] - 484s 1s/step - loss: 0.0554 - accuracy: 0.9823 - val_loss: 0.6505 - val_accuracy: 0.8969
Epoch 198/200
422/422 [==============================] - 487s 1s/step - loss: 0.0501 - accuracy: 0.9834 - val_loss: 0.5260 - val_accuracy: 0.9066
Epoch 199/200
422/422 [==============================] - 488s 1s/step - loss: 0.0509 - accuracy: 0.9841 - val_loss: 0.6538 - val_accuracy: 0.8895
Epoch 200/200
422/422 [==============================] - 488s 1s/step - loss: 0.0655 - accuracy: 0.9803 - val_loss: 0.4954 - val_accuracy: 0.9119
```

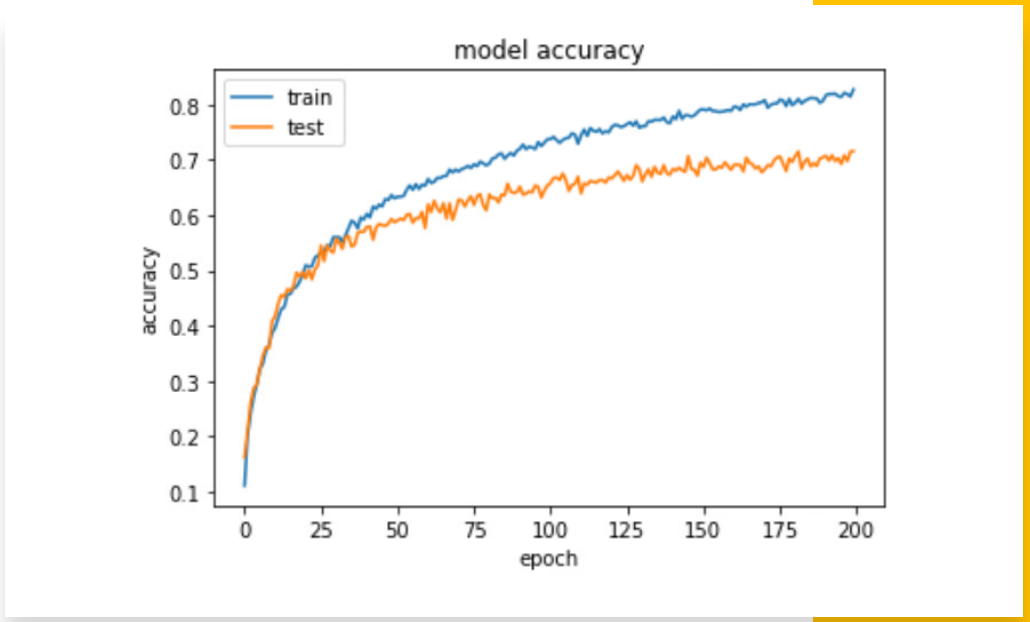


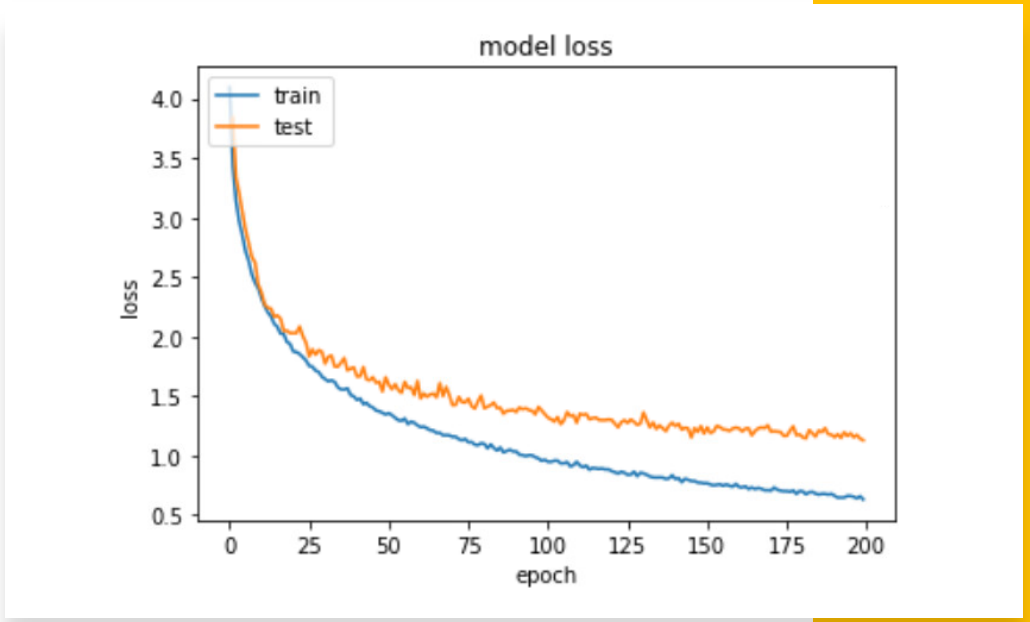



# Additional Discoveries

Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

(https://www.tensorflow.org/api)

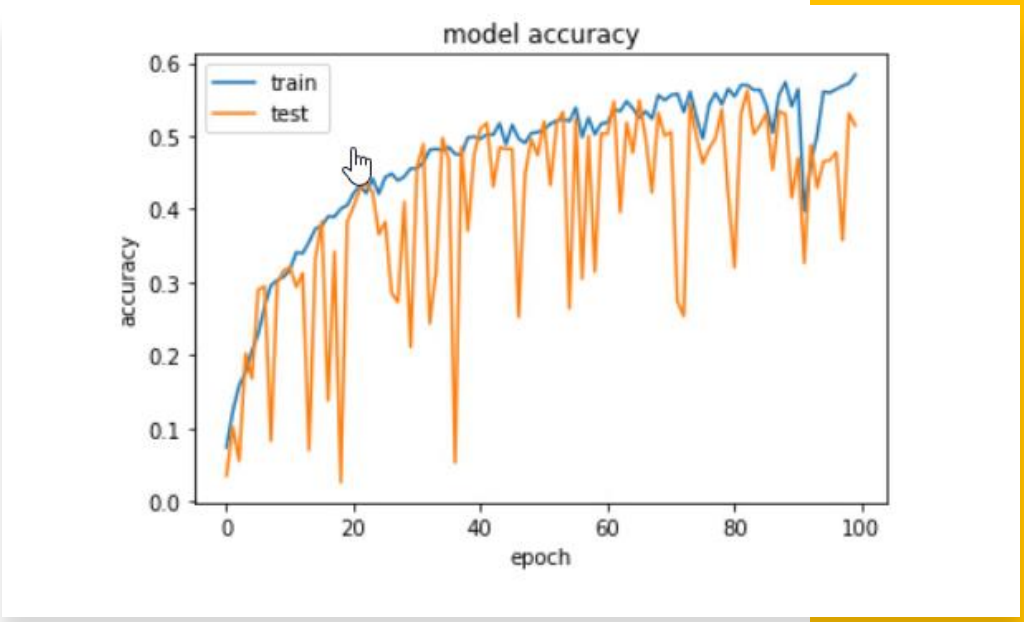


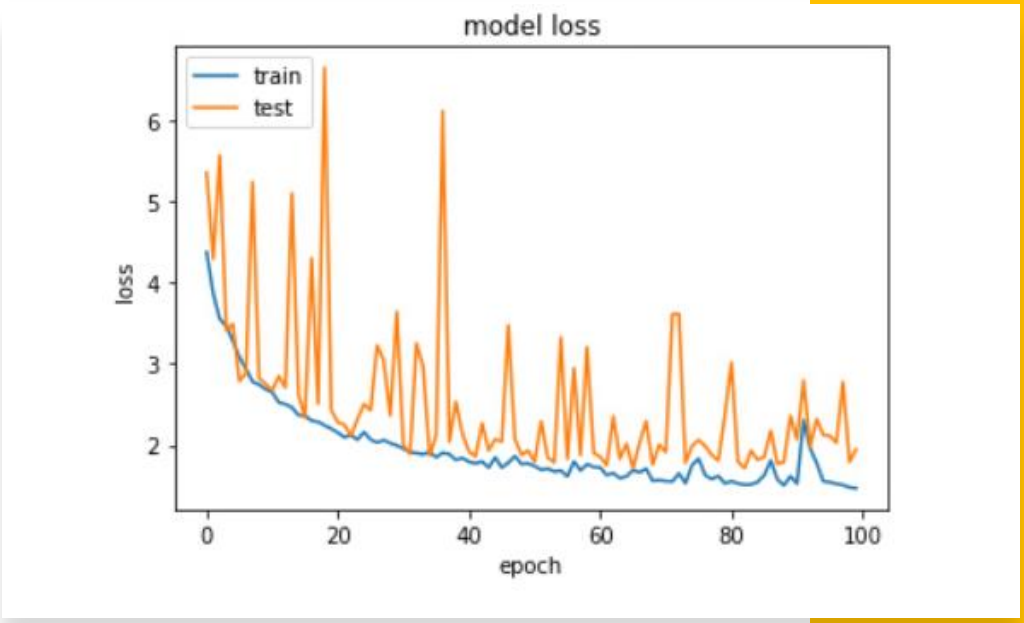



# Additional Discoveries

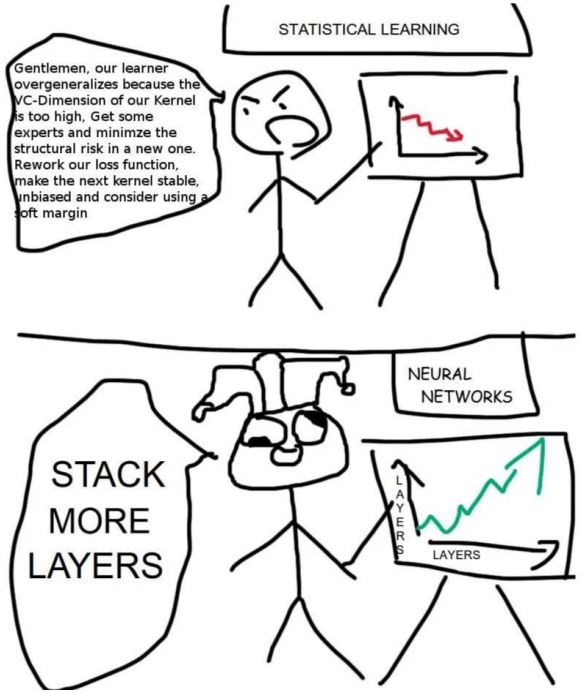
Activation function ‘tanh‘

**Disadvantage**
Also facing the same issue of Vanishing Gradient Problem like a sigmoid function.



# Key Findings




Conv2D Layers

Filters

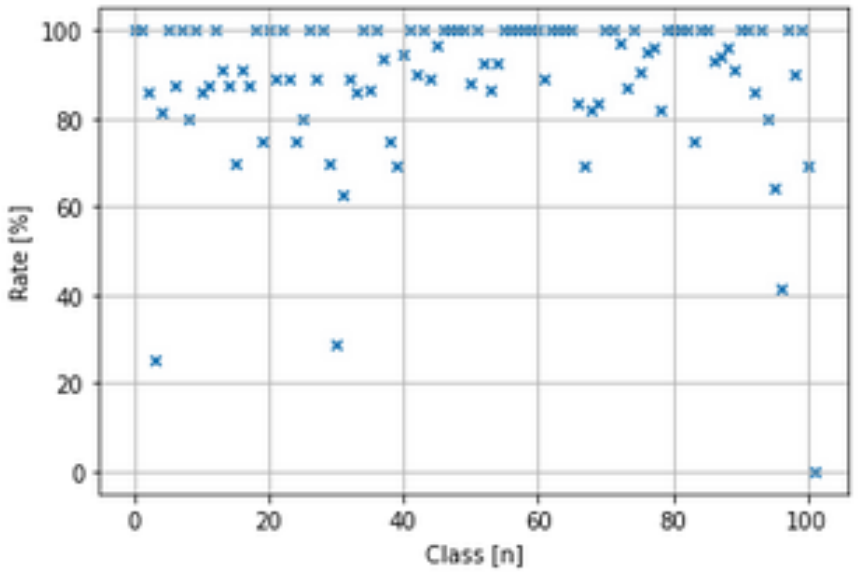
CPU Speed

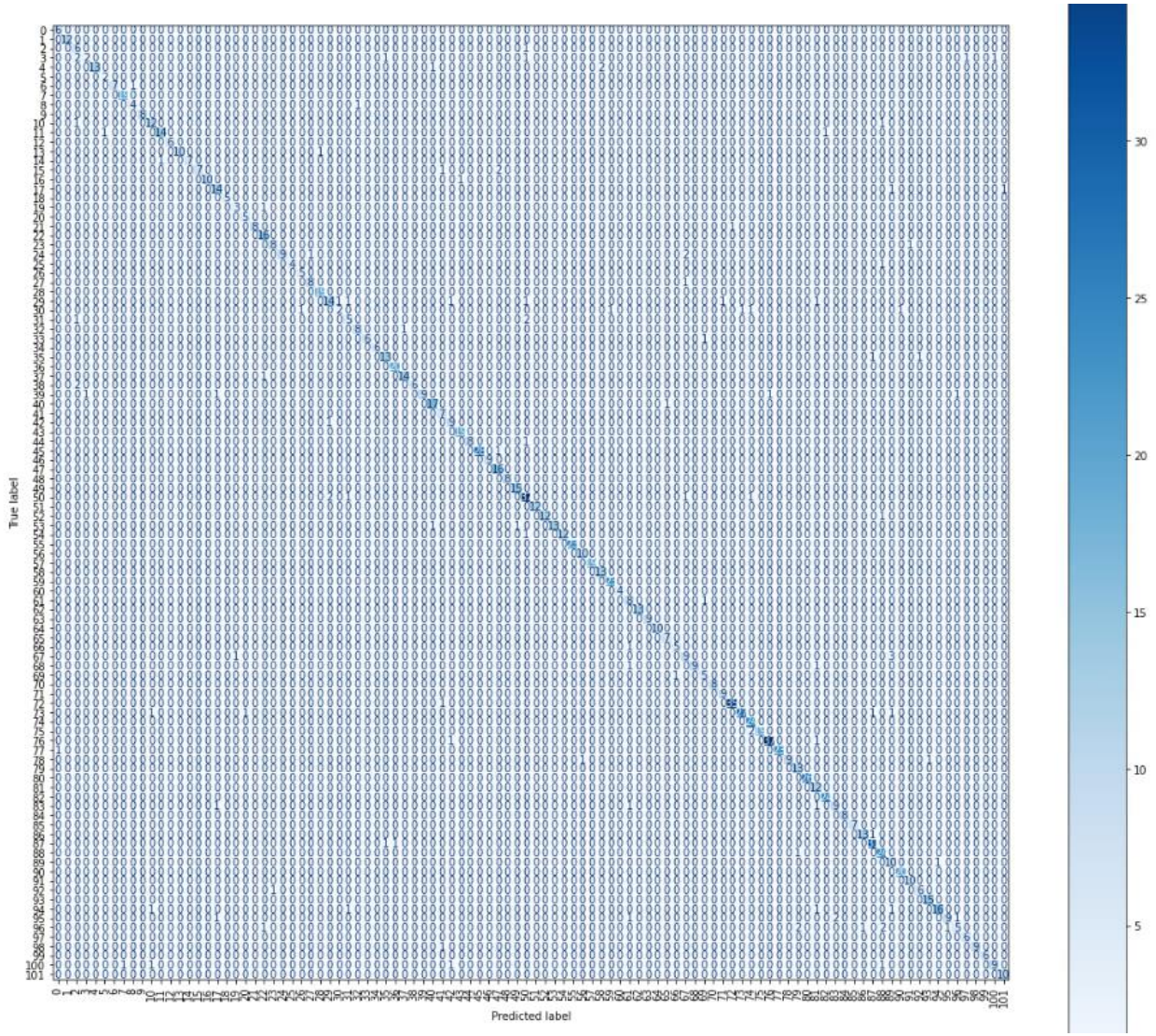
Batch Size

Dense Layer

Normali-zation

# Confusion Matrix

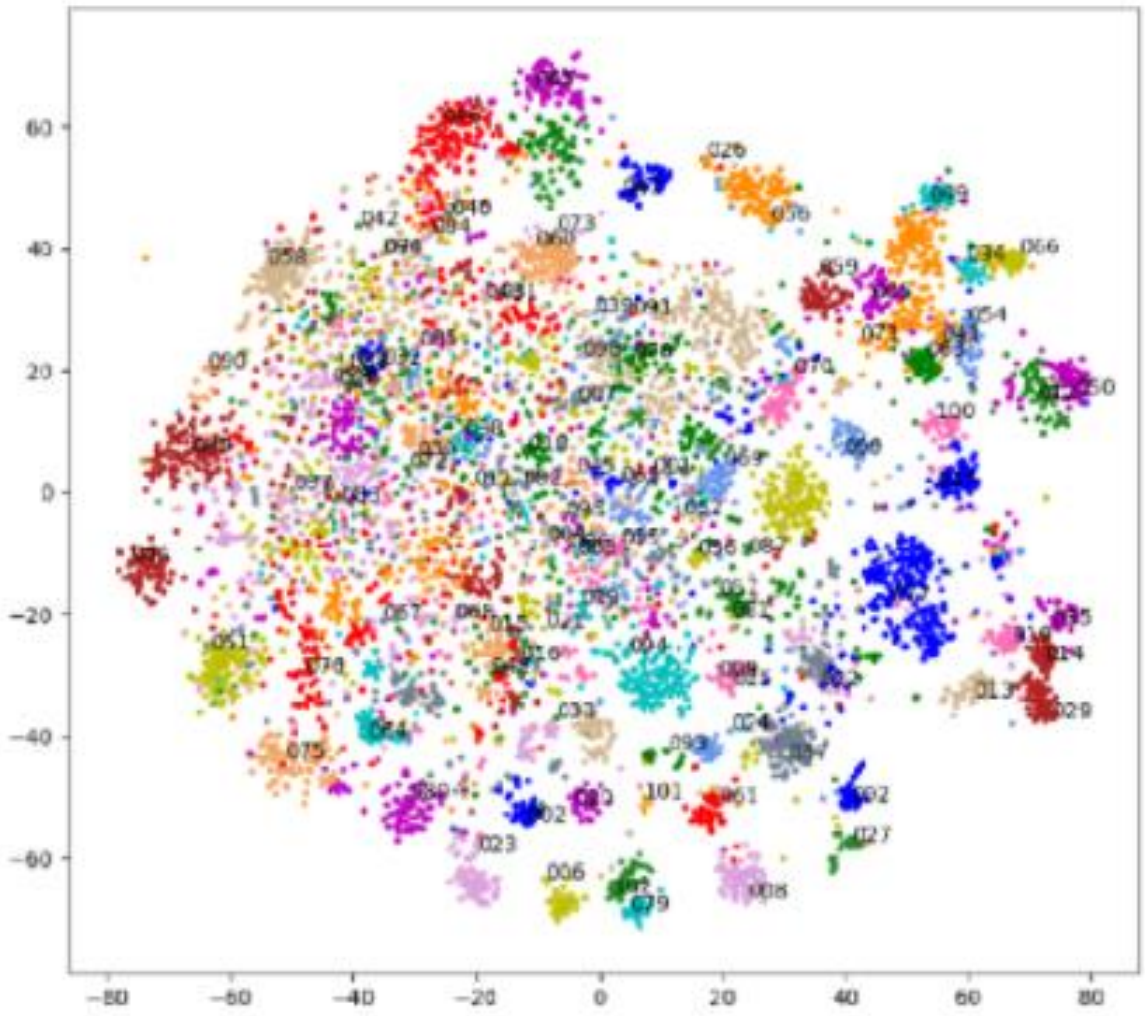


# Wrong predictions

**'sweet pea'**
Class 3: Total=8, matched=2, mismatched=6, rate=25.0%
**'carnation'**
Class 30: Total=7, matched=2, mismatched=5, rate=28.57142857142857%
**'garden phlox'**
Class 31: Total=8, matched=5, mismatched=3, rate=62.5%
**'lenten rose'**
Class 39: Total=13, matched=9, mismatched=4, rate=69.23076923076923%
**'camellia'**
Class 95: Total=14, matched=9, mismatched=5, rate=64.28571428571429%
**'mallow'**
Class 96: Total=12, matched=5, mismatched=7, rate=41.66666666666667%

# Overview of classification




b) Oxford 102 Flowers Dataset

# Literature


••https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adagrad#:~:text=Adagrad%20is%20an%20optimizer%20with,receives%2C%20the%20smaller%20the%20updates.

••https://colab.research.google.com/github/kirankamatmgm/FlowerImageClassifier/blob/master/FlowerspeciesClassifier.ipynb#scrollTo=suYlKhT3v_ZH

••https://www.aitude.com/comparison-of-sigmoid-tanh-and-relu-activation-functions/#:~:text=ReLu%20is%20the%20best%20and,compare%20to%20other%20activation%20function.

••https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-cvi.2017.0155

••https://www.technologiesinindustry4.com/2021/08/plantnet-plant-identification.html