Fruit Recognition Dataset (https://www.kaggle.com/chrisfilo/fruit-recognition)

Dataset:
Fruit classification of 22 different classes (including distinguishing different types of apples (5) and kiwis (3)). Total of 44000 images.
Classes: 'Apple A', 'Apple C', 'Apple D', 'Apple E', 'Apple F', 'Banana', 'Carambola', 'Guava A', 'Guava B', 'Kiwi A', 'Kiwi B', 'Kiwi C', 'Mango', 'Muskmelon', 'Orange', 'Peach', 'Pear', 'Persimmon', 'Pitaya', 'Plum', 'Pomegranate', 'Tomatoes'.

'Apple A'

'Apple C'

'Apple D'

'Apple E'

'Apple F'

'Banana'

'Carambola'

'Guava A'

'Guava B'

'Kiwi A'

'Kiwi B'

'Kiwi C'

'Mango'

'Muskmelon'

'Orange'

'Peach'

'Pear'

'Persimmon

'Pitaya'

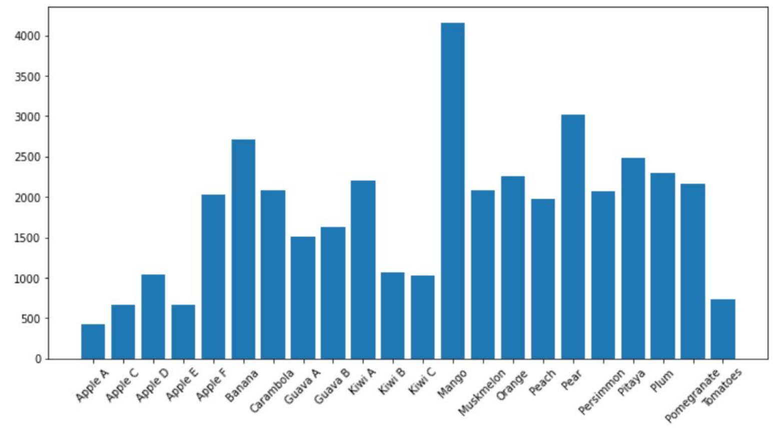
'Plum'

'Pomegranate'

'Tomatoes'.

Although it may seem an easy task for classifying some fruits, other fruit types classification can be quite challenging as there are fruits hard to distinguish by humans, such as between Apple types, Guava types and Kiwi types. Also, in the images there can be different amounts of pieces of fruit, situated in different parts of the scene, with different light intensities, and partial occlusions (where this variability should add complexity to the problem).

The balance among classes is the following →



There are several ways in the literature to deal with class unbalance:

- Weighted loss computation
- Under-sampling
- Over-sampling

(https://towardsdatascience.com/handling-imbalanced-datasets-in-deep-learning-f48407a0e758

Experiments pipeline:

1. Preprocess
2. Find an initial simple architecture (to solve underfit)
3. Complex architecture (to have a CNN that is able to learn the needed patterns → overfitting)
4. Solve overfitting (add regularization to close the gap between train and validation loss curves)
5. Adjust parameters (hyperparameters, add data augmentation)

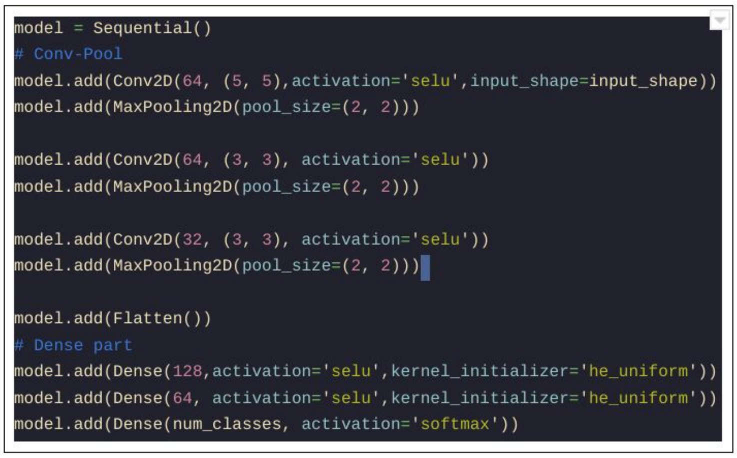
Preprocessing:
There are a lot of images with high resolution (250x320). Considering that in datasets like CIFAR10 the images are of size 32x32, thus we reduced the images to 30% of their original size (resulting in images of 77x96). Also, there were 4000 images with a different resolution (representing 9% of the 44000 images), so we decided to remove them considering that there are a lot of samples.
This reduced the dataset memory **from 8GB to 700MB**. Also, this will speed up training a lot as well as the computation resources needed.

Initial Architecture:
In order to establish a baseline for the dataset, we began with an architecture that consisted of a single convolutional layer, with a single filter. As this model is extremely simple, we knew that it would underfit the data. With this model, we were able to achieve a training accuracy of **86%** and a testing accuracy of **83%**. We are using **Keras** to implement the model.

```
model = Sequential()
model.add(Conv2D(1, 2, 1, activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(num_classes, activation=(tf.nn.softmax)))
```

Complex Architectures:
In order to find a model that overfit the data, we added several additional layers, and increased the parameters for the convolutional layers to more reasonable values. Furthermore, we began experiments with the complex architectures using **500 instances per class** (under-sampling) in order to increase computation time and reserve resources. We trained this model using **50 epochs** with **early-stopping** of 10 epochs, **SELU** activation function as it is an improved version of ReLU, and **Adam** optimizer. Finally, we split the data as **90% train**, **10% test**, with 10% of the train data used for validation.

Model:



```
model = Sequential()
# Conv-Pool
model.add(Conv2D(64, (5, 5),activation='selu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='selu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), activation='selu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
# Dense part
model.add(Dense(128,activation='selu',kernel_initializer='he_uniform'))
model.add(Dense(64, activation='selu',kernel_initializer='he_uniform'))
model.add(Dense(num_classes, activation='softmax'))
```

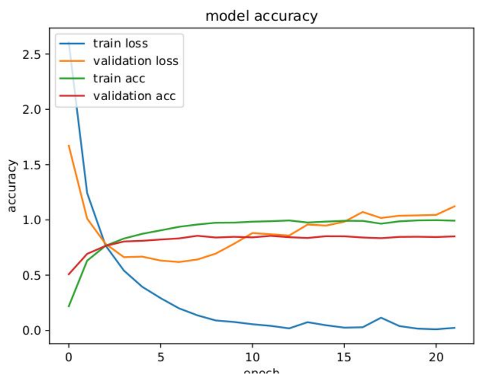
Complex Architecture Experiments:

CNN without Dropout

Results with complex model:

- train_acc 0.9928,
- val_acc: 0.8509,
- test_acc: **0.8271** (with early stopping, getting back to model weights at epoch 7)

There is a clear overfitting that is stopped by **early stopping.**

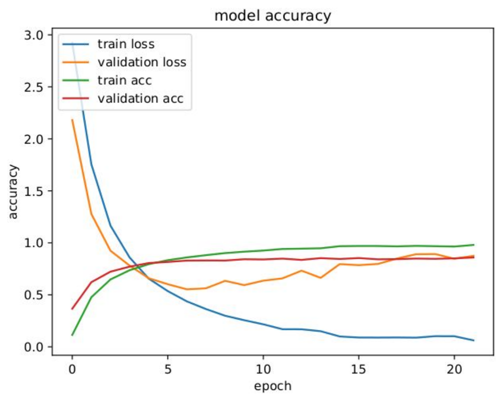


CNN With Dropout

Results with complex model adding 2 spatialDropout2D layers and 1 standard Dropout layer):

- train_acc: 0.9801
- val_acc: 0.8591
- test_acc: **0.849**

When using dropout layers, we can see that the test accuracy increased (by 2%), and the training accuracy decreased, meaning less overfitting. Also, loss curves between validation and train seem to be closer.



Model with dropout layers:

```
model = Sequential()
# Conv-Pool
model.add(Conv2D(64, (5, 5),activation='selu',input_shape=input_shape))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='selu'))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='selu'))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
# Dense part
model.add(Dense(128,activation='selu',kernel_initializer='he_uniform'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='selu',kernel_initializer='he_uniform'))
model.add(Dense(num_classes, activation='softmax'))
```

Results:

	precision	recall	f1-score	support
kiwi A	0.98	0.96	0.97	50
Apple A	0.92	0.96	0.94	48
Apple E	0.98	0.96	0.97	57
Kiwi B	1.00	0.98	0.99	63
Persimmon	0.73	0.81	0.77	47
guava B	0.91	0.96	0.94	53
Tomatoes	0.76	0.81	0.78	42
Peach	0.84	0.86	0.85	59
Banana	0.76	0.63	0.69	46
Pomegranate	0.72	0.78	0.75	54
guava A	1.00	1.00	1.00	49
Apple C	0.96	1.00	0.98	45
Pear	0.85	0.74	0.79	54
Apple D	0.89	0.96	0.92	51
Apple F	0.71	0.71	0.71	41
Pitaya	0.80	0.76	0.78	46
Plum	1.00	0.88	0.93	48
muskmelon	0.67	0.88	0.76	32
Kiwi C	1.00	0.98	0.99	56
Carambola	0.65	0.71	0.68	49
Orange	0.71	0.70	0.70	63
Mango	0.75	0.53	0.62	40
accuracy			0.85	1093
macro avg	0.84	0.84	0.84	1093
weighted avg	0.85	0.85	0.85	1093

```
[[48  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0]
 [ 0 46  0  0  0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0 55  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0]
 [ 1  0  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 38  0  1  0  0  3  0  0  0  0  0  2  0  0  0  2  0  1]
 [ 0  0  0  0  0 51  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0]
 [ 0  0  0  0  0  0 34  1  0  1  0  0  1  0  1  1  0  0  0  1  2  0]
 [ 0  0  0  0  1  0  2 51  0  0  0  0  0  1  0  1  0  0  0  0  1  2]
 [ 0  0  0  0  0  2  2  1 29  5  0  0  0  0  1  0  0  1  0  1  3  1]
 [ 0  0  0  0  3  1  1  0  0 42  0  0  2  0  2  0  0  0  0  1  1  1]
 [ 0  0  0  0  0  0  0  0  0  0 49  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 45  0  0  0  0  0  0  0  0  0  0]
 [ 0  2  0  0  2  0  0  0  2  2  0  0 40  0  0  1  0  3  0  0  1  1]
 [ 0  0  0  0  0  0  1  0  0  0  0  0  0 49  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  1  0  0  0  0  1  0  0 29  0  0  6  0  2  1  1]
 [ 0  0  0  0  1  0  1  1  2  1  0  0  0  0  0 35  0  0  0  5  0  0]
 [ 0  2  0  0  0  0  0  4  0  0  0  0  0  0  0  0 42  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0 28  0  0  2  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0 55  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  0  1  1  0  3  3  0  2  0 35  3  0]
 [ 0  0  1  0  5  1  2  2  1  2  0  0  0  1  3  0  0  0  0  1 44  0]
 [ 0  0  0  0  1  1  0  0  4  1  0  0  3  1  1  0  0  2  0  2  3 21]]
```

(this confusion matrix does not have the labels sorted alphabetically)

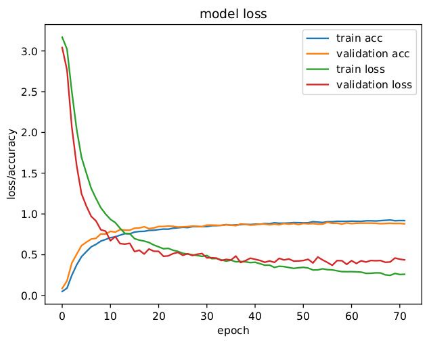
This model is clearly overfitting as it is able to achieve a training accuracy of nearly 100% without dropout, while the testing accuracy is much lower. As the dropout layers were able to improve the results slightly, this model seems to be a good CNN architecture to begin with. However, more regularization is needed to close the gap between validation loss and train loss.

Solve Overfitting:

Model with <u>more Layers</u> and <u>more Regularization</u> (500 instances per class [undersampling])

Goal: Close the gap with validation set (regularization, data augmentation) -> Fit?

Adding more dropout layers helped to close the gap between the train and the validation set. Hence, we improved model generalization and test accuracy obtained was **0.8664**. Also, early-stopping extended to 15 epochs.



train_acc: 0.9186
val_acc: 0.8791
test_acc: **0.8664**

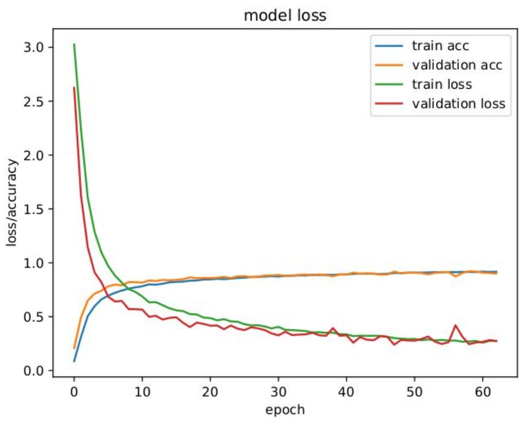
```
model = Sequential()
# (Conv-Pool)*
model.add(Conv2D(128, (5, 5), activation='selu', input_shape=input_shape))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (5, 5), activation='selu'))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='selu'))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='selu'))
model.add(SpatialDropout2D(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
# Fully-connected layers
model.add(Flatten())
model.add(Dense(264, activation='selu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='selu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='selu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='selu'))
model.add(Dense(num_classes, activation='softmax'))
```

Epoch 00072: early stopping.
Training time → 6s/epoch * 72 epochs = 7.2 minutes

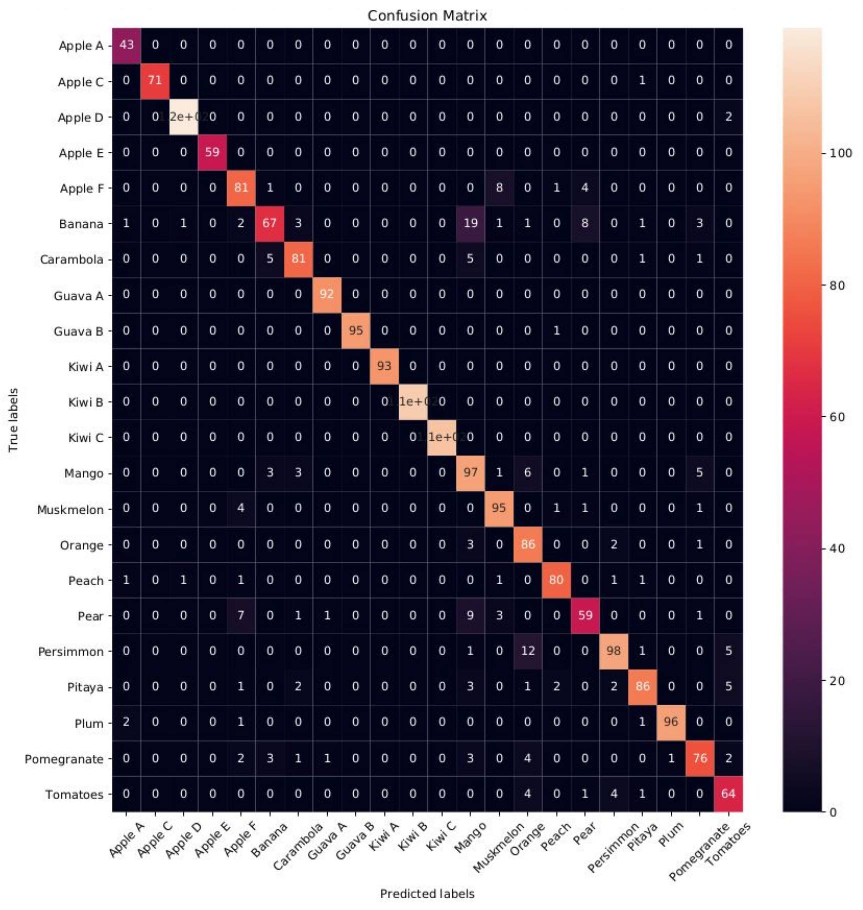
CNN Using more data (imbalanced dataset)

Using 1000 instances per class instead of 500 → a bit of unbalance in some classes that have less than 1000 samples (e.g. apple A, C, ...)

train_acc: 0.9162
val_acc: 0.9013
test_acc: **0.9043**



Epoch 00063: early stopping



	Apple A	Apple C	Apple D	Apple E	Apple F	Banana	Carambola	Guava A	Guava B	Kiwi A	Kiwi B	Kiwi C	Mango	Muskmelon	Orange	Peach	Pear	Persimmon	Pitaya	Plum	Pomegranate	Tomatoes
Apple A	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Apple C	0	71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Apple D	0	0	2e+0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
Apple E	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Apple F	0	0	0	0	81	1	0	0	0	0	0	0	0	8	0	1	4	0	0	0	0	0
Banana	1	0	1	0	2	67	3	0	0	0	0	0	19	1	1	0	8	0	1	0	3	0
Carambola	0	0	0	0	0	5	81	0	0	0	0	0	5	0	0	0	0	0	1	0	1	0
Guava A	0	0	0	0	0	0	0	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Guava B	0	0	0	0	0	0	0	0	95	0	0	0	0	0	0	1	0	0	0	0	0	0
Kiwi A	0	0	0	0	0	0	0	0	0	93	0	0	0	0	0	0	0	0	0	0	0	0
Kiwi B	0	0	0	0	0	0	0	0	0	0	1e+0	0	0	0	0	0	0	0	0	0	0	0
Kiwi C	0	0	0	0	0	0	0	0	0	0	0	1e+0	0	0	0	0	0	0	0	0	0	0
Mango	0	0	0	0	0	3	3	0	0	0	0	0	97	1	6	0	1	0	0	0	5	0
Muskmelon	0	0	0	0	4	0	0	0	0	0	0	0	0	95	0	1	1	0	0	0	1	0
Orange	0	0	0	0	0	0	0	0	0	0	0	0	3	0	86	0	0	2	0	0	1	0
Peach	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	80	0	1	1	0	0	0
Pear	0	0	0	0	7	0	1	1	0	0	0	0	9	3	0	0	59	0	0	0	1	0
Persimmon	0	0	0	0	0	0	0	0	0	0	0	0	1	0	12	0	0	98	1	0	0	5
Pitaya	0	0	0	0	1	0	2	0	0	0	0	0	3	0	1	2	0	2	86	0	0	5
Plum	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	96	0	0
Pomegranate	0	0	0	0	2	3	1	1	0	0	0	0	3	0	4	0	0	0	0	1	76	2
Tomatoes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	1	4	1	0	0	64

Training time → 12s/epoch * 63epochs = 12.6 min

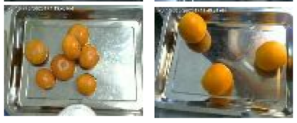
Using more training samples definitely improves model accuracy.

There are classes that the model struggles to classify:

- Muskmelon vs. Apple F

- Banana vs. Mango

- Persimmon vs. Orange



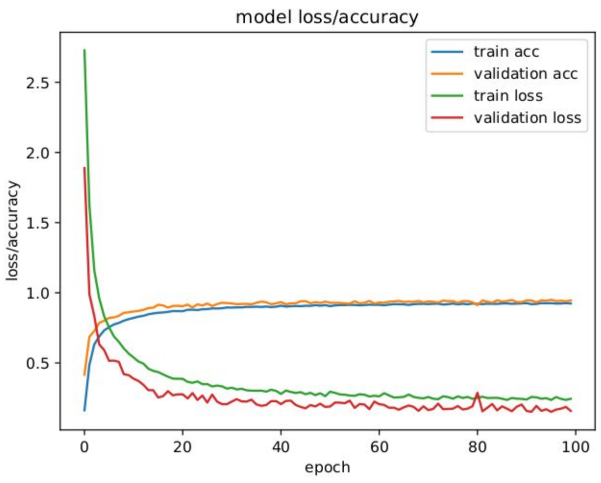
However, it's impressive how perfectly the model is able to classify Apple types, Guava types and Kiwi types without making any mistakes.

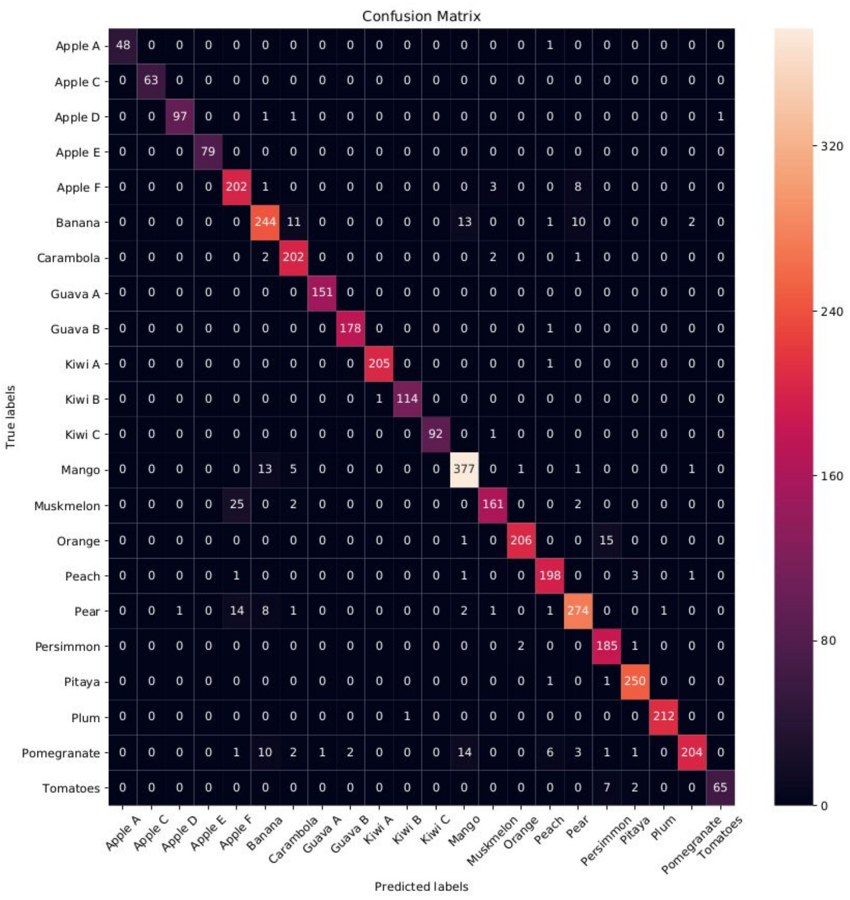
CNN Using all data

Using all available data may create unbalance of instances in some classes, but having more training samples may help to improve overall performance.

train_acc: 0.9237
val_acc: 0.9453
test_acc: **0.9456**

Although having unbalance in classes, the model keeps on improving.





	Apple A	Apple C	Apple D	Apple E	Apple F	Banana	Carambola	Guava A	Guava B	Kiwi A	Kiwi B	Kiwi C	Mango	Muskmelon	Orange	Peach	Pear	Persimmon	Pitaya	Plum	Pomegranate	Tomatoes
Apple A	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Apple C	0	63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Apple D	0	0	97	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Apple E	0	0	0	79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Apple F	0	0	0	0	202	1	0	0	0	0	0	0	0	3	0	0	8	0	0	0	0	0
Banana	0	0	0	0	0	244	11	0	0	0	0	0	13	0	0	1	10	0	0	0	2	0
Carambola	0	0	0	0	0	2	202	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0
Guava A	0	0	0	0	0	0	0	151	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Guava B	0	0	0	0	0	0	0	0	178	0	0	0	0	0	0	1	0	0	0	0	0	0
Kiwi A	0	0	0	0	0	0	0	0	0	205	0	0	0	0	0	1	0	0	0	0	0	0
Kiwi B	0	0	0	0	0	0	0	0	0	1	114	0	0	0	0	0	0	0	0	0	0	0
Kiwi C	0	0	0	0	0	0	0	0	0	0	0	92	0	1	0	0	0	0	0	0	0	0
Mango	0	0	0	0	0	13	5	0	0	0	0	0	377	0	1	0	1	0	0	0	1	0
Muskmelon	0	0	0	0	25	0	2	0	0	0	0	0	0	161	0	0	2	0	0	0	0	0
Orange	0	0	0	0	0	0	0	0	0	0	0	0	1	0	206	0	0	15	0	0	0	0
Peach	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	198	0	0	3	0	1	0
Pear	0	0	1	0	14	8	1	0	0	0	0	0	2	1	0	1	274	0	0	1	0	0
Persimmon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	185	1	0	0	0
Pitaya	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	250	0	0	0
Plum	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	212	0	0
Pomegranate	0	0	0	0	1	10	2	1	2	0	0	0	14	0	0	6	3	1	1	0	204	0
Tomatoes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	2	0	0	65

	precision	recall	f1-score	support
Apple A	1.00	0.98	0.99	49

Apple C	1.00	1.00	1.00	63
Apple D	0.99	0.97	0.98	100
Apple E	1.00	1.00	1.00	79
Apple F	0.83	0.94	0.88	214
Banana	0.87	0.87	0.87	281
Carambola	0.90	0.98	0.94	207
Guava A	0.99	1.00	1.00	151
Guava B	0.99	0.99	0.99	179
Kiwi A	1.00	1.00	1.00	206
Kiwi B	0.99	0.99	0.99	115
Kiwi C	1.00	0.99	0.99	93
Mango	0.92	0.95	0.94	398
Muskmelon	0.96	0.85	0.90	190
Orange	0.99	0.93	0.96	222
Peach	0.94	0.97	0.96	204
Pear	0.92	0.90	0.91	303
Persimmon	0.89	0.98	0.93	188
Pitaya	0.97	0.99	0.98	252
Plum	1.00	1.00	1.00	213
Pomegranate	0.98	0.83	0.90	245
Tomatoes	0.98	0.88	0.93	74

Unbalance in the samples does not seem to affect f1-score, precision and recall of the classes.

Epochs 100
Training time: 23s/epoch * 100 epochs = 38.33 mins

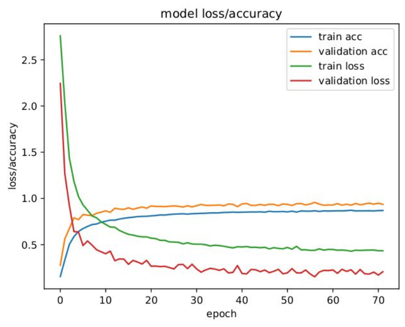
Tuning Final Model

Add data augmentation → implies more time per epoch, hence, longer training times, but helps to build more robust models

- Rotation (up to 20 degrees)
- Horizontal flip randomly
- Shifting 7% height or width

Also used smoothing to remove noise in images (gaussian filter with sigma=1)
Results:

- train_acc: 0.8694
- val_acc: 0.9360
- test_acc: **0.9625**



Epoch 00072: early stopping
Training time: 40s/epoch * 72epochs = 46.6mins

Hyperparameters →

- Learning Rate: at this point is difficult to state which is the optimal LR, but values around 0.001. Did not seem to improve using other values

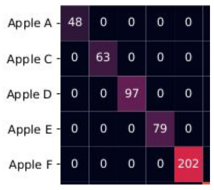
LR	**0.01**	**0.001**	**0.0001**
Test Accuracy	0.9506	**0.9625**	0.9446
Test Accuracy (with weighted class balance)	0.9379	0.9506	0.9433

Balance classes:

- We opted to do **weighted balance** in loss computation which is an accepted parameter in the model fit method in Keras. However, looking at model performance and it does not influence a lot in the resulting model accuracy
-

Conclusions

- Surprised with the intra-class classification performance for Apples types and Kiwi types.



Apple A	48	0	0	0	0
Apple C	0	63	0	0	0
Apple D	0	0	97	0	0
Apple E	0	0	0	79	0
Apple F	0	0	0	0	202

- Regularization (especially dropout layers) definitely help to make a model that generalizes.
- Using imbalance class correction didn't seem to improve the accuracy (it seemed to worsen it a bit).
- We managed to reach a **0.96 accuracy** model on test set