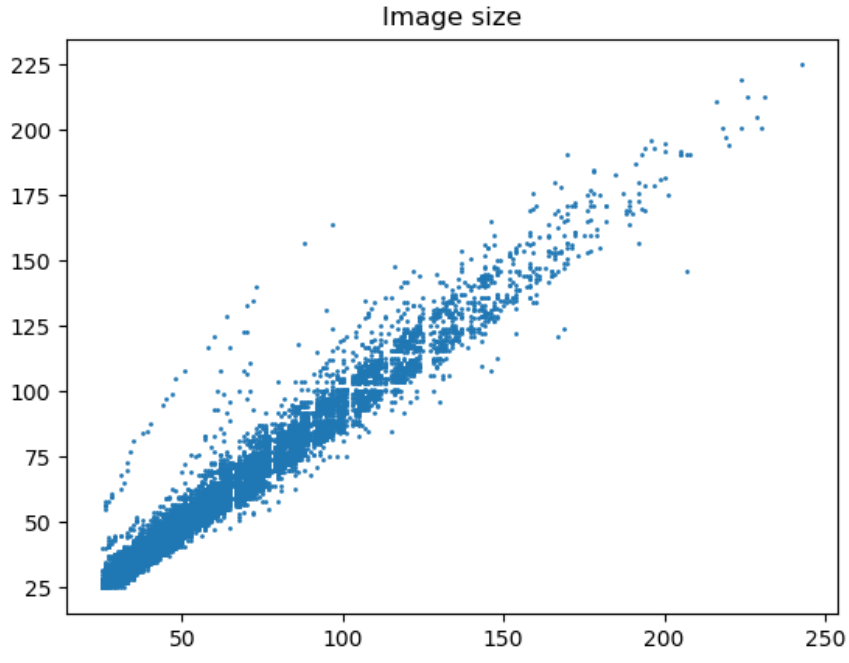
Data Exploration

Total number of images (Germany + Belgium) : 20370

Distribution of the number of images per class

Imbalanced dataset: loss function in my training will be weighted by `class_weight` argument, which is inversely proportional to the number of images per class. In this way, the optimiser will “pay more attention” to the underrepresented classes.

Image size



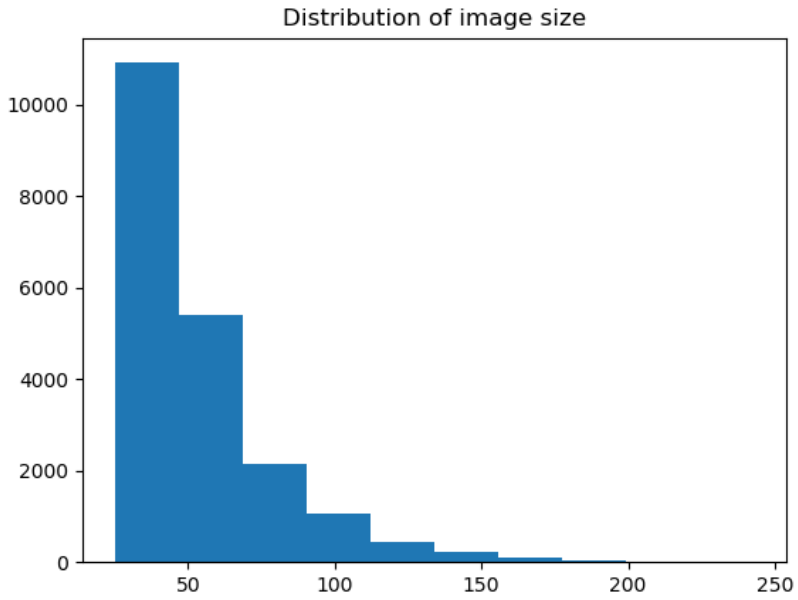


Image size: most of the images have a small size. This is one of the reasons that I suspect that the transfer learning from the models such as ResNet or GoogleNet that are trained on 224x224 images may not be the right choice. The other reason for this is that the image categories does not seem similar as in the ImageNet dataset.
Also, it seems to me that these large neural networks will be unnecessarily too complex for this task and that smaller ConvNets would be good enough.
This is my first impression, so lets see what my experiments will show.

Data Preparation

Data Split

The images are split into 75% training and 25 % validation dataset. The images belonging to the same group (the same beginning of the filename) are put into only one of these datasets in order to prevent data leakage.

The most correct way for the models' evaluation would be to perform a cross-validation (e.g. K-fold). Due to the time constraint I perform only the simple train/validation dataset split.

Data Augmentation

- translating of image
- rotating of image
- Shearing the image
- zooming in/out of the image
- brightness - not applied yet because it works non-reliable (by comments on github)

Rather than generating and saving augmented images to hard disk, I generate them on the fly during training with `ImageDataGenerator`.

Models

Small VGG

Input shape: (32, 32, 3)

Total params: 2,408,501
Trainable params: 2,407,605
Non-trainable params: 896

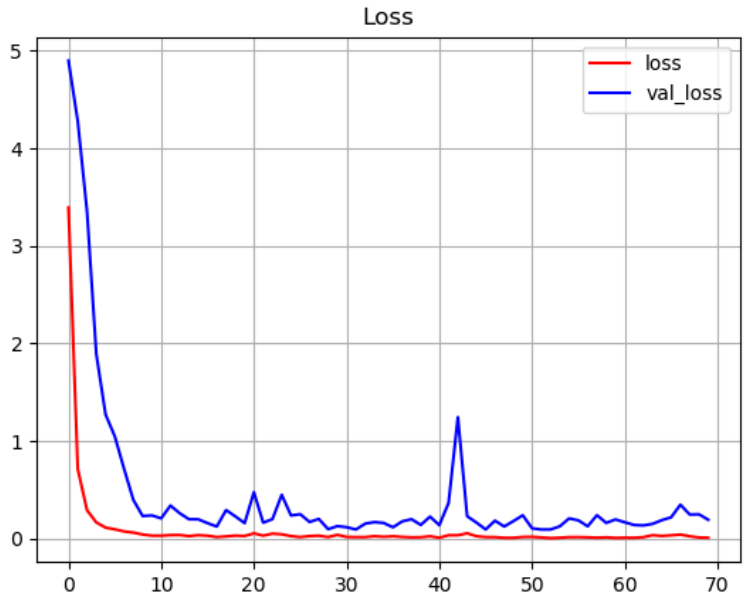
Here I try a small VGG model with the following architecture:

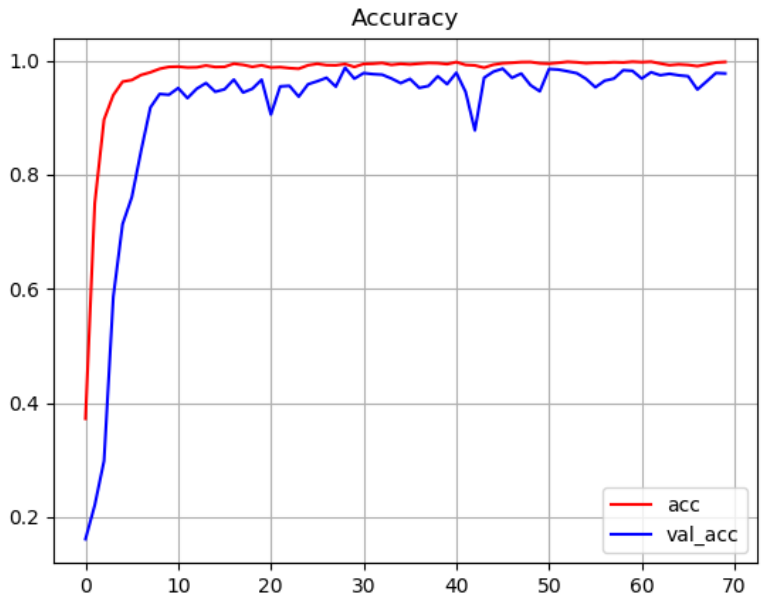
```
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(num_classes, activation='softmax')(x)
```





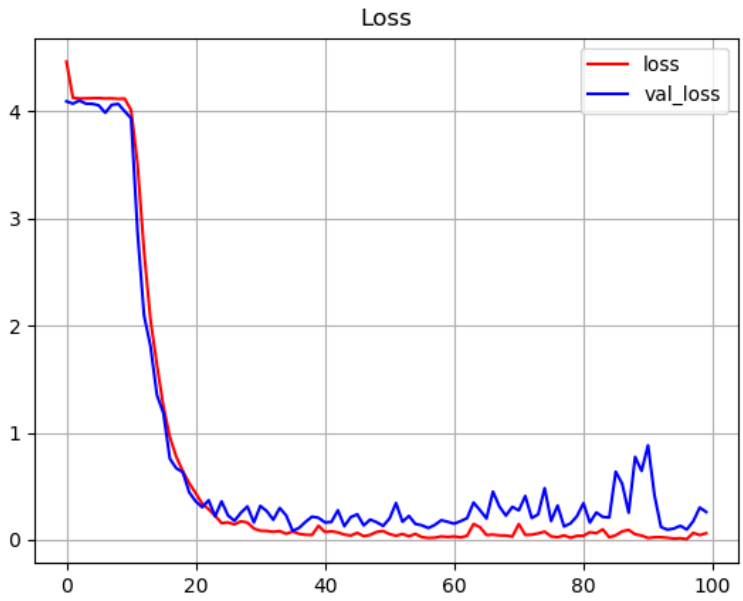
Results on the validation dataset:

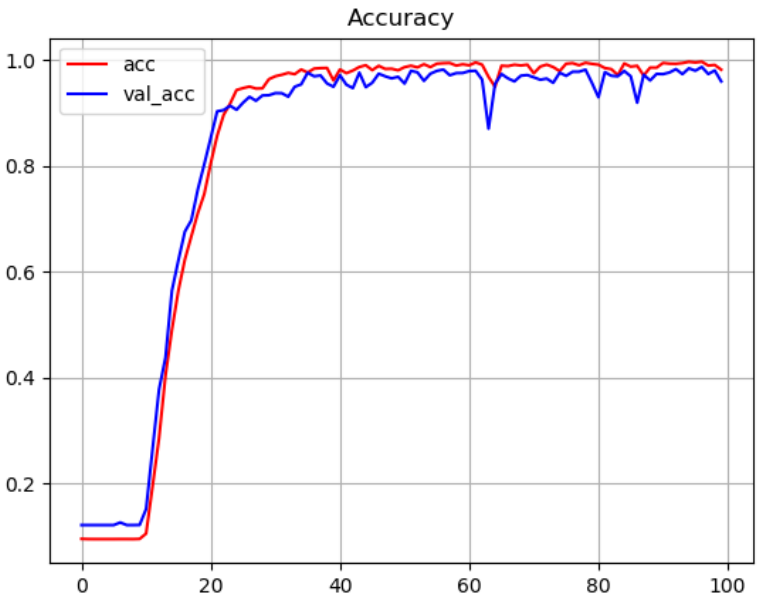
	precision	recall	f1-score	support
0	0.9688	0.9394	0.9538	33
1	0.9857	0.9125	0.9477	377
2	1.0000	0.9947	0.9973	189
3	1.0000	0.9968	0.9984	312
4	1.0000	0.7576	0.8621	33
5	0.9934	0.9869	0.9902	153
6	0.9823	0.9964	0.9893	279
7	0.9204	0.9893	0.9536	187
8	0.8451	1.0000	0.9160	60
9	1.0000	0.9956	0.9978	453
10	1.0000	0.9350	0.9664	123
11	0.9804	1.0000	0.9901	150
12	0.9527	1.0000	0.9758	624
13	0.9919	1.0000	0.9960	123
14	1.0000	0.9200	0.9583	300
15	0.9455	0.9905	0.9674	105
16	0.9964	0.9521	0.9738	585
17	0.9904	0.9986	0.9945	721
18	1.0000	1.0000	1.0000	96
19	0.8590	1.0000	0.9241	134
20	0.9600	1.0000	0.9796	72
accuracy			0.9777	5109
macro avg	0.9701	0.9698	0.9682	5109
weighted avg	0.9791	0.9777	0.9777	5109

VGG16 - Training from scratch

Input shape: (32, 32, 3)

Total params: 14,988,117
Trainable params: 14,988,117
Non-trainable params: 0





Results on the validation dataset:

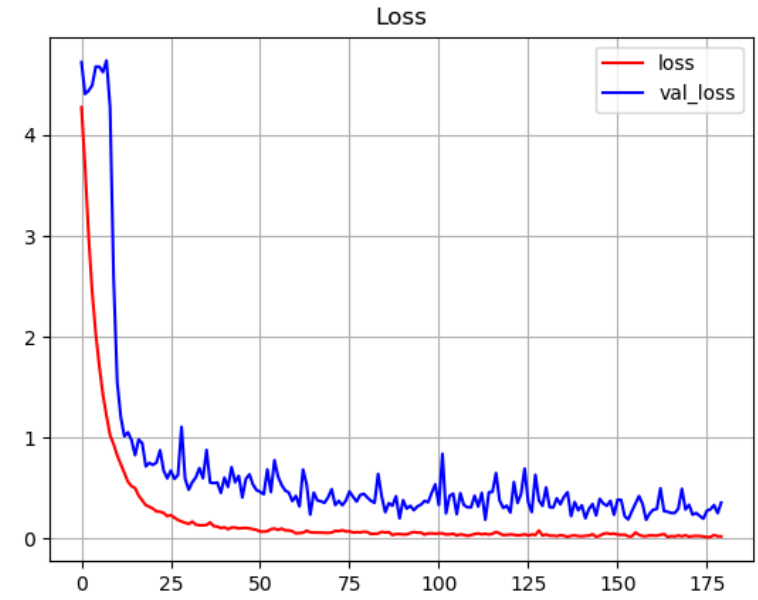
	precision	recall	f1-score	support
0	0.5246	0.9697	0.6809	33
1	0.9389	0.8966	0.9172	377
2	0.9947	0.9947	0.9947	189
3	0.8991	0.9712	0.9337	312
4	0.8696	0.6061	0.7143	33
5	0.9449	0.7843	0.8571	153
6	0.9927	0.9713	0.9819	279
7	0.8235	0.9733	0.8922	187
8	0.9423	0.8167	0.8750	60
9	0.9956	0.9912	0.9934	453
10	0.9918	0.9837	0.9878	123
11	0.9508	0.7733	0.8529	150
12	0.9733	0.9936	0.9833	624
13	0.9111	1.0000	0.9535	123
14	0.9931	0.9567	0.9745	300
15	0.9885	0.8190	0.8958	105
16	0.9932	0.9966	0.9949	585
17	1.0000	0.9917	0.9958	721
18	0.9897	1.0000	0.9948	96
19	0.8758	1.0000	0.9338	134
20	1.0000	0.9861	0.9930	72
accuracy			0.9599	5109
macro avg	0.9330	0.9274	0.9238	5109
weighted avg	0.9637	0.9599	0.9600	5109

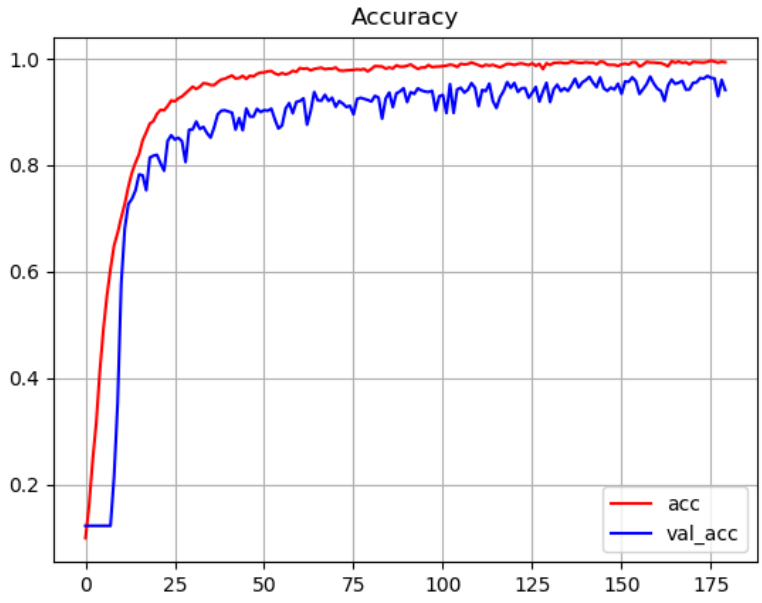
MobileNet - Training from scratch

Input shape: (32, 32, 3)

Total params: 4,299,989
Trainable params: 4,278,101
Non-trainable params: 21,888

This model is an efficient architecture designed for mobile/edge devices.





Results on the validation dataset:

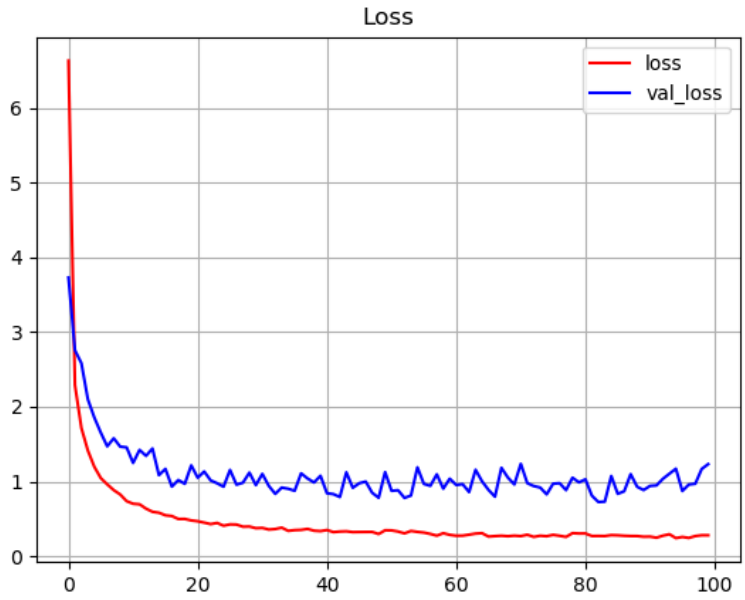
	precision	recall	f1-score	support
0	1.0000	0.8788	0.9355	33
1	0.9522	0.8992	0.9250	377
2	1.0000	0.9894	0.9947	189
3	0.9533	0.9167	0.9346	312
4	0.5789	0.3333	0.4231	33
5	0.8889	0.6797	0.7704	153
6	1.0000	0.7849	0.8795	279
7	0.6978	1.0000	0.8220	187
8	0.5405	1.0000	0.7018	60
9	0.9622	0.9558	0.9590	453
10	0.9291	0.9593	0.9440	123
11	0.8915	0.7667	0.8244	150
12	0.9364	0.9904	0.9626	624
13	0.8855	0.9431	0.9134	123
14	0.8704	0.9400	0.9038	300
15	0.9266	0.9619	0.9439	105
16	1.0000	0.9880	0.9940	585
17	0.9958	0.9764	0.9860	721
18	0.9897	1.0000	0.9948	96
19	0.9741	0.8433	0.9040	134
20	0.7867	0.8194	0.8027	72
accuracy			0.9307	5109
macro avg	0.8933	0.8870	0.8819	5109
weighted avg	0.9388	0.9307	0.9310	5109

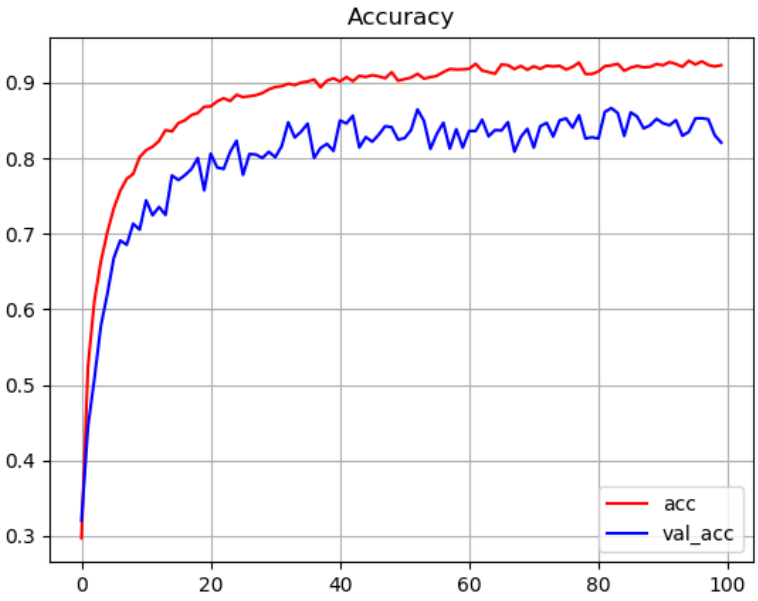
VGG16 - Transfer Learning

Input shape: (224, 224, 3)

Total params: 27,571,029
Trainable params: 12,856,341
Non-trainable params: 14,714,688

VGG16 is frozen and only top two dense layers are trained.





Results on the validation dataset:

	precision	recall	f1-score	support
0	0.6364	0.9333	0.7568	60
1	0.8134	0.9352	0.8701	648
2	1.0000	0.9839	0.9919	310
3	0.8658	0.9214	0.8927	140
4	0.5747	0.7937	0.6667	63
5	0.8824	0.4839	0.6250	93
6	0.7057	0.9935	0.8253	309
7	0.8547	0.8130	0.8333	123
8	1.0000	0.4667	0.6364	90
9	0.5754	0.9717	0.7228	318
10	0.8654	0.5000	0.6338	90
11	0.8559	0.6169	0.7170	154
12	0.9895	0.6013	0.7480	627
13	0.9655	0.4667	0.6292	120
14	0.7661	0.8733	0.8162	150
15	0.9506	0.7130	0.8148	108
16	0.7709	1.0000	0.8707	138
17	1.0000	0.7920	0.8839	125
18	0.9527	0.9938	0.9728	162
19	0.9568	0.9236	0.9399	144
20	1.0000	0.8095	0.8947	126
accuracy			0.8206	4098
macro avg	0.8563	0.7898	0.7972	4098
weighted avg	0.8599	0.8206	0.8163	4098

VGG16 - Fine Tuning

Input shape: (224, 224, 3)

In this phase, several top layers of VGG16 are also trained.
This part is still to be done.

Model optimisation for the two specified classes (Part 2)

This can be done in the following ways:

1. To set up `class_weight` in `model.fit` in such a way that the loss be more penalised for the classes (12 and 13) for which we want to optimise the model

2. First, make a model (Model 1) that has 3 classes: 12, 13 and *other*. Then, a make model for all other classes except for 12 and 13 (Model 2). In inference, first run Model 1 and then run Model 2 if necessary.

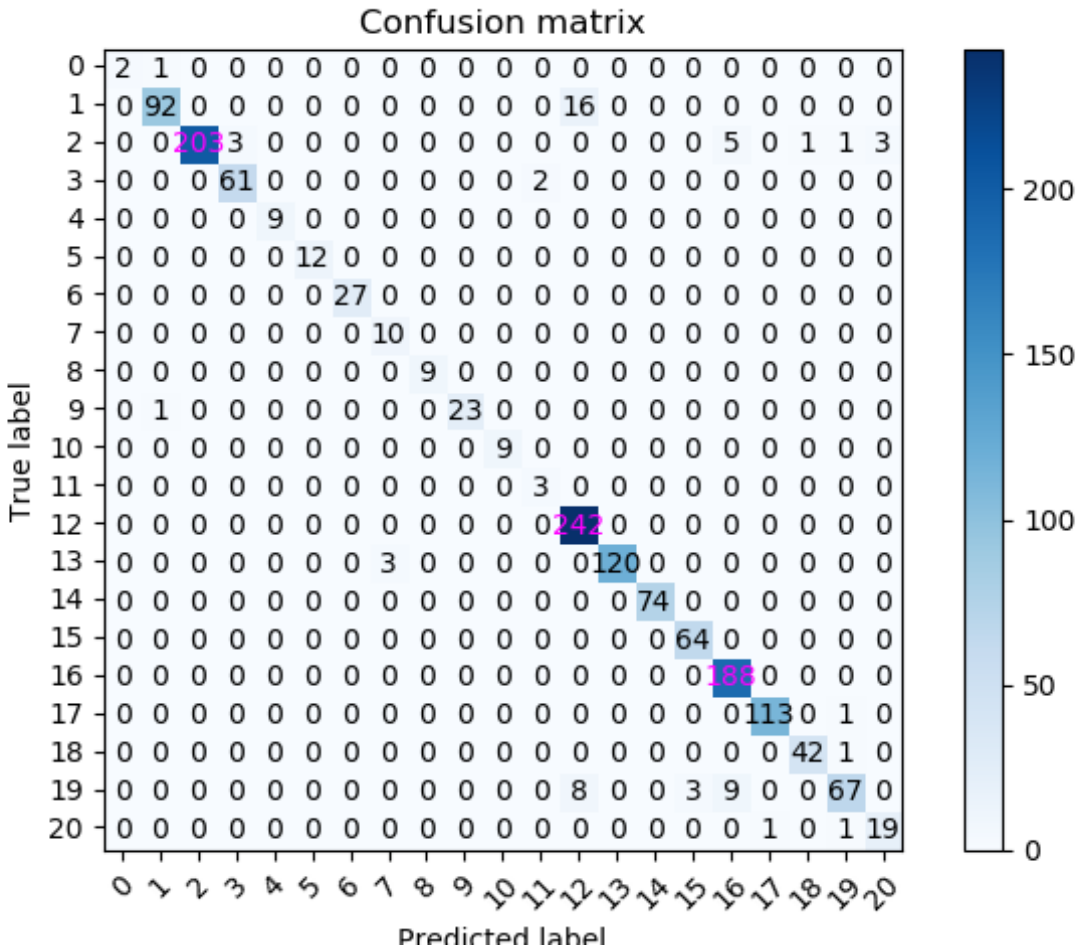
I haven't had enough time to try these approaches.

Evaluation on the test dataset

The best model I have selected from the validation procedure shown above is the "small VGG model".

I didn't have enough time to retrain the model on the whole dataset, thus this evaluation is from the model made on the training dataset only.

	precision	recall	f1-score	support
0	1.0000	0.6667	0.8000	3
1	0.9787	0.8519	0.9109	108
2	1.0000	0.9398	0.9690	216
3	0.9531	0.9683	0.9606	63
4	1.0000	1.0000	1.0000	9
5	1.0000	1.0000	1.0000	12
6	1.0000	1.0000	1.0000	27
7	0.7692	1.0000	0.8696	10
8	1.0000	1.0000	1.0000	9
9	1.0000	0.9583	0.9787	24
10	1.0000	1.0000	1.0000	9
11	0.6000	1.0000	0.7500	3
12	0.9098	1.0000	0.9528	242
13	1.0000	0.9756	0.9877	123
14	1.0000	1.0000	1.0000	74
15	0.9552	1.0000	0.9771	64
16	0.9307	1.0000	0.9641	188
17	0.9912	0.9912	0.9912	114
18	0.9767	0.9767	0.9767	43
19	0.9437	0.7701	0.8481	87
20	0.8636	0.9048	0.8837	21
accuracy			0.9586	1449
macro avg	0.9463	0.9525	0.9438	1449
weighted avg	0.9612	0.9586	0.9580	1449



Possible improvements in modelling

- Make a model in such a way that images don't have to be resized - with the use of global average (maximum) pooling
- Contrast Limited Adaptive Histogram Equalization
- For edge.mobile devices: quantisation - less precision of the model weights
- Try depth wise separable convolution in "small VGG" model
- It is worth trying other architectures such as residual networks
- Try EfficientNet and rescaling the neural network architecture up/down in all three dimensions (depth, width, image resolution)