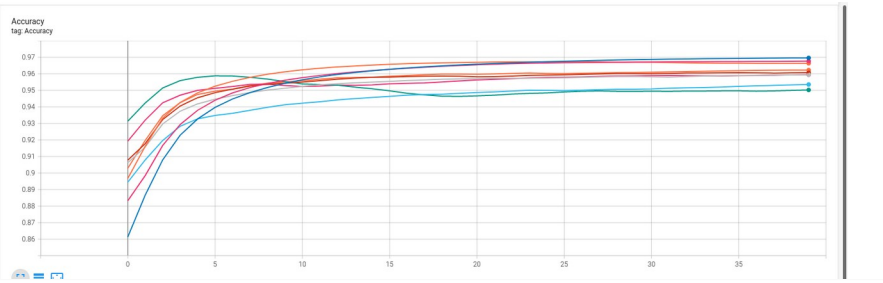
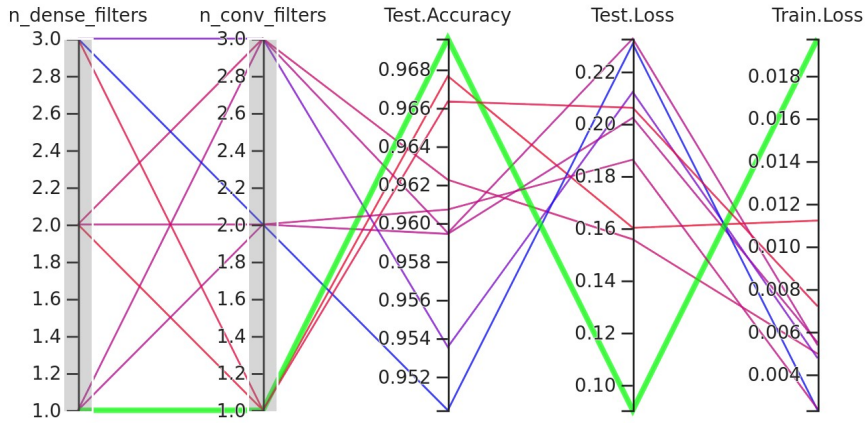
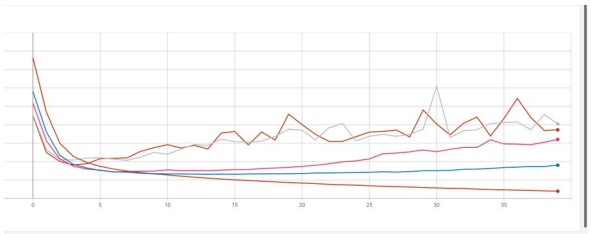
Influence of number of 2d convolutional layers and dense layers
an higher number of dense layers results in the highest test loss and smaller accuracy





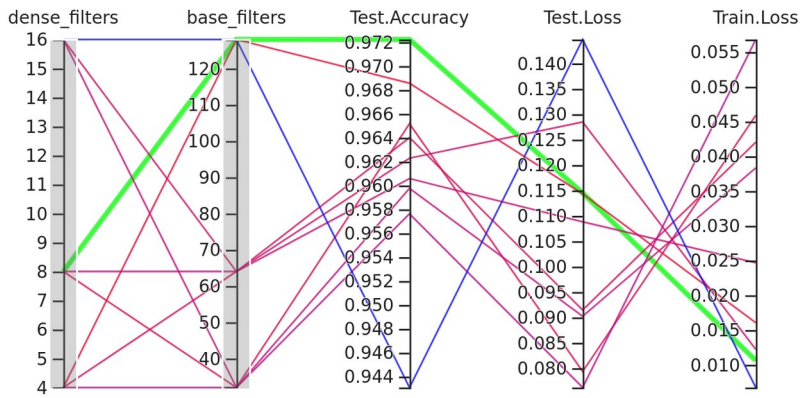
dense should be either 1 or 2. 2 being more robust and 1 potentially giving the best results

1 convolutional layers results in the lowest test loss but the highest train loss. 3 convolutional layers results in the highest test loss and small train loss. 1 or 2 convolutional layers should be kept.
We can see with the loss curves that more layers cause faster over-fitting (especially visible when adding multiple convolutional layers)

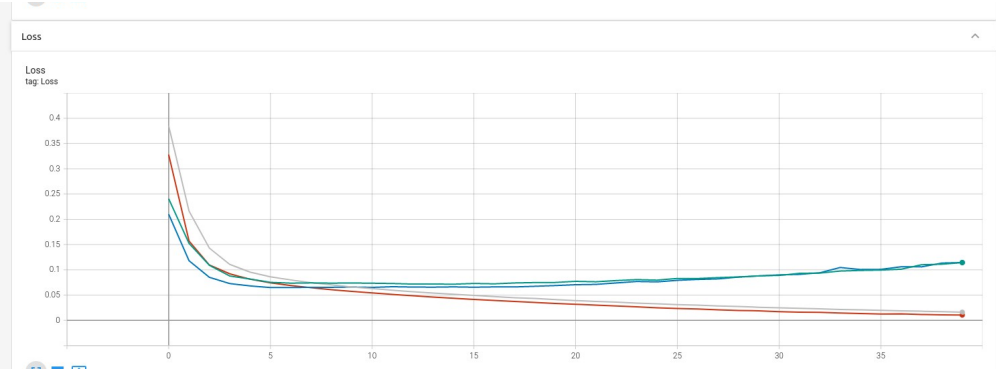


We keep 1 convolutional layer and 1 dense layer

Influence of number of filters for 2d convolutional layers and dense layers

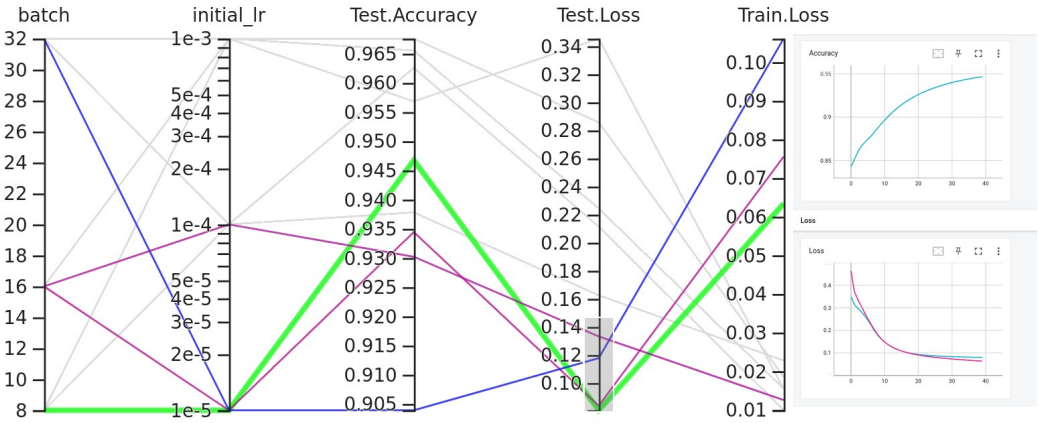


We observed that an higher number of dense filters give the worst results. The contrary is observed for convolutional filters.

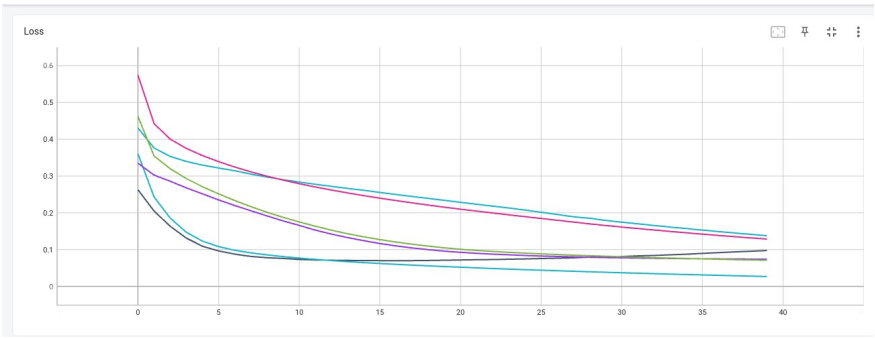


Additionally. A lower number of dense filters results in more over-fitting. We thus select 128 convolutional filters and 8 dense filters

Influence of learning rate and batch size

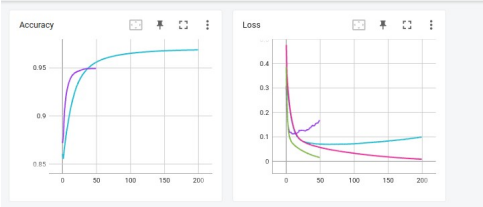


We observe that all the non over-fitting models have an initial_lr of 1e-5. The training is quite stable and the model could be trained more. Lr might be slightly tuned up



In fact a larger lr (4e-5) converge even faster but tends to over-fit soon. Can we achieve better performance with 1e-5 if we train the model long enough?

Influence of number of epochs



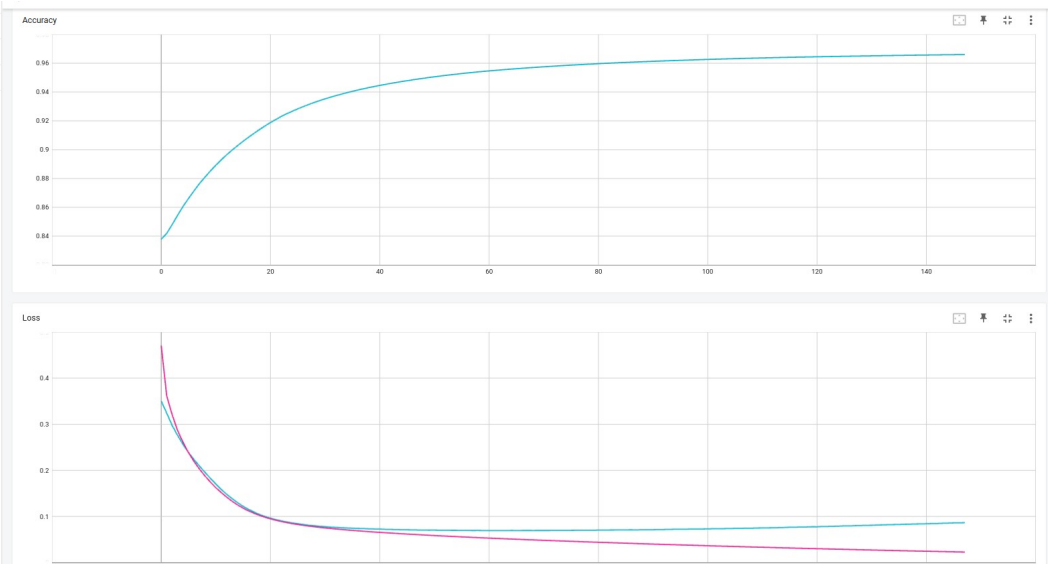
In effect, using a lr of 1e-5 we can train the model longer without encurring in too much over-fitting. We see how 4e-5 diverges quickly.

Parameters chosen

The chosen model architecture and hyper-parameters are:

```
{'epochs': 150, 'batch': 8, 'initial_lr': 1e-05, 'base_filters': 128, 'dense_filters': 8, 'n_conv_filters': 1, 'n_de
nse_filters': 1}

<tf.Tensor: shape=(), dtype=bool, numpy=True>

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 9, 9, 3)           0

 conv2d (Conv2D)             (None, 9, 9, 128)         3584

 batch_normalization (BatchN  (None, 9, 9, 128)        512
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 4, 4, 128)        0
 )

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 8)                 16392

 dense_1 (Dense)             (None, 2)                 18

=================================================================
Total params: 20,506
Trainable params: 20,250
Non-trainable params: 256
_________________________________________________________________
```

The model reaches an accuracy of around 96.6 %
The total training time on a RTX3090 was 9 minutes and 49 seconds



Possible improvements

In order to correctly evaluate model performances a k-fold training strategy should be used. Furthermore, the data-set was only split in training and test for prototyping. The data-set should be split in a training (80%), validation (15%) and test (5%) for an accurate evaluation. The folds may also be exploited in order to create an ensemble of deep neural network using the same hyper-parameters but with different training data. In order to avoid over-fitting two-dimensional dropout layers could be added between convolutions. Finally, the model presented here has a simple architecture composed by only 2D convolutions and dense layers, a more complex architecture could better exploits data features and consequently improving performances.