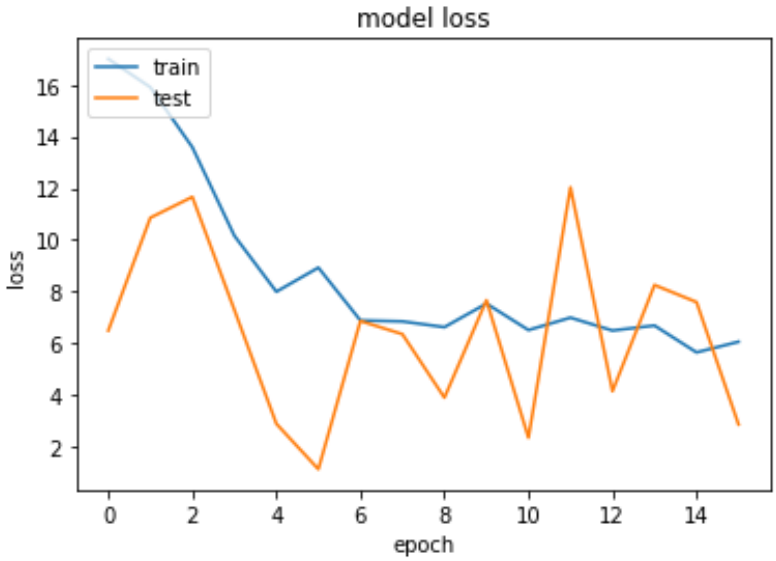
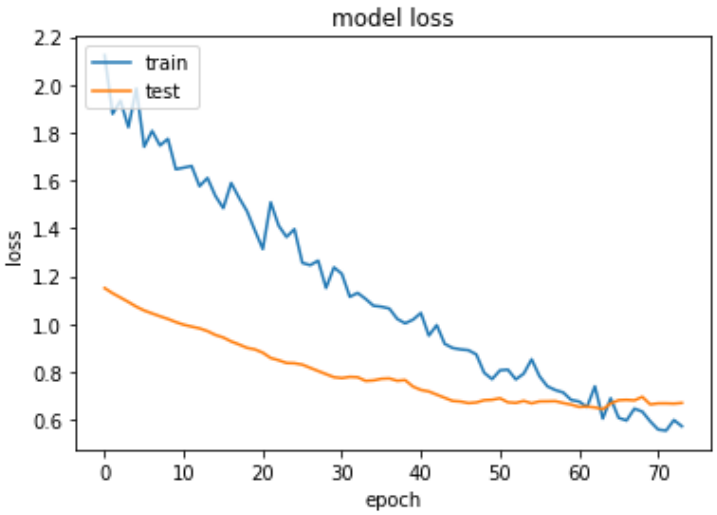
- I delete the early_stopping for now. Because it stopst the training too early like at 16. epoch



- Okk I don't know why but my code start training with too high loss and val acc doesn't increase. So I will start from manos's code and do the changes step by step to see where does the problem come

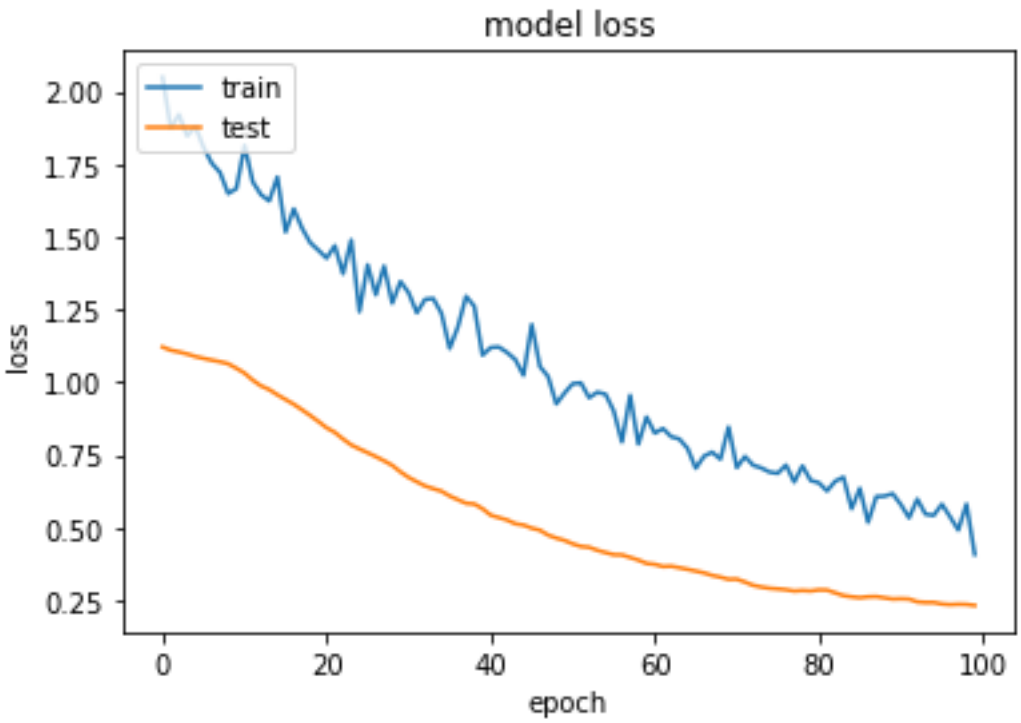
Starting from the beginning.

- Manos'us code without any change (it stops at 74. Epoch and acc's seems ok)



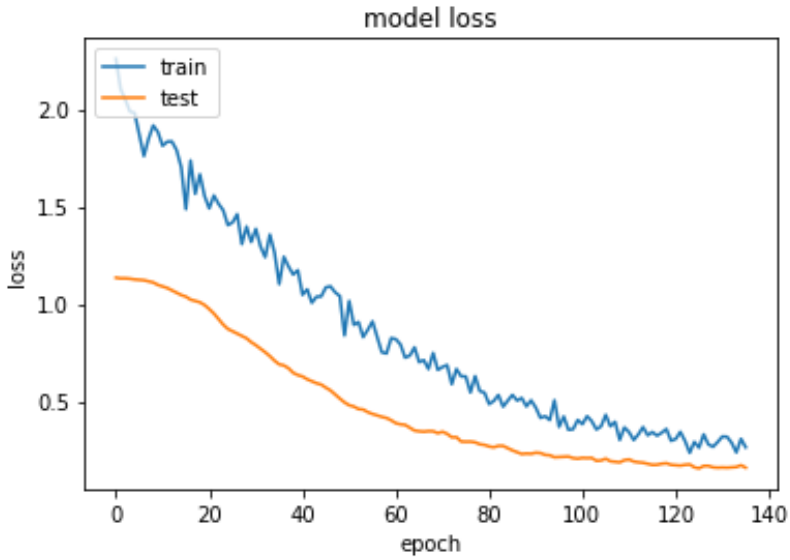

```
16/16 [==============================] - 10s 599ms/step - loss: 0.5544 -
accuracy: 0.7812 - val_loss: 0.6686 - val_accuracy: 0.7525
Epoch 73/100
15/16 [===========================>..] - ETA: 0s - loss: 0.6093 - accuracy:
0.7711
Epoch 00073: val_loss did not improve from 0.64415
16/16 [==============================] - 9s 594ms/step - loss: 0.5987 -
accuracy: 0.7792 - val_loss: 0.6675 - val_accuracy: 0.7525
Epoch 74/100
15/16 [===========================>..] - ETA: 0s - loss: 0.5737 - accuracy:
0.7578
Epoch 00074: val_loss did not improve from 0.64415
16/16 [==============================] - 9s 589ms/step - loss: 0.5728 -
accuracy: 0.7604 - val_loss: 0.6703 - val_accuracy: 0.7475
```

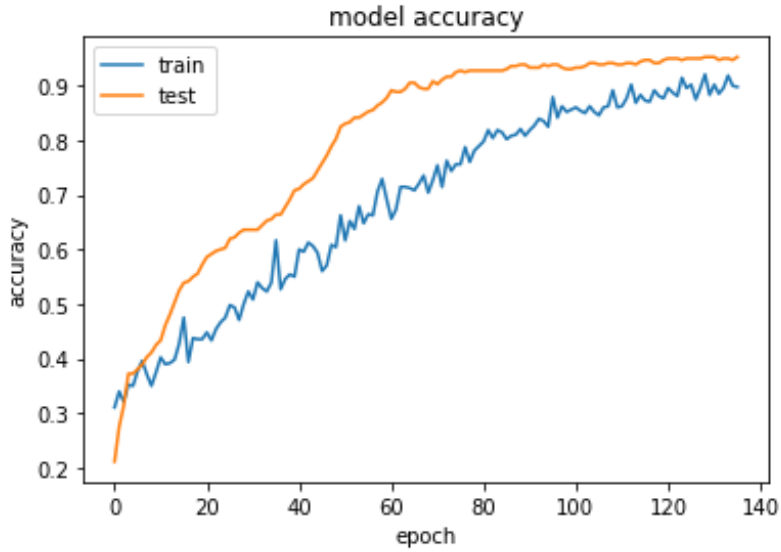
- Manos'us code without any change but with the new dataset (It doesn't go to early stop and actually we see train and val loss still decrease so I will increase the epoch number




```
Epoch 00099: val_loss did not improve from 0.23544
16/16 [==============================] - 13s 823ms/step - loss: 0.5813 -
accuracy: 0.7792 - val_loss: 0.2362 - val_accuracy: 0.9333
Epoch 100/100
15/16 [===========================>..] - ETA: 0s - loss: 0.4173 - accuracy:
0.8400
Epoch 00100: val_loss improved from 0.23544 to 0.23248, saving model to
model.weights.best.embedding_network.hdf5
16/16 [==============================] - 13s 841ms/step - loss: 0.4082 -
accuracy: 0.8438 - val_loss: 0.2325 - val_accuracy: 0.9333
```

- Manos'us code with input size = 100x100 and epoch = 140






```
Epoch 00135: val_loss did not improve from 0.15591
16/16 [==============================] - 12s 774ms/step - loss: 0.3073 -
accuracy: 0.9000 - val_loss: 0.1719 - val_accuracy: 0.9472
Epoch 136/140
15/16 [===========================>..] - ETA: 0s - loss: 0.2552 - accuracy:
0.8956
Epoch 00136: val_loss did not improve from 0.15591
16/16 [==============================] - 12s 772ms/step - loss: 0.2642 -
accuracy: 0.8979 - val_loss: 0.1596 - val_accuracy: 0.9528
```

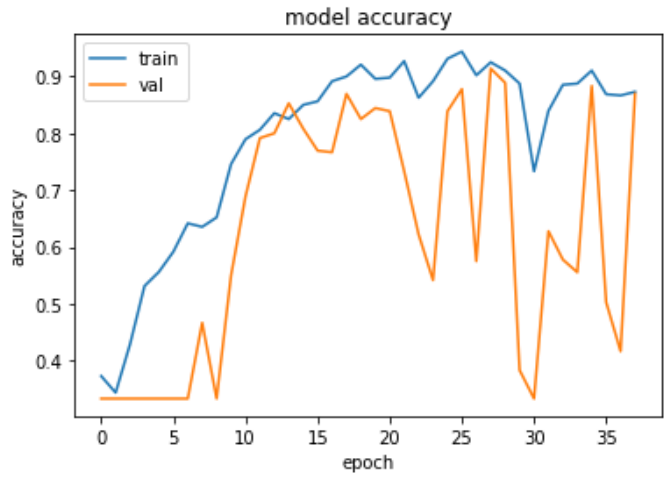
The training stops with early stop at 136 epoch. Soo 140 epoch was okk we don't need more.

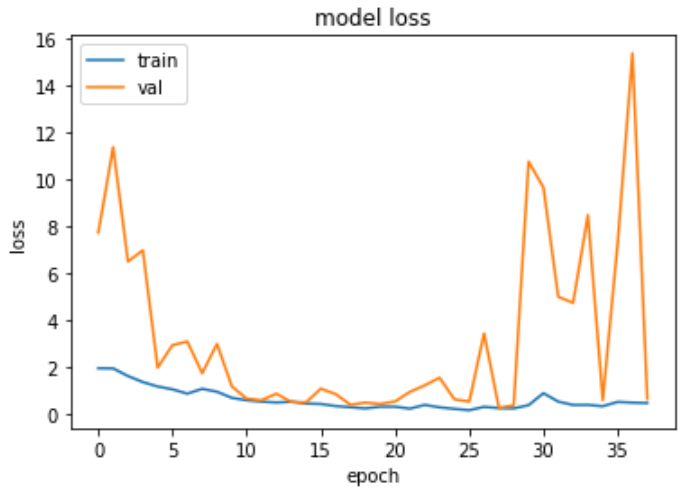
- Changing learning rate from 0.0001 to learning scheduler

```
def lr_scheduler(epoch, lr=0.1):
   i = int(epoch/20)
   if epoch < 20:
     return lr
   else:
     return lr * tf.math.exp(pow(10,-i))

lr_ = tf.keras.callbacks.LearningRateScheduler(lr_scheduler, verbose=1)
```

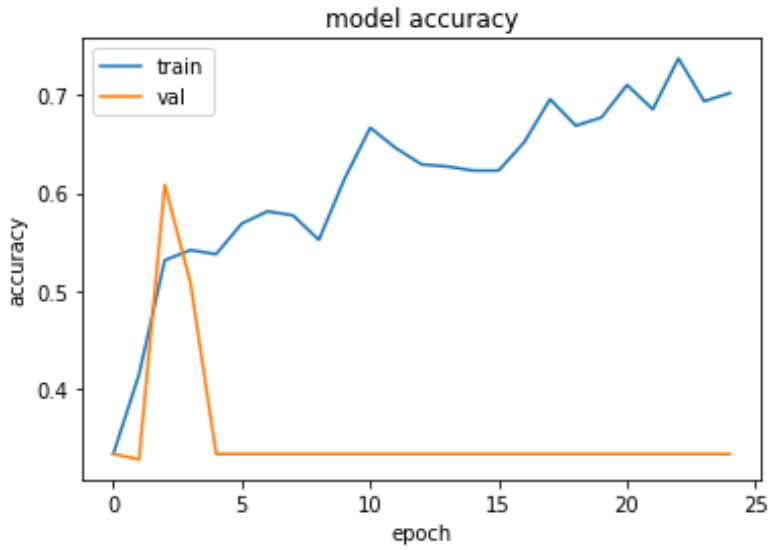
ULALAAA I did it from scratch but look the accuracy got messed up. I will use some google prepared learning scheduler :D

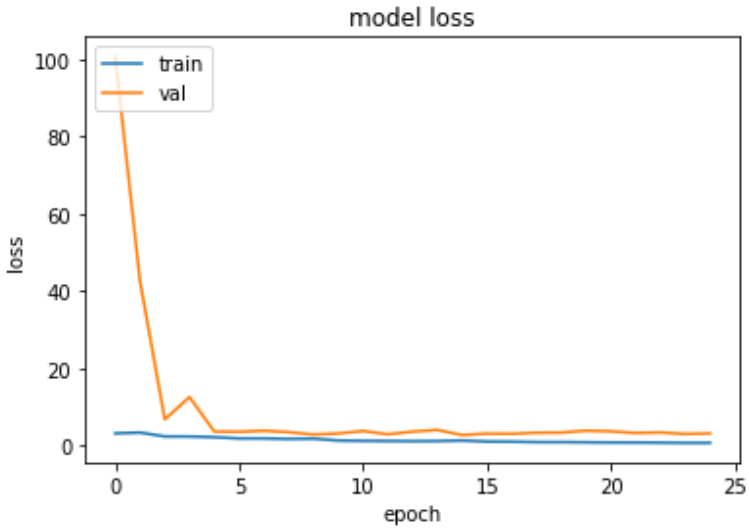




```python
def lr_exp_decay(epoch, lr):
    k = 0.1
    return initial_learning_rate * math.exp(-k*epoch)

lr_ = tf.keras.callbacks.LearningRateScheduler(lr_exp_decay, verbose=1)
```





Okkkk step decay doesn't work for us I don't know whyyy. The possible reason:
In transfer learning, learning rate should be very very low
otherwise the weights change too much even in the first epoch... so we loose the pretrain benefits.

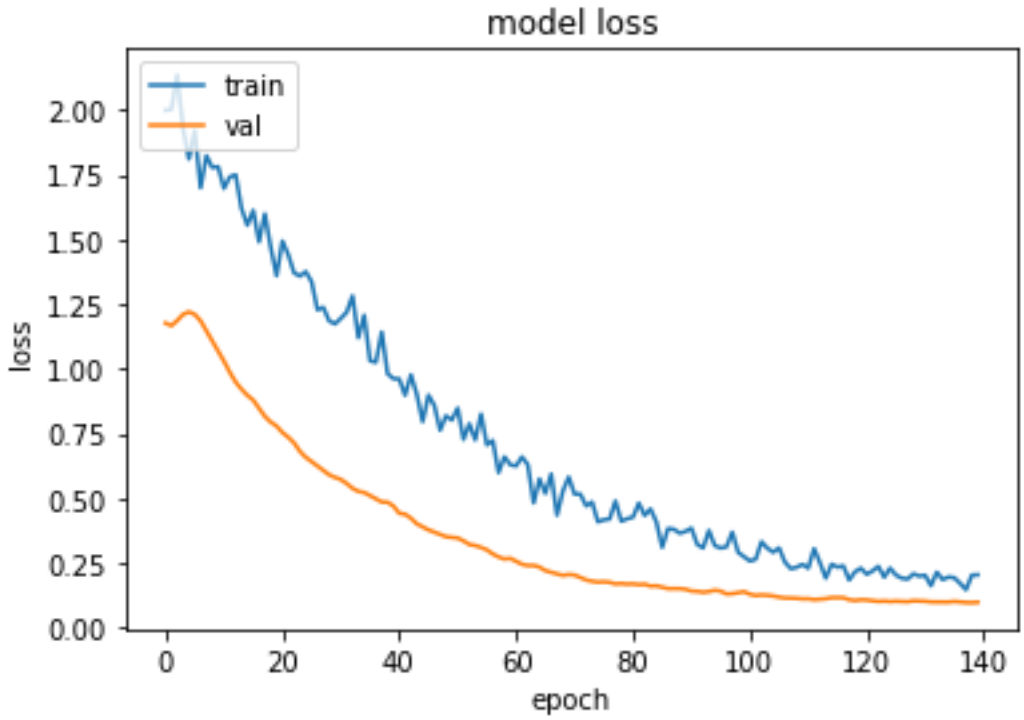
- Why do we have untrainable params after adding last 3 dense to the vgg network??

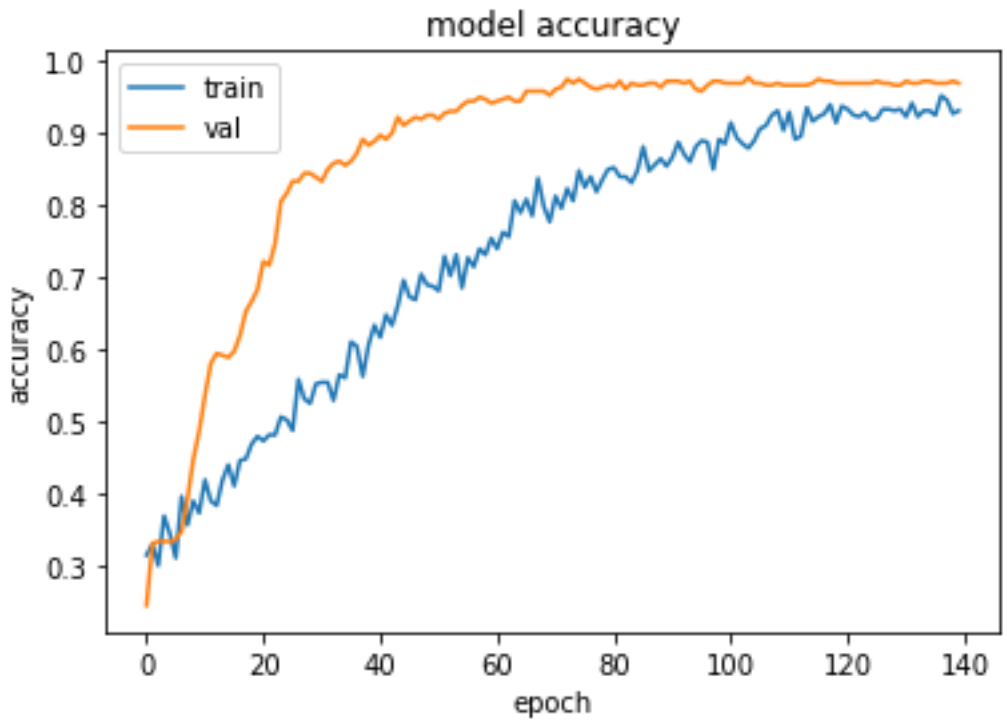
```
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

```
Total params: 15,149,379
Trainable params: 15,147,843
Non-trainable params: 1,536
```

In general, all weights are trainable weights. The only built-in layer that has non-trainable weights is the BatchNormalization layer. It uses non-trainable weights to keep track of the mean and variance of its inputs during training. OKK its normal

- **FINAL MODEL**





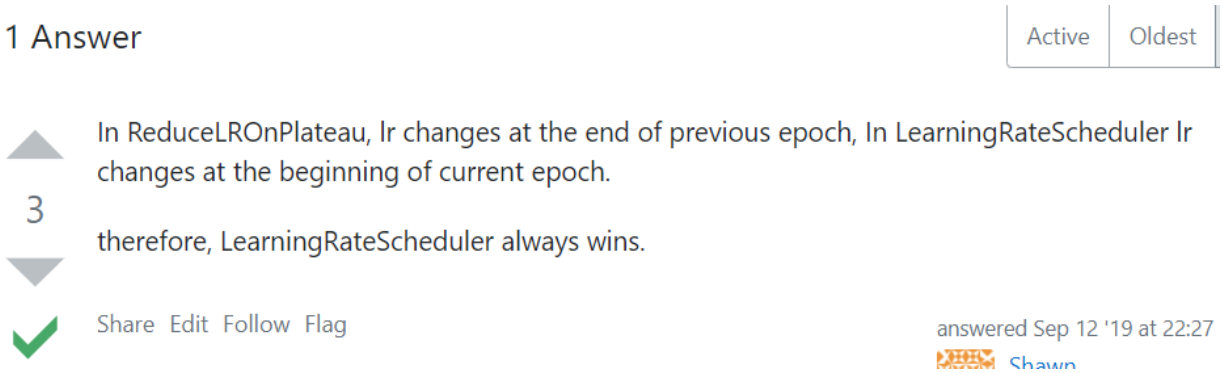

```
0.0986 - val_accuracy: 0.9694
Epoch 138/140
23/24 [============================>..] - ETA: 0s - loss: 0.1488 - accuracy: 0.9457
Epoch 00138: val_loss improved from 0.09831 to 0.09663, saving model to embedding_network.h5
24/24 [==============================] - 11s 471ms/step - loss: 0.1462 - accuracy: 0.9458 - val_loss:
0.0966 - val_accuracy: 0.9694
Epoch 139/140
23/24 [============================>..] - ETA: 0s - loss: 0.2085 - accuracy: 0.9261
Epoch 00139: val_loss improved from 0.09663 to 0.09617, saving model to embedding_network.h5
24/24 [==============================] - 11s 473ms/step - loss: 0.2023 - accuracy: 0.9271 - val_loss:
0.0962 - val_accuracy: 0.9722
Epoch 140/140
23/24 [============================>..] - ETA: 0s - loss: 0.2092 - accuracy: 0.9304
Epoch 00140: val_loss did not improve from 0.09617
24/24 [==============================] - 11s 463ms/step - loss: 0.2036 - accuracy: 0.9312 - val_loss:
0.0976 - val_accuracy: 0.9694
```

```
Test loss: 0.12065759352408349
Test Acc. = 0.9646464646464646
```

pretrain_mano_changes.py
embedding_network.h5

Additional Info I had during the experiments

1)



2)

conda install h5py==2.10.0 otherwise load.model gives "AttributeError: 'str' object has no attribute 'decode' " , while Loading a Keras Saved Model