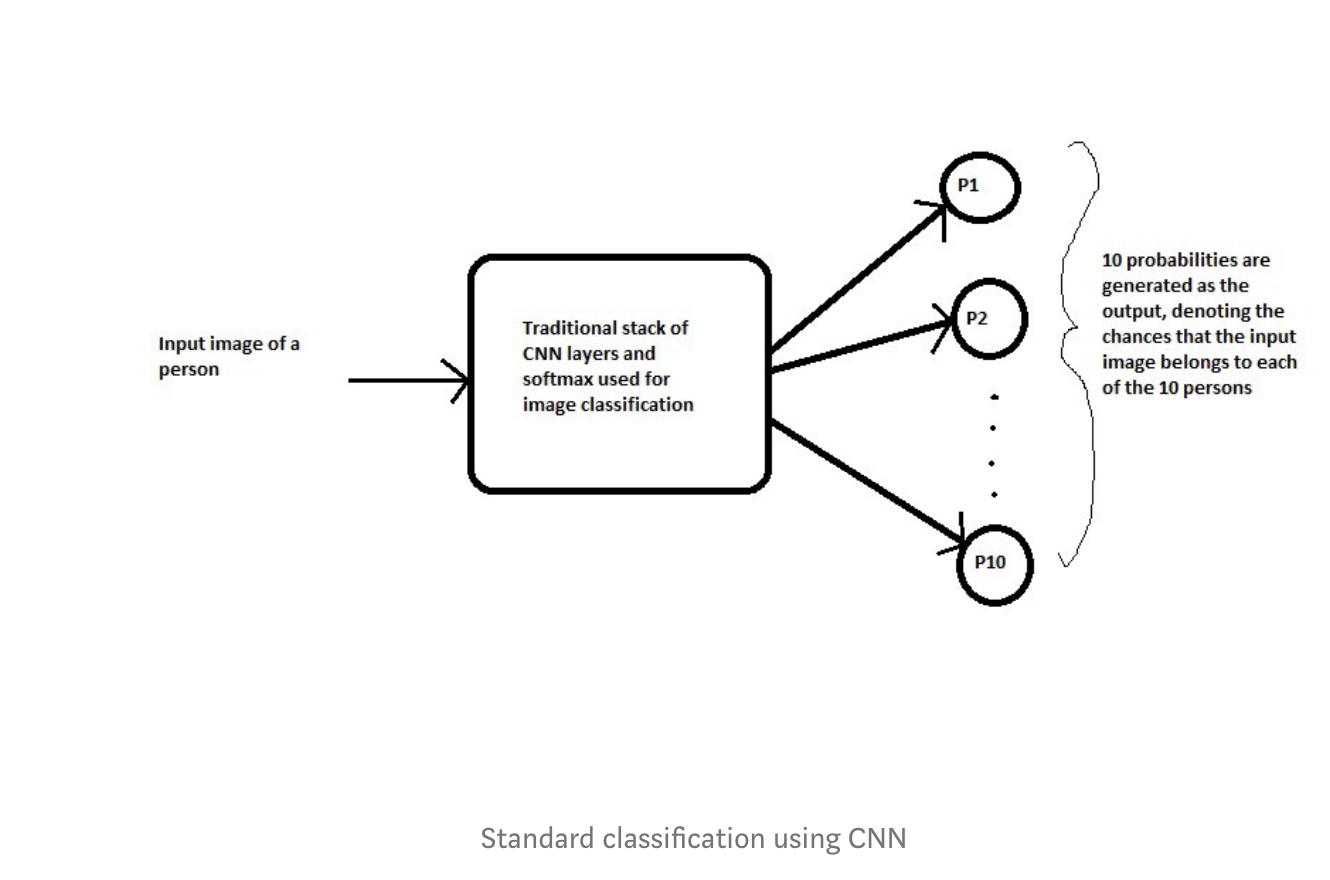
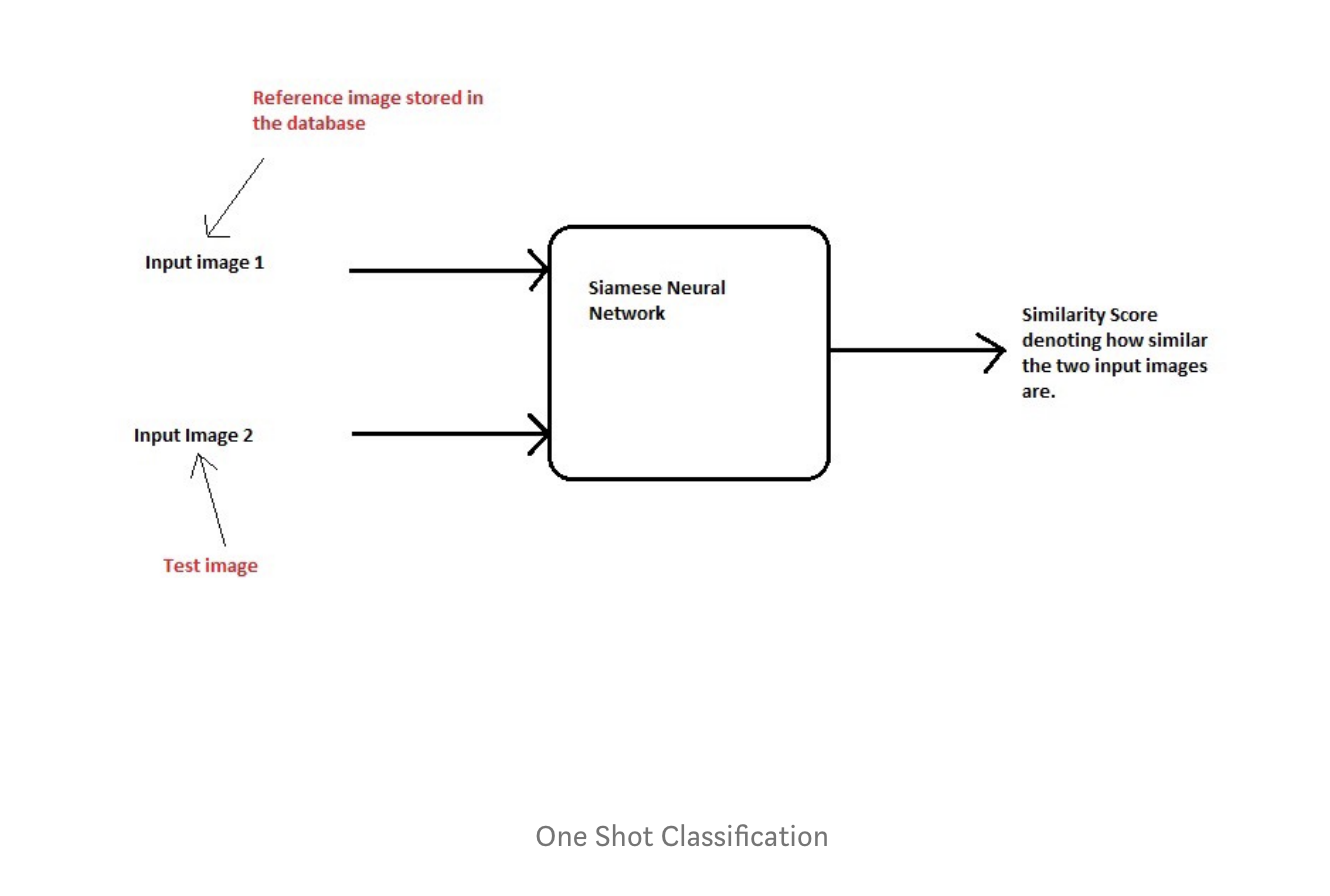
Traditional Classification:



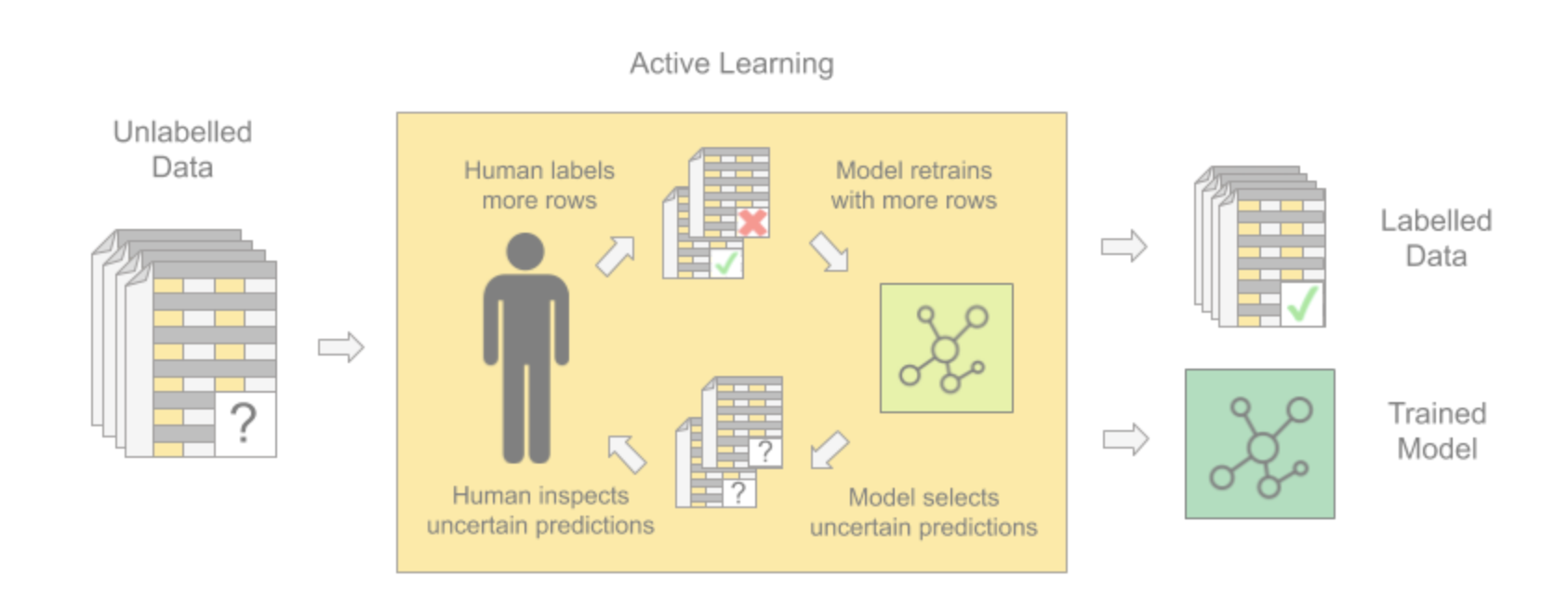
N-Shot (i.e. siamese network)



Instead of learning the characteristics of a new class n-shot uses calculates a similarity score with an example reference.

The intuition is that the feature vectors of the same class should be similar.

Examples: <https://medium.com/@prabhnoor0212/siamese-network-keras-31a3a8f37d04>



Source: <https://www.knime.com/blog/labeling-with-active-learning>

Testing the model:

# 1 Model

input\_1 = Input(shape=(X1.shape[1],))

input\_2 = Input(shape=(X2.shape[1],))

common\_embed = Embedding(name = "Sentence\_Embed",

input\_dim = len(w2v\_model\_2.wv.vocab) + 1 ,

output\_dim = 300,

input\_length = X1.shape[1],

weights = [embedding\_matrix],

trainable=False)

lstm\_1 = common\_embed(input\_1)

lstm\_2 = common\_embed(input\_2)

common\_lstm = LSTM(256,return\_sequences=True, activation="relu")

vector\_1 = common\_lstm(lstm\_1)

vector\_1 = Flatten()(vector\_1)

vector\_2 = common\_lstm(lstm\_2)

vector\_2 = Flatten()(vector\_2)

x3 = Subtract()([vector\_1, vector\_2])

x3 = Multiply()([x3, x3])

x1\_ = Multiply()([vector\_1, vector\_1])

x2\_ = Multiply()([vector\_2, vector\_2])

x4 = Subtract()([x1\_, x2\_])

# https://stackoverflow.com/a/51003359/10650182

# Calculates cosine similarity

x5 = keras.layers.Dot(axes = 1, normalize=True)([vector\_1, vector\_2])

conc = Concatenate(axis = -1)([x5, x4, x3])

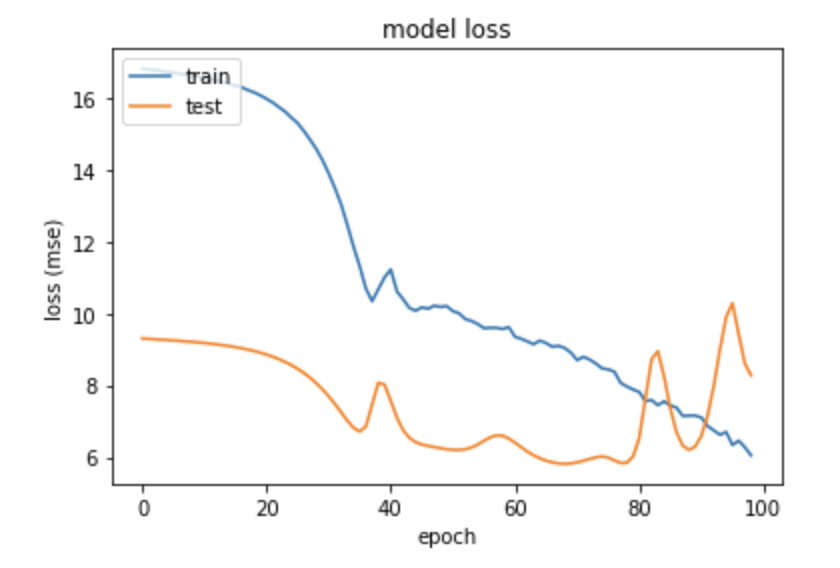
x = Dense(100, activation="relu", name='conc\_layer')(conc)

x = Dropout(0.2)(x)

out = Dense(1, activation="relu", name = 'out')(x)

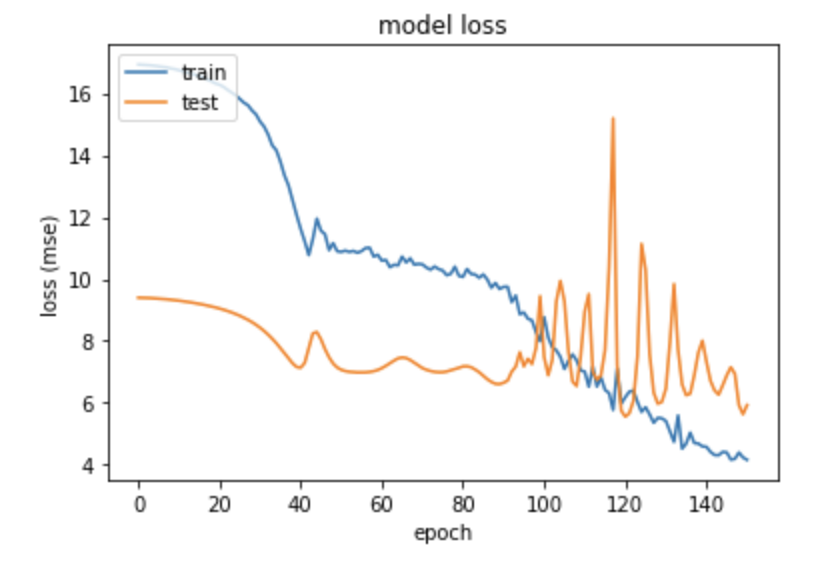
model = Model([input\_1, input\_2], out)

model.compile(loss= "mse", metrics=[RootMeanSquaredError(name='rmse')], optimizer=Adam(0.00005))

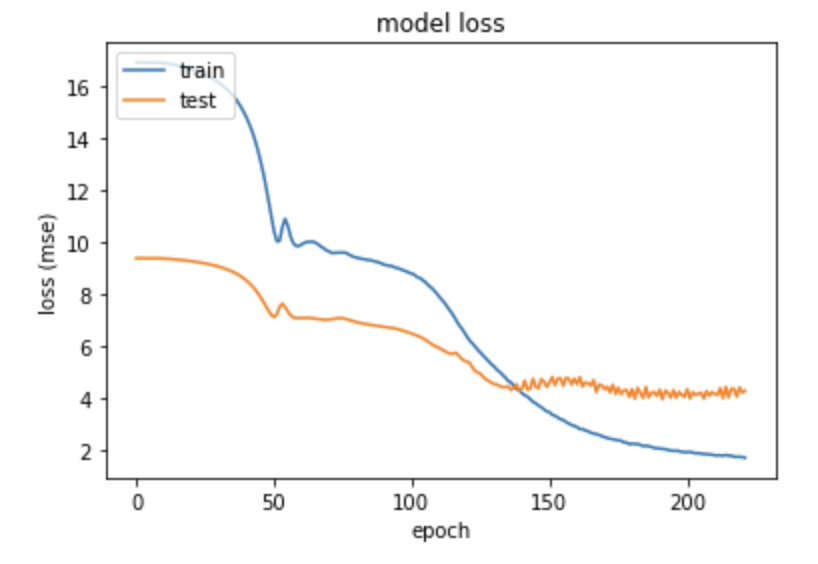


<https://machinelearningmastery.com/best-practices-document-classification-deep-learning/>

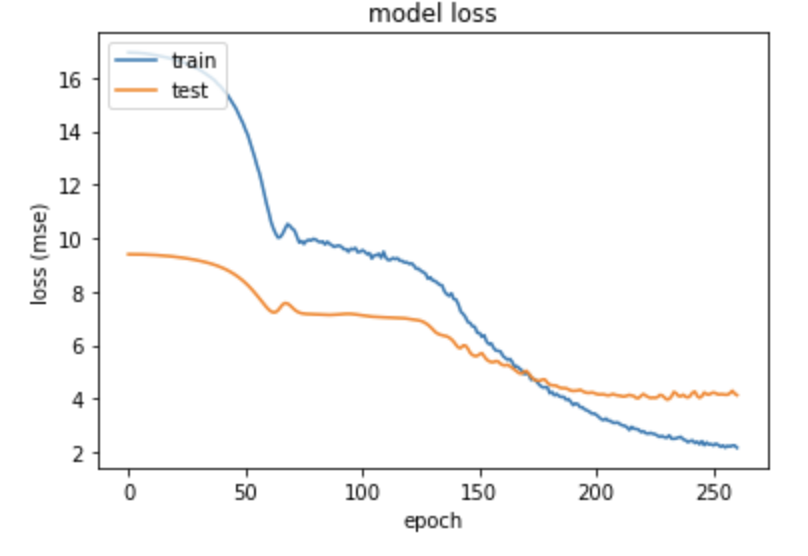
# 2. Model: dropout = 0.5



# 3. Model: dropout = 0.01 (not stemmed from here on out)

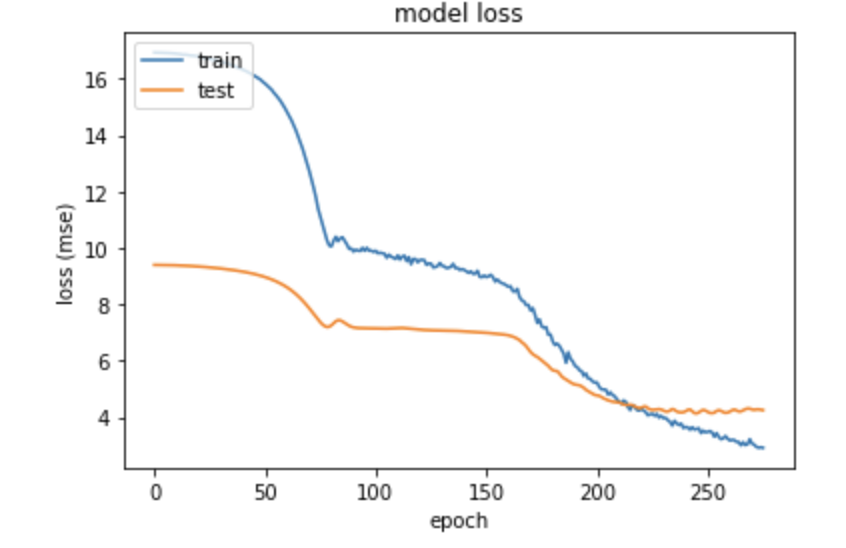


# 4. Model: LSTM(128) -> dropout back at 0.2



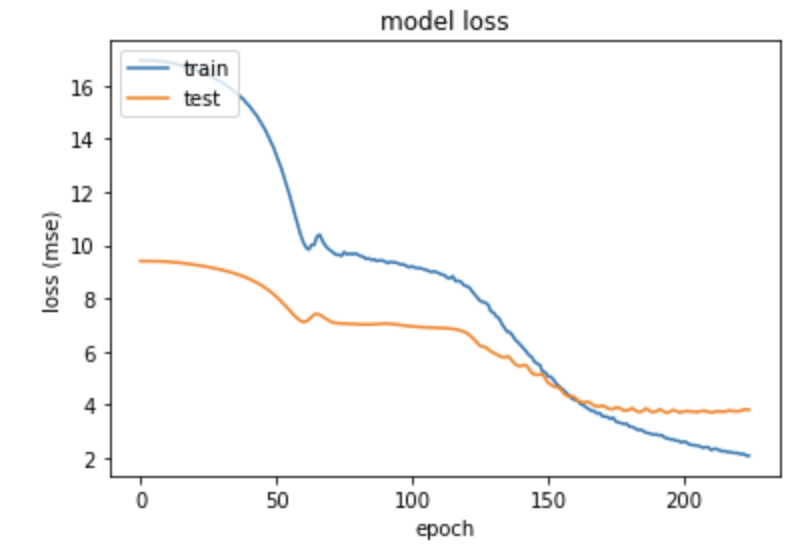
val\_rmse: 2.0327

# 5. Model: LSTM(64)



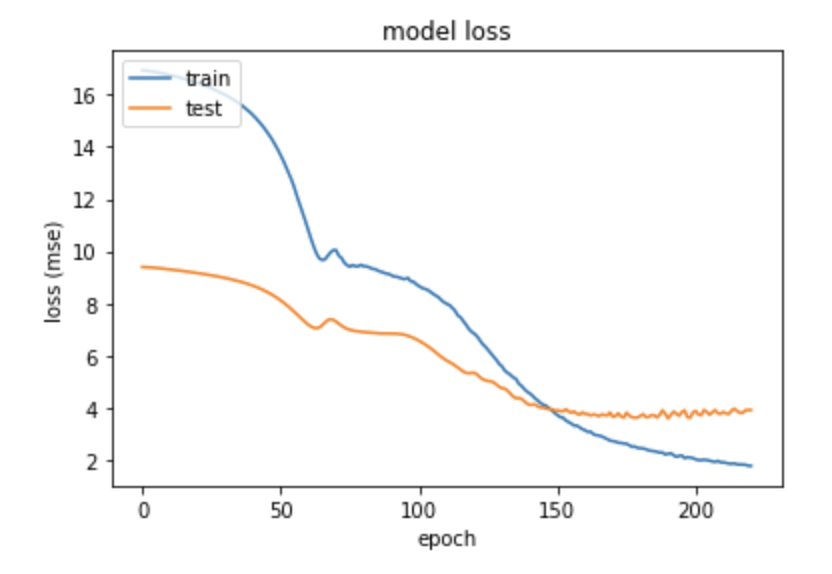
val\_rmse: 2.0612

# 6. Model LSTM(128), Dense(300)



val\_rmse: 1.9505

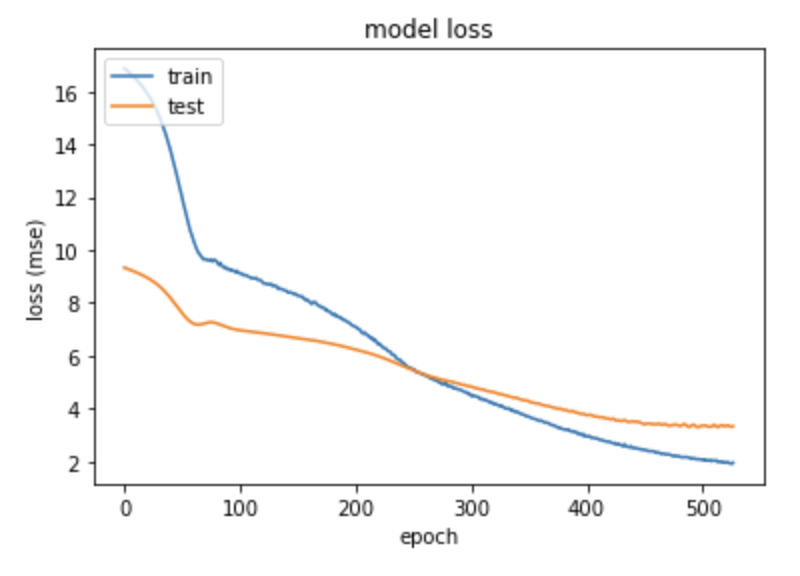
# 6. Model LSTM(128), Dense(500)



val\_rmse: 1.9784 (lowest was 1.91)

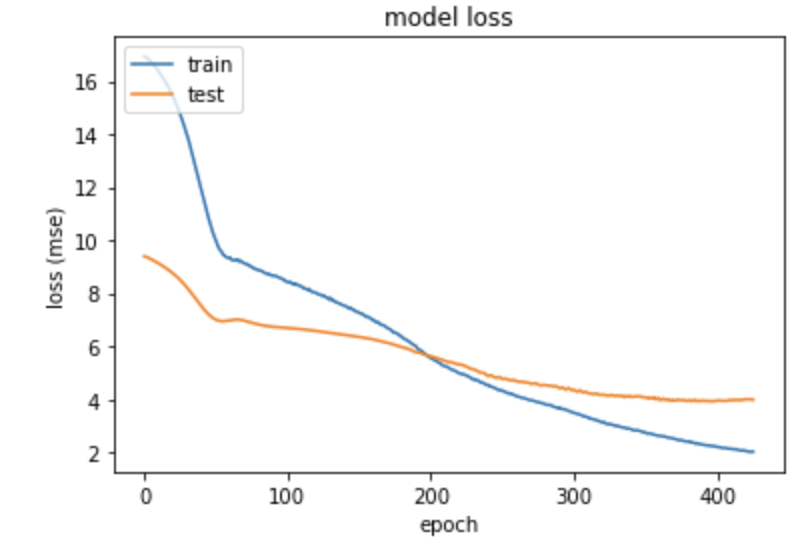
# 6. Model LSTM(128, activation="tanh"), Dense(300)

(activation was originally relu)



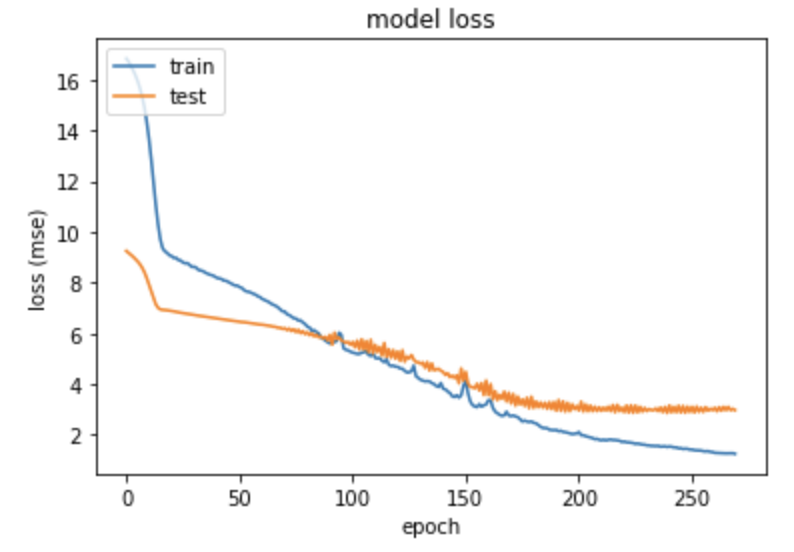
val\_rmse: 1.8241

# 7. Model LSTM(128, activation="tanh"), Dense(300, activation="tanh")



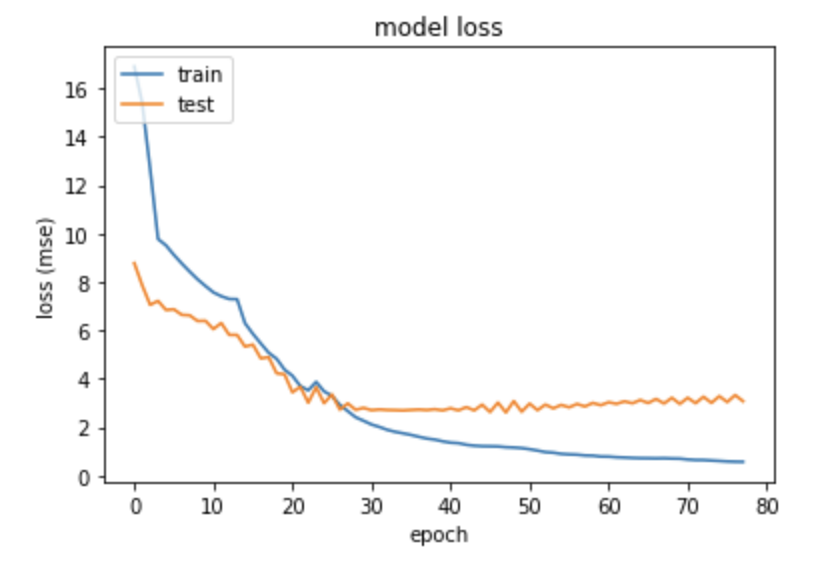
val\_rmse: 1.9976

# 8. Model LSTM(128, activation="tanh"), Dense(300), optimizer=Adadelta(learning\_rate=0.1, rho=0.95)



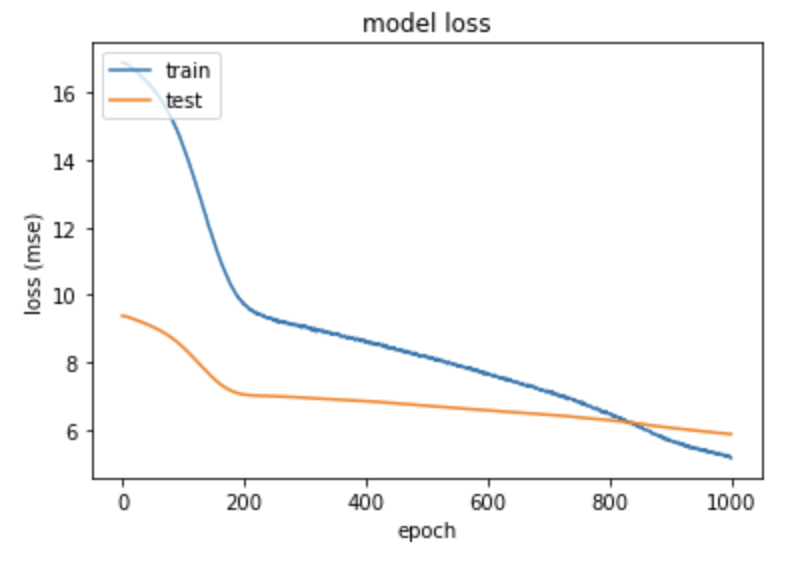
val\_rmse: 1.7250

# 9. Model LSTM(128, activation="tanh"), Dense(300), optimizer=Nadam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-07))



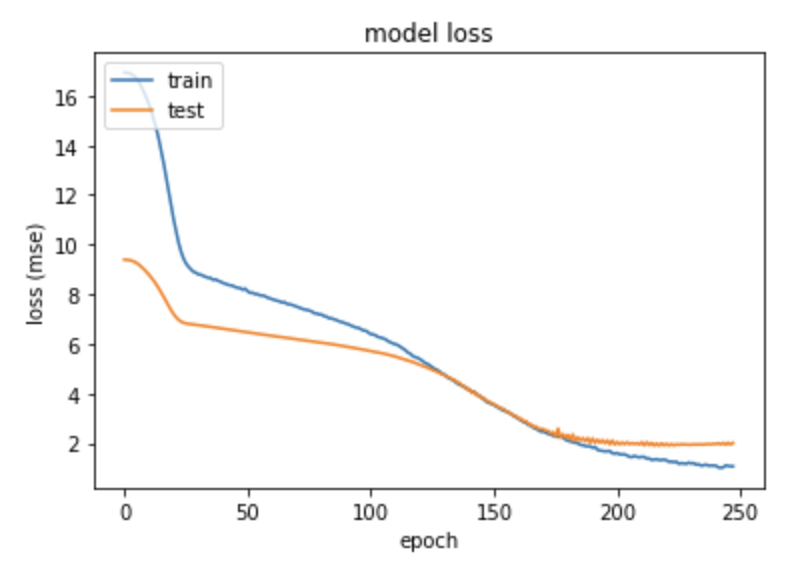
val\_rmse: 1.7532

# 9. Model LSTM(128, activation="tanh"), Dense(300), optimizer=Nadam(learning\_rate=0.0001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-07))



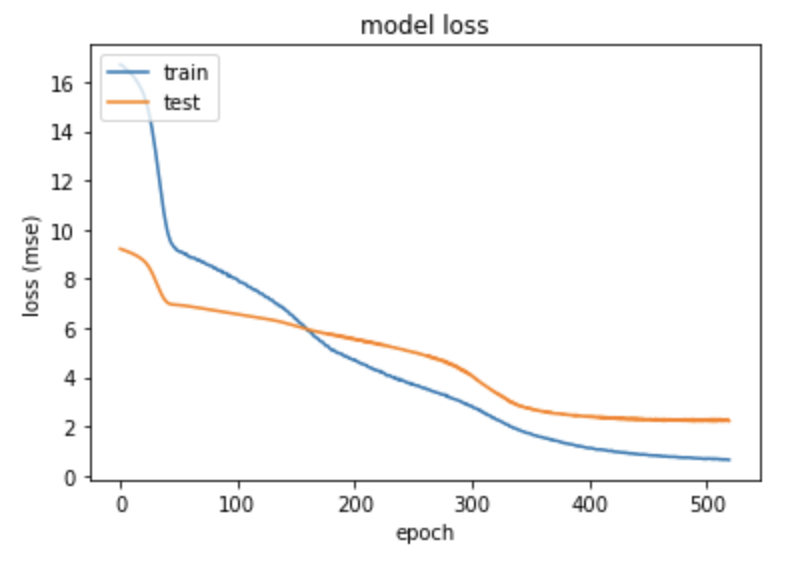
val\_rmse: 2.4228

# 9. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Adadelta(learning\_rate=0.05, rho=0.95))



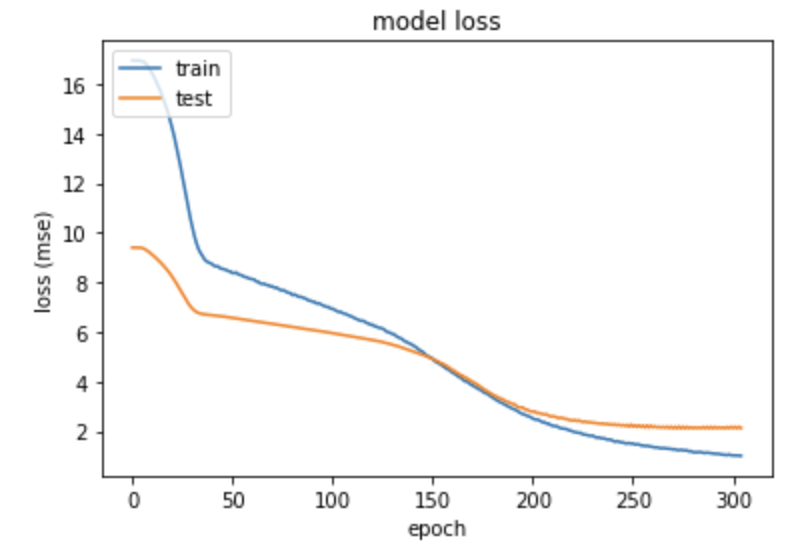
val\_rmse: 1.4182

# 10. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Adadelta(learning\_rate=0.05, rho=0.75))



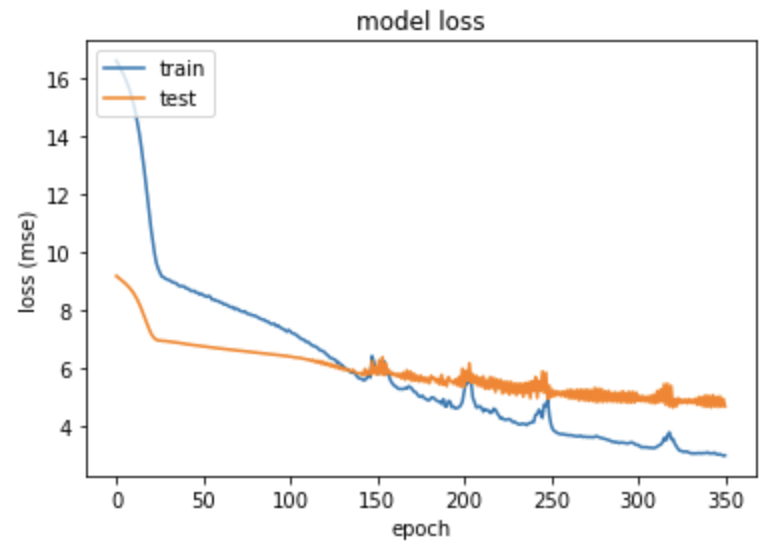
val\_rmse: 1.4920

# 11. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Adadelta(learning\_rate=0.05, rho=0.85))



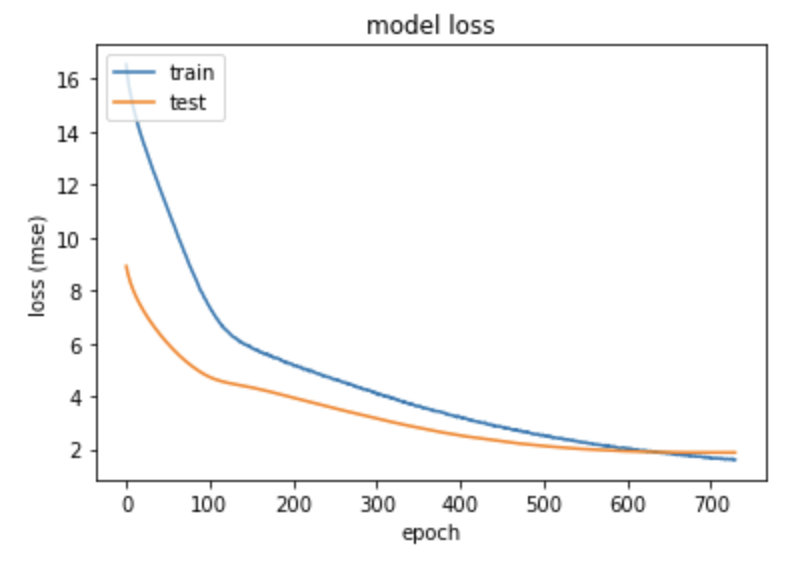
val\_rmse: 1.4562

# 12. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Adadelta(learning\_rate=0.05, rho=0.975))



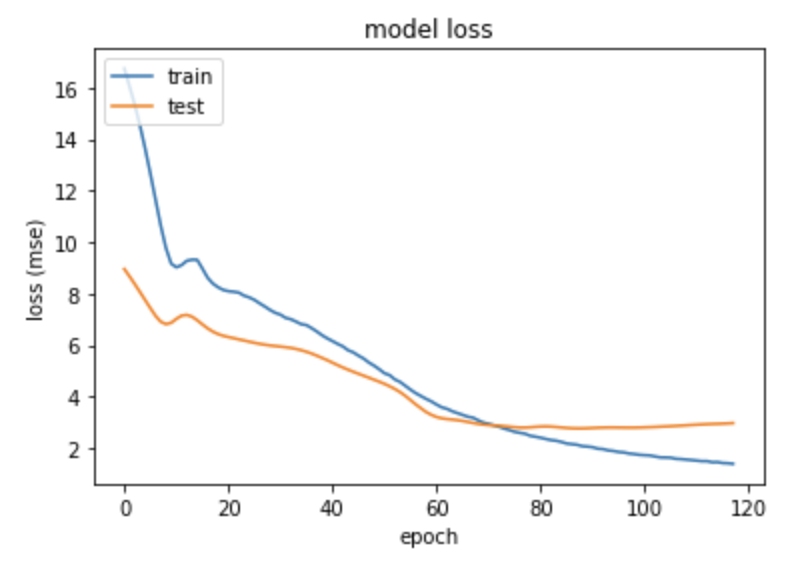
val\_rmse: 2.1574

# 13. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Adagrad(learning\_rate=0.001, initial\_accumulator\_value=0.1, epsilon=1e-07)



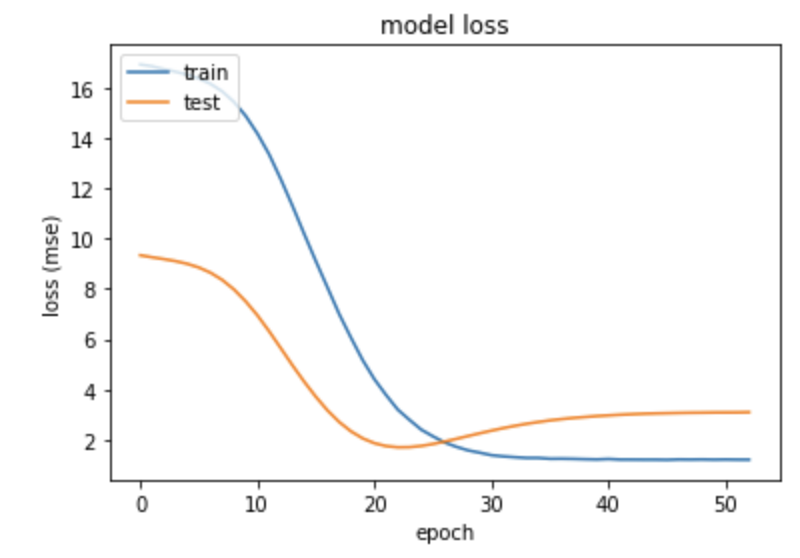
val\_rmse: 1.3695

# 14. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Adamax(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-07)



val\_rmse: 1.7229

# 14. Model LSTM(128, activation="tanh"), Dense(300), optimizer= Ftrl(learning\_rate=0.01, learning\_rate\_power=-0.5, initial\_accumulator\_value=0.1, l1\_regularization\_strength=0.0, l2\_regularization\_strength=0.0)

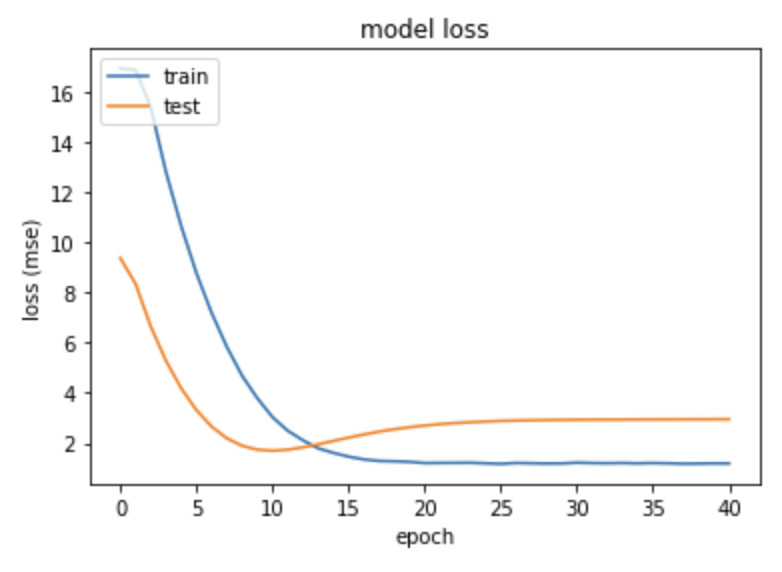


Last: val\_rmse: 1.7583

Best: val\_rmse: 1.3027

# 15. Model RMSprop doesn’t work well

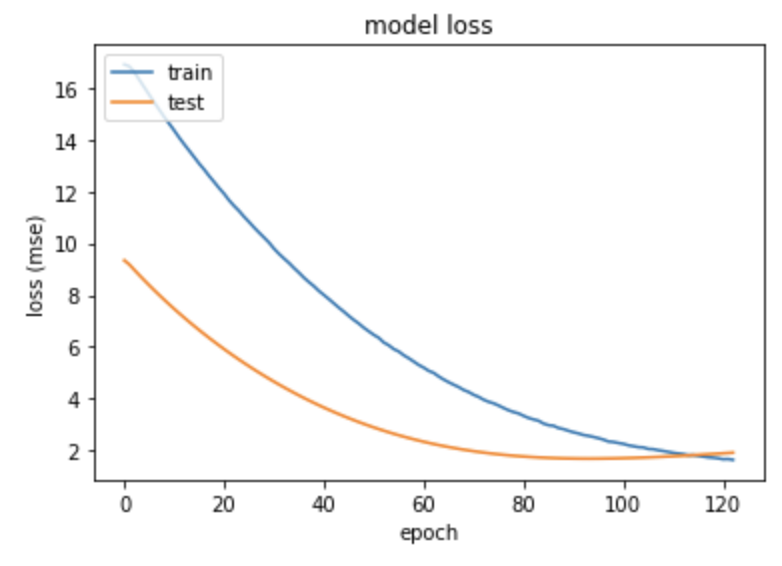
# 16. Model LSTM(128, activation="tanh"), Dense(300), optimizer= SGD(learning\_rate=0.01, momentum=0.0, nesterov=False)



Last: val\_rmse: 1.7194

Best: val\_rmse: 1.3030

# 17. Model LSTM(128, activation="tanh"), Dense(300), optimizer=SGD(learning\_rate=0.001, momentum=0.05, nesterov=True)



Best: val\_rmse: 1.3003