# Train your model and save its history

h\_callback = model.fit(X\_train, y\_train, epochs = 50,

               validation\_data=(X\_test, y\_test))

# Plot train vs test loss during training

plot\_loss(h\_callback.history['loss'], h\_callback.history['val\_loss'])

# Plot train vs test accuracy during training

plot\_accuracy(h\_callback.history['acc'], h\_callback.history['val\_acc'])

# Import the early stopping callback

from keras.callbacks import EarlyStopping

# Define a callback to monitor val\_acc

monitor\_val\_acc = EarlyStopping(monitor='val\_acc',

                       patience=5)

# Train your model using the early stopping callback

model.fit(X\_train, y\_train,

           epochs=1000, validation\_data=(X\_test,y\_test),

           callbacks= [monitor\_val\_acc])

# Import the EarlyStopping and ModelCheckpoint callbacks

from keras.callbacks import EarlyStopping, ModelCheckpoint

# Early stop on validation accuracy

monitor\_val\_acc = EarlyStopping(monitor = 'val\_acc', patience = 3)

# Save the best model as best\_banknote\_model.hdf5

modelCheckpoint = ModelCheckpoint('best\_banknote\_model.hdf5', save\_best\_only = True)

# Fit your model for a stupid amount of epochs

h\_callback = model.fit(X\_train, y\_train,

                    epochs = 1000000000000,

                    callbacks = [monitor\_val\_acc, modelCheckpoint],

                    validation\_data = (X\_test, y\_test))

# Train your model for 60 epochs, using X\_test and y\_test as validation data

h\_callback = model.fit(X\_train, y\_train, epochs = 60, validation\_data = (X\_test, y\_test), verbose=0)

# Extract from the h\_callback object loss and val\_loss to plot the learning curve

plot\_loss(h\_callback.history['loss'], h\_callback.history['val\_loss'])

for size in training\_sizes:

    # Get a fraction of training data (we only care about the training data)

    X\_train\_frac, y\_train\_frac = X\_train[:100], y\_train[:100]

    # Reset the model to the initial weights and train it on the new training data fraction

    model.set\_weights(initial\_weights)

    model.fit(X\_train\_frac, y\_train\_frac, epochs = 50, callbacks = [early\_stop])

    # Evaluate and store both: the training data fraction and the complete test set results

    train\_accs.append(model.evaluate(X\_train, y\_train)[1])

    test\_accs.append(model.evaluate(X\_train\_frac, y\_train\_frac)[1])

# Plot train vs test accuracies

plot\_results(train\_accs, test\_accs)