

Week 7 – HomeWork 4

Paritosh Rai



7335 Machine Learning 2



HW 4

Below you see a tutorial from Keras on using transfer learning.  They train their models on have the digits and predict the second half.   Your homework is to train on all digits and make your own handwritten data set of 5 characters (ie A, B, C, D, E)  and transfer your minist trained model over to them.  Enjoy!

Code From <https://keras.io/examples/mnist_transfer_cnn/>

|  |
| --- |
| from \_\_future\_\_ import print\_function import datetime import keras from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense, Dropout, Activation, Flatten,  Conv2D, MaxPooling2D from keras import backend as K now = datetime.datetime.now batch\_size = 128 num\_classes = 5 epochs = 5 img\_rows, img\_cols = 28, 28 filters = 32 pool\_size = 2 kernel\_size = 3 if K.image\_data\_format() == 'channels\_first':     input\_shape = (1, img\_rows, img\_cols) else:     input\_shape = (img\_rows, img\_cols, 1)  def train\_model(model, train, test, num\_classes):     x\_train = train[0].reshape((train[0].shape[0],) + input\_shape)     x\_test = test[0].reshape((test[0].shape[0],) + input\_shape)     x\_train = x\_train.astype('float32')     x\_test = x\_test.astype('float32')     x\_train /= 255     x\_test /= 255     print('x\_train shape:', x\_train.shape)     print(x\_train.shape[0], 'train samples')     print(x\_test.shape[0], 'test samples')     # convert class vectors to binary class matrices     y\_train = keras.utils.to\_categorical(train[1], num\_classes)     y\_test = keras.utils.to\_categorical(test[1], num\_classes)     model.compile(loss='categorical\_crossentropy',                   optimizer='adadelta',                   metrics=['accuracy'])     t = now()     model.fit(x\_train, y\_train,               batch\_size=batch\_size,               epochs=epochs,               verbose=1,               validation\_data=(x\_test, y\_test))     print('Training time: %s' % (now() - t))     score = model.evaluate(x\_test, y\_test, verbose=0)     print('Test score:', score[0])     print('Test accuracy:', score[1])  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data() x\_train\_lt5 = x\_train[y\_train < 5] y\_train\_lt5 = y\_train[y\_train < 5] x\_test\_lt5 = x\_test[y\_test < 5] y\_test\_lt5 = y\_test[y\_test < 5] x\_train\_gte5 = x\_train[y\_train >= 5] y\_train\_gte5 = y\_train[y\_train >= 5] - 5 x\_test\_gte5 = x\_test[y\_test >= 5] y\_test\_gte5 = y\_test[y\_test >= 5] - 5  feature\_layers = [     Conv2D(filters, kernel\_size,            padding='valid',            input\_shape=input\_shape),     Activation('relu'),     Conv2D(filters, kernel\_size),     Activation('relu'),     MaxPooling2D(pool\_size=pool\_size),     Dropout(0.25),     Flatten(), ]  classification\_layers = [     Dense(128),     Activation('relu'),     Dropout(0.5),     Dense(num\_classes),     Activation('softmax') ]  # create complete model model = Sequential(feature\_layers + classification\_layers)  # train model for 5-digit classification [0..4] train\_model(model,             (x\_train\_lt5, y\_train\_lt5),             (x\_test\_lt5, y\_test\_lt5), num\_classes)  # freeze feature layers and rebuild model for l in feature\_layers:     l.trainable = False  # transfer: train dense layers for new classification task [5..9] train\_model(model,             (x\_train\_gte5, y\_train\_gte5),             (x\_test\_gte5, y\_test\_gte5), num\_classes) |

Entered pulled the data from 0 to 9 and run the data, used provided program:

Converted the pictures to 28 X 28 pick cell matrix.

Epoch 1/1000

1875/1875 [==============================] - 15s 8ms/step - loss: 0.4268 - accuracy: 0.9161 - val\_loss: 0.0691 - val\_accuracy: 0.9794

Epoch 2/1000

1875/1875 [==============================] - 14s 8ms/step - loss: 0.1075 - accuracy: 0.9678 - val\_loss: 0.0496 - val\_accuracy: 0.9834

Epoch 3/1000

1875/1875 [==============================] - 14s 8ms/step - loss: 0.0844 - accuracy: 0.9751 - val\_loss: 0.0436 - val\_accuracy: 0.9859

Epoch 4/1000

1875/1875 [==============================] - 15s 8ms/step - loss: 0.0736 - accuracy: 0.9782 - val\_loss: 0.0426 - val\_accuracy: 0.9872

Epoch 5/1000

1875/1875 [==============================] - 17s 9ms/step - loss: 0.0648 - accuracy: 0.9805 - val\_loss: 0.0365 - val\_accuracy: 0.9882

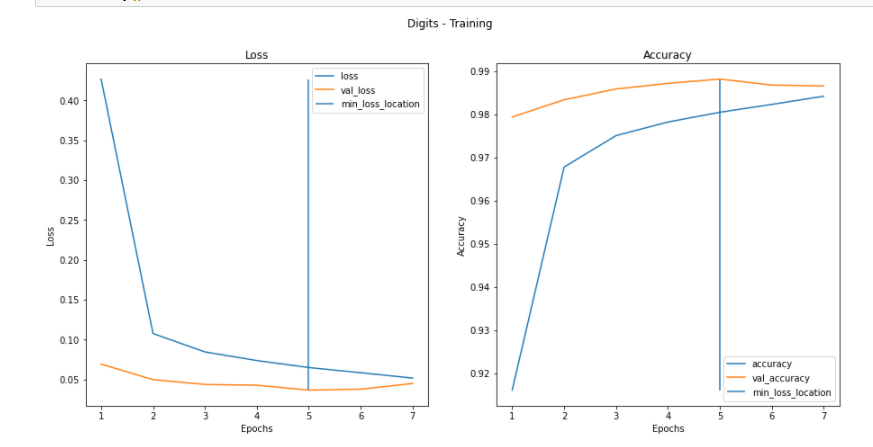
Epoch 6/1000

1875/1875 [==============================] - 17s 9ms/step - loss: 0.0583 - accuracy: 0.9823 - val\_loss: 0.0376 - val\_accuracy: 0.9868

Epoch 7/1000

1875/1875 [==============================] - 17s 9ms/step - loss: 0.0516 - accuracy: 0.9842 - val\_loss: 0.0448 - val\_accuracy: 0.9866

Following is the plot:



The digital Neural Network Model (NN) had 3 layers. Using activation function “Relu” and “penult” layer will have activation function of “softmax”.

A, B, C, D and E data was created. Created 120 samples each, total 120 \*5 = 520 samples. Leverage the transfer learning from 0 to 9 digits. Adjusted the penult\_layer and use optimizer as “adam” and activation function as “softmax to take care of change in input.

Took the A, B, C, D and E function into numeric value 10,11,12, 13,14 and 15.

New model is created using model input from digital letters and newly created outer layer using penult\_layer

new\_model = Model(model.input, output\_layer)

Following is the loss and accuracy of model and validation:

Accuracy started with 20% and got up to lower 60%.

Epoch 1/1000

15/15 [==============================] - 0s 14ms/step - loss: 2.6535 - accuracy: 0.1315 - val\_loss: 2.0079 - val\_accuracy: 0.2000

Epoch 2/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.8413 - accuracy: 0.2693 - val\_loss: 1.5960 - val\_accuracy: 0.3583

Epoch 3/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.5055 - accuracy: 0.4259 - val\_loss: 1.3828 - val\_accuracy: 0.5750

Epoch 4/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.3354 - accuracy: 0.4697 - val\_loss: 1.2722 - val\_accuracy: 0.5833

Epoch 5/1000

15/15 [==============================] - ETA: 0s - loss: 1.2644 - accuracy: 0.49 - 0s 5ms/step - loss: 1.2644 - accuracy: 0.4927 - val\_loss: 1.2129 - val\_accuracy: 0.6083

Epoch 6/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.1974 - accuracy: 0.5303 - val\_loss: 1.1670 - val\_accuracy: 0.5917

Epoch 7/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.1578 - accuracy: 0.5491 - val\_loss: 1.1124 - val\_accuracy: 0.6250

Epoch 8/1000

15/15 [==============================] - ETA: 0s - loss: 1.0337 - accuracy: 0.56 - 0s 5ms/step - loss: 1.0990 - accuracy: 0.5637 - val\_loss: 1.0962 - val\_accuracy: 0.5917

Epoch 9/1000

15/15 [==============================] - 0s 4ms/step - loss: 1.0748 - accuracy: 0.5950 - val\_loss: 1.0581 - val\_accuracy: 0.6500

Epoch 10/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.0413 - accuracy: 0.5783 - val\_loss: 1.0365 - val\_accuracy: 0.6250

Epoch 11/1000

15/15 [==============================] - 0s 5ms/step - loss: 1.0108 - accuracy: 0.6138 - val\_loss: 1.0216 - val\_accuracy: 0.6250

Epoch 12/1000

15/15 [==============================] - 0s 6ms/step - loss: 1.0001 - accuracy: 0.6075 - val\_loss: 1.0046 - val\_accuracy: 0.6333

Epoch 13/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.9693 - accuracy: 0.6263 - val\_loss: 0.9886 - val\_accuracy: 0.6250

Epoch 14/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.9777 - accuracy: 0.6200 - val\_loss: 0.9883 - val\_accuracy: 0.6583

Epoch 15/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.9670 - accuracy: 0.6138 - val\_loss: 0.9734 - val\_accuracy: 0.6167

Epoch 16/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.9342 - accuracy: 0.6347 - val\_loss: 0.9326 - val\_accuracy: 0.6833

Epoch 17/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.8870 - accuracy: 0.6868 - val\_loss: 0.9433 - val\_accuracy: 0.6667

Epoch 18/1000

15/15 [==============================] - ETA: 0s - loss: 0.9032 - accuracy: 0.65 - 0s 5ms/step - loss: 0.9032 - accuracy: 0.6576 - val\_loss: 0.9123 - val\_accuracy: 0.6667

Epoch 19/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.8671 - accuracy: 0.6827 - val\_loss: 0.9202 - val\_accuracy: 0.6667

Epoch 20/1000

15/15 [==============================] - 0s 6ms/step - loss: 0.8477 - accuracy: 0.6785 - val\_loss: 0.9146 - val\_accuracy: 0.6750

Following is the graph showing reduction in loss and improvement of accuracy:

