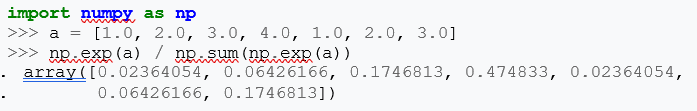
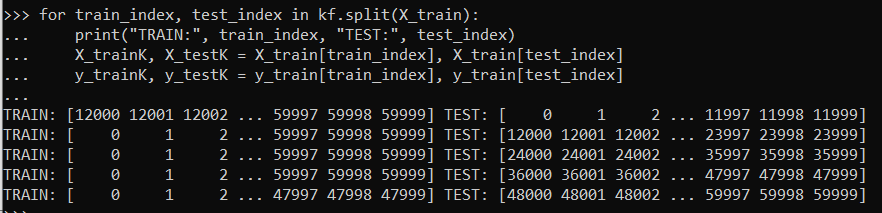
1. What is a Dense layer? Why do we use them?
   1. The Dense layer is the all:all connection between the inputs and the next layer. It is used to change the dimensions of a vector containing trainable parameters
2. What is a Dropout Layer? Why do we use them?
   1. Dropout layers randomly thin/drop a percent of neurons in the network’s input layers during training, thereby reducing over-fitting.
3. What is a convolutional layer? Why do we use them?
   1. Convolutional layers, in image processing, are used for feature extraction. Feature extraction can then be a layer of the model that leads to the decision of choosing a final output.
4. What is a pooling layer? Why do we use them?
   1. Pooling layers allow consolidation of the data. For example, if we were to compress 2x2 pixel squares into one value by averaging them (or taking the maximum intensity), we would then have 75% less data by volume to manipulate.
5. What is the logit layer? Where is it located?
   1. The logit function, which is the inverse of the sigmoid function, is used to convert weighted values to probabilities, and is used on the output layer.
   2. Logits operate on the unscaled output of previous layers after the inputs have been constructed and is used after an all:all layer
6. What is the softmax layer? What is it used for?
   1. The softmax layer is typically the final layer and performs classification by using the normalized exponential function. This layer is used to map the output of a network to a probability distribution of the corresponding possible classes (which will sum to 1). It measures probability error in discrete classification tasks (where the classes do not overlap).
   2. A softmax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the 10 to be selected as the model’s output prediction. Logarithmic loss is used as the loss function (called categorical\_crossentropy in Keras) and the efficient ADAM gradient descent algorithm is used to learn the weights.
   3. From Wikipedia: <https://en.wikipedia.org/wiki/Softmax_function#:~:text=Softmax%20is%20often%20used%20in,distribution%20over%20predicted%20output%20classes.>

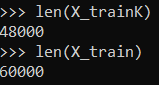


1. What is the loss function? What is it used for?
   1. The loss function is the sum of the squares of the difference between the correct value (typically a 0 or 1 for classification) and the model’s predicted probabilities (from the softmax layer) for all test cases. The loss function is used to summarize the ‘correct-ness’ of a model and minimizing the loss function is used to define the gradient descent and perform back-propagation.
2. What happens in a training epoch?
   1. The model’s weights are updated using training data, each epoch is one pass through the data, with updates after each batch

Modify the MNIST tutorial from above using scikit-learn’s **KFold class** (<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html>) to add a 5-fold validation to your model, train it, then test **with original test data**. What is the end accuracy of the model on your test data**?** Is it better than not using 5 fold validation**?**

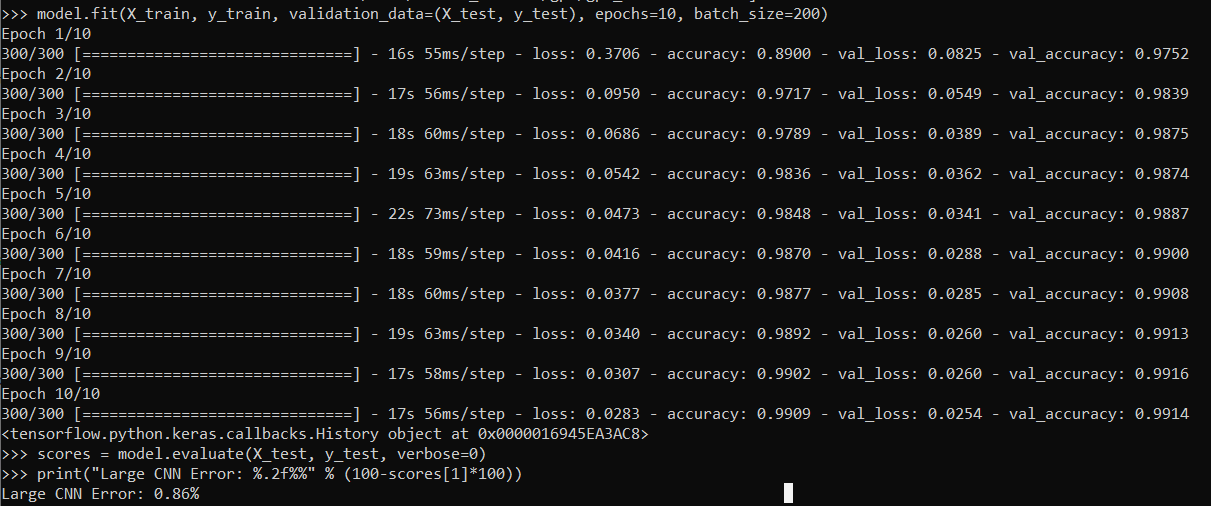
What are your model’s prediction of the first 5 samples from the test data**?**





**model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=200)**

**This one did not use k-fold:**



>>> print("Large CNN Error: %.2f%%" % (100-scores[1]\*100))

Large CNN Error: 0.86%

**Now we will use k-fold, with 5 epochs per training set:**

scoresvec = []

for train\_index, test\_index in kf.split(X\_train): #might need to tack on targets here

print("TRAIN:", train\_index, "TEST:", test\_index)

X\_trainK, X\_testK = X\_train[train\_index], X\_train[test\_index]

y\_trainK, y\_testK = y\_train[train\_index], y\_train[test\_index]

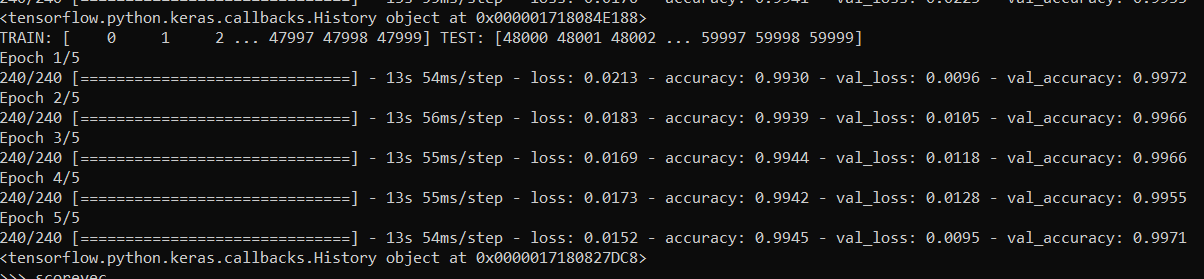
model.fit(X\_trainK, y\_trainK, validation\_data=(X\_testK, y\_testK), epochs=5, batch\_size=200)

scores = model.evaluate(X\_test, y\_test, verbose=0)

scoresvec.append((100-scores[1]\*100))

>>> scoresvec

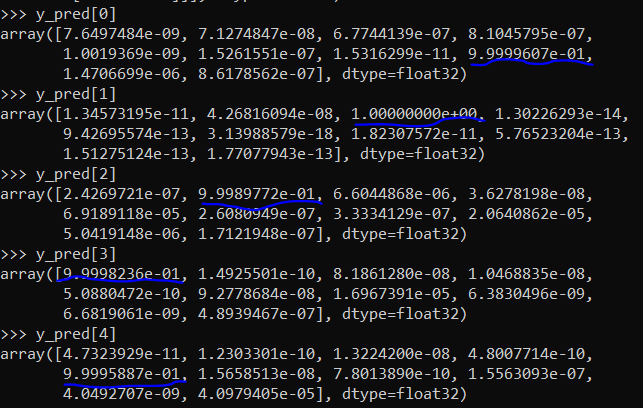
[1.23, 1.06, 0.84, 0.93, 0.70]



The end accuracy is: 99.3% (0.7), which is better than the previously observed 0.86 and 0.93 from the non-kfold examples

Predicition of first 5 pictures from test data:

y\_pred = model.predict(X\_test)



These correspond to predictions of 7, 2, 1, 0, and 4.