Neural Networks & Deep Learning - ICP-6

CS 5720 (CRN 23216)

Student ID: 700745451

Student Name: Kamala Ramesh

1. To implement the use case provided in class – a. To add more Dense layers to the given code and see how accuracy changes.

```
[39] from google.colab import drive
     drive.mount('/content/gdrive')

     Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).

[40] #read the diabetes csv file
     path_to_csv = '/content/gdrive/My Drive/diabetes.csv'
```

```
import keras
import pandas
from keras.models import Sequential
from keras.layers.core import Dense, Activation

# load dataset
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

dataset = pd.read_csv(path_to_csv, header=None).values

X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, Y_test))
```

```
Epoch 1/100
18/18 [==============================] - 1s 3ms/step - loss: 6.9816 - acc: 0.6441
Epoch 2/100
18/18 [==============================] - 0s 3ms/step - loss: 4.5781 - acc: 0.6458
Epoch 3/100
18/18 [==============================] - 0s 4ms/step - loss: 3.6688 - acc: 0.6372
Epoch 4/100
18/18 [==============================] - 0s 4ms/step - loss: 3.1913 - acc: 0.6163
```

+ Code + Text

```
[50]
Epoch 97/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5456 - acc: 0.7413
Epoch 98/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5450 - acc: 0.7378
Epoch 99/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5612 - acc: 0.7170
Epoch 100/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5351 - acc: 0.7483
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_34 (Dense)            (None, 20)                180

 dense_35 (Dense)            (None, 1)                 21

=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.6220 - acc: 0.6667
[0.6220337748527527, 0.6666666865348816]
```

After adding dense layers to the existing code

```
my_second_nn = Sequential() # create model
my_second_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
my_second_nn.add(Dense(10, input_dim=8, activation='relu'))
my_second_nn.add(Dense(5, input_dim=8, activation='relu'))
my_second_nn.add(Dense(1, activation='sigmoid')) # output layer
my_second_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_second_nn_fitted = my_second_nn.fit(X_train, Y_train, epochs=100,
                                       initial_epoch=0)
print(my_second_nn.summary())
print(my_second_nn.evaluate(X_test, Y_test))
```

```
Epoch 82/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5229 - acc: 0.7448
Epoch 83/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5219 - acc: 0.7431
Epoch 84/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5210 - acc: 0.7448
Epoch 85/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5229 - acc: 0.7465
Epoch 86/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5272 - acc: 0.7396
Epoch 87/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5194 - acc: 0.7569
Epoch 88/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5172 - acc: 0.7361
Epoch 89/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5184 - acc: 0.7448
Epoch 90/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5157 - acc: 0.7396
Epoch 91/100
```

```
Epoch 99/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5154 - acc: 0.7448
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5097 - acc: 0.7500
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_36 (Dense)            (None, 20)                180

 dense_37 (Dense)            (None, 10)                210

 dense_38 (Dense)            (None, 5)                 55

 dense_39 (Dense)            (None, 1)                 6

=================================================================
Total params: 451
Trainable params: 451
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.5674 - acc: 0.7031
[0.5673847198486328, 0.703125]
```

For this execution, the loss value decreased, and accuracy increased.

2. Perform the same code as above for the breast cancer dataset and changes the input values as required.

+ Code + Text

```
[1] from google.colab import drive
    drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
[25] path_to_csv_2 = '/content/gdrive/My Drive/breastcancer.csv'
```

```python
import keras
import pandas
from keras.models import Sequential
from keras.layers.core import Dense, Activation


# load dataset
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

dataset = pd.read_csv(path_to_csv_2)

#label the output data from string to numerical value
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
dataset["diagnosis_code"] = lb_make.fit_transform(dataset["diagnosis"])
dataset["diagnosis_code"].value_counts()

#Column 'id' is not included in input as it is unique for each record
X = dataset[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]

y = dataset['diagnosis_code']
```

```python
#Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle=True, random_state=5)

#training the model
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(30, input_dim=30, activation='relu')) # hidden layer
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, y_test))
```

```
Epoch 1/100
15/15 [==============================] - 1s 2ms/step - loss: 78.2931 - acc: 0.6264
Epoch 2/100
15/15 [==============================] - 0s 2ms/step - loss: 17.3186 - acc: 0.6615
Epoch 3/100
15/15 [==============================] - 0s 2ms/step - loss: 4.2325 - acc: 0.5253
Epoch 4/100
15/15 [==============================] - 0s 2ms/step - loss: 0.8222 - acc: 0.8791
Epoch 5/100
15/15 [==============================] - 0s 2ms/step - loss: 0.5686 - acc: 0.8593
Epoch 6/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4340 - acc: 0.9011
Epoch 7/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4172 - acc: 0.8747
Epoch 8/100
15/15 [==============================] - 0s 2ms/step - loss: 0.3826 - acc: 0.9143
```

```
Epoch 96/100
15/15 [==============================] - 0s 2ms/step - loss: 0.2280 - acc: 0.9209
Epoch 97/100
15/15 [==============================] - 0s 2ms/step - loss: 0.1556 - acc: 0.9407
Epoch 98/100
15/15 [==============================] - 0s 2ms/step - loss: 0.1606 - acc: 0.9363
Epoch 99/100
15/15 [==============================] - 0s 2ms/step - loss: 0.1827 - acc: 0.9297
Epoch 100/100
15/15 [==============================] - 0s 2ms/step - loss: 0.1717 - acc: 0.9341
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 30)                930

 dense_21 (Dense)            (None, 1)                 31

=================================================================
Total params: 961
Trainable params: 961
Non-trainable params: 0
_________________________________________________________________
None
4/4 [==============================] - 0s 4ms/step - loss: 0.1465 - acc: 0.9298
[0.1465480476617813, 0.9298245906829834]
```

3. Use the StandardScaler to normalize the data and perform the same.

```
[28] #normalize the data using StandardScaler
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     sc.fit(X)
     X_scaled = sc.transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, stratify=y, test_size=0.2,shuffle=True, random_state=5)

#train the model using the normalized input set
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(30, input_dim=30, activation='relu')) # hidden layer
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, y_test))
```

```
Epoch 1/100
15/15 [==============================] - 1s 2ms/step - loss: 0.5049 - acc: 0.8198
Epoch 2/100
15/15 [==============================] - 0s 2ms/step - loss: 0.3493 - acc: 0.9055
Epoch 3/100
15/15 [==============================] - 0s 2ms/step - loss: 0.2645 - acc: 0.9319
Epoch 4/100
15/15 [==============================] - 0s 2ms/step - loss: 0.2136 - acc: 0.9385
Epoch 5/100
```

```
Epoch 94/100
15/15 [==============================] - 0s 2ms/step - loss: 0.0259 - acc: 0.9912
Epoch 95/100
15/15 [==============================] - 0s 3ms/step - loss: 0.0257 - acc: 0.9912
Epoch 96/100
15/15 [==============================] - 0s 2ms/step - loss: 0.0253 - acc: 0.9912
Epoch 97/100
15/15 [==============================] - 0s 2ms/step - loss: 0.0250 - acc: 0.9912
Epoch 98/100
15/15 [==============================] - 0s 2ms/step - loss: 0.0247 - acc: 0.9912
Epoch 99/100
15/15 [==============================] - 0s 2ms/step - loss: 0.0243 - acc: 0.9934
Epoch 100/100
15/15 [==============================] - 0s 3ms/step - loss: 0.0242 - acc: 0.9934
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 30)                930

 dense_23 (Dense)            (None, 1)                 31

=================================================================
Total params: 961
Trainable params: 961
Non-trainable params: 0
_________________________________________________________________
None
4/4 [==============================] - 0s 4ms/step - loss: 0.0676 - acc: 0.9649
[0.06761771440505981, 0.9649122953414917]
```

When we feed the normalized data to the network the accuracy increases, and the loss decreases.

Image Classification Problem

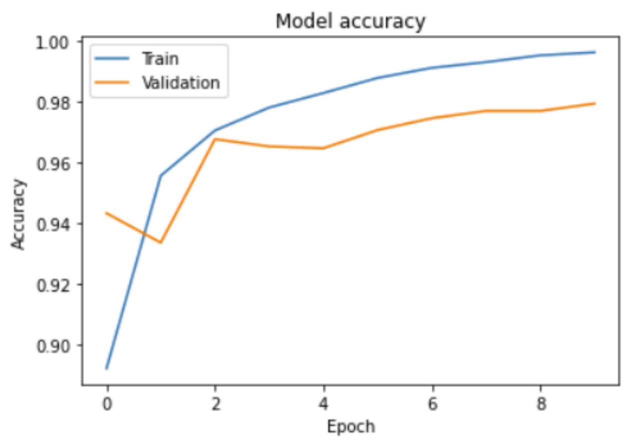
```
[57] from keras import Sequential
     from keras.datasets import mnist
     import numpy as np
     from keras.layers import Dense
     from keras.utils import to_categorical

     (train_images,train_labels),(test_images, test_labels) = mnist.load_data()

     print(train_images.shape[1:])
     #process the data
     #1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
     dimData = np.prod(train_images.shape[1:])
     print(dimData)
     train_data = train_images.reshape(train_images.shape[0],dimData)
     test_data = test_images.reshape(test_images.shape[0],dimData)

     #convert data to float and scale values between 0 and 1
     train_data = train_data.astype('float')
     test_data = test_data.astype('float')
     #scale data
     train_data /=255.0
     test_data /=255.0
     #change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
     train_labels_one_hot = to_categorical(train_labels)
     test_labels_one_hot = to_categorical(test_labels)

     #creating network
     model = Sequential()
     model.add(Dense(512, activation='relu', input_shape=(dimData,)))
     model.add(Dense(512, activation='relu'))
     model.add(Dense(10, activation='softmax'))
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

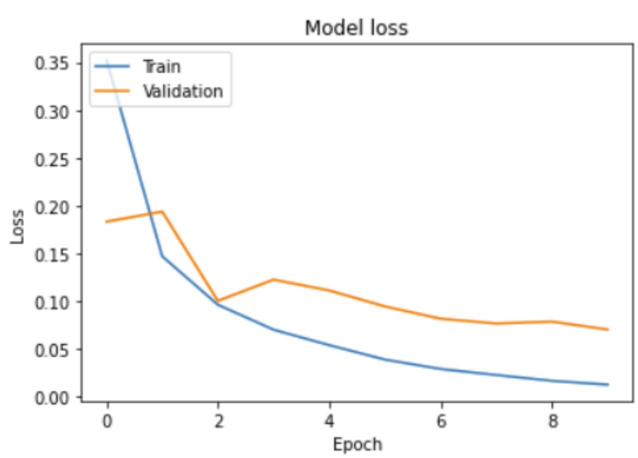
```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 9s 36ms/step - loss: 0.2906 - accuracy: 0.9115 - val_loss: 0.1891 - val_accuracy: 0.9377
Epoch 2/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1021 - accuracy: 0.9683 - val_loss: 0.1172 - val_accuracy: 0.9609
Epoch 3/10
235/235 [==============================] - 8s 34ms/step - loss: 0.0631 - accuracy: 0.9804 - val_loss: 0.0773 - val_accuracy: 0.9752
Epoch 4/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0448 - accuracy: 0.9858 - val_loss: 0.0811 - val_accuracy: 0.9751
Epoch 5/10
235/235 [==============================] - 7s 32ms/step - loss: 0.0318 - accuracy: 0.9894 - val_loss: 0.0750 - val_accuracy: 0.9789
Epoch 6/10
235/235 [==============================] - 8s 35ms/step - loss: 0.0247 - accuracy: 0.9922 - val_loss: 0.0697 - val_accuracy: 0.9809
Epoch 7/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0168 - accuracy: 0.9947 - val_loss: 0.0710 - val_accuracy: 0.9798
Epoch 8/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0135 - accuracy: 0.9959 - val_loss: 0.0661 - val_accuracy: 0.9821
Epoch 9/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0094 - accuracy: 0.9975 - val_loss: 0.0711 - val_accuracy: 0.9823
Epoch 10/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0075 - accuracy: 0.9976 - val_loss: 0.0944 - val_accuracy: 0.9792
```

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code

```
import matplotlib.pyplot as plt
# Plot the training and validation accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

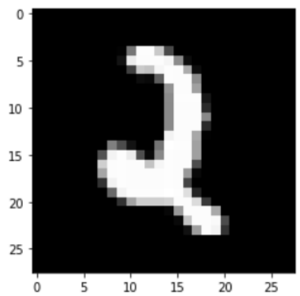



```
[63] # Plot the training and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```
# Select a random image from the test data
idx = np.random.randint(0, test_images.shape[0])
img = test_images[idx]

# Plot the selected image
plt.imshow(img, cmap='gray')
plt.show()

input_image = img.reshape(1, 784).astype('float32') / 255.0

prediction = model.predict(input_image)
print('The image is predicted as:', np.argmax(prediction))
```




```
1/1 [==============================] - 0s 26ms/step
The image is predicted as: 2
```

3. To change the number of hidden layer and the activation to tanh or sigmoid and observe the changes

```
#creating network
model_upd = Sequential()
model_upd.add(Dense(512, activation='tanh', input_shape=(dimData,)))
model_upd.add(Dense(512, activation='tanh'))
model_upd.add(Dense(256, activation='tanh'))
model_upd.add(Dense(128, activation='tanh'))
model_upd.add(Dense(10, activation='softmax'))

model_upd.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_upd.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                   validation_data=(test_data, test_labels_one_hot))
```

```
Epoch 1/10
235/235 [==============================] - 10s 41ms/step - loss: 0.3548 - accuracy: 0.8903 - val_loss: 0.1926 - val_accuracy: 0.9444
Epoch 2/10
235/235 [==============================] - 8s 36ms/step - loss: 0.1455 - accuracy: 0.9558 - val_loss: 0.1749 - val_accuracy: 0.9456
Epoch 3/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0980 - accuracy: 0.9694 - val_loss: 0.1277 - val_accuracy: 0.9602
Epoch 4/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0705 - accuracy: 0.9779 - val_loss: 0.2338 - val_accuracy: 0.9230
Epoch 5/10
235/235 [==============================] - 9s 36ms/step - loss: 0.0543 - accuracy: 0.9831 - val_loss: 0.0902 - val_accuracy: 0.9726
Epoch 6/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0396 - accuracy: 0.9876 - val_loss: 0.1559 - val_accuracy: 0.9531
Epoch 7/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0304 - accuracy: 0.9903 - val_loss: 0.1162 - val_accuracy: 0.9650
Epoch 8/10
235/235 [==============================] - 9s 40ms/step - loss: 0.0236 - accuracy: 0.9929 - val_loss: 0.0875 - val_accuracy: 0.9727
Epoch 9/10
235/235 [==============================] - 8s 36ms/step - loss: 0.0184 - accuracy: 0.9943 - val_loss: 0.0765 - val_accuracy: 0.9765
Epoch 10/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0135 - accuracy: 0.9959 - val_loss: 0.0759 - val_accuracy: 0.9785
```

There is a very minute changes when we add more dense layers and change the activation to function to tanh. Here for the validation data, the loss decreased and the accuracy also decreased.

4. Executing the same code withing scaling the input data

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')

#Commenting the scale data part
#train_data /=255.0
#test_data /=255.0

#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
#Feeding the unscaled data to the network
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 13s 50ms/step - loss: 7.1164 - accuracy: 0.8726 - val_loss: 0.7212 - val_accuracy: 0.9056
Epoch 2/10
235/235 [==============================] - 5s 20ms/step - loss: 0.4010 - accuracy: 0.9453 - val_loss: 0.6054 - val_accuracy: 0.9258
Epoch 3/10
235/235 [==============================] - 5s 21ms/step - loss: 0.2324 - accuracy: 0.9601 - val_loss: 0.4047 - val_accuracy: 0.9423
Epoch 4/10
235/235 [==============================] - 5s 21ms/step - loss: 0.1761 - accuracy: 0.9679 - val_loss: 0.2555 - val_accuracy: 0.9577
Epoch 5/10
235/235 [==============================] - 5s 20ms/step - loss: 0.1583 - accuracy: 0.9727 - val_loss: 0.2695 - val_accuracy: 0.9648
Epoch 6/10
235/235 [==============================] - 5s 22ms/step - loss: 0.1351 - accuracy: 0.9761 - val_loss: 0.2610 - val_accuracy: 0.9659
Epoch 7/10
235/235 [==============================] - 4s 19ms/step - loss: 0.1237 - accuracy: 0.9801 - val_loss: 0.2888 - val_accuracy: 0.9681
Epoch 8/10
235/235 [==============================] - 5s 21ms/step - loss: 0.1173 - accuracy: 0.9807 - val_loss: 0.2874 - val_accuracy: 0.9691
Epoch 9/10
235/235 [==============================] - 6s 26ms/step - loss: 0.1077 - accuracy: 0.9829 - val_loss: 0.2675 - val_accuracy: 0.9734
Epoch 10/10
235/235 [==============================] - 6s 24ms/step - loss: 0.1053 - accuracy: 0.9854 - val_loss: 0.3169 - val_accuracy: 0.9707
```

When we feed the network without scaling the data, the loss increases and the accuracy decreases.