

CS 5720 Neural Network Deep Learning

ICP-6

CRN	23441
NAME	MANOJ BALA
EMAIL	mxb40210@ucmo.edu
STUDENT_ID	700754021

GitHub Repository:

<https://github.com/mxb40210/700754021-NeuralNetworkDeepLearning>

Assignment 6:

<https://github.com/mxb40210/700754021-NeuralNetworkDeepLearning/tree/main/assignments/assignment6>

1. Question 1

```
solutions.ipynb
File Edit View Insert Runtime Tools Help

+ Code + Text

breastcancer_result = breastcancer.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Evaluate
evaluation = breastcancer.evaluate(X_test, Y_test)


breastcancer.summary()
print('\nLoss: {}, Accuracy: {}'.format(evaluation[0], evaluation[1]))

Epoch 85/100
14/14 [=====] - 0s 3ms/step - loss: 0.6642 - accuracy: 0.6197
Epoch 86/100
14/14 [=====] - 0s 3ms/step - loss: 0.6642 - accuracy: 0.6197
Epoch 87/100
14/14 [=====] - 0s 3ms/step - loss: 0.6644 - accuracy: 0.6197
Epoch 88/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 89/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 90/100
14/14 [=====] - 0s 3ms/step - loss: 0.6645 - accuracy: 0.6197
Epoch 91/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 92/100
14/14 [=====] - 0s 3ms/step - loss: 0.6642 - accuracy: 0.6197
Epoch 93/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 94/100
14/14 [=====] - 0s 3ms/step - loss: 0.6645 - accuracy: 0.6197
Epoch 95/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 96/100
14/14 [=====] - 0s 3ms/step - loss: 0.6642 - accuracy: 0.6197
Epoch 97/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 98/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 99/100
14/14 [=====] - 0s 3ms/step - loss: 0.6645 - accuracy: 0.6197
Epoch 100/100
14/14 [=====] - 0s 3ms/step - loss: 0.6641 - accuracy: 0.6197
5/5 [=====] - 0s 7ms/step - loss: 0.6494 - accuracy: 0.6503
Model: "sequential_28"

Layer (type)                 Output Shape              Param #
=====
dense_134 (Dense)            (None, 64)                2048
dropout_18 (Dropout)         (None, 64)                0
dense_135 (Dense)            (None, 32)               2080
dense_136 (Dense)            (None, 16)                528
dense_137 (Dense)            (None, 8)                 136
dense_138 (Dense)            (None, 4)                  36
dense_139 (Dense)            (None, 1)                   5
=====
Total params: 4833 (18.88 KB)
Trainable params: 4833 (18.88 KB)
Non-trainable params: 0 (0.00 Byte)

Loss: 0.649402885437012, Accuracy: 0.6503496766090393
```

Epoch 85/100

 solutions.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+ Code + Text

65

```
"""
1.2 Change the data source to Breast Cancer dataset * available in the source code folder and make required changes.
Report accuracy of the model.
"""

# Read breastcancer dataset
breastcancer_dataset = pd.read_csv(breastcancer_csv_path)
breastcancer_dataset = breastcancer_dataset.dropna(axis=1)

# Map target labels to 'M' and 'B'
label_encoder = LabelEncoder()
breastcancer_dataset['diagnosis'] = label_encoder.fit_transform(breastcancer_dataset['diagnosis'])

# Split the dataset X and Y
X = breastcancer_dataset.drop('diagnosis', axis=1)
Y = breastcancer_dataset['diagnosis']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

# Set some seed
np.random.seed(155)

# Create model
breastcancer = Sequential()

# Add layers
breastcancer.add(Dense(64, input_dim=X.shape[1], activation='relu')) # Input layer
breastcancer.add(Dropout(0.2)) # Dropout layer to prevent overfitting
breastcancer.add(Dense(32, activation='relu')) # Hidden layer
breastcancer.add(Dense(16, activation='relu')) # Hidden layer
breastcancer.add(Dense(8, activation='relu')) # Hidden layer
breastcancer.add(Dense(4, activation='relu')) # Hidden layer
breastcancer.add(Dense(1, activation='sigmoid')) # Output layer

# Optimizer
sgd = SGD(learning_rate=0.01, momentum=0.9)

# Compile
breastcancer.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Train
breastcancer_result = breastcancer.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Evaluate
evaluation = breastcancer.evaluate(X_test, Y_test)

breastcancer.summary()
print('\nLoss: {}, Accuracy: {}'.format(evaluation[0], evaluation[1]))
```




+ Code + Text



6s

```
Epoch 92/100
14/14 [=====] - 0s 3ms/step - loss: 0.6642 - accuracy: 0.6197
Epoch 93/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 94/100
14/14 [=====] - 0s 3ms/step - loss: 0.6645 - accuracy: 0.6197
Epoch 95/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 96/100
14/14 [=====] - 0s 3ms/step - loss: 0.6642 - accuracy: 0.6197
Epoch 97/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 98/100
14/14 [=====] - 0s 3ms/step - loss: 0.6643 - accuracy: 0.6197
Epoch 99/100
14/14 [=====] - 0s 3ms/step - loss: 0.6645 - accuracy: 0.6197
Epoch 100/100
14/14 [=====] - 0s 3ms/step - loss: 0.6641 - accuracy: 0.6197
5/5 [=====] - 0s 7ms/step - loss: 0.6494 - accuracy: 0.6503
Model: "sequential_28"
```

Layer (type)	Output Shape	Param #
dense_134 (Dense)	(None, 64)	2048
dropout_18 (Dropout)	(None, 64)	0
dense_135 (Dense)	(None, 32)	2080
dense_136 (Dense)	(None, 16)	528
dense_137 (Dense)	(None, 8)	136
dense_138 (Dense)	(None, 4)	36
dense_139 (Dense)	(None, 1)	5

```
=====
Total params: 4833 (18.88 KB)
Trainable params: 4833 (18.88 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Loss: 0.6494402885437012, Accuracy: 0.6503496766090393
```

solutions.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

65

```
1.3 Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

from sklearn.preprocessing
import StandardScaler
sc = StandardScaler()

# Read breastcancer dataset
breastcancer_dataset = pd.read_csv(breastcancer_csv_path)
breastcancer_dataset = breastcancer_dataset.dropna(axis=1)

# Map target labels to 'M' and 'B'
label_encoder = LabelEncoder()
breastcancer_dataset['diagnosis'] = label_encoder.fit_transform(breastcancer_dataset['diagnosis'])

# Normalize the input features
sc = StandardScaler()
X_normalized = sc.fit_transform(breastcancer_dataset.drop('diagnosis', axis=1))

# Split the dataset X and Y
X = X_normalized
Y = breastcancer_dataset['diagnosis']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

# Set some seed
np.random.seed(155)

# Create model
breastcancer = Sequential()

# Add layers
breastcancer.add(Dense(64, input_dim=X.shape[1], activation='relu')) # Input layer
breastcancer.add(Dropout(0.2)) # Dropout layer to prevent overfitting
breastcancer.add(Dense(32, activation='relu')) # Hidden layer
breastcancer.add(Dense(16, activation='relu')) # Hidden layer
breastcancer.add(Dense(8, activation='relu')) # Hidden layer
breastcancer.add(Dense(4, activation='relu')) # Hidden layer
breastcancer.add(Dense(1, activation='sigmoid')) # Output layer

# Optimizer
sgd = SGD(learning_rate=0.01, momentum=0.9)

# Compile
breastcancer.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Train
breastcancer_result = breastcancer.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Evaluate
evaluation = breastcancer.evaluate(X_test, Y_test)

breastcancer.summary()
print('\nLoss: {}, Accuracy: {}'.format(evaluation[0], evaluation[1]))
```



+ Code + Text

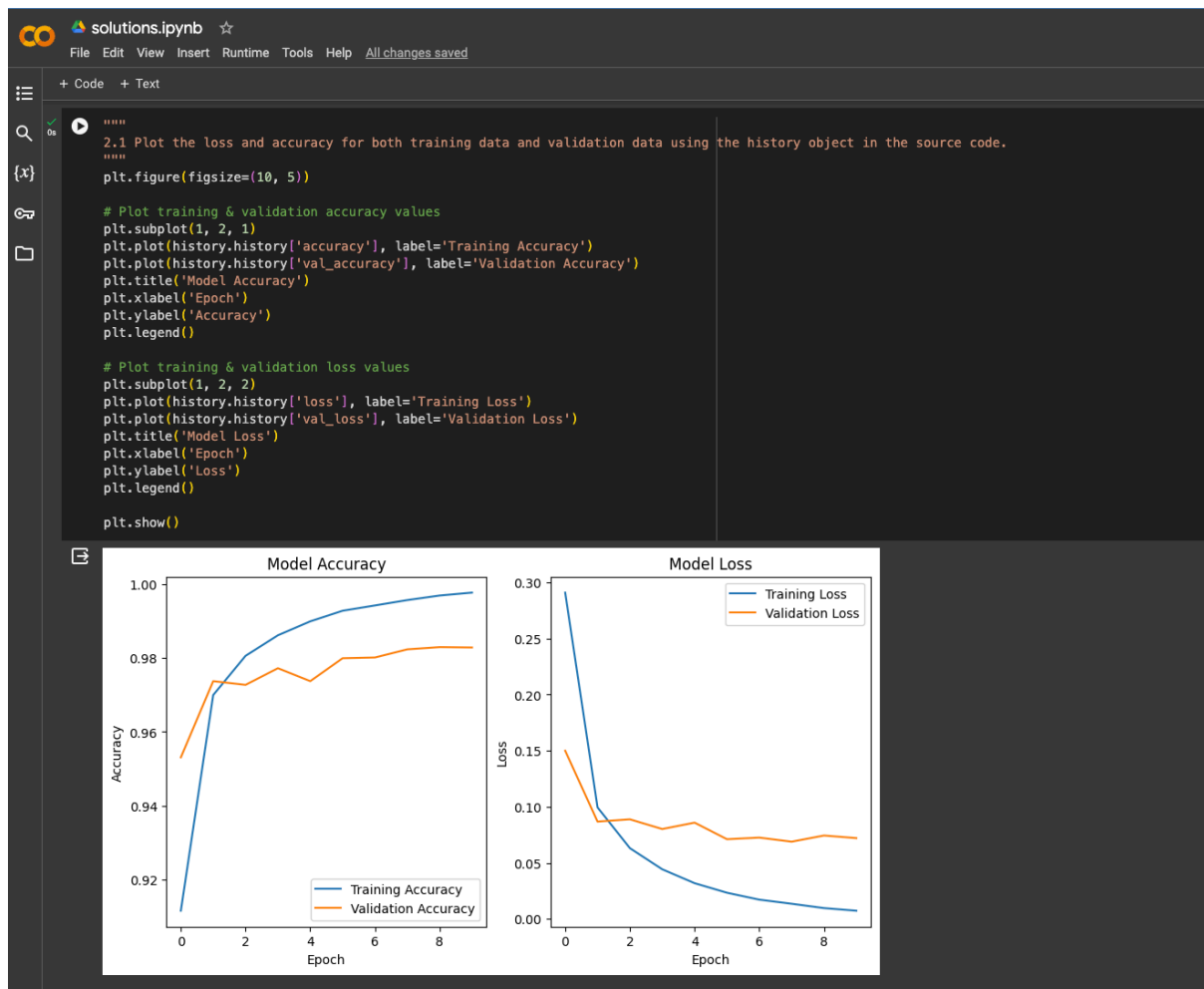
```
14/14 [=====] - 0s 3ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 88/100
14/14 [=====] - 0s 3ms/step - loss: 0.0019 - accuracy: 1.0000
Epoch 89/100
14/14 [=====] - 0s 5ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 90/100
14/14 [=====] - 0s 3ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 91/100
14/14 [=====] - 0s 3ms/step - loss: 0.0039 - accuracy: 0.9977
Epoch 92/100
14/14 [=====] - 0s 3ms/step - loss: 0.0113 - accuracy: 0.9977
Epoch 93/100
14/14 [=====] - 0s 3ms/step - loss: 0.0029 - accuracy: 1.0000
Epoch 94/100
14/14 [=====] - 0s 4ms/step - loss: 0.0024 - accuracy: 1.0000
Epoch 95/100
14/14 [=====] - 0s 3ms/step - loss: 0.0072 - accuracy: 0.9953
Epoch 96/100
14/14 [=====] - 0s 4ms/step - loss: 0.0085 - accuracy: 0.9953
Epoch 97/100
14/14 [=====] - 0s 3ms/step - loss: 0.0015 - accuracy: 1.0000
Epoch 98/100
14/14 [=====] - 0s 3ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 99/100
14/14 [=====] - 0s 3ms/step - loss: 0.0072 - accuracy: 0.9977
Epoch 100/100
14/14 [=====] - 0s 3ms/step - loss: 0.0021 - accuracy: 1.0000
5/5 [=====] - 0s 3ms/step - loss: 0.5798 - accuracy: 0.9720
Model: "sequential_29"
```

Layer (type)	Output Shape	Param #
dense_140 (Dense)	(None, 64)	2048
dropout_19 (Dropout)	(None, 64)	0
dense_141 (Dense)	(None, 32)	2080
dense_142 (Dense)	(None, 16)	528
dense_143 (Dense)	(None, 8)	136
dense_144 (Dense)	(None, 4)	36
dense_145 (Dense)	(None, 1)	5

```
=====  
Total params: 4833 (18.88 KB)  
Trainable params: 4833 (18.88 KB)  
Non-trainable params: 0 (0.00 Byte)
```

```
=====  
Loss: 0.5798060297966003, Accuracy: 0.9720279574394226
```

2. Question 2



solutions.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```

2.2 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.
"""
idx = np.random.randint(0, len(X_test))
image = test_images[idx]
label = test_labels[idx]

plt.imshow(image, cmap='gray')
plt.title('Actual label: {}'.format(label))
plt.axis('off')
plt.show()

# Reshape image and perform inference
image_flat = image.flatten() # Flatten the image
prediction = np.argmax(model.predict(image_flat[np.newaxis, :]))
print('\nModel Prediction: {}'.format(prediction))

```

Actual label: 1

1/1 [=====] - 0s 85ms/step

Model Prediction: 9

solutions.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```

[ ] # Reuse methods

num_hidden_layers = [1, 3] # Try different numbers of hidden layers
activations = ['tanh', 'sigmoid'] # Try different activation functions

# Define the model
def create_model(num_hidden_layers=2, activation='relu'):
    model = Sequential()
    model.add(Dense(512, activation=activation, input_shape=(dimData,)))
    for _ in range(num_hidden_layers - 1):
        model.add(Dense(512, activation=activation))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

```

```

2.3 We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.
"""
# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Process the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float32') / 255.0
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float32') / 255.0
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

# Train and evaluate the model with different configurations
for num_layers in num_hidden_layers:
    for activation in activations:
        print("\nTraining model with {} hidden layers and {} activation function".format(num_layers, activation))
        model = create_model(num_hidden_layers=num_layers, activation=activation)
        history = model.fit((parameter) validation_data: Any | None i ze=256, epochs=10, verbose=1,
                            validation_data=(test_data, test_labels_one_hot))

```

```
solutions.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Epoch 5/10
235/235 [=====] - 8s 34ms/step - loss: 0.0509 - accuracy: 0.9843 - val_loss: 0.0998 - val_accuracy: 0.9684
Epoch 6/10
235/235 [=====] - 9s 36ms/step - loss: 0.0365 - accuracy: 0.9883 - val_loss: 0.0723 - val_accuracy: 0.9769
Epoch 7/10
235/235 [=====] - 8s 36ms/step - loss: 0.0273 - accuracy: 0.9914 - val_loss: 0.0897 - val_accuracy: 0.9739
Epoch 8/10
235/235 [=====] - 8s 32ms/step - loss: 0.0187 - accuracy: 0.9944 - val_loss: 0.0922 - val_accuracy: 0.9732
Epoch 9/10
235/235 [=====] - 9s 40ms/step - loss: 0.0138 - accuracy: 0.9959 - val_loss: 0.0782 - val_accuracy: 0.9762
Epoch 10/10
235/235 [=====] - 7s 31ms/step - loss: 0.0087 - accuracy: 0.9977 - val_loss: 0.0743 - val_accuracy: 0.9798

Training model with 3 hidden layers and sigmoid activation function
Epoch 1/10
235/235 [=====] - 10s 39ms/step - loss: 1.1636 - accuracy: 0.5963 - val_loss: 0.4604 - val_accuracy: 0.8636
Epoch 2/10
235/235 [=====] - 9s 36ms/step - loss: 0.3901 - accuracy: 0.8799 - val_loss: 0.4716 - val_accuracy: 0.8434
Epoch 3/10
235/235 [=====] - 7s 31ms/step - loss: 0.2993 - accuracy: 0.9100 - val_loss: 0.3031 - val_accuracy: 0.9091
Epoch 4/10
235/235 [=====] - 9s 38ms/step - loss: 0.2475 - accuracy: 0.9247 - val_loss: 0.2117 - val_accuracy: 0.9348
Epoch 5/10
235/235 [=====] - 8s 33ms/step - loss: 0.2067 - accuracy: 0.9375 - val_loss: 0.1904 - val_accuracy: 0.9412
Epoch 6/10
235/235 [=====] - 9s 38ms/step - loss: 0.1766 - accuracy: 0.9467 - val_loss: 0.1765 - val_accuracy: 0.9479
Epoch 7/10
235/235 [=====] - 9s 37ms/step - loss: 0.1553 - accuracy: 0.9530 - val_loss: 0.1424 - val_accuracy: 0.9556
Epoch 8/10
235/235 [=====] - 7s 32ms/step - loss: 0.1354 - accuracy: 0.9587 - val_loss: 0.1626 - val_accuracy: 0.9481
Epoch 9/10
235/235 [=====] - 9s 38ms/step - loss: 0.1200 - accuracy: 0.9633 - val_loss: 0.1488 - val_accuracy: 0.9549
Epoch 10/10
235/235 [=====] - 7s 32ms/step - loss: 0.1092 - accuracy: 0.9663 - val_loss: 0.1488 - val_accuracy: 0.9549
```

```
solutions.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

2.4 Run the same code without scaling the images and check the performance

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Process the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float32')
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float32')
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

for num_layers in num_hidden_layers:
    for activation in activations:
        print("\nTraining model with {} hidden layers and {} activation function".format(num_layers, activation))
        model = create_model(num_hidden_layers=num_layers, activation=activation)
        history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                            validation_data=(test_data, test_labels_one_hot))

Training model with 1 hidden layers and tanh activation function
Epoch 1/10
235/235 [=====] - 5s 20ms/step - loss: 0.4227 - accuracy: 0.8728 - val_loss: 0.2778 - val_accuracy: 0.9150
Epoch 2/10
235/235 [=====] - 3s 13ms/step - loss: 0.2493 - accuracy: 0.9246 - val_loss: 0.2308 - val_accuracy: 0.9334
Epoch 3/10
235/235 [=====] - 3s 13ms/step - loss: 0.2119 - accuracy: 0.9363 - val_loss: 0.1920 - val_accuracy: 0.9408
Epoch 4/10
235/235 [=====] - 3s 13ms/step - loss: 0.1863 - accuracy: 0.9442 - val_loss: 0.1824 - val_accuracy: 0.9433
Epoch 5/10
235/235 [=====] - 4s 19ms/step - loss: 0.1697 - accuracy: 0.9484 - val_loss: 0.1701 - val_accuracy: 0.9487
Epoch 6/10
235/235 [=====] - 3s 13ms/step - loss: 0.1585 - accuracy: 0.9527 - val_loss: 0.1571 - val_accuracy: 0.9544
Epoch 7/10
235/235 [=====] - 3s 13ms/step - loss: 0.1464 - accuracy: 0.9556 - val_loss: 0.1693 - val_accuracy: 0.9480
Epoch 8/10
235/235 [=====] - 3s 13ms/step - loss: 0.1368 - accuracy: 0.9582 - val_loss: 0.1449 - val_accuracy: 0.9576
Epoch 9/10
235/235 [=====] - 5s 19ms/step - loss: 0.1307 - accuracy: 0.9599 - val_loss: 0.1425 - val_accuracy: 0.9562
Epoch 10/10
235/235 [=====] - 3s 13ms/step - loss: 0.1277 - accuracy: 0.9608 - val_loss: 0.1322 - val_accuracy: 0.9580

Training model with 1 hidden layers and sigmoid activation function
Epoch 1/10
235/235 [=====] - 4s 13ms/step - loss: 0.4029 - accuracy: 0.8880 - val_loss: 0.2601 - val_accuracy: 0.9219
Epoch 2/10
235/235 [=====] - 3s 13ms/step - loss: 0.2208 - accuracy: 0.9363 - val_loss: 0.1995 - val_accuracy: 0.9405
Epoch 3/10
235/235 [=====] - 4s 19ms/step - loss: 0.1856 - accuracy: 0.9449 - val_loss: 0.1724 - val_accuracy: 0.9488
Epoch 4/10
235/235 [=====] - 3s 13ms/step - loss: 0.1639 - accuracy: 0.9511 - val_loss: 0.1556 - val_accuracy: 0.9546
Epoch 5/10
235/235 [=====] - 3s 13ms/step - loss: 0.1450 - accuracy: 0.9570 - val_loss: 0.1525 - val_accuracy: 0.9527
Epoch 6/10
235/235 [=====] - 3s 13ms/step - loss: 0.1339 - accuracy: 0.9597 - val_loss: 0.1364 - val_accuracy: 0.9589
Epoch 7/10
235/235 [=====] - 4s 19ms/step - loss: 0.1257 - accuracy: 0.9619 - val_loss: 0.1370 - val_accuracy: 0.9573
Epoch 8/10
235/235 [=====] - 3s 13ms/step - loss: 0.1200 - accuracy: 0.9637 - val_loss: 0.1256 - val_accuracy: 0.9607
Epoch 9/10
235/235 [=====] - 3s 13ms/step - loss: 0.1121 - accuracy: 0.9661 - val_loss: 0.1191 - val_accuracy: 0.9632
Epoch 10/10
235/235 [=====] - 3s 13ms/step - loss: 0.1053 - accuracy: 0.9681 - val_loss: 0.1150 - val_accuracy: 0.9644
```