

ICP 4

Name: Venkata Sai Prasad Nakka

700#: 700755524

GitHub Link: <https://github.com/venkat137222/week-4---ICP4>

Video Link:

https://drive.google.com/file/d/1sdCV_lznPPaodR5g0mqdDgVHWwjFHCma/view?usp=drive_link

1. Use the use case in the class:

a. Add more Dense layers to the existing code and check how the accuracy changes.

```
import keras
import pandas
from keras.models import Sequential
from keras.layers 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 1
my_first_nn.add(Dense(15, activation='relu')) # Hidden layer 2
my_first_nn.add(Dense(10, activation='relu')) # Hidden layer 3
my_first_nn.add(Dense(5, activation='relu')) # Hidden layer 4
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 100/100
 18/18 — 0s 6ms/step - acc: 0.7607 - loss: 0.5056
 Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 20)	180
dense_1 (Dense)	(None, 15)	315
dense_2 (Dense)	(None, 10)	160
dense_3 (Dense)	(None, 5)	55
dense_4 (Dense)	(None, 1)	6

Total params: 2,150 (8.40 KB)
 Trainable params: 716 (2.80 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 1,434 (5.61 KB)

None
 6/6 — 0s 18ms/step - acc: 0.6887 - loss: 0.5574
 [0.5714445114135742, 0.6822916865348816]

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

```
[ ] # 2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes.
    # Report accuracy of the model.
```

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

```
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split

dataset = pd.read_csv(path_to_csv)

X = dataset.loc[:, 'radius_mean':'fractal_dimension_worst']
Y = dataset['diagnosis']
# Mapping 'M' to 0 and 'B' to 1 for binary classification
Y = Y.map({'M': 0, 'B': 1}).astype(int)

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

np.random.seed(155)
my_second_nn = Sequential()
my_second_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_second_nn.add(Dense(15, activation='relu'))              # Hidden layer 2
my_second_nn.add(Dense(10, activation='relu'))              # Hidden layer 3
my_second_nn.add(Dense(5, activation='relu'))               # Hidden layer 4
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 98/100
14/14 ----- 0s 4ms/step - acc: 0.9328 - loss: 0.2499
Epoch 99/100
14/14 ----- 0s 4ms/step - acc: 0.9332 - loss: 0.2461
Epoch 100/100
14/14 ----- 0s 5ms/step - acc: 0.9423 - loss: 0.2583
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 20)	620
dense_6 (Dense)	(None, 15)	315
dense_7 (Dense)	(None, 10)	160
dense_8 (Dense)	(None, 5)	55
dense_9 (Dense)	(None, 1)	6

```
Total params: 3,470 (13.56 KB)
Trainable params: 1,156 (4.52 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2,314 (9.04 KB)
None
5/5 ----- 0s 8ms/step - acc: 0.8775 - loss: 0.3952
[0.3371555209159851, 0.9020978808403015]
```

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()

```
[ ] import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

dataset = pd.read_csv(path_to_csv)

X = dataset.loc[:, 'radius_mean':'fractal_dimension_worst']
Y = dataset['diagnosis']
# Map 'M' to 0 and 'B' to 1 for binary classification
Y = Y.map({'M': 0, 'B': 1}).astype(int)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)
np.random.seed(155)

sc = StandardScaler()
normalized_Xtrain = sc.fit_transform(X_train)
normalized_Xtest = sc.transform(X_test)

my_third_nn = Sequential()
my_third_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_third_nn.add(Dense(15, activation='relu')) # Hidden layer 2
my_third_nn.add(Dense(10, activation='relu')) # Hidden layer 3
my_third_nn.add(Dense(5, activation='relu')) # Hidden layer 4
my_third_nn.add(Dense(1, activation='sigmoid'))
my_third_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_third_nn_fitted = my_third_nn.fit(normalized_Xtrain, Y_train, epochs=100, initial_epoch=0)

print(my_third_nn.summary())
print(my_third_nn.evaluate(normalized_Xtest, Y_test))
```

```

Epoch 98/100
14/14 ————— 0s 7ms/step - acc: 1.0000 - loss: 5.8646e-04
Epoch 99/100
14/14 ————— 0s 8ms/step - acc: 1.0000 - loss: 4.9717e-04
Epoch 100/100
14/14 ————— 0s 8ms/step - acc: 1.0000 - loss: 3.8835e-04
Model: "sequential_2"

```

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 20)	620
dense_11 (Dense)	(None, 15)	315
dense_12 (Dense)	(None, 10)	160
dense_13 (Dense)	(None, 5)	55
dense_14 (Dense)	(None, 1)	6

```

Total params: 3,470 (13.56 KB)
Trainable params: 1,156 (4.52 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2,314 (9.04 KB)
None
5/5 ————— 0s 13ms/step - acc: 0.9448 - loss: 0.4658
[0.42359066009521484, 0.9580419659614563]

```

Indeed, the accuracy of the model improved from approximately 0.9161 (91.61%) before normalization to approximately 0.9650 (96.50%) after normalization

Use Image Classification on the hand written digits data set (mnist)

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

```

from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt

(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 from integer 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, test_data, validation_data=(test_data, test_labels_one_hot), batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))

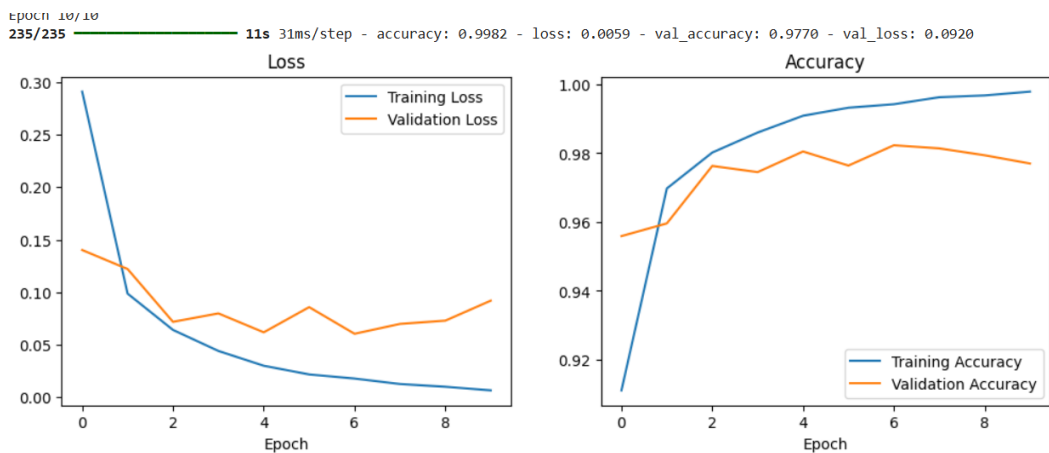
# Extract training history
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']
```

```
# Plot loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.legend()

# Plot accuracy
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.show()
```

Output:



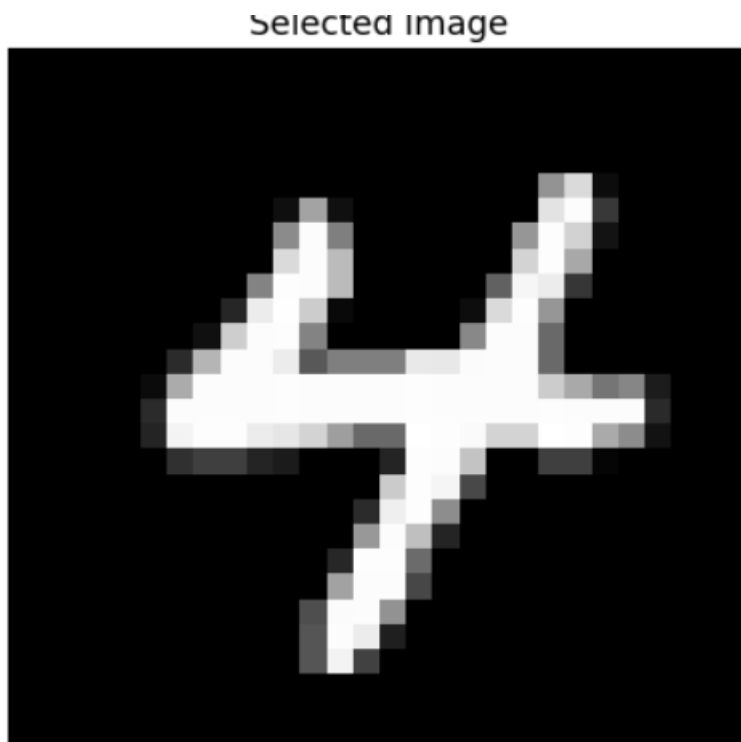
2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```
# plot the selected image
plt.figure()
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.title('Selected Image')

# do inferencing to check the model prediction on the selected image
prediction = model.predict(image.reshape(1, 784))
prediction = np.argmax(prediction)

# print the predicted label
print('Predicted label:', prediction)
```

Output:



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.

```
[ ] # 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.
```

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

    (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 from integer 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='tanh', input_shape=(dimData,)))
model.add(Dense(256, activation='tanh'))
model.add(Dense(128, activation='tanh'))
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))

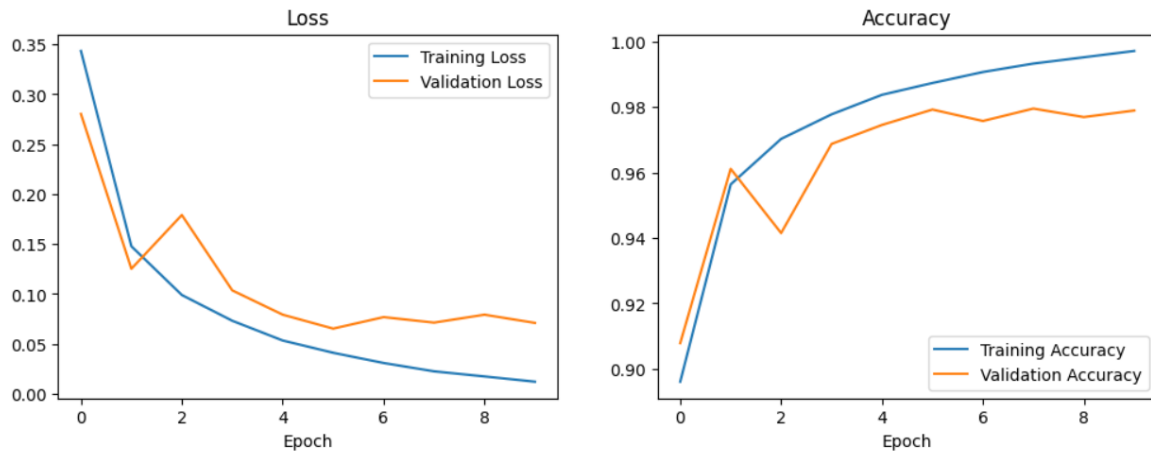
# Extract training history
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']
```

```
# Plot loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.legend()

# Plot accuracy
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.show()
```

Output:



4. Run the same code without scaling the images and check the performance?

```
# 4. Run the same code without scaling the images and check the performance?

import matplotlib.pyplot as plt
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 (no scaling)
train_data = train_data.astype('float')
test_data = test_data.astype('float')

# Change the labels from integer 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='tanh', input_shape=(dimData,)))
model.add(Dense(256, activation='tanh'))
model.add(Dense(128, activation='tanh'))
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=(validation_data, test_labels_one_hot))

# Extract training history
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

# Plot loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.legend()

# Plot accuracy
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.show()

```

Output:

