Lab 11 - TensorFlow / Keras

Name: Nick Satriano and Jake Luther
Class: CSCI 349 - Intro to Data Mining
Semester: Spring 2023
Instructor: Brian King

Exercise 1 – Interview questions about deep learning and keras

1) What is an artificial neural network (ANN)?

An artificial neural network is a system of nodes that mimic the way in which a human brain is structured. There are several layers of these nodes: an input layer, some number of hidden layers, and an output layer.

These nodes are connected to one other, each represent an artifical neuron, and they each have an associated weight and threshold. When a node produces an output value that is greater than the threshold, the node is then activated and it sends data to the next layer of the neural network.

citation: https://www.ibm.com/topics/neural-networks

2) What is deep learning? How does it relate to ANN?

Deep learning refers to the depth of the layers of an ANN. When an ANN has more than 3 layers, it is considered to be a deep learning algorithm.

citation: https://www.ibm.com/topics/neural-networks

3) Name a couple of examples where deep learning has made a tremendous impact.

Deep learning has impacted computer vision in an extreme manner. The current capabilities for image classification and object detection have allowed for the (somewhat) safe and widespread implementation of autonomous vehicles, facial recognition technologies, and medical imaging analysis.

To cite another example, deep learning has had a tremendous impact in language processing. Chatbots, like ChatGPT, have become much more powerful and are capable of generating some freakishly human-like and intelligent responses to human inputs.

citation: ChatGPT

4) Briefly, what is the feedforward algorithm with a neural net?

A feedforward algorithm is one that flows in one direction only: from the input layer to the output layer.

citation: https://www.ibm.com/topics/neural-networks

5) In the context of machine learning, what is a loss function?

A loss function evaluates the accuracy of a model as it is being trained. This is commonly known as the mean squared error (MSE).

citation: https://www.ibm.com/topics/neural-networks

6) What is gradient descent? And how is the loss function a critical part of gradient descent?

Gradient descent is the process through which an algorithm adjusts its weights, which allows the model to determine which direction it must take to reduce the loss function.

7) Training a neural net involves the backpropagation algorithm. In a few sentences, describe what this algorithm does.

The backpropagation algorithm allows data to flow from the output layer of the neural net to the input layer. Backpropagation makes it possible to calculate and find the error with each neuron. This process allows us to adjust and fit the parameters of the model accurately and effectively.

citation: https://www.ibm.com/topics/neural-networks

8) What is the difference between batch gradient descent and stochastic gradient descent?

Batch gradient descent involves calculating the gradient of the cost function with respect to the model parameters for the entire training dataset in one go. The model parameters are then updated once per iteration based on this calculated gradient. The term "batch" refers to the fact that the entire training dataset is used in each iteration.

Stochastic gradient descent, on the other hand, involves calculating the gradient of the cost function with respect to the model parameters for each training example separately and updating the model parameters after each example. The term "stochastic" refers to the random nature of the selection of each training example.

citation: ChatGPT

9) In the context of neural network training, explain the terms epoch and batch.

An epoch is a complete pass through the entire training dataset during training. In other words, one epoch is when the entire training set is seen once by the model.

A batch is a subset of the training dataset that is used to update the model's parameters. During training, the model receives input data in batches, processes it, computes the loss, and updates the parameters based on the gradients of the loss function with respect to the parameters. The size of the batch, or batch size, is a hyperparameter that determines how many samples the model processes at a time

citation: ChatGPT

10) In the context of machine learning, what is a hyperparameter?

A hyperparameter is a parameter whose value is set before the training process of a machine learning model begins, and which influences the behavior and performance of the model.

11) In the context of neural nets training, what are examples of hyperparameters that can affect model performance?

Here are some examples of hyperparameters that can affect model performance:

- *Learning rate:* determines how quickly the model learns from the data during training
- *Number of epochs:* determines how many times the model will be trained on the entire training dataset
- *Batch size:* determines how many samples are used to update the model weights in each training iteration
- *Number of layers:* determines the depth of the neural network architecture
- *Number of neurons per layer:* determines the width of the neural network architecture
- *Activation functions:* determines the non-linear activation functions used in the network's layers

citation: https://www.analyticsvidhya.com/blog/2022/05/impact-of-hyperparameters-on-a-deep-learning-model/

12) What is an activation function?

An activation function is a mathematical function applied to the output of each neuron in a neural network. It determines whether the neuron should be "activated" or not based on the input it receives.

citation: https://www.ibm.com/topics/neural-networks

13) Most agree that the most popular activation functions are sigmoid, hyperbolic tangent (tanh), softmax, and ReLu (rectified linear unit). Compare and contrast each, using whatever resources you want. Again, 1-2 sentences for each Is sufficient.

- *Sigmoid:* it maps any input to a value between 0 and 1, and is useful for models that need to output probabilities. It has a problem of vanishing gradients and is rarely used in deep neural networks.

- *Hyperbolic tangent:* it maps any input to a value between -1 and 1, and is also useful for models that need to output probabilities. It suffers from the same vanishing gradient problem as sigmoid.
- *Softmax:* it maps the inputs to a probability distribution over multiple classes. It is used for multiclass classification problems and ensures that the sum of the probabilities is equal to 1.
- *ReLU:* it is defined as max(0, x), and is a non-linear activation function that is widely used in deep neural networks. It is computationally efficient and has been shown to perform well in practice. However, it can suffer from the "dying ReLU" problem, where a large number of neurons can output zero, effectively killing the gradient and making the neuron "dead".

citation: ChatGPT

14) Why is ReLu so popular for large, deep learning networks?

ReLu (Rectified Linear Unit) is popular for large, deep learning networks because it overcomes the vanishing gradient problem, which can occur with sigmoid and tanh activation functions. The vanishing gradient problem refers to the situation where the gradients during backpropagation become very small, leading to slow or no learning in deeper layers of the network.

15) Why is softmax most appropriate for the output layer, especially for classification problems?

The softmax activation function is most appropriate for the output layer of neural networks in classification problems because it converts the output into a probability distribution over the classes. This means that the outputs of the softmax function represent the probabilities of each class, which can be used to make predictions on the most likely class. Additionally, the softmax function ensures that the probabilities sum up to 1, which is a necessary condition for a probability distribution.

citation: ChatGPT

16) What does ReLu sometimes suffer from, and how does a Leaky ReLu activation address it?

ReLu can sometimes lead to a "dying ReLu" problem, where neurons can become stuck at zero and stop learning. A Leaky ReLu activation function is a modified version of ReLu that addresses this problem. The Leaky ReLu allows a small, non-zero gradient for negative inputs, which helps to prevent the "dying ReLu" problem and can also improve the overall performance of the neural network.

citation: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

17) What is Tensorflow? Who created it?

TensorFlow is a machine learning framework designed and developed by Google.

citation: https://www.tensorflow.org/about

18) What are tensors?

Tensors are mathematical objects representing arrays of data with multiple dimensions.

19) What is keras? Who created it?

Keras is a neural network API written in Python. It allows for fast experimentation with deep neural networks, and it focuses on being user-friendly. Keras was originally developed by Francois Chollet and was released in March of 2015.

citation: https://keras.io/about/

20) Explain the relationship between keras and tensorflow. How are they similar? Different?

Keras acts as a simplified interface for training neural networks, while TensorFlow is a lower-level library for computation that allows for the utilization of complex mathematical operations which are at the basis of creating and training neural networks.

Keras is integrated into TensorFlow as a part of the TensorFlow library. This allows developers to access the high-level interface of Keras while also leveraging the low-level, mathematical functionalities of TensorFlow.

21) Describe at a very top level what the Functional API is.

The Functional API is an interface in Keras that allows for building complex models with multiple input/output layers, shared layers, and other advanced architectures. It provides a way to define a computational graph of layers and how they are connected, giving more flexibility and control over the model architecture compared to the Sequential API.

citation: https://keras.io/api/

22) What is the Sequential class, and how does it compare / contrast to the Functional API?

The Sequential class in Keras is a way to create a neural network model by simply adding one layer at a time, in sequence. In contrast, the Functional API allows for more complex, non-linear architectures.

citation: https://keras.io/api/

23) What is a layer? How is a layer added to a model?

In a neural network, a layer is a set of neurons that process a set of input data and produce a set of output values. Layers are added to a model in Keras using the add() method.

citation: https://keras.io/api/layers/

24) What is a Dense layer?

A Dense layer is a fully connected layer, meaning that all the neurons in a Dense layer receive input from all the neurons in the previous layer.

citation: https://keras.io/api/layers/core_layers/dense/

25) What does the compile method do for a model, and what two parameters are required to compile every model?

The compile method is used to configure a model for training. The two parameters that are required to compile every model are: optimizer and loss.

citation: https://keras.io/api/models/model_training_apis/

Exercise 2 - The return of iris

In [123…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.graph_objects as go
import plotly.express as px
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDi
import sklearn.utils as utils
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
import tensorflow as tf
from tensorflow import keras
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Activation
from tensorflow.keras.optimizers import SGD
print(tf.__version__)
print(keras.__version__)
```

```
2.10.0
2.10.0
```

1) Read in and preprocess the data.

In [6]:
```
df_iris = sns.load_dataset('iris')
df_iris['species'] = df_iris['species'].astype('category')

df_iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype   
---  ------        --------------  -----   
 0   sepal_length  150 non-null    float64 
 1   sepal_width   150 non-null    float64 
 2   petal_length  150 non-null    float64 
 3   petal_width   150 non-null    float64 
 4   species       150 non-null    category
dtypes: category(1), float64(4)
memory usage: 5.1 KB
```

In [9]:
```
X = df_iris.iloc[:,:-1]

X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

In [10]:
```
y = df_iris['species']

y.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 150 entries, 0 to 149
Series name: species
Non-Null Count  Dtype   
--------------  -----   
150 non-null    category
dtypes: category(1)
memory usage: 410.0 bytes
```

2) Shuffle the data in your data frames.

In [12]:
```
X, y = utils.shuffle(X, y, random_state=0)
```

3) Use train_test_split to split your data, but this time, let's use an even smaller split, using a 50/50 split, initializing with a random state of 0.

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=

X_train
```

Out[44]:

	sepal_length	sepal_width	petal_length	petal_width
107	7.3	2.9	6.3	1.8
47	4.6	3.2	1.4	0.2
129	7.2	3.0	5.8	1.6
40	5.0	3.5	1.3	0.3
48	5.3	3.7	1.5	0.2
...	...	...	...	...
71	6.1	2.8	4.0	1.3
143	6.8	3.2	5.9	2.3
125	7.2	3.2	6.0	1.8
65	6.7	3.1	4.4	1.4
144	6.7	3.3	5.7	2.5

75 rows × 4 columns

4) How many inputs will your network need to have?

The network will need 4 inputs since there are four attributes in the dataset: sepal length, sepal width, petal length, and petal width.

5) How do you represent a multi-class target variable in a model like a neural net? For the iris data, what does the final layer of your neural net structure need to look like?

To represent a multi-class target variable in a model like a neural net, we must create a binary vector for each class, where each element in the vector corresponds to whether that observation belongs to that class or not.

For the iris data, the final layer of the neutal net structure will need to have 3 output nodes, each representing one of the three classes of iris: setosa, versicolor, and virginica.

Assisted by: ChatGPT

6) Write the code to convert the iris target variables to a set of binarized variables derived from the target class variable.

In [43]:
```
y_train_bin = pd.get_dummies(y_train)
y_test_bin = pd.get_dummies(y_test)

y_train_bin
```

Out[43]:

	setosa	versicolor	virginica
107	0	0	1
47	1	0	0
129	0	0	1
40	1	0	0
48	1	0	0
...	...	...	...
71	0	1	0
143	0	0	1
125	0	0	1
65	0	1	0
144	0	0	1

75 rows × 3 columns

7) Create your input node using 'Input'.

In [23]:
```
# citation: https://keras.io/guides/functional_api/
inputs = keras.Input(shape=(4,))
```

8) Add the hidden layer(s), and specify the input as you move along that each layer should receive.

In [25]:
```
layer = Dense(12, activation='sigmoid')(inputs)
```

9) Add one more Dense layer representing the output layer.

In [27]:
```
output_layer = Dense(3, activation='softmax')(layer)
```

10) Create an instance of Model. Specify the inputs, the outputs, and name the model "simple_iris_model". Then, report a summary of your model using the summary() function.

In [47]:
```
simple_iris_model = keras.Model(inputs=inputs, outputs=output_layer, name='simple_iris

simple_iris_model.summary()
```

```
Model: "simple_iris_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 4)]               0

 dense_1 (Dense)             (None, 12)                60

 dense_3 (Dense)             (None, 3)                 39

=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________
```

11) Now, compile your model.

In [48]:
```
simple_iris_model.compile(
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)
```

12) Use the fit method to train your model with X_train and your binarized y_train data.

In [64]:
```
history = simple_iris_model.fit(
    X_train, y_train_bin,
    epochs = 100,
    batch_size = 1,
    verbose = 1,
    validation_data = (X_test, y_test_bin)
)
```

```
Epoch 1/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0396 - accuracy: 0.986
7 - val_loss: 0.0912 - val_accuracy: 0.9733
Epoch 2/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0379 - accuracy: 0.986
7 - val_loss: 0.0972 - val_accuracy: 0.9600
Epoch 3/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0331 - accuracy: 0.986
7 - val_loss: 0.1014 - val_accuracy: 0.9600
Epoch 4/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0438 - accuracy: 0.973
3 - val_loss: 0.0939 - val_accuracy: 0.9733
Epoch 5/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0402 - accuracy: 0.986
7 - val_loss: 0.0990 - val_accuracy: 0.9467
Epoch 6/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0348 - accuracy: 0.986
7 - val_loss: 0.0897 - val_accuracy: 0.9733
Epoch 7/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0375 - accuracy: 0.986
7 - val_loss: 0.0972 - val_accuracy: 0.9600
Epoch 8/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0378 - accuracy: 0.973
3 - val_loss: 0.0939 - val_accuracy: 0.9600
Epoch 9/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0362 - accuracy: 0.973
3 - val_loss: 0.0904 - val_accuracy: 0.9733
Epoch 10/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0346 - accuracy: 0.986
7 - val_loss: 0.0890 - val_accuracy: 0.9733
Epoch 11/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0335 - accuracy: 0.986
7 - val_loss: 0.0894 - val_accuracy: 0.9733
Epoch 12/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0325 - accuracy: 1.000
0 - val_loss: 0.0887 - val_accuracy: 0.9733
Epoch 13/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0348 - accuracy: 0.986
7 - val_loss: 0.0902 - val_accuracy: 0.9733
Epoch 14/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0370 - accuracy: 0.973
3 - val_loss: 0.0903 - val_accuracy: 0.9733
Epoch 15/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0363 - accuracy: 0.986
7 - val_loss: 0.0888 - val_accuracy: 0.9733
Epoch 16/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0322 - accuracy: 1.000
0 - val_loss: 0.0881 - val_accuracy: 0.9733
Epoch 17/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0358 - accuracy: 0.986
7 - val_loss: 0.0889 - val_accuracy: 0.9867
Epoch 18/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0327 - accuracy: 0.986
7 - val_loss: 0.0878 - val_accuracy: 0.9733
Epoch 19/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0335 - accuracy: 0.986
7 - val_loss: 0.0888 - val_accuracy: 0.9733
Epoch 20/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0338 - accuracy: 0.986
7 - val_loss: 0.0876 - val_accuracy: 0.9733
```

```
Epoch 21/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0344 - accuracy: 0.973
3 - val_loss: 0.0874 - val_accuracy: 0.9733
Epoch 22/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0341 - accuracy: 0.986
7 - val_loss: 0.0881 - val_accuracy: 0.9867
Epoch 23/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0349 - accuracy: 0.986
7 - val_loss: 0.0890 - val_accuracy: 0.9733
Epoch 24/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0330 - accuracy: 0.986
7 - val_loss: 0.0907 - val_accuracy: 0.9600
Epoch 25/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0332 - accuracy: 1.000
0 - val_loss: 0.0891 - val_accuracy: 0.9600
Epoch 26/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0324 - accuracy: 0.986
7 - val_loss: 0.0874 - val_accuracy: 0.9733
Epoch 27/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0338 - accuracy: 1.000
0 - val_loss: 0.0878 - val_accuracy: 0.9867
Epoch 28/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0320 - accuracy: 0.986
7 - val_loss: 0.0891 - val_accuracy: 0.9600
Epoch 29/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0324 - accuracy: 0.986
7 - val_loss: 0.0866 - val_accuracy: 0.9733
Epoch 30/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0330 - accuracy: 0.986
7 - val_loss: 0.0897 - val_accuracy: 0.9600
Epoch 31/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.986
7 - val_loss: 0.0874 - val_accuracy: 0.9733
Epoch 32/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0303 - accuracy: 0.986
7 - val_loss: 0.0865 - val_accuracy: 0.9733
Epoch 33/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0333 - accuracy: 0.986
7 - val_loss: 0.0864 - val_accuracy: 0.9733
Epoch 34/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0332 - accuracy: 0.986
7 - val_loss: 0.0862 - val_accuracy: 0.9733
Epoch 35/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0316 - accuracy: 0.986
7 - val_loss: 0.0860 - val_accuracy: 0.9733
Epoch 36/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0317 - accuracy: 0.986
7 - val_loss: 0.0860 - val_accuracy: 0.9733
Epoch 37/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0330 - accuracy: 0.986
7 - val_loss: 0.0860 - val_accuracy: 0.9733
Epoch 38/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0319 - accuracy: 0.986
7 - val_loss: 0.0870 - val_accuracy: 0.9733
Epoch 39/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0317 - accuracy: 0.986
7 - val_loss: 0.0890 - val_accuracy: 0.9600
Epoch 40/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0308 - accuracy: 0.986
7 - val_loss: 0.0858 - val_accuracy: 0.9733
```

```
Epoch 41/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0327 - accuracy: 1.000
0 - val_loss: 0.0858 - val_accuracy: 0.9733
Epoch 42/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0299 - accuracy: 0.986
7 - val_loss: 0.0864 - val_accuracy: 0.9733
Epoch 43/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0287 - accuracy: 1.000
0 - val_loss: 0.0860 - val_accuracy: 0.9733
Epoch 44/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0322 - accuracy: 0.986
7 - val_loss: 0.0908 - val_accuracy: 0.9600
Epoch 45/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0306 - accuracy: 0.986
7 - val_loss: 0.0858 - val_accuracy: 0.9733
Epoch 46/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.986
7 - val_loss: 0.0857 - val_accuracy: 0.9867
Epoch 47/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0306 - accuracy: 0.986
7 - val_loss: 0.0917 - val_accuracy: 0.9600
Epoch 48/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0311 - accuracy: 0.986
7 - val_loss: 0.0866 - val_accuracy: 0.9733
Epoch 49/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0303 - accuracy: 0.986
7 - val_loss: 0.0889 - val_accuracy: 0.9600
Epoch 50/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0301 - accuracy: 0.986
7 - val_loss: 0.0893 - val_accuracy: 0.9600
Epoch 51/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0295 - accuracy: 1.000
0 - val_loss: 0.0851 - val_accuracy: 0.9733
Epoch 52/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0315 - accuracy: 0.986
7 - val_loss: 0.0847 - val_accuracy: 0.9733
Epoch 53/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0304 - accuracy: 1.000
0 - val_loss: 0.0846 - val_accuracy: 0.9733
Epoch 54/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0288 - accuracy: 0.986
7 - val_loss: 0.0862 - val_accuracy: 0.9733
Epoch 55/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0296 - accuracy: 0.986
7 - val_loss: 0.0847 - val_accuracy: 0.9733
Epoch 56/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0300 - accuracy: 1.000
0 - val_loss: 0.0846 - val_accuracy: 0.9733
Epoch 57/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0328 - accuracy: 0.973
3 - val_loss: 0.0844 - val_accuracy: 0.9733
Epoch 58/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0289 - accuracy: 0.986
7 - val_loss: 0.0843 - val_accuracy: 0.9733
Epoch 59/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0296 - accuracy: 0.986
7 - val_loss: 0.0842 - val_accuracy: 0.9733
Epoch 60/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0283 - accuracy: 0.986
7 - val_loss: 0.0845 - val_accuracy: 0.9733
```

```
Epoch 61/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0267 - accuracy: 1.000
0 - val_loss: 0.0887 - val_accuracy: 0.9600
Epoch 62/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0314 - accuracy: 0.986
7 - val_loss: 0.0840 - val_accuracy: 0.9733
Epoch 63/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0310 - accuracy: 0.986
7 - val_loss: 0.0873 - val_accuracy: 0.9600
Epoch 64/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0297 - accuracy: 0.986
7 - val_loss: 0.0840 - val_accuracy: 0.9733
Epoch 65/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0290 - accuracy: 0.986
7 - val_loss: 0.0865 - val_accuracy: 0.9600
Epoch 66/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0304 - accuracy: 0.986
7 - val_loss: 0.0838 - val_accuracy: 0.9733
Epoch 67/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0304 - accuracy: 0.986
7 - val_loss: 0.0837 - val_accuracy: 0.9733
Epoch 68/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0307 - accuracy: 0.986
7 - val_loss: 0.0837 - val_accuracy: 0.9733
Epoch 69/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0301 - accuracy: 0.986
7 - val_loss: 0.0836 - val_accuracy: 0.9733
Epoch 70/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0282 - accuracy: 0.986
7 - val_loss: 0.0909 - val_accuracy: 0.9600
Epoch 71/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0292 - accuracy: 1.000
0 - val_loss: 0.0835 - val_accuracy: 0.9733
Epoch 72/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0279 - accuracy: 0.986
7 - val_loss: 0.0847 - val_accuracy: 0.9733
Epoch 73/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0287 - accuracy: 1.000
0 - val_loss: 0.0834 - val_accuracy: 0.9733
Epoch 74/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0281 - accuracy: 0.986
7 - val_loss: 0.0861 - val_accuracy: 0.9600
Epoch 75/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0290 - accuracy: 0.986
7 - val_loss: 0.0894 - val_accuracy: 0.9600
Epoch 76/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0285 - accuracy: 0.986
7 - val_loss: 0.0832 - val_accuracy: 0.9733
Epoch 77/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0275 - accuracy: 0.986
7 - val_loss: 0.0845 - val_accuracy: 0.9733
Epoch 78/100
75/75 [==============================] - 0s 4ms/step - loss: 0.0288 - accuracy: 0.986
7 - val_loss: 0.0862 - val_accuracy: 0.9600
Epoch 79/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0291 - accuracy: 0.986
7 - val_loss: 0.0880 - val_accuracy: 0.9600
Epoch 80/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0291 - accuracy: 0.986
7 - val_loss: 0.0876 - val_accuracy: 0.9600
```

```
Epoch 81/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0299 - accuracy: 0.986
7 - val_loss: 0.0831 - val_accuracy: 0.9733
Epoch 82/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0279 - accuracy: 1.000
0 - val_loss: 0.0830 - val_accuracy: 0.9733
Epoch 83/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0301 - accuracy: 0.986
7 - val_loss: 0.0896 - val_accuracy: 0.9600
Epoch 84/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0279 - accuracy: 1.000
0 - val_loss: 0.0873 - val_accuracy: 0.9600
Epoch 85/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0265 - accuracy: 1.000
0 - val_loss: 0.0833 - val_accuracy: 0.9733
Epoch 86/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0279 - accuracy: 0.986
7 - val_loss: 0.0885 - val_accuracy: 0.9600
Epoch 87/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0281 - accuracy: 1.000
0 - val_loss: 0.0883 - val_accuracy: 0.9600
Epoch 88/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0289 - accuracy: 0.986
7 - val_loss: 0.0828 - val_accuracy: 0.9733
Epoch 89/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0271 - accuracy: 1.000
0 - val_loss: 0.0829 - val_accuracy: 0.9733
Epoch 90/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0284 - accuracy: 0.986
7 - val_loss: 0.0825 - val_accuracy: 0.9733
Epoch 91/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0261 - accuracy: 1.000
0 - val_loss: 0.0844 - val_accuracy: 0.9733
Epoch 92/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0280 - accuracy: 0.986
7 - val_loss: 0.0881 - val_accuracy: 0.9600
Epoch 93/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0270 - accuracy: 1.000
0 - val_loss: 0.0851 - val_accuracy: 0.9733
Epoch 94/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0276 - accuracy: 0.986
7 - val_loss: 0.0838 - val_accuracy: 0.9733
Epoch 95/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0280 - accuracy: 1.000
0 - val_loss: 0.0852 - val_accuracy: 0.9600
Epoch 96/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0297 - accuracy: 0.986
7 - val_loss: 0.0822 - val_accuracy: 0.9733
Epoch 97/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0269 - accuracy: 0.986
7 - val_loss: 0.0829 - val_accuracy: 0.9733
Epoch 98/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0250 - accuracy: 1.000
0 - val_loss: 0.0849 - val_accuracy: 0.9733
Epoch 99/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0290 - accuracy: 0.986
7 - val_loss: 0.0854 - val_accuracy: 0.9600
Epoch 100/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0263 - accuracy: 1.000
0 - val_loss: 0.0821 - val_accuracy: 0.9733
```

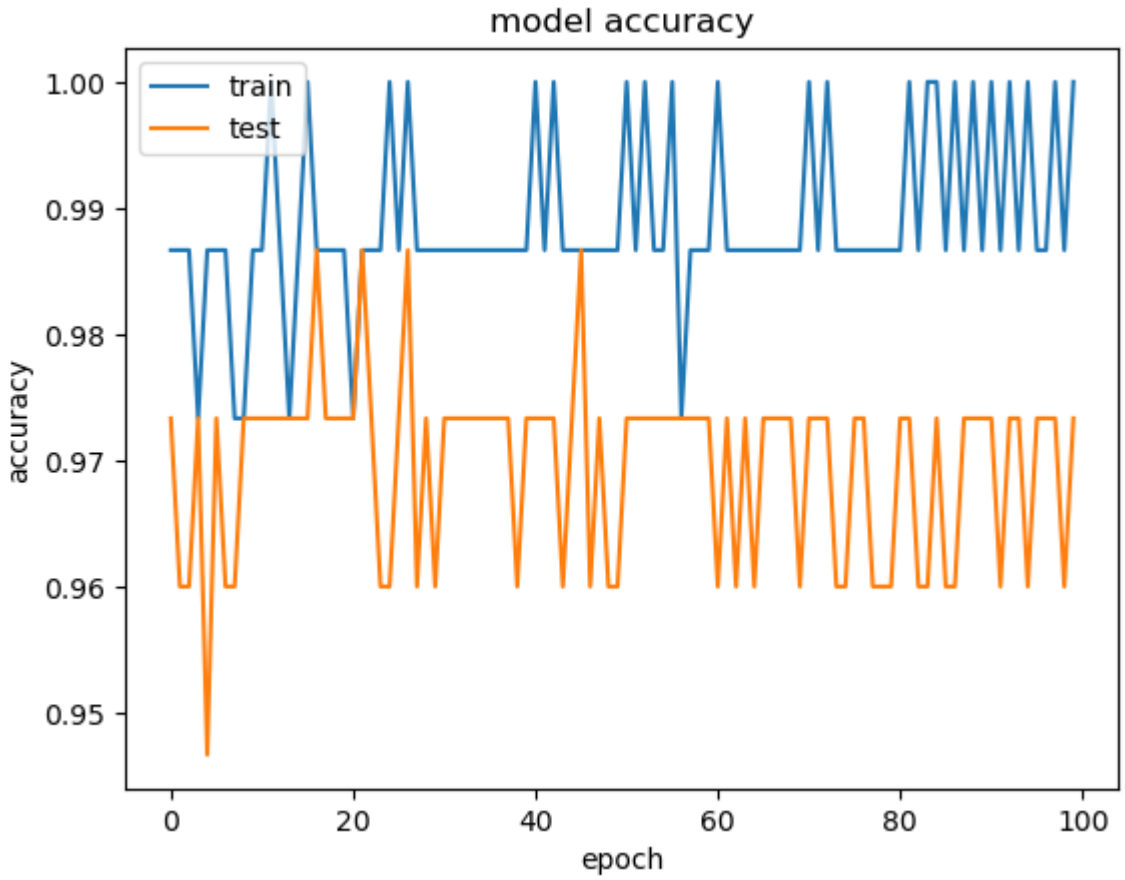
13) Visualize the loss on training and test data.

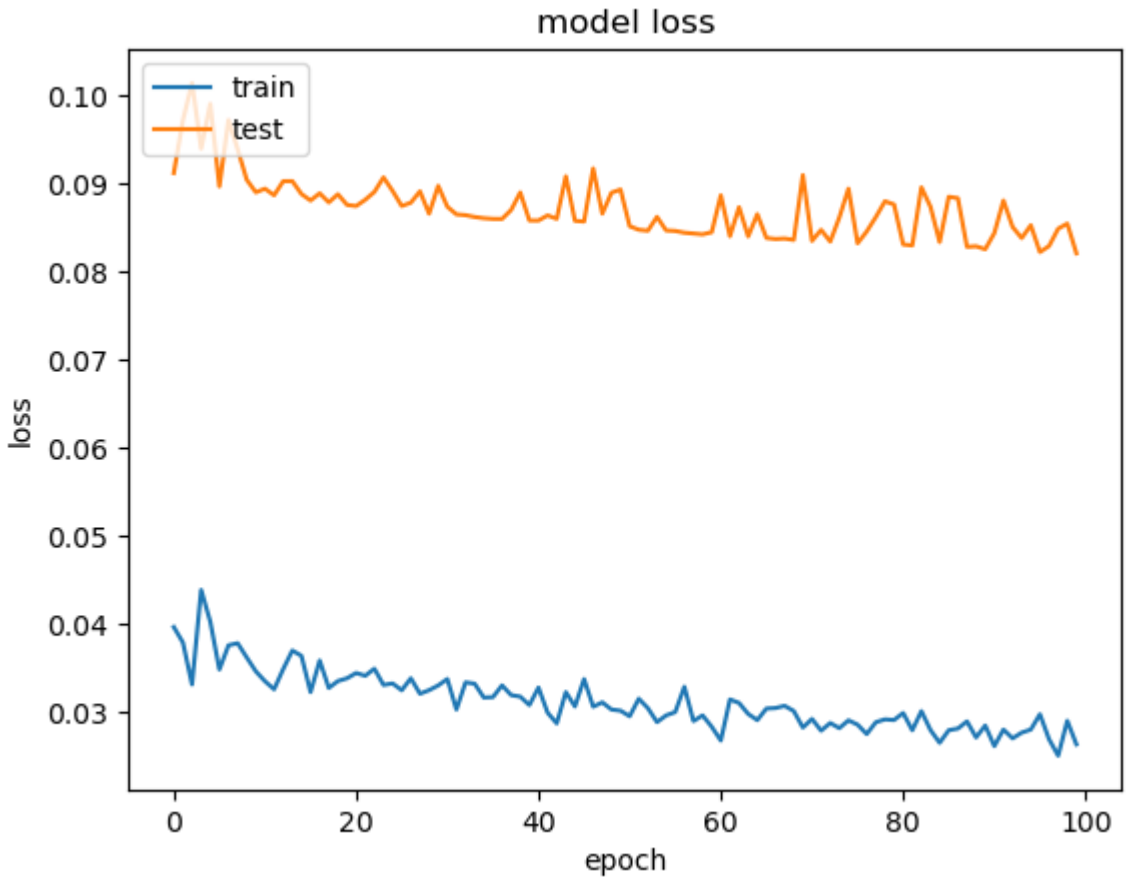
In [65]:
```
print(history.history.keys())

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





15) Try to change some parameters with the model.

In [69]:
```
# creating a new model with different parameters
inputs = Input(shape=(4,))
layer = Dense(16, activation='relu')(inputs)
layer = Dense(8, activation='relu')(layer)
outputs = Dense(3, activation='softmax')(layer)
simple_iris_model_2 = Model(inputs=inputs, outputs=outputs, name="simple_iris_model_2"

# compiling the model
simple_iris_model_2.compile(
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

# creating the history for the new model
history_2 = simple_iris_model_2.fit(
    X_train, y_train_bin,
    epochs = 100,
    batch_size = 1,
    verbose = 1,
    validation_data = (X_test, y_test_bin)
)
```

```
Epoch 1/100
75/75 [==============================] - 1s 5ms/step - loss: 1.1218 - accuracy: 0.373
3 - val_loss: 1.1136 - val_accuracy: 0.2933
Epoch 2/100
75/75 [==============================] - 0s 2ms/step - loss: 1.0899 - accuracy: 0.373
3 - val_loss: 1.1042 - val_accuracy: 0.2933
Epoch 3/100
75/75 [==============================] - 0s 2ms/step - loss: 1.0769 - accuracy: 0.373
3 - val_loss: 1.0921 - val_accuracy: 0.2933
Epoch 4/100
75/75 [==============================] - 0s 2ms/step - loss: 1.0524 - accuracy: 0.373
3 - val_loss: 1.0598 - val_accuracy: 0.2933
Epoch 5/100
75/75 [==============================] - 0s 3ms/step - loss: 1.0304 - accuracy: 0.373
3 - val_loss: 1.0284 - val_accuracy: 0.3467
Epoch 6/100
75/75 [==============================] - 0s 3ms/step - loss: 1.0000 - accuracy: 0.413
3 - val_loss: 1.0012 - val_accuracy: 0.3067
Epoch 7/100
75/75 [==============================] - 0s 2ms/step - loss: 0.9620 - accuracy: 0.520
0 - val_loss: 0.9620 - val_accuracy: 0.2933
Epoch 8/100
75/75 [==============================] - 0s 2ms/step - loss: 0.9260 - accuracy: 0.440
0 - val_loss: 0.9213 - val_accuracy: 0.3467
Epoch 9/100
75/75 [==============================] - 0s 2ms/step - loss: 0.8793 - accuracy: 0.520
0 - val_loss: 0.8665 - val_accuracy: 0.4933
Epoch 10/100
75/75 [==============================] - 0s 2ms/step - loss: 0.8325 - accuracy: 0.720
0 - val_loss: 0.8066 - val_accuracy: 0.6667
Epoch 11/100
75/75 [==============================] - 0s 2ms/step - loss: 0.7772 - accuracy: 0.800
0 - val_loss: 0.7710 - val_accuracy: 0.5867
Epoch 12/100
75/75 [==============================] - 0s 2ms/step - loss: 0.7195 - accuracy: 0.746
7 - val_loss: 0.6915 - val_accuracy: 0.9333
Epoch 13/100
75/75 [==============================] - 0s 2ms/step - loss: 0.6498 - accuracy: 0.893
3 - val_loss: 0.6373 - val_accuracy: 0.8800
Epoch 14/100
75/75 [==============================] - 0s 2ms/step - loss: 0.5797 - accuracy: 0.960
0 - val_loss: 0.5816 - val_accuracy: 0.8400
Epoch 15/100
75/75 [==============================] - 0s 3ms/step - loss: 0.5151 - accuracy: 0.893
3 - val_loss: 0.5049 - val_accuracy: 0.9200
Epoch 16/100
75/75 [==============================] - 0s 2ms/step - loss: 0.4501 - accuracy: 0.946
7 - val_loss: 0.4484 - val_accuracy: 0.9733
Epoch 17/100
75/75 [==============================] - 0s 2ms/step - loss: 0.4006 - accuracy: 0.960
0 - val_loss: 0.4155 - val_accuracy: 0.9733
Epoch 18/100
75/75 [==============================] - 0s 2ms/step - loss: 0.3724 - accuracy: 0.906
7 - val_loss: 0.3879 - val_accuracy: 0.9733
Epoch 19/100
75/75 [==============================] - 0s 2ms/step - loss: 0.3254 - accuracy: 0.960
0 - val_loss: 0.3377 - val_accuracy: 0.8533
Epoch 20/100
75/75 [==============================] - 0s 3ms/step - loss: 0.2864 - accuracy: 1.000
0 - val_loss: 0.3565 - val_accuracy: 0.8667
```

```
Epoch 21/100
75/75 [==============================] - 0s 3ms/step - loss: 0.2660 - accuracy: 0.960
0 - val_loss: 0.3553 - val_accuracy: 0.8667
Epoch 22/100
75/75 [==============================] - 0s 3ms/step - loss: 0.2450 - accuracy: 0.973
3 - val_loss: 0.2690 - val_accuracy: 0.9467
Epoch 23/100
75/75 [==============================] - 0s 3ms/step - loss: 0.2261 - accuracy: 0.986
7 - val_loss: 0.2587 - val_accuracy: 0.9600
Epoch 24/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2025 - accuracy: 0.986
7 - val_loss: 0.2353 - val_accuracy: 0.9467
Epoch 25/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1944 - accuracy: 0.973
3 - val_loss: 0.2310 - val_accuracy: 0.9733
Epoch 26/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1812 - accuracy: 0.986
7 - val_loss: 0.2195 - val_accuracy: 0.9733
Epoch 27/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1489 - accuracy: 1.000
0 - val_loss: 0.2622 - val_accuracy: 0.8933
Epoch 28/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1853 - accuracy: 0.960
0 - val_loss: 0.2061 - val_accuracy: 0.9467
Epoch 29/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1767 - accuracy: 0.946
7 - val_loss: 0.1908 - val_accuracy: 0.9600
Epoch 30/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1465 - accuracy: 0.986
7 - val_loss: 0.1847 - val_accuracy: 0.9733
Epoch 31/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1567 - accuracy: 0.973
3 - val_loss: 0.1833 - val_accuracy: 0.9733
Epoch 32/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1395 - accuracy: 0.973
3 - val_loss: 0.1810 - val_accuracy: 0.9600
Epoch 33/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1306 - accuracy: 0.986
7 - val_loss: 0.1716 - val_accuracy: 0.9467
Epoch 34/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1161 - accuracy: 0.973
3 - val_loss: 0.1958 - val_accuracy: 0.9200
Epoch 35/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1198 - accuracy: 0.986
7 - val_loss: 0.1712 - val_accuracy: 0.9467
Epoch 36/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1149 - accuracy: 0.986
7 - val_loss: 0.1541 - val_accuracy: 0.9600
Epoch 37/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1112 - accuracy: 0.973
3 - val_loss: 0.1553 - val_accuracy: 0.9733
Epoch 38/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1030 - accuracy: 0.973
3 - val_loss: 0.1581 - val_accuracy: 0.9467
Epoch 39/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1129 - accuracy: 0.986
7 - val_loss: 0.1466 - val_accuracy: 0.9600
Epoch 40/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0976 - accuracy: 0.986
7 - val_loss: 0.1689 - val_accuracy: 0.9200
```

```
Epoch 41/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0962 - accuracy: 0.986
7 - val_loss: 0.1622 - val_accuracy: 0.9200
Epoch 42/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0973 - accuracy: 0.960
0 - val_loss: 0.1365 - val_accuracy: 0.9733
Epoch 43/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0945 - accuracy: 0.973
3 - val_loss: 0.1367 - val_accuracy: 0.9600
Epoch 44/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0877 - accuracy: 0.986
7 - val_loss: 0.1348 - val_accuracy: 0.9600
Epoch 45/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0862 - accuracy: 0.986
7 - val_loss: 0.1352 - val_accuracy: 0.9733
Epoch 46/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0848 - accuracy: 0.973
3 - val_loss: 0.1348 - val_accuracy: 0.9467
Epoch 47/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0765 - accuracy: 1.000
0 - val_loss: 0.1838 - val_accuracy: 0.9067
Epoch 48/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0904 - accuracy: 0.960
0 - val_loss: 0.1227 - val_accuracy: 0.9733
Epoch 49/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0813 - accuracy: 0.986
7 - val_loss: 0.1339 - val_accuracy: 0.9600
Epoch 50/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0806 - accuracy: 0.986
7 - val_loss: 0.1502 - val_accuracy: 0.9200
Epoch 51/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 1.000
0 - val_loss: 0.1707 - val_accuracy: 0.9067
Epoch 52/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0716 - accuracy: 0.986
7 - val_loss: 0.1161 - val_accuracy: 0.9733
Epoch 53/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0766 - accuracy: 0.973
3 - val_loss: 0.1239 - val_accuracy: 0.9733
Epoch 54/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0715 - accuracy: 0.986
7 - val_loss: 0.1344 - val_accuracy: 0.9333
Epoch 55/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0771 - accuracy: 0.960
0 - val_loss: 0.1175 - val_accuracy: 0.9733
Epoch 56/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0847 - accuracy: 0.973
3 - val_loss: 0.1197 - val_accuracy: 0.9600
Epoch 57/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0627 - accuracy: 0.986
7 - val_loss: 0.1350 - val_accuracy: 0.9200
Epoch 58/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0728 - accuracy: 0.973
3 - val_loss: 0.1225 - val_accuracy: 0.9600
Epoch 59/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0698 - accuracy: 0.973
3 - val_loss: 0.1078 - val_accuracy: 0.9733
Epoch 60/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0664 - accuracy: 0.973
3 - val_loss: 0.1087 - val_accuracy: 0.9600
```

```
Epoch 61/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0710 - accuracy: 0.973
3 - val_loss: 0.1065 - val_accuracy: 0.9733
Epoch 62/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0592 - accuracy: 0.973
3 - val_loss: 0.1331 - val_accuracy: 0.9200
Epoch 63/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0640 - accuracy: 0.973
3 - val_loss: 0.1054 - val_accuracy: 0.9600
Epoch 64/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0629 - accuracy: 0.986
7 - val_loss: 0.1108 - val_accuracy: 0.9733
Epoch 65/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0605 - accuracy: 0.973
3 - val_loss: 0.1030 - val_accuracy: 0.9733
Epoch 66/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0645 - accuracy: 0.986
7 - val_loss: 0.1203 - val_accuracy: 0.9467
Epoch 67/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0578 - accuracy: 0.986
7 - val_loss: 0.1229 - val_accuracy: 0.9333
Epoch 68/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0485 - accuracy: 0.986
7 - val_loss: 0.2110 - val_accuracy: 0.9333
Epoch 69/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0716 - accuracy: 0.973
3 - val_loss: 0.1002 - val_accuracy: 0.9733
Epoch 70/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0569 - accuracy: 0.973
3 - val_loss: 0.1391 - val_accuracy: 0.9467
Epoch 71/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0579 - accuracy: 0.986
7 - val_loss: 0.1045 - val_accuracy: 0.9600
Epoch 72/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0680 - accuracy: 0.973
3 - val_loss: 0.0997 - val_accuracy: 0.9600
Epoch 73/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0540 - accuracy: 0.986
7 - val_loss: 0.0995 - val_accuracy: 0.9733
Epoch 74/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0548 - accuracy: 0.986
7 - val_loss: 0.1038 - val_accuracy: 0.9600
Epoch 75/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0503 - accuracy: 0.986
7 - val_loss: 0.1014 - val_accuracy: 0.9733
Epoch 76/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0523 - accuracy: 0.986
7 - val_loss: 0.0984 - val_accuracy: 0.9600
Epoch 77/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0610 - accuracy: 0.960
0 - val_loss: 0.0969 - val_accuracy: 0.9733
Epoch 78/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0543 - accuracy: 0.986
7 - val_loss: 0.0952 - val_accuracy: 0.9733
Epoch 79/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0664 - accuracy: 0.973
3 - val_loss: 0.0999 - val_accuracy: 0.9600
Epoch 80/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0458 - accuracy: 1.000
0 - val_loss: 0.1083 - val_accuracy: 0.9600
```

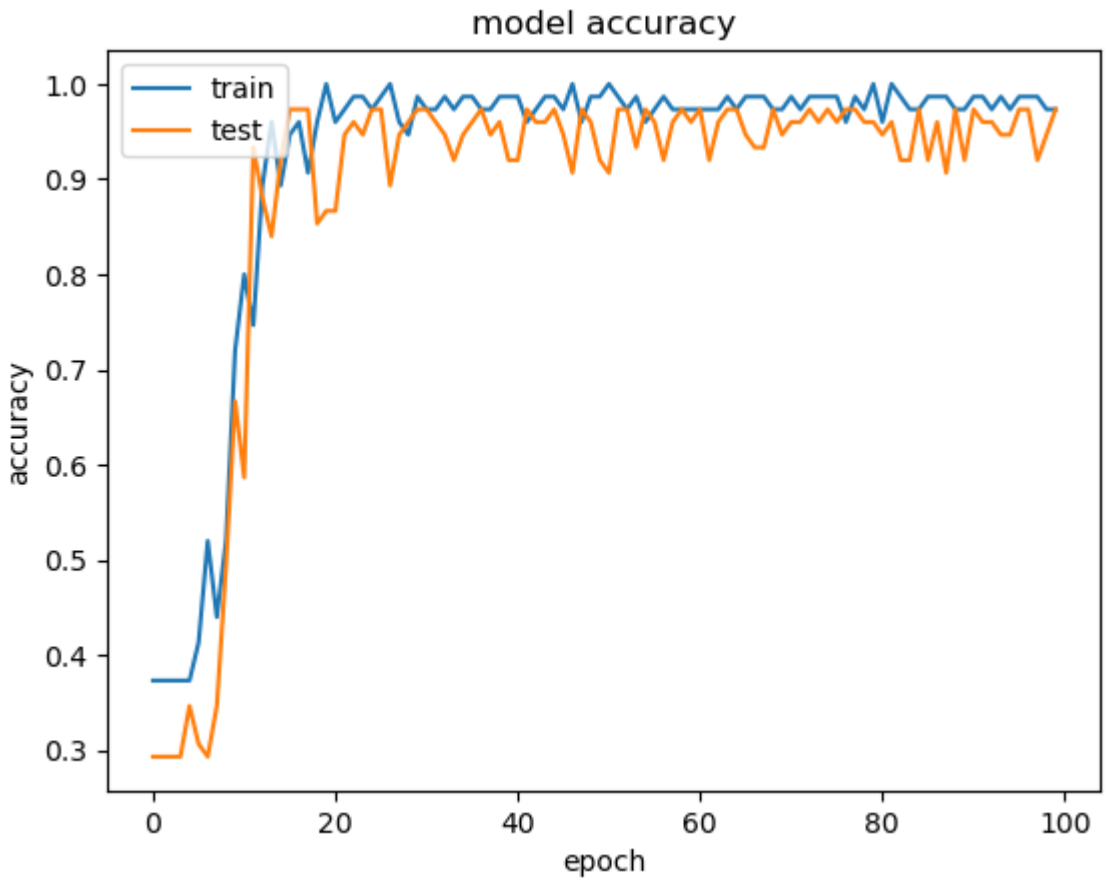
```
Epoch 81/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0682 - accuracy: 0.960
0 - val_loss: 0.1212 - val_accuracy: 0.9467
Epoch 82/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0458 - accuracy: 1.000
0 - val_loss: 0.1045 - val_accuracy: 0.9600
Epoch 83/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0479 - accuracy: 0.986
7 - val_loss: 0.1236 - val_accuracy: 0.9200
Epoch 84/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0683 - accuracy: 0.973
3 - val_loss: 0.1870 - val_accuracy: 0.9200
Epoch 85/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0646 - accuracy: 0.973
3 - val_loss: 0.0959 - val_accuracy: 0.9733
Epoch 86/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0472 - accuracy: 0.986
7 - val_loss: 0.1371 - val_accuracy: 0.9200
Epoch 87/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0479 - accuracy: 0.986
7 - val_loss: 0.0987 - val_accuracy: 0.9600
Epoch 88/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0410 - accuracy: 0.986
7 - val_loss: 0.1473 - val_accuracy: 0.9067
Epoch 89/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0596 - accuracy: 0.973
3 - val_loss: 0.0950 - val_accuracy: 0.9733
Epoch 90/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0601 - accuracy: 0.973
3 - val_loss: 0.1239 - val_accuracy: 0.9200
Epoch 91/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0570 - accuracy: 0.986
7 - val_loss: 0.0914 - val_accuracy: 0.9733
Epoch 92/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0442 - accuracy: 0.986
7 - val_loss: 0.0987 - val_accuracy: 0.9600
Epoch 93/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0516 - accuracy: 0.973
3 - val_loss: 0.0946 - val_accuracy: 0.9600
Epoch 94/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0446 - accuracy: 0.986
7 - val_loss: 0.1068 - val_accuracy: 0.9467
Epoch 95/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0695 - accuracy: 0.973
3 - val_loss: 0.1083 - val_accuracy: 0.9467
Epoch 96/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0498 - accuracy: 0.986
7 - val_loss: 0.0893 - val_accuracy: 0.9733
Epoch 97/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0452 - accuracy: 0.986
7 - val_loss: 0.0892 - val_accuracy: 0.9733
Epoch 98/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0517 - accuracy: 0.986
7 - val_loss: 0.1174 - val_accuracy: 0.9200
Epoch 99/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0569 - accuracy: 0.973
3 - val_loss: 0.1851 - val_accuracy: 0.9467
Epoch 100/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0686 - accuracy: 0.973
3 - val_loss: 0.0904 - val_accuracy: 0.9733
```

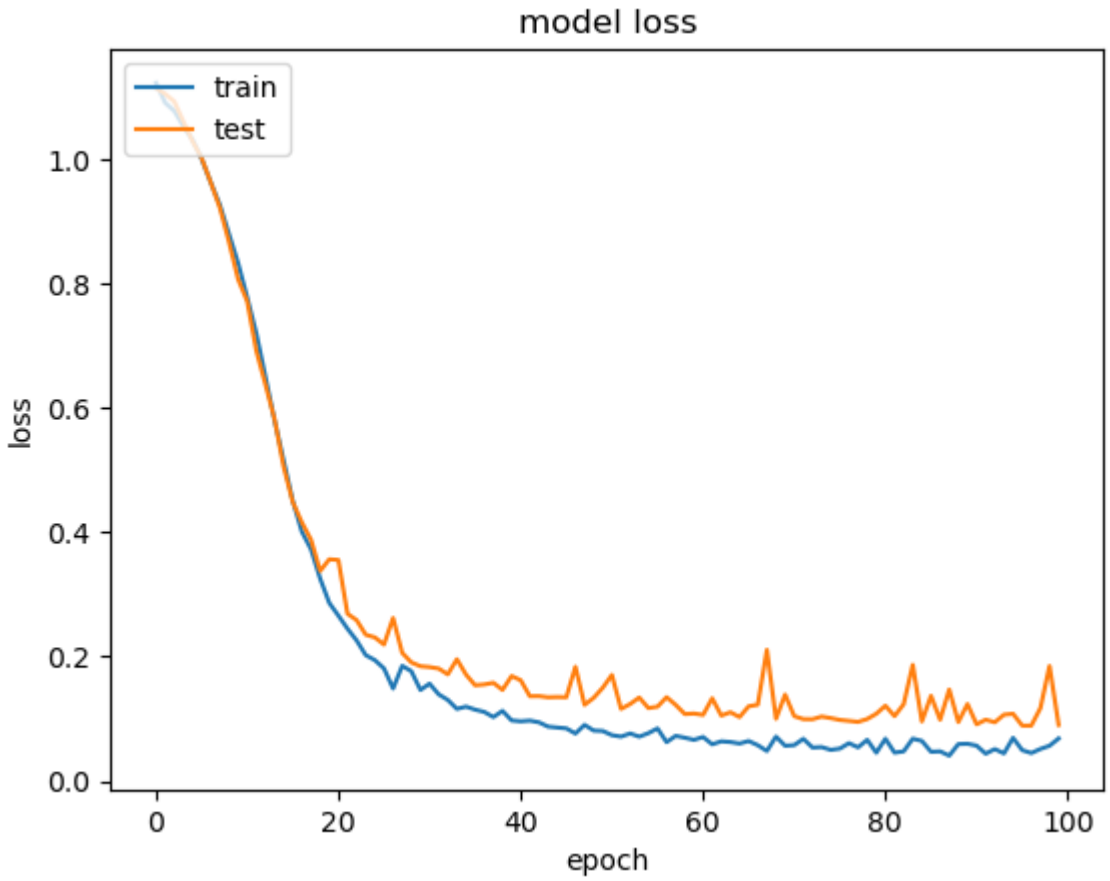
In [71]:
```
# creating visualizations for the new model
print(history_2.history.keys())

# summarize history for accuracy
plt.plot(history_2.history['accuracy'])
plt.plot(history_2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





16) This time, copy the same model, but use an SGD optimizer. Also, instantiate your optimizer.

In [73]:
```
# assisted by ChatGPT
inputs = Input(shape=(4,))
layer = Dense(12, activation='sigmoid')(inputs)
outputs = Dense(3, activation='softmax')(layer)

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

model_sgd = Model(inputs=inputs, outputs=outputs, name="simple_iris_model_sgd")
model_sgd.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy']

# creating the history for the new model
history_sgd = model_sgd.fit(
    X_train, y_train_bin,
    epochs = 100,
    batch_size = 1,
    verbose = 1,
    validation_data = (X_test, y_test_bin)
)
```

```
Epoch 1/100
75/75 [==============================] - 1s 4ms/step - loss: 0.9495 - accuracy: 0.533
3 - val_loss: 0.8667 - val_accuracy: 0.6000
Epoch 2/100
75/75 [==============================] - 0s 2ms/step - loss: 0.6069 - accuracy: 0.733
3 - val_loss: 0.6678 - val_accuracy: 0.6000
Epoch 3/100
75/75 [==============================] - 0s 2ms/step - loss: 0.4927 - accuracy: 0.786
7 - val_loss: 0.4495 - val_accuracy: 0.9467
Epoch 4/100
75/75 [==============================] - 0s 2ms/step - loss: 0.4040 - accuracy: 0.853
3 - val_loss: 0.3980 - val_accuracy: 0.9733
Epoch 5/100
75/75 [==============================] - 0s 2ms/step - loss: 0.3845 - accuracy: 0.840
0 - val_loss: 0.6064 - val_accuracy: 0.6000
Epoch 6/100
75/75 [==============================] - 0s 2ms/step - loss: 0.3478 - accuracy: 0.826
7 - val_loss: 0.6174 - val_accuracy: 0.6133
Epoch 7/100
75/75 [==============================] - 0s 2ms/step - loss: 0.3101 - accuracy: 0.893
3 - val_loss: 0.3030 - val_accuracy: 0.9600
Epoch 8/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2600 - accuracy: 0.933
3 - val_loss: 0.3677 - val_accuracy: 0.8400
Epoch 9/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2570 - accuracy: 0.893
3 - val_loss: 0.2810 - val_accuracy: 0.8933
Epoch 10/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2463 - accuracy: 0.880
0 - val_loss: 0.2615 - val_accuracy: 0.9067
Epoch 11/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2470 - accuracy: 0.906
7 - val_loss: 0.5006 - val_accuracy: 0.6533
Epoch 12/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2125 - accuracy: 0.946
7 - val_loss: 0.2548 - val_accuracy: 0.8800
Epoch 13/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1813 - accuracy: 0.920
0 - val_loss: 0.2101 - val_accuracy: 0.9333
Epoch 14/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1526 - accuracy: 0.960
0 - val_loss: 0.2949 - val_accuracy: 0.8800
Epoch 15/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2034 - accuracy: 0.933
3 - val_loss: 0.2066 - val_accuracy: 0.9200
Epoch 16/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2158 - accuracy: 0.893
3 - val_loss: 0.1918 - val_accuracy: 0.9200
Epoch 17/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1239 - accuracy: 0.973
3 - val_loss: 0.5991 - val_accuracy: 0.6800
Epoch 18/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2962 - accuracy: 0.840
0 - val_loss: 0.1769 - val_accuracy: 0.9467
Epoch 19/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1195 - accuracy: 0.946
7 - val_loss: 0.1821 - val_accuracy: 0.9467
Epoch 20/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1737 - accuracy: 0.946
7 - val_loss: 0.1510 - val_accuracy: 0.9467
```

```
Epoch 21/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1194 - accuracy: 0.946
7 - val_loss: 0.1415 - val_accuracy: 0.9600
Epoch 22/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0975 - accuracy: 0.986
7 - val_loss: 0.1648 - val_accuracy: 0.9467
Epoch 23/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0985 - accuracy: 0.973
3 - val_loss: 0.1253 - val_accuracy: 0.9733
Epoch 24/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1648 - accuracy: 0.946
7 - val_loss: 0.4607 - val_accuracy: 0.8000
Epoch 25/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2344 - accuracy: 0.866
7 - val_loss: 0.3608 - val_accuracy: 0.8267
Epoch 26/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1218 - accuracy: 0.946
7 - val_loss: 0.1271 - val_accuracy: 0.9600
Epoch 27/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0996 - accuracy: 0.973
3 - val_loss: 0.5545 - val_accuracy: 0.7867
Epoch 28/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1034 - accuracy: 0.973
3 - val_loss: 0.1338 - val_accuracy: 0.9467
Epoch 29/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0765 - accuracy: 0.986
7 - val_loss: 0.1289 - val_accuracy: 0.9333
Epoch 30/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0791 - accuracy: 0.973
3 - val_loss: 0.1568 - val_accuracy: 0.9067
Epoch 31/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1807 - accuracy: 0.906
7 - val_loss: 0.2018 - val_accuracy: 0.9067
Epoch 32/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0607 - accuracy: 0.986
7 - val_loss: 0.3083 - val_accuracy: 0.8533
Epoch 33/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0926 - accuracy: 0.960
0 - val_loss: 0.4449 - val_accuracy: 0.8400
Epoch 34/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0585 - accuracy: 1.000
0 - val_loss: 0.2208 - val_accuracy: 0.8933
Epoch 35/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0573 - accuracy: 0.986
7 - val_loss: 0.4020 - val_accuracy: 0.8533
Epoch 36/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2630 - accuracy: 0.906
7 - val_loss: 0.1719 - val_accuracy: 0.9467
Epoch 37/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1768 - accuracy: 0.960
0 - val_loss: 0.1537 - val_accuracy: 0.9467
Epoch 38/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0832 - accuracy: 0.986
7 - val_loss: 0.1390 - val_accuracy: 0.9200
Epoch 39/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0641 - accuracy: 0.986
7 - val_loss: 0.1280 - val_accuracy: 0.9600
Epoch 40/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0900 - accuracy: 0.960
0 - val_loss: 0.1072 - val_accuracy: 0.9600
```

```
Epoch 41/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0774 - accuracy: 0.973
3 - val_loss: 0.1043 - val_accuracy: 0.9733
Epoch 42/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1000 - accuracy: 0.960
0 - val_loss: 0.1619 - val_accuracy: 0.9467
Epoch 43/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0878 - accuracy: 0.973
3 - val_loss: 0.1010 - val_accuracy: 0.9733
Epoch 44/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2433 - accuracy: 0.880
0 - val_loss: 0.5688 - val_accuracy: 0.7733
Epoch 45/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1699 - accuracy: 0.906
7 - val_loss: 0.2086 - val_accuracy: 0.9067
Epoch 46/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1216 - accuracy: 0.946
7 - val_loss: 0.3084 - val_accuracy: 0.8533
Epoch 47/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1071 - accuracy: 0.933
3 - val_loss: 0.2546 - val_accuracy: 0.8933
Epoch 48/100
75/75 [==============================] - 0s 3ms/step - loss: 0.2263 - accuracy: 0.920
0 - val_loss: 0.1329 - val_accuracy: 0.9600
Epoch 49/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1142 - accuracy: 0.920
0 - val_loss: 0.1669 - val_accuracy: 0.9067
Epoch 50/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0562 - accuracy: 0.986
7 - val_loss: 0.0999 - val_accuracy: 0.9600
Epoch 51/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1882 - accuracy: 0.906
7 - val_loss: 0.1157 - val_accuracy: 0.9600
Epoch 52/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0789 - accuracy: 0.973
3 - val_loss: 0.1120 - val_accuracy: 0.9600
Epoch 53/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0949 - accuracy: 0.973
3 - val_loss: 0.1803 - val_accuracy: 0.9467
Epoch 54/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0803 - accuracy: 0.960
0 - val_loss: 0.1076 - val_accuracy: 0.9600
Epoch 55/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0490 - accuracy: 0.986
7 - val_loss: 0.3210 - val_accuracy: 0.8933
Epoch 56/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1228 - accuracy: 0.946
7 - val_loss: 0.1056 - val_accuracy: 0.9733
Epoch 57/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0642 - accuracy: 0.973
3 - val_loss: 0.0966 - val_accuracy: 0.9733
Epoch 58/100
75/75 [==============================] - 0s 3ms/step - loss: 0.1963 - accuracy: 0.933
3 - val_loss: 0.1129 - val_accuracy: 0.9733
Epoch 59/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1427 - accuracy: 0.946
7 - val_loss: 0.1377 - val_accuracy: 0.9333
Epoch 60/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0427 - accuracy: 0.986
7 - val_loss: 0.1873 - val_accuracy: 0.9067
```

```
Epoch 61/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0509 - accuracy: 0.986
7 - val_loss: 0.1892 - val_accuracy: 0.9067
Epoch 62/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1785 - accuracy: 0.933
3 - val_loss: 0.1081 - val_accuracy: 0.9600
Epoch 63/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0602 - accuracy: 0.960
0 - val_loss: 0.2974 - val_accuracy: 0.8667
Epoch 64/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1825 - accuracy: 0.946
7 - val_loss: 0.1285 - val_accuracy: 0.9600
Epoch 65/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1479 - accuracy: 0.920
0 - val_loss: 0.3908 - val_accuracy: 0.8667
Epoch 66/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0643 - accuracy: 0.986
7 - val_loss: 0.2866 - val_accuracy: 0.8800
Epoch 67/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0713 - accuracy: 0.946
7 - val_loss: 0.0989 - val_accuracy: 0.9600
Epoch 68/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0462 - accuracy: 0.986
7 - val_loss: 0.1015 - val_accuracy: 0.9733
Epoch 69/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1157 - accuracy: 0.946
7 - val_loss: 0.3274 - val_accuracy: 0.8933
Epoch 70/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0643 - accuracy: 0.986
7 - val_loss: 0.1071 - val_accuracy: 0.9600
Epoch 71/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1645 - accuracy: 0.946
7 - val_loss: 0.3714 - val_accuracy: 0.8800
Epoch 72/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2802 - accuracy: 0.880
0 - val_loss: 0.3531 - val_accuracy: 0.8267
Epoch 73/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1734 - accuracy: 0.920
0 - val_loss: 0.1064 - val_accuracy: 0.9600
Epoch 74/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0626 - accuracy: 0.986
7 - val_loss: 0.1037 - val_accuracy: 0.9733
Epoch 75/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2212 - accuracy: 0.920
0 - val_loss: 0.1751 - val_accuracy: 0.9200
Epoch 76/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1439 - accuracy: 0.933
3 - val_loss: 0.1587 - val_accuracy: 0.9333
Epoch 77/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0869 - accuracy: 0.973
3 - val_loss: 0.1060 - val_accuracy: 0.9600
Epoch 78/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1188 - accuracy: 0.960
0 - val_loss: 0.1063 - val_accuracy: 0.9733
Epoch 79/100
75/75 [==============================] - 0s 2ms/step - loss: 0.2419 - accuracy: 0.893
3 - val_loss: 0.2504 - val_accuracy: 0.8667
Epoch 80/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1313 - accuracy: 0.933
3 - val_loss: 0.1822 - val_accuracy: 0.9067
```

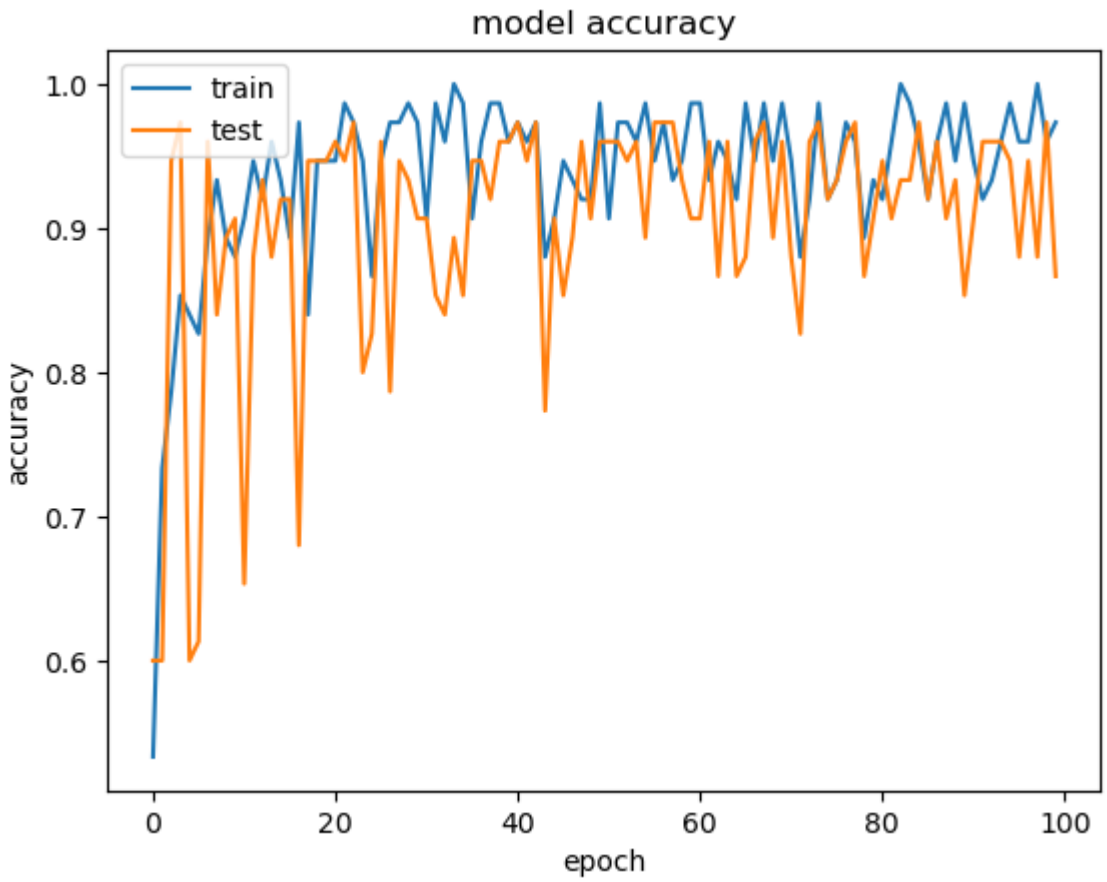
```
Epoch 81/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1909 - accuracy: 0.920
0 - val_loss: 0.1261 - val_accuracy: 0.9467
Epoch 82/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0960 - accuracy: 0.960
0 - val_loss: 0.1859 - val_accuracy: 0.9067
Epoch 83/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0456 - accuracy: 1.000
0 - val_loss: 0.1239 - val_accuracy: 0.9333
Epoch 84/100
75/75 [==============================] - 0s 3ms/step - loss: 0.0443 - accuracy: 0.986
7 - val_loss: 0.1206 - val_accuracy: 0.9333
Epoch 85/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0685 - accuracy: 0.960
0 - val_loss: 0.1008 - val_accuracy: 0.9733
Epoch 86/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1851 - accuracy: 0.920
0 - val_loss: 0.1374 - val_accuracy: 0.9200
Epoch 87/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0955 - accuracy: 0.960
0 - val_loss: 0.1082 - val_accuracy: 0.9600
Epoch 88/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0805 - accuracy: 0.986
7 - val_loss: 0.2001 - val_accuracy: 0.9067
Epoch 89/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0845 - accuracy: 0.946
7 - val_loss: 0.1448 - val_accuracy: 0.9333
Epoch 90/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0542 - accuracy: 0.986
7 - val_loss: 0.5035 - val_accuracy: 0.8533
Epoch 91/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1189 - accuracy: 0.946
7 - val_loss: 0.1793 - val_accuracy: 0.9067
Epoch 92/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1930 - accuracy: 0.920
0 - val_loss: 0.1038 - val_accuracy: 0.9600
Epoch 93/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1459 - accuracy: 0.933
3 - val_loss: 0.1058 - val_accuracy: 0.9600
Epoch 94/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0854 - accuracy: 0.960
0 - val_loss: 0.1027 - val_accuracy: 0.9600
Epoch 95/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0593 - accuracy: 0.986
7 - val_loss: 0.1509 - val_accuracy: 0.9467
Epoch 96/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.960
0 - val_loss: 0.2608 - val_accuracy: 0.8800
Epoch 97/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0907 - accuracy: 0.960
0 - val_loss: 0.1299 - val_accuracy: 0.9467
Epoch 98/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0337 - accuracy: 1.000
0 - val_loss: 0.3137 - val_accuracy: 0.8800
Epoch 99/100
75/75 [==============================] - 0s 2ms/step - loss: 0.1602 - accuracy: 0.960
0 - val_loss: 0.1007 - val_accuracy: 0.9733
Epoch 100/100
75/75 [==============================] - 0s 2ms/step - loss: 0.0482 - accuracy: 0.973
3 - val_loss: 0.3192 - val_accuracy: 0.8667
```

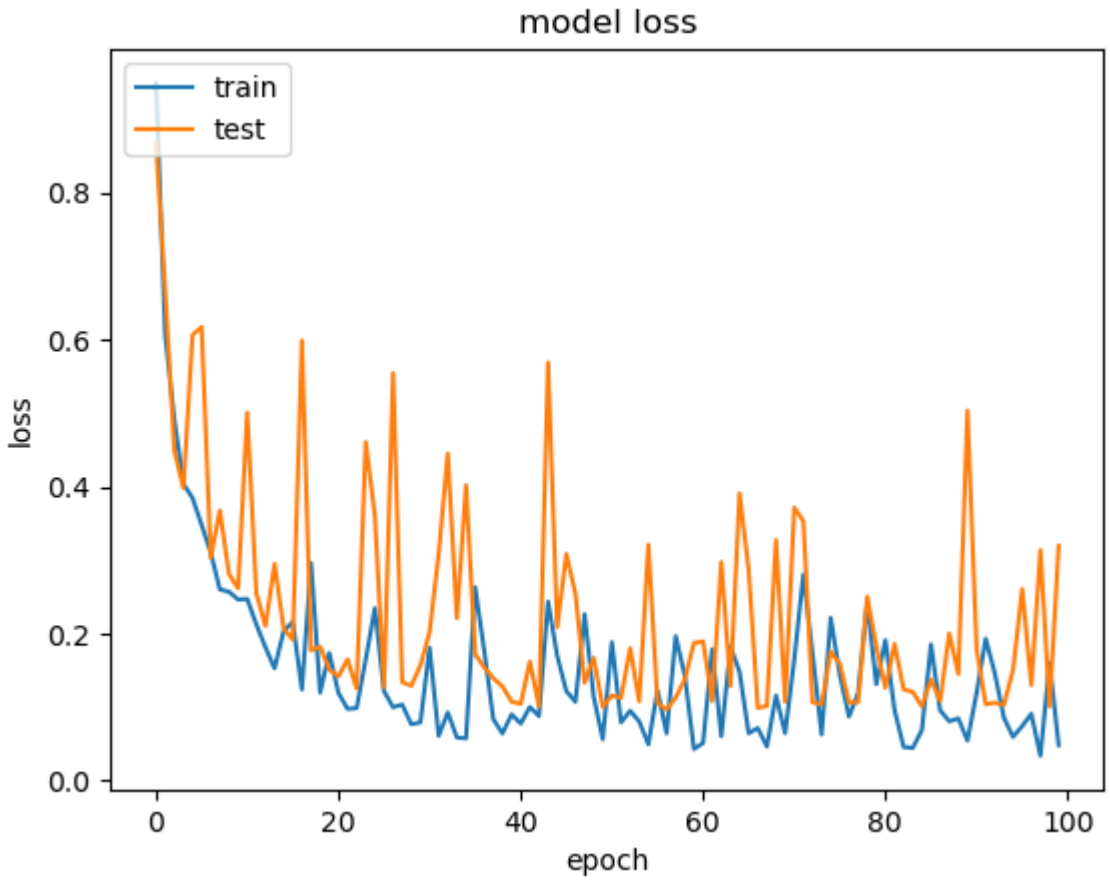
In [74]:
```
# creating visualizations for the new model
print(history_sgd.history.keys())

# summarize history for accuracy
plt.plot(history_sgd.history['accuracy'])
plt.plot(history_sgd.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history_sgd.history['loss'])
plt.plot(history_sgd.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





The model accuracy and model loss plots of the sgd optimization model were much better and more consistent compared to that of the original model that we had generated. But, the relu optimized model seems to be the most consistent and best fitting model.

17) Use your model to predict the classes for the test data (using the function predict on the model object itself) and store the results as y_pred.

In [100…
```
y_pred = model_sgd.predict(X_test)
```

```
3/3 [==============================] - 0s 997us/step
```

18) Notice the structure of y_pred. Remember, softmax generates probabilistic output. So, turn this into a new variable called y_pred_class that predicts the actual class label.

In [101…
```
y_pred_class = np.argmax(y_pred, axis=1)
```

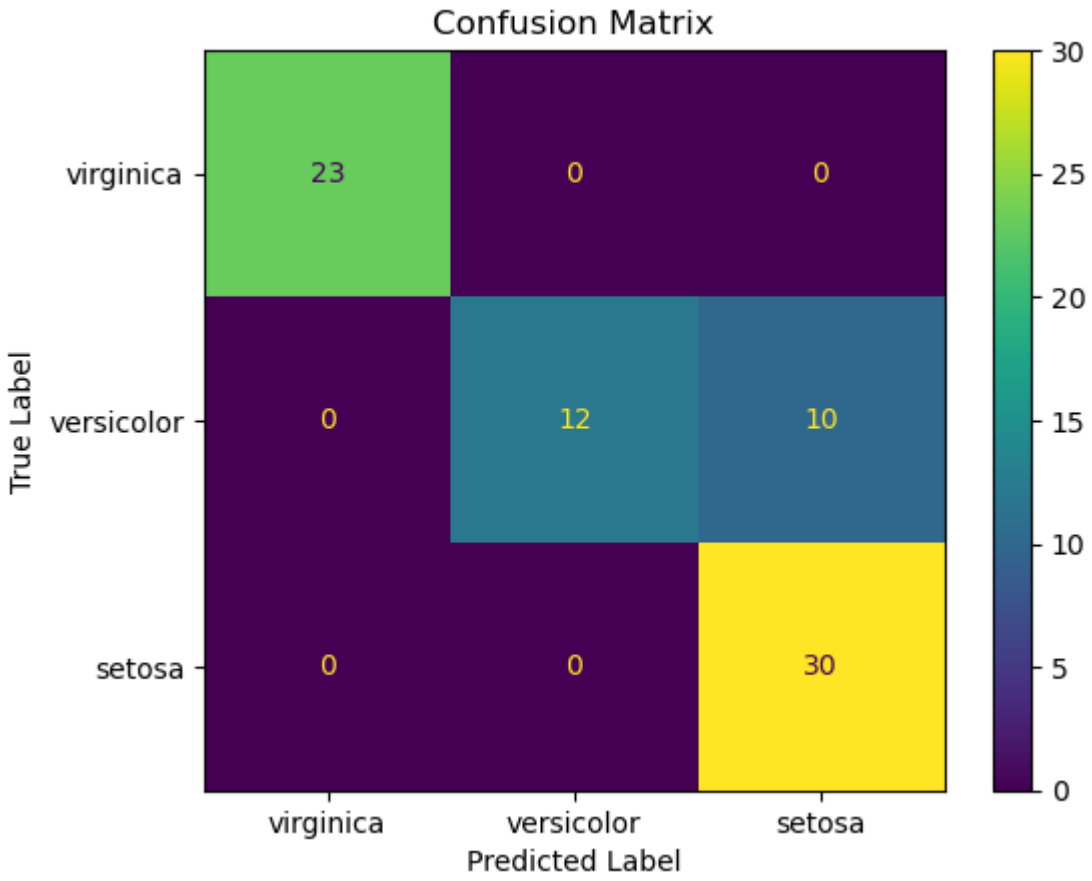
19) Finally, using your code from the lab on classification, output the confusion_matrix and the classification_report (from scikit-learn's metric package) to print out the complete performance results.

In [114…
```
# convert string labels to integers
le = LabelEncoder()
y_test_num = le.fit_transform(y_test)

# compute confusion matrix and classification report
cm = confusion_matrix(y_test_num, y_pred_class)
```

```
cm_disp = ConfusionMatrixDisplay(cm, display_labels=y.unique())
cm_disp.plot()
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

Out[114]:
```
Text(0, 0.5, 'True Label')
```



In [115…
```
cr = classification_report(y_test_num, y_pred_class)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       1.00      0.55      0.71        22
           2       0.75      1.00      0.86        30

    accuracy                           0.87        75
   macro avg       0.92      0.85      0.85        75
weighted avg       0.90      0.87      0.86        75
```

Exercise 3 – Wrapping keras in the scikit-learn framework.

20) Import the following class into your code. Then write a function called create_keras_model(). Copy all of your code that creates the Model instance and adds the layers into this function.

In [116…
```
from scikeras.wrappers import KerasClassifier
```

In [117…
```
def create_keras_model():
    inputs = Input(shape=(4,))
    layer = Dense(12, activation='sigmoid')(inputs)
    outputs = Dense(3,activation="softmax")(layer)
    model = Model(inputs=inputs,
        outputs=outputs,
        name="simple_iris_model_3")
    return model

# Create a instance of our optimizer
sgd = SGD(learning_rate=0.02,momentum=0.6)

# Create a wrapped keras model
clf = KerasClassifier(model=create_keras_model,
                      optimizer=sgd,
                      loss="categorical_crossentropy",
                      metrics=["accuracy"],
                      verbose=1, epochs=100, batch_size=1)
```

21) So, using the clf classifier above, use it just like you would any other classifier. Run the fit method on your classifier. Use X_train and the one hot encoded y_train data. Finally, use clf.predict() to generate class predictions on X_test. Store the results in y_pred just like you would with scikit-learn.

In [118…
```
clf.fit(X_train, y_train_bin)
clf
```

```
Epoch 1/100
75/75 [==============================] - 0s 885us/step - loss: 1.1875 - accuracy: 0.2
000
Epoch 2/100
75/75 [==============================] - 0s 865us/step - loss: 0.9619 - accuracy: 0.5
733
Epoch 3/100
75/75 [==============================] - 0s 899us/step - loss: 0.7636 - accuracy: 0.6
667
Epoch 4/100
75/75 [==============================] - 0s 912us/step - loss: 0.6021 - accuracy: 0.7
733
Epoch 5/100
75/75 [==============================] - 0s 973us/step - loss: 0.5456 - accuracy: 0.7
600
Epoch 6/100
75/75 [==============================] - 0s 959us/step - loss: 0.4916 - accuracy: 0.7
333
Epoch 7/100
75/75 [==============================] - 0s 1ms/step - loss: 0.4665 - accuracy: 0.800
0
Epoch 8/100
75/75 [==============================] - 0s 953us/step - loss: 0.4254 - accuracy: 0.8
133
Epoch 9/100
75/75 [==============================] - 0s 892us/step - loss: 0.4185 - accuracy: 0.8
267
Epoch 10/100
75/75 [==============================] - 0s 892us/step - loss: 0.4057 - accuracy: 0.8
533
Epoch 11/100
75/75 [==============================] - 0s 877us/step - loss: 0.3310 - accuracy: 0.8
800
Epoch 12/100
75/75 [==============================] - 0s 926us/step - loss: 0.3586 - accuracy: 0.8
800
Epoch 13/100
75/75 [==============================] - 0s 932us/step - loss: 0.3351 - accuracy: 0.9
200
Epoch 14/100
75/75 [==============================] - 0s 961us/step - loss: 0.3272 - accuracy: 0.9
067
Epoch 15/100
75/75 [==============================] - 0s 926us/step - loss: 0.2924 - accuracy: 0.9
200
Epoch 16/100
75/75 [==============================] - 0s 878us/step - loss: 0.2944 - accuracy: 0.9
067
Epoch 17/100
75/75 [==============================] - 0s 986us/step - loss: 0.2634 - accuracy: 0.9
333
Epoch 18/100
75/75 [==============================] - 0s 899us/step - loss: 0.2623 - accuracy: 0.9
467
Epoch 19/100
75/75 [==============================] - 0s 926us/step - loss: 0.2565 - accuracy: 0.9
067
Epoch 20/100
75/75 [==============================] - 0s 1ms/step - loss: 0.2201 - accuracy: 0.973
3
```

```
Epoch 21/100
75/75 [==============================] - 0s 1ms/step - loss: 0.2310 - accuracy: 0.946
7
Epoch 22/100
75/75 [==============================] - 0s 912us/step - loss: 0.2190 - accuracy: 0.9
333
Epoch 23/100
75/75 [==============================] - 0s 885us/step - loss: 0.2184 - accuracy: 0.9
467
Epoch 24/100
75/75 [==============================] - 0s 892us/step - loss: 0.1988 - accuracy: 0.9
600
Epoch 25/100
75/75 [==============================] - 0s 903us/step - loss: 0.1930 - accuracy: 0.9
600
Epoch 26/100
75/75 [==============================] - 0s 926us/step - loss: 0.2096 - accuracy: 0.9
200
Epoch 27/100
75/75 [==============================] - 0s 926us/step - loss: 0.1778 - accuracy: 0.9
467
Epoch 28/100
75/75 [==============================] - 0s 932us/step - loss: 0.1819 - accuracy: 0.9
200
Epoch 29/100
75/75 [==============================] - 0s 872us/step - loss: 0.1699 - accuracy: 0.9
333
Epoch 30/100
75/75 [==============================] - 0s 899us/step - loss: 0.1527 - accuracy: 0.9
600
Epoch 31/100
75/75 [==============================] - 0s 892us/step - loss: 0.1800 - accuracy: 0.9
333
Epoch 32/100
75/75 [==============================] - 0s 899us/step - loss: 0.1561 - accuracy: 0.9
467
Epoch 33/100
75/75 [==============================] - 0s 939us/step - loss: 0.1688 - accuracy: 0.9
467
Epoch 34/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1623 - accuracy: 0.946
7
Epoch 35/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1191 - accuracy: 0.973
3
Epoch 36/100
75/75 [==============================] - 0s 987us/step - loss: 0.1568 - accuracy: 0.9
467
Epoch 37/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1488 - accuracy: 0.960
0
Epoch 38/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1597 - accuracy: 0.933
3
Epoch 39/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1452 - accuracy: 0.946
7
Epoch 40/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1394 - accuracy: 0.946
7
```

```
Epoch 41/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1638 - accuracy: 0.893
3
Epoch 42/100
75/75 [==============================] - 0s 993us/step - loss: 0.1101 - accuracy: 0.9
733
Epoch 43/100
75/75 [==============================] - 0s 994us/step - loss: 0.1432 - accuracy: 0.9
467
Epoch 44/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1449 - accuracy: 0.946
7
Epoch 45/100
75/75 [==============================] - 0s 994us/step - loss: 0.0924 - accuracy: 0.9
867
Epoch 46/100
75/75 [==============================] - 0s 986us/step - loss: 0.1490 - accuracy: 0.9
333
Epoch 47/100
75/75 [==============================] - 0s 980us/step - loss: 0.1271 - accuracy: 0.9
600
Epoch 48/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1409 - accuracy: 0.933
3
Epoch 49/100
75/75 [==============================] - 0s 993us/step - loss: 0.1347 - accuracy: 0.9
333
Epoch 50/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1035 - accuracy: 0.973
3
Epoch 51/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1200 - accuracy: 0.960
0
Epoch 52/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0861 - accuracy: 0.960
0
Epoch 53/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1260 - accuracy: 0.960
0
Epoch 54/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0792 - accuracy: 0.973
3
Epoch 55/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1024 - accuracy: 0.960
0
Epoch 56/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0704 - accuracy: 0.986
7
Epoch 57/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0915 - accuracy: 0.960
0
Epoch 58/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0824 - accuracy: 0.973
3
Epoch 59/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1081 - accuracy: 0.973
3
Epoch 60/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1374 - accuracy: 0.946
7
```

```
Epoch 61/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1453 - accuracy: 0.920
0
Epoch 62/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1269 - accuracy: 0.960
0
Epoch 63/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1488 - accuracy: 0.933
3
Epoch 64/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1187 - accuracy: 0.960
0
Epoch 65/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1221 - accuracy: 0.946
7
Epoch 66/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1144 - accuracy: 0.933
3
Epoch 67/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1045 - accuracy: 0.946
7
Epoch 68/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1703 - accuracy: 0.933
3
Epoch 69/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0727 - accuracy: 0.986
7
Epoch 70/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1317 - accuracy: 0.960
0
Epoch 71/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1188 - accuracy: 0.946
7
Epoch 72/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0776 - accuracy: 0.986
7
Epoch 73/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1255 - accuracy: 0.946
7
Epoch 74/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0677 - accuracy: 0.986
7
Epoch 75/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1409 - accuracy: 0.933
3
Epoch 76/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0894 - accuracy: 0.973
3
Epoch 77/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0716 - accuracy: 0.973
3
Epoch 78/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0616 - accuracy: 0.986
7
Epoch 79/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0621 - accuracy: 0.986
7
Epoch 80/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1117 - accuracy: 0.960
0
```

```
Epoch 81/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0938 - accuracy: 0.946
7
Epoch 82/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1344 - accuracy: 0.933
3
Epoch 83/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0709 - accuracy: 0.973
3
Epoch 84/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1542 - accuracy: 0.920
0
Epoch 85/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0603 - accuracy: 0.986
7
Epoch 86/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1627 - accuracy: 0.946
7
Epoch 87/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0675 - accuracy: 0.986
7
Epoch 88/100
75/75 [==============================] - 0s 924us/step - loss: 0.1037 - accuracy: 0.9
467
Epoch 89/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1022 - accuracy: 0.946
7
Epoch 90/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0540 - accuracy: 0.986
7
Epoch 91/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1046 - accuracy: 0.946
7
Epoch 92/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0890 - accuracy: 0.946
7
Epoch 93/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1050 - accuracy: 0.933
3
Epoch 94/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0703 - accuracy: 0.960
0
Epoch 95/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0520 - accuracy: 0.986
7
Epoch 96/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0741 - accuracy: 0.960
0
Epoch 97/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0946 - accuracy: 0.960
0
Epoch 98/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0679 - accuracy: 0.973
3
Epoch 99/100
75/75 [==============================] - 0s 1ms/step - loss: 0.0986 - accuracy: 0.960
0
Epoch 100/100
75/75 [==============================] - 0s 1ms/step - loss: 0.1323 - accuracy: 0.946
7
```

Out[118]:

KerasClassifier

```
KerasClassifier(
        model=<function create_keras_model at 0x000001AEACFB65E0>
        build_fn=None
        warm_start=False
        random_state=None
        optimizer=<keras.optimizers.optimizer_v2.gradient_descent.SGD obj
ect at 0x000001AEAD22E280>
        loss=categorical_crossentropy
        metrics=['accuracy']
        batch_size=1
        validation_batch_size=None
```

In [119…

```
y_pred = clf.predict(X_test)
y_pred_class = np.argmax(y_pred, axis=1)
```

```
75/75 [==============================] - 0s 689us/step
```

22) Use the predictions to generate a confusion matrix.

In [120…

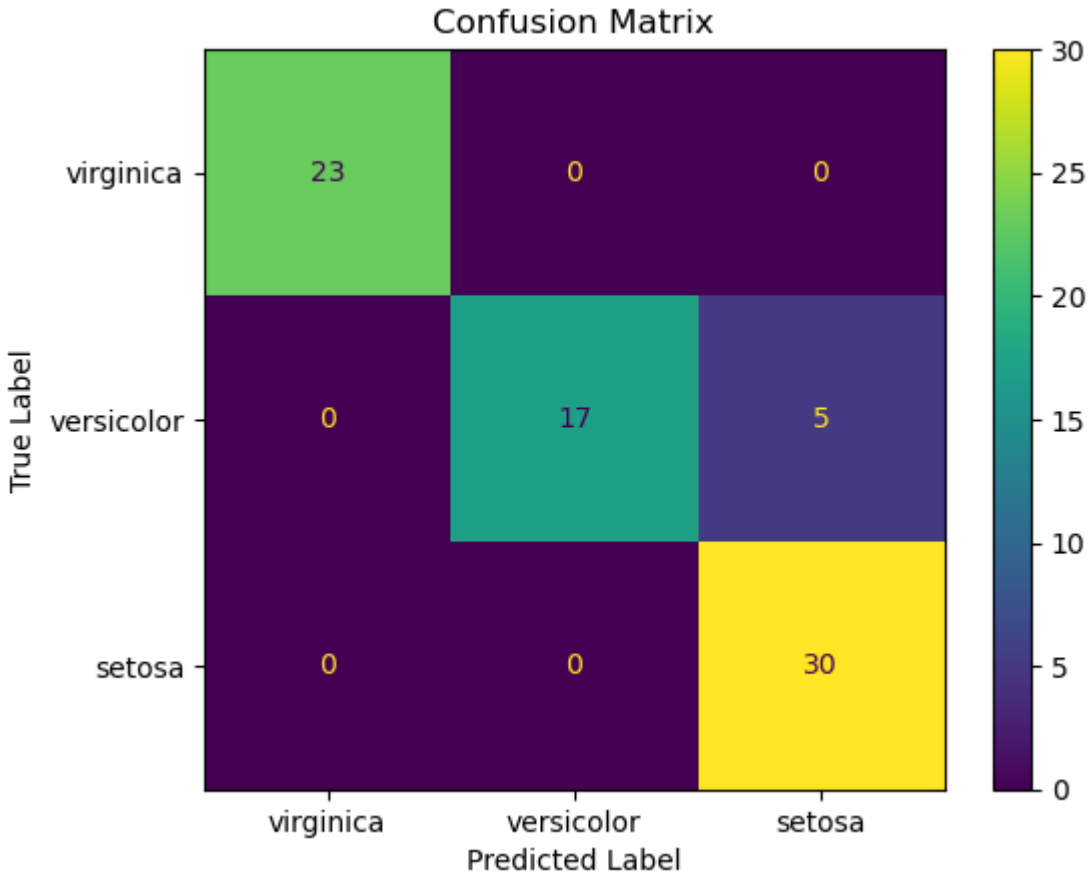
```
# convert string labels to integers
le = LabelEncoder()
y_test_num = le.fit_transform(y_test)

# compute confusion matrix and classification report
cmat = confusion_matrix(y_test_num, y_pred_class)

cmat_disp = ConfusionMatrixDisplay(cmat, display_labels=y.unique())
cmat_disp.plot()
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

Out[120]:

```
Text(0, 0.5, 'True Label')
```



23) Generate a performance report with the classification_report function.

In [121…
```
crep = classification_report(y_test_num, y_pred_class)
print("Classification Report:")
print(crep)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       1.00      0.77      0.87        22
           2       0.86      1.00      0.92        30

    accuracy                           0.93        75
   macro avg       0.95      0.92      0.93        75
weighted avg       0.94      0.93      0.93        75
```

24) Use your code from lab10 that performed a full cross validation, with K set to 5. AND, because deep learning models can take a while to train each model, it is a good idea to generate some output in your loop to show that the cross validation is progressing

In [125…
```
kfold = KFold(n_splits = 5, shuffle = True, random_state = 100)

results_df = pd.DataFrame(columns=['true_label', 'dt_default'])
for i, (train_index, test_index) in enumerate(kfold.split(X)):
    print("Starting Fold,", i+1)
    X_train, X_test = X[X.index.isin(train_index)], X[X.index.isin(test_index)]
    y_train, y_test = y[y.index.isin(train_index)], y[y.index.isin(test_index)]
```

```
    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)

    y_pred_test = clf.predict(X_test)
    fold_df = pd.DataFrame({'true_label': y_test, 'dt_default': y_pred_test})
    results_df = pd.concat([results_df, fold_df], ignore_index=True)
    print("Fold,", i+1, "complete")

results_df
```

```
Starting Fold, 1
Fold, 1 complete
Starting Fold, 2
Fold, 2 complete
Starting Fold, 3
Fold, 3 complete
Starting Fold, 4
Fold, 4 complete
Starting Fold, 5
Fold, 5 complete
```

Out[125]:

	true_label	dt_default
0	versicolor	versicolor
1	setosa	setosa
2	versicolor	versicolor
3	setosa	setosa
4	versicolor	versicolor
...	...	...
145	versicolor	versicolor
146	versicolor	versicolor
147	virginica	virginica
148	versicolor	versicolor
149	setosa	setosa

150 rows × 2 columns

25) Generate a full confusion matrix and final classification report based on your 5-fold cross validation of the keras model.

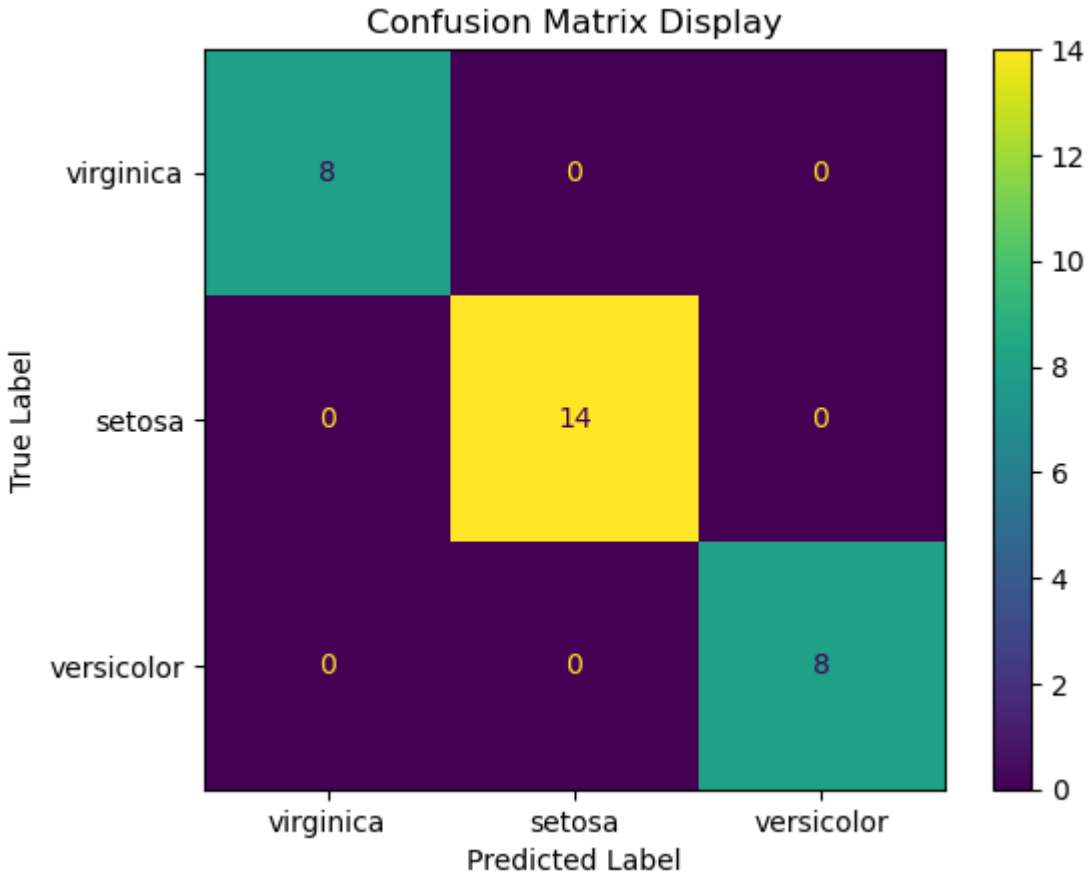
In [127…

```
conf_mat = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix:\n", conf_mat)

conf_mat_disp = ConfusionMatrixDisplay(conf_mat, display_labels=y_test.unique())
conf_mat_disp.plot()
plt.title('Confusion Matrix Display')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
Confusion Matrix:
 [[ 8  0  0]
 [ 0 14  0]
 [ 0  0  8]]
```

Out[127]: Text(0, 0.5, 'True Label')



In [126…

```
test_report = classification_report(y_test, y_pred_test, target_names=y_test.unique(),
test_report_df = pd.DataFrame(test_report).transpose()
print("\nClassication Report on Test Data:\n", test_report_df)
```

```
Classication Report on Test Data:
              precision  recall  f1-score  support
virginica           1.0     1.0       1.0      8.0
setosa              1.0     1.0       1.0     14.0
versicolor          1.0     1.0       1.0      8.0
accuracy            1.0     1.0       1.0      1.0
macro avg           1.0     1.0       1.0     30.0
weighted avg        1.0     1.0       1.0     30.0
```