PRCP-1002-Handwritten Digits Recognition

Problem Statement:

MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. Your goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. We encourage you to experiment with different algorithms to learn first-hand what works well and how techniques compare.

Task 1:-Prepare a complete data analysis report on the given data.

Task 2:- Classify a given image of a handwritten digit into one of the 10 classes representing integer values from 0 to 9.

Task3:- Compare between various models and find the classifier that works better.

Things to be covered in this Project:

- Install the latest Tensorflow library
- Prepare the dataset for the model
- Develop Single Layer Perceptron model for classifying the handwritten digits
- Plot the change in accuracy per epochs
- Evaluate the model on the testing data
- Analyze the model summary
- Add hidden layer to the model to make it Multi-Layer Perceptron
- Add Dropout to prevent overfitting and check its effect on accuracy
- Increasing the number of Hidden Layer neuron and check its effect on accuracy
- Use different optimizers and check its effect on accuracy
- Increase the hidden layers and check its effect on accuracy
- Manipulate the batch_size and epochs and check its effect on accuracy

MNIST Dataset Description

The MNIST Handwritten Digit Recognition Dataset contains 60,000 training and 10,000 testing labelled handwritten digit pictures.

Each picture is 28 pixels in height and 28 pixels wide, for a total of 784 (28×28) pixels. Each pixel has a single pixel value associated with it. It indicates how bright or dark that pixel is (larger numbers indicates darker pixel). This pixel value is an integer ranging from 0 to 255.

Task 1:-Prepare a complete data analysis report on the given data.

Install the latest Tensorflow 2.x version

In [1]:
```
#installing the latest version of tensorflow
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\user\anaconda3\lib\site-packages (2.13.0)
Requirement already satisfied: tensorflow-intel==2.13.0 in c:\users\user\anaconda3\lib\site-packages (from tens
orflow) (2.13.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-intel=
=2.13.0->tensorflow) (3.7.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-i
ntel==2.13.0->tensorflow) (2.3.0)
Requirement already satisfied: six>=1.12.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-intel=
=2.13.0->tensorflow) (1.16.0)
Requirement already satisfied: numpy<=1.24.3,>=1.22 in c:\users\user\anaconda3\lib\site-packages (from tensorfl
ow-intel==2.13.0->tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-
intel==2.13.0->tensorflow) (3.3.0)
Requirement already satisfied: typing-extensions<4.6.0,>=3.6.6 in c:\users\user\anaconda3\lib\site-packages (fr
om tensorflow-intel==2.13.0->tensorflow) (4.4.0)
Requirement already satisfied: setuptools in c:\users\user\anaconda3\lib\site-packages (from tensorflow-intel==
2.13.0->tensorflow) (65.6.3)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\user\anaconda3\lib\site-packages (from tensorflo
w-intel==2.13.0->tensorflow) (0.2.0)
Requirement already satisfied: tensorboard<2.14,>=2.13 in c:\users\user\anaconda3\lib\site-packages (from tenso
rflow-intel==2.13.0->tensorflow) (2.13.0)
Requirement already satisfied: flatbuffers>=23.1.21 in c:\users\user\anaconda3\lib\site-packages (from tensorfl
ow-intel==2.13.0->tensorflow) (23.5.26)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-inte
l==2.13.0->tensorflow) (1.14.1)
Requirement already satisfied: packaging in c:\users\user\anaconda3\lib\site-packages (from tensorflow-intel==2
.13.0->tensorflow) (22.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\user\anaconda3\lib\site-packages (from tensorflo
w-intel==2.13.0->tensorflow) (0.4.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\user\anaconda3\lib\site-package
s (from tensorflow-intel==2.13.0->tensorflow) (0.31.0)
Requirement already satisfied: keras<2.14,>=2.13.1 in c:\users\user\anaconda3\lib\site-packages (from tensorflo
w-intel==2.13.0->tensorflow) (2.13.1)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-
intel==2.13.0->tensorflow) (1.6.3)
Requirement already satisfied: libclang>=13.0.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-i
ntel==2.13.0->tensorflow) (16.0.6)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\user\anaconda3\lib\site-packages (from tensorflo
w-intel==2.13.0->tensorflow) (1.58.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in c:\users\user\anaconda3\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow) (4.24.3)
Requirement already satisfied: tensorflow-estimator<2.14,>=2.13.0 in c:\users\user\anaconda3\lib\site-packages
(from tensorflow-intel==2.13.0->tensorflow) (2.13.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-int
el==2.13.0->tensorflow) (1.4.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\user\anaconda3\lib\site-packages (from astunparse
>=1.6.0->tensorflow-intel==2.13.0->tensorflow) (0.38.4)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in c:\users\user\anaconda3\lib\site-packages (fro
m tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in c:\users\user\anaconda3\lib\site-packages (from tensorboard<2
.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (3.4.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\user\anaconda3\lib\site-packages (from tensorboard<2
.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (2.2.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\user\anaconda3\lib\site-packages (from tensorb
oard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (2.23.0)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\user\anaconda3\lib\site-packages (from tensorboa
rd<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (2.28.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users\user\anaconda3\lib\site-packag
es (from tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (0.7.1)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\user\anaconda3\lib\site-packages (from google-auth<3,>
=1.6.3->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (4.9)
Requirement already satisfied: urllib3<2.0 in c:\users\user\anaconda3\lib\site-packages (from google-auth<3,>=1
.6.3->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (1.26.14)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\user\anaconda3\lib\site-packages (from google-
auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (0.2.8)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\user\anaconda3\lib\site-packages (from google
-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (5.3.1)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\user\anaconda3\lib\site-packages (from goog
le-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (1.3.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\user\anaconda3\lib\site-packages (from requests<3
,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (2023.7.22)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\user\anaconda3\lib\site-packages (from requ
ests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\user\anaconda3\lib\site-packages (from requests<3,>=2.2
1.0->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (3.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\user\anaconda3\lib\site-packages (from werkzeug>=1
.0.1->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\users\user\anaconda3\lib\site-packages (from pyasn1-m
odules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\user\anaconda3\lib\site-packages (from requests-oaut
hlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (3.
2.2)
```

In [2]:
```
import tensorflow as tf
from tensorflow import keras
```

Preparing the Handwritten Digit Recognition dataset

In [3]:
```
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

This block of code will load the images of handwritten digits from mnist dataset and randomly split the it as X_train, Y_train (to separate the features and label of training data) and X_test, Y_test (features and label of testing data). These split datasets will be used to train and test our model.

Let's check the number of entries in our dataset. For this we will be printing the shape of X_train and X_test

In [4]:
```
# Data Exploration
print(X_train.shape)
print(X_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

From the above output we can see that we have 60000 entries (images) as part of train data with 28×28 pixel values and 10000 entries as a part of test of data of same size. You can check the individual pixels of any of the image, eg: X_train[0].

Let's preprocess our data for further usage. We will reshape the dataset from 28×28 to 784 and convert it into float32 datatype for training our neural network.

- Reshape the data
- Change the datatype to float32
- Normalize the dataset
- Perform One-Hot Encoding on the labels

In [5]:
```
# X_train is 60000 rows of 28x28 values; we reshape it to # 60000 x 784.
RESHAPED = 784 # 28x28 = 784 neurons
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
```

In [6]:
```
# Data is converted into float32 to use 32-bit precision # when training a neural network
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

In [7]:
```
# Normalizing the input to be within the range [0,1]
X_train /= 255
#intensity of each pixel is divided by 255, the maximum intensity value
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

In [8]:
```
# One-hot representation of the labels.
Y_train = tf.keras.utils.to_categorical(Y_train, 10)
Y_test = tf.keras.utils.to_categorical(Y_test, 10)
```

- The output tells the number of records within the train and test data.
- Now that we have prepared our data. Next we will be using this data to build our model.

Task3:- Compare between various models and find the classifier that works better.

Task-2 Has Been Done After This Task-3 Because We Have To Build First Model So

Building the Handwritten Digit Recognition Models

Preparing the 1st Model: Single layer Perceptron

- This model is the most basic sequential model with 0 hidden layers in it.

Adding the model layer

- We will be building the simplest model defined in the Sequential class as a linear stack of Layers

Adding Activation Function to the model layer

Activation function is defined in the dense layer of the model and is used to squeeze the value within a particular range. In simple term it is a function which is used to convert the input signal of a node to an output signal. tf.keras comes with the following predefined activation functions to choose from:

- softmax
- sigmoid
- tanh
- relu

In [9]:
```
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras import Sequential
model_1 = Sequential()

# Now the model will take as input arrays of shape (*, 784)# and output arrays of shape (*, 10)
model_1.add(Dense(10,input_shape=(784,),name='dense_layer', activation='softmax'))
```

In the above code we are importing the sequential keras model with 0 hidden layers. We have defined the output layer as 10. This is our dense layer. 10 is chosen as we have numbers from 0 to 9 to be classified in the dataset. shape. Total number of neurons in the input layer is 784. The activation function chosen in the dense layer is softmax. We will learn more about the softmax function in detail in our next blog. In simple terms, the model will have 784 input neurons to give the output between 0-9 numbers.

Compiling the model

- Next step is to compile the model. For compiling we need to define three parameters: optimizer, loss, and metrics. ## 1. Optimizer: While training a deep learning model, we need to alter the weights of each epoch and minimize the loss function. An optimizer is a function or algorithm that adjusts the neural network's properties such as weights and learning rate. As a result, it helps to reduce total loss and enhance accuracy of your model.

Some of the popular Gradient Descent Optimizers are:

- SGD: Stochastic gradient descent, to reduce the computation cost of gradient
- RMSprop: Adaptive learning rate optimization method which utilizes the magnitude of recent gradients to normalize the gradients
- Adam: Adaptive Moment Estimation (Adam) leverages the power of adaptive learning rates methods to find individual learning rates for each parameter

2. Loss: Loss functions are a measure of how well your model predicts the predicted outcome.

Some of the popular Model Loss Function are:

- mse : for mean squared error
- binary_crossentropy:for binary logarithmic loss (logloss)
- categorical_crossentropy: for multi class logarithmic loss (logloss)

In [10]:
```
# Compiling the model.
model_1.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Training the model

Model will be now trained on the on the training data. For this we will be defining the epochs, batchsize, and validation size

- epoch: Number of times that the model will run through the training dataset
- batch_size: Number of training instances to be shown to the model before a weight is updated
- validation_split: Defines the fraction of data to be used for validation purpose

In [11]:
```
# Training the model.
```

```
training = model_1.fit(X_train, Y_train, batch_size=64, epochs=70, validation_split=0.2)
```

```
Epoch 1/70
750/750 [==============================] - 3s 3ms/step - loss: 1.1015 - accuracy: 0.7395 - val_loss: 0.6605 - v
al_accuracy: 0.8576
Epoch 2/70
750/750 [==============================] - 2s 2ms/step - loss: 0.6094 - accuracy: 0.8556 - val_loss: 0.5100 - v
al_accuracy: 0.8770
Epoch 3/70
750/750 [==============================] - 2s 2ms/step - loss: 0.5129 - accuracy: 0.8700 - val_loss: 0.4510 - v
al_accuracy: 0.8876
Epoch 4/70
750/750 [==============================] - 2s 2ms/step - loss: 0.4664 - accuracy: 0.8785 - val_loss: 0.4184 - v
al_accuracy: 0.8932
Epoch 5/70
750/750 [==============================] - 2s 2ms/step - loss: 0.4378 - accuracy: 0.8845 - val_loss: 0.3972 - v
al_accuracy: 0.8969
Epoch 6/70
750/750 [==============================] - 2s 2ms/step - loss: 0.4180 - accuracy: 0.8880 - val_loss: 0.3818 - v
al_accuracy: 0.9007
Epoch 7/70
750/750 [==============================] - 3s 4ms/step - loss: 0.4033 - accuracy: 0.8913 - val_loss: 0.3703 - v
al_accuracy: 0.9027
Epoch 8/70
750/750 [==============================] - 3s 4ms/step - loss: 0.3917 - accuracy: 0.8942 - val_loss: 0.3612 - v
al_accuracy: 0.9041
Epoch 9/70
750/750 [==============================] - 3s 4ms/step - loss: 0.3822 - accuracy: 0.8965 - val_loss: 0.3537 - v
al_accuracy: 0.9051
Epoch 10/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3743 - accuracy: 0.8976 - val_loss: 0.3476 - v
al_accuracy: 0.9068
Epoch 11/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3676 - accuracy: 0.8990 - val_loss: 0.3423 - v
al_accuracy: 0.9072
Epoch 12/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3618 - accuracy: 0.9004 - val_loss: 0.3377 - v
al_accuracy: 0.9090
Epoch 13/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3567 - accuracy: 0.9015 - val_loss: 0.3336 - v
al_accuracy: 0.9093
Epoch 14/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3522 - accuracy: 0.9025 - val_loss: 0.3301 - v
al_accuracy: 0.9098
Epoch 15/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3481 - accuracy: 0.9032 - val_loss: 0.3268 - v
al_accuracy: 0.9113
Epoch 16/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3445 - accuracy: 0.9046 - val_loss: 0.3241 - v
al_accuracy: 0.9113
Epoch 17/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3411 - accuracy: 0.9054 - val_loss: 0.3214 - v
al_accuracy: 0.9115
Epoch 18/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3381 - accuracy: 0.9057 - val_loss: 0.3190 - v
al_accuracy: 0.9119
Epoch 19/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3354 - accuracy: 0.9071 - val_loss: 0.3167 - v
al_accuracy: 0.9122
Epoch 20/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3327 - accuracy: 0.9079 - val_loss: 0.3151 - v
al_accuracy: 0.9127
Epoch 21/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3303 - accuracy: 0.9083 - val_loss: 0.3130 - v
al_accuracy: 0.9132
Epoch 22/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3281 - accuracy: 0.9092 - val_loss: 0.3112 - v
al_accuracy: 0.9139
Epoch 23/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3261 - accuracy: 0.9099 - val_loss: 0.3096 - v
al_accuracy: 0.9141
Epoch 24/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3241 - accuracy: 0.9102 - val_loss: 0.3081 - v
al_accuracy: 0.9147
Epoch 25/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3223 - accuracy: 0.9108 - val_loss: 0.3066 - v
al_accuracy: 0.9154
Epoch 26/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3205 - accuracy: 0.9115 - val_loss: 0.3053 - v
al_accuracy: 0.9153
Epoch 27/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3189 - accuracy: 0.9117 - val_loss: 0.3041 - v
al_accuracy: 0.9156
Epoch 28/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3174 - accuracy: 0.9123 - val_loss: 0.3029 - v
al_accuracy: 0.9155
Epoch 29/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3159 - accuracy: 0.9124 - val_loss: 0.3019 - v
al_accuracy: 0.9157
```

```
Epoch 30/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3145 - accuracy: 0.9129 - val_loss: 0.3007 - v
al_accuracy: 0.9163
Epoch 31/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3132 - accuracy: 0.9138 - val_loss: 0.2998 - v
al_accuracy: 0.9165
Epoch 32/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3119 - accuracy: 0.9134 - val_loss: 0.2989 - v
al_accuracy: 0.9166
Epoch 33/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3107 - accuracy: 0.9142 - val_loss: 0.2979 - v
al_accuracy: 0.9167
Epoch 34/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3095 - accuracy: 0.9141 - val_loss: 0.2971 - v
al_accuracy: 0.9172
Epoch 35/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3083 - accuracy: 0.9143 - val_loss: 0.2964 - v
al_accuracy: 0.9176
Epoch 36/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3074 - accuracy: 0.9146 - val_loss: 0.2955 - v
al_accuracy: 0.9179
Epoch 37/70
750/750 [==============================] - 2s 2ms/step - loss: 0.3064 - accuracy: 0.9153 - val_loss: 0.2949 - v
al_accuracy: 0.9179
Epoch 38/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3053 - accuracy: 0.9154 - val_loss: 0.2942 - v
al_accuracy: 0.9184
Epoch 39/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3044 - accuracy: 0.9157 - val_loss: 0.2934 - v
al_accuracy: 0.9182
Epoch 40/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3035 - accuracy: 0.9160 - val_loss: 0.2927 - v
al_accuracy: 0.9185
Epoch 41/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3027 - accuracy: 0.9162 - val_loss: 0.2920 - v
al_accuracy: 0.9183
Epoch 42/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3018 - accuracy: 0.9163 - val_loss: 0.2914 - v
al_accuracy: 0.9186
Epoch 43/70
750/750 [==============================] - 2s 2ms/step - loss: 0.3010 - accuracy: 0.9164 - val_loss: 0.2910 - v
al_accuracy: 0.9191
Epoch 44/70
750/750 [==============================] - 2s 3ms/step - loss: 0.3002 - accuracy: 0.9165 - val_loss: 0.2904 - v
al_accuracy: 0.9191
Epoch 45/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2994 - accuracy: 0.9166 - val_loss: 0.2897 - v
al_accuracy: 0.9195
Epoch 46/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2986 - accuracy: 0.9172 - val_loss: 0.2894 - v
al_accuracy: 0.9198
Epoch 47/70
750/750 [==============================] - 2s 2ms/step - loss: 0.2980 - accuracy: 0.9172 - val_loss: 0.2887 - v
al_accuracy: 0.9201
Epoch 48/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2972 - accuracy: 0.9175 - val_loss: 0.2882 - v
al_accuracy: 0.9192
Epoch 49/70
750/750 [==============================] - 2s 2ms/step - loss: 0.2966 - accuracy: 0.9172 - val_loss: 0.2878 - v
al_accuracy: 0.9197
Epoch 50/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2960 - accuracy: 0.9177 - val_loss: 0.2874 - v
al_accuracy: 0.9204
Epoch 51/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2953 - accuracy: 0.9179 - val_loss: 0.2868 - v
al_accuracy: 0.9200
Epoch 52/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2947 - accuracy: 0.9181 - val_loss: 0.2865 - v
al_accuracy: 0.9200
Epoch 53/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2941 - accuracy: 0.9185 - val_loss: 0.2861 - v
al_accuracy: 0.9202
Epoch 54/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2935 - accuracy: 0.9185 - val_loss: 0.2856 - v
al_accuracy: 0.9207
Epoch 55/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2929 - accuracy: 0.9186 - val_loss: 0.2853 - v
al_accuracy: 0.9201
Epoch 56/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2924 - accuracy: 0.9187 - val_loss: 0.2850 - v
al_accuracy: 0.9206
Epoch 57/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2918 - accuracy: 0.9189 - val_loss: 0.2847 - v
al_accuracy: 0.9202
Epoch 58/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2914 - accuracy: 0.9189 - val_loss: 0.2840 - v
al_accuracy: 0.9211
Epoch 59/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2908 - accuracy: 0.9191 - val_loss: 0.2838 - v
```

```
al_accuracy: 0.9204
Epoch 60/70
750/750 [==============================] - 3s 3ms/step - loss: 0.2903 - accuracy: 0.9194 - val_loss: 0.2833 - v
al_accuracy: 0.9215
Epoch 61/70
750/750 [==============================] - 3s 4ms/step - loss: 0.2898 - accuracy: 0.9194 - val_loss: 0.2830 - v
al_accuracy: 0.9217
Epoch 62/70
750/750 [==============================] - 3s 4ms/step - loss: 0.2893 - accuracy: 0.9195 - val_loss: 0.2830 - v
al_accuracy: 0.9218
Epoch 63/70
750/750 [==============================] - 3s 5ms/step - loss: 0.2888 - accuracy: 0.9197 - val_loss: 0.2825 - v
al_accuracy: 0.9212
Epoch 64/70
750/750 [==============================] - 4s 5ms/step - loss: 0.2884 - accuracy: 0.9197 - val_loss: 0.2822 - v
al_accuracy: 0.9217
Epoch 65/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2879 - accuracy: 0.9199 - val_loss: 0.2819 - v
al_accuracy: 0.9219
Epoch 66/70
750/750 [==============================] - 3s 4ms/step - loss: 0.2875 - accuracy: 0.9202 - val_loss: 0.2817 - v
al_accuracy: 0.9214
Epoch 67/70
750/750 [==============================] - 3s 4ms/step - loss: 0.2870 - accuracy: 0.9199 - val_loss: 0.2815 - v
al_accuracy: 0.9215
Epoch 68/70
750/750 [==============================] - 3s 4ms/step - loss: 0.2866 - accuracy: 0.9200 - val_loss: 0.2810 - v
al_accuracy: 0.9220
Epoch 69/70
750/750 [==============================] - 3s 4ms/step - loss: 0.2862 - accuracy: 0.9201 - val_loss: 0.2810 - v
al_accuracy: 0.9221
Epoch 70/70
750/750 [==============================] - 2s 3ms/step - loss: 0.2858 - accuracy: 0.9205 - val_loss: 0.2808 - v
al_accuracy: 0.9218
```

From the above output you can see that with each epoch the loss is reduced and the val_accuracy is being improved.

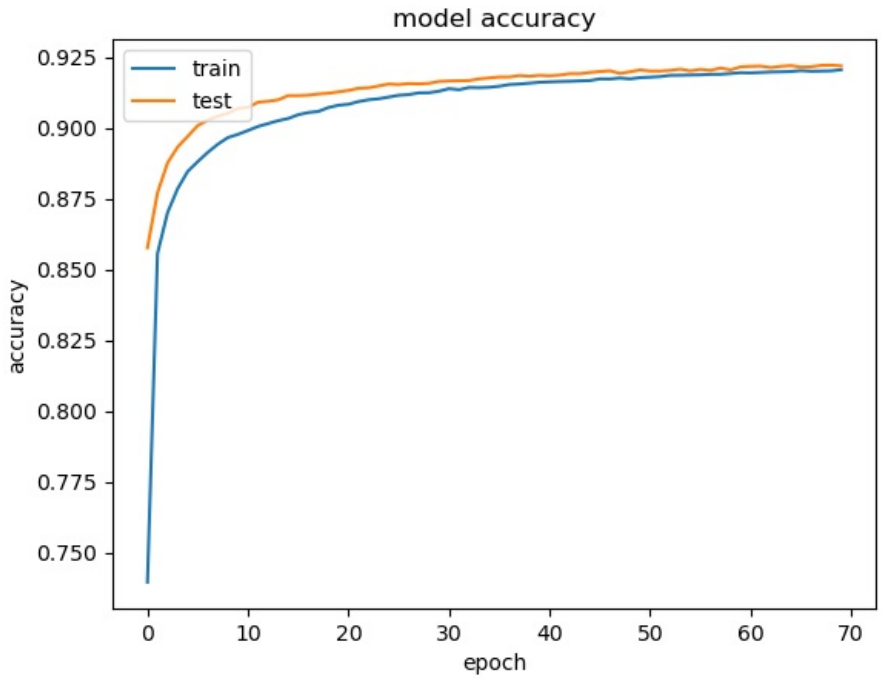
Plot the change in accuracy and loss per epochs

You can plot a curve to check the variation of accuracy and loss as the number of epochs increases. For this you can use, matplotlib to plot the curve.

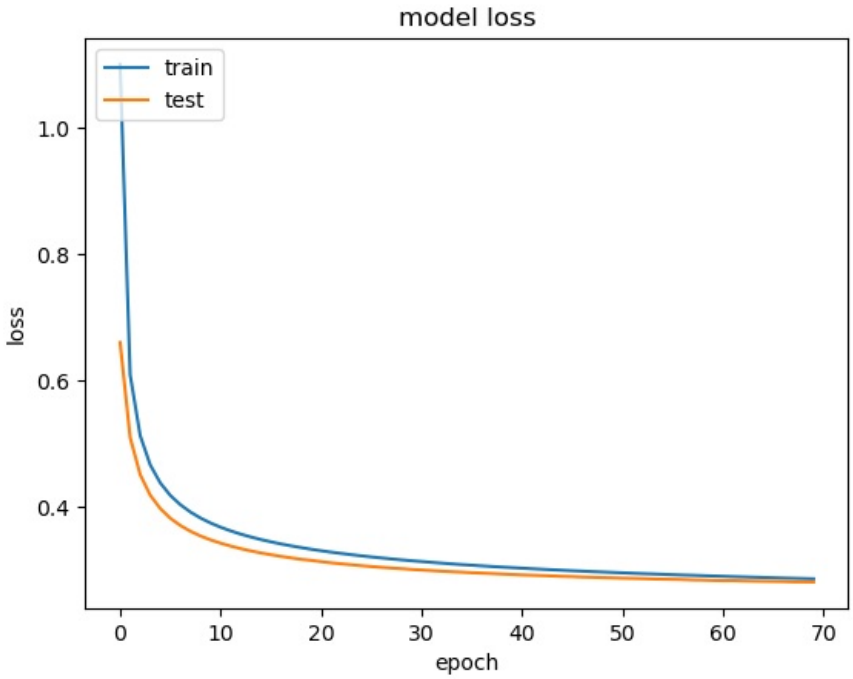
In [12]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# list all data in training
print(training.history.keys())
```

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

In [13]:
```
# summarize training for accuracy
plt.plot(training.history['accuracy'])
plt.plot(training.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [14]:
```
# summarize traning for loss
plt.plot(training.history['loss'])
plt.plot(training.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Evaluating the Handwritten Digit Recognition Model on Test Data

We will now test the accuracy of the model on the testing dataset.

In [15]:
```
#evaluate the model
test_loss, test_acc = model_1.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2824 - accuracy: 0.9216
Test accuracy: 0.9215999841690063
```

Checking Configuration

In [16]:
```
model_1.get_config()
```

Out[16]:

```
{'name': 'sequential',
 'layers': [{'module': 'keras.layers',
   'class_name': 'InputLayer',
   'config': {'batch_input_shape': (None, 784),
    'dtype': 'float32',
    'sparse': False,
    'ragged': False,
    'name': 'dense_layer_input'},
   'registered_name': None},
  {'module': 'keras.layers',
   'class_name': 'Dense',
   'config': {'name': 'dense_layer',
    'trainable': True,
    'dtype': 'float32',
    'batch_input_shape': (None, 784),
    'units': 10,
    'activation': 'softmax',
    'use_bias': True,
    'kernel_initializer': {'module': 'keras.initializers',
     'class_name': 'GlorotUniform',
     'config': {'seed': None},
     'registered_name': None},
    'bias_initializer': {'module': 'keras.initializers',
     'class_name': 'Zeros',
     'config': {},
     'registered_name': None},
    'kernel_regularizer': None,
    'bias_regularizer': None,
    'activity_regularizer': None,
    'kernel_constraint': None,
    'bias_constraint': None},
   'registered_name': None,
   'build_config': {'input_shape': (None, 784)}}]}
```


Next step is to improve the base model we just created. This base model is a single layer perceptron with zero hidden layers. Let's add some hidden layers to our model to check if it improves the accuracy.

Improved Model 2: Adding Hidden Layer – Multi Layer Perceptron

- In the model we will add a hidden layer and 3 dense layer. The hidden layer consists of 64 neurons. The new dense_layer_2 has 64 neurons and relu activation layer. Let's experiment by increasing the number of epochs to 100 in this model.

In [17]:
```
#Most common type of model is a stack of layers
model_2 = tf.keras.Sequential()
N_hidden = 64
```

In [18]:
```
# Adds a densely-connected layer with 64 units to the model:

model_2.add(Dense(N_hidden, name='dense_layer', input_shape=(784,), activation = 'relu'))
# Now the model will take as input arrays of shape (*, 784)# and output arrays of shape (*, 64)

# Adding another dense layer:
model_2.add(Dense(N_hidden, name='dense_layer_2', activation='relu'))
```

In [19]:
```
# After the first layer, you don't need to specify the size of the input anymore:
# Add an output layer with 10 output units (10 different classes):
model_2.add(Dense(10, name='dense_layer_3', activation = 'softmax'))
```

In [20]:
```
# Compiling the model.
model_2.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [21]:
```
# Training the model.
training = model_2.fit(X_train, Y_train, batch_size=64, epochs=100, validation_split=0.2)
```

```
Epoch 1/100
750/750 [==============================] - 4s 4ms/step - loss: 1.1254 - accuracy: 0.6883 - val_loss: 0.5109 - v
al_accuracy: 0.8688
Epoch 2/100
750/750 [==============================] - 3s 4ms/step - loss: 0.4510 - accuracy: 0.8786 - val_loss: 0.3660 - v
al_accuracy: 0.8978
Epoch 3/100
750/750 [==============================] - 3s 4ms/step - loss: 0.3636 - accuracy: 0.8981 - val_loss: 0.3177 - v
al_accuracy: 0.9083
Epoch 4/100
750/750 [==============================] - 3s 4ms/step - loss: 0.3249 - accuracy: 0.9080 - val_loss: 0.2923 - v
al_accuracy: 0.9162
Epoch 5/100
750/750 [==============================] - 3s 4ms/step - loss: 0.2987 - accuracy: 0.9152 - val_loss: 0.2721 - v
al_accuracy: 0.9222
Epoch 6/100
```

```
750/750 [==============================] - 3s 4ms/step - loss: 0.2783 - accuracy: 0.9219 - val_loss: 0.2568 - v
al_accuracy: 0.9252
Epoch 7/100
750/750 [==============================] - 4s 5ms/step - loss: 0.2612 - accuracy: 0.9253 - val_loss: 0.2473 - v
al_accuracy: 0.9277
Epoch 8/100
750/750 [==============================] - 4s 5ms/step - loss: 0.2471 - accuracy: 0.9295 - val_loss: 0.2353 - v
al_accuracy: 0.9330
Epoch 9/100
750/750 [==============================] - 3s 4ms/step - loss: 0.2340 - accuracy: 0.9337 - val_loss: 0.2223 - v
al_accuracy: 0.9366
Epoch 10/100
750/750 [==============================] - 4s 5ms/step - loss: 0.2226 - accuracy: 0.9367 - val_loss: 0.2152 - v
al_accuracy: 0.9373
Epoch 11/100
750/750 [==============================] - 3s 4ms/step - loss: 0.2121 - accuracy: 0.9395 - val_loss: 0.2103 - v
al_accuracy: 0.9389
Epoch 12/100
750/750 [==============================] - 4s 5ms/step - loss: 0.2022 - accuracy: 0.9429 - val_loss: 0.2003 - v
al_accuracy: 0.9429
Epoch 13/100
750/750 [==============================] - 3s 5ms/step - loss: 0.1937 - accuracy: 0.9451 - val_loss: 0.1920 - v
al_accuracy: 0.9467
Epoch 14/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1857 - accuracy: 0.9465 - val_loss: 0.1848 - v
al_accuracy: 0.9473
Epoch 15/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1782 - accuracy: 0.9493 - val_loss: 0.1799 - v
al_accuracy: 0.9492
Epoch 16/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1712 - accuracy: 0.9510 - val_loss: 0.1750 - v
al_accuracy: 0.9505
Epoch 17/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1645 - accuracy: 0.9526 - val_loss: 0.1712 - v
al_accuracy: 0.9515
Epoch 18/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1585 - accuracy: 0.9546 - val_loss: 0.1642 - v
al_accuracy: 0.9532
Epoch 19/100
750/750 [==============================] - 4s 5ms/step - loss: 0.1529 - accuracy: 0.9561 - val_loss: 0.1623 - v
al_accuracy: 0.9554
Epoch 20/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1479 - accuracy: 0.9584 - val_loss: 0.1569 - v
al_accuracy: 0.9548
Epoch 21/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1428 - accuracy: 0.9592 - val_loss: 0.1538 - v
al_accuracy: 0.9572
Epoch 22/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1384 - accuracy: 0.9610 - val_loss: 0.1523 - v
al_accuracy: 0.9567
Epoch 23/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1339 - accuracy: 0.9623 - val_loss: 0.1488 - v
al_accuracy: 0.9580
Epoch 24/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1298 - accuracy: 0.9634 - val_loss: 0.1467 - v
al_accuracy: 0.9578
Epoch 25/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1258 - accuracy: 0.9646 - val_loss: 0.1443 - v
al_accuracy: 0.9592
Epoch 26/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1223 - accuracy: 0.9651 - val_loss: 0.1421 - v
al_accuracy: 0.9601
Epoch 27/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1188 - accuracy: 0.9663 - val_loss: 0.1402 - v
al_accuracy: 0.9599
Epoch 28/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1153 - accuracy: 0.9676 - val_loss: 0.1397 - v
al_accuracy: 0.9612
Epoch 29/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1123 - accuracy: 0.9684 - val_loss: 0.1354 - v
al_accuracy: 0.9604
Epoch 30/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1093 - accuracy: 0.9695 - val_loss: 0.1353 - v
al_accuracy: 0.9622
Epoch 31/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1061 - accuracy: 0.9702 - val_loss: 0.1336 - v
al_accuracy: 0.9615
Epoch 32/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1035 - accuracy: 0.9709 - val_loss: 0.1304 - v
al_accuracy: 0.9633
Epoch 33/100
750/750 [==============================] - 3s 4ms/step - loss: 0.1006 - accuracy: 0.9718 - val_loss: 0.1294 - v
al_accuracy: 0.9644
Epoch 34/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0984 - accuracy: 0.9724 - val_loss: 0.1277 - v
al_accuracy: 0.9640
Epoch 35/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0957 - accuracy: 0.9733 - val_loss: 0.1273 - v
al_accuracy: 0.9632
```

```
Epoch 36/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0934 - accuracy: 0.9736 - val_loss: 0.1257 - v
al_accuracy: 0.9652
Epoch 37/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0913 - accuracy: 0.9746 - val_loss: 0.1243 - v
al_accuracy: 0.9653
Epoch 38/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0891 - accuracy: 0.9750 - val_loss: 0.1247 - v
al_accuracy: 0.9646
Epoch 39/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0870 - accuracy: 0.9758 - val_loss: 0.1229 - v
al_accuracy: 0.9653
Epoch 40/100
750/750 [==============================] - 3s 5ms/step - loss: 0.0850 - accuracy: 0.9762 - val_loss: 0.1225 - v
al_accuracy: 0.9653
Epoch 41/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0834 - accuracy: 0.9769 - val_loss: 0.1229 - v
al_accuracy: 0.9650
Epoch 42/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0814 - accuracy: 0.9776 - val_loss: 0.1214 - v
al_accuracy: 0.9649
Epoch 43/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0793 - accuracy: 0.9776 - val_loss: 0.1192 - v
al_accuracy: 0.9656
Epoch 44/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0778 - accuracy: 0.9781 - val_loss: 0.1190 - v
al_accuracy: 0.9665
Epoch 45/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0762 - accuracy: 0.9788 - val_loss: 0.1167 - v
al_accuracy: 0.9672
Epoch 46/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0745 - accuracy: 0.9791 - val_loss: 0.1163 - v
al_accuracy: 0.9670
Epoch 47/100
750/750 [==============================] - 3s 5ms/step - loss: 0.0729 - accuracy: 0.9795 - val_loss: 0.1155 - v
al_accuracy: 0.9665
Epoch 48/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0713 - accuracy: 0.9800 - val_loss: 0.1167 - v
al_accuracy: 0.9667
Epoch 49/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0701 - accuracy: 0.9808 - val_loss: 0.1134 - v
al_accuracy: 0.9667
Epoch 50/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0685 - accuracy: 0.9809 - val_loss: 0.1146 - v
al_accuracy: 0.9666
Epoch 51/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0671 - accuracy: 0.9813 - val_loss: 0.1131 - v
al_accuracy: 0.9671
Epoch 52/100
750/750 [==============================] - 2s 3ms/step - loss: 0.0655 - accuracy: 0.9815 - val_loss: 0.1130 - v
al_accuracy: 0.9668
Epoch 53/100
750/750 [==============================] - 2s 3ms/step - loss: 0.0643 - accuracy: 0.9821 - val_loss: 0.1134 - v
al_accuracy: 0.9667
Epoch 54/100
750/750 [==============================] - 2s 3ms/step - loss: 0.0632 - accuracy: 0.9828 - val_loss: 0.1138 - v
al_accuracy: 0.9674
Epoch 55/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0620 - accuracy: 0.9833 - val_loss: 0.1122 - v
al_accuracy: 0.9677
Epoch 56/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0607 - accuracy: 0.9834 - val_loss: 0.1114 - v
al_accuracy: 0.9672
Epoch 57/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0597 - accuracy: 0.9835 - val_loss: 0.1107 - v
al_accuracy: 0.9673
Epoch 58/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0582 - accuracy: 0.9840 - val_loss: 0.1114 - v
al_accuracy: 0.9679
Epoch 59/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0572 - accuracy: 0.9846 - val_loss: 0.1100 - v
al_accuracy: 0.9681
Epoch 60/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0562 - accuracy: 0.9848 - val_loss: 0.1106 - v
al_accuracy: 0.9678
Epoch 61/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0553 - accuracy: 0.9858 - val_loss: 0.1112 - v
al_accuracy: 0.9689
Epoch 62/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0544 - accuracy: 0.9855 - val_loss: 0.1098 - v
al_accuracy: 0.9688
Epoch 63/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0531 - accuracy: 0.9859 - val_loss: 0.1108 - v
al_accuracy: 0.9684
Epoch 64/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0522 - accuracy: 0.9863 - val_loss: 0.1108 - v
al_accuracy: 0.9685
Epoch 65/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0513 - accuracy: 0.9862 - val_loss: 0.1111 - v
```

```
al_accuracy: 0.9682
Epoch 66/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0504 - accuracy: 0.9866 - val_loss: 0.1106 - v
al_accuracy: 0.9690
Epoch 67/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0493 - accuracy: 0.9869 - val_loss: 0.1103 - v
al_accuracy: 0.9687
Epoch 68/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0485 - accuracy: 0.9872 - val_loss: 0.1097 - v
al_accuracy: 0.9691
Epoch 69/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0477 - accuracy: 0.9875 - val_loss: 0.1099 - v
al_accuracy: 0.9688
Epoch 70/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0465 - accuracy: 0.9878 - val_loss: 0.1105 - v
al_accuracy: 0.9683
Epoch 71/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0459 - accuracy: 0.9883 - val_loss: 0.1095 - v
al_accuracy: 0.9695
Epoch 72/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0452 - accuracy: 0.9881 - val_loss: 0.1092 - v
al_accuracy: 0.9687
Epoch 73/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0442 - accuracy: 0.9888 - val_loss: 0.1098 - v
al_accuracy: 0.9688
Epoch 74/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0434 - accuracy: 0.9891 - val_loss: 0.1080 - v
al_accuracy: 0.9689
Epoch 75/100
750/750 [==============================] - 2s 3ms/step - loss: 0.0426 - accuracy: 0.9891 - val_loss: 0.1088 - v
al_accuracy: 0.9691
Epoch 76/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0420 - accuracy: 0.9893 - val_loss: 0.1078 - v
al_accuracy: 0.9699
Epoch 77/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0412 - accuracy: 0.9893 - val_loss: 0.1076 - v
al_accuracy: 0.9695
Epoch 78/100
750/750 [==============================] - 3s 5ms/step - loss: 0.0404 - accuracy: 0.9899 - val_loss: 0.1088 - v
al_accuracy: 0.9696
Epoch 79/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0398 - accuracy: 0.9899 - val_loss: 0.1086 - v
al_accuracy: 0.9692
Epoch 80/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0389 - accuracy: 0.9905 - val_loss: 0.1086 - v
al_accuracy: 0.9706
Epoch 81/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0384 - accuracy: 0.9905 - val_loss: 0.1084 - v
al_accuracy: 0.9701
Epoch 82/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0375 - accuracy: 0.9910 - val_loss: 0.1087 - v
al_accuracy: 0.9695
Epoch 83/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0370 - accuracy: 0.9910 - val_loss: 0.1083 - v
al_accuracy: 0.9701
Epoch 84/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0363 - accuracy: 0.9909 - val_loss: 0.1082 - v
al_accuracy: 0.9700
Epoch 85/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0356 - accuracy: 0.9915 - val_loss: 0.1069 - v
al_accuracy: 0.9707
Epoch 86/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0350 - accuracy: 0.9918 - val_loss: 0.1099 - v
al_accuracy: 0.9699
Epoch 87/100
750/750 [==============================] - 2s 3ms/step - loss: 0.0345 - accuracy: 0.9918 - val_loss: 0.1095 - v
al_accuracy: 0.9700
Epoch 88/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0338 - accuracy: 0.9920 - val_loss: 0.1089 - v
al_accuracy: 0.9693
Epoch 89/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0332 - accuracy: 0.9922 - val_loss: 0.1088 - v
al_accuracy: 0.9704
Epoch 90/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0325 - accuracy: 0.9927 - val_loss: 0.1085 - v
al_accuracy: 0.9703
Epoch 91/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0322 - accuracy: 0.9923 - val_loss: 0.1086 - v
al_accuracy: 0.9703
Epoch 92/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0316 - accuracy: 0.9925 - val_loss: 0.1088 - v
al_accuracy: 0.9704
Epoch 93/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0309 - accuracy: 0.9928 - val_loss: 0.1096 - v
al_accuracy: 0.9698
Epoch 94/100
750/750 [==============================] - 4s 5ms/step - loss: 0.0304 - accuracy: 0.9927 - val_loss: 0.1101 - v
al_accuracy: 0.9698
Epoch 95/100
```

```
750/750 [==============================] - 3s 4ms/step - loss: 0.0299 - accuracy: 0.9934 - val_loss: 0.1095 - v
al_accuracy: 0.9703
Epoch 96/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0294 - accuracy: 0.9934 - val_loss: 0.1092 - v
al_accuracy: 0.9706
Epoch 97/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0287 - accuracy: 0.9938 - val_loss: 0.1098 - v
al_accuracy: 0.9704
Epoch 98/100
750/750 [==============================] - 3s 3ms/step - loss: 0.0283 - accuracy: 0.9936 - val_loss: 0.1108 - v
al_accuracy: 0.9704
Epoch 99/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0279 - accuracy: 0.9939 - val_loss: 0.1106 - v
al_accuracy: 0.9701
Epoch 100/100
750/750 [==============================] - 3s 4ms/step - loss: 0.0273 - accuracy: 0.9936 - val_loss: 0.1096 - v
al_accuracy: 0.9706
```

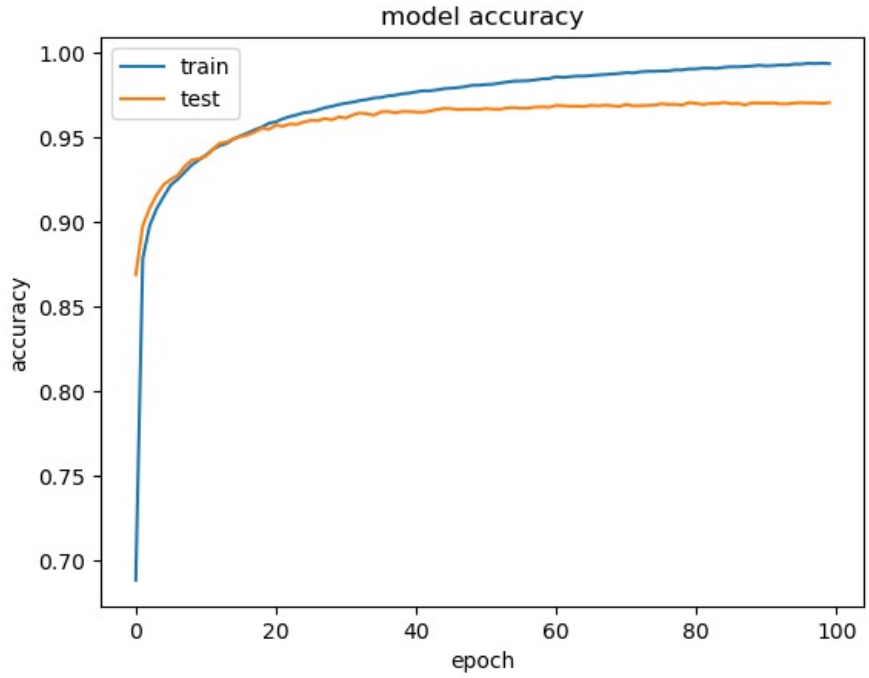
Plot the change in accuracy and loss per epochs

Plotting the change in metrices per epochs using matplotlib

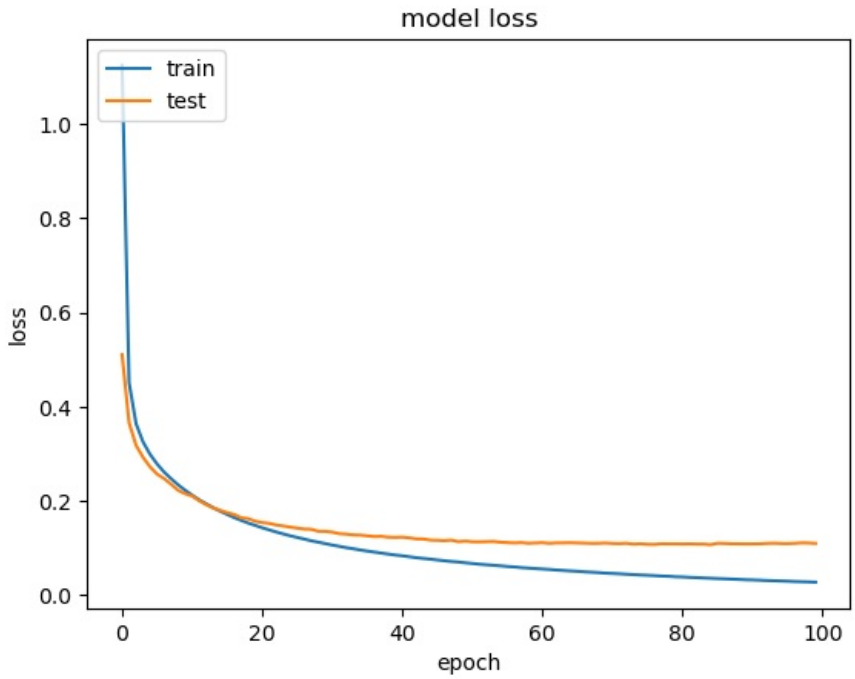
In [22]:
```
# list all data in training
print(training.history.keys())
```

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

In [23]:
```
# summarize training for accuracy
plt.plot(training.history['accuracy'])
plt.plot(training.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [24]:
```
# summarize traning for loss
plt.plot(training.history['loss'])
plt.plot(training.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Model 2 – Evaluate the Handwritten Digit Recognition Model on Test Data

In [25]:
```
#evaluate the model
test_loss, test_acc = model_1.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2824 - accuracy: 0.9216
Test accuracy: 0.9215999841690063
```

Improved Model 3 – Adding Dropout to Avoid Overfitting

In this new improved model we will be adding an dropout of 0.3 to avoid the overfitting.

In [26]:
```
from tensorflow.keras.layers import Dropout

#Most common type of model is a stack of layers
model_3 = tf.keras.Sequential()
# Adds a densely-connected layer with 64 units to the model:
N_hidden = 128


# Now the model will take as input arrays of shape (*, 784)# and output arrays of shape (*, 64)
model_3.add(Dense(N_hidden, name='dense_layer', input_shape=(784,), activation = 'relu'))
```

In [27]:
```
#Adding a dropout layer to avoid the overfitting</strong>
model_3.add(Dropout(0.3))
```

In [28]:
```
# Adding another dense layer:
model_3.add(Dense(N_hidden, name='dense_layer_2', activation='relu'))
model_3.add(Dropout(0.3))
```

In [29]:
```
# Add an output layer with 10 output units (10 different classes):
model_3.add(Dense(10, name='dense_layer_3', activation = 'softmax'))
```

In [30]:
```
# Compiling the model.
model_3.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [31]:
```
# Training the model.
training = model_3.fit(X_train, Y_train, batch_size=64, epochs=50, validation_split=0.2)
```

```
Epoch 1/50
750/750 [==============================] - 5s 6ms/step - loss: 1.3047 - accuracy: 0.5905 - val_loss: 0.5375 - v
al_accuracy: 0.8666
Epoch 2/50
750/750 [==============================] - 3s 5ms/step - loss: 0.6430 - accuracy: 0.8018 - val_loss: 0.3731 - v
al_accuracy: 0.8988
Epoch 3/50
750/750 [==============================] - 5s 7ms/step - loss: 0.5130 - accuracy: 0.8461 - val_loss: 0.3133 - v
al_accuracy: 0.9113
```

```
Epoch 4/50
750/750 [==============================] - 4s 6ms/step - loss: 0.4417 - accuracy: 0.8690 - val_loss: 0.2787 - v
al_accuracy: 0.9187
Epoch 5/50
750/750 [==============================] - 4s 5ms/step - loss: 0.3992 - accuracy: 0.8827 - val_loss: 0.2545 - v
al_accuracy: 0.9262
Epoch 6/50
750/750 [==============================] - 4s 5ms/step - loss: 0.3656 - accuracy: 0.8916 - val_loss: 0.2350 - v
al_accuracy: 0.9322
Epoch 7/50
750/750 [==============================] - 4s 5ms/step - loss: 0.3435 - accuracy: 0.8982 - val_loss: 0.2196 - v
al_accuracy: 0.9366
Epoch 8/50
750/750 [==============================] - 4s 5ms/step - loss: 0.3218 - accuracy: 0.9055 - val_loss: 0.2063 - v
al_accuracy: 0.9414
Epoch 9/50
750/750 [==============================] - 4s 5ms/step - loss: 0.3043 - accuracy: 0.9100 - val_loss: 0.1962 - v
al_accuracy: 0.9441
Epoch 10/50
750/750 [==============================] - 3s 5ms/step - loss: 0.2879 - accuracy: 0.9160 - val_loss: 0.1875 - v
al_accuracy: 0.9472
Epoch 11/50
750/750 [==============================] - 3s 5ms/step - loss: 0.2762 - accuracy: 0.9194 - val_loss: 0.1813 - v
al_accuracy: 0.9482
Epoch 12/50
750/750 [==============================] - 3s 5ms/step - loss: 0.2627 - accuracy: 0.9217 - val_loss: 0.1734 - v
al_accuracy: 0.9507
Epoch 13/50
750/750 [==============================] - 3s 5ms/step - loss: 0.2562 - accuracy: 0.9244 - val_loss: 0.1675 - v
al_accuracy: 0.9528
Epoch 14/50
750/750 [==============================] - 3s 5ms/step - loss: 0.2453 - accuracy: 0.9277 - val_loss: 0.1624 - v
al_accuracy: 0.9539
Epoch 15/50
750/750 [==============================] - 4s 5ms/step - loss: 0.2396 - accuracy: 0.9298 - val_loss: 0.1569 - v
al_accuracy: 0.9556
Epoch 16/50
750/750 [==============================] - 3s 5ms/step - loss: 0.2308 - accuracy: 0.9318 - val_loss: 0.1530 - v
al_accuracy: 0.9558
Epoch 17/50
750/750 [==============================] - 4s 5ms/step - loss: 0.2240 - accuracy: 0.9331 - val_loss: 0.1482 - v
al_accuracy: 0.9572
Epoch 18/50
750/750 [==============================] - 4s 5ms/step - loss: 0.2176 - accuracy: 0.9352 - val_loss: 0.1447 - v
al_accuracy: 0.9571
Epoch 19/50
750/750 [==============================] - 4s 5ms/step - loss: 0.2109 - accuracy: 0.9372 - val_loss: 0.1417 - v
al_accuracy: 0.9582
Epoch 20/50
750/750 [==============================] - 4s 5ms/step - loss: 0.2061 - accuracy: 0.9396 - val_loss: 0.1384 - v
al_accuracy: 0.9597
Epoch 21/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1999 - accuracy: 0.9408 - val_loss: 0.1346 - v
al_accuracy: 0.9607
Epoch 22/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1936 - accuracy: 0.9445 - val_loss: 0.1319 - v
al_accuracy: 0.9618
Epoch 23/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1875 - accuracy: 0.9442 - val_loss: 0.1295 - v
al_accuracy: 0.9625
Epoch 24/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1873 - accuracy: 0.9439 - val_loss: 0.1259 - v
al_accuracy: 0.9631
Epoch 25/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1797 - accuracy: 0.9483 - val_loss: 0.1246 - v
al_accuracy: 0.9638
Epoch 26/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1792 - accuracy: 0.9469 - val_loss: 0.1228 - v
al_accuracy: 0.9643
Epoch 27/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1724 - accuracy: 0.9489 - val_loss: 0.1204 - v
al_accuracy: 0.9645
Epoch 28/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1703 - accuracy: 0.9500 - val_loss: 0.1186 - v
al_accuracy: 0.9652
Epoch 29/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1682 - accuracy: 0.9504 - val_loss: 0.1176 - v
al_accuracy: 0.9654
Epoch 30/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1640 - accuracy: 0.9522 - val_loss: 0.1149 - v
al_accuracy: 0.9669
Epoch 31/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1627 - accuracy: 0.9523 - val_loss: 0.1133 - v
al_accuracy: 0.9667
Epoch 32/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1611 - accuracy: 0.9528 - val_loss: 0.1113 - v
al_accuracy: 0.9675
Epoch 33/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1524 - accuracy: 0.9540 - val_loss: 0.1102 - v
```

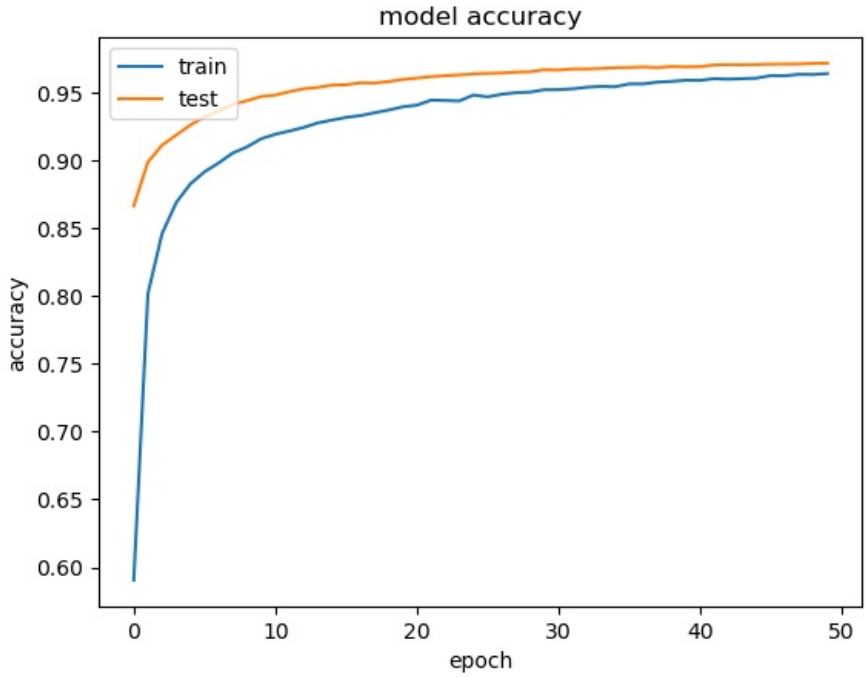
```
al_accuracy: 0.9674
Epoch 34/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1520 - accuracy: 0.9546 - val_loss: 0.1093 - v
al_accuracy: 0.9679
Epoch 35/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1514 - accuracy: 0.9544 - val_loss: 0.1081 - v
al_accuracy: 0.9684
Epoch 36/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1449 - accuracy: 0.9565 - val_loss: 0.1071 - v
al_accuracy: 0.9686
Epoch 37/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1451 - accuracy: 0.9565 - val_loss: 0.1063 - v
al_accuracy: 0.9689
Epoch 38/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1417 - accuracy: 0.9578 - val_loss: 0.1052 - v
al_accuracy: 0.9685
Epoch 39/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1401 - accuracy: 0.9583 - val_loss: 0.1034 - v
al_accuracy: 0.9693
Epoch 40/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1373 - accuracy: 0.9591 - val_loss: 0.1023 - v
al_accuracy: 0.9690
Epoch 41/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1384 - accuracy: 0.9590 - val_loss: 0.1016 - v
al_accuracy: 0.9692
Epoch 42/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1324 - accuracy: 0.9604 - val_loss: 0.1015 - v
al_accuracy: 0.9704
Epoch 43/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1327 - accuracy: 0.9600 - val_loss: 0.0996 - v
al_accuracy: 0.9707
Epoch 44/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1319 - accuracy: 0.9604 - val_loss: 0.0989 - v
al_accuracy: 0.9706
Epoch 45/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1286 - accuracy: 0.9607 - val_loss: 0.0984 - v
al_accuracy: 0.9707
Epoch 46/50
750/750 [==============================] - 4s 5ms/step - loss: 0.1265 - accuracy: 0.9626 - val_loss: 0.0975 - v
al_accuracy: 0.9710
Epoch 47/50
750/750 [==============================] - 3s 5ms/step - loss: 0.1261 - accuracy: 0.9624 - val_loss: 0.0968 - v
al_accuracy: 0.9711
Epoch 48/50
750/750 [==============================] - 4s 6ms/step - loss: 0.1232 - accuracy: 0.9635 - val_loss: 0.0965 - v
al_accuracy: 0.9712
Epoch 49/50
750/750 [==============================] - 6s 7ms/step - loss: 0.1224 - accuracy: 0.9634 - val_loss: 0.0954 - v
al_accuracy: 0.9716
Epoch 50/50
750/750 [==============================] - 5s 7ms/step - loss: 0.1211 - accuracy: 0.9641 - val_loss: 0.0957 - v
al_accuracy: 0.9718
```

Model 3 – Plot the change in metrices per epochs

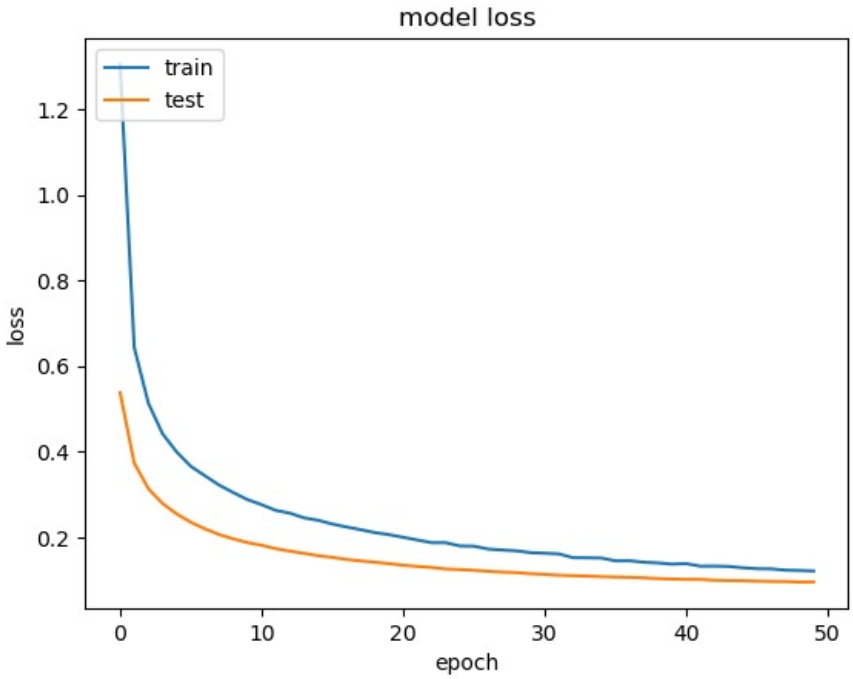
In [32]:
```
# list all data in training
print(training.history.keys())
```

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

In [33]:
```
# summarize training for accuracy
plt.plot(training.history['accuracy'])
plt.plot(training.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [34]:
```
# summarize traning for loss
plt.plot(training.history['loss'])
plt.plot(training.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Model 3 – Evaluate the Handwritten Digit Recognition Model on Test Data

In [35]:
```
#evaluate the model
test_loss, test_acc = model_2.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0956 - accuracy: 0.9719
Test accuracy: 0.9718999862670898
```

Improved Model 4: Increasing the number of Hidden Layer neuron

In [36]:
```
#Most common type of model is a stack of layers
model_4 = tf.keras.Sequential()
N_hidden = 512
```

In [37]:
```
# Adds a densely-connected layer with 64 units to the model:
model_4.add(Dense(N_hidden, name='dense_layer', input_shape=(784,), activation = 'relu'))
# Now the model will take as input arrays of shape (*, 784)# and output arrays of shape (*, 64)
model_4.add(Dropout(0.3))
```

In [38]:
```
# Adding another dense layer:
model_4.add(Dense(N_hidden, name='dense_layer_2', activation='relu'))
model_4.add(Dropout(0.3))
# After the first layer, you don't need to specify# the size of the input anymore:
# Add an output layer with 10 output units (10 different classes):
model_4.add(Dense(10, name='dense_layer_3', activation = 'softmax'))
```

In [39]:
```
# Compiling the model.
model_4.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Training the model.
training = model_4.fit(X_train, Y_train, batch_size=128, epochs=31, validation_split=0.2)
```

```
Epoch 1/31
375/375 [==============================] - 13s 31ms/step - loss: 0.3015 - accuracy: 0.9079 - val_loss: 0.1247 -
val_accuracy: 0.9628
Epoch 2/31
375/375 [==============================] - 11s 30ms/step - loss: 0.1268 - accuracy: 0.9611 - val_loss: 0.0978 -
val_accuracy: 0.9700
Epoch 3/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0910 - accuracy: 0.9714 - val_loss: 0.0852 -
val_accuracy: 0.9756
Epoch 4/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0716 - accuracy: 0.9778 - val_loss: 0.0806 -
val_accuracy: 0.9766
Epoch 5/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0580 - accuracy: 0.9811 - val_loss: 0.0799 -
val_accuracy: 0.9772
Epoch 6/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0521 - accuracy: 0.9832 - val_loss: 0.0863 -
val_accuracy: 0.9762
Epoch 7/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0471 - accuracy: 0.9846 - val_loss: 0.0808 -
val_accuracy: 0.9787
Epoch 8/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0406 - accuracy: 0.9868 - val_loss: 0.0797 -
val_accuracy: 0.9797
Epoch 9/31
375/375 [==============================] - 12s 31ms/step - loss: 0.0386 - accuracy: 0.9864 - val_loss: 0.0789 -
val_accuracy: 0.9789
Epoch 10/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0327 - accuracy: 0.9892 - val_loss: 0.0826 -
val_accuracy: 0.9788
Epoch 11/31
375/375 [==============================] - 13s 34ms/step - loss: 0.0303 - accuracy: 0.9899 - val_loss: 0.0920 -
val_accuracy: 0.9785
Epoch 12/31
375/375 [==============================] - 13s 35ms/step - loss: 0.0320 - accuracy: 0.9894 - val_loss: 0.0917 -
val_accuracy: 0.9782
Epoch 13/31
375/375 [==============================] - 14s 39ms/step - loss: 0.0280 - accuracy: 0.9902 - val_loss: 0.0828 -
val_accuracy: 0.9798
Epoch 14/31
375/375 [==============================] - 14s 37ms/step - loss: 0.0253 - accuracy: 0.9922 - val_loss: 0.0928 -
val_accuracy: 0.9782
Epoch 15/31
375/375 [==============================] - 12s 32ms/step - loss: 0.0263 - accuracy: 0.9913 - val_loss: 0.0859 -
val_accuracy: 0.9799
Epoch 16/31
375/375 [==============================] - 12s 31ms/step - loss: 0.0248 - accuracy: 0.9913 - val_loss: 0.0939 -
val_accuracy: 0.9797
Epoch 17/31
375/375 [==============================] - 11s 29ms/step - loss: 0.0253 - accuracy: 0.9915 - val_loss: 0.0884 -
val_accuracy: 0.9803
Epoch 18/31
375/375 [==============================] - 11s 31ms/step - loss: 0.0233 - accuracy: 0.9925 - val_loss: 0.0913 -
val_accuracy: 0.9810
Epoch 19/31
375/375 [==============================] - 12s 31ms/step - loss: 0.0225 - accuracy: 0.9921 - val_loss: 0.0954 -
val_accuracy: 0.9797
Epoch 20/31
375/375 [==============================] - 12s 33ms/step - loss: 0.0212 - accuracy: 0.9927 - val_loss: 0.1010 -
val_accuracy: 0.9807
Epoch 21/31
375/375 [==============================] - 12s 33ms/step - loss: 0.0219 - accuracy: 0.9930 - val_loss: 0.0942 -
val_accuracy: 0.9819
Epoch 22/31
375/375 [==============================] - 14s 37ms/step - loss: 0.0211 - accuracy: 0.9932 - val_loss: 0.0881 -
val_accuracy: 0.9812
Epoch 23/31
375/375 [==============================] - 12s 33ms/step - loss: 0.0181 - accuracy: 0.9939 - val_loss: 0.0957 -
val_accuracy: 0.9818
```

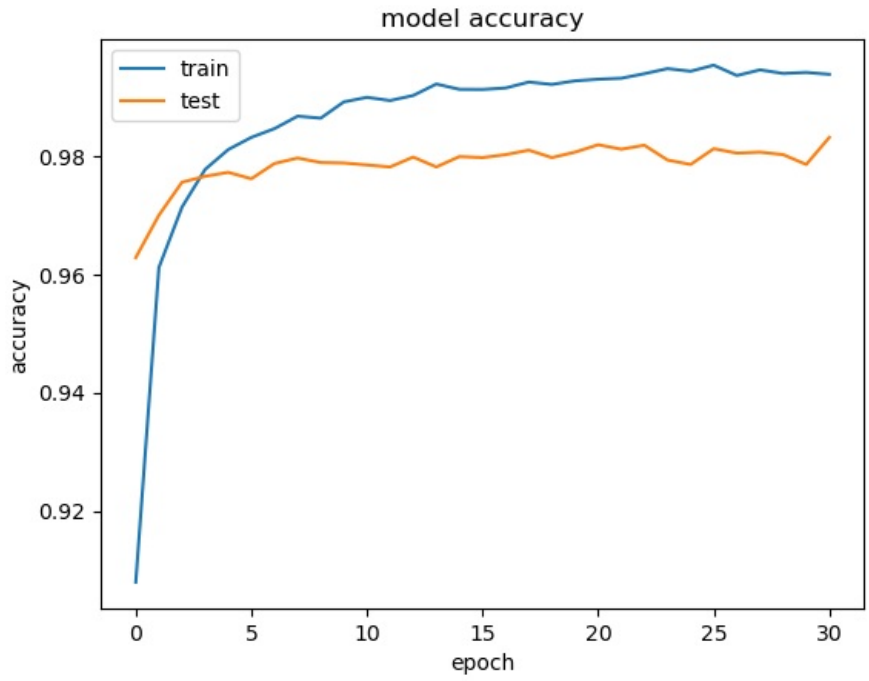
```
Epoch 24/31
375/375 [==============================] - 12s 32ms/step - loss: 0.0157 - accuracy: 0.9948 - val_loss: 0.1096 -
val_accuracy: 0.9793
Epoch 25/31
375/375 [==============================] - 15s 39ms/step - loss: 0.0169 - accuracy: 0.9944 - val_loss: 0.1065 -
val_accuracy: 0.9786
Epoch 26/31
375/375 [==============================] - 12s 32ms/step - loss: 0.0141 - accuracy: 0.9954 - val_loss: 0.0978 -
val_accuracy: 0.9812
Epoch 27/31
375/375 [==============================] - 12s 31ms/step - loss: 0.0200 - accuracy: 0.9936 - val_loss: 0.1097 -
val_accuracy: 0.9805
Epoch 28/31
375/375 [==============================] - 11s 30ms/step - loss: 0.0164 - accuracy: 0.9946 - val_loss: 0.1021 -
val_accuracy: 0.9807
Epoch 29/31
375/375 [==============================] - 11s 29ms/step - loss: 0.0184 - accuracy: 0.9940 - val_loss: 0.1035 -
val_accuracy: 0.9803
Epoch 30/31
375/375 [==============================] - 11s 29ms/step - loss: 0.0177 - accuracy: 0.9941 - val_loss: 0.1318 -
val_accuracy: 0.9786
Epoch 31/31
375/375 [==============================] - 12s 32ms/step - loss: 0.0189 - accuracy: 0.9938 - val_loss: 0.0974 -
val_accuracy: 0.9832
```

Model 4 – Plot the change in metrices per epochs

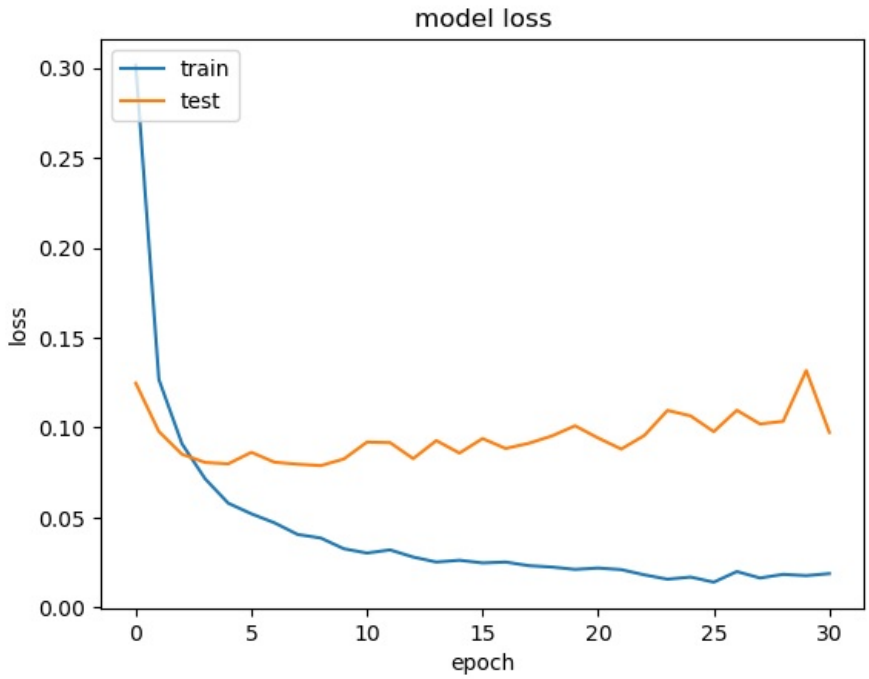
In [40]:
```
# list all data in training
print(training.history.keys())
```

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

In [41]:
```
# summarize training for accuracy
plt.plot(training.history['accuracy'])
plt.plot(training.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [42]:
```
# summarize traning for loss
plt.plot(training.history['loss'])
plt.plot(training.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Model 4 – Evaluate the Handwritten Digit Recognition Model on Test Data

In [43]:
```
#evaluate the model_4
test_loss, test_acc = model_4.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

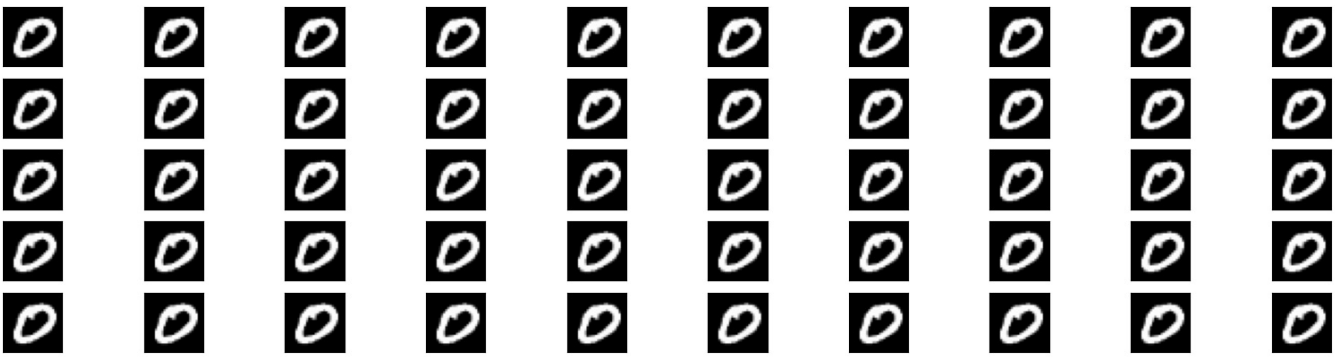
```
313/313 [==============================] - 2s 5ms/step - loss: 0.0814 - accuracy: 0.9826
Test accuracy: 0.9825999736785889
```

Task 2:- Classify a given image of a handwritten digit into one of the 10 classes representing integer values from 0 to 9.

In [44]:
```
im=mnist.load_data()
import numpy as np
```

Visualize the First 24 Training Images

In [49]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(20,5))
for i in range(50):
    ax = fig.add_subplot(5, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(X_test[idx].reshape(28, 28), cmap='gray')
```



In [46]:
```
# get predictions on the test set
y_hat = model_4.predict(X_test)
```

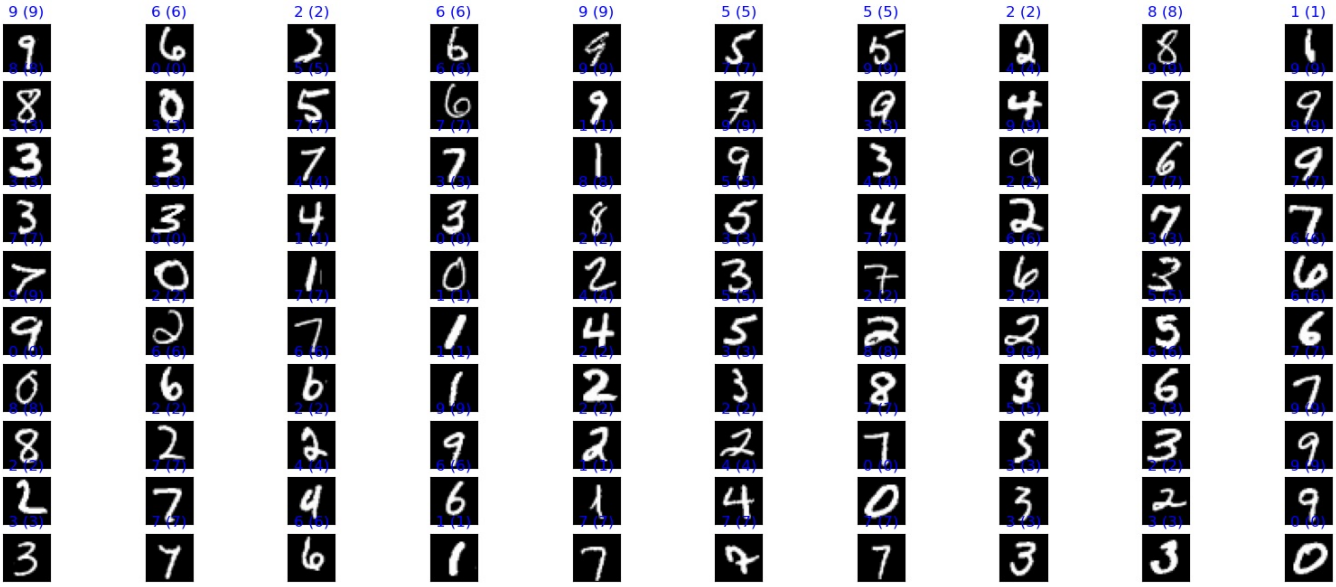
```
313/313 [==============================] - 1s 4ms/step
```

In [47]:
```
# define text labels (source: https://www.cs.toronto.edu/~kriz/cifar.html)
cifar10_labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Visualize Some Predictions

Visualize Some Predictions

In [48]:
```
# plot a random sample of test images, their predicted labels, and ground truth
fig = plt.figure(figsize=(20, 8))
for i, idx in enumerate(np.random.choice(X_test.shape[0], size=100, replace=False)):
    ax = fig.add_subplot(10, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    pred_idx = np.argmax(y_hat[idx])
    true_idx = np.argmax(Y_test[idx])
    ax.set_title("{} ({})".format(cifar10_labels[pred_idx], cifar10_labels[true_idx]),
                 color=("blue" if pred_idx == true_idx else "red"))
```



Model Comparison Report

- For Our Handwritten Digits Recognition Dataset We Have Attempted Simple Neural Network,Convolutional Neural Network,Recurrent Neural Network In This Three Types we Have Seen Similarities
- So We Are Suggesting Convolutional Neural Network Because By Using This Model We Can Abled To Get 98% Accuracy

Report on Challenges faced

The challenges we faced during this project are as follows:

- The HandWritten DataSet Is a Huge Dataset with 60,000 training and 10,000 testing labelled handwritten digit pictures.
- Its Very Difficult To Get 98% Accuracy For That Only We Have Builded Four Models And After That We Have Gone Through Prediction
- In Prediction The Challenge Is To Show 0-9 Numbers We Have Written a Code For That
- While Doing Project If We Come To Task Means We Have Builded First Model Building after That We Have Done Prediction

Conclusion

the handwritten digit recognition model trained on the MNIST dataset has demonstrated exceptional performance with an accuracy of 98%. Achieving such high accuracy indicates that the model has effectively learned to distinguish between different handwritten digits. The success of this model highlights the effectiveness of the chosen architecture, training approach, and hyperparameters.

The high accuracy suggests that the model generalizes well to unseen data, making it a reliable tool for recognizing handwritten digits in various applications. It could be deployed in digit recognition tasks for forms, checks, or any scenario where automated recognition of handwritten numerical input is required.

However, it's essential to acknowledge that achieving even higher accuracy might be challenging and could require more sophisticated architectures or additional data. Additionally, model evaluation should not solely rely on accuracy; other metrics like precision, recall, and F1 score should be considered, especially if there is an imbalance in the distribution of digits.

In summary, the 98% accuracy achieved in this handwritten digit recognition task is a notable accomplishment, demonstrating the effectiveness of the chosen machine learning approach and the model's ability to generalize well to new handwritten digit samples.

In [ ]: