# Activity 1: Nhận diện chữ số viết tay (MNIST Dataset)

#### Khai báo thư viện sử dụng

In [ ]:

```
from tensorflow.keras.datasets import mnist
import numpy as np
from matplotlib import pyplot as plt
import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input, Reshape
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

#### Load data

In [ ]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape, X_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.n
pz
11490434/11490434 [==============================] - 0s 0us/step
(60000, 28, 28) (10000, 28, 28)
```

#### Chuẩn hóa dữ liệu input

In [ ]:

```
X_train_scaled = np.array(X_train)/255.
X_test_scaled = np.array(X_test)/255.
```

#### Chuẩn hóa dữ liệu output

In [ ]:

```
# OnehotVector output
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
encoder.fit(y_train.reshape(-1,1))
y_train = encoder.transform(y_train.reshape(-1,1)).toarray()
y_test = encoder.transform(y_test.reshape(-1,1)).toarray()
```

#### Xây dựng mô hình CNN ban đầu

In [ ]:

```
# CNN model
inp = Input(shape = (28,28,1)) # input shape
cnn = Conv2D(filters = 8, kernel_size = 3, activation='relu')(inp)
pooling = MaxPooling2D(pool_size=(2,2))(cnn)
drop = Dropout(0.2)(pooling)

cnn = Conv2D(filters = 16, kernel_size = 4, activation='relu')(drop)
pooling = MaxPooling2D(pool_size=(2,2))(cnn)
drop = Dropout(0.2)(pooling)

cnn = Conv2D(filters = 32, kernel_size = 4, activation='relu')(drop)
pooling = MaxPooling2D(pool_size=(2,2))(cnn)

f = Flatten()(pooling)
```

```
fc1 = Dense(units = 32, activation = 'relu')(f)
fc2 = Dense(units = 16, activation = 'relu')(fc1)
out = Dense(units = 10, activation = 'softmax')(fc2)

model = Model(inputs = inp, outputs = out)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0         
                                                                 
 conv2d (Conv2D)             (None, 26, 26, 8)         80        
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 8)        0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 13, 13, 8)         0         
                                                                 
 conv2d_1 (Conv2D)           (None, 10, 10, 16)        2064      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 16)         0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 5, 5, 16)          0         
                                                                 
 conv2d_2 (Conv2D)           (None, 2, 2, 32)          8224      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 1, 1, 32)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 32)                0         
                                                                 
 dense (Dense)               (None, 32)                1056      
                                                                 
 dense_1 (Dense)             (None, 16)                528       
                                                                 
 dense_2 (Dense)             (None, 10)                170       
                                                                 
=================================================================
Total params: 12,122
Trainable params: 12,122
Non-trainable params: 0
_________________________________________________________________
```

### Huấn luyện mô hình ban đầu

In [ ]:

```
optimizer1 = tensorflow.keras.optimizers.Adam(learning_rate = 0.001)
model.compile(optimizer = optimizer1, loss='categorical_crossentropy',metrics = ['accura
cy'])

history = model.fit(X_train_scaled,y_train,batch_size=64,
                    epochs = 50, validation_data = (X_test_scaled, y_test))
```

```
Epoch 1/50
938/938 [==============================] - 23s 7ms/step - loss: 0.5440 - accuracy: 0.8188
- val_loss: 0.1270 - val_accuracy: 0.9635
Epoch 2/50
938/938 [==============================] - 6s 6ms/step - loss: 0.1699 - accuracy: 0.9472
- val_loss: 0.0851 - val_accuracy: 0.9745
Epoch 3/50
938/938 [==============================] - 7s 7ms/step - loss: 0.1259 - accuracy: 0.9606
- val_loss: 0.0794 - val_accuracy: 0.9747
Epoch 4/50
938/938 [==============================] - 7s 8ms/step - loss: 0.1018 - accuracy: 0.9682
- val_loss: 0.0536 - val_accuracy: 0.9838
Epoch 5/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0876 - accuracy: 0.9725
- val_loss: 0.0475 - val_accuracy: 0.9857
```

```
- val_loss: 0.0475 - val_accuracy: 0.9857
Epoch 6/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0788 - accuracy: 0.9751
- val_loss: 0.0417 - val_accuracy: 0.9861
Epoch 7/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0704 - accuracy: 0.9779
- val_loss: 0.0379 - val_accuracy: 0.9881
Epoch 8/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0671 - accuracy: 0.9790
- val_loss: 0.0373 - val_accuracy: 0.9878
Epoch 9/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0595 - accuracy: 0.9807
- val_loss: 0.0400 - val_accuracy: 0.9862
Epoch 10/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0580 - accuracy: 0.9817
- val_loss: 0.0353 - val_accuracy: 0.9878
Epoch 11/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0558 - accuracy: 0.9825
- val_loss: 0.0364 - val_accuracy: 0.9867
Epoch 12/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0542 - accuracy: 0.9823
- val_loss: 0.0381 - val_accuracy: 0.9878
Epoch 13/50
938/938 [==============================] - 8s 8ms/step - loss: 0.0489 - accuracy: 0.9843
- val_loss: 0.0307 - val_accuracy: 0.9899
Epoch 14/50
938/938 [==============================] - 6s 7ms/step - loss: 0.0503 - accuracy: 0.9840
- val_loss: 0.0285 - val_accuracy: 0.9903
Epoch 15/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0462 - accuracy: 0.9852
- val_loss: 0.0322 - val_accuracy: 0.9886
Epoch 16/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0464 - accuracy: 0.9854
- val_loss: 0.0302 - val_accuracy: 0.9894
Epoch 17/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0434 - accuracy: 0.9856
- val_loss: 0.0292 - val_accuracy: 0.9890
Epoch 18/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0435 - accuracy: 0.9862
- val_loss: 0.0326 - val_accuracy: 0.9898
Epoch 19/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0406 - accuracy: 0.9866
- val_loss: 0.0304 - val_accuracy: 0.9906
Epoch 20/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0410 - accuracy: 0.9866
- val_loss: 0.0261 - val_accuracy: 0.9920
Epoch 21/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0391 - accuracy: 0.9875
- val_loss: 0.0311 - val_accuracy: 0.9900
Epoch 22/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0400 - accuracy: 0.9870
- val_loss: 0.0307 - val_accuracy: 0.9890
Epoch 23/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0389 - accuracy: 0.9874
- val_loss: 0.0300 - val_accuracy: 0.9887
Epoch 24/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0358 - accuracy: 0.9885
- val_loss: 0.0311 - val_accuracy: 0.9884
Epoch 25/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0367 - accuracy: 0.9886
- val_loss: 0.0279 - val_accuracy: 0.9903
Epoch 26/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0363 - accuracy: 0.9881
- val_loss: 0.0271 - val_accuracy: 0.9920
Epoch 27/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0373 - accuracy: 0.9879
- val_loss: 0.0283 - val_accuracy: 0.9907
Epoch 28/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0341 - accuracy: 0.9884
- val_loss: 0.0287 - val_accuracy: 0.9903
Epoch 29/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0354 - accuracy: 0.9887
- val_loss: 0.0298 - val_accuracy: 0.9903
```

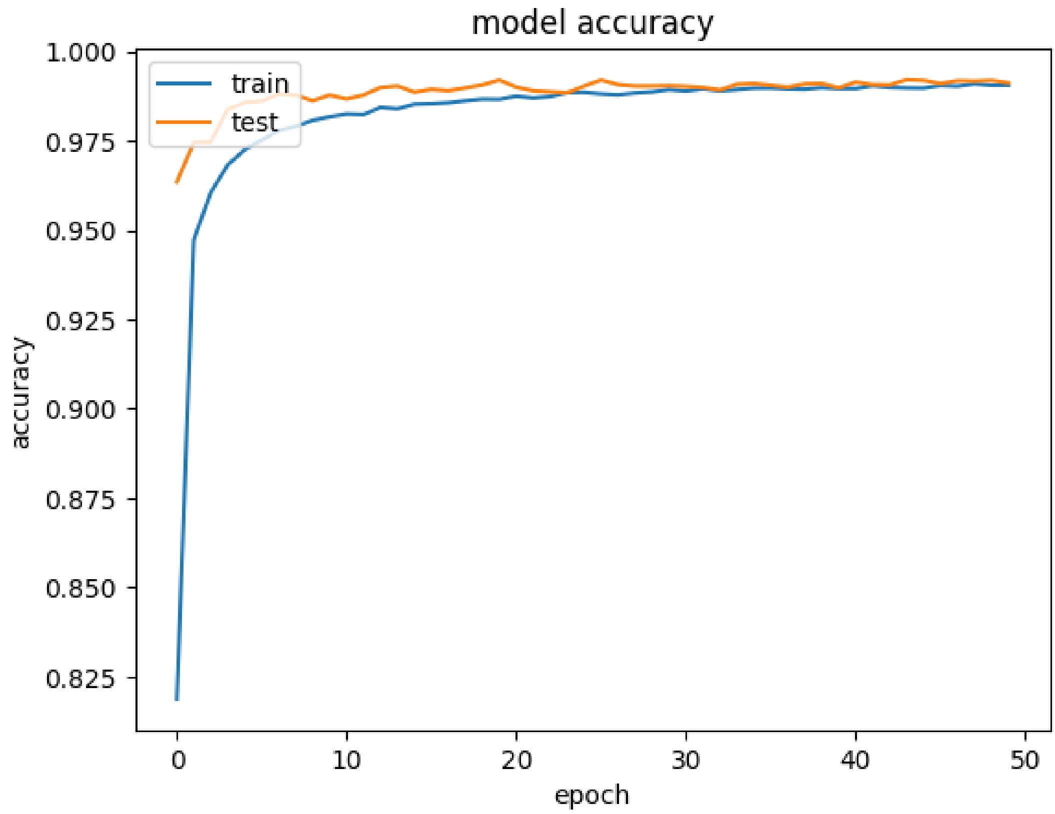
```
- val_loss: 0.0296 - val_accuracy: 0.9905
Epoch 30/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0337 - accuracy: 0.9893
- val_loss: 0.0289 - val_accuracy: 0.9904
Epoch 31/50
938/938 [==============================] - 8s 8ms/step - loss: 0.0333 - accuracy: 0.9889
- val_loss: 0.0295 - val_accuracy: 0.9902
Epoch 32/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0347 - accuracy: 0.9895
- val_loss: 0.0286 - val_accuracy: 0.9899
Epoch 33/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0342 - accuracy: 0.9890
- val_loss: 0.0293 - val_accuracy: 0.9892
Epoch 34/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0325 - accuracy: 0.9893
- val_loss: 0.0281 - val_accuracy: 0.9908
Epoch 35/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0327 - accuracy: 0.9897
- val_loss: 0.0275 - val_accuracy: 0.9910
Epoch 36/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0301 - accuracy: 0.9898
- val_loss: 0.0263 - val_accuracy: 0.9905
Epoch 37/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0309 - accuracy: 0.9895
- val_loss: 0.0313 - val_accuracy: 0.9900
Epoch 38/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0318 - accuracy: 0.9894
- val_loss: 0.0274 - val_accuracy: 0.9909
Epoch 39/50
938/938 [==============================] - 6s 7ms/step - loss: 0.0317 - accuracy: 0.9898
- val_loss: 0.0288 - val_accuracy: 0.9910
Epoch 40/50
938/938 [==============================] - 7s 7ms/step - loss: 0.0314 - accuracy: 0.9896
- val_loss: 0.0294 - val_accuracy: 0.9898
Epoch 41/50
938/938 [==============================] - 7s 7ms/step - loss: 0.0321 - accuracy: 0.9896
- val_loss: 0.0285 - val_accuracy: 0.9914
Epoch 42/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0286 - accuracy: 0.9904
- val_loss: 0.0308 - val_accuracy: 0.9907
Epoch 43/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0313 - accuracy: 0.9900
- val_loss: 0.0316 - val_accuracy: 0.9906
Epoch 44/50
938/938 [==============================] - 8s 8ms/step - loss: 0.0303 - accuracy: 0.9898
- val_loss: 0.0259 - val_accuracy: 0.9921
Epoch 45/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0311 - accuracy: 0.9897
- val_loss: 0.0269 - val_accuracy: 0.9919
Epoch 46/50
938/938 [==============================] - 7s 7ms/step - loss: 0.0291 - accuracy: 0.9905
- val_loss: 0.0261 - val_accuracy: 0.9910
Epoch 47/50
938/938 [==============================] - 6s 6ms/step - loss: 0.0305 - accuracy: 0.9902
- val_loss: 0.0271 - val_accuracy: 0.9918
Epoch 48/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0295 - accuracy: 0.9909
- val_loss: 0.0257 - val_accuracy: 0.9917
Epoch 49/50
938/938 [==============================] - 5s 6ms/step - loss: 0.0293 - accuracy: 0.9905
- val_loss: 0.0268 - val_accuracy: 0.9919
Epoch 50/50
938/938 [==============================] - 5s 5ms/step - loss: 0.0291 - accuracy: 0.9905
- val_loss: 0.0279 - val_accuracy: 0.9912
```

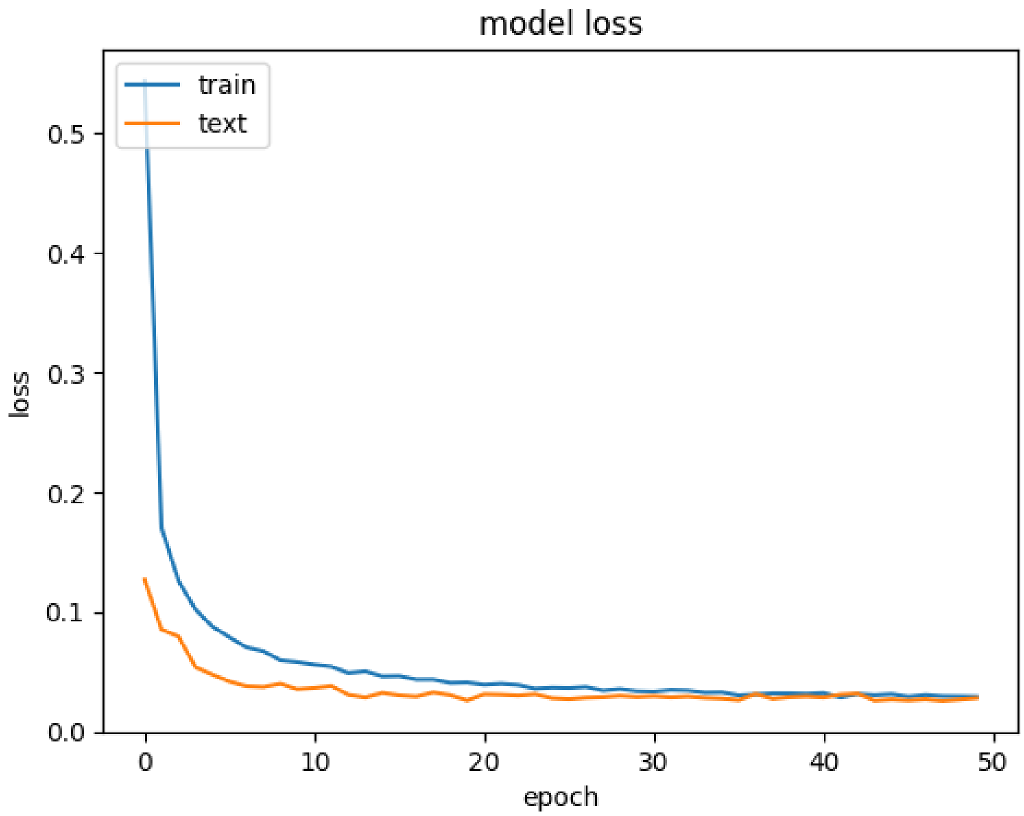
#### Trực quan hóa kết quả Accuracy và Loss trên tập Train và Test

In [ ]:

```
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','text'],loc='upper left')
plt.show()
```




**Lưu mô hình ban đầu, load mô hình đã lưu từ máy**

In [ ]:

```
model.save('model1.h5')
from tensorflow.keras.models import load_model
model1 = load_model('/content/model1.h5')
```

**Load ảnh và sử dụng mô hình đã huấn luyện để nhận diện**

In [ ]:

```
# import cv2
# img = cv2.imread('/content/so2.png')
# print(img.shape)
# plt.imshow(img)
```

In [ ]:

```
# gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# img_new = cv2.resize(gray_img,(28,28))
# print(gray_img.shape)
# plt.imshow(img_new,cmap = 'gray')
```

In [ ]:

```
# img_scaled = np.array([img_new/255.])
# print(img_scaled.shape)
# y_hat = model1.predict(img_scaled)
# print(y_hat)
# print(np.argmax(y_hat))
```

**Tính y dự đoán từ mô hình ban đầu đã lưu**

In [ ]:

```
y_hat = model1.predict(X_test_scaled)
```

```
313/313 [==============================] - 1s 2ms/step
```

In [ ]:

```
print(y_hat.shape)
print(y_test.shape)
```

```
(10000, 10)
(10000, 10)
```

**Lấy argmax của y dự đoán và y test**

In [ ]:

```
y_test = np.argmax(y_test, axis=1)
y_hat = np.argmax(y_hat, axis=1)
```

# Activity 2

## Sử dụng classification_report trong thư viện Sklearn đánh giá kết quả mô hình ban đầu dựa trên kết quả dự đoán tập test

In [ ]:

```
from sklearn.metrics import classification_report
target_names = ['0', '1', '2', '3','4','5','6','7','8','9']
print(classification_report(y_test, y_hat,target_names=target_names))
```

```
              precision    recall  f1-score   support
```

```
           0       0.99      1.00      1.00       980
           1       0.99      1.00      1.00      1135
           2       1.00      0.98      0.99      1032
           3       0.99      1.00      0.99      1010
           4       1.00      0.99      0.99       982
           5       0.98      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       1.00      0.99      0.99       974
           9       0.98      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

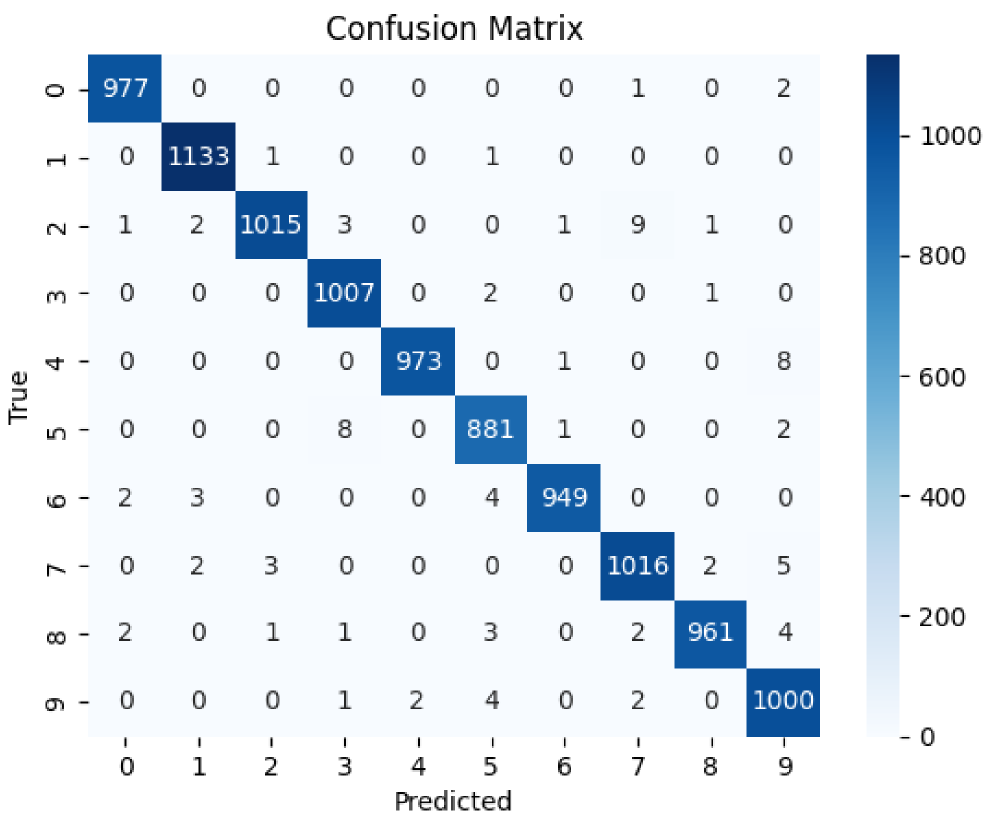
### Sử dụng Confusion_matrix trong thư viện Sklearn biểu diễn kết quả dự đoán trên tập test

In [ ]:

```
import sklearn.metrics
import seaborn as sn

# Tạo confusion matrix
cm = sklearn.metrics.confusion_matrix(y_test, y_hat)

# Vẽ confusion matrix
plt.figure()
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




#### Load data, chuẩn hoá input

In [ ]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
X_train_scaled = np.array(X_train)/255.
X_test_scaled = np.array(X_test)/255.
```

#### Chuẩn hoá output

In [ ]:

```
# OnehotVector output
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
encoder.fit(y_train.reshape(-1,1))
y_train = encoder.transform(y_train.reshape(-1,1)).toarray()
y_test = encoder.transform(y_test.reshape(-1,1)).toarray()
```

## Thay đổi mạng CNN theo ý kiến của riêng mình sao cho kết quả tốt nhất

In [ ]:

```
# Khai báo thư viện BatchNormalization
from tensorflow.keras.layers import BatchNormalization

inp = Input(shape = (28, 28, 1))

cnn = Conv2D(filters = 32, kernel_size = 3, activation = 'relu', padding = 'same')(inp)
# Thêm padding
cnn = BatchNormalization()(cnn)
cnn = Conv2D(filters = 32, kernel_size = 3, activation = 'relu', padding = 'same')(cnn)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2, 2))(cnn)

cnn = Conv2D(filters = 64, kernel_size = 3, activation = 'relu', padding = 'same')(pooli
ng)
cnn = BatchNormalization()(cnn)
cnn = Conv2D(filters = 64, kernel_size = 3, activation = 'relu', padding = 'same')(cnn)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2, 2))(cnn)

cnn = Conv2D(filters = 128, kernel_size = 3, activation = 'relu', padding = 'same')(pool
ing)
cnn = BatchNormalization()(cnn)
cnn = Conv2D(filters = 128, kernel_size = 3, activation = 'relu', padding = 'same')(cnn)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2, 2))(cnn)

cnn = Conv2D(filters = 256, kernel_size = 3, activation = 'relu', padding = 'same')(pool
ing)
cnn = BatchNormalization()(cnn)
cnn = Conv2D(filters = 256, kernel_size = 3, activation = 'relu', padding = 'same')(cnn)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2, 2))(cnn)

f = Flatten()(pooling)

fc1 = Dense(units = 512, activation = 'relu') (f)
drop = Dropout(0.2)(fc1)
fc2 = Dense(units = 256, activation = 'relu') (drop)
fc3 = Dense(units = 128, activation = 'relu') (fc2)
fc4 = Dense(units = 64, activation = 'relu') (fc3)
fc5 = Dense(units = 32, activation = 'relu') (fc4)
out = Dense(units = 10, activation = 'softmax') (fc5)

model = Model(inputs=inp, outputs=out)
model.summary()
```

```
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_14 (InputLayer)       [(None, 28, 28, 1)]       0         
```

```
 conv2d_43 (Conv2D)           (None, 28, 28, 32)        320

 batch_normalization (BatchN  (None, 28, 28, 32)        128
 ormalization)

 conv2d_44 (Conv2D)           (None, 28, 28, 32)        9248

 batch_normalization_1 (Batc  (None, 28, 28, 32)        128
 hNormalization)

 max_pooling2d_40 (MaxPoolin  (None, 14, 14, 32)        0
 g2D)

 conv2d_45 (Conv2D)           (None, 14, 14, 64)        18496

 batch_normalization_2 (Batc  (None, 14, 14, 64)        256
 hNormalization)

 conv2d_46 (Conv2D)           (None, 14, 14, 64)        36928

 batch_normalization_3 (Batc  (None, 14, 14, 64)        256
 hNormalization)

 max_pooling2d_41 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 conv2d_47 (Conv2D)           (None, 7, 7, 128)         73856

 batch_normalization_4 (Batc  (None, 7, 7, 128)         512
 hNormalization)

 conv2d_48 (Conv2D)           (None, 7, 7, 128)         147584

 batch_normalization_5 (Batc  (None, 7, 7, 128)         512
 hNormalization)

 max_pooling2d_42 (MaxPoolin  (None, 3, 3, 128)         0
 g2D)

 conv2d_49 (Conv2D)           (None, 3, 3, 256)         295168

 batch_normalization_6 (Batc  (None, 3, 3, 256)         1024
 hNormalization)

 conv2d_50 (Conv2D)           (None, 3, 3, 256)         590080

 batch_normalization_7 (Batc  (None, 3, 3, 256)         1024
 hNormalization)

 max_pooling2d_43 (MaxPoolin  (None, 1, 1, 256)         0
 g2D)

 flatten_9 (Flatten)          (None, 256)               0

 dense_35 (Dense)             (None, 512)               131584

 dropout_43 (Dropout)         (None, 512)               0

 dense_36 (Dense)             (None, 256)               131328

 dense_37 (Dense)             (None, 128)               32896

 dense_38 (Dense)             (None, 64)                8256

 dense_39 (Dense)             (None, 32)                2080

 dense_40 (Dense)             (None, 10)                330

=================================================================
Total params: 1,481,994
Trainable params: 1,480,074
Non-trainable params: 1,920
```

### Huấn luyện mô hình sau khi thay đổi cấu trúc mạng CNN

In [ ]:

```
optimizer1 = tensorflow.keras.optimizers.Adam(learning_rate = 0.001)
model.compile(optimizer = optimizer1, loss='categorical_crossentropy',metrics = ['accura
cy'])

history = model.fit(X_train_scaled,y_train,batch_size=64,
                    epochs = 50, validation_data = (X_test_scaled, y_test))
```

```
Epoch 1/50
938/938 [==============================] - 20s 13ms/step - loss: 0.1407 - accuracy: 0.961
0 - val_loss: 0.0402 - val_accuracy: 0.9885
Epoch 2/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0539 - accuracy: 0.986
5 - val_loss: 0.0574 - val_accuracy: 0.9839
Epoch 3/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0449 - accuracy: 0.988
8 - val_loss: 0.0534 - val_accuracy: 0.9858
Epoch 4/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0401 - accuracy: 0.990
4 - val_loss: 0.0550 - val_accuracy: 0.9871
Epoch 5/50
938/938 [==============================] - 12s 13ms/step - loss: 0.0330 - accuracy: 0.992
2 - val_loss: 0.0262 - val_accuracy: 0.9934
Epoch 6/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0279 - accuracy: 0.993
1 - val_loss: 0.0399 - val_accuracy: 0.9908
Epoch 7/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0279 - accuracy: 0.993
4 - val_loss: 0.0257 - val_accuracy: 0.9938
Epoch 8/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0241 - accuracy: 0.994
6 - val_loss: 0.0424 - val_accuracy: 0.9916
Epoch 9/50
938/938 [==============================] - 10s 11ms/step - loss: 0.0232 - accuracy: 0.995
2 - val_loss: 0.0479 - val_accuracy: 0.9909
Epoch 10/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0221 - accuracy: 0.995
1 - val_loss: 0.0419 - val_accuracy: 0.9920
Epoch 11/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0185 - accuracy: 0.995
7 - val_loss: 0.0340 - val_accuracy: 0.9931
Epoch 12/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0183 - accuracy: 0.996
5 - val_loss: 0.0398 - val_accuracy: 0.9913
Epoch 13/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0161 - accuracy: 0.996
4 - val_loss: 0.0286 - val_accuracy: 0.9939
Epoch 14/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0129 - accuracy: 0.996
9 - val_loss: 0.0374 - val_accuracy: 0.9924
Epoch 15/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0120 - accuracy: 0.997
5 - val_loss: 0.0381 - val_accuracy: 0.9919
Epoch 16/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0179 - accuracy: 0.996
2 - val_loss: 0.0358 - val_accuracy: 0.9929
Epoch 17/50
938/938 [==============================] - 10s 11ms/step - loss: 0.0099 - accuracy: 0.998
0 - val_loss: 0.0313 - val_accuracy: 0.9938
Epoch 18/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0119 - accuracy: 0.997
6 - val_loss: 0.0355 - val_accuracy: 0.9933
Epoch 19/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0138 - accuracy: 0.997
1 - val_loss: 0.0370 - val_accuracy: 0.9928
Epoch 20/50
```

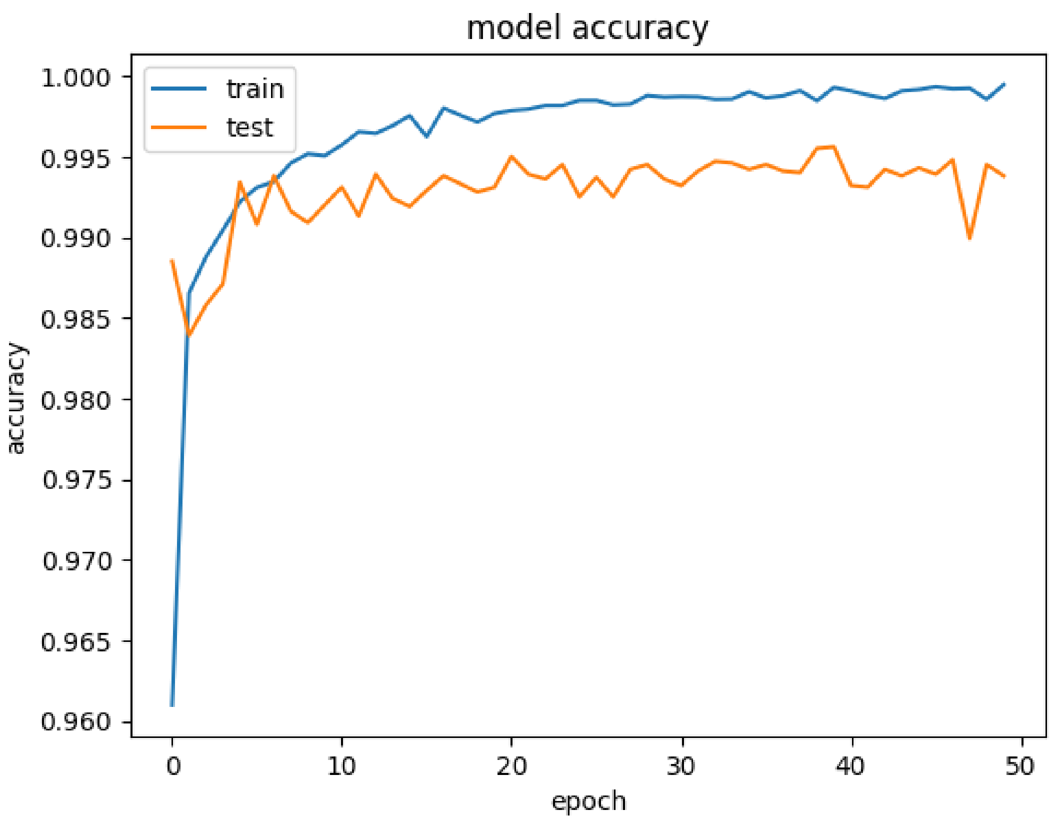
```
938/938 [==============================] - 11s 11ms/step - loss: 0.0107 - accuracy: 0.997
7 - val_loss: 0.0371 - val_accuracy: 0.9931
Epoch 21/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0102 - accuracy: 0.997
9 - val_loss: 0.0288 - val_accuracy: 0.9950
Epoch 22/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0097 - accuracy: 0.997
9 - val_loss: 0.0288 - val_accuracy: 0.9939
Epoch 23/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0085 - accuracy: 0.998
2 - val_loss: 0.0453 - val_accuracy: 0.9936
Epoch 24/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0108 - accuracy: 0.998
2 - val_loss: 0.0308 - val_accuracy: 0.9945
Epoch 25/50
938/938 [==============================] - 10s 11ms/step - loss: 0.0073 - accuracy: 0.998
5 - val_loss: 0.0535 - val_accuracy: 0.9925
Epoch 26/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0079 - accuracy: 0.998
5 - val_loss: 0.0491 - val_accuracy: 0.9937
Epoch 27/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0098 - accuracy: 0.998
2 - val_loss: 0.0743 - val_accuracy: 0.9925
Epoch 28/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0102 - accuracy: 0.998
3 - val_loss: 0.0488 - val_accuracy: 0.9942
Epoch 29/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0065 - accuracy: 0.998
8 - val_loss: 0.0381 - val_accuracy: 0.9945
Epoch 30/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0058 - accuracy: 0.998
7 - val_loss: 0.0482 - val_accuracy: 0.9936
Epoch 31/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0066 - accuracy: 0.998
7 - val_loss: 0.0471 - val_accuracy: 0.9932
Epoch 32/50
938/938 [==============================] - 10s 11ms/step - loss: 0.0078 - accuracy: 0.998
7 - val_loss: 0.0568 - val_accuracy: 0.9941
Epoch 33/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0069 - accuracy: 0.998
5 - val_loss: 0.0433 - val_accuracy: 0.9947
Epoch 34/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0072 - accuracy: 0.998
6 - val_loss: 0.0379 - val_accuracy: 0.9946
Epoch 35/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0046 - accuracy: 0.999
0 - val_loss: 0.0572 - val_accuracy: 0.9942
Epoch 36/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0068 - accuracy: 0.998
6 - val_loss: 0.0340 - val_accuracy: 0.9945
Epoch 37/50
938/938 [==============================] - 12s 13ms/step - loss: 0.0061 - accuracy: 0.998
8 - val_loss: 0.0395 - val_accuracy: 0.9941
Epoch 38/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0043 - accuracy: 0.999
1 - val_loss: 0.0482 - val_accuracy: 0.9940
Epoch 39/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0078 - accuracy: 0.998
5 - val_loss: 0.0358 - val_accuracy: 0.9955
Epoch 40/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0041 - accuracy: 0.999
3 - val_loss: 0.0390 - val_accuracy: 0.9956
Epoch 41/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0060 - accuracy: 0.999
1 - val_loss: 0.0422 - val_accuracy: 0.9932
Epoch 42/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0063 - accuracy: 0.998
8 - val_loss: 0.0509 - val_accuracy: 0.9931
Epoch 43/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0069 - accuracy: 0.998
6 - val_loss: 0.0519 - val_accuracy: 0.9942
Epoch 44/50
```

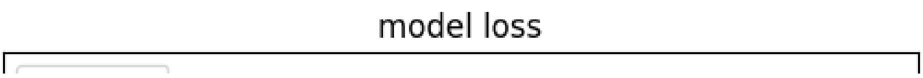
```
938/938 [==============================] - 11s 11ms/step - loss: 0.0057 - accuracy: 0.999
1 - val_loss: 0.0349 - val_accuracy: 0.9938
Epoch 45/50
938/938 [==============================] - 11s 12ms/step - loss: 0.0049 - accuracy: 0.999
1 - val_loss: 0.0507 - val_accuracy: 0.9943
Epoch 46/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0043 - accuracy: 0.999
3 - val_loss: 0.0640 - val_accuracy: 0.9939
Epoch 47/50
938/938 [==============================] - 12s 12ms/step - loss: 0.0044 - accuracy: 0.999
2 - val_loss: 0.0610 - val_accuracy: 0.9948
Epoch 48/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0048 - accuracy: 0.999
2 - val_loss: 0.1020 - val_accuracy: 0.9899
Epoch 49/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0080 - accuracy: 0.998
5 - val_loss: 0.0436 - val_accuracy: 0.9945
Epoch 50/50
938/938 [==============================] - 11s 11ms/step - loss: 0.0024 - accuracy: 0.999
5 - val_loss: 0.0608 - val_accuracy: 0.9938
```

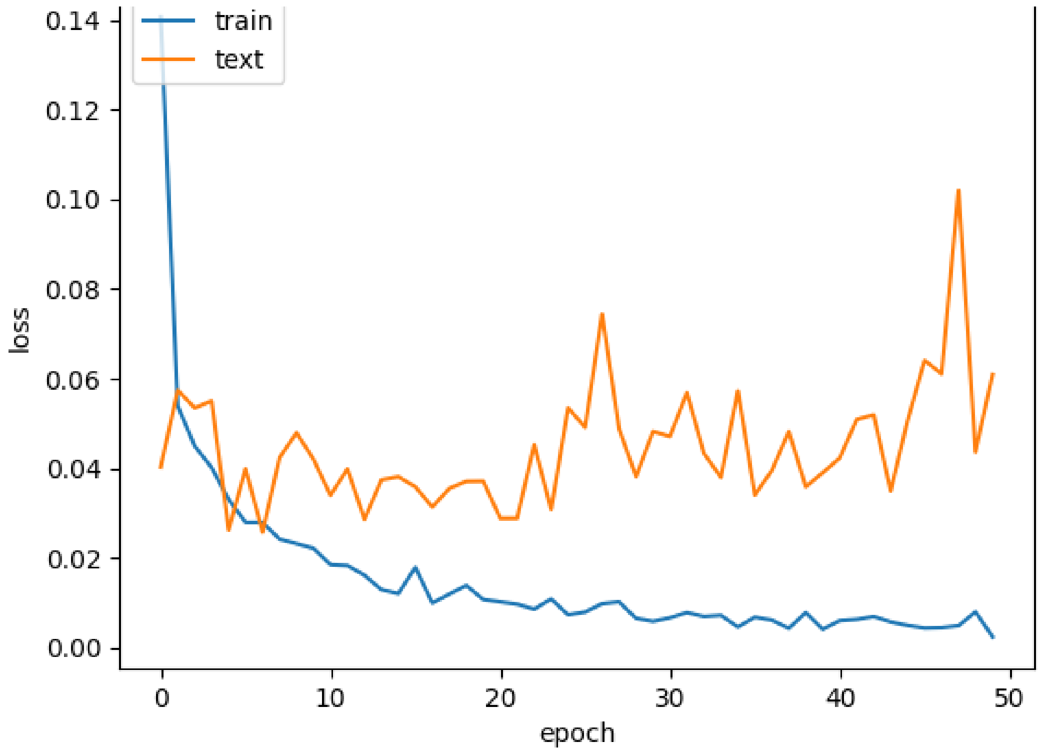
**Trực quan hóa kết quả Accuracy và Loss trên tập Train và Test đối với mô hình mới**

In [ ]:

```python
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','text'],loc='upper left')
plt.show()
```






**Lưu mô hình mới**

In [ ]:

```
model.save('model2.h5')
from tensorflow.keras.models import load_model
model2 = load_model('/content/model2.h5')
```

**Tính y dự đoán từ mô hình mới đã lưu, lấy argmax của y dự đoán và y test**

In [ ]:

```
y_hat = model2.predict(X_test_scaled)
y_test = np.argmax(y_test, axis=1)
y_hat = np.argmax(y_hat, axis=1)
```

```
313/313 [==============================] - 2s 5ms/step
```

## Sử dụng classification_report trong thư viện Sklearn đánh giá kết quả mô hình mới dựa trên kết quả dự đoán tập test

In [ ]:

```
from sklearn.metrics import classification_report
target_names = ['0', '1', '2', '3','4','5','6','7','8','9']
print(classification_report(y_test, y_hat,target_names=target_names))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       0.99      1.00      0.99      1135
           2       1.00      1.00      1.00      1032
           3       0.99      1.00      0.99      1010
           4       0.99      1.00      1.00       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       1.00      0.98      0.99      1028
           8       0.99      0.99      0.99       974
           9       1.00      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

### Sử dụng Confusion_matrix trong thư viện Sklearn biểu diễn kết quả dự đoán trên tập test

In [ ]:

```
import sklearn.metrics
import seaborn as sn

# Tạo confusion matrix
cm = sklearn.metrics.confusion_matrix(y_test, y_hat)

# Vẽ confusion matrix
plt.figure()
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```