1. Introduction

Handwritten character recognition is one of the practically important issues in pattern recognition applications. The applications of digit recognition includes in bank check processing, form data entry, etc. It will reduce the manual effort for bank check processing, and validation of handwritten form.

In this project we will start with digit recognition, which can be extended to alphabates for the broadening coverage, and solving other issues related to bank check processing and form validation.

Dataset we are using is MNIST(Modified National Institute of Standards and Technology) Data, it has more than 60000 record for training and testing.

2. Data Wrangling

This step focuses on collecting data, organizing it, and making sure it's well defined.

2.1 Imports

Importing dependencies required for Digit recognition

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import decomposition
from sklearn.preprocessing import StandardScaler
```

2.2 Load Data

We will start with Training data:

In [2]:

```
train_df=pd.read_csv('Raw Data/mnist_train.csv',header=None)
```

In [3]:

```
print("traing data shape:",train_df.shape)
```

```
traing data shape: (60000, 785)
```

In [4]:

```
train_df.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	784
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Now let's also load the testing data:

In [5]:

```
test_df=pd.read_csv('Raw Data/mnist_test.csv',header=None)
```

In [6]:

```
print("traing data shape:",test_df.shape)
```

```
traing data shape: (10000, 785)
```

In [7]:

```
test_df.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	784
0	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [8]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, 0 to 784
dtypes: int64(785)
memory usage: 359.3 MB
```

In [9]:

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 785 entries, 0 to 784
dtypes: int64(785)
memory usage: 59.9 MB
```

At this stage, we just got information that we have 60000 2d images in our training dataset, and 10000 in test dataset. Columns of training data from 1 to 784 represents the pixel values, and column 0 represents the corrosponding label, same goes for test data.

2.3 Data Exploration

2.3.1 Missing values

Firstly, we will look for any missing values in training dataset:

In [10]:

```
missing_in_train = train_df.isnull().sum()>0
print(missing_in_train.sum())
```

```
0
```

Now, Let check in testing dataset:

In [11]:

```
missing_in_test = test_df.isnull().sum()>0
print(missing_in_test.sum())
```

```
0
```

There are no missing data in these datasets. Let's check the data distribution.

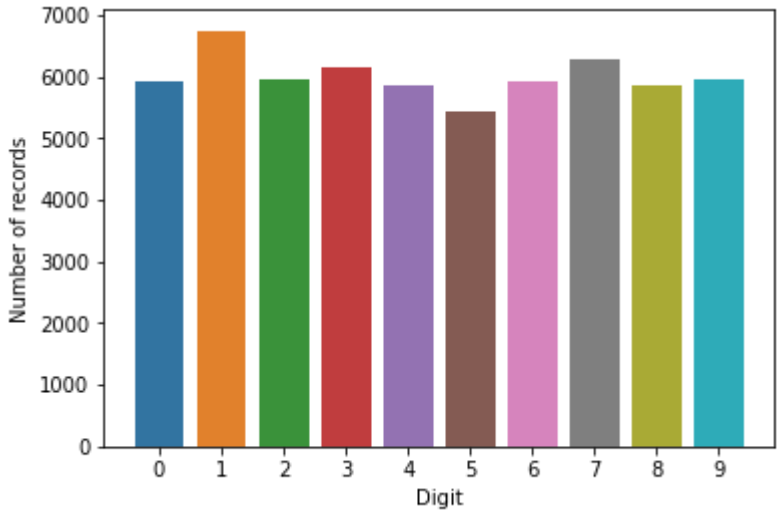
In [12]:

```
sns.countplot(train_df[0])
plt.xlabel('Digit')
plt.ylabel('Number of records')
plt.plot()
```

```
C:\Users\61435\anaconda3\lib\site-packages\seaborn\_decorators.py:43: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[12]:

```
[]
```



Test dataset is not baised. It has nearly equal number of records for each digits.

Summary:

As there are no missing data, data bais or any other issue, we can move to next step.

3. Exploratory Data Analysis

Let's perform EDA to examine relationships between variables and other patterns in the data.

3.1 Initial Analysis

In [13]:

```
train_df.describe()
```

Out[13]:

	0	1	2	3	4	5	6	7	8	
count	60000.000000	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	6(
mean	4.453933	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
std	2.889270	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

8 rows × 785 columns

There are some columns with all 0's, but these might be useful in testing data.

3.2 Normalisation

As datasets have values from 0-255(pixel values), it will be a good idea to normalise the data.

In [14]:

```
x_train = train_df.drop(0,axis=1).values
y_train = train_df[0].values
x_test = test_df.drop(0,axis=1).values
y_test = test_df[0].values
```

In [15]:

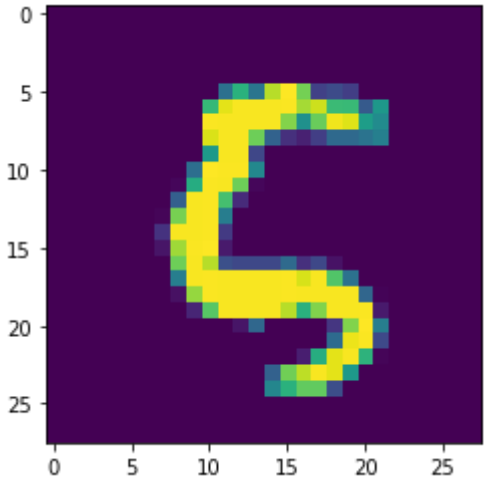
```
x_train = x_train/255
x_test = x_test/255
```

3.3 Visulaize a single digit

In [16]:

```
print(y_train[2000])
plt.imshow(x_train[2000].reshape(28,28));
```

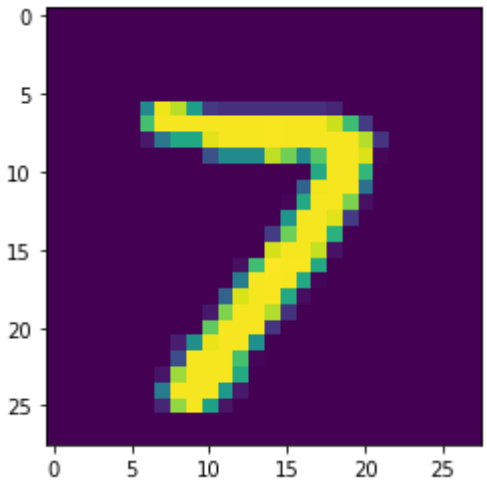
5



In [17]:

```
print(y_train[4000])
plt.imshow(x_train[4000].reshape(28,28));
```

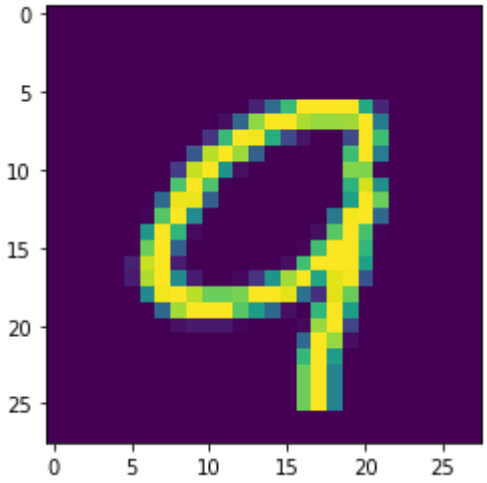
7



In [18]:

```
print(y_train[3000])
plt.imshow(x_train[3000].reshape(28,28));
```

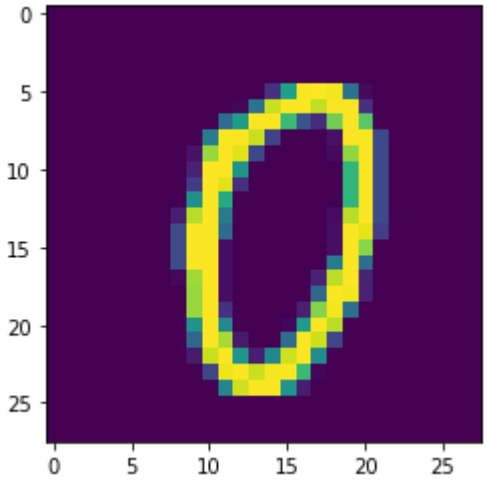
9



In [19]:

```
print(y_train[1000])
plt.imshow(x_train[1000].reshape(28,28));
```

0



3.4 PCA

In [20]:

```
standardized_scalar = StandardScaler()
standardized_data = standardized_scalar.fit_transform(x_train)
standardized_data.shape
```

Out[20]:

```
(60000, 784)
```

In [21]:

```
pca = decomposition.PCA()
pca.n_components = 2
pca_data = pca.fit_transform(standardized_data)
pca_data.shape
```

Out[21]:

```
(60000, 2)
```

In [22]:

```
principal_df = pd.DataFrame(data = pca_data, columns = ['principal component 1', 'principal
principal_df.head()
```

Out[22]:

	principal component 1	principal component 2
0	-0.922172	-4.814323
1	8.708978	-7.754135
2	2.328408	9.428706
3	-6.582196	-3.746666
4	-5.183261	3.133669

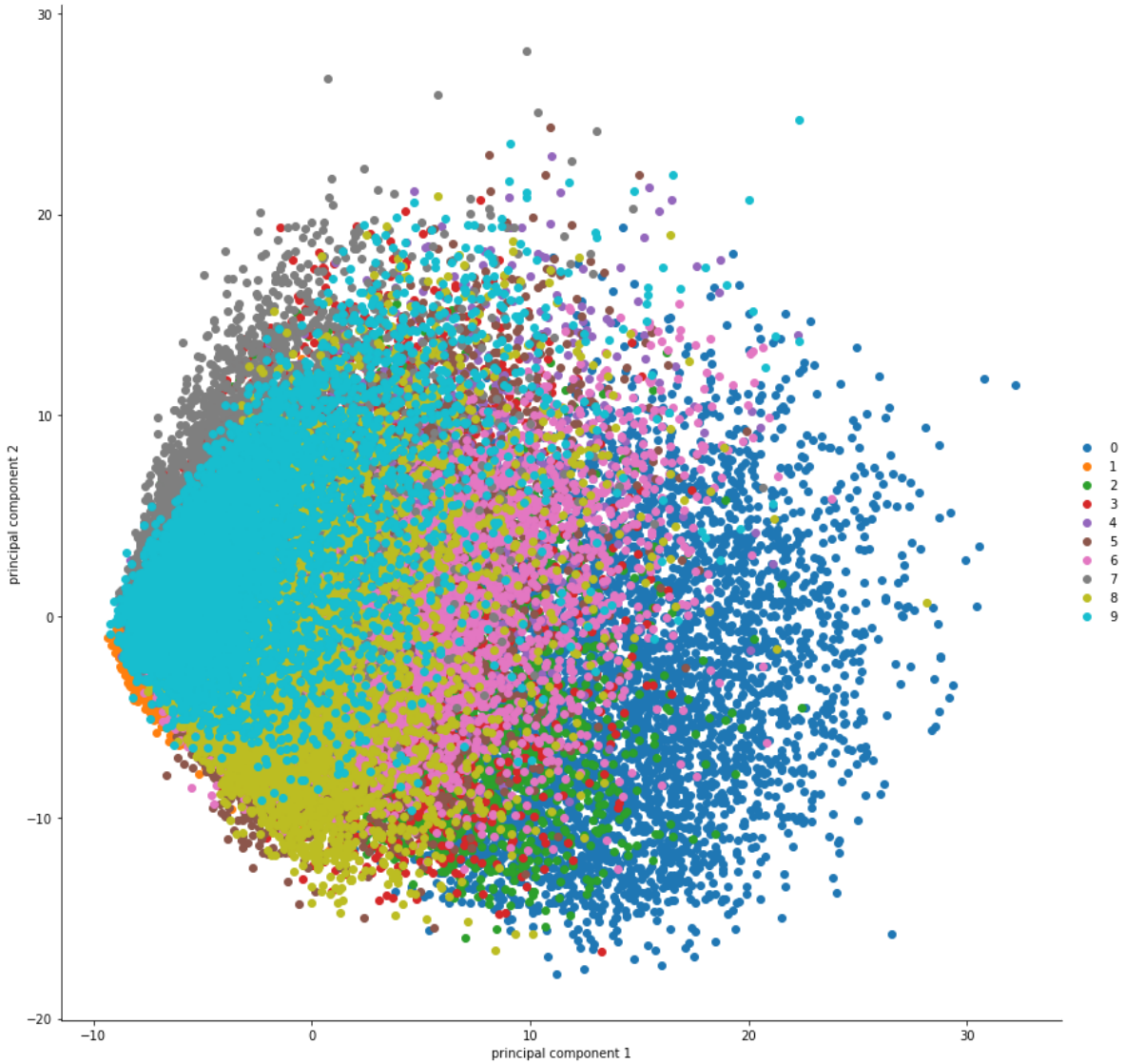
In [23]:

```
new_df = pd.concat([principal_df,train_df[0] ], axis=1)
```

In [24]:

```
sns.FacetGrid(new_df,hue=0, size=12).map(plt.scatter, 'principal component 1', 'principal c
plt.savefig("PCA_FacetGrid.png")
plt.show()
```

```
C:\Users\61435\anaconda3\lib\site-packages\seaborn\axisgrid.py:316: UserWarn
ing: The `size` parameter has been renamed to `height`; please update your c
ode.
  warnings.warn(msg, UserWarning)
```



It can be deduced form PCA that most of the digits are forming clusters, but most of them are overlaping clusters. We still albe to determine that cluster of 0 is distingusable from others.

Summary:

We have divided datasets in features and output. We have also normalised the features, so that it will be easier to understand the data. Finally, we performed the PCA to determine the patterns in data.

4.Preprocessing

4.0 Import

In [25]:

```
from tensorflow.keras import optimizers
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Lambda, Flatten, BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPool2D, AvgPool2D
from tensorflow.keras.optimizers import Adadelta
from keras.utils.np_utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import LearningRateScheduler
from sklearn.metrics import accuracy_score
```

4.1 Converting dataframe into arrays

In [26]:

```
#x_train = train_df.drop(0,axis=1).values
#y_train = train_df[0].values
#x_test = test_df.drop(0,axis=1).values
#y_test = test_df[0].values

train_data = np.array(x_train)
train_label = np.array(y_train)
```

4.2 Reshaping the input shapes to get it in the shape which model uses

In [27]:

```
train_data = train_data.reshape(train_data.shape[0], 28, 28, 1)
print(train_data.shape, train_label.shape)
```

```
(60000, 28, 28, 1) (60000,)
```

4.3 Encoding train labels

In [28]:

```
# Encoding the labels and making them as the class value and finally converting them as cat
nclasses = train_label.max() - train_label.min() + 1
train_label = to_categorical(train_label, num_classes = nclasses)
print("Shape of ytrain after encoding: ", train_label.shape)
```

```
Shape of ytrain after encoding:  (60000, 10)
```

5. Modelling

5.1 Base Model ANN(Simple Neural Network)

In [113]:

```
model = Sequential()
model.add(Dense(units=20, activation='sigmoid',name="hidden_layer"))
model.add(Dense(units=10, activation='sigmoid',name="output_layer"))
model(x_train)
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden_layer (Dense)         (60000, 20)               15700
_________________________________________________________________
output_layer (Dense)         (60000, 10)               210
=================================================================
Total params: 15,910
Trainable params: 15,910
Non-trainable params: 0
_________________________________________________________________
```

In [114]:

```
optimizer = optimizers.Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
# This builds the model for the first time:
for epoch in range(15):
    history= model.fit(x_train,pd.get_dummies(y_train).values,epochs=epoch)
```

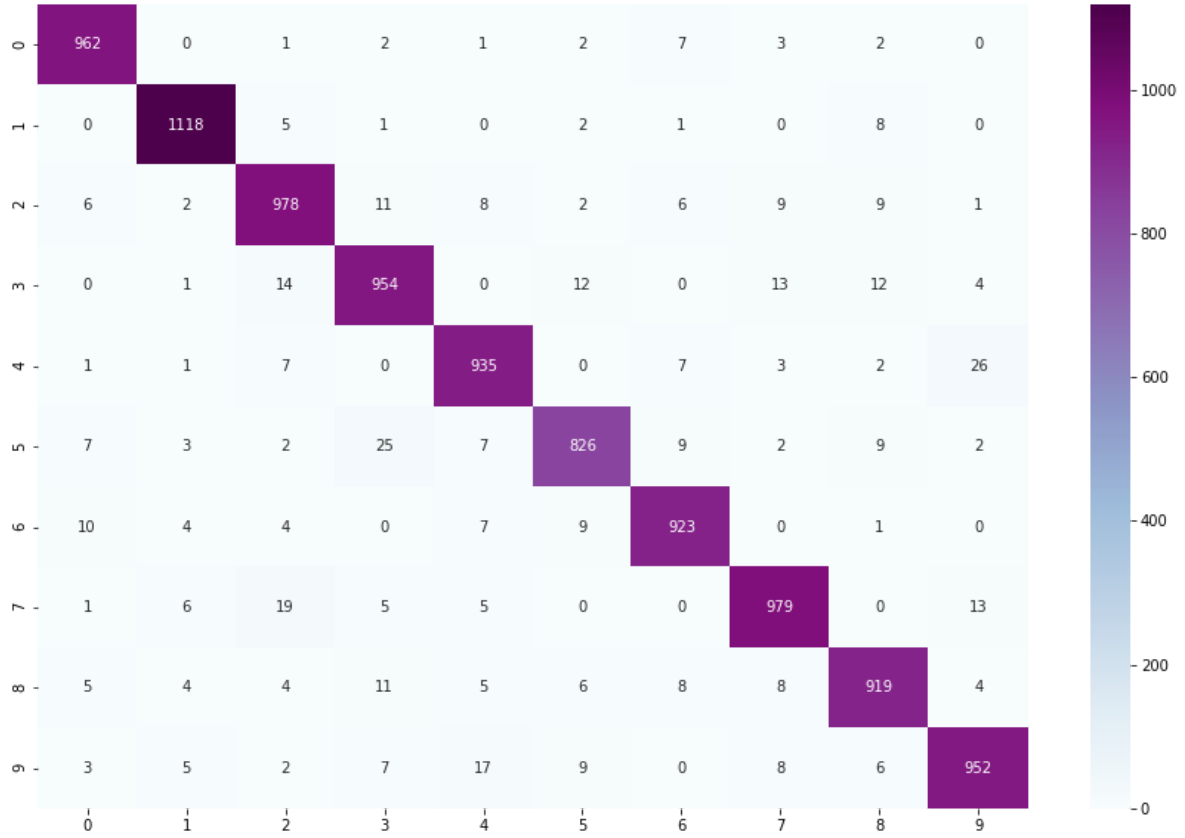
```
accuracy: 0.9624
Epoch 8/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1317 -
accuracy: 0.9627
Epoch 9/13
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1311 -
accuracy: 0.9630
Epoch 10/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1304 -
accuracy: 0.9631
Epoch 11/13
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1297 -
accuracy: 0.9632
Epoch 12/13
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1291 -
accuracy: 0.9635
Epoch 13/13
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1284 -
accuracy: 0.9634
Epoch 1/14
```

In [115]:

```
y_pred  = model.predict(x_test)
```

In [116]:

```
cf =tf.math.confusion_matrix(labels=y_test,predictions=[np.argmax(i) for i in y_pred])
plt.figure(figsize=(15,10))
sns.heatmap(cf,annot=True,fmt='d',cmap='BuPu');
```



As we can see from the confusion matrix that :

1. some 2 were identified as 3(11),
2. some 3 were identified as 2(14), 5(12), 7(13), 8(12)
3. some 4 were identified as 9(26)
4. some 5 were identified as 3(25)
5. some 6 were identified as 0(10)
6. some 7 were identified as 2(19), 9(13)
7. some 8 were identified as 3(11)
8. some 9 were identified as 4(17)

In [117]:

```
y_pred = np.argmax(y_pred, axis=1)
accuracy_score(y_pred,y_test)
```

Out[117]:

```
0.9546
```

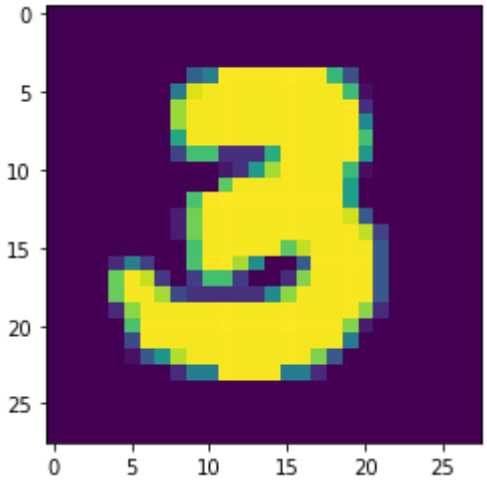
In [39]:

```
def sample(i):
        print("actual label ", y_test[i])
        print("predicted label ", np.argmax(y_pred[i]))
        print("The corrosponding image")
        plt.imshow(x_test[i].reshape(28,28))
```

In [44]:

```
sample(200)
```

```
actual label  3
predicted label  0
The corrosponding image
```



Accuracy for the base ANN model with one hidden layer is 95.46%, which is good. Let's apply CNN model as it is more suitable for image data. But before that let increase the number of node in the hidden layer.

5.2 ANN with one hidden layer(Increasing the number of node in the hidden layer)

In [45]:

```
model_2 =Sequential()
model_2.add(Dense(128,input_shape=(784,),activation='relu'))
model_2.add(Dense(10,activation='sigmoid'))


model_2.compile(optimizer='adam',
             loss = 'sparse_categorical_crossentropy',
             metrics=['accuracy']
             )
for epoch in range(15):
    model_2.fit(x_train,y_train,epochs=epoch)
```

```
accuracy: 0.9994
Epoch 8/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0021 -
accuracy: 0.9991
Epoch 9/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0022 -
accuracy: 0.9994
Epoch 10/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0025 -
accuracy: 0.9992
Epoch 11/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0020 -
accuracy: 0.9993
Epoch 12/13
1875/1875 [==============================] - 2s 1ms/step - loss: 8.9543e-
04 - accuracy: 0.9997
Epoch 13/13
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0037 -
accuracy: 0.9990
```

In [46]:

```
model_2.evaluate(x_test,y_test)
```

```
313/313 [==============================] - 0s 543us/step - loss: 0.2325 - ac
curacy: 0.9791
```

Out[46]:

```
[0.2324547916650772, 0.9790999889373779]
```

In [47]:

```
y_pred2 = model_2.predict(x_test)
```

In [120]:

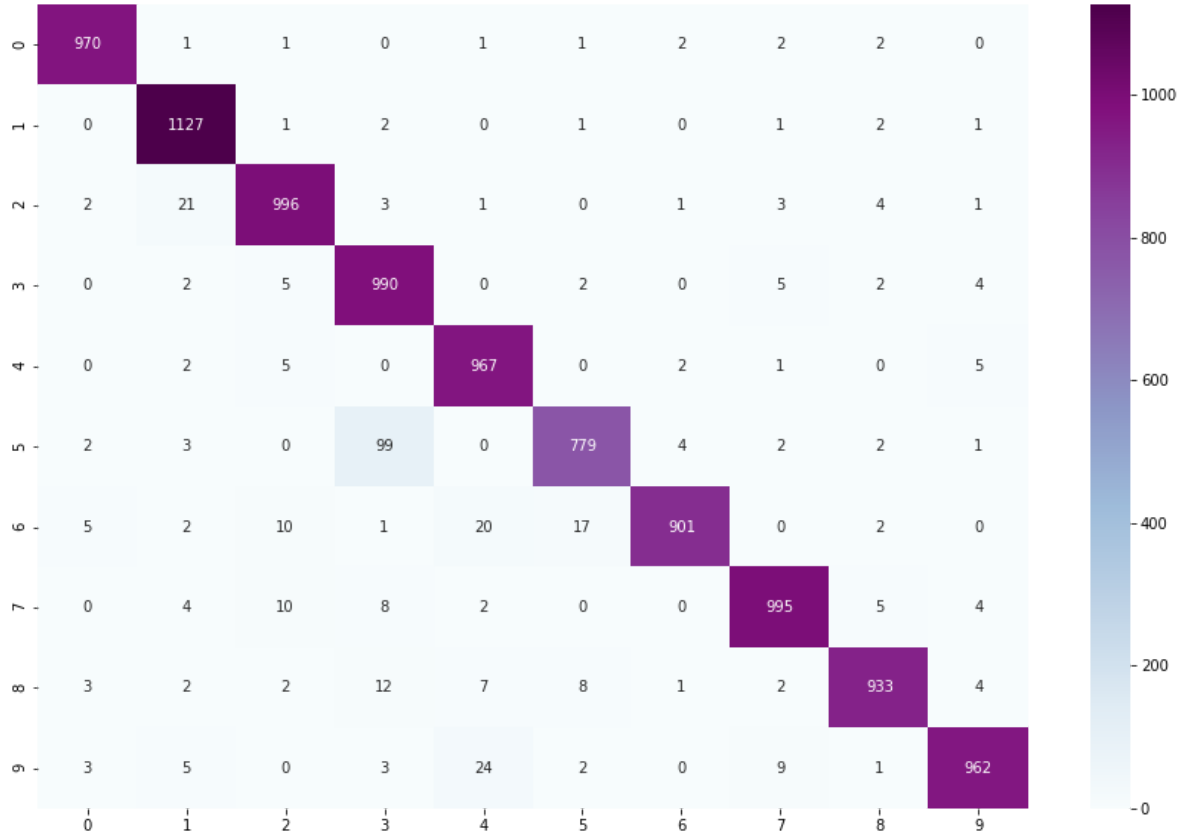
```
y_pred2 = np.argmax(y_pred2, axis=1)
accuracy_score(y_pred2,y_test)
```

Out[120]:

```
0.962
```

In [48]:

```
cf =tf.math.confusion_matrix(labels=y_test,predictions=[np.argmax(i) for i in y_pred2])
plt.figure(figsize=(15,10))
sns.heatmap(cf,annot=True,fmt='d',cmap='BuPu');
```



As we can see from the confusion matrix that :

1. some 5 were identified as 3(99)
2. some 6 were identified as 2(10), 4(20), 5(17)
3. some 7 were identified as 2(10)
4. some 8 were identified as 3(12)
5. some 9 were identified as 4(24)

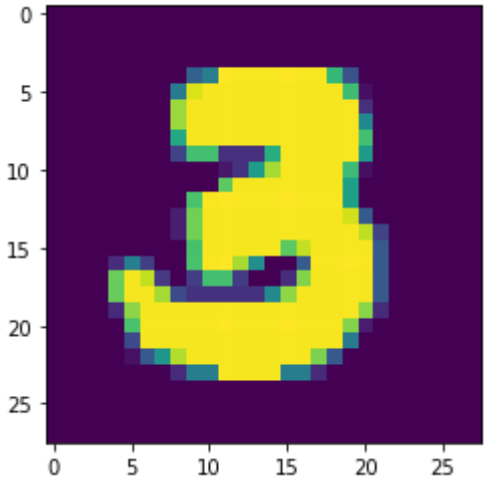
In [50]:

```
def sample(i):
        print("actual label ", y_test[i])
        print("predicted label ", np.argmax(y_pred2[i]))
        print("The corrosponding image")
        plt.imshow(x_test[i].reshape(28,28))
```

In [55]:

```
sample(200)
```

```
actual label  3
predicted label  3
The corrosponding image
```



Accuracy for the this model with one hidden layer(more number of nodes) is 96.2%, which is improvment compared to previous ANN model. Let's build more complex CCN model to achieve more accuracy.

5.3 CNN model

First we will buid functions to build, compile and train model, which will makes modelling easy.

In [56]:

```
def build_model(input_shape=(28, 28, 1)):
    model = Sequential()
    model.add(Conv2D(32, kernel_size = 3, activation='relu', input_shape = input_shape))
    model.add(BatchNormalization())
    model.add(Conv2D(32, kernel_size = 3, activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2D(32, kernel_size = 5, strides=2, padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.4))

    model.add(Conv2D(64, kernel_size = 3, activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, kernel_size = 3, activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, kernel_size = 5, strides=2, padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.4))

    model.add(Conv2D(128, kernel_size = 4, activation='relu'))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(10, activation='softmax'))
    return model


def compile_model(model, optimizer='adam', loss='categorical_crossentropy'):
    model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])


def train_model(model, train, test, epochs, split):
    history = model.fit(train, test, shuffle=True, epochs=epochs, validation_split=split)
    return history
```

In [57]:

```
cnn_model = build_model((28, 28, 1))

compile_model(cnn_model, 'adam', 'categorical_crossentropy')

model_history = train_model(cnn_model, train_data, train_label, 40, 0.2)
```

```
Epoch 28/40
1500/1500 [==============================] - 135s 90ms/step - loss: 0.008
3 - accuracy: 0.9973 - val_loss: 0.0294 - val_accuracy: 0.9939
Epoch 29/40
1500/1500 [==============================] - 124s 83ms/step - loss: 0.010
5 - accuracy: 0.9967 - val_loss: 0.0255 - val_accuracy: 0.9946
Epoch 30/40
1500/1500 [==============================] - 122s 81ms/step - loss: 0.008
5 - accuracy: 0.9971 - val_loss: 0.0227 - val_accuracy: 0.9948
Epoch 31/40
1500/1500 [==============================] - 121s 81ms/step - loss: 0.009
6 - accuracy: 0.9969 - val_loss: 0.0268 - val_accuracy: 0.9944
Epoch 32/40
1500/1500 [==============================] - 120s 80ms/step - loss: 0.008
3 - accuracy: 0.9973 - val_loss: 0.0261 - val_accuracy: 0.9946
Epoch 33/40
1500/1500 [==============================] - 121s 81ms/step - loss: 0.008
8 - accuracy: 0.9972 - val_loss: 0.0250 - val_accuracy: 0.9948
Epoch 34/40
1500/1500 [==============================] - 124s 82ms/step - loss: 0.007
```

Model Performance Analysis:

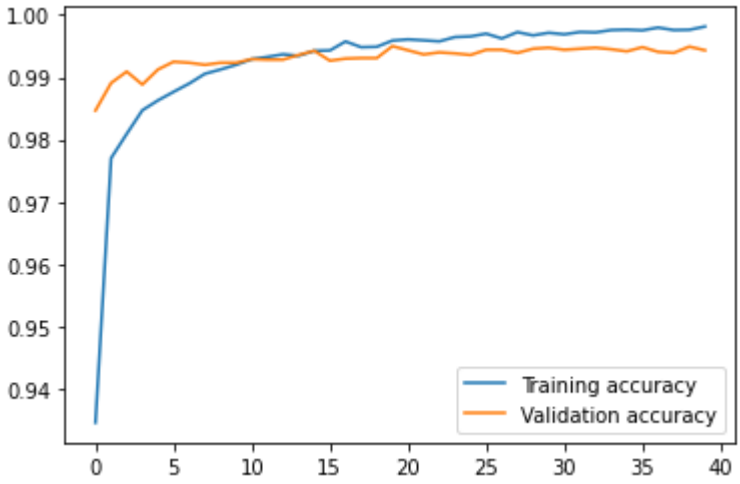
In [58]:

```
def plot_model_performance(metric, validations_metric):
    plt.plot(model_history.history[metric],label = str('Training ' + metric))
    plt.plot(model_history.history[validations_metric],label = str('Validation ' + metric))
    plt.legend()
    plt.savefig(str(metric + '_plot.png'))
```

Plot for Accuracy:

In [59]:

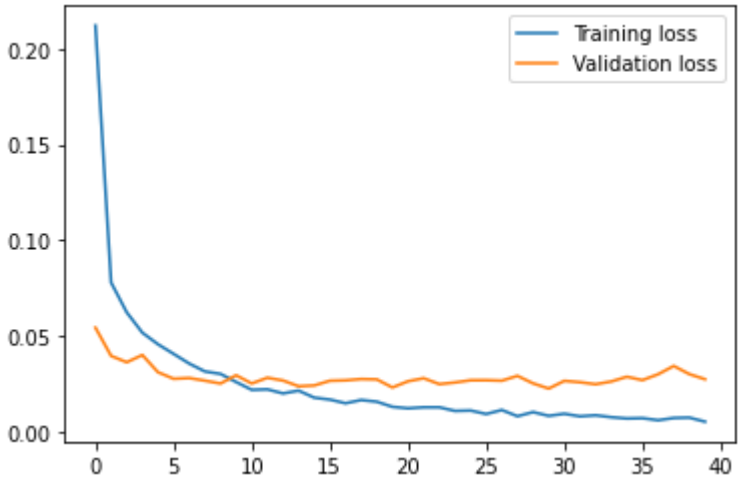
```
plot_model_performance('accuracy', 'val_accuracy')
```



Plot for loss:

In [60]:

```
plot_model_performance('loss', 'val_loss')
```



Transforming testing data

In [61]:

```
#x_train = train_df.drop(0,axis=1).values
#y_train = train_df[0].values
#x_test = test_df.drop(0,axis=1).values
#y_test = test_df[0].values

test_data = np.array(x_test)
test_data = test_data.reshape(test_data.shape[0], 28, 28, 1)
print(test_data.shape)
```

```
(10000, 28, 28, 1)
```

Prediction:

In [62]:

```
predictions = cnn_model.predict(test_data)
```

In [81]:

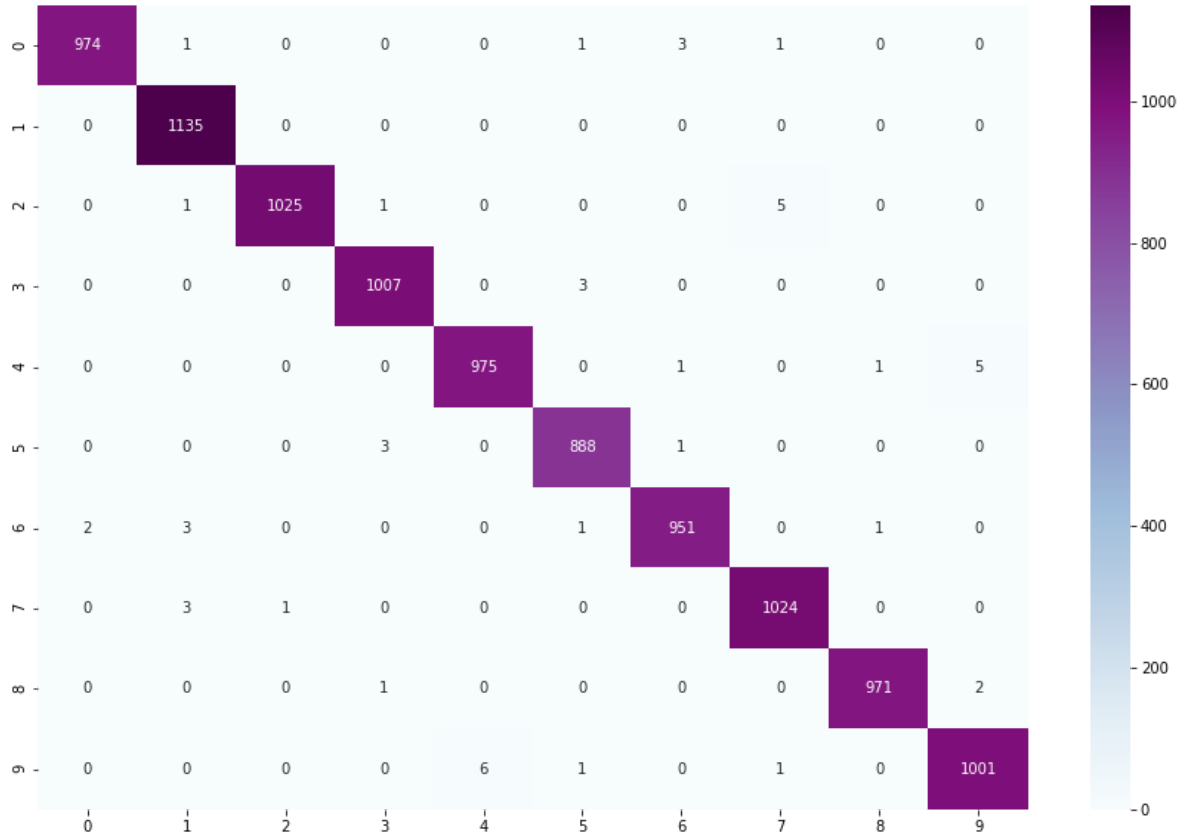
```
y_pred3 = np.argmax(predictions, axis=1)
accuracy_score(y_pred3,y_test)
```

Out[81]:

```
0.9951
```

In [82]:

```
cf =tf.math.confusion_matrix(labels=y_test,predictions=[np.argmax(i) for i in predictions])
plt.figure(figsize=(15,10))
sns.heatmap(cf,annot=True,fmt='d',cmap='BuPu');
```



As we can see from the confusion matrix that :

1. some 2 were identified as 7(5)
2. some 4 were identified as 9(5)
3. some 9 were identified as 4(6)

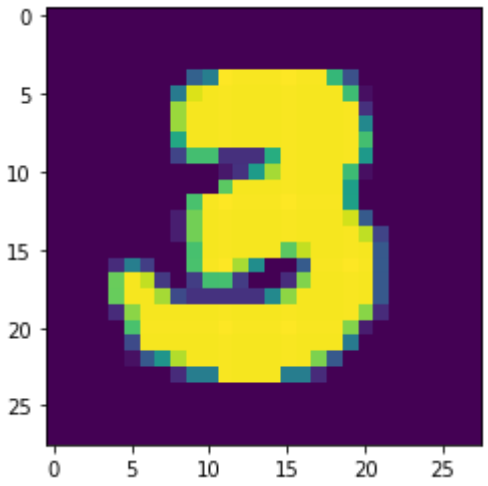
In [89]:

```
def sample(i):
        print("actual label ", y_test[i])
        print("predicted label ", np.argmax(predictions[i]))
        print("The corrosponding image")
        plt.imshow(x_test[i].reshape(28,28))
```

In [91]:

```
sample(200)
```

```
actual label  3
predicted label  3
The corrosponding image
```



We achieved the accuracy of 99.51% using CNN model, which is great.

Now, let's test this model with new images. Loading more test data:

5.4 Testing with more images

In [92]:

```
batch_size=50
img_height=28
img_width=28

ds_test=tf.keras.preprocessing.image_dataset_from_directory(
'Raw Data/Self Drawn image',
    labels='inferred',
    label_mode='int',
    #class_names=[],
    color_mode='grayscale',
    batch_size=batch_size,
    image_size=(img_height, img_width),
    shuffle=True,
    seed=123,


)
```

```
Found 50 files belonging to 10 classes.
```

In [93]:

```
import tensorflow_datasets as tfds
x_test2=None
for image, lable in tfds.as_numpy(ds_test):
    x_test2=image
    y_test2=lable
```

In [94]:

```
print(x_test2.shape, y_test2.shape)
```

```
(50, 28, 28, 1) (50,)
```

In [103]:

```
y_pred4  = cnn_model.predict(x_test2)
```

In [111]:

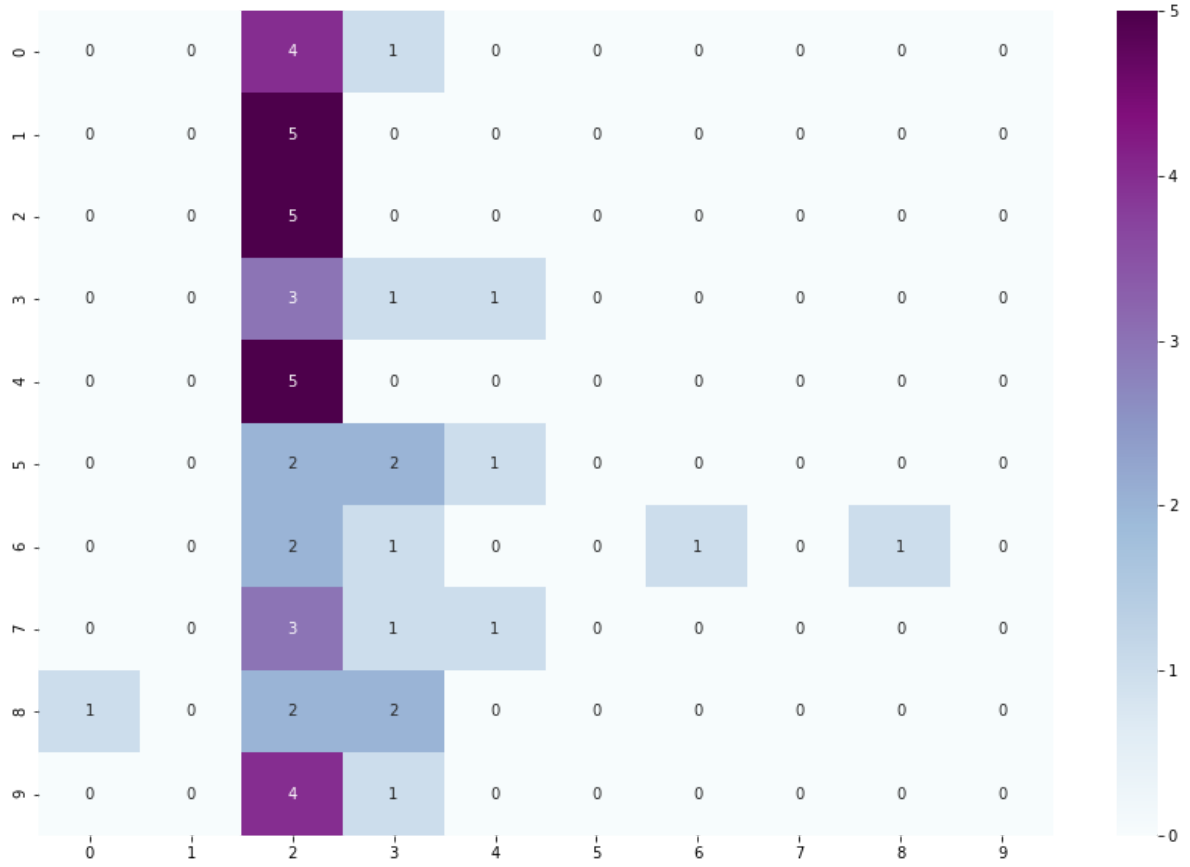
```
y_pred4 = np.argmax(y_pred4, axis=1)
accuracy_score(y_pred4,y_test2)
```

Out[111]:

```
0.14
```

In [104]:

```
cf =tf.math.confusion_matrix(labels=y_test2,predictions=[np.argmax(i) for i in y_pred4])
plt.figure(figsize=(15,10))
sns.heatmap(cf,annot=True,fmt='d',cmap='BuPu');
```



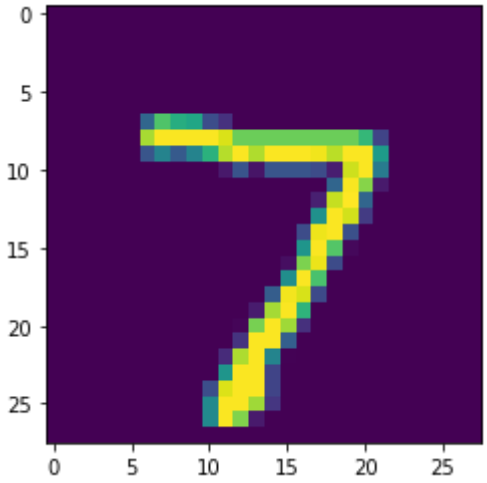
In [105]:

```
def sample(i):
        print("actual label ", y_test[i])
        print("predicted label ", np.argmax(y_pred4[i]))
        print("The corrosponding image")
        plt.imshow(x_test[i].reshape(28,28))
```

In [110]:

```
sample(0)
```

```
actual label  7
predicted label  2
The corrosponding image
```

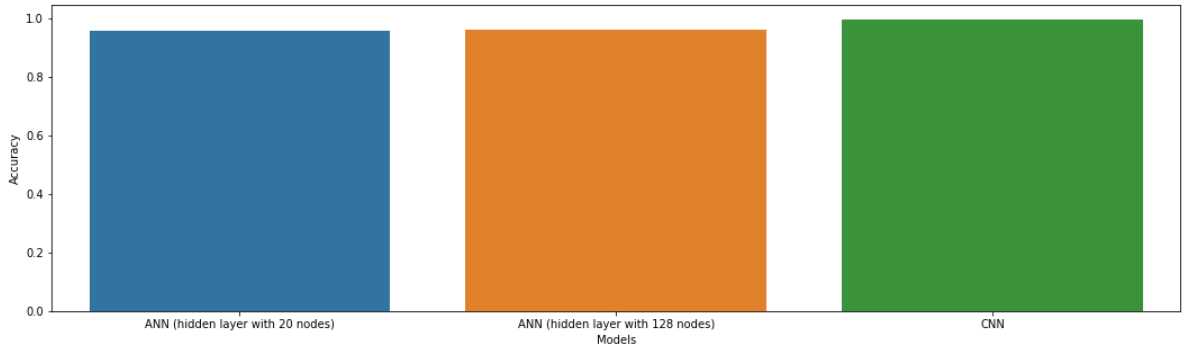


5.5 Conclusion

Model comparison:

In [121]:

```
f, ax = plt.subplots(figsize=(18,5))
sns.barplot(x=["ANN (hidden layer with 20 nodes)","ANN (hidden layer with 128 nodes)","CNN"
            y=[accuracy_score(y_pred,y_test),accuracy_score(y_pred2,y_test),accuracy_score(
plt.ylabel("Accuracy")
plt.xlabel("Models")
plt.show()
```



Conclusion:

As we can see from the model comparison that CNN has the maximum accuracy(99.51 %). We were able to identify most of the digits accurately.