## NAME-BACHU GANESH

## ROLL NUMBER-CH.EN.U4AIE20003

### Question -1

Task - Handwritten Digit Classification

1. Build a CNN model to classify the digits.

2. Change the number of layers and check the performance of the model. Observe the results.

3. Try with different activation function and check the performance of the model. Observe the results.

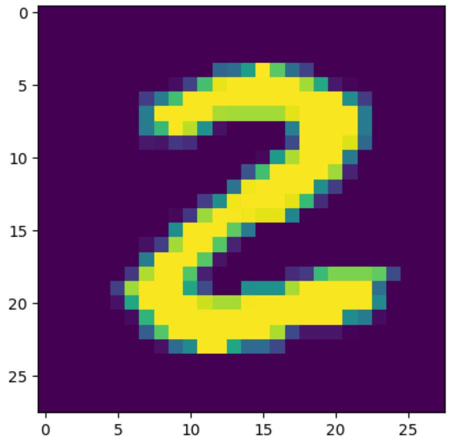
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2

from PIL import Image
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
from keras.datasets import mnist
from tensorflow.math import confusion_matrix
```

In [2]:
```
(X_train, Y_train), (X_test, Y_test) =  mnist.load_data()
```

In [3]:
```
# Shape of the numpy arrays
print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

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

In [4]:
```
# Displaying the image
plt.imshow(X_train[25])
plt.show()
```



In [5]:
```python
# print the corresponding label
print(Y_train[25])
```

```
2
```

In [6]:
```python
print(Y_train.shape, Y_test.shape)
```

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

In [7]:
```python
# Unique values in Y_train
print(np.unique(Y_train))
# Unique values in Y_test
print(np.unique(Y_test))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

In [8]:
```python
# Scaling the values
#X_train = X_train/255
#X_test = X_test/255

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train, X_test = X_train / 255.0, X_test / 255.0
```

In [10]:
```python
# Setting up the layers of the Neural  Network
model = keras.Sequential([
    keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28,1)),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation ='relu'),
    keras.layers.Dense(10, activation='softmax')])
```

In [11]:
```python
# Compiling the Neural Network
model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

In [12]:
```python
# Training the Neural Network
model.fit(X_train, Y_train, epochs=2)
```

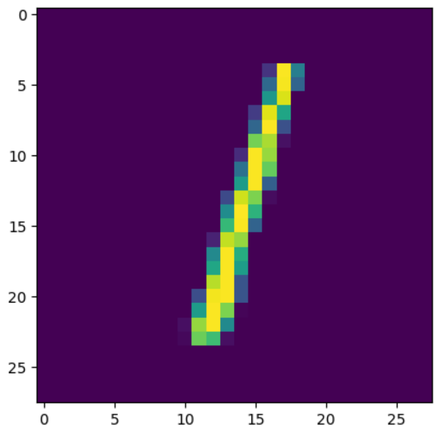
```
Epoch 1/2
1875/1875 [==============================] - 53s 28ms/step - loss: 0.1569 - accuracy: 0.9531
Epoch 2/2
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0529 - accuracy: 0.9836
```

Out[12]:
```
<keras.callbacks.History at 0x25c027195e0>
```

In [13]:
```python
loss, accuracy = model.evaluate(X_test, Y_test)
print(accuracy)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0491 - accuracy: 0.9837
0.9836999773979187
```

In [14]:
```python
# First data point in X_test
plt.imshow(X_test[2])
plt.show()
```



In [15]:
```
print(Y_test[2])
```
```
1
```

In [16]:
```
Y_pred = model.predict(X_test)
label_for_first_test_image = np.argmax(Y_pred[2])
print(label_for_first_test_image)
```
```
313/313 [==============================] - 1s 4ms/step
1
```

In [17]:
```
Y_pred_label=np.argmax(Y_pred, axis=1)
print(Y_pred_label)
```
```
[7 2 1 ... 4 5 6]
```

In [18]:
```
#Confusion matrix
conf_mat = confusion_matrix(Y_test, Y_pred_label)
print(conf_mat)
```
```
tf.Tensor(
[[ 969    0    3    0    0    4    1    1    2    0]
 [    0 1123    5    1    1    1    2    0    2    0]
 [    0    1 1023    0    1    0    0    3    4    0]
 [    1    0    2  986    0   15    0    3    3    0]
 [    0    0    1    0  965    0    0    0    3   13]
 [    1    0    0    1    0  888    2    0    0    0]
 [    4    2    1    1    2    2  943    0    3    0]
 [    1    1   11    2    0    0    0 1010    2    1]
 [    5    0    4    0    1    2    1    4  954    3]
 [    4    2    1    1    4    7    0    5    9  976]], shape=(10, 10), dtype=int32)
```

After completing task 1 I came up with a conclusion that the model layers between the convolution layer (Input layer) and the output layer, the dense layers have increased the accuracy linear and the time taken to compile the model this not take much time with add extra dense layers but then add extra convolution layers the time taken to train the model has increased exponentially and there is no change in accuracy. The last layer must have sigmoid or SoftMax as activation function as we need the results between 0 to 1. There was no much different in accuracy or time taken when activation function as changed in last layer. In the other layers RELU has given the best accuracy when compared to tan-h and  Leaky ReLU.

## Question-2

Task – Image Classification task

1. Build a CNN model to classify the different classes of images.

2. Change the training and testing split ratio. Observe the results.

3. Try with different cross validation approach, check the performance of the model. Observe the results.

4. Try removing max pooling and normalization layers, check the performance of the model. Observe the results.

I used a total of 3 different cross validations which are:-

1. Basic Cross Validation
2. K-Folds Cross Validation
3. Monte Carlo Cross Validation

## Basic Cross Validation

In [13]:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

from keras.datasets import cifar10
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

In [14]:

```
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()

# View first image
plt.imshow(X_train[0])
plt.show()

import numpy as np
from sklearn.model_selection import train_test_split

# Concatenate train and test images
X = np.concatenate((X_train,X_test))
y = np.concatenate((Y_train,Y_test))

# Check shape
print(X.shape) # (60000, 32, 32, 3)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10000, random_state=1234)

# Check shape
print(X_train.shape) # (50000, 32, 32, 3)

# View first image
plt.imshow(X_train[0])
plt.show()
```

In [15]:
```python
X_test.shape
```
Out[15]: (10000, 32, 32, 3)

In [16]:
```python
y_train.shape
```
Out[16]: (50000, 1)

In [17]:
```python
y_train[:5]
```
Out[17]:
```
array([[4],
       [0],
       [1],
       [3],
       [8]], dtype=uint8)
```

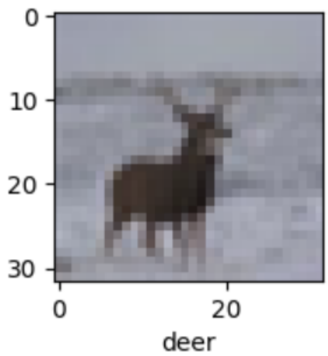
In [18]:
```python
y_train = y_train.reshape(-1,)
y_train[:5]
```
Out[18]: array([4, 0, 1, 3, 8], dtype=uint8)

In [19]:
```python
y_test = y_test.reshape(-1,)
```

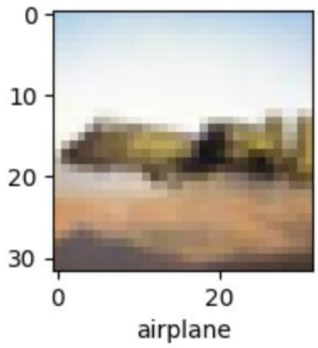
In [20]:
```python
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [21]:
```python
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [22]:
```python
plot_sample(X_train, y_train, 0)
```




In [23]:
```python
plot_sample(X_train, y_train, 1)
```




In [24]:
```python
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [25]:
```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

In [26]:
```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [30]:
```
#cnn.fit(X_train, y_train, epochs=2)

n_folds=3
epochs=20
batch_size=128

#save the model history in a list after fitting so that we can plot later
model_history = []

for i in range(n_folds):
    print("Training on Fold: ",i+1)
    cnn.fit(X_train, y_train, epochs=2)
    print("======="*12, end="\n\n\n")
```

```
Training on Fold:  1
Epoch 1/2
1563/1563 [==============================] - 59s 38ms/step - loss: 1.0039 - accuracy: 0.6495
Epoch 2/2
1563/1563 [==============================] - 65s 41ms/step - loss: 0.9305 - accuracy: 0.6778
====================================================================================


Training on Fold:  2
Epoch 1/2
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8639 - accuracy: 0.6999
Epoch 2/2
1563/1563 [==============================] - 57s 37ms/step - loss: 0.8124 - accuracy: 0.7165
====================================================================================


Training on Fold:  3
Epoch 1/2
1563/1563 [==============================] - 57s 36ms/step - loss: 0.7666 - accuracy: 0.7350
Epoch 2/2
1563/1563 [==============================] - 68s 44ms/step - loss: 0.7168 - accuracy: 0.7508
====================================================================================
```

In [31]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 5s 13ms/step - loss: 0.9167 - accuracy: 0.6939
```

Out[31]:
```
[0.9167031049728394, 0.6938999891281128]
```

In [32]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]

clas=np.argmax(y_pred,axis=1)
clas
```

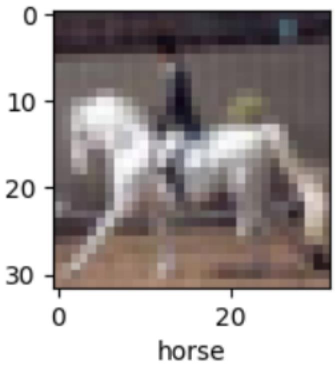
```
313/313 [==============================] - 4s 13ms/step
```

Out[32]:
```
array([8, 5, 2, ..., 2, 6, 3], dtype=int64)
```

In [33]:
```python
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```
Out[33]: [8, 5, 2, 7, 8]

In [34]:
```python
y_test[:5]
```
Out[34]: array([8, 3, 3, 7, 8], dtype=uint8)

In [35]:
```python
plot_sample(X_test, y_test,3)
```




In [36]:
```python
classes[y_classes[3]]
```
Out[36]: 'horse'

## K-Folds Cross Validation

In [1]:
```python
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import StratifiedKFold
```

In [2]:
```python
# Load the CIFAR10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

```

In [3]:
```python
# Define the model architecture
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10)
])
```

In [4]:
```python
# Compile the model
model.compile(optimizer='adam', loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

In [5]:
```python
# Define the number of folds for k-fold cross validation
k = 5

# Define the cross validation iterator
skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=1)

# Initialize an array to store the accuracy scores for each fold
scores = []
```

```
In [6]:
# Loop over the folds
for train_index, val_index in skf.split(x_train, y_train):
    # Split the data into training and validation sets for the current fold
    x_train_k, x_val_k = x_train[train_index], x_train[val_index]
    y_train_k, y_val_k = y_train[train_index], y_train[val_index]

    # Train the model on the current fold
    model.fit(x_train_k, y_train_k, batch_size=32, epochs=10, validation_data=(x_val_k, y_val_k))

    # Evaluate the model on the validation set for the current fold
    val_loss, val_accuracy = model.evaluate(x_val_k, y_val_k)

    # Append the accuracy score for the current fold to the scores array
    scores.append(val_accuracy)
```

```
Epoch 5/10
1250/1250 [==============================] - 75s 60ms/step - loss: 0.0801 - accuracy: 0.9737 - val_loss: 0.1415 - val_acc
uracy: 0.9535
Epoch 6/10
1250/1250 [==============================] - 75s 60ms/step - loss: 0.0615 - accuracy: 0.9789 - val_loss: 0.1502 - val_acc
uracy: 0.9515
Epoch 7/10
1250/1250 [==============================] - 76s 61ms/step - loss: 0.0591 - accuracy: 0.9797 - val_loss: 0.2837 - val_acc
uracy: 0.9239
Epoch 8/10
1250/1250 [==============================] - 75s 60ms/step - loss: 0.0696 - accuracy: 0.9757 - val_loss: 0.2065 - val_acc
```

```
In [20]:
# Print the mean accuracy and standard deviation of the accuracy scores
print("Accuracy: %.2f%% (+/- %.2f%%)" % (np.mean(scores) * 100, np.std(scores) * 100))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy: %.2f%%" % (test_accuracy * 100))
```

```
Accuracy: 82.23% (+/- 9.14%)
313/313 [==============================] - 2s 7ms/step - loss: 3.1590 - accuracy: 0.6625
Test Accuracy: 66.25%
```

## Monte Carlo Cross Validation

```
In [7]:
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import ShuffleSplit
```

```
In [8]:
# Load the CIFAR10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

```
In [9]:
# Define the model architecture
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10)
])

# Compile the model
model.compile(optimizer='adam', loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

```
In [10]: # Define the number of iterations for Monte Carlo cross validation
n_iter = 5

# Define the cross validation iterator
shuffle_split = ShuffleSplit(n_splits=n_iter, test_size=0.3, random_state=1)

# Initialize an array to store the accuracy scores for each iteration
scores = []

# Loop over the iterations
for train_index, val_index in shuffle_split.split(x_train, y_train):
    # Split the data into training and validation sets for the current iteration
    x_train_mc, x_val_mc = x_train[train_index], x_train[val_index]
    y_train_mc, y_val_mc = y_train[train_index], y_train[val_index]

    # Train the model on the current iteration
    model.fit(x_train_mc, y_train_mc, batch_size=32, epochs=10, validation_data=(x_val_mc, y_val_mc))

    # Evaluate the model on the validation set for the current iteration
    val_loss, val_accuracy = model.evaluate(x_val_mc, y_val_mc)

    # Append the accuracy score for the current iteration to the scores array
    scores.append(val_accuracy)
```

```
1094/1094 [==============================] - 21s 19ms/step - loss: 0.3137 - accuracy: 0.9116 - val_loss: 0.2893 - val_accurac
y: 0.9179
```

```
In [11]: # Print the mean accuracy and standard deviation of the accuracy scores
print("Accuracy: %.2f%% (+/- %.2f%%)" % (np.mean(scores) * 100, np.std(scores) * 100))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy: %.2f%%" % (test_accuracy * 100))
```

```
Accuracy: 78.11% (+/- 7.37%)
313/313 [==============================] - 3s 9ms/step - loss: 3.0424 - accuracy: 0.6675
Test Accuracy: 66.75%
```

On removing the Max pooling layer, the time required to compile the models has reduced and the accuracy also dropped this is due to over fitting in the model. On removing the normalization layer, the training has increased and the accuracy has dropped this is due to the large number the model has to compute. On increasing the training dataset size the accuracy has increased but this had lead to over fitting in the model and the training time has increase linearly.