```
# This is the dataset found in tensorflow
# With 100 classes containing 600 images each. There are 500 training images and 100 te
# The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes w
# and a "coarse" label (the superclass to which it belongs).
```

```
import tensorflow as tf
import numpy as np
import pandas as pd
from tensorflow.keras import datasets, layers, models

```

Loading data from tf datasets

```
(x_train, y_train), (x_test, y_test)= datasets.cifar100.load_data()

```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169009152/169001437 [==============================] - 2s 0us/step
169017344/169001437 [==============================] - 2s 0us/step
```

```
x_train.shape, x_test.shape         # 50000 for training, 10000 for testing
```

```
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

```
y_train.shape            # y is 2 dim array, we want them in 1D so reshape

```

```
(50000, 1)
```

```
y_train= y_train.reshape(-1,)
y_test= y_test.reshape(-1)

y_train.shape, y_test.shape                                        # we got t
```

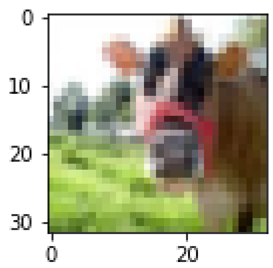
```
((50000,), (10000,))
```

```
x_train[0]                 # seeing the 1st image in training set
```

```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [195, 205, 193],
        [212, 224, 204],
        [182, 194, 167]],

       [[255, 255, 255],
        [254, 254, 254],
```

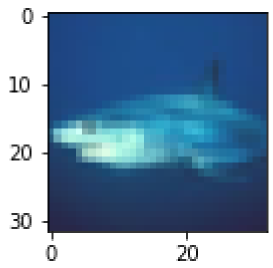
```
        [254, 254, 254],
        ...,
        [170, 176, 150],
        [161, 168, 130],
        [146, 154, 113]],

       [[255, 255, 255],
        [254, 254, 254],
        [255, 255, 255],
        ...,
        [189, 199, 169],
        [166, 178, 130],
        [121, 133,  87]],

       ...,

       [[148, 185,  79],
        [142, 182,  57],
        [140, 179,  60],
        ...,
        [ 30,  17,   1],
        [ 65,  62,  15],
        [ 76,  77,  20]],

       [[122, 157,  66],
        [120, 155,  58],
        [126, 160,  71],
        ...,
        [ 22,  16,   3],
        [ 97, 112,  56],
        [141, 161,  87]],

       [[ 87, 122,  41],
        [ 88, 122,  39],
        [101, 134,  56],
        ...,
        [ 34,  36,  10],
        [105, 133,  59],
        [138, 173,  79]]], dtype=uint8)
```

```
# lets see the image of 0th index
import matplotlib.pyplot as plt

plt.figure(figsize=(20,2))              # reducing the image size to have more clarity
plt.imshow(x_train[0])
plt.show()                             # seems like its a cow
```




```
plt.figure(figsize=(20,2))              # reducing the image size to have more clarity
```

```
plt.imshow(x_train[-1])
plt.show()
```




```
y_train[-3:-1]
```

```
array([3, 7])
```

lets keep all 100 classes in one list

```
Class_dictionary = {0: 'apple',
 1: 'aquarium_fish',
 2: 'baby',
 3: 'bear',
 4: 'beaver',
 5: 'bed',
 6: 'bee',
 7: 'beetle',
 8: 'bicycle',
 9: 'bottle',
 10: 'bowl',
 11: 'boy',
 12: 'bridge',
 13: 'bus',
 14: 'butterfly',
 15: 'camel',
 16: 'can',
 17: 'castle',
 18: 'caterpillar',
 19: 'cattle',
 20: 'chair',
 21: 'chimpanzee',
 22: 'clock',
 23: 'cloud',
 24: 'cockroach',
 25: 'couch',
 26: 'crab',
 27: 'crocodile',
 28: 'cup',
 29: 'dinosaur',
 30: 'dolphin',
 31: 'elephant',
 32: 'flatfish',
 33: 'forest',
```

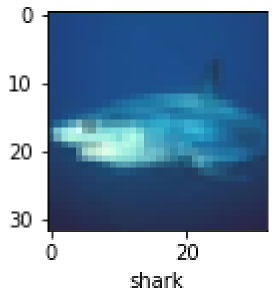
```
34: 'fox',
35: 'girl',
36: 'hamster',
37: 'house',
38: 'kangaroo',
39: 'computer_keyboard',
40: 'lamp',
41: 'lawn_mower',
42: 'leopard',
43: 'lion',
44: 'lizard',
45: 'lobster',
46: 'man',
47: 'maple_tree',
48: 'motorcycle',
49: 'mountain',
50: 'mouse',
51: 'mushroom',
52: 'oak_tree',
53: 'orange',
54: 'orchid',
55: 'otter',
56: 'palm_tree',
57: 'pear',
58: 'pickup_truck',
59: 'pine_tree',
60: 'plain',
61: 'plate',
62: 'poppy',
63: 'porcupine',
64: 'possum',
65: 'rabbit',
66: 'raccoon',
67: 'ray',
68: 'road',
69: 'rocket',
70: 'rose',
71: 'sea',
72: 'seal',
73: 'shark',
74: 'shrew',
75: 'skunk',
76: 'skyscraper',
77: 'snail',
78: 'snake',
79: 'spider',
80: 'squirrel',
81: 'streetcar',
82: 'sunflower',
83: 'sweet_pepper',
84: 'table',
85: 'tank',
86: 'telephone',
87: 'television',
88: 'tiger',
```

```
89: 'tractor',
90: 'train',
91: 'trout',
92: 'tulip',
93: 'turtle',
94: 'wardrobe',
95: 'whale',
96: 'willow_tree',
97: 'wolf',
98: 'woman',
99: 'worm'}
```

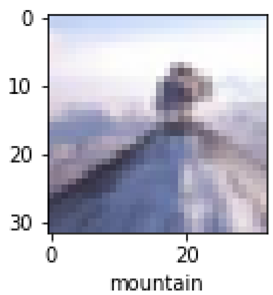
```
Class_dictionary[88]
```

```
'tiger'
```

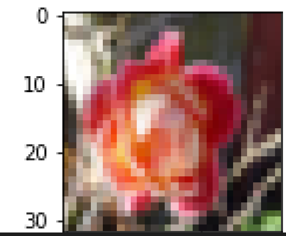
```
plt.figure(figsize=(20,2))              # reducing the image size to have more clarity
plt.imshow(x_train[-1])
plt.xlabel(Class_dictionary[y_train[-1]])
plt.show()
```




```
#  plot for test set
plt.figure(figsize=(20,2))
plt.imshow(x_test[0])
plt.xlabel(Class_dictionary[y_test[0]])
plt.show()
```




```
plt.figure(figsize=(20,2))
plt.imshow(x_test[-1])
plt.xlabel(y_test[-1])
plt.show()
```



```
Class_dictionary[70]                   # class belongs to index 70
```

```
'rose'
```

```
# categorical conversion on y sets for vectorization on mubers
# Syntax: tf.keras.utils.to_categorical(y, num_classes=None, dtype="float32")

num_classes= 100                    # number of classes we have

# Import libraries for preprocessing images
from tensorflow.keras.utils import to_categorical

# Normalize images
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
# Transform labels to one hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
y_train[0]                    # all are vectorized
```

```
array([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., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
      dtype=float32)
```

```
x_train[0]                    # we have them in range of 0 to 1
```

```
array([[[1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        ...,
        [0.7647059 , 0.8039216 , 0.75686276],
        [0.83137256, 0.8784314 , 0.8       ],
        [0.7137255 , 0.7607843 , 0.654902  ]],

       [[1.        , 1.        , 1.        ],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.6666667 , 0.6901961 , 0.5882353 ],
        [0.6313726 , 0.65882355, 0.50980395],
```

```
         [0.57254905, 0.6039216 , 0.44313726]],

        [[1.        , 1.        , 1.        ],
         [0.99607843, 0.99607843, 0.99607843],
         [1.        , 1.        , 1.        ],
         ...,
         [0.7411765 , 0.78039217, 0.6627451 ],
         [0.6509804 , 0.69803923, 0.50980395],
         [0.4745098 , 0.52156866, 0.34117648]],

        ...,

        [[0.5803922 , 0.7254902 , 0.30980393],
         [0.5568628 , 0.7137255 , 0.22352941],
         [0.54901963, 0.7019608 , 0.23529412],
         ...,
         [0.11764706, 0.06666667, 0.00392157],
         [0.25490198, 0.24313726, 0.05882353],
         [0.29803923, 0.3019608 , 0.07843138]],

        [[0.47843137, 0.6156863 , 0.25882354],
         [0.47058824, 0.60784316, 0.22745098],
         [0.49411765, 0.627451  , 0.2784314 ],
         ...,
         [0.08627451, 0.0627451 , 0.01176471],
         [0.38039216, 0.4392157 , 0.21960784],
         [0.5529412 , 0.6313726 , 0.34117648]],

        [[0.34117648, 0.47843137, 0.16078432],
         [0.34509805, 0.47843137, 0.15294118],
         [0.39607844, 0.5254902 , 0.21960784],
         ...,
         [0.13333334, 0.14117648, 0.03921569],
         [0.4117647 , 0.52156866, 0.23137255],
         [0.5411765 , 0.6784314 , 0.30980393]]], dtype=float32)
```

Will go for CNN with batch normalization model

```
# Import Libraries for CNN
from keras.models import Sequential
from keras.layers import Conv2D, Flatten, Dense, Activation, Dropout, BatchNormalizatio

from keras.layers.pooling import MaxPool2D
from keras.layers.core import Dense,Activation,Dropout,Flatten
```

```
model = Sequential()

model.add(Conv2D(256,(3,3),padding='same',input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(256,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

```
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(100,activation='softmax'))

```

```
model.summary()
```

```
 max_pooling2d_1 (MaxPooling  (None, 8, 8, 512)        0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 512)         0

 conv2d_4 (Conv2D)           (None, 8, 8, 512)         2359808

 batch_normalization_4 (Batc  (None, 8, 8, 512)        2048
 hNormalization)

 activation_4 (Activation)   (None, 8, 8, 512)         0

 conv2d_5 (Conv2D)           (None, 8, 8, 512)         2359808
```

```
 conv2d_5 (Conv2D)           (None, 8, 8, 512)         2359808

 batch_normalization_5 (Batc  (None, 8, 8, 512)        2048
 hNormalization)

 activation_5 (Activation)   (None, 8, 8, 512)         0

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 512)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 512)         0

 conv2d_6 (Conv2D)           (None, 4, 4, 512)         2359808

 batch_normalization_6 (Batc  (None, 4, 4, 512)        2048
 hNormalization)

 activation_6 (Activation)   (None, 4, 4, 512)         0

 conv2d_7 (Conv2D)           (None, 4, 4, 512)         2359808

 batch_normalization_7 (Batc  (None, 4, 4, 512)        2048
 hNormalization)

 activation_7 (Activation)   (None, 4, 4, 512)         0

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 512)        0
 2D)

 dropout_3 (Dropout)         (None, 2, 2, 512)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 activation_8 (Activation)   (None, 1024)              0

 dropout_4 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 100)               102500

=================================================================
Total params: 15,791,460
Trainable params: 15,784,292
Non-trainable params: 7,168
_________________________________________________________________
```

```
#  compiling the model
from tensorflow.keras.optimizers import Adam
model.compile(loss='categorical_crossentropy',
              optimizer= Adam(learning_rate=1e-4),
              metrics=['accuracy'])
```

Training the model

```
# will keep the model history in a variable history so that we can plot it later
```

```
# will keep the model history in a variable history so that we can plot it later
model.fit(x_train, y_train, batch_size= 64, epochs= 16, validation_data=(x_test, y_test
```

```
Epoch 1/16
782/782 [==============================] - 96s 104ms/step - loss: 3.8770 - accuracy:
Epoch 2/16
782/782 [==============================] - 80s 102ms/step - loss: 3.1106 - accuracy:
Epoch 3/16
782/782 [==============================] - 80s 102ms/step - loss: 2.5618 - accuracy:
Epoch 4/16
782/782 [==============================] - 80s 102ms/step - loss: 2.1818 - accuracy:
Epoch 5/16
782/782 [==============================] - 80s 102ms/step - loss: 1.8989 - accuracy:
Epoch 6/16
782/782 [==============================] - 80s 102ms/step - loss: 1.6691 - accuracy:
Epoch 7/16
782/782 [==============================] - 80s 102ms/step - loss: 1.4850 - accuracy:
Epoch 8/16
782/782 [==============================] - 80s 102ms/step - loss: 1.3259 - accuracy:
Epoch 9/16
782/782 [==============================] - 80s 102ms/step - loss: 1.1837 - accuracy:
Epoch 10/16
782/782 [==============================] - 80s 102ms/step - loss: 1.0506 - accuracy:
Epoch 11/16
782/782 [==============================] - 80s 102ms/step - loss: 0.9333 - accuracy:
Epoch 12/16
782/782 [==============================] - 80s 102ms/step - loss: 0.8218 - accuracy:
Epoch 13/16
782/782 [==============================] - 80s 102ms/step - loss: 0.7241 - accuracy:
Epoch 14/16
782/782 [==============================] - 80s 102ms/step - loss: 0.6329 - accuracy:
Epoch 15/16
782/782 [==============================] - 80s 102ms/step - loss: 0.5572 - accuracy:
Epoch 16/16
782/782 [==============================] - 80s 102ms/step - loss: 0.4940 - accuracy:
<keras.callbacks.History at 0x7f1a4031f390>
```

```
# model accuarcy on test set

scores = model.evaluate(x_test, y_test)
print(f'accuracy on test set: {model.metrics_names[1]} of {scores[1]*100}')
```

```
313/313 [==============================] - 7s 18ms/step - loss: 1.5067 - accuracy: 0
accuracy on test set: accuracy of 62.41000294685364
```

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

```
print('predicted output: ', np.argmax(y_pred[3]) )                # comparing predicted
print('actual ouput: ', np.argmax(y_test[3]))

```

```
predicted output:  51
actual ouput:  51
```

```
# Class_dictionary[51]                # checing what class is this
```

```
print('predicted output: ', np.argmax(y_pred[50]) )                # comparing predicte
print('actual ouput: ', np.argmax(y_test[50]))
```

```
predicted output:  4
actual ouput:  4
```

```
y_pred.shape         # y_pred is 2D ; so we should make it 1D
```

```
(10000, 100)
```

```
# from each index, we will take only the max mumber from each array of y_test and y_pre
import numpy as np
from numpy import argmax
prediction = []
true_labels = []

# pred = model.predict(test_imxages)
print(y_test.shape[0])
for i in range(y_test.shape[0]):
  prediction.append(argmax(y_pred[i]))
  true_labels.append(argmax(y_test[i]))
```

```
10000
```

```
len(prediction), len(true_labels)
```

```
(10000, 10000)
```

```
true_labels[:5]
```

```
[49, 33, 72, 51, 71]
```

```
prediction[:5]
```

```
[90, 33, 93, 51, 71]
```

```
# Calculating f1 score

from sklearn.metrics import f1_score
print(f"f1 score: {f1_score(true_labels, prediction, average='weighted')}")
```

```
f1 score: 0.6256260864142461
```

will put all the classes in the list for getting a classification report on each class so that we get to know which class

```
# Name of all classes in CIFAR-100
classes = ['beaver', 'dolphin', 'otter', 'seal', 'whale',
'aquarium' ,'fish', 'ray', 'shark', 'trout',
'orchids', 'poppies', 'roses', 'sunflowers', 'tulips',
'bottles', 'bowls', 'cans', 'cups', 'plates',
'apples', 'mushrooms', 'oranges', 'pears', 'sweet peppers',
'clock', 'computer keyboard', 'lamp', 'telephone', 'television', 'bed', 'chair', 'couch
'bee', 'beetle', 'butterfly', 'caterpillar', 'cockroach',
'bear', 'leopard', 'lion', 'tiger', 'wolf',
'bridge', 'castle', 'house', 'road', 'skyscraper',
'cloud', 'forest', 'mountain', 'plain', 'sea',
'camel', 'cattle', 'chimpanzee', 'elephant', 'kangaroo',
'fox', 'porcupine', 'possum', 'raccoon', 'skunk',
'crab', 'lobster', 'snail', 'spider', 'worm',
'baby', 'boy', 'girl', 'man', 'woman',
'crocodile', 'dinosaur', 'lizard', 'snake', 'turtle',
'hamster', 'mouse', 'rabbit', 'shrew', 'squirrel',
'maple', 'oak', 'palm', 'pine', 'willow',
'bicycle', 'bus', 'motorcycle', 'pickup truck', 'train',
'lawn-mower', 'rocket', 'streetcar', 'tank', 'tractor']
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(true_labels, prediction, target_names= classes))
```

```
            shark       0.81      0.82      0.82       100
            trout       0.66      0.79      0.72       100
          orchids       0.68      0.30      0.42       100
          poppies       0.46      0.36      0.40       100
            roses       0.52      0.78      0.62       100
       sunflowers       0.73      0.40      0.52       100
           tulips       0.70      0.53      0.60       100
          bottles       0.53      0.72      0.61       100
            bowls       0.88      0.58      0.70       100
             cans       0.76      0.81      0.79       100
             cups       0.52      0.62      0.57       100
           plates       0.57      0.63      0.60       100

           apples       0.84      0.80      0.82       100
        mushrooms       0.68      0.78      0.73       100
          oranges       0.59      0.60      0.59       100
            pears       0.81      0.71      0.76       100
    sweet peppers       0.71      0.79      0.75       100
            clock       0.50      0.53      0.52       100
computer keyboard       0.68      0.52      0.59       100
             lamp       0.61      0.34      0.44       100
        telephone       0.91      0.69      0.78       100
       television       0.60      0.54      0.57       100
```

```
        bed       0.54      0.59      0.56       100
      chair       0.60      0.58      0.59       100
      couch       0.54      0.51      0.52       100
      table       0.54      0.64      0.59       100
   wardrobe       0.69      0.56      0.62       100
        bee       0.39      0.44      0.41       100
     beetle       0.72      0.69      0.70       100
  butterfly       0.70      0.61      0.65       100
caterpillar       0.45      0.54      0.49       100
  cockroach       0.68      0.84      0.75       100
       bear       0.61      0.49      0.54       100
    leopard       0.86      0.83      0.85       100
       lion       0.84      0.49      0.62       100
      tiger       0.68      0.68      0.68       100
       wolf       0.35      0.48      0.40       100
     bridge       0.51      0.52      0.51       100
     castle       0.46      0.32      0.38       100
      house       0.74      0.49      0.59       100
       road       0.88      0.84      0.86       100
 skyscraper       0.93      0.62      0.74       100
      cloud       0.32      0.55      0.40       100
     forest       0.70      0.69      0.70       100
   mountain       0.63      0.62      0.63       100
      plain       0.87      0.82      0.85       100
        sea       0.76      0.81      0.78       100
      camel       0.30      0.45      0.36       100
     cattle       0.82      0.82      0.82       100
 chimpanzee       0.75      0.67      0.71       100
   elephant       0.87      0.68      0.76       100
   kangaroo       0.54      0.78      0.64       100
        fox       0.61      0.92      0.74       100
  porcupine       0.69      0.66      0.68       100
     possum       0.77      0.62      0.69       100
    raccoon       0.81      0.52      0.63       100
      skunk       0.56      0.53      0.54       100
```

```
1
```

```
1
```

0s completed at 11:11 PM