```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras import datasets, layers, models
```

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

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

```
X_test.shape
```

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

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
# y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to
y_train = y_train.reshape(-1,)
y_train[:5]
```

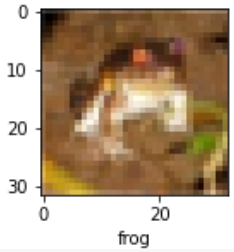
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

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

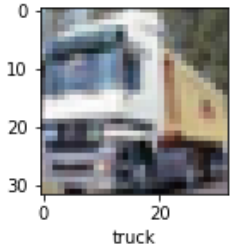
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

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

```
plot_sample(X_train, y_train, 0)
```




```
plot_sample(X_train, y_train, 1)
```




```
# Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel

X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
# model building

model = Sequential()
model.add(Flatten(input_shape = (32,32, 3)))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))

# add output layer
model.add(Dense(10, activation = 'softmax'))

model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

history = model.fit(X_train, y_train, epochs = 40, validation_split = 0.2)
```

```
Epoch 1/40
1250/1250 [==============================] - 15s 11ms/step - loss: 1.8789 - accuracy: 0.3202 - val_
Epoch 2/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.7112 - accuracy: 0.3853 - val_
Epoch 3/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.6411 - accuracy: 0.4107 - val_
Epoch 4/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.5861 - accuracy: 0.4341 - val_
Epoch 5/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.5533 - accuracy: 0.4421 - val_
Epoch 6/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.5273 - accuracy: 0.4544 - val_
Epoch 7/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.5104 - accuracy: 0.4572 - val_
Epoch 8/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.4833 - accuracy: 0.4703 - val_
Epoch 9/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.4686 - accuracy: 0.4747 - val_
Epoch 10/40
1250/1250 [==============================] - 11s 9ms/step - loss: 1.4561 - accuracy: 0.4784 - val_
Epoch 11/40
1250/1250 [==============================] - 11s 9ms/step - loss: 1.4457 - accuracy: 0.4816 - val_
Epoch 12/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.4311 - accuracy: 0.4877 - val_
Epoch 13/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.4185 - accuracy: 0.4915 - val_
```

```
Epoch 14/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.4118 - accuracy: 0.4952 - val_
Epoch 15/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.4035 - accuracy: 0.4971 - val_
Epoch 16/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.3904 - accuracy: 0.5000 - val_
Epoch 17/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.3837 - accuracy: 0.5048 - val_l
Epoch 18/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.3791 - accuracy: 0.5053 - val_l
Epoch 19/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.3731 - accuracy: 0.5078 - val_l
Epoch 20/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.3686 - accuracy: 0.5076 - val_
Epoch 21/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.3617 - accuracy: 0.5121 - val_l
Epoch 22/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.3529 - accuracy: 0.5135 - val_
Epoch 23/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.3445 - accuracy: 0.5181 - val_
Epoch 24/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.3382 - accuracy: 0.5194 - val_
Epoch 25/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.3386 - accuracy: 0.5216 - val_
Epoch 26/40
1250/1250 [==============================] - 12s 9ms/step - loss: 1.3351 - accuracy: 0.5217 - val_l
Epoch 27/40
1250/1250 [==============================] - 13s 10ms/step - loss: 1.3306 - accuracy: 0.5244 - val_
Epoch 28/40
1250/1250 [==============================] - 12s 10ms/step - loss: 1.3238 - accuracy: 0.5253 - val
```

```
yprob=model.predict(X_test)
yprob[0]
```

```
313/313 [==============================] - 3s 6ms/step
array([0.02376734, 0.03564779, 0.11818496, 0.2855256 , 0.15272914,
       0.28611714, 0.00505482, 0.01683522, 0.061261  , 0.01487709],
      dtype=float32)
```

```
ypred=yprob.argmax(axis=1)
ypred
```

```
array([5, 8, 8, ..., 5, 4, 7])
```

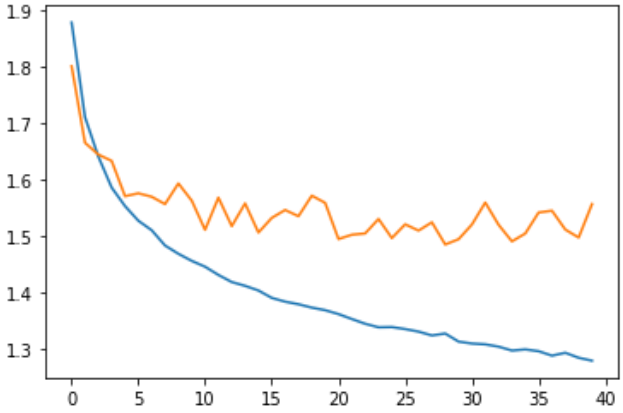
```
from sklearn.metrics import classification_report
print(classification_report(y_test,ypred))
```

```
              precision    recall  f1-score   support

           0       0.54      0.45      0.49      1000
           1       0.65      0.50      0.57      1000
           2       0.38      0.27      0.32      1000
           3       0.42      0.13      0.20      1000
           4       0.33      0.54      0.41      1000
           5       0.41      0.38      0.40      1000
           6       0.49      0.50      0.49      1000
           7       0.43      0.56      0.49      1000
           8       0.53      0.68      0.59      1000
           9       0.48      0.57      0.52      1000

    accuracy                           0.46     10000
   macro avg       0.47      0.46      0.45     10000
weighted avg       0.47      0.46      0.45     10000
```

```
plt.plot(history.history["loss"])
```

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
```

```
[<matplotlib.lines.Line2D at 0x7f299171b9d0>]
```



Colab paid products - Cancel contracts here