```
from tensorflow.keras import datasets
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
load_data=datasets.cifar100.load_data()
load_data
```

```
             [[[246, 246, 242],
               [240, 238, 232],
               [214, 212, 199],
               ...,
               [ 74,  32,  35],
               [ 77,  34,  37],
               [ 81,  34,  35]],

              [[210, 205, 196],
               [243, 240, 230],
               [229, 225, 214],
               ...,
               [ 75,  33,  35],
               [ 79,  35,  38],
               [ 83,  34,  36]],

              [[144, 134, 112],
               [175, 163, 144],
               [158, 144, 130],
               ...,
               [ 74,  33,  35],
               [ 79,  35,  38],
               [ 82,  33,  36]],

              ...,

              [[198, 190, 176],
               [111, 111,  66],
               [ 58,  55,  27],
               ...,
               [ 62,  81,  41],
               [ 72, 100,  41],
               [ 80, 107,  49]],

              [[167, 160, 144],
               [ 62,  64,  27],
               [ 85,  85,  68],
               ...,
               [ 92, 126,  58],
               [143, 183, 104],
               [160, 199, 118]],

              [[115, 108,  94],
               [ 42,  37,  21],
               [139, 136, 127],
               ...,
               [139, 172, 114],
               [167, 204, 141],
               [146, 182, 118]]]], dtype=uint8),
       array([[49],
              [33],
              [72],
              ...,
              [51],
              [42],
              [70]]))
```

```
(x_train,y_train),(x_test,y_test)=load_data
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

```
((50000, 32, 32, 3), (50000, 1), (10000, 32, 32, 3), (10000, 1))
```

```
min(y_train),max(y_train)
```

```
(array([0]), array([99]))
```

```
plt.figure(figsize=(10,16))
for i in range(100):
    plt.subplot(20,5,i+1)
    plt.imshow(x_train[i])
    plt.xlabel(y_train[i])
    plt.colorbar()
plt.show()
```

Normalize the image

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

Create neural netork

```
model=Sequential()

#input layer
model.add(Flatten(input_shape=(32,32,3)))

#hidden layer
model.add(Dense(128,activation="relu"))
model.add(Dense(64,activation="relu"))
model.add(Dense(32,activation="relu"))

#output layer
model.add(Dense(100,activation="softmax"))

model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(**kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 3072)	0
dense_7 (Dense)	(None, 128)	393,344
dense_8 (Dense)	(None, 64)	8,256
dense_9 (Dense)	(None, 32)	2,080
dense_10 (Dense)	(None, 100)	3,300

```
Total params: 406,980 (1.55 MB)
Trainable params: 406,980 (1.55 MB)
```

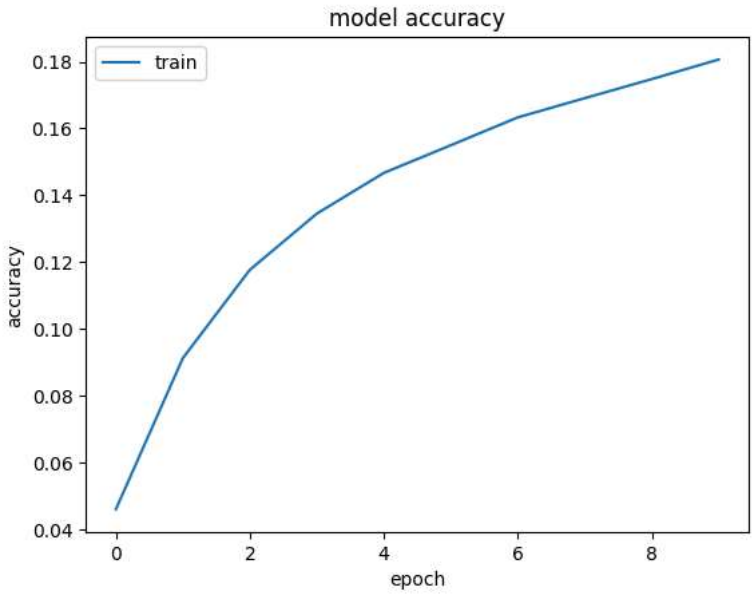


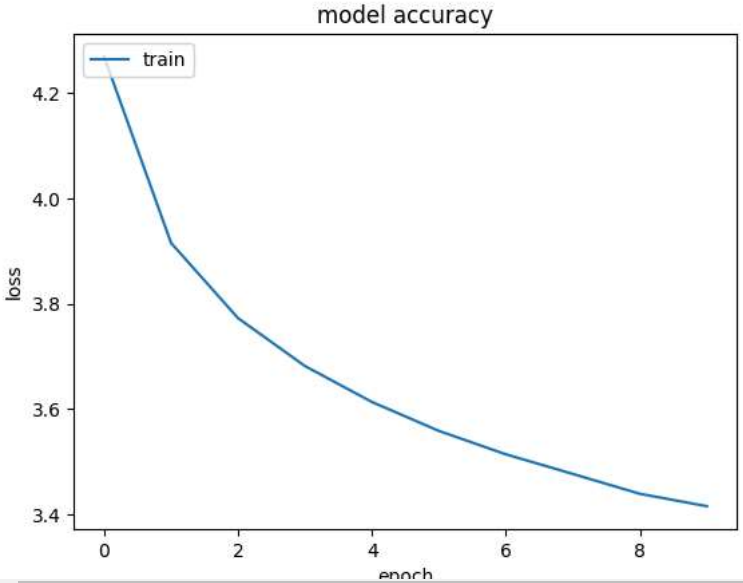
```
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

history=model.fit(x_train,
                  y_train,
                  epochs=10,
                  validation_data=(x_test,y_test),
                  batch_size=64,
                  verbose=True)
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.0262 - loss: 4.4540 - val_accuracy: 0.0742 - val_loss: 4.0407
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.0851 - loss: 3.9441 - val_accuracy: 0.0988 - val_loss: 3.8662
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 10ms/step - accuracy: 0.1159 - loss: 3.7895 - val_accuracy: 0.1235 - val_loss: 3.7480
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.1336 - loss: 3.6830 - val_accuracy: 0.1429 - val_loss: 3.6926
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.1479 - loss: 3.6115 - val_accuracy: 0.1448 - val_loss: 3.6511
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - accuracy: 0.1511 - loss: 3.5637 - val_accuracy: 0.1525 - val_loss: 3.6229
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.1610 - loss: 3.5141 - val_accuracy: 0.1623 - val_loss: 3.5531
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 11ms/step - accuracy: 0.1693 - loss: 3.4772 - val_accuracy: 0.1705 - val_loss: 3.5238
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.1737 - loss: 3.4404 - val_accuracy: 0.1736 - val_loss: 3.5087
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.1789 - loss: 3.4223 - val_accuracy: 0.1697 - val_loss: 3.4924
```

Evaluation the model

```
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()
```






```
test_loss,test_accuracy=model.evaluate(x_test,y_test)
print("Test Loss: ",test_loss)
print("Test Accuracy: ",test_accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1700 - loss: 3.4928
Test Loss:  3.4924304485321045
Test Accuracy:  0.1696999967098236
```

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[1.1419026e-05, 2.7707010e-04, 2.5688522e-03, ..., 1.4955287e-03,
        4.9699057e-04, 6.0108858e-03],
       [1.1900964e-04, 9.8066265e-04, 1.9675605e-03, ..., 1.7086994e-02,
        6.6163263e-04, 2.4320973e-02],
       [3.3979528e-04, 1.1853998e-03, 4.1729985e-03, ..., 7.0962459e-03,
        3.2206541e-03, 1.4013842e-04],
       ...,
       [2.4347380e-02, 9.3268519e-03, 1.7814628e-04, ..., 1.3687591e-03,
        3.3769262e-04, 2.9882709e-02],
       [1.7253592e-03, 4.0978249e-03, 1.0072134e-02, ..., 1.8652624e-02,
        7.8726001e-03, 1.1716263e-02],
       [7.6772317e-02, 5.7872724e-02, 1.2218509e-02, ..., 7.5582375e-06,
        5.0205751e-03, 4.3883469e-04]], dtype=float32)
```

Prediction_details

```
max_prob=np.max(y_predict[0])
index=np.argmax(y_predict[0])
print(max_prob)
print(index)
```

```
0.1890933
95
```

```
# max_prob=[np.max(y_predict[i]) for i in range(100)]
# index=[np.argmax(y_predict[i]) for i in range(100)]
# predict_class=[i for i in [i for i in index]]
# ground_truth_class=[i[0] for i in [i for i in y_test]]

# df=pd.DataFrame(list(zip(predict_class,ground_truth_class,max_prob,index)),
#                 columns=["predict_class","ground_truth_class","max_prob","index"])
# df
```

	predict_class	ground_truth_class	max_prob	index
0	95	49	0.189093	95
1	74	33	0.080984	74
2	56	72	0.089622	56
3	11	51	0.061609	11
4	92	71	0.250650	92
...	...	...	...	...
95	31	21	0.088731	31
96	29	50	0.078197	29
97	29	75	0.058143	29
98	68	37	0.172054	68
99	94	35	0.163473	94

100 rows × 4 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Start coding or generate with AI.