Import necessary packages.

In [ ]:

```
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.datasets import mnist
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import numpy as
```

Load the training and testing data (MINIST/CIFAR10)

In [ ]:

```
# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)
print("[INFO] accessing MNIST...")
((trainX, trainY), (testX, testY)) = mnist.load_data()

# each image in the MNIST dataset is represented as a 28x28x1
# image, but in order to apply a standard neural network we must
# first "flatten" the image to be simple list of 28x28=784 pixels
trainX = trainX.reshape((trainX.shape[0], 28 * 28 * 1))
testX = testX.reshape((testX.shape[0], 28 * 28 * 1))

# scale data to the range of [0, 1]
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0
```

In [3]:

```
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
```

Define the network architecture using keras

In [4]:

```
# define the 784-256-128-10 architecture using Keras
model = Sequential()
model.add(Dense(256, input_shape=(784,), activation="sigmoid"))
model.add(Dense(128, activation="sigmoid"))
model.add(Dense(10, activation="softmax"))
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Train the model using SGD

In [5]:

```
# train the model using SGD
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd,
    metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),
    epochs=100, batch_size=32)
```

```
[INFO] training network...
Epoch 1/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.2482 - loss: 2.23
62 - val_accuracy: 0.4821 - val_loss: 1.8935
Epoch 2/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.5991 - loss: 1.70
11 - val_accuracy: 0.7827 - val_loss: 1.1291
Epoch 3/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7766 - loss: 1.02
48 - val_accuracy: 0.8274 - val_loss: 0.7469
Epoch 4/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8322 - loss: 0.70
98 - val_accuracy: 0.8564 - val_loss: 0.5774
Epoch 5/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8554 - loss: 0.57
10 - val_accuracy: 0.8732 - val_loss: 0.4869
Epoch 6/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8709 - loss: 0.49
17 - val_accuracy: 0.8870 - val_loss: 0.4340
```

Evaluate the network

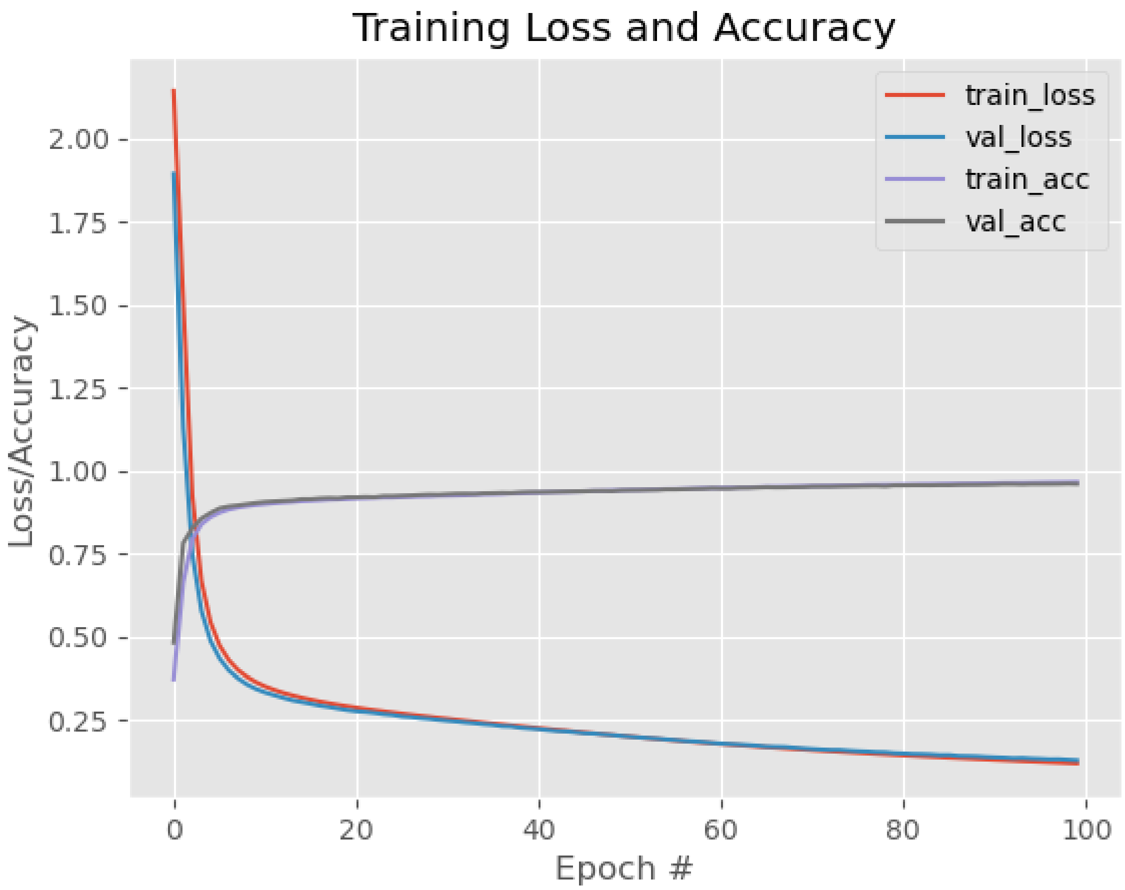
In [7]:

```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=128)
print(classification_report(testY.argmax(axis=1),
    predictions.argmax(axis=1),
    target_names=[str(x) for x in lb.classes_]))
```

```
[INFO] evaluating network...
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.98      0.98      0.98      1135
           2       0.97      0.96      0.96      1032
           3       0.95      0.97      0.96      1010
           4       0.97      0.96      0.96       982
           5       0.96      0.94      0.95       892
           6       0.96      0.97      0.96       958
           7       0.96      0.95      0.95      1028
           8       0.95      0.95      0.95       974
           9       0.94      0.94      0.94      1009

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

In [8]:

```
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()
```



In [9]:

```
# load the training and testing data, scale it into the range [0, 1],
# then reshape the design matrix
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
trainX = trainX.reshape((trainX.shape[0], 3072))
testX = testX.reshape((testX.shape[0], 3072))
```

```
[INFO] loading CIFAR-10 data...
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.g
z (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 207s 1us/step
```

In [10]:

```
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
    "dog", "frog", "horse", "ship", "truck"]
```

Define the network architecture using Keras (Relu)

In [11]:

```
# define the 3072-1024-512-10 architecture using Keras
model = Sequential()
model.add(Dense(1024, input_shape=(3072,), activation="relu"))
model.add(Dense(512, activation="relu"))
model.add(Dense(10, activation="softmax"))
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [12]:

```
# train the model using SGD
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd,
    metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),
    epochs=100, batch_size=32)
```

```
[INFO] training network...
Epoch 1/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.2937 - loss: 1.9
516 - val_accuracy: 0.3603 - val_loss: 1.7878
Epoch 2/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.4043 - loss: 1.6
755 - val_accuracy: 0.4305 - val_loss: 1.6198
Epoch 3/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.4404 - loss: 1.5
785 - val_accuracy: 0.4354 - val_loss: 1.6121
Epoch 4/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.4638 - loss: 1.5
202 - val_accuracy: 0.4350 - val_loss: 1.5764
Epoch 5/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.4798 - loss: 1.4
721 - val_accuracy: 0.4540 - val_loss: 1.5307
Epoch 6/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.5023 - loss: 1.4
205 - val_accuracy: 0.4784 - val_loss: 1.4796
```

evaluate the network

In [13]:

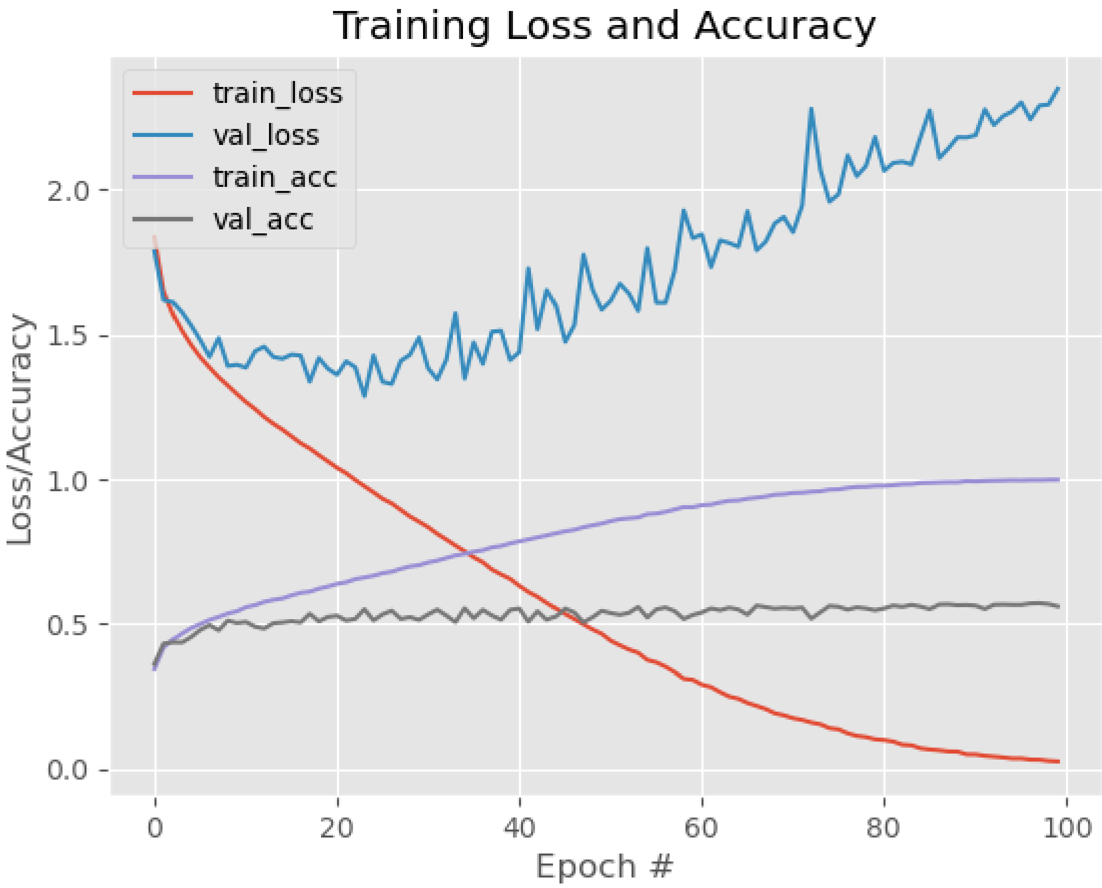
```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
    predictions.argmax(axis=1), target_names=labelNames))
```

```
[INFO] evaluating network...
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
              precision    recall  f1-score   support

    airplane       0.66      0.63      0.65      1000
  automobile       0.70      0.63      0.66      1000
        bird       0.47      0.43      0.45      1000
         cat       0.35      0.52      0.42      1000
        deer       0.52      0.45      0.48      1000
         dog       0.46      0.46      0.46      1000
        frog       0.66      0.55      0.60      1000
       horse       0.67      0.58      0.63      1000
        ship       0.63      0.73      0.68      1000
       truck       0.60      0.61      0.60      1000

    accuracy                           0.56     10000
   macro avg       0.57      0.56      0.56     10000
weighted avg       0.57      0.56      0.56     10000
```

plot the training loss and accuracy

In [14]:

```
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()
```



In [ ]: