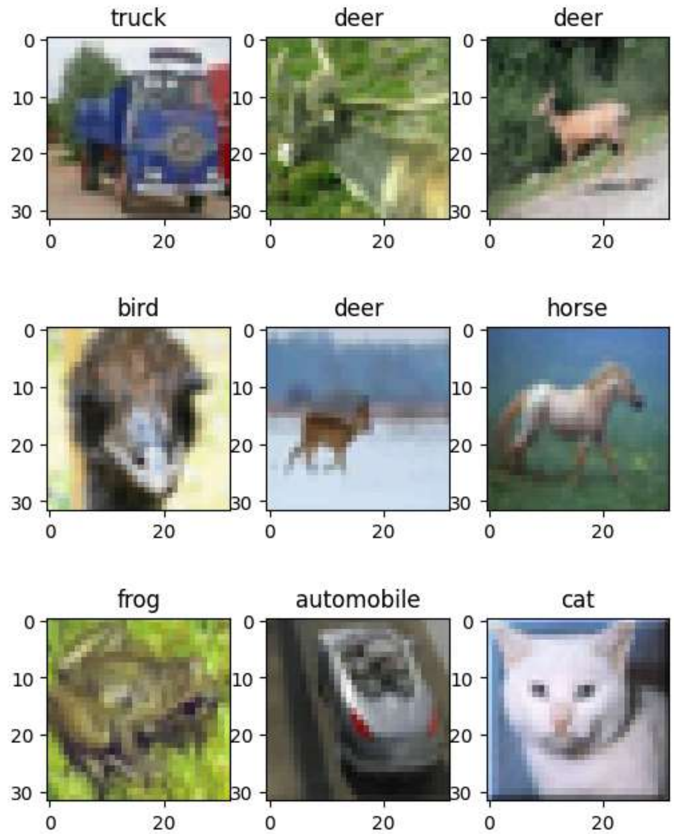
```
import os
import time
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
import tensorflow_datasets as tfds
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import (models,layers,)
from matplotlib import pyplot as plt
import numpy as np
import random
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt6


cifar_10_labels = {
0: 'airplane',
1: 'automobile',
2: 'bird',
3: 'cat',
4: 'deer',
5: 'dog',
6: 'frog',
7: 'horse',
8: 'ship',
9: 'truck'
}


model_directory = 'models'

class RandomIntegers:
    def __init__(self):
        pass

    def generate(self, n, length):
        # Generate n unique random integers between 0 and length
        return random.sample(range(length), n)

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0



def display_images():
    random_integers = RandomIntegers().generate(9, len(x_train))
    plt.figure(figsize=(6, 8))
    for counter, i in enumerate(random_integers):
        plt.subplot(3, 3, counter + 1)
        plt.imshow(x_train[i])
        plt.title(cifar_10_labels[y_train[i][0]])
    plt.show()

display_images()

# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()]
)
```



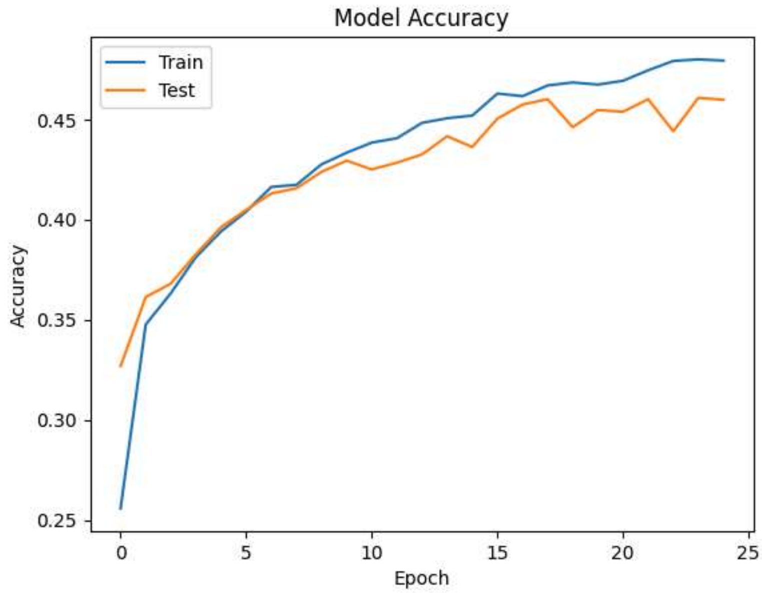

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(**kwargs)
```

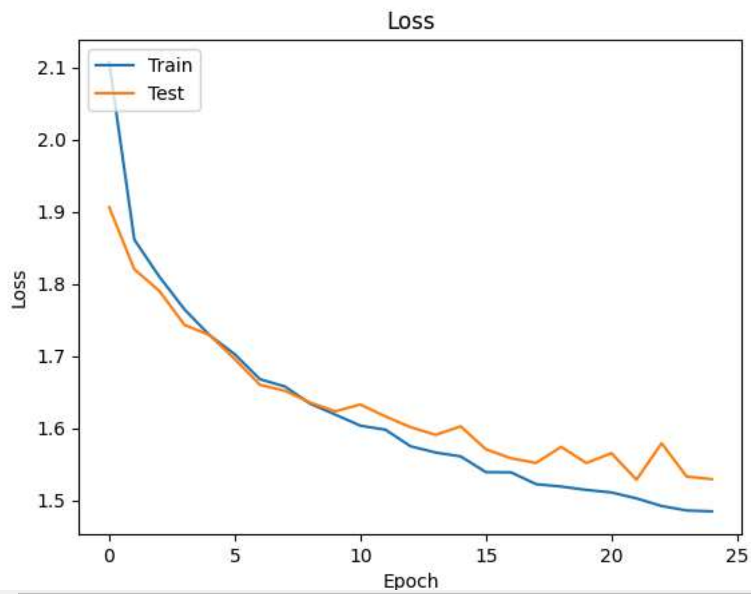
```
# Train the model
start_time = time.time()
history = model.fit(
    x_train, y_train,
    epochs=25,
    batch_size=512,
    validation_data=(x_test, y_test)
)
end_time = time.time()

# Print training time
total_time = end_time - start_time
print("Time taken for training: ", total_time, " seconds")
```

```
Epoch 1/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 5s 38ms/step - loss: 2.3474 - sparse_categorical_accuracy: 0.1974 - val_loss: 1.9062 - val_sparse_categorical
Epoch 2/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 5s 37ms/step - loss: 1.8769 - sparse_categorical_accuracy: 0.3413 - val_loss: 1.8202 - val_sparse_categorical
Epoch 3/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - loss: 1.8138 - sparse_categorical_accuracy: 0.3612 - val_loss: 1.7901 - val_sparse_categorical
Epoch 4/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - loss: 1.7734 - sparse_categorical_accuracy: 0.3796 - val_loss: 1.7431 - val_sparse_categorical
Epoch 5/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 8s 38ms/step - loss: 1.7320 - sparse_categorical_accuracy: 0.3918 - val_loss: 1.7289 - val_sparse_categorical
Epoch 6/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 6s 57ms/step - loss: 1.7080 - sparse_categorical_accuracy: 0.4019 - val_loss: 1.6956 - val_sparse_categorical
Epoch 7/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.6725 - sparse_categorical_accuracy: 0.4123 - val_loss: 1.6601 - val_sparse_categorical
Epoch 8/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.6600 - sparse_categorical_accuracy: 0.4149 - val_loss: 1.6517 - val_sparse_categorical
Epoch 9/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - loss: 1.6412 - sparse_categorical_accuracy: 0.4219 - val_loss: 1.6355 - val_sparse_categorical
Epoch 10/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - loss: 1.6256 - sparse_categorical_accuracy: 0.4294 - val_loss: 1.6231 - val_sparse_categorical
Epoch 11/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.6027 - sparse_categorical_accuracy: 0.4404 - val_loss: 1.6327 - val_sparse_categorical
Epoch 12/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.5938 - sparse_categorical_accuracy: 0.4405 - val_loss: 1.6162 - val_sparse_categorical
Epoch 13/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - loss: 1.5734 - sparse_categorical_accuracy: 0.4471 - val_loss: 1.6013 - val_sparse_categorical
Epoch 14/25
```

```
98/98 ━━━━━━━━━━━━━━━━━━━━ 4s 31ms/step - loss: 1.5690 - sparse_categorical_accuracy: 0.4522 - val_loss: 1.5907 - val_sparse_categorical
Epoch 15/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.5628 - sparse_categorical_accuracy: 0.4518 - val_loss: 1.6024 - val_sparse_categorical
Epoch 16/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - loss: 1.5398 - sparse_categorical_accuracy: 0.4661 - val_loss: 1.5707 - val_sparse_categorical
Epoch 17/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - loss: 1.5402 - sparse_categorical_accuracy: 0.4620 - val_loss: 1.5582 - val_sparse_categorical
Epoch 18/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - loss: 1.5234 - sparse_categorical_accuracy: 0.4640 - val_loss: 1.5517 - val_sparse_categorical
Epoch 19/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 6s 44ms/step - loss: 1.5129 - sparse_categorical_accuracy: 0.4709 - val_loss: 1.5740 - val_sparse_categorical
Epoch 20/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - loss: 1.5141 - sparse_categorical_accuracy: 0.4671 - val_loss: 1.5516 - val_sparse_categorical
Epoch 21/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - loss: 1.5210 - sparse_categorical_accuracy: 0.4660 - val_loss: 1.5652 - val_sparse_categorical
Epoch 22/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - loss: 1.5014 - sparse_categorical_accuracy: 0.4735 - val_loss: 1.5284 - val_sparse_categorical
Epoch 23/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - loss: 1.4857 - sparse_categorical_accuracy: 0.4819 - val_loss: 1.5789 - val_sparse_categorical
Epoch 24/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.4920 - sparse_categorical_accuracy: 0.4768 - val_loss: 1.5328 - val_sparse_categorical
Epoch 25/25
98/98 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - loss: 1.4788 - sparse_categorical_accuracy: 0.4831 - val_loss: 1.5292 - val_sparse_categorical
Time taken for training:  114.48651099205017  seconds
```

```python
# Plot accuracy
plt.plot(history.history['sparse_categorical_accuracy'])
plt.plot(history.history['val_sparse_categorical_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```






```
# Save the model
if not os.path.exists(model_directory):
    os.makedirs(model_directory)
model_path = os.path.join(model_directory, "model_nn_xavier.h5")
model.save(model_path)

# Load the saved model
loaded_model = tf.keras.models.load_model(model_path)
loaded_model.summary()
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 3072)	0
dense_5 (Dense)	(None, 128)	393,344
dense_6 (Dense)	(None, 10)	1,290

Total params: 394,636 (1.51 MB)
Trainable params: 394,634 (1.51 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)

```
# Predict and evaluate
predicted_labels = np.argmax(model.predict(x_test), axis=-1)

# Flatten true labels for consistency
true_labels = y_test.flatten()

# Generate the confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Plot the confusion matrix
plt.imshow(cm, cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(10), cifar_10_labels.values())
plt.yticks(np.arange(10), cifar_10_labels.values())
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```