```
With ten epochs it resulted in: Training loss (for one batch) at step 100: 0.
018822837620973587. This was decreasing nicely. So, I think I will adjust the
epochs to 20 to see how much better it can get. After 20 epochs it resulted in:
Training loss (for one batch) at step 100: 0.017837101593613625. This is better
and it is still getting better. So, more epochs may be reasonable.
```

With ten epochs it resulted in: Training loss (for one batch) at step 100: 0.018822837620973587. This was decreasing nicely. So, I think I will adjust the epochs to 20 to see how much better it can get. After 20 epochs it resulted in: Training loss (for one batch) at step 100: 0.017837101593613625. This is better and it is still getting better. So, more epochs may be reasonable.

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Load and preprocess the CIFAR-10 dataset
(x_train, _), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Pre-scaled data

# Define the data augmentation function and apply it to the training dataset
data_augmentation = keras.Sequential([
    layers.RandomCrop(height=32, width=32),
    layers.RandomFlip("horizontal"),
    layers.RandomZoom(0.2),
])

batch_size = 256

# Convert the training data into a TensorFlow Dataset, apply augmentation, cache and prefetch
train_dataset = tf.data.Dataset.from_tensor_slices(x_train)
train_dataset = train_dataset.map(lambda x: data_augmentation(x, training=True), num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)

# Define the base encoder network
def create_encoder():
    inputs = layers.Input(shape=(32, 32, 3))
    x = layers.Conv2D(32, 3, activation='relu')(inputs)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(64, 3, activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(128, 3, activation='relu')(x)
    x = layers.GlobalMaxPooling2D()(x)
    outputs = layers.Dense(128)(x)
    return keras.Model(inputs=inputs, outputs=outputs)

# Define the projection head
def create_projection_head():
    inputs = keras.Input(shape=(128,))
    x = layers.Dense(256, activation='relu')(inputs)
    outputs = layers.Dense(128)(x)
    return keras.Model(inputs=inputs, outputs=outputs)

encoder = create_encoder()
projection_head = create_projection_head()
```

```
    optimizer = keras.optimizers.Adam()
    epochs = 20
    temperature = 0.1

    # Custom training loop
    for epoch in range(epochs):
        print(f"\nStart of epoch {epoch}")
        for step, x_batch_train in enumerate(train_dataset):
            with tf.GradientTape() as tape:
                # Forward pass
                encoded = encoder(x_batch_train, training=True)
                projected = projection_head(encoded, training=True)
                normalized = tf.math.l2_normalize(projected, axis=1)
                cosine_similarity = tf.matmul(normalized, normalized, transpose_b=True)
                logits = cosine_similarity / temperature
                labels = tf.range(x_batch_train.shape[0])
                loss_value = keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)
                loss_value = tf.reduce_mean(loss_value)

            gradients = tape.gradient(loss_value, encoder.trainable_weights + projection_head.trainable_weights)
            optimizer.apply_gradients(zip(gradients, encoder.trainable_weights + projection_head.trainable_weights))

            if step % 100 == 0:
                print(f"Training loss (for one batch) at step {step}: {float(loss_value)}")

    # Extract representations after training
    representations = encoder.predict(x_test)

    # t-SNE visualization
    tsne = TSNE(n_components=2, perplexity=30, n_iter=3000)
    tsne_results = tsne.fit_transform(representations)

    plt.figure(figsize=(8, 8))
    plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=y_test.flatten(), cmap='rainbow')
    plt.title('t-SNE Visualization of Learned Representations')
    plt.colorbar()
    plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step

Start of epoch 0
Training loss (for one batch) at step 0: 5.225990295410156
Training loss (for one batch) at step 100: 0.035465534776449203

Start of epoch 1
Training loss (for one batch) at step 0: 0.027825651690363884
Training loss (for one batch) at step 100: 0.025812309235334396

Start of epoch 2
Training loss (for one batch) at step 0: 0.024271875619888306
Training loss (for one batch) at step 100: 0.023882072418928146

Start of epoch 3
Training loss (for one batch) at step 0: 0.02266448177397251
Training loss (for one batch) at step 100: 0.02274133823812008

Start of epoch 4
Training loss (for one batch) at step 0: 0.021400025114417076
Training loss (for one batch) at step 100: 0.021811731159687042

Start of epoch 5
Training loss (for one batch) at step 0: 0.020570112392306328
Training loss (for one batch) at step 100: 0.02096063271164894

Start of epoch 6
Training loss (for one batch) at step 0: 0.020268408581614494
Training loss (for one batch) at step 100: 0.02039061300456524

Start of epoch 7
Training loss (for one batch) at step 0: 0.019772401079535484
Training loss (for one batch) at step 100: 0.0198917668312788

Start of epoch 8
Training loss (for one batch) at step 0: 0.019373904913663864
Training loss (for one batch) at step 100: 0.019696513190865517

Start of epoch 9
Training loss (for one batch) at step 0: 0.019114818423986435
Training loss (for one batch) at step 100: 0.019542798399925232

Start of epoch 10
Training loss (for one batch) at step 0: 0.01894354447722435
Training loss (for one batch) at step 100: 0.019336562603712082

Start of epoch 11
Training loss (for one batch) at step 0: 0.018947482109069824
Training loss (for one batch) at step 100: 0.01901734247803688

Start of epoch 12
Training loss (for one batch) at step 0: 0.018921809270977974
Training loss (for one batch) at step 100: 0.01873650960624218

Start of epoch 13
Training loss (for one batch) at step 0: 0.018810607492923737
Training loss (for one batch) at step 100: 0.018818102777004242

Start of epoch 14
Training loss (for one batch) at step 0: 0.018570512533187866
Training loss (for one batch) at step 100: 0.01881370320916176

Start of epoch 15
```

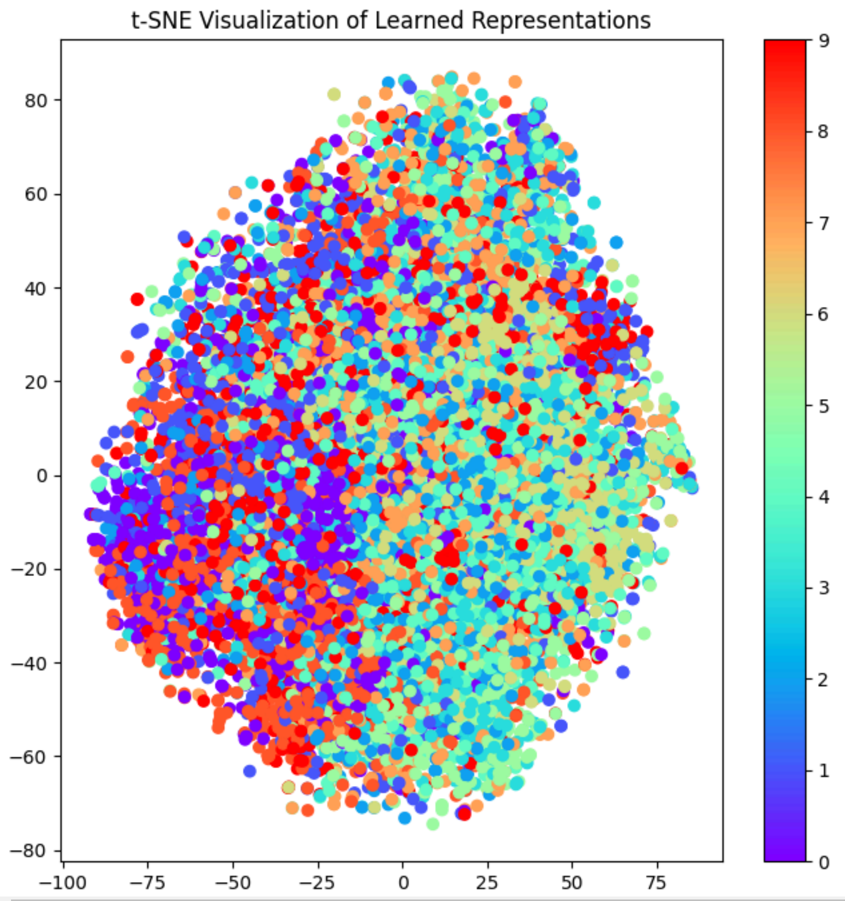
```
Training loss (for one batch) at step 0: 0.018478496000170708
Training loss (for one batch) at step 100: 0.01851455122232437

Start of epoch 16
Training loss (for one batch) at step 0: 0.018535461276769638
Training loss (for one batch) at step 100: 0.018314233049750328

Start of epoch 17
Training loss (for one batch) at step 0: 0.018039794638752937
Training loss (for one batch) at step 100: 0.018186140805482864

Start of epoch 18
Training loss (for one batch) at step 0: 0.017798926681280136
Training loss (for one batch) at step 100: 0.018238458782434464

Start of epoch 19
Training loss (for one batch) at step 0: 0.017827004194259644
Training loss (for one batch) at step 100: 0.017837101593613625
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```



The SimCLR approach aims to learn meaningful representations of images in a self-supervised manner. It does this by maximizing the cosine similarity between representations of augmented views of the same image (positive pairs) and minimizing the cosine similarity between representations of different images (negative pairs). The contrastive loss encourages the encoder to learn representations that are invariant to the applied data augmentations.

The visualization using t-SNE allows us to assess the quality of the learned representations by observing how well the different classes are separated in the low-dimensional space.

This code demonstrates the implementation of a self-supervised learning approach called SimCLR (Simple Framework for Contrastive Learning of Visual Representations) using the CIFAR-10 dataset.

1. Importing Libraries:
    - The necessary libraries, including TensorFlow, Keras, Matplotlib, and scikit-learn, are imported.
2. Loading and Preprocessing Data:
    - The CIFAR-10 dataset is loaded using `keras.datasets.cifar10.load_data()`.
    - The pixel values of the training and testing data are divided by 255.0 to scale them to the range [0, 1].
3. Data Augmentation:
    - A data augmentation pipeline is defined using `keras.Sequential()` to apply random cropping, flipping, and zooming to the input images.
    - The training data is converted into a TensorFlow Dataset using `tf.data.Dataset.from_tensor_slices()`.
    - The data augmentation pipeline is applied to each example in the dataset using `map()`.
    - The dataset is batched, cached, and prefetched for efficient training.
4. Defining the Encoder Network:
    - The `create_encoder` function defines the base encoder network using convolutional layers, max pooling, and a global max pooling layer.
    - The encoder network takes an input image and outputs a 128-dimensional feature representation.
5. Defining the Projection Head:
    - The `create_projection_head` function defines the projection head network using dense layers.
    - The projection head takes the 128-dimensional feature representation from the encoder and projects it to a 128-dimensional space.
6. Training Loop:
    - The training loop runs for a specified number of epochs.
    - For each batch of training data:
        - The encoder and projection head are applied to the batch of images.
        - The resulting projected representations are L2-normalized.
        - The cosine similarity matrix is computed between the normalized representations.
        - The contrastive loss is calculated using the cosine similarity matrix and temperature scaling.
        - The gradients of the loss with respect to the weights of the encoder and projection head are computed.
        - The optimizer is used to update the weights of the encoder and projection head.
7. Extracting Representations:
    - After training, the trained encoder is used to extract representations (embeddings) for the test images.