```
import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split


# Load CIFAR-100 dataset
(x_train, y_train), (x_test, y_test) = cifar100.load_data()
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
```

```
     Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
     169001437/169001437 [==============================] - 4s 0us/step
```

```
# Normalize pixel values to the range [0, 1]
x_train, x_val, x_test = x_train / 255.0, x_val / 255.0, x_test / 255.0


# One-hot encode labels
y_train = to_categorical(y_train, 100)
y_val = to_categorical(y_val, 100)
y_test = to_categorical(y_test, 100)


# Load pre-trained DenseNet121 model (without top layers)
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

```
     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_
     29084464/29084464 [==============================] - 0s 0us/step
```

```
# Add custom top layers for CIFAR-100
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(1024, activation='relu')(x)
predictions = Dense(100, activation='softmax')(x)


# Create the model
model = Model(inputs=base_model.input, outputs=predictions)


# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])


# Fine-tune the model on CIFAR-100 data
history = model.fit(x_train, y_train, batch_size=64, epochs=100, validation_data=(x_val, y_val))
```

```
Epoch 90/100
625/625 [==============================] - 50s 80ms/step - loss: 0.0431 - accuracy: 0.9872 - val_loss: 3.0287 - val_accuracy: 0.5
Epoch 91/100
625/625 [==============================] - 49s 78ms/step - loss: 0.0334 - accuracy: 0.9892 - val_loss: 3.0194 - val_accuracy: 0.5
Epoch 92/100
625/625 [==============================] - 50s 80ms/step - loss: 0.0361 - accuracy: 0.9883 - val_loss: 3.0235 - val_accuracy: 0.5
Epoch 93/100
625/625 [==============================] - 49s 79ms/step - loss: 0.0321 - accuracy: 0.9900 - val_loss: 3.1048 - val_accuracy: 0.5
Epoch 94/100
625/625 [==============================] - 50s 81ms/step - loss: 0.0389 - accuracy: 0.9870 - val_loss: 2.9980 - val_accuracy: 0.5
Epoch 95/100
625/625 [==============================] - 48s 76ms/step - loss: 0.0346 - accuracy: 0.9888 - val_loss: 3.0090 - val_accuracy: 0.5
Epoch 96/100
625/625 [==============================] - 49s 78ms/step - loss: 0.0301 - accuracy: 0.9904 - val_loss: 3.1393 - val_accuracy: 0.5
Epoch 97/100
625/625 [==============================] - 51s 81ms/step - loss: 0.0317 - accuracy: 0.9905 - val_loss: 3.1570 - val_accuracy: 0.5
Epoch 98/100
625/625 [==============================] - 49s 79ms/step - loss: 0.0398 - accuracy: 0.9880 - val_loss: 3.0660 - val_accuracy: 0.5
Epoch 99/100
625/625 [==============================] - 50s 81ms/step - loss: 0.0279 - accuracy: 0.9908 - val_loss: 3.0513 - val_accuracy: 0.5
Epoch 100/100
625/625 [==============================] - 51s 81ms/step - loss: 0.0325 - accuracy: 0.9894 - val_loss: 3.0692 - val_accuracy: 0.5
```

```
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Error Rate: {(1-test_accuracy) * 100:.2f}%')
```

```
313/313 [==============================] - 5s 13ms/step - loss: 3.1360 - accuracy: 0.5856
Error Rate: 41.44%
```

```
# You can also save the model for future use
model.save('densenet_cifar100.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
```