```
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Dropout,
LayerNormalization, MultiHeadAttention, Embedding, Concatenate
from tensorflow.keras.models import Model

class Transformer(Model):
    def __init__(self, vocab_size, max_sequence_length, d_model,
num_heads, num_layers, dropout_rate=0.1):
        super(Transformer, self).__init__()

        # Define embedding layer
        self.embedding_layer = Embedding(vocab_size, d_model)

        # Define positional encoding layer
        self.positional_encoding =
self.get_positional_encoding(max_sequence_length, d_model)

        # Define transformer layers
        self.transformer_layers = [self.create_transformer_layer(d_model,
num_heads, dropout_rate) for _ in range(num_layers)]

        # Define output layer
        self.output_layer = Dense(vocab_size)

    def get_positional_encoding(self, max_sequence_length, d_model):
        # Calculate positional encodings
        positional_encoding = []
        for pos in range(max_sequence_length):
            pos_encoding = [pos / tf.pow(tf.constant(10000,
dtype=tf.float32), 2 * (i // 2) / tf.cast(d_model, tf.float32)) for i in
range(d_model)]
            if pos % 2 == 0:
                positional_encoding.append(tf.math.sin(pos_encoding))
            else:
                positional_encoding.append(tf.math.cos(pos_encoding))
        positional_encoding = tf.stack(positional_encoding)
        return tf.expand_dims(positional_encoding, axis=0)

    def create_transformer_layer(self, d_model, num_heads, dropout_rate):
        # Create transformer layer
        return MultiHeadAttention(num_heads=num_heads, key_dim=d_model //
num_heads, dropout=dropout_rate)

    def call(self, inputs, training=False):
        # Define forward pass of the model
        input_sequence, target_sequence = inputs

        # Embed input sequence and add positional encoding
        embedded_input = self.embedding_layer(input_sequence) +
self.positional_encoding[:, :tf.shape(input_sequence)[1], :]

        # Apply transformer layers sequentially
        transformer_output = embedded_input
        for layer in self.transformer_layers:
```

```
            transformer_output = layer(query=transformer_output,
value=transformer_output, attention_mask=None, training=training)

        # Apply output layer
        output_logits = self.output_layer(transformer_output)

        return output_logits

# Example usage:
vocab_size = 10000  # Example vocabulary size
max_sequence_length = 50  # Example maximum sequence length
d_model = 128  # Example model dimensionality
num_heads = 4  # Example number of attention heads
num_layers = 2  # Example number of transformer layers
dropout_rate = 0.1  # Example dropout rate

# Instantiate the Transformer model
transformer_model = Transformer(vocab_size, max_sequence_length, d_model,
num_heads, num_layers, dropout_rate)

# Compile the model
transformer_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')

# Define and initialize tokenizer object (replace this with your actual
tokenizer)
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=vocab_size)

# Example usage with text data
input_text = ["This is an example sentence", "Another example sentence"]
target_text = ["Dies ist ein Beispiel Satz", "Ein weiterer Beispiel
Satz"]

# Tokenize input sequences
input_sequences = tokenizer.texts_to_sequences(input_text)

# Pad sequences to ensure equal length
input_sequences_padded =
tf.keras.preprocessing.sequence.pad_sequences(input_sequences,
maxlen=max_sequence_length, padding='post')

# Tokenize target sequences (if applicable)
target_sequences = tokenizer.texts_to_sequences(target_text)
target_sequences_padded =
tf.keras.preprocessing.sequence.pad_sequences(target_sequences,
maxlen=max_sequence_length, padding='post')

# Define model inputs
inputs = (input_sequences_padded, target_sequences_padded)

# Feed data into the model
output_logits = transformer_model(inputs)

# Extract predictions (if applicable)
```

```
predicted_sequences = tf.argmax(output_logits, axis=-1)
```