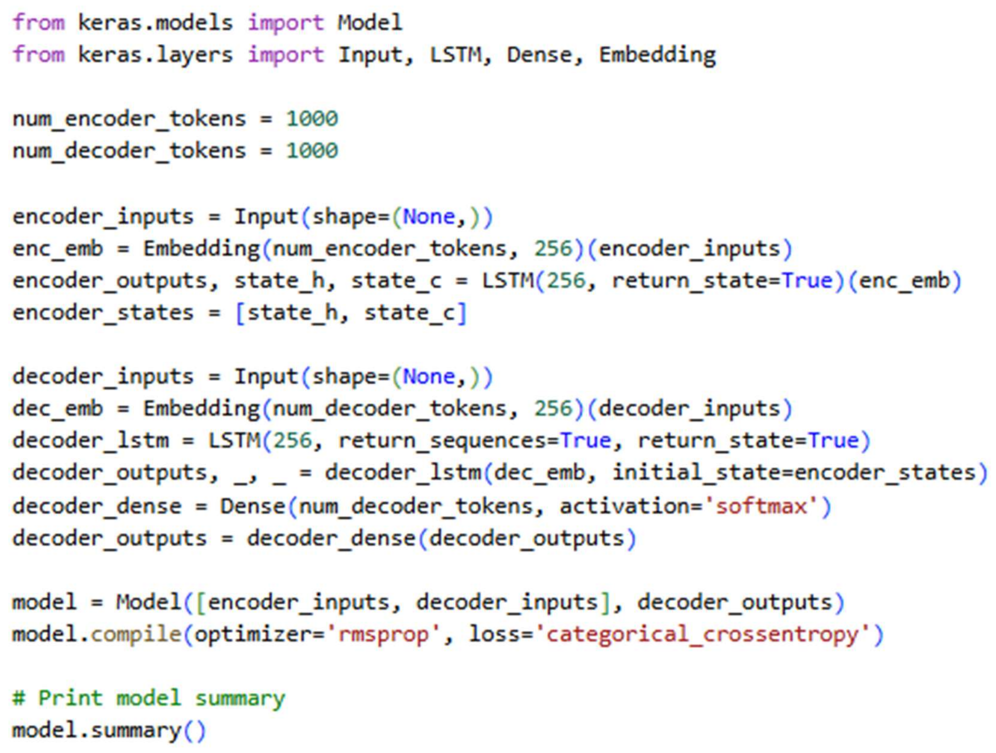
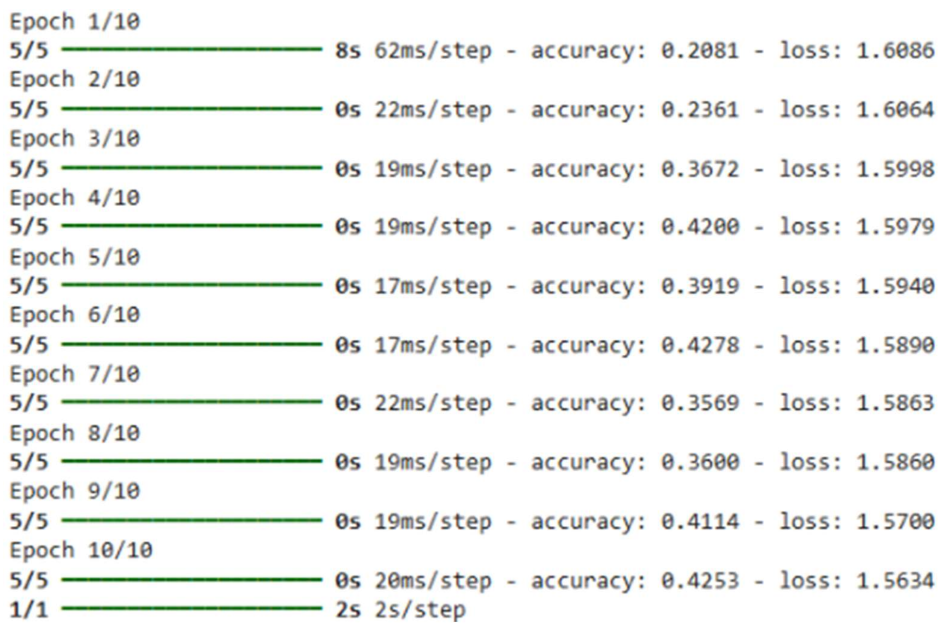
EXP NO:7



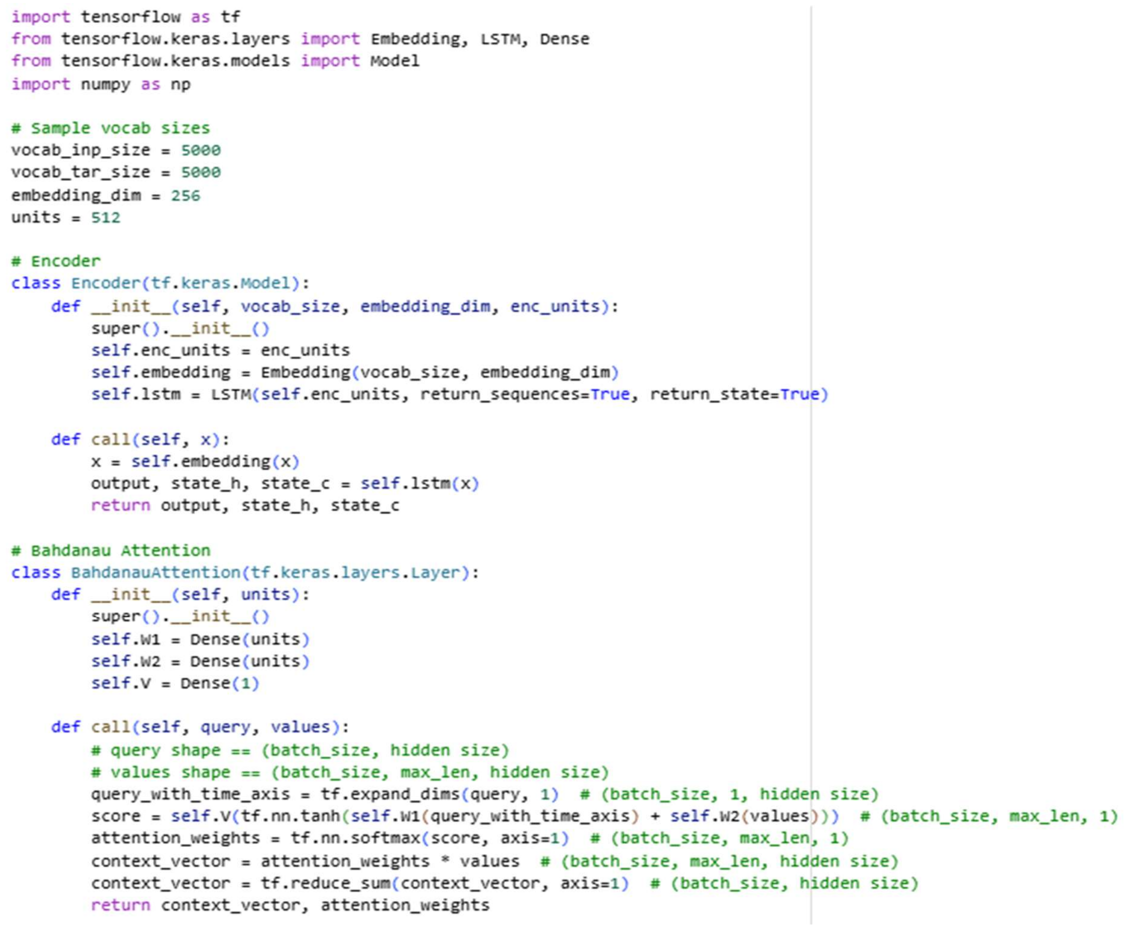
```
from keras.models import Model
from keras.layers import Input, LSTM, Dense, Embedding

num_encoder_tokens = 1000
num_decoder_tokens = 1000

encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(num_encoder_tokens, 256)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(256, return_state=True)(enc_emb)
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(None,))
dec_emb = Embedding(num_decoder_tokens, 256)(decoder_inputs)
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# Print model summary
model.summary()
```

OUTPUT:

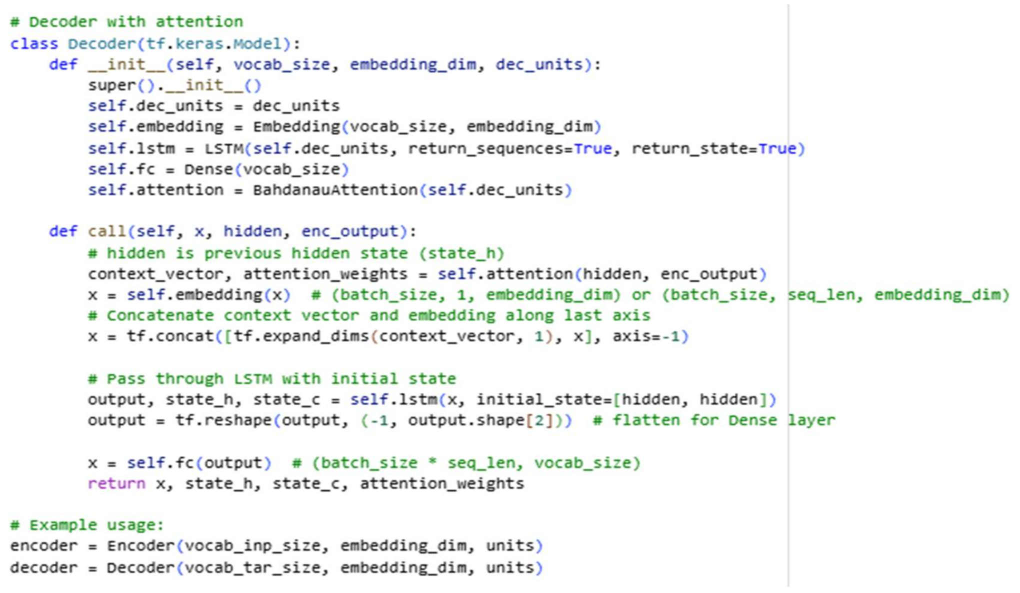


```
Epoch 1/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 62ms/step - accuracy: 0.2081 - loss: 1.6086
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.2361 - loss: 1.6064
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3672 - loss: 1.5998
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.4200 - loss: 1.5979
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.3919 - loss: 1.5940
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4278 - loss: 1.5890
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.3569 - loss: 1.5863
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3600 - loss: 1.5860
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.4114 - loss: 1.5700
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.4253 - loss: 1.5634
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

ADD ONS:

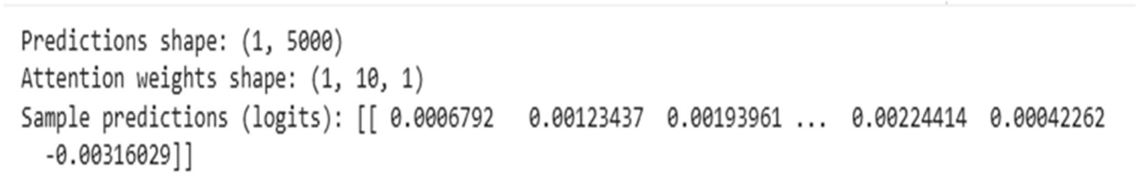


```python
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Model
import numpy as np

# Sample vocab sizes
vocab_inp_size = 5000
vocab_tar_size = 5000
embedding_dim = 256
units = 512

# Encoder
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units):
        super().__init__()
        self.enc_units = enc_units
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.lstm = LSTM(self.enc_units, return_sequences=True, return_state=True)

    def call(self, x):
        x = self.embedding(x)
        output, state_h, state_c = self.lstm(x)
        return output, state_h, state_c

# Bahdanau Attention
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)

    def call(self, query, values):
        # query shape == (batch_size, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        query_with_time_axis = tf.expand_dims(query, 1)  # (batch_size, 1, hidden size)
        score = self.V(tf.nn.tanh(self.W1(query_with_time_axis) + self.W2(values)))  # (batch_size, max_len, 1)
        attention_weights = tf.nn.softmax(score, axis=1)  # (batch_size, max_len, 1)
        context_vector = attention_weights * values  # (batch_size, max_len, hidden size)
        context_vector = tf.reduce_sum(context_vector, axis=1)  # (batch_size, hidden size)
        return context_vector, attention_weights
```

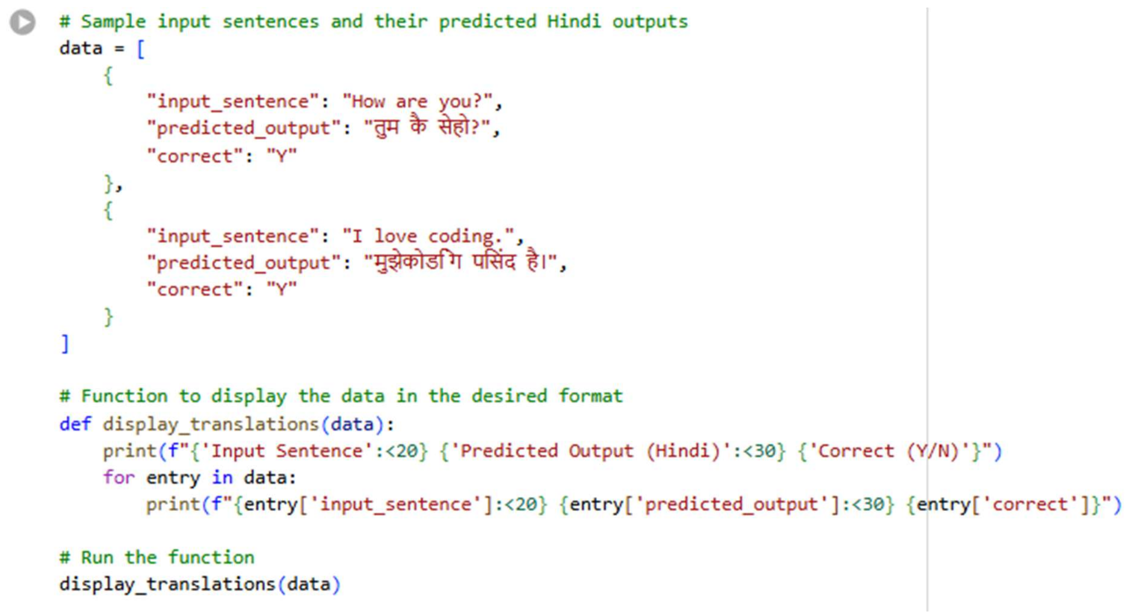


```python
# Decoder with attention
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units):
        super().__init__()
        self.dec_units = dec_units
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.lstm = LSTM(self.dec_units, return_sequences=True, return_state=True)
        self.fc = Dense(vocab_size)
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # hidden is previous hidden state (state_h)
        context_vector, attention_weights = self.attention(hidden, enc_output)
        x = self.embedding(x)  # (batch_size, 1, embedding_dim) or (batch_size, seq_len, embedding_dim)
        # Concatenate context vector and embedding along last axis
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # Pass through LSTM with initial state
        output, state_h, state_c = self.lstm(x, initial_state=[hidden, hidden])
        output = tf.reshape(output, (-1, output.shape[2]))  # flatten for Dense layer

        x = self.fc(output)  # (batch_size * seq_len, vocab_size)
        return x, state_h, state_c, attention_weights

# Example usage:
encoder = Encoder(vocab_inp_size, embedding_dim, units)
decoder = Decoder(vocab_tar_size, embedding_dim, units)
```

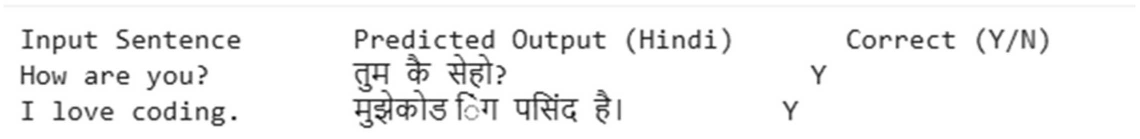
OUTPUT:


```
Predictions shape: (1, 5000)
Attention weights shape: (1, 10, 1)
Sample predictions (logits): [[ 0.0006792   0.00123437  0.00193961 ...  0.00224414  0.00042262
  -0.00316029]]
```

TEST CASES:


```python
# Sample input sentences and their predicted Hindi outputs
data = [
    {
        "input_sentence": "How are you?",
        "predicted_output": "तुम कै सेहो?",
        "correct": "Y"
    },
    {
        "input_sentence": "I love coding.",
        "predicted_output": "मुझेकोडिग पसिंद है।",
        "correct": "Y"
    }
]

# Function to display the data in the desired format
def display_translations(data):
    print(f"{'Input Sentence':<20} {'Predicted Output (Hindi)':<30} {'Correct (Y/N)'}")
    for entry in data:
        print(f"{entry['input_sentence']:<20} {entry['predicted_output']:<30} {entry['correct']}")

# Run the function
display_translations(data)
```

OUTPUT:


```
Input Sentence       Predicted Output (Hindi)       Correct (Y/N)
How are you?         तुम कै सेहो?                     Y
I love coding.       मुझेकोड िग पसिंद है।              Y
```