Exercise 1: Signal Trace Analysis

Trace Embedding

```python
class TraceEmbedding(layers.Layer):
    def __init__(self, embed_dim):
        super(TraceEmbedding, self).__init__()
        self.dense = layers.Dense(embed_dim,
                                  activation="tanh")

    def call(self, inp, training=None):
        inp = tf.expand_dims(inp, -1)
        return self.dense(inp)
```

TransformerBlock

```python
def call(self, input1, input2, training=None):
    attn_output = self.att(input1, input2)
    attn_output = self.dropout1(attn_output, training=training)
    out1 = self.layernorm1(input1 + attn_output)
    out1 = self.dropout1(out1, training=training)
    ffn_output = self.ffn(out1)
    return self.layernorm2(out1 + ffn_output)
```

Positional Encoding

```python
def get_angles(i, delta, d_model):
    return i / (10000. ** (2 * (delta // 2) / d_model))


def sin_cos_encoding(num_pos, d_model):
    angle_rads = get_angles(np.arange(num_pos)[:, np.newaxis],
                            np.arange(d_model)[np.newaxis, :],
                            d_model)
    # apply sin to even indices in the array; 2i
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
    # apply cos to odd indices in the array; 2i+1
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
    pos_encoding = angle_rads[np.newaxis, ...]
    return tf.cast(pos_encoding, dtype=tf.float32)


class PositionalEncoder(layers.Layer):
    def __init__(self, num_pos, d_model):
        super(PositionalEncoder, self).__init__()
        self.pos_enc = sin_cos_encoding(num_pos, d_model)

    def call(self, inp, training=None):
        encoded_input = layers.Add()([inp, self.pos_enc])
        return encoded_input
```