VADOT Dylan

Documentation Semestral project

Image search and understanding

I -Introduction

The purpose of the project is to have a better image understanding. It includes the capacity to determine important objects in an image, ignore background to predict items attributes from the image.

The approach adopted for this subject was to work with attention mechanism that allow to enhance some parts of the input data while diminishing other parts. The transformers were chosen over the Recurrent Neural Network because of the short memory issue of the CNN and the fact that transformers leave more room for parallelization.

II – Model

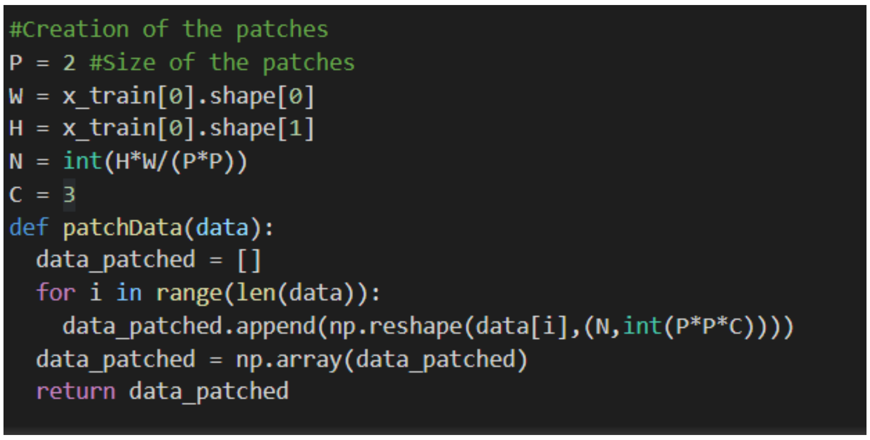
We work with the cifar10 database. Firstly, the objective was to work with image with higher resolution and with more classes in order to recognize more elements on images and have image with a more standard resolution, but because of computational and memory limits, we restrict the work to only 10 classes.

Input:

The input images have a resolution of 32*32 pixels.

We separate each image into 2*2 pixels patches.

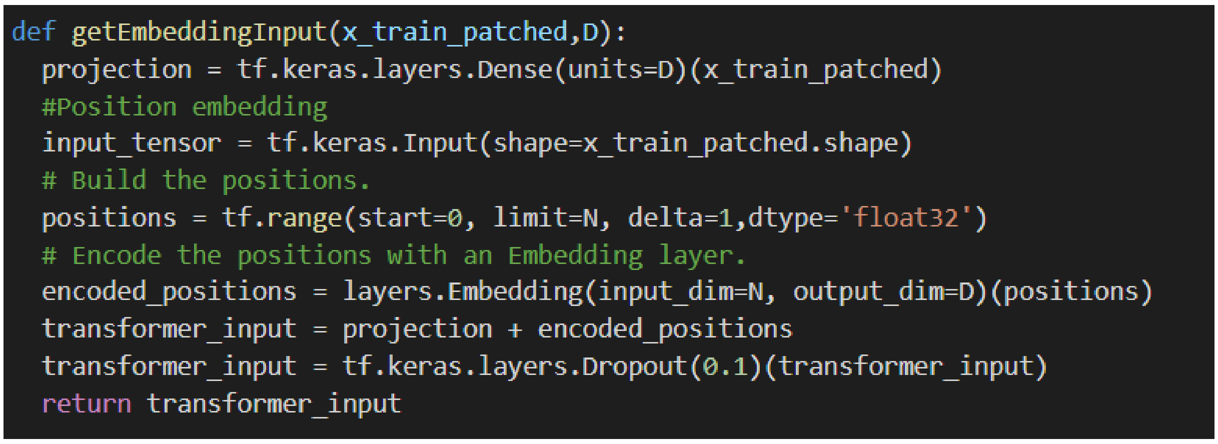
This operation is made by the function patchData.



```
#Creation of the patches
P = 2 #Size of the patches
W = x_train[0].shape[0]
H = x_train[0].shape[1]
N = int(H*W/(P*P))
C = 3
def patchData(data):
  data_patched = []
  for i in range(len(data)):
    data_patched.append(np.reshape(data[i],(N,int(P*P*C))))
  data_patched = np.array(data_patched)
  return data_patched
```

Then the patches were flattened and mapped to D dimensions with a trainable linear projection. After several tests, we chose D=128 (see annex for the tests). As self-attention is invariant, a learnable position embedding was added to retain positional information.

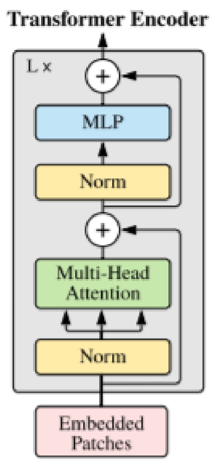
This operation is realized by the function getEmbeddingInput



```
def getEmbeddingInput(x_train_patched,D):
  projection = tf.keras.layers.Dense(units=D)(x_train_patched)
  #Position embedding
  input_tensor = tf.keras.Input(shape=x_train_patched.shape)
  # Build the positions.
  positions = tf.range(start=0, limit=N, delta=1,dtype='float32')
  # Encode the positions with an Embedding layer.
  encoded_positions = layers.Embedding(input_dim=N, output_dim=D)(positions)
  transformer_input = projection + encoded_positions
  transformer_input = tf.keras.layers.Dropout(0.1)(transformer_input)
  return transformer_input
```

The sequence of patches obtain for an image is the input of the transformer encoder.

<u>The transformer encoder</u>:



Originally a 8-head-Multi Head Attention layer was supposed to be used (that is the most common value for transformers), but because of the memory limitations of the device, only 4 heads were used.

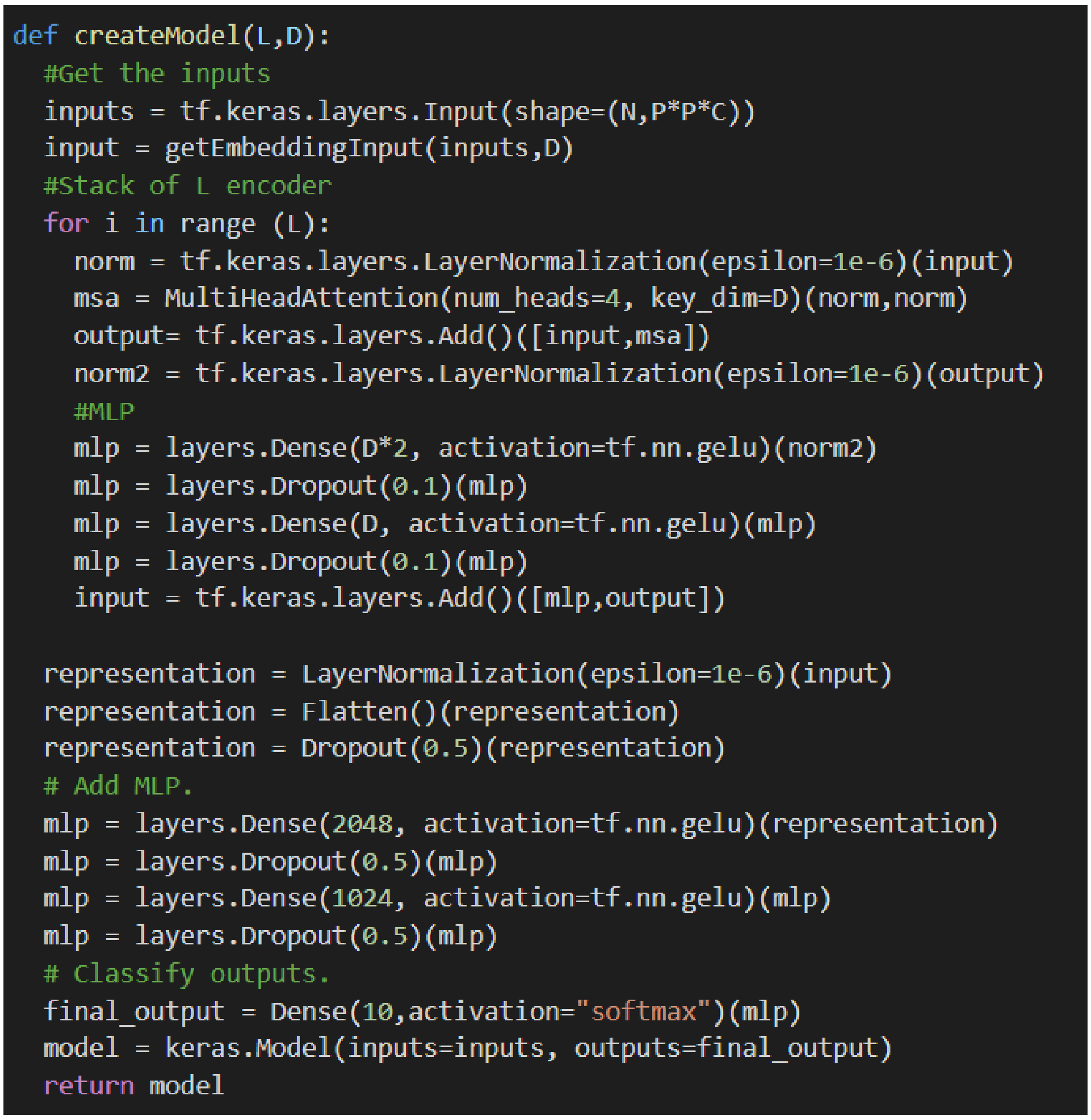
After several tests, a L=8 stack of transformer encoder was used. The queries, keys, and values are all from the outputs of the previous encoder layer.

A residual connection is employed by using the Add Layer and then normalize for scaling issue.

The MLP contains two layers with a GELU non-linearity of size 2048 and 1024. Drop Out layers where added to prevent from overfitting.

Classification:

At the output of the transformer a Multi-Layer Perceptron was used for the classification with a size of 2048 then 1024 then 10.



```
def createModel(L,D):
  #Get the inputs
  inputs = tf.keras.layers.Input(shape=(N,P*P*C))
  input = getEmbeddingInput(inputs,D)
  #Stack of L encoder
  for i in range (L):
    norm = tf.keras.layers.LayerNormalization(epsilon=1e-6)(input)
    msa = MultiHeadAttention(num_heads=4, key_dim=D)(norm,norm)
    output= tf.keras.layers.Add()([input,msa])
    norm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(output)
    #MLP
    mlp = layers.Dense(D*2, activation=tf.nn.gelu)(norm2)
    mlp = layers.Dropout(0.1)(mlp)
    mlp = layers.Dense(D, activation=tf.nn.gelu)(mlp)
    mlp = layers.Dropout(0.1)(mlp)
    input = tf.keras.layers.Add()([mlp,output])

  representation = LayerNormalization(epsilon=1e-6)(input)
  representation = Flatten()(representation)
  representation = Dropout(0.5)(representation)
  # Add MLP.
  mlp = layers.Dense(2048, activation=tf.nn.gelu)(representation)
  mlp = layers.Dropout(0.5)(mlp)
  mlp = layers.Dense(1024, activation=tf.nn.gelu)(mlp)
  mlp = layers.Dropout(0.5)(mlp)
  # Classify outputs.
  final_output = Dense(10,activation="softmax")(mlp)
  model = keras.Model(inputs=inputs, outputs=final_output)
  return model
```

A model summary can be found in annex.

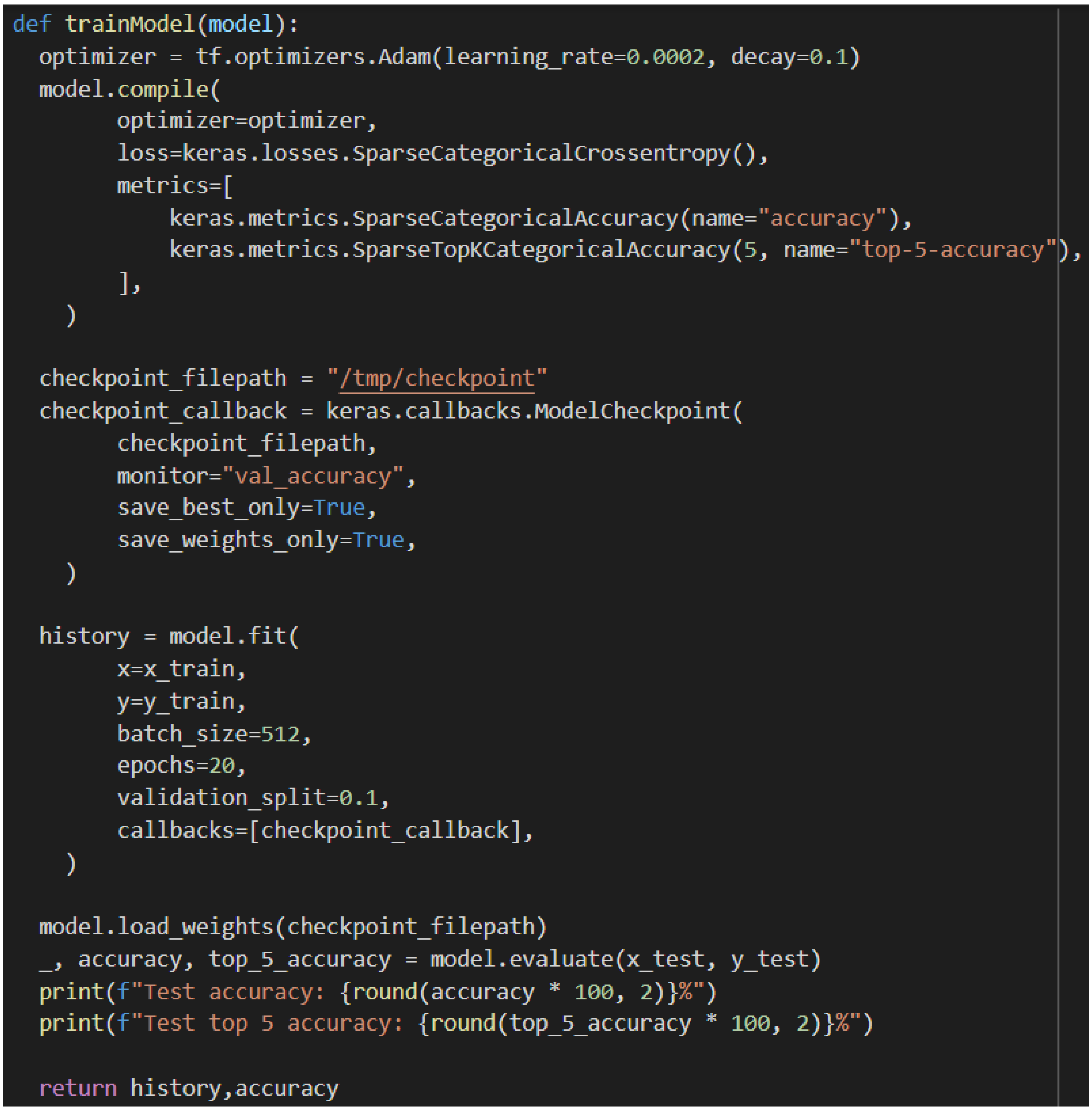
III – Training

Due to memory limitation, the training has been done only with 5000 images and the validation with 1000 images.

For the training part the Adam optimizer has been used with a learning rate of 0.0002 and a decay of 0.1.

The Sparse Categorical Crossentropy has been chosen because it is adapted when there are 2 or more labels.

After several tests of batch size and epochs, the best result was obtained for a batch size of 512 and 20 epochs in the limit of the available memory.



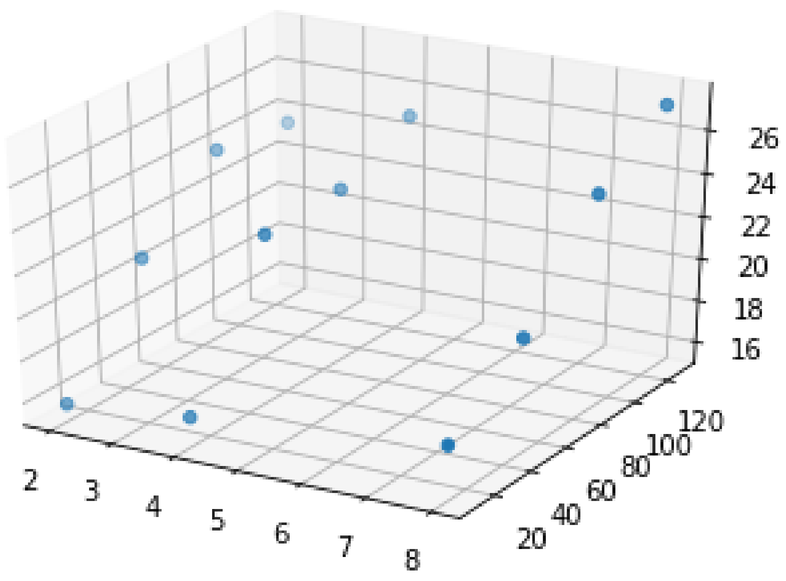
```python
def trainModel(model):
  optimizer = tf.optimizers.Adam(learning_rate=0.0002, decay=0.1)
  model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(),
        metrics=[
            keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
            keras.metrics.SparseTopKCategoricalAccuracy(5, name="top-5-accuracy"),
        ],
    )

  checkpoint_filepath = "/tmp/checkpoint"
  checkpoint_callback = keras.callbacks.ModelCheckpoint(
        checkpoint_filepath,
        monitor="val_accuracy",
        save_best_only=True,
        save_weights_only=True,
    )

  history = model.fit(
        x=x_train,
        y=y_train,
        batch_size=512,
        epochs=20,
        validation_split=0.1,
        callbacks=[checkpoint_callback],
    )

  model.load_weights(checkpoint_filepath)
  _, accuracy, top_5_accuracy = model.evaluate(x_test, y_test)
  print(f"Test accuracy: {round(accuracy * 100, 2)}%")
  print(f"Test top 5 accuracy: {round(top_5_accuracy * 100, 2)}%")

  return history,accuracy
```

Results:

The best results get obtained for L=8 and L=128 where we got an accuracy of 27.3%. Even if it is 3 times better than random classification, this result is very low. As seen by the the tests, more L and D are high, more the accuracy is high. Unfortunately, because of the memory limitations it was impossible to train the model for higher values. Moreover, the training has been made only on 5000 images and the validation on 1000 images. We should have used a lot more images but the computational time were to high and sometimes the memory not big enough.

Annex:

Test:



Accuracy for different values of L (2-8) and D(10-128)

Model summary:

```
Model: "model"
__________________________________________________________________________________________________
____
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
====
 input_1 (InputLayer)           [(None, 256, 12)]     0           []

 dense (Dense)                  (None, 256, 128)      1664        ['input_1[0][0]']

 tf.__operators__.add (TFOpLamb  (None, 256, 128)     0           ['dense[0][0]']
 da)

 dropout (Dropout)              (None, 256, 128)      0
['tf.__operators__.add[0][0]']

 layer_normalization (LayerNorm  (None, 256, 128)     256         ['dropout[0][0]']
 alization)

 multi_head_attention (MultiHea  (None, 256, 128)     263808      ['layer_normalization[0][0]',
 dAttention)                                                       'layer_normalization[0][0]']

 add (Add)                      (None, 256, 128)      0           ['dropout[0][0]',

'multi_head_attention[0][0]']

 layer_normalization_1 (LayerNo  (None, 256, 128)     256         ['add[0][0]']
 rmalization)

 dense_1 (Dense)                (None, 256, 256)      33024
['layer_normalization_1[0][0]']

 dropout_1 (Dropout)            (None, 256, 256)      0           ['dense_1[0][0]']

 dense_2 (Dense)                (None, 256, 128)      32896       ['dropout_1[0][0]']
 dropout_2 (Dropout)            (None, 256, 128)     0           ['dense_2[0][0]']

 add_1 (Add)                    (None, 256, 128)     0           ['dropout_2[0][0]',
                                                                  'add[0][0]']

 layer_normalization_2 (LayerNo  (None, 256, 128)    256         ['add_1[0][0]']
 rmalization)

 multi_head_attention_1 (MultiH  (None, 256, 128)    263808
['layer_normalization_2[0][0]',
 eadAttention)
'layer_normalization_2[0][0]']

 add_2 (Add)                    (None, 256, 128)     0           ['add_1[0][0]',

'multi_head_attention_1[0][0]']

 layer_normalization_3 (LayerNo  (None, 256, 128)    256         ['add_2[0][0]']
 rmalization)

 dense_3 (Dense)                (None, 256, 256)     33024
['layer_normalization_3[0][0]']

 dropout_3 (Dropout)            (None, 256, 256)     0           ['dense_3[0][0]']

 dense_4 (Dense)                (None, 256, 128)     32896       ['dropout_3[0][0]']

 dropout_4 (Dropout)            (None, 256, 128)     0           ['dense_4[0][0]']

 add_3 (Add)                    (None, 256, 128)     0           ['dropout_4[0][0]',
                                                                  'add_2[0][0]']

 layer_normalization_4 (LayerNo  (None, 256, 128)    256         ['add_3[0][0]']
 rmalization)

 multi_head_attention_2 (MultiH  (None, 256, 128)    263808
['layer_normalization_4[0][0]',
 eadAttention)
'layer_normalization_4[0][0]']

 add_4 (Add)                    (None, 256, 128)     0           ['add_3[0][0]',

'multi_head_attention_2[0][0]']

 layer_normalization_5 (LayerNo  (None, 256, 128)    256         ['add_4[0][0]']
 rmalization)

 dense_5 (Dense)                (None, 256, 256)     33024
['layer_normalization_5[0][0]']

 dropout_5 (Dropout)            (None, 256, 256)     0           ['dense_5[0][0]']

 dense_6 (Dense)                (None, 256, 128)     32896       ['dropout_5[0][0]']

 dropout_6 (Dropout)            (None, 256, 128)     0           ['dense_6[0][0]']

 add_5 (Add)                    (None, 256, 128)     0           ['dropout_6[0][0]',
                                                                  'add_4[0][0]']

 layer_normalization_6 (LayerNo  (None, 256, 128)    256         ['add_5[0][0]']
 rmalization)

 multi_head_attention_3 (MultiH  (None, 256, 128)    263808
['layer_normalization_6[0][0]',
 eadAttention)
'layer_normalization_6[0][0]']

 add_6 (Add)                    (None, 256, 128)     0           ['add_5[0][0]',

'multi_head_attention_3[0][0]']

 layer_normalization_7 (LayerNo  (None, 256, 128)    256         ['add_6[0][0]']
 rmalization)

 dense_7 (Dense)                (None, 256, 256)     33024
['layer_normalization_7[0][0]']
```

```
 dropout_7 (Dropout)             (None, 256, 256)     0           ['dense_7[0][0]']

 dense_8 (Dense)                 (None, 256, 128)     32896       ['dropout_7[0][0]']

 dropout_8 (Dropout)             (None, 256, 128)     0           ['dense_8[0][0]']

 add_7 (Add)                     (None, 256, 128)     0           ['dropout_8[0][0]',
                                                                   'add_6[0][0]']

 layer_normalization_8 (LayerNo  (None, 256, 128)     256         ['add_7[0][0]']
 rmalization)

 multi_head_attention_4 (MultiH  (None, 256, 128)     263808
['layer_normalization_8[0][0]',
 eadAttention)
'layer_normalization_8[0][0]']

 add_8 (Add)                     (None, 256, 128)     0           ['add_7[0][0]',

'multi_head_attention_4[0][0]']

 layer_normalization_9 (LayerNo  (None, 256, 128)     256         ['add_8[0][0]']
 rmalization)

 dense_9 (Dense)                 (None, 256, 256)     33024
['layer_normalization_9[0][0]']

 dropout_9 (Dropout)             (None, 256, 256)     0           ['dense_9[0][0]']

 dense_10 (Dense)                (None, 256, 128)     32896       ['dropout_9[0][0]']

 dropout_10 (Dropout)            (None, 256, 128)     0           ['dense_10[0][0]']

 add_9 (Add)                     (None, 256, 128)     0           ['dropout_10[0][0]',
                                                                   'add_8[0][0]']

 layer_normalization_10 (LayerN  (None, 256, 128)     256         ['add_9[0][0]']
 ormalization)

 multi_head_attention_5 (MultiH  (None, 256, 128)     263808
['layer_normalization_10[0][0]',
 eadAttention)
'layer_normalization_10[0][0]']

 add_10 (Add)                    (None, 256, 128)     0           ['add_9[0][0]',

'multi_head_attention_5[0][0]']

 layer_normalization_11 (LayerN  (None, 256, 128)     256         ['add_10[0][0]']
 ormalization)

 dense_11 (Dense)                (None, 256, 256)     33024
['layer_normalization_11[0][0]']

 dropout_11 (Dropout)            (None, 256, 256)     0           ['dense_11[0][0]']

 dense_12 (Dense)                (None, 256, 128)     32896       ['dropout_11[0][0]']

 dropout_12 (Dropout)            (None, 256, 128)     0           ['dense_12[0][0]']

 add_11 (Add)                    (None, 256, 128)     0           ['dropout_12[0][0]',
                                                                   'add_10[0][0]']

 layer_normalization_12 (LayerN  (None, 256, 128)     256         ['add_11[0][0]']
 ormalization)

 multi_head_attention_6 (MultiH  (None, 256, 128)     263808
['layer_normalization_12[0][0]',
 eadAttention)
'layer_normalization_12[0][0]']

 add_12 (Add)                    (None, 256, 128)     0           ['add_11[0][0]',

'multi_head_attention_6[0][0]']

 layer_normalization_13 (LayerN  (None, 256, 128)     256         ['add_12[0][0]']
 ormalization)
 dense_13 (Dense)               (None, 256, 256)     33024
['layer_normalization_13[0][0]']

 dropout_13 (Dropout)           (None, 256, 256)     0            ['dense_13[0][0]']

 dense_14 (Dense)               (None, 256, 128)     32896        ['dropout_13[0][0]']

 dropout_14 (Dropout)           (None, 256, 128)     0            ['dense_14[0][0]']

 add_13 (Add)                   (None, 256, 128)     0            ['dropout_14[0][0]',
                                                                   'add_12[0][0]']

 layer_normalization_14 (LayerN  (None, 256, 128)    256          ['add_13[0][0]']
 ormalization)

 multi_head_attention_7 (MultiH  (None, 256, 128)    263808
['layer_normalization_14[0][0]',
 eadAttention)
'layer_normalization_14[0][0]']

 add_14 (Add)                   (None, 256, 128)     0            ['add_13[0][0]',

'multi_head_attention_7[0][0]']

 layer_normalization_15 (LayerN  (None, 256, 128)    256          ['add_14[0][0]']
 ormalization)

 dense_15 (Dense)               (None, 256, 256)     33024
['layer_normalization_15[0][0]']

 dropout_15 (Dropout)           (None, 256, 256)     0            ['dense_15[0][0]']

 dense_16 (Dense)               (None, 256, 128)     32896        ['dropout_15[0][0]']

 dropout_16 (Dropout)           (None, 256, 128)     0            ['dense_16[0][0]']

 add_15 (Add)                   (None, 256, 128)     0            ['dropout_16[0][0]',
                                                                   'add_14[0][0]']

 layer_normalization_16 (LayerN  (None, 256, 128)    256          ['add_15[0][0]']
 ormalization)

 flatten (Flatten)              (None, 32768)        0
['layer_normalization_16[0][0]']

 dropout_17 (Dropout)           (None, 32768)        0            ['flatten[0][0]']

 dense_17 (Dense)               (None, 2048)         67110912     ['dropout_17[0][0]']

 dropout_18 (Dropout)           (None, 2048)         0            ['dense_17[0][0]']

 dense_18 (Dense)               (None, 1024)         2098176      ['dropout_18[0][0]']

 dropout_19 (Dropout)           (None, 1024)         0            ['dense_18[0][0]']

 dense_19 (Dense)               (None, 10)           10250        ['dropout_19[0][0]']

==============================================================================================
====
Total params: 71,863,178
Trainable params: 71,863,178
Non-trainable params: 0
```