```
import keras
print("Keras version:", keras.__version__)
```

```
Keras version: 3.5.0
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.17.1
```

```
!pip install -q 'tensorflow-text==2.17.*'
!pip install -q tensorflow_datasets
!pip install -q einops
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.2/5.2 MB 26.8 MB/s eta 0:00:00
```

```
import numpy as np
import matplotlib.pyplot as plt
import os
import pathlib
import collections
import time
import string
import re
import einops
from tqdm.auto import tqdm
from PIL import Image
import nltk
from nltk.util import ngrams

import tensorflow as tf
import tensorflow_text as tf_text
import tensorflow_datasets as tfds


os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.get_logger().setLevel('ERROR')

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
def get_data(path = 'flickr8k'):
    path = pathlib.Path('flickr8k')

    tf.keras.utils.get_file(origin = 'https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip',
                            cache_dir = '.',
                            cache_subdir = path,
                            extract = True)

    tf.keras.utils.get_file(origin = 'https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip',
                            cache_dir = '.',
                            cache_subdir = path,
                            extract = True)


def get_dataset(path = 'flickr8k'):
    path = pathlib.Path('flickr8k')
    captions = (path/'Flickr8k.token.txt').read_text().splitlines()
    captions = [cap.split('\t') for cap in captions]
    captions = [(img_path.split('#')[0], cap) for (img_path, cap) in captions]
    cap_dict = collections.defaultdict(list)
    for img_path, cap in captions:
        cap_dict[img_path].append(cap)

    train_imgs_path = (path/'Flickr_8k.trainImages.txt').read_text().splitlines()
    test_imgs_path = (path/'Flickr_8k.testImages.txt').read_text().splitlines()

    train_caps = [(str(path/'Flicker8k_Dataset'/img_path), cap_dict[img_path]) for img_path in train_imgs_path]
    test_caps = [(str(path/'Flicker8k_Dataset'/img_path), cap_dict[img_path]) for img_path in test_imgs_path]

    train_raw = tf.data.experimental.from_list(train_caps)
```

```
    test_raw = tf.data.experimental.from_list(test_caps)

    return train_raw, test_raw


get_data()
train_raw, test_raw = get_dataset()
```

```
Downloading data from https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip
1115419746/1115419746 ━━━━━━━━━━━━━━━━━━━━ 14s 0us/step
Downloading data from https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip
2340801/2340801 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
print(len(train_raw), len(test_raw))
print(train_raw.element_spec)

for img_path, captions in train_raw.take(1):
    break

print(img_path)
print(captions)
```

```
6000 1000
(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(5,), dtype=tf.string, name=None))
tf.Tensor(b'flickr8k/Flicker8k_Dataset/2513260012_03d33305cf.jpg', shape=(), dtype=string)
tf.Tensor(
[b'A black dog is running after a white dog in the snow .'
 b'Black dog chasing brown dog through snow'
 b'Two dogs chase each other across the snowy ground .'
 b'Two dogs play together in the snow .'
 b'Two dogs running through a low lying body of water .'], shape=(5,), dtype=string)
```

```
image_shape = (224, 224, 3)
feature_extractor = tf.keras.applications.MobileNetV3Small(input_shape = image_shape, include_preprocessing = True, include_top = False)
feature_extractor.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v3/weights_mobilenet_v3_small_224_1.0_float
4334752/4334752 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
def load_img(img_path):
    img = tf.io.read_file(img_path)
    img = tf.io.decode_jpeg(img, channels = 3)
    img = tf.image.resize(img, image_shape[:-1])
    return img


img = load_img(img_path.numpy().decode('utf-8'))
print(img.shape)
print(feature_extractor(img[tf.newaxis, ...]).shape)
```

```
(224, 224, 3)
(1, 7, 7, 576)
```

```
def standardize(text):
    text = tf.strings.lower(text)
    text = tf.strings.regex_replace(text, f'[{re.escape(string.punctuation)}]', '')
    text = tf.strings.join(['[START]',text,'[END]'], separator = ' ')
    return text


standardize('A black dog is running after a white dog in the snow .')
```

```
<tf.Tensor: shape=(), dtype=string, numpy=b'[START] a black dog is running after a white dog in the snow  [END]'>
```

```
vocab_size = 5000
vectorizer = tf.keras.layers.TextVectorization(max_tokens = vocab_size,
                                               standardize = standardize,
                                               ragged = True)

vectorizer.adapt(train_raw.map(lambda img_path, cap: cap).unbatch().batch(1024))


print(vectorizer.get_vocabulary()[:10])
```

```
['', '[UNK]', 'a', '[START]', '[END]', 'in', 'the', 'on', 'is', 'and']
```

```
text_to_id_vectorizer = tf.keras.layers.StringLookup(vocabulary = vectorizer.get_vocabulary(), mask_token = '')
id_to_text_vectorizer = tf.keras.layers.StringLookup(vocabulary = vectorizer.get_vocabulary(), mask_token = '', invert = True)


def id_to_text(token_ids, reserved_tokens = ['', '[UNK]', '[START]', '[END]']):
    words = id_to_text_vectorizer(token_ids)
    bad_tokens = [re.escape(tok) for tok in reserved_tokens if tok != '[UNK]']
    bad_tokens_re = '|'.join(bad_tokens)
    bad_mask = tf.strings.regex_full_match(words, bad_tokens_re)
    words = tf.ragged.boolean_mask(words, ~bad_mask)

    return tf.strings.reduce_join(words, axis = -1, separator = ' ')

def serialize_data(images, captions):
    captions_shape = einops.parse_shape(captions, 'b c')
    captions = einops.rearrange(captions, 'b c -> (b c)')
    images = einops.repeat(images, 'b ...  -> (b c) ...', c = captions_shape['c'])
    return images, captions


for img, cap in train_raw.batch(1024).take(1):
    break

print(img.shape)
print(cap.shape)
img_serialize, cap_serialize = serialize_data(img, cap)
print(img_serialize.shape, cap_serialize.shape)
```

```
(1024,)
(1024, 5)
(5120,) (5120,)
```

```
def prepare_batch(img, cap):
    cap_tokenized = vectorizer(cap)
    cap_tokenized_in = cap_tokenized[:, :-1]
    cap_tokenized_out = cap_tokenized[:, 1:]

    return (img, cap_tokenized_in.to_tensor()), cap_tokenized_out.to_tensor()


def save_dataset(raw_ds, file_path, image_feature_extractor, vectorizer, shards = 20, batch_size = 64):
    raw_ds = (raw_ds
              .map(lambda img_path, cap: (load_img(img_path), cap), tf.data.AUTOTUNE)
              .batch(batch_size))

    def gen():
        for (img, cap) in tqdm(raw_ds):
            img_features = image_feature_extractor(img)
            img_features, cap = serialize_data(img_features, cap)

            yield img_features, cap

    ds = tf.data.Dataset.from_generator(gen,
                                        output_signature = (
                                            tf.TensorSpec(shape = image_feature_extractor.output_shape),
                                            tf.TensorSpec(shape = (None,), dtype = tf.string)
                                        ))

    ds = (ds
          .map(prepare_batch, tf.data.AUTOTUNE)
          .unbatch()
          .shuffle(1000))

    def shard_func(i, data):
        return i % shards

    ds.enumerate().save(file_path, shard_func = shard_func)


%%time
save_dataset(train_raw, 'train_cache', feature_extractor, vectorizer)
save_dataset(test_raw, 'test_cache', feature_extractor, vectorizer)
```

```
100%                                                  94/94 [02:23<00:00,  1.18s/it]

100%                                                  16/16 [00:24<00:00,  1.25s/it]

CPU times: user 2min 40s, sys: 24.9 s, total: 3min 5s
```

```
def load_dataset(file_path, batch_size = 64, cycle_length = 2):
    def reader_func(ds):
        ds = ds.shuffle(1000)
        return ds.interleave(lambda x: x, cycle_length = cycle_length)

    def drop_index(i, x):
        return x

    ds = tf.data.Dataset.load(file_path, reader_func = reader_func)

    ds = (ds
          .map(drop_index)
          .shuffle(1000)
          .padded_batch(batch_size)
          .prefetch(tf.data.AUTOTUNE))
    return ds


batch_size = 64
train_ds = load_dataset('train_cache', batch_size)
test_ds = load_dataset('test_cache', batch_size)


for (img, cap), cap_labels in train_ds.take(1):
    break

print(img.shape)
print(cap.shape)
print(cap_labels.shape)

print(cap[0])
print(cap_labels[0])
```

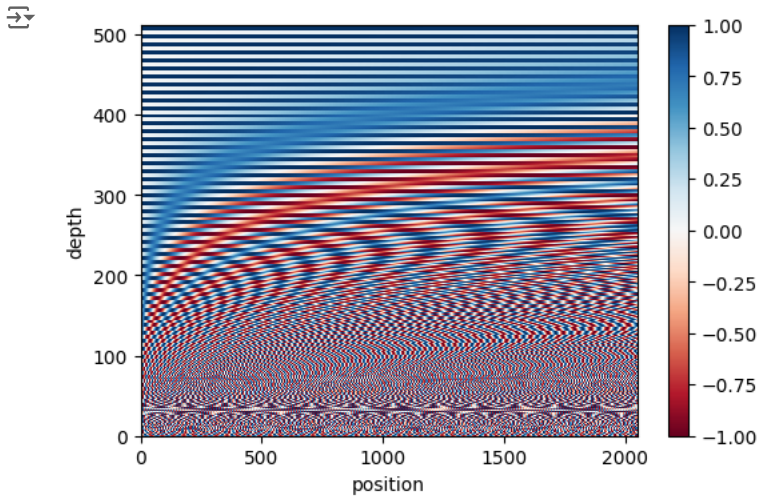
```
(64, 7, 7, 576)
(64, 37)
(64, 37)
tf.Tensor(
[   3    2   12   11    2  587    9  217    8   40    5   50   13 3570
 1035    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0], shape=(37,), dtype=int64)
tf.Tensor(
[   2   12   11    2  587    9  217    8   40    5   50   13 3570 1035
    4    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0], shape=(37,), dtype=int64)
```

```
train_ds.element_spec
```

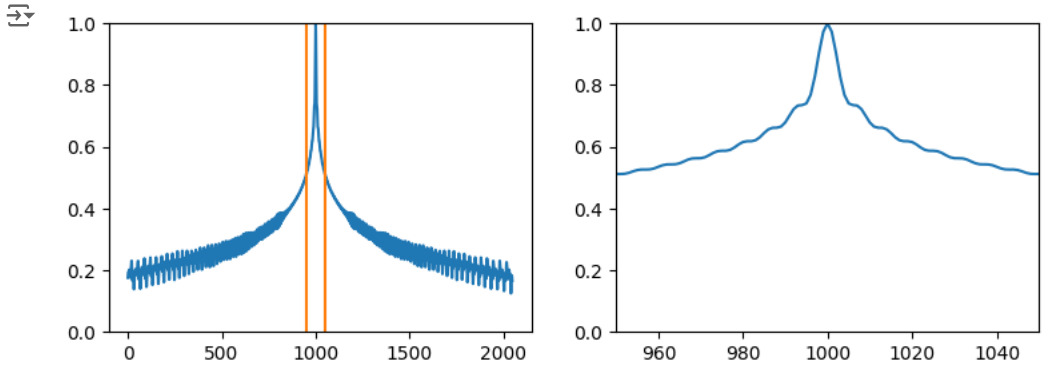
```
((TensorSpec(shape=(None, 7, 7, 576), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, None), dtype=tf.int64, name=None)),
 TensorSpec(shape=(None, None), dtype=tf.int64, name=None))
```

```
def positional_encoding(length, depth):
    pos = tf.cast(tf.range(length)[:, tf.newaxis], tf.float32)
    dep = tf.cast(tf.range(depth)[tf.newaxis, :], tf.float32)
    dep = ((dep // 2)*2)/tf.cast(depth, tf.float32)

    angle_rates = 1 / (10000**dep)
    angle_rads = pos*angle_rates

    out = tf.Variable(tf.zeros((length, depth)))
    out[:, 0::2].assign(tf.math.sin(angle_rads[:, 0::2]))
    out[:, 1::2].assign(tf.math.cos(angle_rads[:, 1::2]))

    return out[tf.newaxis, ...]


sample_enc = positional_encoding(length = 2048, depth = 512)[0]


plt.figure(figsize = (6, 4))
plt.pcolormesh(tf.transpose(sample_enc), cmap = 'RdBu')
```

```
plt.xlabel('position')
plt.ylabel('depth')
plt.colorbar()
plt.show();
```




```
p_norm = tf.linalg.l2_normalize(sample_enc[1000][tf.newaxis, :])
sample_enc_norm = tf.linalg.l2_normalize(sample_enc, axis = 1)
dots = tf.linalg.matmul(sample_enc_norm, p_norm, transpose_b = True)

plt.figure(figsize = (9, 3))
plt.subplot(1, 2, 1)
plt.plot(dots)
plt.ylim([0, 1])
plt.plot([950, 950, float('nan'), 1050, 1050], [0, 1, float('nan'), 0, 1])

plt.subplot(1, 2, 2)
plt.plot(dots)
plt.ylim([0, 1])
plt.xlim([950, 1050])
plt.show()
```




```
class PositionalEmbedding(tf.keras.layers.Layer):
    def __init__(self, vocab_size, d_model):
        super().__init__()
        self.d_model = d_model
        self.embedding = tf.keras.layers.Embedding(vocab_size, d_model, mask_zero = True)
        self.pos_enc = positional_encoding(length = 2048, depth = d_model)

    def compute_mask(self, *args, **kwargs):
        return self.embedding.compute_mask(*args, **kwargs)

    def call(self, x):
        length = tf.shape(x)[1]
        x_emb = self.embedding(x)
        x_pos_enc = self.pos_enc[:, :length, :]

        x_emb *= tf.cast(self.d_model, tf.float32)
        x_emb += x_pos_enc
```

```
        return x_emb


sample_pos_emb = PositionalEmbedding(vocab_size, d_model = 512)
cap_emb = sample_pos_emb(cap)
print(cap_emb.shape)
```

```
(64, 37, 512)
```

```
class BaseAttention(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super().__init__()
        self.mha = tf.keras.layers.MultiHeadAttention(**kwargs)
        self.add = tf.keras.layers.Add()
        self.layernorm = tf.keras.layers.LayerNormalization()


class CausalAttention(BaseAttention):
    def call(self, x):
        attn_output = self.mha(query = x,
                               key = x,
                               value = x,
                               use_causal_mask = True)

        x = self.add([x, attn_output])
        x = self.layernorm(x)
        return x


sample_csa = CausalAttention(num_heads = 8, key_dim = 512)
sample_csa_out = sample_csa(cap_emb)
print(sample_csa_out.shape)
```

```
(64, 37, 512)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_1' (of type CausalAttention)
  warnings.warn(
```

```
# class CrossAttention(BaseAttention):
#     def call(self, context, x):
#         attn_out, attn_scores = self.mha(query = x,
#                                          key = context,
#                                          value = context,
#                                          return_attention_scores = True)

#         self.last_attention_scores = attn_scores

#         x = self.add([x, attn_out])
#         x = self.layernorm(x)
#         return x
class CrossAttention(BaseAttention):
    def __init__(self, num_heads, key_dim, dropout=0.1, **kwargs):
        super().__init__(num_heads=num_heads, key_dim=key_dim, dropout=dropout, **kwargs)
        self.project_context = tf.keras.layers.Dense(key_dim)  # Projection layer for context

    def call(self, context, x):
        # Project context to match the feature dimension of x (query)
        context = self.project_context(context)

        attn_out, attn_scores = self.mha(query=x, key=context, value=context, return_attention_scores=True)

        self.last_attention_scores = attn_scores

        x = self.add([x, attn_out])
        x = self.layernorm(x)
        return x



img_ = einops.rearrange(img, 'b h w c -> b (h w) c')
print(f"Original shape: {img.shape}, Rearranged shape: {img_.shape}")
print(img.shape, img_.shape)
```

```
Original shape: (64, 7, 7, 576), Rearranged shape: (64, 49, 576)
(64, 7, 7, 576) (64, 49, 576)
```

```
print("x shape:", sample_csa_out.shape)   # (64, 37, 512)
print("context shape:", img_.shape)      # (64, 49, 576)

# Optionally project context to match x's feature size
context = tf.keras.layers.Dense(512)(img_)  # Project context to 512 features
```

```
x shape: (64, 37, 512)
context shape: (64, 49, 576)
```

```
# Now pass to CrossAttention
sample_ca = CrossAttention(num_heads = 8, key_dim = 512)
sample_ca_out = sample_ca(context=context, x=sample_csa_out)
print(sample_ca_out.shape)  # Should print (64, 37, 512) or similar
```

```
(64, 37, 512)
```

```
class FeedForward(tf.keras.layers.Layer):
    def __init__(self, d_model, dff, dropout_rate = 0.1):
        super().__init__()
        self.seq = tf.keras.Sequential([
            tf.keras.layers.Dense(dff, activation = 'relu'),
            tf.keras.layers.Dense(d_model),
            tf.keras.layers.Dropout(rate = dropout_rate)
        ])

        self.add = tf.keras.layers.Add()
        self.layernorm = tf.keras.layers.LayerNormalization()

    def call(self, x):
        x = self.add([x, self.seq(x)])
        x = self.layernorm(x)
        return x


sample_ffn = FeedForward(d_model = 512, dff = 2048)
sample_ffn_out = sample_ffn(sample_ca_out)
print(sample_ffn_out.shape)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential' (of type Sequential) was passed a
  warnings.warn(
(64, 37, 512)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward' (of type FeedForward) was passe
  warnings.warn(
```

```
class DecoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, dff, num_heads, dropout_rate = 0.1):
        super().__init__()
        self.causal_attention = CausalAttention(num_heads = num_heads,
                                                key_dim = d_model,
                                                dropout = dropout_rate)
        self.cross_attention = CrossAttention(num_heads = num_heads,
                                              key_dim = d_model,
                                              dropout = dropout_rate)
        self.ffn = FeedForward(d_model = d_model,
                               dff = dff,
                               dropout_rate = dropout_rate)
        self.last_attention_scores = None

    def call(self, context, x):
        x = self.causal_attention(x)
        x = self.cross_attention(context = context, x = x)
        x = self.ffn(x)
        self.last_attention_scores = self.cross_attention.last_attention_scores

        return x



sample_decoder_layer = DecoderLayer(d_model = 512, dff = 2048, num_heads = 8, dropout_rate = 0.2)
sample_decoder_layer_out = sample_decoder_layer(context = img_, x = cap_emb)
print(sample_decoder_layer_out.shape)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_2' (of type CausalAttention)
  warnings.warn(
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_1' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_1' (of type FeedForward) was pas
  warnings.warn(
(64, 37, 512)
```

```
class OutputLayer(tf.keras.layers.Layer):
    def __init__(self, vocab, bad_tokens = ('', '[UNK]', '[START]')):
        super().__init__()
        self.vocab = vocab
        self.bad_tokens = bad_tokens
        self.bias = 0
        self.dense_layer = tf.keras.layers.Dense(len(vocab), activation = tf.nn.log_softmax)

    def adapt(self, cap_ds):
        word_idx = {word : idx for idx, word in enumerate(self.vocab)}
        counts = collections.Counter()
        for tokens in cap_ds:
            counts.update(tokens.numpy().flatten())

        counts_arr = np.zeros((len(self.vocab), ))
        for token_id, cnt in counts.items():
            counts_arr[token_id] = cnt

        bad_indices = np.array([word_idx[word] for word in self.bad_tokens])
        counts_arr[bad_indices] = 0

        counts_prob = counts_arr / counts_arr.sum()
        counts_prob[counts_arr == 0] = 1
        log_p = np.log(counts_prob)

        entropy = (-counts_prob*log_p).sum()

        print(f'uniform_entropy : {np.log(len(self.vocab))}')
        print(f'curr_entropy : {entropy}')
        log_p[counts_arr == 0] = -1e9

        self.bias = log_p[tf.newaxis, tf.newaxis, :]

    def call(self, x):
        return self.dense_layer(x) + self.bias


vocab = vectorizer.get_vocabulary()
sample_output_layer = OutputLayer(vocab)
sample_output_layer.adapt(train_ds.map(lambda img, cap: cap))
sample_output_layer_out = sample_output_layer(sample_decoder_layer_out)
print(sample_output_layer_out.shape)
```

```
uniform_entropy : 8.517193191416238
curr_entropy : 5.292547935444457
(64, 37, 5000)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'output_layer' (of type OutputLayer) was passe
  warnings.warn(
```

```
class Decoder(tf.keras.layers.Layer):
    def __init__(self, num_layers, num_heads, d_model, dff, dropout_rate = 0.1):
        super().__init__()
        self.num_layers = num_layers

        self.positional_embedding = PositionalEmbedding(vocab_size, d_model)
        self.decoder_layers = [DecoderLayer(d_model, dff, num_heads, dropout_rate) for _ in range(num_layers)]
        self.last_attention_scores = None

    def call(self, context, x):
        x = self.positional_embedding(x)
        for i in range(self.num_layers):
            x = self.decoder_layers[i](context = context, x = x)

        self.last_attention_scores = self.decoder_layers[-1].last_attention_scores
        return x


for (img, cap), cap_labels in train_ds.take(1):
    break
```

```
print(img.shape)
print(cap.shape)
print(cap_labels.shape)

img_features = einops.rearrange(img, 'b h w c -> b (h w) c')
print(img_features.shape)
```

```
(64, 7, 7, 576)
(64, 37)
(64, 37)
(64, 49, 576)
```

```
%%time
sample_decoder = Decoder(num_layers = 6, num_heads = 8, d_model = 512, dff = 2048)
sample_decoder_output = sample_decoder(context = img_features, x = cap)
print(sample_decoder_output.shape)
print(sample_decoder.last_attention_scores.shape)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_3' (of type CausalAttention)
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_2' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_2' (of type FeedForward) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_4' (of type CausalAttention)
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_3' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_3' (of type FeedForward) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_5' (of type CausalAttention)
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_4' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_4' (of type FeedForward) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_6' (of type CausalAttention)
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_5' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_5' (of type FeedForward) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_7' (of type CausalAttention)
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_6' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_6' (of type FeedForward) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_8' (of type CausalAttention)
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_7' (of type Sequential) was passed
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_7' (of type FeedForward) was pas
  warnings.warn(
(64, 37, 512)
(64, 8, 37, 49)
CPU times: user 29.6 s, sys: 5.41 s, total: 35 s
Wall time: 23.6 s
```

```
class Captioner(tf.keras.Model):
    @classmethod
    def add_method(cls, fun):
        setattr(cls, fun.__name__, fun)
        return fun

    def __init__(self, vectorizer, feature_extractor, output_layer, num_layers, num_heads, d_model, dff, pred_max_len =  50, dropout_rate =
        super().__init__()
        self.feature_extractor = feature_extractor
        self.vectorizer = vectorizer
        self.output_layer = output_layer
        self.decoder = Decoder(num_layers, num_heads, d_model, dff, dropout_rate)
        self.max_len = pred_max_len
        self.vocab = self.vectorizer.get_vocabulary()

    def call(self, inputs):
        context, cap = inputs
        if context.shape[-1] == 3:
```

```
            context = self.feature_extractor(context)
        context = einops.rearrange(context, 'b h w c -> b (h w) c')

        if cap.dtype == tf.string:
            cap = self.vectorizer([cap])


        x = self.decoder(context = context, x = cap)
        x = self.output_layer(x)
        return x


%%time
sample_captioner = Captioner(vectorizer = vectorizer,
                             feature_extractor = feature_extractor,
                             output_layer = sample_output_layer,
                             num_layers = 6,
                             num_heads = 8,
                             d_model = 512,
                             dff = 2048,
                             dropout_rate = 0.2)

print(sample_captioner((img, cap)).shape)
print(cap_labels.shape)
print(sample_captioner.summary())
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_9' (of type CausalAttenti
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_8' (of type Sequential) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_8' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_10' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_9' (of type Sequential) was pas
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_9' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_11' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_10' (of type Sequential) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_10' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_12' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_11' (of type Sequential) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_11' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_13' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_12' (of type Sequential) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_12' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_14' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_13' (of type Sequential) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_13' (of type FeedForward) was
  warnings.warn(
(64, 37, 5000)
(64, 37)
Model: "captioner"
```

Layer (type)	Output Shape	Param #
MobileNetV3Small (Functional)	(None, 7, 7, 576)	939,120
text_vectorization (TextVectorization)	(None, None)	0
output_layer (OutputLayer)	?	2,565,000
decoder_1 (Decoder)	?	117,766,144

```
 Total params: 121,270,264 (462.61 MB)
 Trainable params: 120,331,144 (459.03 MB)
 Non-trainable params: 939,120 (3.58 MB)
None
CPU times: user 33.6 s, sys: 7.58 s, total: 41.2 s
Wall time: 31.4 s
```

```
d_model = 128
dff = 128
dropout_rate = 0.4
num_layers = 2
num_heads = 2
output_layer = OutputLayer(vocab)
output_layer.adapt(train_ds.map(lambda img_feature, cap: cap))
```

```
uniform_entropy : 8.517193191416238
curr_entropy : 5.292547935444457
```

```
for (img_feature, cap), cap_labels in train_ds.take(1):
    break
print(img_feature.shape)
print(cap.shape)
print(cap_labels.shape)
```

```
(64, 7, 7, 576)
(64, 34)
(64, 34)
```

```
%%time
captioner_model = Captioner(vectorizer = vectorizer,
                            feature_extractor = feature_extractor,
                            output_layer = output_layer,
                            num_layers = num_layers,
                            num_heads = num_heads,
                            d_model = d_model,
                            dff = dff,
                            dropout_rate = dropout_rate)

print(captioner_model((img_feature, cap), training = False).shape)
captioner_model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_15' (of type CausalAttention
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_14' (of type Sequential) was passe
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_14' (of type FeedForward) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_16' (of type CausalAttention
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_15' (of type Sequential) was passe
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_15' (of type FeedForward) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'output_layer_1' (of type OutputLayer) was pas
  warnings.warn(
(64, 34, 5000)
Model: "captioner_1"
```

Layer (type)	Output Shape	Param #
MobileNetV3Small (Functional)	(None, 7, 7, 576)	939,120
text_vectorization (TextVectorization)	(None, None)	0
output_layer_1 (OutputLayer)	?	645,000
decoder_2 (Decoder)	?	1,383,168

```
 Total params: 2,967,288 (11.32 MB)
 Trainable params: 2,028,168 (7.74 MB)
 Non-trainable params: 939,120 (3.58 MB)
CPU times: user 2.95 s, sys: 304 ms, total: 3.25 s
Wall time: 2.99 s
```

```
@Captioner.add_method
def generate_text(self, img, temperature = 0.5):
    if img.shape[-1] == 3:
        img = self.feature_extractor(img)
    start_token = text_to_id_vectorizer([['[START]']])

    start_idx = self.vocab.index('[START]')
    end_idx = self.vocab.index('[END]')

    for i in range(self.max_len):
        preds = self((img, start_token))
        preds = preds[:, -1, :]
        if temperature == 0.0:
            pred_idx = tf.argmax(preds, axis = -1)[:, tf.newaxis]
        else:
            preds /= temperature
            pred_idx = tf.random.categorical(preds, num_samples = 1)

        start_token = tf.concat([start_token, pred_idx], axis = -1)

        if pred_idx[0][0] == end_idx:
            break

    return id_to_text(start_token).numpy()[0].decode('utf-8')


def brevity_penalty(can, ref):
    can_tokens = nltk.word_tokenize(can)
    ref_tokens = nltk.word_tokenize(ref)
    if len(can_tokens) == 0:
        return 0.0
    return min(1, np.exp(1 - (len(ref_tokens) / len(can_tokens))))
```

```
def precision(can, ref, n):
    can_n = collections.Counter(ngrams(nltk.word_tokenize(can), n))
    ref_n = collections.Counter(ngrams(nltk.word_tokenize(ref), n))
    total = sum(can_n.values())

    if total == 0:
        return 0

    for n_g in can_n:
        if n_g in ref_n:
            can_n[n_g] = min(can_n[n_g], ref_n[n_g])
        else:
            can_n[n_g] = 0
    return sum(can_n.values()) / total

def bleu_score(can , ref, n_gram_range = 2):
    precisions = []
    b_p = brevity_penalty(can, ref)
    for n in range(1, n_gram_range + 1):
        precisions.append(precision(can, ref, n))
    precisions = np.array(precisions)

    if 0 in precisions:
        # As log of 0 will be -inf and exp of that will be back to 0 with warning.
        return 0.0
    return b_p * np.exp(np.log(precisions).mean())

def recall(can, ref, n):
    can_n = collections.Counter(ngrams(nltk.word_tokenize(can), n))
    ref_n = collections.Counter(ngrams(nltk.word_tokenize(ref), n))
    total = sum(ref_n.values())

    if total == 0:
        return 0

    for n_g in ref_n:
        if n_g in can_n:
            ref_n[n_g] = min(can_n[n_g], ref_n[n_g])
        else:
            ref_n[n_g] = 0
    return sum(ref_n.values()) / total

def rouge_score(can , ref, n_gram_range = 2):
    recalls = []
    b_p = brevity_penalty(can, ref)
    for n in range(1, n_gram_range + 1):
        recalls.append(recall(can, ref, n))
    recalls = np.array(recalls)

    if 0 in recalls:
        # As log of 0 will be -inf and exp of that will be back to 0 with warning.
        return 0.0
    return b_p * np.exp(np.log(recalls).mean())


@Captioner.add_method
def f_score(self, can, refs, n_gram_range = 1):
    b_scores = [bleu_score(can, ref, n_gram_range) for ref in refs]
    r_scores = [rouge_score(can, ref, n_gram_range)  for ref in refs]
    f_vals = []
    for b_score, r_score in zip(b_scores, r_scores):
        if b_score + r_score == 0:
            return 0
        f_vals.append((2*b_score*r_score) / (b_score + r_score))

    return max(f_vals)


def masked_loss(labels, preds):
    loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True, reduction = 'none')
    loss = tf.cast(loss_fn(labels, preds), tf.float32)
    mask = ((labels != 0) & (loss < 1e8))
    mask = tf.cast(mask, tf.float32)
    loss *= mask
    return tf.math.reduce_sum(loss) / tf.math.reduce_sum(mask)

def masked_accuracy(labels, preds):
```

```
    preds = tf.cast(tf.argmax(preds, axis = -1), tf.float32)
    labels = tf.cast(labels, tf.float32)
    mask = tf.cast(labels != 0, tf.float32)
    acc = tf.cast(preds == labels, tf.float32)
    acc *= mask
    return tf.math.reduce_sum(acc) / tf.math.reduce_sum(mask)


for (img, cap), cap_labels in train_ds.take(1):
    break
print(img.shape)
print(cap.shape)
print(cap_labels.shape)

preds = captioner_model((img, cap))
print(preds.shape)

print(masked_loss(cap, preds))
print(masked_accuracy(cap, preds))
```

```
(64, 7, 7, 576)
(64, 37)
(64, 37)
(64, 37, 5000)
tf.Tensor(5.779553, shape=(), dtype=float32)
tf.Tensor(0.11827957, shape=(), dtype=float32)
```

```
for img_path, caps in train_raw.take(1):
    break
print(img_path)
img = load_img(img_path)
print(img.shape)
caps = [cap.numpy().decode('utf-8') for cap in caps]
print(caps)
Image.open(img_path.numpy().decode('utf-8'))
```

```
tf.Tensor(b'flickr8k/Flicker8k_Dataset/2513260012_03d33305cf.jpg', shape=(), dtype=string)
(224, 224, 3)
['A black dog is running after a white dog in the snow .', 'Black dog chasing brown dog through snow', 'Two dogs chase each other across
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
import nltk
nltk.download('punkt', force=True)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
import nltk
print(nltk.data.find('tokenizers/punkt'))
```

```
/root/nltk_data/tokenizers/punkt
```

```
!pip uninstall nltk
!pip install nltk
```

```
Found existing installation: nltk 3.9.1
Uninstalling nltk-3.9.1:
  Would remove:
    /usr/local/bin/nltk
    /usr/local/lib/python3.10/dist-packages/nltk-3.9.1.dist-info/*
    /usr/local/lib/python3.10/dist-packages/nltk/*
Proceed (Y/n)? ERROR: Operation cancelled by user
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/cli/base_command.py", line 179, in exc_logging_wrapper
    status = run_func(*args)
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/commands/uninstall.py", line 106, in run
    uninstall_pathset = req.uninstall(
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/req/req_install.py", line 722, in uninstall
    uninstalled_pathset.remove(auto_confirm, verbose)
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/req/req_uninstall.py", line 364, in remove
    if auto_confirm or self._allowed_to_proceed(verbose):
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/req/req_uninstall.py", line 404, in _allowed_to_proceed
    return ask("Proceed (Y/n)? ", ("y", "n", "")) != "n"
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/utils/misc.py", line 235, in ask
    response = input(message)
KeyboardInterrupt

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/bin/pip3", line 8, in <module>
    sys.exit(main())
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/cli/main.py", line 80, in main
    return command.main(cmd_args)
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/cli/base_command.py", line 100, in main
    return self._main(args)
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/cli/base_command.py", line 232, in _main
    return run(options, args)
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/cli/base_command.py", line 216, in exc_logging_wrapper
    logger.debug("Exception information:", exc_info=True)
  File "/usr/lib/python3.10/logging/__init__.py", line 1465, in debug
    self._log(DEBUG, msg, args, **kwargs)
  File "/usr/lib/python3.10/logging/__init__.py", line 1624, in _log
    self.handle(record)
  File "/usr/lib/python3.10/logging/__init__.py", line 1634, in handle
    self.callHandlers(record)
  File "/usr/lib/python3.10/logging/__init__.py", line 1696, in callHandlers
    hdlr.handle(record)
  File "/usr/lib/python3.10/logging/__init__.py", line 968, in handle
    self.emit(record)
  File "/usr/lib/python3.10/logging/handlers.py", line 75, in emit
    logging.FileHandler.emit(self, record)
  File "/usr/lib/python3.10/logging/__init__.py", line 1218, in emit
    StreamHandler.emit(self, record)
  File "/usr/lib/python3.10/logging/__init__.py", line 1100, in emit
    msg = self.format(record)
  File "/usr/lib/python3.10/logging/__init__.py", line 943, in format
    return fmt.format(record)
  File "/usr/local/lib/python3.10/dist-packages/pip/_internal/utils/logging.py", line 112, in format
    formatted = super().format(record)
  File "/usr/lib/python3.10/logging/__init__.py", line 686, in format
    record.exc_text = self.formatException(record.exc_info)
  File "/usr/lib/python3.10/logging/__init__.py", line 636, in formatException
```

```
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
from nltk.tokenize import word_tokenize

test_sentence = "This is a test sentence."
tokens = word_tokenize(test_sentence)
print(tokens)
```

```
['This', 'is', 'a', 'test', 'sentence', '.']
```

```
# class GenerateText(tf.keras.callbacks.Callback):
#     def __init__(self, img = img):
#         self.image = img[tf.newaxis, ...]
#         self.caps = caps

#     def on_epoch_end(self, epochs = None, logs = None):
#         print('\n')
#         for temp in (0, 0.5, 1):
#             gen_text = self.model.generate_text(self.image)
#             f_val = self.model.f_score(gen_text, self.caps, n_gram_range = 1)
#             print(f'Generated_text: {gen_text}, \t f_score: {f_val}')
class GenerateText:
    def __init__(self, img, caps):
        self.image = img[tf.newaxis, ...]
        self.caps = caps
        self.model = None  # Allow manual assignment of model.

    def set_model(self, model):
        """Set the model for the callback."""
        self.model = model
    def set_params(self, params):
        """Set the training parameters for the callback."""
        self.params = params

    def on_train_begin(self, logs=None):
        """Called at the beginning of training."""
        print("\nTraining is starting...")
    def on_train_end(self, logs=None):
        """Called at the ending of training."""
        print("\nTraining is ending...")
    def on_epoch_begin(self, epoch, logs=None):
        """Called at the beginning of each epoch."""
        print(f"\nStarting epoch {epoch + 1}...")
    def on_train_batch_begin(self, batch, logs=None):
        pass

    def on_train_batch_end(self, batch, logs=None):
        pass

    def on_test_begin(self, logs=None):
        print("Validation is starting...")

    def on_test_end(self, logs=None):
        print("Validation has ended.")

    def on_test_batch_begin(self, batch, logs=None):
        pass

    def on_test_batch_end(self, batch, logs=None):
        pass

    def on_epoch_end(self, epoch=None, logs=None):
        if self.model is None:
            raise ValueError("Model is not set for GenerateText.")
        print('\n')
        for temp in (0, 0.5, 1):
            gen_text = self.model.generate_text(self.image, temperature=temp)
            f_val = self.model.f_score(gen_text, self.caps, n_gram_range=1)
            print(f'Temperature {temp}: Generated text: {gen_text}, F-score: {f_val}')



# sample_gen_text = GenerateText()
# sample_gen_text.model = captioner_model
# sample_gen_text.on_epoch_end(0)
sample_gen_text = GenerateText(img=img, caps=caps)
sample_gen_text.model = captioner_model
sample_gen_text.on_epoch_end(0)
```

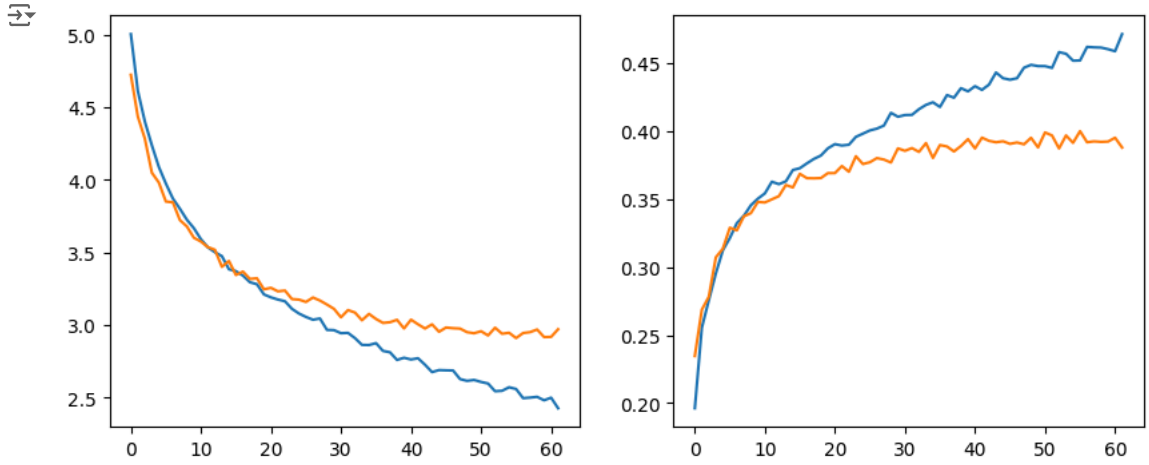
```
/usr/local/lib/python3.10/dist-packages/keras/src/ops/nn.py:545: UserWarning: You are using a softmax over axis 3 of a tensor of shape (
  warnings.warn(
Temperature 0: Generated text: a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a, F-sc
Temperature 0.5: Generated text: the the a a, F-score: 0
Temperature 1: Generated text: woman with children stick a steps red a glasses stands road camera brown for day s drinks, F-score: 0
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0001)
callbacks = [GenerateText(img=img, caps=caps),
             tf.keras.callbacks.EarlyStopping(patience = 6, restore_best_weights = True)]


captioner_model.compile(loss = masked_loss, optimizer = optimizer, metrics = [masked_accuracy])


hist = captioner_model.fit(
    train_ds.repeat(),
    steps_per_epoch = 100,
    validation_data = test_ds.repeat(),
    validation_steps = 20,
    epochs = 150,
    callbacks = callbacks
)
```

```
Training is starting...

Starting epoch 1...
Epoch 1/150
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 5.1317 - masked_accuracy: 0.1700Validation is starting...
Validation has ended.


Temperature 0: Generated text: a dog is a dog is, F-score: 0
Temperature 0.5: Generated text: a white and is a dog with a dog in a, F-score: 0
Temperature 1: Generated text: a writing shopping at young mask shower is big family in the, F-score: 0
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - loss: 5.1304 - masked_accuracy: 0.1702 - val_loss: 4.7240 - val_masked_accuracy: 0.2347

Starting epoch 2...
Epoch 2/150
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 4.6785 - masked_accuracy: 0.2496Validation is starting...
Validation has ended.


Temperature 0: Generated text: a dog is in a, F-score: 0
Temperature 0.5: Generated text: a dog is the the air, F-score: 0.13111714691140958
Temperature 1: Generated text: a brown and bird is car, F-score: 0
100/100 ━━━━━━━━━━━━━━━━━━━━ 113s 1s/step - loss: 4.6779 - masked_accuracy: 0.2497 - val_loss: 4.4359 - val_masked_accuracy: 0.2687

Starting epoch 3...
Epoch 3/150
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 4.4399 - masked_accuracy: 0.2702Validation is starting...
Validation has ended.


Temperature 0: Generated text: a dog is in a white dog is in a white dog is in a white dog is in a white dog is in a white dog, F-scc
Temperature 0.5: Generated text: a dog is on a snow, F-score: 0
Temperature 1: Generated text: two the native hiking floating, F-score: 0
100/100 ━━━━━━━━━━━━━━━━━━━━ 119s 1s/step - loss: 4.4395 - masked_accuracy: 0.2703 - val_loss: 4.2844 - val_masked_accuracy: 0.2783

Starting epoch 4...
Epoch 4/150
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 4.2742 - masked_accuracy: 0.2902Validation is starting...
Validation has ended.


Temperature 0: Generated text: a dog is running in a dog is running in the snow, F-score: 0.5888284253627668
Temperature 0.5: Generated text: a dog in the the dog is running on a grass, F-score: 0.4863558688771886
Temperature 1: Generated text: the dog play in them at a jacket while children in area, F-score: 0.3
100/100 ━━━━━━━━━━━━━━━━━━━━ 121s 1s/step - loss: 4.2739 - masked_accuracy: 0.2902 - val_loss: 4.0495 - val_masked_accuracy: 0.3072

Starting epoch 5...
Epoch 5/150
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 4.1440 - masked_accuracy: 0.3077Validation is starting...
Validation has ended.


Temperature 0: Generated text: a dog is running in the snow, F-score: 0.34675115990007266
Temperature 0.5: Generated text: a dog is running through the snow, F-score: 0.42857142857142855
Temperature 1: Generated text: and yellow dog colors is wearing a man in a black and green frisbee, F-score: 0
100/100 ━━━━━━━━━━━━━━━━━━━━ 119s 1s/step - loss: 4.1435 - masked_accuracy: 0.3077 - val_loss: 3.9816 - val_masked_accuracy: 0.3136
```

```
plt.figure(figsize = (10, 4))
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'], label = 'loss')
```

```
plt.plot(hist.history['val_loss'], label = 'val_loss')

plt.subplot(1, 2, 2)
plt.plot(hist.history['masked_accuracy'], label = 'masked_accuracy')
plt.plot(hist.history['val_masked_accuracy'], label = 'val_masked_accuracy')
plt.show()
```




```
from keras.saving import register_keras_serializable

@register_keras_serializable(package="Custom", name="Captioner")
class Captioner(tf.keras.Model):
    # Your existing Captioner class definition
    def __init__(self, vectorizer, feature_extractor, output_layer, num_layers, num_heads, d_model, dff, pred_max_len=50, dropout_rate=0.1):
        super().__init__()
        self.feature_extractor = feature_extractor
        self.vectorizer = vectorizer
        self.output_layer = output_layer
        self.decoder = Decoder(num_layers, num_heads, d_model, dff, dropout_rate)
        self.max_len = pred_max_len
        self.vocab = self.vectorizer.get_vocabulary()

    def call(self, inputs):
        context, cap = inputs
        if context.shape[-1] == 3:
            context = self.feature_extractor(context)
        context = einops.rearrange(context, 'b h w c -> b (h w) c')

        if cap.dtype == tf.string:
            cap = self.vectorizer([cap])

        x = self.decoder(context=context, x=cap)
        x = self.output_layer(x)
        return x


captioner_model.save("captioner_model.keras")


captioner_model.save("captioner_model1.h5")
```

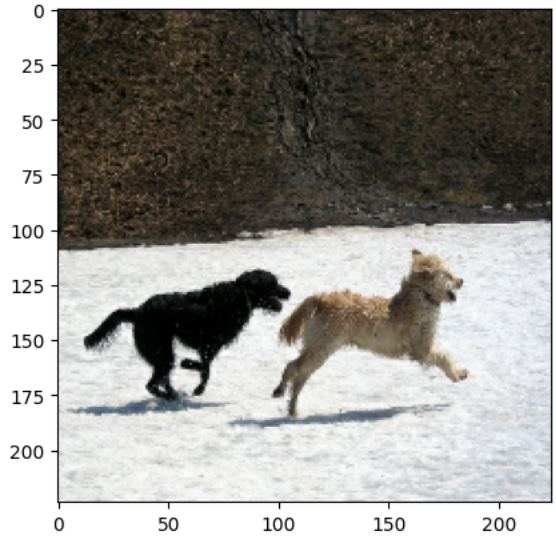
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

```
captioner_model.export("captioner_model_tf")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'query' (of type EinsumDense) was passed ar
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'key' (of type EinsumDense) was passed an i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'value' (of type EinsumDense) was passed ar
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_15' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_14' (of type Sequential) was pa
```

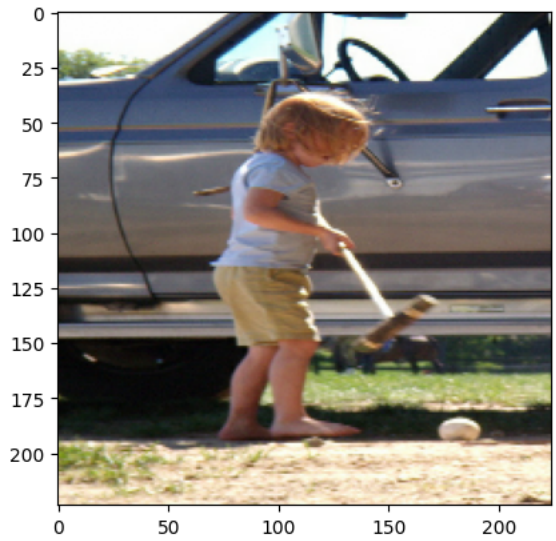
```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_14' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_16' (of type CausalAttent
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_15' (of type Sequential) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_15' (of type FeedForward) was
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'output_layer_1' (of type OutputLayer) was
  warnings.warn(
Saved artifact at 'captioner_model_tf'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): Tuple[TensorSpec(shape=(None, 7, 7, 576), dtype=tf.float32, name=None), TensorSpec(shape=(None, 34), dtyp
Output Type:
  TensorSpec(shape=(None, 34, 5000), dtype=tf.float32, name=None)
Captures:
  133146388307760: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146387851296: TensorSpec(shape=(1, 2048, 128), dtype=tf.float32, name=None)
  133146478324624: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146478326032: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146477701856: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146477699392: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386862624: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386869488: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386859104: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133147181133216: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146388422624: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146477691296: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146388415936: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386863328: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386865792: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386862272: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146506148000: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386860336: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386861920: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146386870016: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146478328320: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146478014032: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146479032832: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146479022976: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146479024736: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146478012624: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146479019632: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146478007168: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146479026496: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133146479021040: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

```
#loaded_model = tf.keras.models.load_model("captioner_model.keras")
#loaded_model = tf.keras.models.load_model(
#    "captioner_model.keras",
#    custom_objects={"Captioner": Captioner}
#)


for img_path, caps in train_raw.batch(4).take(1):
    break

img = load_img(img_path[0].numpy().decode('utf-8'))
plt.imshow(img/255.0)
plt.show()
print(f'Generated Caption: {captioner_model.generate_text(img[tf.newaxis, ...])}')
```



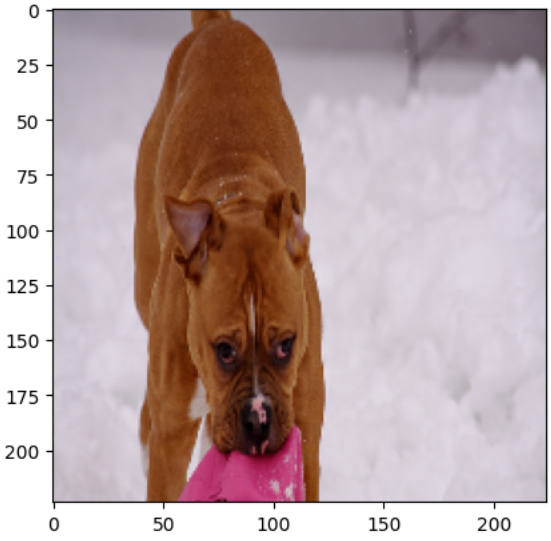
```
Generated Caption: three dogs are running on a snowy hill
```

```
img = load_img(img_path[1].numpy().decode('utf-8'))
plt.imshow(img/255.0)
plt.show()
print(f'Generated Caption: {captioner_model.generate_text(img[tf.newaxis, ...])}')
```



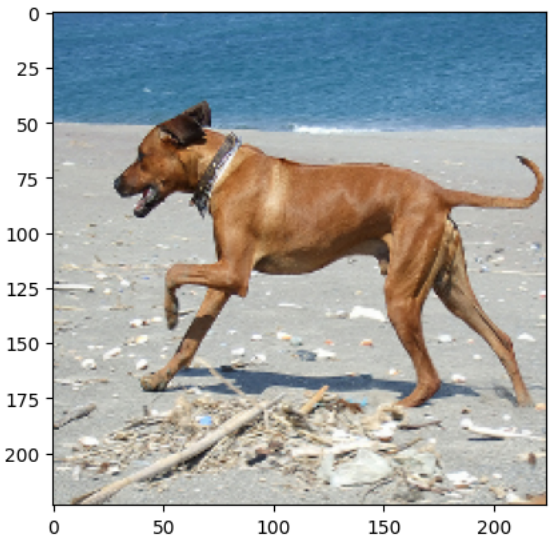
```
Generated Caption: a girl is standing in the grass with a woman in the background
```

```
img = load_img(img_path[2].numpy().decode('utf-8'))
plt.imshow(img/255.0)
plt.show()
print(f'Generated Caption: {captioner_model.generate_text(img[tf.newaxis, ...])}')
```



```
Generated Caption: a brown dog is running on the grass
```

```
img = load_img(img_path[3].numpy().decode('utf-8'))
plt.imshow(img/255.0)
plt.show()
print(f'Generated Caption: {captioner_model.generate_text(img[tf.newaxis, ...])}')
```



```
Generated Caption: a brown dog is running on the beach
```

```
for img_path, cap in train_raw.take(1):
    break
img = load_img(img_path)
cap_gen = captioner_model.generate_text(img[tf.newaxis, ...])

plt.imshow(img/255)
plt.axis('off')
plt.show()
print(cap_gen)
```

two dogs are running on the snow

```
for img_path, cap in test_raw.take(1):
    break
img = load_img(img_path)
cap_gen = captioner_model.generate_text(img[tf.newaxis, ...])

plt.imshow(img/255)
plt.axis('off')
plt.show()
print(cap_gen)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'query' (of type EinsumDense) was passed an in
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'key' (of type EinsumDense) was passed an inpu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'value' (of type EinsumDense) was passed an in
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/ops/nn.py:545: UserWarning: You are using a softmax over axis 3 of a tensor of shape (
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_15' (of type CausalAttention
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_14' (of type Sequential) was passe
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_14' (of type FeedForward) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'causal_attention_16' (of type CausalAttention
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'sequential_15' (of type Sequential) was passe
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'feed_forward_15' (of type FeedForward) was pa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/layer.py:934: UserWarning: Layer 'output_layer_1' (of type OutputLayer) was pas
  warnings.warn(
```

a dog is running in the snow

```
cap_gen_tokens = cap_gen.split() + ['[END]']
print(len(cap_gen_tokens))
```

——