2.2 Formula Recognition Using seq2seq Models - Training the model

August 5, 2019

1 2.2 Formula Recognition Using seq2seq Models - Training the model

In this notebook, we train a seq2seq model that transcribes typeset formulas from images into Latex markup. The model consists of a few components:

An encoder

- This is the same model as we used in the previous notebook, consisting of 3 convolutional layers with increasing hidden units.

A decoder

- the decoder takes the features from our encoder, and generates an "embedding" for them, which is a fixed-sized vector that represents the visual content of the input image. We apply Bahdanau attention over this feature vector, and concatenate it with the hidden state of a GRU. This is then fed back into a GRU. Finally, we use two fully-connected layers where the last layer has a softmax activation in order to produce our final prediction.

References

- Image Captioning With Attention
- im2latex paper
- Guillaume Genthial's im2latex implementation

```
In [2]: import tensorflow as tf
```

```
In [3]: print(tf.__version__)
        print(tf.test.is_gpu_available())
        print(tf.test.is_built_with_cuda())
        print(tf.test.gpu_device_name())
```

```
2.0.0-beta1
False
False
```

As before, the directory structure is defined as:

```
project/
    data/ -- Contains the data used in the project (both original and derived)
    doc/ -- Project documentation (including report & summary)
    figs/ -- Any saved figures generated by the project
    notebooks/ -- All notebooks for the project
    scripts/ -- Scripts used for various reasons, such as pre-processing
    src/ -- Regular python code
```

In [41]:
```python
import os
import glob

### Make sure our data is in order
data_base_dir = "../data"
figs_base_dir = "../figs"

original_data_path = data_base_dir + "/original/formula/"
processed_data_path = data_base_dir + "/processed/formula/"
pickle_data_path = data_base_dir + "/pickle/formula/"

assert os.path.exists(original_data_path), "Original data path does not exist."
```

In [42]:
```python
import re

def atoi(text):
    return int(text) if text.isdigit() else text

def natural_keys(text):
    '''
    alist.sort(key=natural_keys) sorts in human order
    http://nedbatchelder.com/blog/200712/human_sorting.html
    (See Toothy's implementation in the comments)
    '''
    return [ atoi(c) for c in re.split(r'(\d+)', text) ]
```

In [67]:
```python
training_images = glob.glob(f"{processed_data_path}/images/train/*.png")
training_images.sort(key=natural_keys)
validation_images = glob.glob(f"{processed_data_path}/images/validate/*.png")
validation_images.sort(key=natural_keys)
test_images = glob.glob(f"{processed_data_path}/images/test/*.png")
test_images.sort(key=natural_keys)

print(f"Found {len(training_images)} training images.")
print(f"Found {len(validation_images)} validation images.")
print(f"Found {len(test_images)} test images.")
```

```
Found 17 training images.
Found 17 validation images.
```

```
Found 17 test images.
```

```
In [44]: import pandas as pd
         import numpy as np

         def load_labels(labels_path, matches_path):
             with open(labels_path) as f:
                 labels = np.array(f.read().splitlines())
             matches =  pd.read_csv(matches_path, sep=' ', header=None).values
             return labels[[list(map(lambda f: f[1], matches))][0]]

         train_labels_path = f"{original_data_path}train.formulas.norm.txt"
         train_matches_path = f"{processed_data_path}images/train/train.matching.txt"
         train_labels = load_labels(train_labels_path, train_matches_path)
         print(f"Got {len(train_labels)} training labels.")

         validate_labels_path = f"{original_data_path}val.formulas.norm.txt"
         validate_matches_path = f"{processed_data_path}images/validate/val.matching.txt"
         validate_labels = load_labels(validate_labels_path, validate_matches_path)
         print(f"Got {len(validate_labels)} validation labels.")
```

```
Got 17 training labels.
Got 17 validation labels.
```

The vocabulary class encapsulates the vocabulary (in other words, all possible tokens). It also has methods for tokenizing, padding, and performing a reverse lookup.

```
In [45]: class Vocab(object):
             def __init__(self, vocab_path):
                 self.build_vocab(vocab_path)

             def build_vocab(self, vocab_path):
                 '''
                 Builds the complete vocabulary, including special tokens
                 '''
                 self.unk   = "<UNK>"
                 self.start = "<SOS>"
                 self.end   = "<END>"
                 self.pad   = "<PAD>"

                 # First, load our vocab from disk & determine
                 # highest index in mapping.
                 vocab = self.load_vocab(vocab_path)
                 max_index = max(vocab.values())

                 # Compile special token mapping
                 special_tokens = {
```

```
            self.unk : max_index + 1,
            self.start : max_index + 2,
            self.end : max_index + 3,
            self.pad : max_index + 4
        }

        # Merge dicts to produce final word index
        self.token_index = {**vocab, **special_tokens}
        self.reverse_index = {v: k for k, v in self.token_index.items()}

    def load_vocab(self, vocab_path):
        '''
        Load vocabulary from file
        '''
        token_index = {}
        with open(vocab_path) as f:
            for idx, token in enumerate(f):
                token = token.strip()
                token_index[token] = idx
        assert len(token_index) > 0, "Could not build word index"
        return token_index

    def tokenize_formula(self, formula):
        '''
        Converts a formula into a sequence of tokens using the vocabulary
        '''
        def lookup_token(token):
            return self.token_index[token] if token in self.token_index else self.toke
        tokens = formula.strip().split(' ')
        return list(map(lambda f: lookup_token(f), tokens))

    def pad_formula(self, formula, max_length):
        '''
        Pads a formula to max_length with pad_token, appending end_token.
        '''
        # Extra space for the end token
        padded_formula = self.token_index[self.pad] * np.ones(max_length + 1)
        padded_formula[len(formula)] = self.token_index[self.end]
        padded_formula[:len(formula)] = formula
        return padded_formula

    @property
    def length(self):
        return len(self.token_index)

vocab = Vocab(f"{processed_data_path}/vocab.txt")
```

1.0.1 Hyperparameters

While a full hyperparameter search was not performed, the below hyperparameters seemed to get reasonable results.

In [46]:
```
buffer_size = 1000
batch_size = 16
embedding_dim = 256
vocab_size = vocab.length
hidden_units = 256
num_datapoints = 35000
num_training_steps = num_datapoints // batch_size
num_validation_datapoints = 8474
num_validation_steps = num_validation_datapoints // batch_size
epochs = 50
train_new_model = False
max_image_size=(80,400)
max_formula_length = 150
```

1.0.2 Loading the data

Tensorflow provides the `tf.data.Dataset` API for doing ETL – loading, transforming and streaming your data to the appropriate compute device. Originally, our implementation used regular numpy based processing, but that had some limitations: - No parallel processing – bound by the performance of a single thread - No streaming operations – all data must be put into memory

Using the `tf.data.Dataset` API had it's own limitations; mainly that it was somewhat tricky to rewrite the operations to be nodes in the Tensorflow graph.

Note that this API also allows you to distribute your data easily to multiple GPUs or TPUs - a fact that will come in handy once we start training the model.

In [47]:
```
# This hash table is used to perform token lookups in the vocab
table = tf.lookup.StaticHashTable(
    initializer=tf.lookup.KeyValueTensorInitializer(
        keys=tf.constant(list(vocab.token_index.keys())),
        values=tf.constant(list(vocab.token_index.values())),
    ),
    default_value=tf.constant(vocab.token_index[vocab.unk]),
    name="class_weight"
)

def load_and_decode_img(path):
    ''' Load the image and decode from png'''
    image = tf.io.read_file(path)
    image = tf.image.decode_png(image)
    return tf.image.rgb_to_grayscale(image)

@tf.function
def lookup_token(token):
    ''' Lookup the given token in the vocab'''
```

```
        table.lookup(token)
        return  table.lookup(token)

    def process_label(label):
        ''' Split to tokens, lookup & append <END> token'''
        tokens = tf.strings.split(label, " ")
        tokens = tf.map_fn(lookup_token, tokens, dtype=tf.int32)
        return tf.concat([tokens, [vocab.token_index[vocab.end]]], 0)

    def process_datum(path, label):
        return load_and_decode_img(path), process_label(label)

    # Tokenize formulas
    train_dataset = tf.data.Dataset.from_tensor_slices((training_images, train_labels)).ma
    validation_dataset = tf.data.Dataset.from_tensor_slices((validation_images, validate_l
```

"Visualize" what data we have before filtering:

```
In [48]: # Print some values from the dataset (pre-filter)
         for datum in train_dataset.take(5):
             print(datum[1])
             print("\n")
```

```
tf.Tensor(
[3 3 1 3 2 3 3 3 3 3 3 1 3 3 3 2 3 1 3 2 3 3 3 1 3 2 3 3 3 3 1 3 1 3 2 1 3
 3 1 3 3 3 2 2 2 3 3 3 5], shape=(49,), dtype=int32)


tf.Tensor([3 3 3 3 3 3 1 3 2 1 3 3 3 1 3 2 2 3 3 1 3 3 2 3 1 3 2 1 3 3 3 1 3 2 2 3 5], shape=(3


tf.Tensor([3 3 1 3 2 3 3 3 3 3 3 3 3 3 3 3 3 5], shape=(18,), dtype=int32)


tf.Tensor([3 3 3 1 3 2 3 3 1 3 2 3 3 3 1 0 2 3 1 3 3 2 3 3 5], shape=(25,), dtype=int32)


tf.Tensor(
[1 3 1 3 3 2 1 3 3 2 2 3 3 1 3 1 3 3 2 1 3 3 1 3 3 1 3 2 2 3 1 0 2 2 2 3 3
 5], shape=(38,), dtype=int32)
```

We want to exclude large images because they will make the training process slower, as well as long formulas. Due to the vanishing gradient problem, long formulas will likely not be predicted very well.

```
In [49]: def filter_by_size(image, label):
             '''Filter the dataset by the size of the image & length of label'''
```

```
        label_length = tf.shape(label)
        image_size = tf.shape(image)

        # Does this image meet our size constraint?
        keep_image = tf.math.reduce_all(
            tf.math.greater_equal(max_image_size, image_size[:2])
        )
        # Does this image meet our formula length constraint?
        keep_label = tf.math.reduce_all(
            tf.math.greater_equal(max_formula_length, label_length[0])
        )
        return tf.math.logical_and(keep_image, keep_label)

    train_dataset = train_dataset.filter(filter_by_size).take(num_datapoints)
    validation_dataset = validation_dataset.filter(filter_by_size).take(num_validation_da
```

```
In [50]: for datum in train_dataset.take(5):
             print(datum[1])
             print("\n")
```

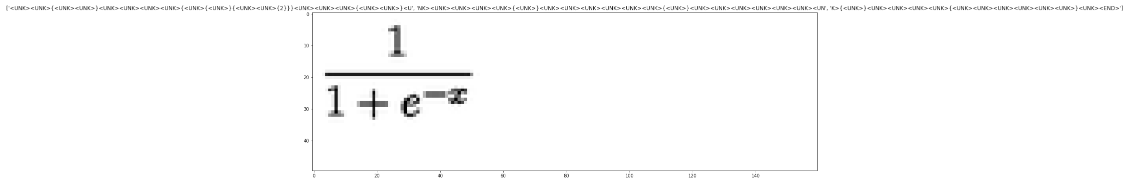
```
tf.Tensor(
[3 3 1 3 2 3 3 3 3 3 3 1 3 3 3 2 3 1 3 2 3 3 3 1 3 2 3 3 3 3 1 3 1 3 2 1 3
 3 1 3 3 3 2 2 2 3 3 3 5], shape=(49,), dtype=int32)


tf.Tensor([3 3 3 3 3 3 1 3 2 1 3 3 3 1 3 2 2 3 3 1 3 3 2 3 1 3 2 1 3 3 3 1 3 2 2 3 5], shape=(3


tf.Tensor([3 3 1 3 2 3 3 3 3 3 3 3 3 3 3 3 3 5], shape=(18,), dtype=int32)


tf.Tensor([3 3 3 1 3 2 3 3 1 3 2 3 3 3 1 0 2 3 1 3 3 2 3 3 5], shape=(25,), dtype=int32)


tf.Tensor(
[1 3 1 3 3 2 1 3 3 2 2 3 3 1 3 1 3 3 2 1 3 3 1 3 3 1 3 2 2 3 1 0 2 2 2 3 3
 5], shape=(38,), dtype=int32)
```

Leverage the power of the `tf.data.Dataset` API to shuffle, batch, pad and cache, all in just a few lines of python code!

```
In [51]: # Shuffle and batch (dropping any batches that are < BATCH_SIZE long)
         # also pad each batch to the largest image size + formulas to
         # max_formula_length.
         shapes = (tf.TensorShape([None,None,1]),tf.TensorShape([max_formula_length]))
         values = (tf.constant(255, dtype=tf.uint8), tf.constant(vocab.token_index[vocab.pad])
```

```
train_dataset = train_dataset.shuffle(buffer_size).padded_batch(
    batch_size,
    padded_shapes=shapes,
    padding_values=values,
    drop_remainder=True
)
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE).cach

validation_dataset = validation_dataset.shuffle(buffer_size).padded_batch(
    batch_size,
    padded_shapes=shapes,
    padding_values=values,
    drop_remainder=True
)

# Prefetch and cache for performance
validation_dataset = validation_dataset.prefetch(buffer_size=tf.data.experimental.AUTO
```

In [52]:
```
import random
import textwrap
import matplotlib.pyplot as plt

def plot_example(example):
    ''' Plot a single example'''
    if example is None:
        return

    image_tensor, label = example

    label = "".join([vocab.reverse_index[token.numpy()] for token in label])
    label = label.replace("<PAD>","")

    fig = plt.figure(figsize=(20,20))
    plt.imshow(image_tensor[:,:,0], cmap='gray')
    plt.title(textwrap.wrap(label,100))
    plt.show()

# Plot a few image from a random batch
for img_tensor, labels in train_dataset.take(5):
    if len(img_tensor.shape) == 4:
        image = img_tensor[0, :, :, :]
        label = labels[0]
        plot_example((image, label))
    else:
        plot_example((img_tensor, labels))
```



1.0.3 Model definition

Below are the components of the network: - The timing signal function – adds sinusoids to the features so that our flattening operations doesn't mean we lose a sense of order - Bahdanau attenntion - The encoder (from the last notebook) - The decoder (using the timing signal function and the attention layer)

```
In [53]: from __future__ import division
         import math
         import numpy as np
         from six.moves import xrange
         import tensorflow as tf


         # taken from https://github.com/tensorflow/tensor2tensor/blob/37465a1759e278e8f073cd0
         def add_timing_signal_nd(x, min_timescale=1.0, max_timescale=1.0e4):
             """Adds a bunch of sinusoids of different frequencies to a Tensor.

             Each channel of the input Tensor is incremented by a sinusoid of a difft
             frequency and phase in one of the positional dimensions.

             This allows attention to learn to use absolute and relative positions.
             Timing signals should be added to some precursors of both the query and the
             memory inputs to attention.

             The use of relative position is possible because sin(a+b) and cos(a+b) can
             be expressed in terms of b, sin(a) and cos(a).

             x is a Tensor with n "positional" dimensions, e.g. one dimension for a
             sequence or two dimensions for an image

             We use a geometric sequence of timescales starting with
             min_timescale and ending with max_timescale.  The number of different
             timescales is equal to channels // (n * 2). For each timescale, we
             generate the two sinusoidal signals sin(timestep/timescale) and
             cos(timestep/timescale).  All of these sinusoids are concatenated in
             the channels dimension.

             Args:
```

```
        x: a Tensor with shape [batch, d1 ... dn, channels]
        min_timescale: a float
        max_timescale: a float

    Returns:
        a Tensor the same shape as x.

    """
    static_shape = x.get_shape().as_list()
    num_dims = len(static_shape) - 2
    channels = tf.shape(x)[-1]
    num_timescales = channels // (num_dims * 2)
    log_timescale_increment = (
            math.log(float(max_timescale) / float(min_timescale)) /
            (tf.cast(num_timescales, tf.float32) - 1))
    inv_timescales = min_timescale * tf.exp(
            tf.cast(tf.range(num_timescales), tf.float32) * -log_timescale_increment)
    for dim in xrange(num_dims):
        length = tf.shape(x)[dim + 1]
        position = tf.cast(tf.range(length), tf.float32)
        scaled_time = tf.expand_dims(position, 1) * tf.expand_dims(
                inv_timescales, 0)
        signal = tf.concat([tf.sin(scaled_time), tf.cos(scaled_time)], axis=1)
        prepad = dim * 2 * num_timescales
        postpad = channels - (dim + 1) * 2 * num_timescales
        signal = tf.pad(signal, [[0, 0], [prepad, postpad]])
        for _ in xrange(1 + dim):
            signal = tf.expand_dims(signal, 0)
        for _ in xrange(num_dims - 1 - dim):
            signal = tf.expand_dims(signal, -2)
        x += signal
    return x
```

In [54]:
```
from tensorflow.keras import metrics, layers, Model

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

    def call(self, features, hidden):
        # First, flatten the image
        shape = tf.shape(features)
        if len(shape) == 4:
            batch_size = shape[0]
            img_height = shape[1]
```

```
            img_width  = shape[2]
            channels   = shape[3]
            features = tf.reshape(features, shape=(batch_size, img_height*img_width, c
        else:
            print(f"Image shape not supported: {shape}.")
            raise NotImplementedError

        # features(CNN_encoder_output)
        # shape => (batch_size, flattened_image_size, embedding_size)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # score
        # shape => (batch_size, flattened_image_size, hidden_size)
        score = tf.nn.tanh(self.W1(features) + self.W2(hidden_with_time_axis))

        # attention_weights
        # shape => (batch_size, flattened_image_size, 1)
        attention_weights = tf.nn.softmax(self.V(score), axis=1)

        # context_vector
        # shape after sum => (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

```
In [55]: class CNNEncoder(tf.keras.Model):
             def __init__(self, embedding_dim):
                 super(CNNEncoder, self).__init__()

                 self.fc = tf.keras.layers.Dense(embedding_dim)

             def build(self, input_shape):
                 self.cnn_1 = layers.Conv2D(64, (3, 3), activation='relu', input_shape=(input_s
                 self.max_pool_1 = layers.MaxPooling2D((2, 2))

                 self.cnn_2 = layers.Conv2D(256, (3, 3), activation='relu')
                 self.max_pool_2 = layers.MaxPooling2D((2, 2))

                 self.cnn_3 = layers.Conv2D(512, (3, 3), activation='relu')

             def call(self, images):
                 images = tf.cast(images, tf.float32)
                 x = self.cnn_1(images)
                 x = self.max_pool_1(x)
```

```
        x = self.cnn_2(x)
        x = self.max_pool_2(x)
        x = self.cnn_3(x)
        return add_timing_signal_nd(x)
```

In [56]:
```
class RNNDecoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNNDecoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(vocab_size)

        self.attention = BahdanauAttention(self.units)


    def call(self, x, features, hidden):
        # attend over the image features
        context_vector, attention_weights = self.attention(features, hidden)

        # convert our input vector to an embedding
        x = self.embedding(x)

        # concat the embedding and the context vector (from attention)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # pass to GRU
        output, state = self.gru(x)

        x = self.fc1(output)

        x = tf.reshape(x, (-1, x.shape[2]))

        # This produces a distribution over the vocab
        x = self.fc2(x)

        return x, state, attention_weights

    def reset_state(self, batch_size):
        return tf.zeros((batch_size, self.units))
```

In [57]:
```
encoder = CNNEncoder(embedding_dim=embedding_dim)
decoder = RNNDecoder(embedding_dim, hidden_units, vocab_size)
```

1.0.4 Loss function

Define a loss function for our model: Sparse categorical cross-entropy is appropriate here since we're making multi-class predictions using integer labels.

```
In [58]: optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
         loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reductio

         def loss_function(real, pred):
             # In order to avoid <PAD> tokens contributing to the loss, we mask those tokens.
             # First, we create the mask, and compute the loss.
             mask = tf.math.logical_not(tf.math.equal(real, vocab.token_index[vocab.pad]))
             loss_ = loss_object(real, pred)

             # Second, we multiply the computed loss by the mask to zero out contributions fro
             mask = tf.cast(mask, dtype=loss_.dtype)
             loss_ *= mask
             return tf.reduce_mean(loss_)
```

Checkpoints allow us to save state during training – a real boon since training will take quite a while.

```
In [59]: checkpoint_path = "./checkpoints/train"
         checkpoint = tf.train.Checkpoint(encoder=encoder,
                                          decoder=decoder,
                                          optimizer=optimizer)
         checkpoint_manager = tf.train.CheckpointManager(checkpoint, checkpoint_path, max_to_ke

         # Attempt to restore from training checkpoint
         start_epoch = 0
         save_at_epoch = 5
         if train_new_model is False and checkpoint_manager.latest_checkpoint:
             checkpoint.restore(checkpoint_manager.latest_checkpoint)
             start_epoch = int(checkpoint_manager.latest_checkpoint.split('-')[-1])*save_at_epo
             print(f"Restored from checkpoint: {checkpoint_manager.latest_checkpoint}.")
             print(f"Start epoch: {start_epoch}.")
         else:
             print("Did not restore from a checkpoint -- training new model!")
```

```
Did not restore from a checkpoint -- training new model!
```

1.0.5 Training

Finally, we define a custom training and validation loop for our model. This is heavily inspired by the Tensorflow example linked at the top the notebook. Notable is the use of teacher forcing (described nit he code comment below).

```
In [73]: train_epoch_losses = []
         validation_epoch_losses = []
```

In [61]:
```
@tf.function
def train_step(img_tensor, target):
    ''' Function that encapsulates training logic'''
    loss = 0

    # reset the decoder state, since Latex is different for each image
    hidden = decoder.reset_state(batch_size=target.shape[0])

    # shape => (batch_size, 1)
    dec_input = tf.expand_dims([vocab.token_index[vocab.start]] * batch_size, 1)

    sequence_length = target.shape[1]
    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        for i in range(0, sequence_length):
            # passing the features through the decoder
            predictions, hidden, _ = decoder(dec_input, features, hidden)

            ground_truth_token = target[:, i]
            loss += loss_function(ground_truth_token, predictions)

            # Teacher forcing: feed the correct word in as the next input to the
            # encoder, to provide the decoder with the proper context to predict
            # the following token in the sequence
            dec_input = tf.expand_dims(ground_truth_token, 1)

    total_loss = (loss / int(sequence_length))
    trainable_variables = encoder.trainable_variables + decoder.trainable_variables
    gradients = tape.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))
    return loss, total_loss
```

In [62]:
```
@tf.function
def validate_step(img_tensor, target):
    ''' Function that encapsulates training logic'''
    loss = 0

    # reset the decoder state, since Latex is different for each image
    hidden = decoder.reset_state(batch_size=target.shape[0])
    # shape => (batch_size, 1)
    dec_input = tf.expand_dims([vocab.token_index[vocab.start]] * batch_size, 1)

    sequence_length = target.shape[1]
    features = encoder(img_tensor)
    for i in range(0, sequence_length):
        # passing the features through the decoder
        predictions, hidden, _ = decoder(dec_input, features, hidden)
```

```
            ground_truth_token = target[:, i]
            loss += loss_function(ground_truth_token, predictions)
            dec_input = tf.expand_dims(ground_truth_token, 1)

        total_loss = (loss / int(sequence_length))
        return loss, total_loss
```

In [74]:
```
import time
import datetime

print(f"[Started training at: {datetime.datetime.now()}. Training for {epochs} epochs
print(f"[Starting epoch: {start_epoch}]")
for epoch in range(start_epoch, epochs):
    start = time.time()
    train_loss = 0
    validation_loss = 0

    # Training loop
    train_batches = 0
    for (batch, (img_tensor, target)) in enumerate(train_dataset):
        batch_loss, sequence_loss = train_step(img_tensor, target)
        train_loss += sequence_loss
        train_batches = batch
        if batch % 50 == 0:
            print(f"[Epoch: {epoch + 1} | Batch: {batch} | Loss: {sequence_loss:.4f}]"

    # Print training loss before starting validation loop
    print(f"[Epoch: {epoch + 1} | Training loss: {train_loss / (train_batches + 1)}]")

    # Validation loop
    validation_batches = 0
    for (batch, (img_tensor, target)) in enumerate(validation_dataset):
        batch_loss, sequence_loss = validate_step(img_tensor, target)
        validation_loss += sequence_loss
        validation_batches = batch

    print(f"[Epoch: {epoch + 1} | Validation loss: {validation_loss / (validation_bat

    # Save epoch losses
    train_epoch_losses.append(train_loss / (train_batches + 1))
    validation_epoch_losses.append(validation_loss / (validation_batches + 1))

    # Save checkpoint (if required)
    if epoch % save_at_epoch == 0:
        checkpoint_manager.save()

    print(f"[Time elapsed for epoch: {format(time.time() - start)} seconds.] \n")
```

```
[Started training at: 2019-08-04 16:30:50.073180. Training for 50 epochs.]
[Starting epoch: 0]
[Epoch: 1 | Batch: 0 | Loss: 0.4328]
[Epoch: 1 | Training loss: 0.43280482292175293]
[Epoch: 1 | Validation loss: 0.3729066848754883]
[Time elapsed for epoch: 6.088000059127808 seconds.]

[Epoch: 2 | Batch: 0 | Loss: 0.4337]
[Epoch: 2 | Training loss: 0.433720201253891]
[Epoch: 2 | Validation loss: 0.37391233444213867]
[Time elapsed for epoch: 6.141319751739502 seconds.]

[Epoch: 3 | Batch: 0 | Loss: 0.4323]
[Epoch: 3 | Training loss: 0.43230941891670227]
[Epoch: 3 | Validation loss: 0.3752366602420807]
[Time elapsed for epoch: 6.029435873031616 seconds.]

[Epoch: 4 | Batch: 0 | Loss: 0.4321]
[Epoch: 4 | Training loss: 0.4320918321609497]
[Epoch: 4 | Validation loss: 0.374604731798172]
[Time elapsed for epoch: 6.068495988845825 seconds.]

[Epoch: 5 | Batch: 0 | Loss: 0.4319]
[Epoch: 5 | Training loss: 0.4319472908973694]
[Epoch: 5 | Validation loss: 0.3743206262588501]
[Time elapsed for epoch: 5.908013105392456 seconds.]

[Epoch: 6 | Batch: 0 | Loss: 0.4318]
[Epoch: 6 | Training loss: 0.4317789375782013]
[Epoch: 6 | Validation loss: 0.37626776099205017]
[Time elapsed for epoch: 6.077317714691162 seconds.]

[Epoch: 7 | Batch: 0 | Loss: 0.4315]
[Epoch: 7 | Training loss: 0.4315233826637268]
[Epoch: 7 | Validation loss: 0.3764217495918274]
[Time elapsed for epoch: 5.957525014877319 seconds.]

[Epoch: 8 | Batch: 0 | Loss: 0.4312]
[Epoch: 8 | Training loss: 0.43122947216033936]
[Epoch: 8 | Validation loss: 0.3756418526172638]
[Time elapsed for epoch: 5.928221940994263 seconds.]

[Epoch: 9 | Batch: 0 | Loss: 0.4308]
[Epoch: 9 | Training loss: 0.43078044056892395]
[Epoch: 9 | Validation loss: 0.3763998746871948]
[Time elapsed for epoch: 6.061295747756958 seconds.]

[Epoch: 10 | Batch: 0 | Loss: 0.4309]
```

```
[Epoch: 10 | Training loss: 0.4309038519859314]
[Epoch: 10 | Validation loss: 0.37639573216438293]
[Time elapsed for epoch: 6.174771308898926 seconds.]

[Epoch: 11 | Batch: 0 | Loss: 0.4301]
[Epoch: 11 | Training loss: 0.4301297664642334]
[Epoch: 11 | Validation loss: 0.37671521306037903]
[Time elapsed for epoch: 6.158113956451416 seconds.]

[Epoch: 12 | Batch: 0 | Loss: 0.4301]
[Epoch: 12 | Training loss: 0.43014100193977356]
[Epoch: 12 | Validation loss: 0.37675824761390686]
[Time elapsed for epoch: 6.271286249160767 seconds.]

[Epoch: 13 | Batch: 0 | Loss: 0.4300]
[Epoch: 13 | Training loss: 0.42999178171157837]
[Epoch: 13 | Validation loss: 0.376176118850708]
[Time elapsed for epoch: 6.631333827972412 seconds.]

[Epoch: 14 | Batch: 0 | Loss: 0.4296]
[Epoch: 14 | Training loss: 0.4296094477176666]
[Epoch: 14 | Validation loss: 0.3766115605831146]
[Time elapsed for epoch: 6.179867267608643 seconds.]

[Epoch: 15 | Batch: 0 | Loss: 0.4298]
[Epoch: 15 | Training loss: 0.4297952353954315]
[Epoch: 15 | Validation loss: 0.3769102990627289]
[Time elapsed for epoch: 7.319034099578857 seconds.]

[Epoch: 16 | Batch: 0 | Loss: 0.4292]
[Epoch: 16 | Training loss: 0.4291546642780304]
[Epoch: 16 | Validation loss: 0.3776816129684448]
[Time elapsed for epoch: 8.729957103729248 seconds.]

[Epoch: 17 | Batch: 0 | Loss: 0.4291]
[Epoch: 17 | Training loss: 0.429097443819046]
[Epoch: 17 | Validation loss: 0.37767019867897034]
[Time elapsed for epoch: 8.920700788497925 seconds.]

[Epoch: 18 | Batch: 0 | Loss: 0.4290]
[Epoch: 18 | Training loss: 0.4289799630641937]
[Epoch: 18 | Validation loss: 0.3759458363056183]
[Time elapsed for epoch: 9.11909008026123 seconds.]

[Epoch: 19 | Batch: 0 | Loss: 0.4297]
[Epoch: 19 | Training loss: 0.42968228459358215]
[Epoch: 19 | Validation loss: 0.3839322626590729]
[Time elapsed for epoch: 9.045357942581177 seconds.]
```

```
[Epoch: 20 | Batch: 0 | Loss: 0.4345]
[Epoch: 20 | Training loss: 0.43449097871780396]
[Epoch: 20 | Validation loss: 0.37716159224510193]
[Time elapsed for epoch: 8.704407930374146 seconds.]

[Epoch: 21 | Batch: 0 | Loss: 0.4323]
[Epoch: 21 | Training loss: 0.4323433041572571]
[Epoch: 21 | Validation loss: 0.3760915696620941]
[Time elapsed for epoch: 8.484700918197632 seconds.]

[Epoch: 22 | Batch: 0 | Loss: 0.4305]
[Epoch: 22 | Training loss: 0.430453896522522]
[Epoch: 22 | Validation loss: 0.37813010811805725]
[Time elapsed for epoch: 7.396198034286499 seconds.]

[Epoch: 23 | Batch: 0 | Loss: 0.4312]
[Epoch: 23 | Training loss: 0.43124625086784363]
[Epoch: 23 | Validation loss: 0.37761902809143066]
[Time elapsed for epoch: 6.945310354232788 seconds.]

[Epoch: 24 | Batch: 0 | Loss: 0.4300]
[Epoch: 24 | Training loss: 0.4300049841403961]
[Epoch: 24 | Validation loss: 0.37535855174064636]
[Time elapsed for epoch: 7.035900354385376 seconds.]

[Epoch: 25 | Batch: 0 | Loss: 0.4305]
[Epoch: 25 | Training loss: 0.43047434091567993]
[Epoch: 25 | Validation loss: 0.3757206201553345]
[Time elapsed for epoch: 7.86501407623291 seconds.]

[Epoch: 26 | Batch: 0 | Loss: 0.4297]
[Epoch: 26 | Training loss: 0.42966702580451965]
[Epoch: 26 | Validation loss: 0.3771822154521942]
[Time elapsed for epoch: 7.487479209899902 seconds.]

[Epoch: 27 | Batch: 0 | Loss: 0.4295]
[Epoch: 27 | Training loss: 0.4295448362827301]
[Epoch: 27 | Validation loss: 0.3771308958530426]
[Time elapsed for epoch: 7.531407117843628 seconds.]

[Epoch: 28 | Batch: 0 | Loss: 0.4291]
[Epoch: 28 | Training loss: 0.42905667424201965]
[Epoch: 28 | Validation loss: 0.37456297874450684]
[Time elapsed for epoch: 7.878756761550903 seconds.]

[Epoch: 29 | Batch: 0 | Loss: 0.4294]
[Epoch: 29 | Training loss: 0.42941874265670776]
```

```
[Epoch: 29 | Validation loss: 0.37630295753479004]
[Time elapsed for epoch: 8.038648843765259 seconds.]

[Epoch: 30 | Batch: 0 | Loss: 0.4280]
[Epoch: 30 | Training loss: 0.42800599336624146]
[Epoch: 30 | Validation loss: 0.3784244954586029]
[Time elapsed for epoch: 7.902118921279907 seconds.]

[Epoch: 31 | Batch: 0 | Loss: 0.4289]
[Epoch: 31 | Training loss: 0.4289153516292572]
[Epoch: 31 | Validation loss: 0.3763226568698883]
[Time elapsed for epoch: 7.8783040046691895 seconds.]

[Epoch: 32 | Batch: 0 | Loss: 0.4276]
[Epoch: 32 | Training loss: 0.4275738000869751]
[Epoch: 32 | Validation loss: 0.37549081444740295]
[Time elapsed for epoch: 7.289207935333252 seconds.]

[Epoch: 33 | Batch: 0 | Loss: 0.4282]
[Epoch: 33 | Training loss: 0.42817196249961853]
[Epoch: 33 | Validation loss: 0.3797770142555237]
[Time elapsed for epoch: 7.223494291305542 seconds.]

[Epoch: 34 | Batch: 0 | Loss: 0.4299]
[Epoch: 34 | Training loss: 0.4299106001853943]
[Epoch: 34 | Validation loss: 0.3763534128665924]
[Time elapsed for epoch: 7.393735885620117 seconds.]

[Epoch: 35 | Batch: 0 | Loss: 0.4282]
[Epoch: 35 | Training loss: 0.4282092750072479]
[Epoch: 35 | Validation loss: 0.3770476281642914]
[Time elapsed for epoch: 7.576486110687256 seconds.]

[Epoch: 36 | Batch: 0 | Loss: 0.4295]
[Epoch: 36 | Training loss: 0.42953768372535706]
[Epoch: 36 | Validation loss: 0.3779384195804596]
[Time elapsed for epoch: 7.850259065628052 seconds.]

[Epoch: 37 | Batch: 0 | Loss: 0.4279]
[Epoch: 37 | Training loss: 0.42790350317955017]
[Epoch: 37 | Validation loss: 0.37824326753616333]
[Time elapsed for epoch: 7.939767122268677 seconds.]

[Epoch: 38 | Batch: 0 | Loss: 0.4285]
[Epoch: 38 | Training loss: 0.42851522564888]
[Epoch: 38 | Validation loss: 0.37616968154907227]
[Time elapsed for epoch: 7.516835689544678 seconds.]
```

```
[Epoch: 39 | Batch: 0 | Loss: 0.4276]
[Epoch: 39 | Training loss: 0.42755377292633057]
[Epoch: 39 | Validation loss: 0.37854263186454773]
[Time elapsed for epoch: 7.407878160476685 seconds.]

[Epoch: 40 | Batch: 0 | Loss: 0.4280]
[Epoch: 40 | Training loss: 0.42799124121665955]
[Epoch: 40 | Validation loss: 0.3787980079650879]
[Time elapsed for epoch: 7.327160120010376 seconds.]

[Epoch: 41 | Batch: 0 | Loss: 0.4273]
[Epoch: 41 | Training loss: 0.4273271858692169]
[Epoch: 41 | Validation loss: 0.37788906693458557]
[Time elapsed for epoch: 7.549849271774292 seconds.]

[Epoch: 42 | Batch: 0 | Loss: 0.4272]
[Epoch: 42 | Training loss: 0.4272112250328064]
[Epoch: 42 | Validation loss: 0.37594982981681824]
[Time elapsed for epoch: 7.200824975967407 seconds.]

[Epoch: 43 | Batch: 0 | Loss: 0.4269]
[Epoch: 43 | Training loss: 0.42693156003952026]
[Epoch: 43 | Validation loss: 0.3782337009906769]
[Time elapsed for epoch: 7.465975761413574 seconds.]

[Epoch: 44 | Batch: 0 | Loss: 0.4268]
[Epoch: 44 | Training loss: 0.426753968000412]
[Epoch: 44 | Validation loss: 0.38179242610931396]
[Time elapsed for epoch: 7.403193950653076 seconds.]

[Epoch: 45 | Batch: 0 | Loss: 0.4285]
[Epoch: 45 | Training loss: 0.42850354313850403]
[Epoch: 45 | Validation loss: 0.37620317935943604]
[Time elapsed for epoch: 7.5887370109558105 seconds.]

[Epoch: 46 | Batch: 0 | Loss: 0.4271]
[Epoch: 46 | Training loss: 0.42706650495529175]
[Epoch: 46 | Validation loss: 0.3790723383426666]
[Time elapsed for epoch: 7.768675088882446 seconds.]

[Epoch: 47 | Batch: 0 | Loss: 0.4266]
[Epoch: 47 | Training loss: 0.42657536268234253]
[Epoch: 47 | Validation loss: 0.3783749043941498]
[Time elapsed for epoch: 7.7456676959991455 seconds.]

[Epoch: 48 | Batch: 0 | Loss: 0.4257]
[Epoch: 48 | Training loss: 0.42569392919540405]
[Epoch: 48 | Validation loss: 0.3764776587486267]
```

```
[Time elapsed for epoch: 8.05068325996399 seconds.]

[Epoch: 49 | Batch: 0 | Loss: 0.4262]
[Epoch: 49 | Training loss: 0.42618921399116516]
[Epoch: 49 | Validation loss: 0.3764073848724365]
[Time elapsed for epoch: 8.146775960922241 seconds.]

[Epoch: 50 | Batch: 0 | Loss: 0.4252]
[Epoch: 50 | Training loss: 0.42520302534103394]
[Epoch: 50 | Validation loss: 0.3773685693740845]
[Time elapsed for epoch: 8.710307121276855 seconds.]
```

1.0.6 Results

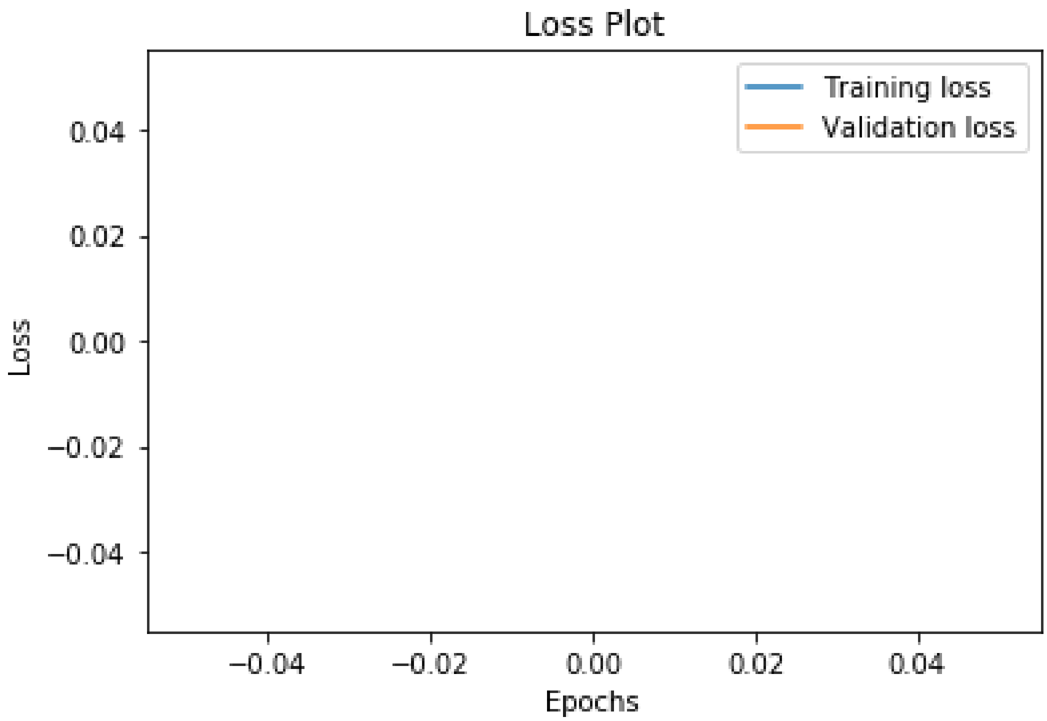
Below, we plot some examples taken from the test set, make a prediction and attempt to render the latex back into an image. Subjectively, we find that of the latex is invalid; a bracket is missing, or a character is in an invalid position. Nonetheless, when trained with enough data, the predictions often have significant overlap with the ground truth label. This is the nature of deep learning models – without a set of rules to govern the output, it all must be learned from data.

```
In [71]: validation_epoch_losses
```

```
Out[71]: [<tf.Tensor: id=430131, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430251, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430284, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430317, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430350, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430383, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430496, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430529, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430571, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430684, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430717, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430750, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430783, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430816, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430929, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430962, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=430995, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431028, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431061, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431174, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431207, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431240, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431273, shape=(), dtype=float32, numpy=inf>,
          <tf.Tensor: id=431306, shape=(), dtype=float32, numpy=inf>,
```

```
 <tf.Tensor: id=431419, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431452, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431485, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431518, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431551, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431664, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431697, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431730, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431763, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431796, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431909, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431942, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=431975, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432008, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432041, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432154, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432187, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432220, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432253, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432286, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432399, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432432, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432465, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432498, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432531, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432644, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432677, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432710, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432743, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432776, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432889, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432922, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432955, shape=(), dtype=float32, numpy=inf>,
 <tf.Tensor: id=432988, shape=(), dtype=float32, numpy=inf>]
```

In [69]:
```python
import matplotlib.pyplot as plt

plt.plot(train_epoch_losses)
plt.plot(validation_epoch_losses)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.legend(["Training loss", "Validation loss"])
plt.savefig(f"{figs_base_dir}/{datetime.datetime.now()}.png")
plt.show()
```



In [154]:
```python
import matplotlib.pyplot as plt

def load_image(path):
    img_raw = tf.io.read_file(path)
    return tf.image.decode_png(img_raw)


def plot_features(features):
    layer_count = features.shape[3]

    num_rows = 16
    num_cols = int(layer_count / num_rows)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(32, 32))
    for i, ax in enumerate(axes.flat):
        feat = features[0,:,:,i]
        ax.imshow(feat)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

def plot_attention(attn):
    layer_count = attn.shape[0]
```

```
        num_rows = 16
        num_cols = math.floor(layer_count / num_rows)

        fig, axes = plt.subplots(num_rows, num_cols, figsize=(32, 20))
        for i, ax in enumerate(axes.flat):
            if i >= layer_count:
                break
            feat = attn[i,:,:]
            ax.imshow(feat)
            ax.axis('off')
        plt.tight_layout()
        plt.show()

    def evaluate(img, max_formula_length):
        index_token_mapping = {v: k for k, v in vocab.token_index.items()}

        hidden = decoder.reset_state(batch_size=1)

        # Convert to grayscale so network can process it
        # and add a dimension at the start to simulate batch (size=1)
        image = tf.image.rgb_to_grayscale(img)
        temp_input = tf.expand_dims(image, 0)

        # Pass through encoder to obtain features
        features = encoder(temp_input)

        # Signal to the decoder that we are starting feeding in a new sequence
        decoder_input = tf.expand_dims(vocab.tokenize_formula(vocab.start), 1)

        result = []
        attention_plot = np.zeros((max_formula_length, features.shape[1], features.shape
        for i in range(max_formula_length):
            predictions, hidden, attention_weights = decoder(decoder_input, features, hid

            #attn weights: (1, feat_width * feat_height, 1)
            attention_plot[i] = tf.reshape(attention_weights, (features.shape[1], feature

            predicted_id = tf.argmax(predictions[0]).numpy()
            result.append(index_token_mapping[predicted_id])

            if index_token_mapping[predicted_id] == vocab.end:
                # Strip end token when returning
                return result[:-1], attention_plot, features

            decoder_input = tf.expand_dims([predicted_id], 0)
        return result, attention_plot, features
```

In [ ]:
```
import subprocess
```

```python
def create_from_template(latex):
    '''Uses a simple template to create a valid latex document'''
    pre = """
    \documentclass[preview]{standalone}
    \\begin{document}
    \\begin{equation}
    """

    post = """
    \end{equation}
    \end{document}
    """
    return pre + latex + post


def render_latex(latex):
    ''' Renders a latex string, creating a png'''
    # Write temp tex file
    latex_out_dir = f"{processed_data_path}latex/"
    temp_tex_filepath = f"{processed_data_path}latex/tmp.tex"
    temp_dvi_filepath = os.path.splitext(temp_tex_filepath)[0]+'.dvi'
    temp_png_filepath = os.path.splitext(temp_tex_filepath)[0]+'1.png'

    with open(temp_tex_filepath, 'w+') as f:
        f.seek(0)
        f.write(create_from_template(latex))

    # Render it
    cmd = ['latex', '-interaction=nonstopmode', '--halt-on-error', "-output-directory"
    output = subprocess.run(cmd, capture_output=True)
    if output.returncode != 0:
        print(f"Could not render latex (status code={output.returncode}. Output follows
        print(output.stdout)
        print(output.stderr)
        print("\n")
        return None

    render_png_cmd = ['dvipng','-D','300', "-o", temp_png_filepath, temp_dvi_filepath]
    output = subprocess.run(render_png_cmd, capture_output=True)
    if output.returncode != 0:
        print("Could not render .dvi file into .pdf! Output follows: \n")
        print(output.stdout)
        print(output.stderr)
        print("\n")
        return None
    return temp_png_filepath
```

```python
def plot_image_clean(png):
    ''' plots an image without axes'''
    fig = plt.figure(figsize=(20,10))
    ax = fig.add_axes([0, 0, 1, 1])
    ax.imshow(plt.imread(out_file))
    ax.axis('off')
    plt.show()

def process_latex(latex):
    return " ".join(latex)

# Example params
test_index = 5
test_img = load_image(validation_images[test_index])

# Make prediction
result, attention_plot, features = evaluate(test_img, max_formula_length)

# Plot original image
plt.imshow(test_img)
plt.show()

# Render latex from the predicted formula
latex = process_latex(result)
out_file = render_latex(latex)
if out_file is not None:
    plot_image_clean(out_file)

print(f"Predicted formula: \n {result}")
plot_attention(attention_plot)
plot_features(features[:,:,:,0:64])
```

In [ ]:
```python
def search_for_valid_predictions(count, example_count, train_image_path = f"{processed
    ''' Search for predictions with latex that compiles'''
    success_count = 0
    index = 0
    success_indexes = []
    while success_count < count:

        if index >= example_count:
            return success_indexes

        # predict
        test_img = load_image(f"{train_image_path}/{index}.png")
        result, attention_plot, features = evaluate(test_img, max_formula_length)

        # attemp to render
        latex = process_latex(result)
```

```python
        out_file = render_latex(latex)

        if out_file is not None:
            success_indexes.append(index)

        index += 1

search_for_valid_predictions(10, 200)
```