Language Translation

In this project, you're going to take a peek into the realm of neural network machine translation. You'll be training a sequence to sequence model on a dataset of English and French sentences that can translate new sentences from English to French.

Get the Data ¶

Since translating the whole language of English to French will take lots of time to train, we have provided you with a small portion of the English corpus.

In [1]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper
import problem_unittests as tests

source_path = 'data/small_vocab_en'
target_path = 'data/small_vocab_fr'
source_text = helper.load_data(source_path)
target_text = helper.load_data(target_path)
```

Explore the Data

Play around with view_sentence_range to view different parts of the data.

In [2]:
```
view_sentence_range = (0, 10)

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np

print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None fo
r word in source_text.split()})))

sentences = source_text.split('\n')
word_counts = [len(sentence.split()) for sentence in sentences]
print('Number of sentences: {}'.format(len(sentences)))
print('Average number of words in a sentence: {}'.format(np.average(word
_counts)))

print()
print('English sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(source_text.split('\n')[view_sentence_range[0]:view_sent
ence_range[1]]))
print()
print('French sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(target_text.split('\n')[view_sentence_range[0]:view_sent
ence_range[1]]))
```

```
Dataset Stats
Roughly the number of unique words: 227
Number of sentences: 137861
Average number of words in a sentence: 13.225277634719028

English sentences 0 to 10:
new jersey is sometimes quiet during autumn , and it is snowy in april
.
the united states is usually chilly during july , and it is usually fre
ezing in november .
california is usually quiet during march , and it is usually hot in jun
e .
the united states is sometimes mild during june , and it is cold in sep
tember .
your least liked fruit is the grape , but my least liked is the apple .
his favorite fruit is the orange , but my favorite is the grape .
paris is relaxing during december , but it is usually chilly in july .
new jersey is busy during spring , and it is never hot in march .
our least liked fruit is the lemon , but my least liked is the grape .
the united states is sometimes busy during january , and it is sometime
s warm in november .

French sentences 0 to 10:
new jersey est parfois calme pendant l' automne , et il est neigeux en
avril .
les états-unis est généralement froid en juillet , et il gèle habituell
ement en novembre .
california est généralement calme en mars , et il est généralement chau
d en juin .
les états-unis est parfois légère en juin , et il fait froid en septemb
re .
votre moins aimé fruit est le raisin , mais mon moins aimé est la pomme
.
son fruit préféré est l'orange , mais mon préféré est le raisin .
paris est relaxant en décembre , mais il est généralement froid en juil
let .
new jersey est occupé au printemps , et il est jamais chaude en mars .
notre fruit est moins aimé le citron , mais mon moins aimé est le raisi
n .
les états-unis est parfois occupé en janvier , et il est parfois chaud
en novembre .
```

Implement Preprocessing Function

Text to Word Ids

As you did with other RNNs, you must turn the text into a number so the computer can understand it. In the function `text_to_ids()`, you'll turn `source_text` and `target_text` from words to ids. However, you need to add the `<EOS>` word id at the end of `target_text`. This will help the neural network predict when the sentence should end.

You can get the `<EOS>` word id by doing:

```
target_vocab_to_int['<EOS>']
```

You can get other word ids using `source_vocab_to_int` and `target_vocab_to_int`.

In [3]:
```
def text_to_ids(source_text, target_text, source_vocab_to_int, target_vo
cab_to_int):
    """
    Convert source and target text to proper word ids
    :param source_text: String that contains all the source text.
    :param target_text: String that contains all the target text.
    :param source_vocab_to_int: Dictionary to go from the source words t
o an id
    :param target_vocab_to_int: Dictionary to go from the target words t
o an id
    :return: A tuple of lists (source_id_text, target_id_text)
    """
    # TODO: Implement Function

    # source
    source_id_text = []
    source_sentences = source_text.split('\n')
    for source_sentence in source_sentences:
       source_words = [ word for word in source_sentence.split(' ') if w
ord ]
       source_id_text.append( [ source_vocab_to_int[source_words[k]]  fo
r k in range(len(source_words)) ] )

    #target
    target_id_text = []
    target_sentences = target_text.split('\n')
    for target_sentence in target_sentences:
       target_words = [ word for word in target_sentence.split(' ') if w
ord] + ['<EOS>']
       target_id_text.append( [ target_vocab_to_int[target_words[k]]  fo
r k in range(len(target_words)) ] )


    return source_id_text, target_id_text

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_text_to_ids(text_to_ids)
```

```
Tests Passed
```

Preprocess all the data and save it

Running the code cell below will preprocess all the data and save it to file.

In [4]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
helper.preprocess_and_save_data(source_path, target_path, text_to_ids)
```

Check Point

This is your first checkpoint. If you ever decide to come back to this notebook or have to restart the notebook, you can start from here. The preprocessed data has been saved to disk.

In [5]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np
import helper
import problem_unittests as tests

(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_t
o_int), _ = helper.load_preprocess()
```

Check the Version of TensorFlow and Access to GPU

This will check to make sure you have the correct version of TensorFlow and access to a GPU

In [6]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
from distutils.version import LooseVersion
import warnings
import tensorflow as tf
from tensorflow.python.layers.core import Dense

# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.1'), 'Please use
 TensorFlow version 1.1 or newer'
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please use a GPU to train your neural n
etwork.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.3.0
Default GPU Device: /gpu:0
```

Build the Neural Network

You'll build the components necessary to build a Sequence-to-Sequence model by implementing the following functions below:

- `model_inputs`
- `process_decoder_input`
- `encoding_layer`
- `decoding_layer_train`
- `decoding_layer_infer`
- `decoding_layer`
- `seq2seq_model`

Input

Implement the `model_inputs()` function to create TF Placeholders for the Neural Network. It should create the following placeholders:

- Input text placeholder named "input" using the TF Placeholder name parameter with rank 2.
- Targets placeholder with rank 2.
- Learning rate placeholder with rank 0.
- Keep probability placeholder named "keep_prob" using the TF Placeholder name parameter with rank 0.
- Target sequence length placeholder named "target_sequence_length" with rank 1
- Max target sequence length tensor named "max_target_len" getting its value from applying tf.reduce_max on the target_sequence_length placeholder. Rank 0.
- Source sequence length placeholder named "source_sequence_length" with rank 1

Return the placeholders in the following the tuple (input, targets, learning rate, keep probability, target sequence length, max target sequence length, source sequence length)

In [7]:
```
def model_inputs():
    """
    Create TF Placeholders for input, targets, learning rate, and length
s of source and target sequences.
    :return: Tuple (input, targets, learning rate, keep probability, tar
get sequence length,
    max target sequence length, source sequence length)
    """
    # TODO: Implement Function
    input   = tf.placeholder(  tf.int32, shape = [None,None], name = 'in
put' )
    targets = tf.placeholder(  tf.int32, shape = [None,None], name = 'ta
rgets' )
    learning_rate = tf.placeholder( tf.float32, name = 'learning_rate',
shape=() )
    keep_prob = tf.placeholder( tf.float32, name = 'keep_prob', shape =
() )
    target_sequence_length = tf.placeholder(  tf.int32, shape = [None],
name = 'target_sequence_length' )
    max_target_len = tf.reduce_max(target_sequence_length)
    source_sequence_length = tf.placeholder(  tf.int32, shape = [None],
name = 'source_sequence_length' )

    return input, targets, learning_rate, keep_prob, target_sequence_len
gth, max_target_len, source_sequence_length


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_inputs(model_inputs)
```

```
Tests Passed
```

Process Decoder Input

Implement `process_decoder_input` by removing the last word id from each batch in `target_data` and concat the GO ID to the begining of each batch.

In [8]:
```
def process_decoder_input(target_data, target_vocab_to_int, batch_size):
    """
    Preprocess target data for encoding
    :param target_data: Target Placehoder
    :param target_vocab_to_int: Dictionary to go from the target words t
o an id
    :param batch_size: Batch Size
    :return: Preprocessed target data
    """
    # TODO: Implement Function

    L = target_data.shape[1]
    target_data_preproc =  tf.concat( [ [target_vocab_to_int['<GO>'] ],
target_data[0,0:L-1]] , 0 )

    for k in range(1,batch_size):
        batch =  tf.concat( [ [target_vocab_to_int['<GO>'] ], target_dat
a[k,0:L-1]] , 0 )
        target_data_preproc = tf.concat( [ target_data_preproc, batch ],
0  )

    target_data_preproc = tf.reshape( target_data_preproc, [batch_size,
-1]  )

    return target_data_preproc

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_process_encoding_input(process_decoder_input)
```

```
Tests Passed
```

Encoding

Implement `encoding_layer()` to create a Encoder RNN layer:

- Embed the encoder input using `tf.contrib.layers.embed_sequence` (https://www.tensorflow.org/api_docs/python/tf/contrib/layers/embed_sequence)
- Construct a stacked (https://github.com/tensorflow/tensorflow/blob/6947f65a374ebf29e74bb71e36fd82760056d82c/tensorflow/d multiple-lstms) `tf.contrib.rnn.LSTMCell` (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/LSTMCell) wrapped in a `tf.contrib.rnn.DropoutWrapper` (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/DropoutWrapper)
- Pass cell and embedded input to `tf.nn.dynamic_rnn()` (https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn)

In [9]:
```
from imp import reload
reload(tests)

def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob,
                   source_sequence_length, source_vocab_size,
                   encoding_embedding_size):
    """
    Create encoding layer
    :param rnn_inputs: Inputs for the RNN
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param keep_prob: Dropout keep probability
    :param source_sequence_length: a list of the lengths of each sequenc
e in the batch
    :param source_vocab_size: vocabulary size of source data
    :param encoding_embedding_size: embedding size of source data
    :return: tuple (RNN output, RNN state)
    """
    # TODO: Implement Function
    embedded_input = tf.contrib.layers.embed_sequence(rnn_inputs, source
_vocab_size, encoding_embedding_size)

    # stacked lstm = multicell
    def lstm_cell():
       lstm = tf.contrib.rnn.LSTMCell(rnn_size, initializer=tf.random_un
iform_initializer(-0.1, 0.1, seed=2))
       drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_
prob)
       return drop

    stacked_lstm = tf.contrib.rnn.MultiRNNCell(
      [lstm_cell() for _ in range(num_layers)])


    # run the multicell
    rnn_output, rnn_state = tf.nn.dynamic_rnn(stacked_lstm, embedded_inp
ut, source_sequence_length, dtype=tf.float32 )

    return rnn_output, rnn_state

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_encoding_layer(encoding_layer)
```

```
Tests Passed
```

Decoding - Training

Create a training decoding layer:

- Create a `tf.contrib.seq2seq.TrainingHelper` (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/TrainingHelper)
- Create a `tf.contrib.seq2seq.BasicDecoder` (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BasicDecoder)
- Obtain the decoder outputs from `tf.contrib.seq2seq.dynamic_decode` (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/dynamic_decode)

In [10]:
```
def decoding_layer_train(encoder_state, dec_cell, dec_embed_input,
                         target_sequence_length, max_summary_length,
                         output_layer, keep_prob):
    """
    Create a decoding layer for training
    :param encoder_state: Encoder State
    :param dec_cell: Decoder RNN Cell
    :param dec_embed_input: Decoder embedded input
    :param target_sequence_length: The lengths of each sequence in the t
arget batch
    :param max_summary_length: The length of the longest sequence in the
batch
    :param output_layer: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: BasicDecoderOutput containing training logits and sample_id
    """
    # TODO: Implement Function

    # setup training helper for decoder
    training_helper = tf.contrib.seq2seq.TrainingHelper(inputs = dec_emb
ed_input,
                                                         sequence_length
= target_sequence_length,
                                                         time_major = Fal
se )

    # regularize output layer
    dec_cell_wrap = tf.contrib.rnn.DropoutWrapper(dec_cell, output_keep_
prob=keep_prob)


    # setup decoder
    basic_decoder =  tf.contrib.seq2seq.BasicDecoder(dec_cell_wrap,train
ing_helper, encoder_state, output_layer)


    # decode
    #(final_outputs, final_state, final_sequence_lengths)
    decoder_output = tf.contrib.seq2seq.dynamic_decode(basic_decoder, im
pute_finished=True,
                                                        maximum_iteration
s=max_summary_length)

    basic_decoder_output = decoder_output[0]   # final_outputs is type B
asicDecoderOutput
    return basic_decoder_output



"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_train(decoding_layer_train)
```

```
Tests Passed
```

Decoding - Inference

Create inference decoder:

- Create a [`tf.contrib.seq2seq.GreedyEmbeddingHelper`](https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/GreedyEmbeddingHelper) (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/GreedyEmbeddingHelper)
- Create a [`tf.contrib.seq2seq.BasicDecoder`](https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BasicDecoder) (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BasicDecoder)
- Obtain the decoder outputs from [`tf.contrib.seq2seq.dynamic_decode`](https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/dynamic_decode) (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/dynamic_decode)

In [11]:
```
def decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_
of_sequence_id,
                         end_of_sequence_id, max_target_sequence_length,
                         vocab_size, output_layer, batch_size, keep_prob
):
    """
    Create a decoding layer for inference
    :param encoder_state: Encoder state
    :param dec_cell: Decoder RNN Cell
    :param dec_embeddings: Decoder embeddings
    :param start_of_sequence_id: GO ID
    :param end_of_sequence_id: EOS Id
    :param max_target_sequence_length: Maximum length of target sequence
s
    :param vocab_size: Size of decoder/target vocabulary
    :param decoding_scope: TenorFlow Variable Scope for decoding
    :param output_layer: Function to apply the output layer
    :param batch_size: Batch size
    :param keep_prob: Dropout keep probability
    :return: BasicDecoderOutput containing inference logits and sample_i
d
    """
    # TODO: Implement Function
    #print( dec_embeddings.shape)
    #print( vocab_size)

    # validate inputs
    if( vocab_size != dec_embeddings.shape[0] ):
        print("warning: size mismatch, vocab_size != dec_embeddings.shap
e[0] ")


    start_tokens = tf.tile(tf.constant( [start_of_sequence_id], dtype=tf
.int32), [batch_size], name='start_tokens')


    # Helper for the inference process.
    inference_helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(dec_embe
ddings,
                                                                start_to
kens,
                                                                end_of_s
equence_id)

    # setup inference decoder
    inference_decoder = tf.contrib.seq2seq.BasicDecoder(dec_cell,
                                                        inference_helper
,
                                                        encoder_state,
                                                        output_layer)

    # Perform dynamic decoding using the decoder
    inference_decoder_output = tf.contrib.seq2seq.dynamic_decode(inferen
ce_decoder,
                                                            impute_finis
hed=True,
```

```
                                                                maximum_iter
ations=max_target_sequence_length)

    basic_decoder_output = inference_decoder_output[0]   # final_output
 is type BasicDecoderOutput
    return basic_decoder_output



"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_infer(decoding_layer_infer)
```

```
Tests Passed
```

Build the Decoding Layer

Implement `decoding_layer()` to create a Decoder RNN layer.

- Embed the target sequences
- Construct the decoder LSTM cell (just like you constructed the encoder cell above)
- Create an output layer to map the outputs of the decoder to the elements of our vocabulary
- Use the your `decoding_layer_train(encoder_state, dec_cell, dec_embed_input, target_sequence_length, max_target_sequence_length, output_layer, keep_prob)` function to get the training logits.
- Use your `decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_sequence_id, end_of_sequence_id, max_target_sequence_length, vocab_size, output_layer, batch_size, keep_prob)` function to get the inference logits.

Note: You'll need to use tf.variable_scope (https://www.tensorflow.org/api_docs/python/tf/variable_scope) to share variables between training and inference.

In [12]:
```
def decoding_layer(dec_input, encoder_state,
                   target_sequence_length, max_target_sequence_length,
                   rnn_size,
                   num_layers, target_vocab_to_int, target_vocab_size,
                   batch_size, keep_prob, decoding_embedding_size):
    """
    Create decoding layer
    :param dec_input: Decoder input
    :param encoder_state: Encoder state
    :param target_sequence_length: The lengths of each sequence in the t
arget batch
    :param max_target_sequence_length: Maximum length of target sequence
s
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target words t
o an id
    :param target_vocab_size: Size of target vocabulary
    :param batch_size: The size of the batch
    :param keep_prob: Dropout keep probability
    :param decoding_embedding_size: Decoding embedding size
    :return: Tuple of (Training BasicDecoderOutput, Inference BasicDecod
erOutput)
    """
    # TODO: Implement Function

    # Embed target sequences
    dec_embeddings = tf.Variable(tf.random_uniform([target_vocab_size, d
ecoding_embedding_size]))
    dec_embed_input = tf.nn.embedding_lookup(dec_embeddings, dec_input)


    # Create the decoder cell
    def make_dec_cell(rnn_size):
        dec_cell = tf.contrib.rnn.LSTMCell(rnn_size,
                                           initializer=tf.random_uniform
_initializer(-0.1, 0.1, seed=2))
        return dec_cell

    dec_cell = tf.contrib.rnn.MultiRNNCell([make_dec_cell(rnn_size) for
_ in range(num_layers)])


    # Dense layer to translate the decoder output at each timestep
    output_layer = Dense(target_vocab_size,
                         kernel_initializer = tf.truncated_normal_initia
lizer(mean = 0.0, stddev=0.1))


     # training decoder output
    with tf.variable_scope("decode"):
            train_decoder_output = decoding_layer_train(encoder_state, d
ec_cell, dec_embed_input,
                                                       target_sequence_le
ngth, max_target_sequence_length,
                                                       output_layer, keep
```

```
_prob)

     # inference decoder output
    with tf.variable_scope("decode", reuse=True):
        start_of_sequence_id = target_vocab_to_int['<GO>']
        end_of_sequence_id = target_vocab_to_int['<EOS>']
        infer_decoder_output = decoding_layer_infer(encoder_state, dec_c
ell, dec_embeddings, start_of_sequence_id,
                                                     end_of_sequence_id,
max_target_sequence_length,
                                                     target_vocab_size,
output_layer, batch_size, keep_prob)


    return train_decoder_output, infer_decoder_output  # note: type = Ba
sicDecoderOutput



"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer(decoding_layer)
```

```
Tests Passed
```

Build the Neural Network

Apply the functions you implemented above to:

- Encode the input using your `encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob,  source_sequence_length, source_vocab_size, encoding_embedding_size)`.
- Process target data using your `process_decoder_input(target_data, target_vocab_to_int, batch_size)` function.
- Decode the encoded input using your `decoding_layer(dec_input, enc_state, target_sequence_length, max_target_sentence_length, rnn_size, num_layers, target_vocab_to_int, target_vocab_size, batch_size, keep_prob, dec_embedding_size)` function.

In [13]:
```
def seq2seq_model(input_data, target_data, keep_prob, batch_size,
                  source_sequence_length, target_sequence_length,
                  max_target_sentence_length,
                  source_vocab_size, target_vocab_size,
                  enc_embedding_size, dec_embedding_size,
                  rnn_size, num_layers, target_vocab_to_int):
    """
    Build the Sequence-to-Sequence part of the neural network
    :param input_data: Input placeholder
    :param target_data: Target placeholder
    :param keep_prob: Dropout keep probability placeholder
    :param batch_size: Batch Size
    :param source_sequence_length: Sequence Lengths of source sequences
 in the batch
    :param target_sequence_length: Sequence Lengths of target sequences
 in the batch
    :param source_vocab_size: Source vocabulary size
    :param target_vocab_size: Target vocabulary size
    :param enc_embedding_size: Decoder embedding size
    :param dec_embedding_size: Encoder embedding size
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target words t
o an id
    :return: Tuple of (Training BasicDecoderOutput, Inference BasicDecod
erOutput)
    """
    # TODO: Implement Function

    # pass input data to encoder and get back the encoder state
    enc_output, enc_state = encoding_layer(input_data, rnn_size, num_lay
ers, keep_prob,
                                             source_sequence_length, sourc
e_vocab_size,
                                             enc_embedding_size)

    # pre-process decoder input data
    dec_input = process_decoder_input(target_data, target_vocab_to_int,
batch_size)



    # Pass encoder state and decoder inputs to the decoders
    training_decoder_output, inference_decoder_output = decoding_layer(d
ec_input, enc_state,
                                                                          t
arget_sequence_length,
                                                                          m
ax_target_sentence_length, rnn_size,
                                                                          n
um_layers, target_vocab_to_int,
                                                                          t
arget_vocab_size, batch_size, keep_prob,
                                                                          d
ec_embedding_size)
```

```
    return training_decoder_output, inference_decoder_output



"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_seq2seq_model(seq2seq_model)
```

```
Tests Passed
```

Neural Network Training

Hyperparameters

Tune the following parameters:

- Set `epochs` to the number of epochs.
- Set `batch_size` to the batch size.
- Set `rnn_size` to the size of the RNNs.
- Set `num_layers` to the number of layers.
- Set `encoding_embedding_size` to the size of the embedding for the encoder.
- Set `decoding_embedding_size` to the size of the embedding for the decoder.
- Set `learning_rate` to the learning rate.
- Set `keep_probability` to the Dropout keep probability
- Set `display_step` to state how many steps between each debug output statement

In [22]:
```
# Number of Epochs
epochs = 25
# Batch Size
batch_size = 256
# RNN Size
rnn_size = 256
# Number of Layers
num_layers = 3
# Embedding Size
encoding_embedding_size = 200
decoding_embedding_size = 200
# Learning Rate
learning_rate = 1e-3
# Dropout Keep Probability
keep_probability = 0.5
display_step = 25
```

Build the Graph

Build the graph using the neural network you implemented.

In [23]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
save_path = 'checkpoints/dev'
(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_t
o_int), _ = helper.load_preprocess()
max_target_sentence_length = max([len(sentence) for sentence in source_i
nt_text])

train_graph = tf.Graph()
with train_graph.as_default():
    input_data, targets, lr, keep_prob, target_sequence_length, max_targ
et_sequence_length, source_sequence_length = model_inputs()

    #sequence_length = tf.placeholder_with_default(max_target_sentence_l
ength, None, name='sequence_length')
    input_shape = tf.shape(input_data)

    train_logits, inference_logits = seq2seq_model(tf.reverse(input_data
, [-1]),
                                                   targets,
                                                   keep_prob,
                                                   batch_size,
                                                   source_sequence_lengt
h,
                                                   target_sequence_lengt
h,
                                                   max_target_sequence_l
ength,
                                                   len(source_vocab_to_i
nt),
                                                   len(target_vocab_to_i
nt),
                                                   encoding_embedding_si
ze,
                                                   decoding_embedding_si
ze,
                                                   rnn_size,
                                                   num_layers,
                                                   target_vocab_to_int)


    training_logits = tf.identity(train_logits.rnn_output, name='logits'
)
    inference_logits = tf.identity(inference_logits.sample_id, name='pre
dictions')

    masks = tf.sequence_mask(target_sequence_length, max_target_sequence
_length, dtype=tf.float32, name='masks')

    with tf.name_scope("optimization"):
        # Loss function
        cost = tf.contrib.seq2seq.sequence_loss(
            training_logits,
            targets,
            masks)
```

```
        # Optimizer
        optimizer = tf.train.AdamOptimizer(lr)

        # Gradient Clipping
        gradients = optimizer.compute_gradients(cost)
        capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for g
rad, var in gradients if grad is not None]
        train_op = optimizer.apply_gradients(capped_gradients)
```

Batch and pad the source and target sequences

In [24]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
def pad_sentence_batch(sentence_batch, pad_int):
    """Pad sentences with <PAD> so that each sentence of a batch has the
same length"""
    max_sentence = max([len(sentence) for sentence in sentence_batch])
    return [sentence + [pad_int] * (max_sentence - len(sentence)) for se
ntence in sentence_batch]


def get_batches(sources, targets, batch_size, source_pad_int, target_pad
_int):
    """Batch targets, sources, and the lengths of their sentences togeth
er"""
    for batch_i in range(0, len(sources)//batch_size):
        start_i = batch_i * batch_size

        # Slice the right amount for the batch
        sources_batch = sources[start_i:start_i + batch_size]
        targets_batch = targets[start_i:start_i + batch_size]

        # Pad
        pad_sources_batch = np.array(pad_sentence_batch(sources_batch, s
ource_pad_int))
        pad_targets_batch = np.array(pad_sentence_batch(targets_batch, t
arget_pad_int))

        # Need the lengths for the _lengths parameters
        pad_targets_lengths = []
        for target in pad_targets_batch:
            pad_targets_lengths.append(len(target))

        pad_source_lengths = []
        for source in pad_sources_batch:
            pad_source_lengths.append(len(source))

        yield pad_sources_batch, pad_targets_batch, pad_source_lengths,
pad_targets_lengths
```

Train

Train the neural network on the preprocessed data. If you have a hard time getting a good loss, check the forms to see if anyone is having the same problem.

In [25]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
def get_accuracy(target, logits):
    """
    Calculate accuracy
    """
    max_seq = max(target.shape[1], logits.shape[1])
    if max_seq - target.shape[1]:
        target = np.pad(
            target,
            [(0,0),(0,max_seq - target.shape[1])],
            'constant')
    if max_seq - logits.shape[1]:
        logits = np.pad(
            logits,
            [(0,0),(0,max_seq - logits.shape[1])],
            'constant')

    return np.mean(np.equal(target, logits))

# Split data to training and validation sets
train_source = source_int_text[batch_size:]
train_target = target_int_text[batch_size:]
valid_source = source_int_text[:batch_size]
valid_target = target_int_text[:batch_size]
(valid_sources_batch, valid_targets_batch, valid_sources_lengths, valid_
targets_lengths ) = next(get_batches(valid_source,

valid_target,

batch_size,

source_vocab_to_int['<PAD>'],

target_vocab_to_int['<PAD>']))
with tf.Session(graph=train_graph) as sess:
    sess.run(tf.global_variables_initializer())

    for epoch_i in range(epochs):
        for batch_i, (source_batch, target_batch, sources_lengths, targe
ts_lengths) in enumerate(
                get_batches(train_source, train_target, batch_size,
                            source_vocab_to_int['<PAD>'],
                            target_vocab_to_int['<PAD>'])):

            _, loss = sess.run(
                [train_op, cost],
                {input_data: source_batch,
                 targets: target_batch,
                 lr: learning_rate,
                 target_sequence_length: targets_lengths,
                 source_sequence_length: sources_lengths,
                 keep_prob: keep_probability})
```

```
            if batch_i % display_step == 0 and batch_i > 0:


                batch_train_logits = sess.run(
                    inference_logits,
                    {input_data: source_batch,
                     source_sequence_length: sources_lengths,
                     target_sequence_length: targets_lengths,
                     keep_prob: 1.0})


                batch_valid_logits = sess.run(
                    inference_logits,
                    {input_data: valid_sources_batch,
                     source_sequence_length: valid_sources_lengths,
                     target_sequence_length: valid_targets_lengths,
                     keep_prob: 1.0})

                train_acc = get_accuracy(target_batch, batch_train_logit
s)

                valid_acc = get_accuracy(valid_targets_batch, batch_vali
d_logits)

                print('Epoch {:>3} Batch {:>4}/{} - Train Accuracy: {:>
6.4f}, Validation Accuracy: {:>6.4f}, Loss: {:>6.4f}'
                      .format(epoch_i, batch_i, len(source_int_text) //
batch_size, train_acc, valid_acc, loss))

    # Save Model
    saver = tf.train.Saver()
    saver.save(sess, save_path)
    print('Model Trained and Saved')
```

```
Epoch   0 Batch   25/538 - Train Accuracy: 0.3775, Validation Accuracy:
0.4238, Loss: 3.0133
Epoch   0 Batch   50/538 - Train Accuracy: 0.4453, Validation Accuracy:
0.4753, Loss: 2.4807
Epoch   0 Batch   75/538 - Train Accuracy: 0.4952, Validation Accuracy:
0.5201, Loss: 1.9982
Epoch   0 Batch  100/538 - Train Accuracy: 0.4777, Validation Accuracy:
0.5270, Loss: 1.7909
Epoch   0 Batch  125/538 - Train Accuracy: 0.5201, Validation Accuracy:
0.5467, Loss: 1.6026
Epoch   0 Batch  150/538 - Train Accuracy: 0.5160, Validation Accuracy:
0.5392, Loss: 1.5027
Epoch   0 Batch  175/538 - Train Accuracy: 0.4842, Validation Accuracy:
0.5533, Loss: 1.4314
Epoch   0 Batch  200/538 - Train Accuracy: 0.5156, Validation Accuracy:
0.5419, Loss: 1.2931
Epoch   0 Batch  225/538 - Train Accuracy: 0.5603, Validation Accuracy:
0.5577, Loss: 1.1267
Epoch   0 Batch  250/538 - Train Accuracy: 0.5307, Validation Accuracy:
0.5687, Loss: 1.0745
Epoch   0 Batch  275/538 - Train Accuracy: 0.5547, Validation Accuracy:
0.5803, Loss: 1.0447
Epoch   0 Batch  300/538 - Train Accuracy: 0.6006, Validation Accuracy:
0.5984, Loss: 0.9218
Epoch   0 Batch  325/538 - Train Accuracy: 0.5919, Validation Accuracy:
0.6035, Loss: 0.8834
Epoch   0 Batch  350/538 - Train Accuracy: 0.6071, Validation Accuracy:
0.6135, Loss: 0.8619
Epoch   0 Batch  375/538 - Train Accuracy: 0.6135, Validation Accuracy:
0.6163, Loss: 0.7781
Epoch   0 Batch  400/538 - Train Accuracy: 0.5975, Validation Accuracy:
0.6110, Loss: 0.7811
Epoch   0 Batch  425/538 - Train Accuracy: 0.6021, Validation Accuracy:
0.6119, Loss: 0.7432
Epoch   0 Batch  450/538 - Train Accuracy: 0.6324, Validation Accuracy:
0.6317, Loss: 0.7367
Epoch   0 Batch  475/538 - Train Accuracy: 0.6081, Validation Accuracy:
0.6390, Loss: 0.6680
Epoch   0 Batch  500/538 - Train Accuracy: 0.6657, Validation Accuracy:
0.6584, Loss: 0.5956
Epoch   0 Batch  525/538 - Train Accuracy: 0.6763, Validation Accuracy:
0.6539, Loss: 0.6024
Epoch   1 Batch   25/538 - Train Accuracy: 0.6389, Validation Accuracy:
0.6694, Loss: 0.5904
Epoch   1 Batch   50/538 - Train Accuracy: 0.6754, Validation Accuracy:
0.6726, Loss: 0.5524
Epoch   1 Batch   75/538 - Train Accuracy: 0.7063, Validation Accuracy:
0.6966, Loss: 0.4997
Epoch   1 Batch  100/538 - Train Accuracy: 0.7268, Validation Accuracy:
0.7001, Loss: 0.4757
Epoch   1 Batch  125/538 - Train Accuracy: 0.7323, Validation Accuracy:
0.7097, Loss: 0.4529
Epoch   1 Batch  150/538 - Train Accuracy: 0.7316, Validation Accuracy:
0.7156, Loss: 0.4456
Epoch   1 Batch  175/538 - Train Accuracy: 0.7355, Validation Accuracy:
0.7230, Loss: 0.4409
Epoch   1 Batch  200/538 - Train Accuracy: 0.7668, Validation Accuracy:
```

```
0.7406, Loss: 0.3839
Epoch   1 Batch  225/538 - Train Accuracy: 0.7734, Validation Accuracy:
0.7488, Loss: 0.3711
Epoch   1 Batch  250/538 - Train Accuracy: 0.7855, Validation Accuracy:
0.7646, Loss: 0.3441
Epoch   1 Batch  275/538 - Train Accuracy: 0.8074, Validation Accuracy:
0.7761, Loss: 0.3452
Epoch   1 Batch  300/538 - Train Accuracy: 0.8149, Validation Accuracy:
0.7892, Loss: 0.3114
Epoch   1 Batch  325/538 - Train Accuracy: 0.8508, Validation Accuracy:
0.8319, Loss: 0.2799
Epoch   1 Batch  350/538 - Train Accuracy: 0.8142, Validation Accuracy:
0.8180, Loss: 0.2965
Epoch   1 Batch  375/538 - Train Accuracy: 0.8438, Validation Accuracy:
0.8263, Loss: 0.2271
Epoch   1 Batch  400/538 - Train Accuracy: 0.8337, Validation Accuracy:
0.8031, Loss: 0.2533
Epoch   1 Batch  425/538 - Train Accuracy: 0.8705, Validation Accuracy:
0.8510, Loss: 0.2411
Epoch   1 Batch  450/538 - Train Accuracy: 0.8616, Validation Accuracy:
0.8375, Loss: 0.2356
Epoch   1 Batch  475/538 - Train Accuracy: 0.8406, Validation Accuracy:
0.8269, Loss: 0.2434
Epoch   1 Batch  500/538 - Train Accuracy: 0.8846, Validation Accuracy:
0.8290, Loss: 0.1886
Epoch   1 Batch  525/538 - Train Accuracy: 0.8876, Validation Accuracy:
0.8553, Loss: 0.1904
Epoch   2 Batch   25/538 - Train Accuracy: 0.8783, Validation Accuracy:
0.8635, Loss: 0.1811
Epoch   2 Batch   50/538 - Train Accuracy: 0.8898, Validation Accuracy:
0.8608, Loss: 0.1551
Epoch   2 Batch   75/538 - Train Accuracy: 0.8679, Validation Accuracy:
0.8729, Loss: 0.1627
Epoch   2 Batch  100/538 - Train Accuracy: 0.9025, Validation Accuracy:
0.8761, Loss: 0.1426
Epoch   2 Batch  125/538 - Train Accuracy: 0.8949, Validation Accuracy:
0.8897, Loss: 0.1432
Epoch   2 Batch  150/538 - Train Accuracy: 0.9078, Validation Accuracy:
0.8786, Loss: 0.1264
Epoch   2 Batch  175/538 - Train Accuracy: 0.9025, Validation Accuracy:
0.8892, Loss: 0.1340
Epoch   2 Batch  200/538 - Train Accuracy: 0.9133, Validation Accuracy:
0.9007, Loss: 0.1118
Epoch   2 Batch  225/538 - Train Accuracy: 0.9141, Validation Accuracy:
0.8833, Loss: 0.1268
Epoch   2 Batch  250/538 - Train Accuracy: 0.9270, Validation Accuracy:
0.8912, Loss: 0.1137
Epoch   2 Batch  275/538 - Train Accuracy: 0.9025, Validation Accuracy:
0.8897, Loss: 0.1206
Epoch   2 Batch  300/538 - Train Accuracy: 0.9081, Validation Accuracy:
0.9057, Loss: 0.1065
Epoch   2 Batch  325/538 - Train Accuracy: 0.9167, Validation Accuracy:
0.9110, Loss: 0.0965
Epoch   2 Batch  350/538 - Train Accuracy: 0.9064, Validation Accuracy:
0.9164, Loss: 0.1078
Epoch   2 Batch  375/538 - Train Accuracy: 0.9169, Validation Accuracy:
0.8894, Loss: 0.0876
```

```
Epoch   2 Batch  400/538 - Train Accuracy: 0.9230, Validation Accuracy:
0.9144, Loss: 0.0924
Epoch   2 Batch  425/538 - Train Accuracy: 0.8919, Validation Accuracy:
0.8855, Loss: 0.1087
Epoch   2 Batch  450/538 - Train Accuracy: 0.9126, Validation Accuracy:
0.9002, Loss: 0.1093
Epoch   2 Batch  475/538 - Train Accuracy: 0.9152, Validation Accuracy:
0.9059, Loss: 0.0897
Epoch   2 Batch  500/538 - Train Accuracy: 0.9430, Validation Accuracy:
0.9020, Loss: 0.0652
Epoch   2 Batch  525/538 - Train Accuracy: 0.9284, Validation Accuracy:
0.9102, Loss: 0.0841
Epoch   3 Batch   25/538 - Train Accuracy: 0.9172, Validation Accuracy:
0.9084, Loss: 0.0801
Epoch   3 Batch   50/538 - Train Accuracy: 0.9248, Validation Accuracy:
0.9109, Loss: 0.0702
Epoch   3 Batch   75/538 - Train Accuracy: 0.9312, Validation Accuracy:
0.9327, Loss: 0.0773
Epoch   3 Batch  100/538 - Train Accuracy: 0.9354, Validation Accuracy:
0.9244, Loss: 0.0640
Epoch   3 Batch  125/538 - Train Accuracy: 0.9219, Validation Accuracy:
0.9327, Loss: 0.0755
Epoch   3 Batch  150/538 - Train Accuracy: 0.9359, Validation Accuracy:
0.9185, Loss: 0.0667
Epoch   3 Batch  175/538 - Train Accuracy: 0.9436, Validation Accuracy:
0.9082, Loss: 0.0592
Epoch   3 Batch  200/538 - Train Accuracy: 0.9418, Validation Accuracy:
0.9238, Loss: 0.0587
Epoch   3 Batch  225/538 - Train Accuracy: 0.9576, Validation Accuracy:
0.9173, Loss: 0.0649
Epoch   3 Batch  250/538 - Train Accuracy: 0.9320, Validation Accuracy:
0.9254, Loss: 0.0592
Epoch   3 Batch  275/538 - Train Accuracy: 0.9398, Validation Accuracy:
0.9331, Loss: 0.0641
Epoch   3 Batch  300/538 - Train Accuracy: 0.9291, Validation Accuracy:
0.9345, Loss: 0.0670
Epoch   3 Batch  325/538 - Train Accuracy: 0.9407, Validation Accuracy:
0.9414, Loss: 0.0531
Epoch   3 Batch  350/538 - Train Accuracy: 0.9462, Validation Accuracy:
0.9414, Loss: 0.0711
Epoch   3 Batch  375/538 - Train Accuracy: 0.9418, Validation Accuracy:
0.9320, Loss: 0.0490
Epoch   3 Batch  400/538 - Train Accuracy: 0.9609, Validation Accuracy:
0.9343, Loss: 0.0591
Epoch   3 Batch  425/538 - Train Accuracy: 0.9208, Validation Accuracy:
0.9487, Loss: 0.0751
Epoch   3 Batch  450/538 - Train Accuracy: 0.9230, Validation Accuracy:
0.9466, Loss: 0.0739
Epoch   3 Batch  475/538 - Train Accuracy: 0.9408, Validation Accuracy:
0.9102, Loss: 0.0552
Epoch   3 Batch  500/538 - Train Accuracy: 0.9666, Validation Accuracy:
0.9382, Loss: 0.0474
Epoch   3 Batch  525/538 - Train Accuracy: 0.9386, Validation Accuracy:
0.9366, Loss: 0.0556
Epoch   4 Batch   25/538 - Train Accuracy: 0.9443, Validation Accuracy:
0.9425, Loss: 0.0543
Epoch   4 Batch   50/538 - Train Accuracy: 0.9531, Validation Accuracy:
```

```
0.9363, Loss: 0.0514
Epoch   4 Batch   75/538 - Train Accuracy: 0.9332, Validation Accuracy: 
0.9487, Loss: 0.0550
Epoch   4 Batch  100/538 - Train Accuracy: 0.9637, Validation Accuracy: 
0.9444, Loss: 0.0424
Epoch   4 Batch  125/538 - Train Accuracy: 0.9568, Validation Accuracy: 
0.9450, Loss: 0.0496
Epoch   4 Batch  150/538 - Train Accuracy: 0.9605, Validation Accuracy: 
0.9437, Loss: 0.0473
Epoch   4 Batch  175/538 - Train Accuracy: 0.9602, Validation Accuracy: 
0.9238, Loss: 0.0444
Epoch   4 Batch  200/538 - Train Accuracy: 0.9529, Validation Accuracy: 
0.9434, Loss: 0.0407
Epoch   4 Batch  225/538 - Train Accuracy: 0.9656, Validation Accuracy: 
0.9276, Loss: 0.0444
Epoch   4 Batch  250/538 - Train Accuracy: 0.9547, Validation Accuracy: 
0.9297, Loss: 0.0519
Epoch   4 Batch  275/538 - Train Accuracy: 0.9518, Validation Accuracy: 
0.9284, Loss: 0.0555
Epoch   4 Batch  300/538 - Train Accuracy: 0.9513, Validation Accuracy: 
0.9355, Loss: 0.0449
Epoch   4 Batch  325/538 - Train Accuracy: 0.9611, Validation Accuracy: 
0.9441, Loss: 0.0406
Epoch   4 Batch  350/538 - Train Accuracy: 0.9600, Validation Accuracy: 
0.9553, Loss: 0.0529
Epoch   4 Batch  375/538 - Train Accuracy: 0.9522, Validation Accuracy: 
0.9522, Loss: 0.0411
Epoch   4 Batch  400/538 - Train Accuracy: 0.9563, Validation Accuracy: 
0.9535, Loss: 0.0457
Epoch   4 Batch  425/538 - Train Accuracy: 0.9286, Validation Accuracy: 
0.9544, Loss: 0.0544
Epoch   4 Batch  450/538 - Train Accuracy: 0.9355, Validation Accuracy: 
0.9636, Loss: 0.0540
Epoch   4 Batch  475/538 - Train Accuracy: 0.9589, Validation Accuracy: 
0.9629, Loss: 0.0417
Epoch   4 Batch  500/538 - Train Accuracy: 0.9615, Validation Accuracy: 
0.9505, Loss: 0.0330
Epoch   4 Batch  525/538 - Train Accuracy: 0.9632, Validation Accuracy: 
0.9501, Loss: 0.0431
Epoch   5 Batch   25/538 - Train Accuracy: 0.9584, Validation Accuracy: 
0.9599, Loss: 0.0406
Epoch   5 Batch   50/538 - Train Accuracy: 0.9566, Validation Accuracy: 
0.9492, Loss: 0.0363
Epoch   5 Batch   75/538 - Train Accuracy: 0.9529, Validation Accuracy: 
0.9441, Loss: 0.0388
Epoch   5 Batch  100/538 - Train Accuracy: 0.9646, Validation Accuracy: 
0.9554, Loss: 0.0305
Epoch   5 Batch  125/538 - Train Accuracy: 0.9693, Validation Accuracy: 
0.9709, Loss: 0.0417
Epoch   5 Batch  150/538 - Train Accuracy: 0.9643, Validation Accuracy: 
0.9423, Loss: 0.0346
Epoch   5 Batch  175/538 - Train Accuracy: 0.9730, Validation Accuracy: 
0.9515, Loss: 0.0407
Epoch   5 Batch  200/538 - Train Accuracy: 0.9607, Validation Accuracy: 
0.9547, Loss: 0.0322
Epoch   5 Batch  225/538 - Train Accuracy: 0.9647, Validation Accuracy: 
0.9498, Loss: 0.0368
```

```
Epoch   5 Batch  250/538 - Train Accuracy: 0.9695, Validation Accuracy:
0.9515, Loss: 0.0361
Epoch   5 Batch  275/538 - Train Accuracy: 0.9625, Validation Accuracy:
0.9423, Loss: 0.0407
Epoch   5 Batch  300/538 - Train Accuracy: 0.9552, Validation Accuracy:
0.9426, Loss: 0.0359
Epoch   5 Batch  325/538 - Train Accuracy: 0.9691, Validation Accuracy:
0.9464, Loss: 0.0325
Epoch   5 Batch  350/538 - Train Accuracy: 0.9632, Validation Accuracy:
0.9466, Loss: 0.0392
Epoch   5 Batch  375/538 - Train Accuracy: 0.9708, Validation Accuracy:
0.9588, Loss: 0.0313
Epoch   5 Batch  400/538 - Train Accuracy: 0.9779, Validation Accuracy:
0.9586, Loss: 0.0306
Epoch   5 Batch  425/538 - Train Accuracy: 0.9539, Validation Accuracy:
0.9636, Loss: 0.0465
Epoch   5 Batch  450/538 - Train Accuracy: 0.9479, Validation Accuracy:
0.9659, Loss: 0.0435
Epoch   5 Batch  475/538 - Train Accuracy: 0.9643, Validation Accuracy:
0.9624, Loss: 0.0345
Epoch   5 Batch  500/538 - Train Accuracy: 0.9775, Validation Accuracy:
0.9641, Loss: 0.0237
Epoch   5 Batch  525/538 - Train Accuracy: 0.9591, Validation Accuracy:
0.9576, Loss: 0.0354
Epoch   6 Batch   25/538 - Train Accuracy: 0.9484, Validation Accuracy:
0.9602, Loss: 0.0373
Epoch   6 Batch   50/538 - Train Accuracy: 0.9607, Validation Accuracy:
0.9577, Loss: 0.0343
Epoch   6 Batch   75/538 - Train Accuracy: 0.9524, Validation Accuracy:
0.9554, Loss: 0.0319
Epoch   6 Batch  100/538 - Train Accuracy: 0.9760, Validation Accuracy:
0.9723, Loss: 0.0255
Epoch   6 Batch  125/538 - Train Accuracy: 0.9663, Validation Accuracy:
0.9577, Loss: 0.0345
Epoch   6 Batch  150/538 - Train Accuracy: 0.9617, Validation Accuracy:
0.9528, Loss: 0.0310
Epoch   6 Batch  175/538 - Train Accuracy: 0.9734, Validation Accuracy:
0.9387, Loss: 0.0290
Epoch   6 Batch  200/538 - Train Accuracy: 0.9715, Validation Accuracy:
0.9583, Loss: 0.0228
Epoch   6 Batch  225/538 - Train Accuracy: 0.9552, Validation Accuracy:
0.9561, Loss: 0.0299
Epoch   6 Batch  250/538 - Train Accuracy: 0.9779, Validation Accuracy:
0.9471, Loss: 0.0297
Epoch   6 Batch  275/538 - Train Accuracy: 0.9680, Validation Accuracy:
0.9503, Loss: 0.0326
Epoch   6 Batch  300/538 - Train Accuracy: 0.9665, Validation Accuracy:
0.9402, Loss: 0.0298
Epoch   6 Batch  325/538 - Train Accuracy: 0.9684, Validation Accuracy:
0.9519, Loss: 0.0277
Epoch   6 Batch  350/538 - Train Accuracy: 0.9751, Validation Accuracy:
0.9714, Loss: 0.0320
Epoch   6 Batch  375/538 - Train Accuracy: 0.9682, Validation Accuracy:
0.9551, Loss: 0.0272
Epoch   6 Batch  400/538 - Train Accuracy: 0.9719, Validation Accuracy:
0.9789, Loss: 0.0259
Epoch   6 Batch  425/538 - Train Accuracy: 0.9570, Validation Accuracy:
```

```
0.9597, Loss: 0.0395
Epoch   6 Batch  450/538 - Train Accuracy: 0.9479, Validation Accuracy:
0.9718, Loss: 0.0370
Epoch   6 Batch  475/538 - Train Accuracy: 0.9643, Validation Accuracy:
0.9853, Loss: 0.0260
Epoch   6 Batch  500/538 - Train Accuracy: 0.9776, Validation Accuracy:
0.9528, Loss: 0.0215
Epoch   6 Batch  525/538 - Train Accuracy: 0.9691, Validation Accuracy:
0.9572, Loss: 0.0281
Epoch   7 Batch   25/538 - Train Accuracy: 0.9576, Validation Accuracy:
0.9606, Loss: 0.0284
Epoch   7 Batch   50/538 - Train Accuracy: 0.9717, Validation Accuracy:
0.9599, Loss: 0.0237
Epoch   7 Batch   75/538 - Train Accuracy: 0.9611, Validation Accuracy:
0.9540, Loss: 0.0247
Epoch   7 Batch  100/538 - Train Accuracy: 0.9814, Validation Accuracy:
0.9657, Loss: 0.0223
Epoch   7 Batch  125/538 - Train Accuracy: 0.9749, Validation Accuracy:
0.9608, Loss: 0.0285
Epoch   7 Batch  150/538 - Train Accuracy: 0.9812, Validation Accuracy:
0.9466, Loss: 0.0215
Epoch   7 Batch  175/538 - Train Accuracy: 0.9750, Validation Accuracy:
0.9441, Loss: 0.0230
Epoch   7 Batch  200/538 - Train Accuracy: 0.9600, Validation Accuracy:
0.9535, Loss: 0.0245
Epoch   7 Batch  225/538 - Train Accuracy: 0.9779, Validation Accuracy:
0.9519, Loss: 0.0252
Epoch   7 Batch  250/538 - Train Accuracy: 0.9832, Validation Accuracy:
0.9561, Loss: 0.0226
Epoch   7 Batch  275/538 - Train Accuracy: 0.9664, Validation Accuracy:
0.9515, Loss: 0.0269
Epoch   7 Batch  300/538 - Train Accuracy: 0.9741, Validation Accuracy:
0.9695, Loss: 0.0214
Epoch   7 Batch  325/538 - Train Accuracy: 0.9743, Validation Accuracy:
0.9604, Loss: 0.0204
Epoch   7 Batch  350/538 - Train Accuracy: 0.9758, Validation Accuracy:
0.9616, Loss: 0.0268
Epoch   7 Batch  375/538 - Train Accuracy: 0.9723, Validation Accuracy:
0.9650, Loss: 0.0196
Epoch   7 Batch  400/538 - Train Accuracy: 0.9771, Validation Accuracy:
0.9654, Loss: 0.0248
Epoch   7 Batch  425/538 - Train Accuracy: 0.9617, Validation Accuracy:
0.9648, Loss: 0.0394
Epoch   7 Batch  450/538 - Train Accuracy: 0.9548, Validation Accuracy:
0.9638, Loss: 0.0333
Epoch   7 Batch  475/538 - Train Accuracy: 0.9853, Validation Accuracy:
0.9696, Loss: 0.0206
Epoch   7 Batch  500/538 - Train Accuracy: 0.9755, Validation Accuracy:
0.9647, Loss: 0.0212
Epoch   7 Batch  525/538 - Train Accuracy: 0.9734, Validation Accuracy:
0.9592, Loss: 0.0284
Epoch   8 Batch   25/538 - Train Accuracy: 0.9619, Validation Accuracy:
0.9576, Loss: 0.0270
Epoch   8 Batch   50/538 - Train Accuracy: 0.9705, Validation Accuracy:
0.9728, Loss: 0.0191
Epoch   8 Batch   75/538 - Train Accuracy: 0.9771, Validation Accuracy:
0.9680, Loss: 0.0215
```

```
Epoch   8 Batch  100/538 - Train Accuracy: 0.9797, Validation Accuracy:
0.9583, Loss: 0.0171
Epoch   8 Batch  125/538 - Train Accuracy: 0.9788, Validation Accuracy:
0.9741, Loss: 0.0270
Epoch   8 Batch  150/538 - Train Accuracy: 0.9814, Validation Accuracy:
0.9734, Loss: 0.0210
Epoch   8 Batch  175/538 - Train Accuracy: 0.9861, Validation Accuracy:
0.9588, Loss: 0.0159
Epoch   8 Batch  200/538 - Train Accuracy: 0.9781, Validation Accuracy:
0.9675, Loss: 0.0154
Epoch   8 Batch  225/538 - Train Accuracy: 0.9767, Validation Accuracy:
0.9585, Loss: 0.0196
Epoch   8 Batch  250/538 - Train Accuracy: 0.9908, Validation Accuracy:
0.9558, Loss: 0.0189
Epoch   8 Batch  275/538 - Train Accuracy: 0.9789, Validation Accuracy:
0.9585, Loss: 0.0184
Epoch   8 Batch  300/538 - Train Accuracy: 0.9751, Validation Accuracy:
0.9711, Loss: 0.0224
Epoch   8 Batch  325/538 - Train Accuracy: 0.9812, Validation Accuracy:
0.9698, Loss: 0.0190
Epoch   8 Batch  350/538 - Train Accuracy: 0.9814, Validation Accuracy:
0.9636, Loss: 0.0222
Epoch   8 Batch  375/538 - Train Accuracy: 0.9860, Validation Accuracy:
0.9645, Loss: 0.0215
Epoch   8 Batch  400/538 - Train Accuracy: 0.9777, Validation Accuracy:
0.9677, Loss: 0.0212
Epoch   8 Batch  425/538 - Train Accuracy: 0.9624, Validation Accuracy:
0.9609, Loss: 0.0310
Epoch   8 Batch  450/538 - Train Accuracy: 0.9615, Validation Accuracy:
0.9604, Loss: 0.0275
Epoch   8 Batch  475/538 - Train Accuracy: 0.9838, Validation Accuracy:
0.9730, Loss: 0.0175
Epoch   8 Batch  500/538 - Train Accuracy: 0.9840, Validation Accuracy:
0.9679, Loss: 0.0141
Epoch   8 Batch  525/538 - Train Accuracy: 0.9784, Validation Accuracy:
0.9645, Loss: 0.0227
Epoch   9 Batch   25/538 - Train Accuracy: 0.9807, Validation Accuracy:
0.9597, Loss: 0.0213
Epoch   9 Batch   50/538 - Train Accuracy: 0.9809, Validation Accuracy:
0.9735, Loss: 0.0183
Epoch   9 Batch   75/538 - Train Accuracy: 0.9753, Validation Accuracy:
0.9675, Loss: 0.0178
Epoch   9 Batch  100/538 - Train Accuracy: 0.9887, Validation Accuracy:
0.9716, Loss: 0.0161
Epoch   9 Batch  125/538 - Train Accuracy: 0.9870, Validation Accuracy:
0.9647, Loss: 0.0236
Epoch   9 Batch  150/538 - Train Accuracy: 0.9861, Validation Accuracy:
0.9609, Loss: 0.0179
Epoch   9 Batch  175/538 - Train Accuracy: 0.9869, Validation Accuracy:
0.9554, Loss: 0.0166
Epoch   9 Batch  200/538 - Train Accuracy: 0.9752, Validation Accuracy:
0.9592, Loss: 0.0158
Epoch   9 Batch  225/538 - Train Accuracy: 0.9680, Validation Accuracy:
0.9583, Loss: 0.0217
Epoch   9 Batch  250/538 - Train Accuracy: 0.9805, Validation Accuracy:
0.9576, Loss: 0.0214
Epoch   9 Batch  275/538 - Train Accuracy: 0.9770, Validation Accuracy:
```

```
0.9643, Loss: 0.0186
Epoch   9 Batch  300/538 - Train Accuracy: 0.9784, Validation Accuracy:
0.9755, Loss: 0.0187
Epoch   9 Batch  325/538 - Train Accuracy: 0.9857, Validation Accuracy:
0.9700, Loss: 0.0152
Epoch   9 Batch  350/538 - Train Accuracy: 0.9758, Validation Accuracy:
0.9735, Loss: 0.0217
Epoch   9 Batch  375/538 - Train Accuracy: 0.9821, Validation Accuracy:
0.9638, Loss: 0.0204
Epoch   9 Batch  400/538 - Train Accuracy: 0.9868, Validation Accuracy:
0.9712, Loss: 0.0213
Epoch   9 Batch  425/538 - Train Accuracy: 0.9760, Validation Accuracy:
0.9719, Loss: 0.0248
Epoch   9 Batch  450/538 - Train Accuracy: 0.9647, Validation Accuracy:
0.9759, Loss: 0.0265
Epoch   9 Batch  475/538 - Train Accuracy: 0.9773, Validation Accuracy:
0.9778, Loss: 0.0134
Epoch   9 Batch  500/538 - Train Accuracy: 0.9911, Validation Accuracy:
0.9677, Loss: 0.0129
Epoch   9 Batch  525/538 - Train Accuracy: 0.9771, Validation Accuracy:
0.9640, Loss: 0.0212
Epoch  10 Batch   25/538 - Train Accuracy: 0.9793, Validation Accuracy:
0.9572, Loss: 0.0216
Epoch  10 Batch   50/538 - Train Accuracy: 0.9705, Validation Accuracy:
0.9732, Loss: 0.0133
Epoch  10 Batch   75/538 - Train Accuracy: 0.9734, Validation Accuracy:
0.9650, Loss: 0.0193
Epoch  10 Batch  100/538 - Train Accuracy: 0.9873, Validation Accuracy:
0.9675, Loss: 0.0126
Epoch  10 Batch  125/538 - Train Accuracy: 0.9859, Validation Accuracy:
0.9741, Loss: 0.0197
Epoch  10 Batch  150/538 - Train Accuracy: 0.9910, Validation Accuracy:
0.9647, Loss: 0.0161
Epoch  10 Batch  175/538 - Train Accuracy: 0.9936, Validation Accuracy:
0.9624, Loss: 0.0135
Epoch  10 Batch  200/538 - Train Accuracy: 0.9879, Validation Accuracy:
0.9711, Loss: 0.0135
Epoch  10 Batch  225/538 - Train Accuracy: 0.9864, Validation Accuracy:
0.9537, Loss: 0.0145
Epoch  10 Batch  250/538 - Train Accuracy: 0.9820, Validation Accuracy:
0.9695, Loss: 0.0184
Epoch  10 Batch  275/538 - Train Accuracy: 0.9758, Validation Accuracy:
0.9748, Loss: 0.0207
Epoch  10 Batch  300/538 - Train Accuracy: 0.9766, Validation Accuracy:
0.9691, Loss: 0.0150
Epoch  10 Batch  325/538 - Train Accuracy: 0.9831, Validation Accuracy:
0.9686, Loss: 0.0164
Epoch  10 Batch  350/538 - Train Accuracy: 0.9741, Validation Accuracy:
0.9716, Loss: 0.0210
Epoch  10 Batch  375/538 - Train Accuracy: 0.9855, Validation Accuracy:
0.9744, Loss: 0.0131
Epoch  10 Batch  400/538 - Train Accuracy: 0.9872, Validation Accuracy:
0.9739, Loss: 0.0167
Epoch  10 Batch  425/538 - Train Accuracy: 0.9676, Validation Accuracy:
0.9711, Loss: 0.0233
Epoch  10 Batch  450/538 - Train Accuracy: 0.9786, Validation Accuracy:
0.9753, Loss: 0.0229
```

```
Epoch  10 Batch  475/538 - Train Accuracy: 0.9892, Validation Accuracy:
0.9798, Loss: 0.0129
Epoch  10 Batch  500/538 - Train Accuracy: 0.9837, Validation Accuracy:
0.9730, Loss: 0.0106
Epoch  10 Batch  525/538 - Train Accuracy: 0.9745, Validation Accuracy:
0.9712, Loss: 0.0211
Epoch  11 Batch   25/538 - Train Accuracy: 0.9648, Validation Accuracy:
0.9560, Loss: 0.0213
Epoch  11 Batch   50/538 - Train Accuracy: 0.9730, Validation Accuracy:
0.9711, Loss: 0.0145
Epoch  11 Batch   75/538 - Train Accuracy: 0.9786, Validation Accuracy:
0.9721, Loss: 0.0148
Epoch  11 Batch  100/538 - Train Accuracy: 0.9893, Validation Accuracy:
0.9672, Loss: 0.0128
Epoch  11 Batch  125/538 - Train Accuracy: 0.9881, Validation Accuracy:
0.9782, Loss: 0.0216
Epoch  11 Batch  150/538 - Train Accuracy: 0.9891, Validation Accuracy:
0.9759, Loss: 0.0150
Epoch  11 Batch  175/538 - Train Accuracy: 0.9844, Validation Accuracy:
0.9652, Loss: 0.0166
Epoch  11 Batch  200/538 - Train Accuracy: 0.9807, Validation Accuracy:
0.9748, Loss: 0.0109
Epoch  11 Batch  225/538 - Train Accuracy: 0.9777, Validation Accuracy:
0.9593, Loss: 0.0147
Epoch  11 Batch  250/538 - Train Accuracy: 0.9836, Validation Accuracy:
0.9716, Loss: 0.0155
Epoch  11 Batch  275/538 - Train Accuracy: 0.9801, Validation Accuracy:
0.9735, Loss: 0.0176
Epoch  11 Batch  300/538 - Train Accuracy: 0.9870, Validation Accuracy:
0.9664, Loss: 0.0145
Epoch  11 Batch  325/538 - Train Accuracy: 0.9939, Validation Accuracy:
0.9735, Loss: 0.0143
Epoch  11 Batch  350/538 - Train Accuracy: 0.9860, Validation Accuracy:
0.9705, Loss: 0.0198
Epoch  11 Batch  375/538 - Train Accuracy: 0.9818, Validation Accuracy:
0.9629, Loss: 0.0123
Epoch  11 Batch  400/538 - Train Accuracy: 0.9903, Validation Accuracy:
0.9748, Loss: 0.0136
Epoch  11 Batch  425/538 - Train Accuracy: 0.9680, Validation Accuracy:
0.9663, Loss: 0.0235
Epoch  11 Batch  450/538 - Train Accuracy: 0.9803, Validation Accuracy:
0.9684, Loss: 0.0184
Epoch  11 Batch  475/538 - Train Accuracy: 0.9879, Validation Accuracy:
0.9773, Loss: 0.0111
Epoch  11 Batch  500/538 - Train Accuracy: 0.9856, Validation Accuracy:
0.9711, Loss: 0.0123
Epoch  11 Batch  525/538 - Train Accuracy: 0.9831, Validation Accuracy:
0.9725, Loss: 0.0146
Epoch  12 Batch   25/538 - Train Accuracy: 0.9797, Validation Accuracy:
0.9636, Loss: 0.0131
Epoch  12 Batch   50/538 - Train Accuracy: 0.9801, Validation Accuracy:
0.9751, Loss: 0.0127
Epoch  12 Batch   75/538 - Train Accuracy: 0.9840, Validation Accuracy:
0.9668, Loss: 0.0130
Epoch  12 Batch  100/538 - Train Accuracy: 0.9877, Validation Accuracy:
0.9743, Loss: 0.0114
Epoch  12 Batch  125/538 - Train Accuracy: 0.9862, Validation Accuracy:
```

```
0.9712, Loss: 0.0186
Epoch  12 Batch  150/538 - Train Accuracy: 0.9842, Validation Accuracy:
0.9693, Loss: 0.0115
Epoch  12 Batch  175/538 - Train Accuracy: 0.9873, Validation Accuracy:
0.9703, Loss: 0.0100
Epoch  12 Batch  200/538 - Train Accuracy: 0.9852, Validation Accuracy:
0.9719, Loss: 0.0131
Epoch  12 Batch  225/538 - Train Accuracy: 0.9766, Validation Accuracy:
0.9641, Loss: 0.0159
Epoch  12 Batch  250/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9750, Loss: 0.0153
Epoch  12 Batch  275/538 - Train Accuracy: 0.9809, Validation Accuracy:
0.9700, Loss: 0.0128
Epoch  12 Batch  300/538 - Train Accuracy: 0.9901, Validation Accuracy:
0.9608, Loss: 0.0138
Epoch  12 Batch  325/538 - Train Accuracy: 0.9907, Validation Accuracy:
0.9700, Loss: 0.0132
Epoch  12 Batch  350/538 - Train Accuracy: 0.9888, Validation Accuracy:
0.9771, Loss: 0.0145
Epoch  12 Batch  375/538 - Train Accuracy: 0.9913, Validation Accuracy:
0.9732, Loss: 0.0106
Epoch  12 Batch  400/538 - Train Accuracy: 0.9870, Validation Accuracy:
0.9739, Loss: 0.0152
Epoch  12 Batch  425/538 - Train Accuracy: 0.9782, Validation Accuracy:
0.9759, Loss: 0.0180
Epoch  12 Batch  450/538 - Train Accuracy: 0.9691, Validation Accuracy:
0.9700, Loss: 0.0197
Epoch  12 Batch  475/538 - Train Accuracy: 0.9710, Validation Accuracy:
0.9703, Loss: 0.0158
Epoch  12 Batch  500/538 - Train Accuracy: 0.9877, Validation Accuracy:
0.9609, Loss: 0.0309
Epoch  12 Batch  525/538 - Train Accuracy: 0.9840, Validation Accuracy:
0.9748, Loss: 0.0224
Epoch  13 Batch   25/538 - Train Accuracy: 0.9848, Validation Accuracy:
0.9567, Loss: 0.0164
Epoch  13 Batch   50/538 - Train Accuracy: 0.9801, Validation Accuracy:
0.9782, Loss: 0.0127
Epoch  13 Batch   75/538 - Train Accuracy: 0.9849, Validation Accuracy:
0.9702, Loss: 0.0143
Epoch  13 Batch  100/538 - Train Accuracy: 0.9953, Validation Accuracy:
0.9792, Loss: 0.0089
Epoch  13 Batch  125/538 - Train Accuracy: 0.9890, Validation Accuracy:
0.9771, Loss: 0.0136
Epoch  13 Batch  150/538 - Train Accuracy: 0.9918, Validation Accuracy:
0.9766, Loss: 0.0107
Epoch  13 Batch  175/538 - Train Accuracy: 0.9854, Validation Accuracy:
0.9730, Loss: 0.0133
Epoch  13 Batch  200/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9767, Loss: 0.0092
Epoch  13 Batch  225/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9734, Loss: 0.0122
Epoch  13 Batch  250/538 - Train Accuracy: 0.9916, Validation Accuracy:
0.9703, Loss: 0.0104
Epoch  13 Batch  275/538 - Train Accuracy: 0.9891, Validation Accuracy:
0.9812, Loss: 0.0139
Epoch  13 Batch  300/538 - Train Accuracy: 0.9914, Validation Accuracy:
0.9735, Loss: 0.0083
```

```
Epoch  13 Batch  325/538 - Train Accuracy: 0.9916, Validation Accuracy:
0.9716, Loss: 0.0114
Epoch  13 Batch  350/538 - Train Accuracy: 0.9900, Validation Accuracy:
0.9691, Loss: 0.0130
Epoch  13 Batch  375/538 - Train Accuracy: 0.9901, Validation Accuracy:
0.9711, Loss: 0.0078
Epoch  13 Batch  400/538 - Train Accuracy: 0.9918, Validation Accuracy:
0.9744, Loss: 0.0105
Epoch  13 Batch  425/538 - Train Accuracy: 0.9773, Validation Accuracy:
0.9796, Loss: 0.0193
Epoch  13 Batch  450/538 - Train Accuracy: 0.9578, Validation Accuracy:
0.9743, Loss: 0.0157
Epoch  13 Batch  475/538 - Train Accuracy: 0.9872, Validation Accuracy:
0.9828, Loss: 0.0104
Epoch  13 Batch  500/538 - Train Accuracy: 0.9849, Validation Accuracy:
0.9702, Loss: 0.0092
Epoch  13 Batch  525/538 - Train Accuracy: 0.9879, Validation Accuracy:
0.9794, Loss: 0.0137
Epoch  14 Batch   25/538 - Train Accuracy: 0.9848, Validation Accuracy:
0.9773, Loss: 0.0139
Epoch  14 Batch   50/538 - Train Accuracy: 0.9838, Validation Accuracy:
0.9817, Loss: 0.0126
Epoch  14 Batch   75/538 - Train Accuracy: 0.9905, Validation Accuracy:
0.9688, Loss: 0.0090
Epoch  14 Batch  100/538 - Train Accuracy: 0.9969, Validation Accuracy:
0.9764, Loss: 0.0061
Epoch  14 Batch  125/538 - Train Accuracy: 0.9842, Validation Accuracy:
0.9755, Loss: 0.0126
Epoch  14 Batch  150/538 - Train Accuracy: 0.9803, Validation Accuracy:
0.9821, Loss: 0.0107
Epoch  14 Batch  175/538 - Train Accuracy: 0.9924, Validation Accuracy:
0.9709, Loss: 0.0091
Epoch  14 Batch  200/538 - Train Accuracy: 0.9918, Validation Accuracy:
0.9773, Loss: 0.0089
Epoch  14 Batch  225/538 - Train Accuracy: 0.9903, Validation Accuracy:
0.9764, Loss: 0.0109
Epoch  14 Batch  250/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9819, Loss: 0.0086
Epoch  14 Batch  275/538 - Train Accuracy: 0.9893, Validation Accuracy:
0.9755, Loss: 0.0098
Epoch  14 Batch  300/538 - Train Accuracy: 0.9920, Validation Accuracy:
0.9744, Loss: 0.0087
Epoch  14 Batch  325/538 - Train Accuracy: 0.9950, Validation Accuracy:
0.9805, Loss: 0.0061
Epoch  14 Batch  350/538 - Train Accuracy: 0.9816, Validation Accuracy:
0.9707, Loss: 0.0122
Epoch  14 Batch  375/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9794, Loss: 0.0113
Epoch  14 Batch  400/538 - Train Accuracy: 0.9946, Validation Accuracy:
0.9806, Loss: 0.0086
Epoch  14 Batch  425/538 - Train Accuracy: 0.9818, Validation Accuracy:
0.9810, Loss: 0.0195
Epoch  14 Batch  450/538 - Train Accuracy: 0.9736, Validation Accuracy:
0.9778, Loss: 0.0135
Epoch  14 Batch  475/538 - Train Accuracy: 0.9870, Validation Accuracy:
0.9819, Loss: 0.0127
Epoch  14 Batch  500/538 - Train Accuracy: 0.9846, Validation Accuracy:
```

```
0.9755, Loss: 0.0084
Epoch  14 Batch  525/538 - Train Accuracy: 0.9883, Validation Accuracy:
0.9703, Loss: 0.0125
Epoch  15 Batch   25/538 - Train Accuracy: 0.9789, Validation Accuracy:
0.9789, Loss: 0.0114
Epoch  15 Batch   50/538 - Train Accuracy: 0.9861, Validation Accuracy:
0.9783, Loss: 0.0131
Epoch  15 Batch   75/538 - Train Accuracy: 0.9836, Validation Accuracy:
0.9778, Loss: 0.0104
Epoch  15 Batch  100/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9798, Loss: 0.0074
Epoch  15 Batch  125/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9762, Loss: 0.0117
Epoch  15 Batch  150/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9790, Loss: 0.0089
Epoch  15 Batch  175/538 - Train Accuracy: 0.9916, Validation Accuracy:
0.9771, Loss: 0.0102
Epoch  15 Batch  200/538 - Train Accuracy: 0.9902, Validation Accuracy:
0.9782, Loss: 0.0074
Epoch  15 Batch  225/538 - Train Accuracy: 0.9914, Validation Accuracy:
0.9842, Loss: 0.0123
Epoch  15 Batch  250/538 - Train Accuracy: 0.9836, Validation Accuracy:
0.9755, Loss: 0.0119
Epoch  15 Batch  275/538 - Train Accuracy: 0.9854, Validation Accuracy:
0.9643, Loss: 0.0087
Epoch  15 Batch  300/538 - Train Accuracy: 0.9911, Validation Accuracy:
0.9803, Loss: 0.0093
Epoch  15 Batch  325/538 - Train Accuracy: 0.9853, Validation Accuracy:
0.9666, Loss: 0.0154
Epoch  15 Batch  350/538 - Train Accuracy: 0.9805, Validation Accuracy:
0.9696, Loss: 0.0184
Epoch  15 Batch  375/538 - Train Accuracy: 0.9881, Validation Accuracy:
0.9727, Loss: 0.0116
Epoch  15 Batch  400/538 - Train Accuracy: 0.9851, Validation Accuracy:
0.9741, Loss: 0.0131
Epoch  15 Batch  425/538 - Train Accuracy: 0.9788, Validation Accuracy:
0.9711, Loss: 0.0146
Epoch  15 Batch  450/538 - Train Accuracy: 0.9812, Validation Accuracy:
0.9773, Loss: 0.0153
Epoch  15 Batch  475/538 - Train Accuracy: 0.9855, Validation Accuracy:
0.9803, Loss: 0.0091
Epoch  15 Batch  500/538 - Train Accuracy: 0.9858, Validation Accuracy:
0.9696, Loss: 0.0101
Epoch  15 Batch  525/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9741, Loss: 0.0146
Epoch  16 Batch   25/538 - Train Accuracy: 0.9822, Validation Accuracy:
0.9826, Loss: 0.0115
Epoch  16 Batch   50/538 - Train Accuracy: 0.9803, Validation Accuracy:
0.9755, Loss: 0.0111
Epoch  16 Batch   75/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9750, Loss: 0.0102
Epoch  16 Batch  100/538 - Train Accuracy: 0.9930, Validation Accuracy:
0.9764, Loss: 0.0072
Epoch  16 Batch  125/538 - Train Accuracy: 0.9885, Validation Accuracy:
0.9815, Loss: 0.0105
Epoch  16 Batch  150/538 - Train Accuracy: 0.9881, Validation Accuracy:
0.9789, Loss: 0.0089
```

```
Epoch  16 Batch  175/538 - Train Accuracy: 0.9963, Validation Accuracy:
0.9721, Loss: 0.0075
Epoch  16 Batch  200/538 - Train Accuracy: 0.9947, Validation Accuracy:
0.9762, Loss: 0.0067
Epoch  16 Batch  225/538 - Train Accuracy: 0.9926, Validation Accuracy:
0.9728, Loss: 0.0076
Epoch  16 Batch  250/538 - Train Accuracy: 0.9902, Validation Accuracy:
0.9771, Loss: 0.0102
Epoch  16 Batch  275/538 - Train Accuracy: 0.9979, Validation Accuracy:
0.9766, Loss: 0.0079
Epoch  16 Batch  300/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9750, Loss: 0.0106
Epoch  16 Batch  325/538 - Train Accuracy: 0.9879, Validation Accuracy:
0.9805, Loss: 0.0093
Epoch  16 Batch  350/538 - Train Accuracy: 0.9870, Validation Accuracy:
0.9698, Loss: 0.0089
Epoch  16 Batch  375/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9775, Loss: 0.0059
Epoch  16 Batch  400/538 - Train Accuracy: 0.9892, Validation Accuracy:
0.9721, Loss: 0.0078
Epoch  16 Batch  425/538 - Train Accuracy: 0.9860, Validation Accuracy:
0.9709, Loss: 0.0142
Epoch  16 Batch  450/538 - Train Accuracy: 0.9851, Validation Accuracy:
0.9821, Loss: 0.0120
Epoch  16 Batch  475/538 - Train Accuracy: 0.9911, Validation Accuracy:
0.9814, Loss: 0.0089
Epoch  16 Batch  500/538 - Train Accuracy: 0.9920, Validation Accuracy:
0.9666, Loss: 0.0043
Epoch  16 Batch  525/538 - Train Accuracy: 0.9929, Validation Accuracy:
0.9760, Loss: 0.0096
Epoch  17 Batch   25/538 - Train Accuracy: 0.9893, Validation Accuracy:
0.9780, Loss: 0.0105
Epoch  17 Batch   50/538 - Train Accuracy: 0.9877, Validation Accuracy:
0.9806, Loss: 0.0100
Epoch  17 Batch   75/538 - Train Accuracy: 0.9922, Validation Accuracy:
0.9799, Loss: 0.0083
Epoch  17 Batch  100/538 - Train Accuracy: 0.9889, Validation Accuracy:
0.9801, Loss: 0.0066
Epoch  17 Batch  125/538 - Train Accuracy: 0.9864, Validation Accuracy:
0.9865, Loss: 0.0121
Epoch  17 Batch  150/538 - Train Accuracy: 0.9871, Validation Accuracy:
0.9842, Loss: 0.0108
Epoch  17 Batch  175/538 - Train Accuracy: 0.9977, Validation Accuracy:
0.9879, Loss: 0.0061
Epoch  17 Batch  200/538 - Train Accuracy: 0.9971, Validation Accuracy:
0.9771, Loss: 0.0057
Epoch  17 Batch  225/538 - Train Accuracy: 0.9810, Validation Accuracy:
0.9867, Loss: 0.0089
Epoch  17 Batch  250/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9828, Loss: 0.0078
Epoch  17 Batch  275/538 - Train Accuracy: 0.9883, Validation Accuracy:
0.9735, Loss: 0.0085
Epoch  17 Batch  300/538 - Train Accuracy: 0.9942, Validation Accuracy:
0.9803, Loss: 0.0070
Epoch  17 Batch  325/538 - Train Accuracy: 0.9946, Validation Accuracy:
0.9773, Loss: 0.0078
Epoch  17 Batch  350/538 - Train Accuracy: 0.9888, Validation Accuracy:
```

```
0.9830, Loss: 0.0089
Epoch  17 Batch  375/538 - Train Accuracy: 0.9888, Validation Accuracy:
0.9739, Loss: 0.0096
Epoch  17 Batch  400/538 - Train Accuracy: 0.9913, Validation Accuracy:
0.9693, Loss: 0.0080
Epoch  17 Batch  425/538 - Train Accuracy: 0.9913, Validation Accuracy:
0.9700, Loss: 0.0120
Epoch  17 Batch  450/538 - Train Accuracy: 0.9892, Validation Accuracy:
0.9755, Loss: 0.0091
Epoch  17 Batch  475/538 - Train Accuracy: 0.9953, Validation Accuracy:
0.9769, Loss: 0.0060
Epoch  17 Batch  500/538 - Train Accuracy: 0.9988, Validation Accuracy:
0.9831, Loss: 0.0049
Epoch  17 Batch  525/538 - Train Accuracy: 0.9892, Validation Accuracy:
0.9810, Loss: 0.0110
Epoch  18 Batch   25/538 - Train Accuracy: 0.9906, Validation Accuracy:
0.9780, Loss: 0.0113
Epoch  18 Batch   50/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9838, Loss: 0.0077
Epoch  18 Batch   75/538 - Train Accuracy: 0.9864, Validation Accuracy:
0.9757, Loss: 0.0076
Epoch  18 Batch  100/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9824, Loss: 0.0037
Epoch  18 Batch  125/538 - Train Accuracy: 0.9903, Validation Accuracy:
0.9883, Loss: 0.0085
Epoch  18 Batch  150/538 - Train Accuracy: 0.9854, Validation Accuracy:
0.9815, Loss: 0.0059
Epoch  18 Batch  175/538 - Train Accuracy: 0.9922, Validation Accuracy:
0.9755, Loss: 0.0120
Epoch  18 Batch  200/538 - Train Accuracy: 0.9922, Validation Accuracy:
0.9640, Loss: 0.0097
Epoch  18 Batch  225/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9734, Loss: 0.0067
Epoch  18 Batch  250/538 - Train Accuracy: 0.9900, Validation Accuracy:
0.9863, Loss: 0.0091
Epoch  18 Batch  275/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9805, Loss: 0.0075
Epoch  18 Batch  300/538 - Train Accuracy: 0.9918, Validation Accuracy:
0.9778, Loss: 0.0079
Epoch  18 Batch  325/538 - Train Accuracy: 0.9939, Validation Accuracy:
0.9764, Loss: 0.0079
Epoch  18 Batch  350/538 - Train Accuracy: 0.9933, Validation Accuracy:
0.9790, Loss: 0.0105
Epoch  18 Batch  375/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9824, Loss: 0.0073
Epoch  18 Batch  400/538 - Train Accuracy: 0.9926, Validation Accuracy:
0.9766, Loss: 0.0085
Epoch  18 Batch  425/538 - Train Accuracy: 0.9866, Validation Accuracy:
0.9810, Loss: 0.0124
Epoch  18 Batch  450/538 - Train Accuracy: 0.9784, Validation Accuracy:
0.9780, Loss: 0.0116
Epoch  18 Batch  475/538 - Train Accuracy: 0.9913, Validation Accuracy:
0.9798, Loss: 0.0065
Epoch  18 Batch  500/538 - Train Accuracy: 0.9906, Validation Accuracy:
0.9812, Loss: 0.0079
Epoch  18 Batch  525/538 - Train Accuracy: 0.9920, Validation Accuracy:
0.9785, Loss: 0.0106
```

```
Epoch  19 Batch   25/538 - Train Accuracy: 0.9873, Validation Accuracy:
0.9844, Loss: 0.0104
Epoch  19 Batch   50/538 - Train Accuracy: 0.9812, Validation Accuracy:
0.9831, Loss: 0.0111
Epoch  19 Batch   75/538 - Train Accuracy: 0.9972, Validation Accuracy:
0.9753, Loss: 0.0073
Epoch  19 Batch  100/538 - Train Accuracy: 0.9936, Validation Accuracy:
0.9821, Loss: 0.0075
Epoch  19 Batch  125/538 - Train Accuracy: 0.9860, Validation Accuracy:
0.9716, Loss: 0.0107
Epoch  19 Batch  150/538 - Train Accuracy: 0.9900, Validation Accuracy:
0.9822, Loss: 0.0079
Epoch  19 Batch  175/538 - Train Accuracy: 0.9990, Validation Accuracy:
0.9789, Loss: 0.0060
Epoch  19 Batch  200/538 - Train Accuracy: 0.9883, Validation Accuracy:
0.9806, Loss: 0.0065
Epoch  19 Batch  225/538 - Train Accuracy: 0.9944, Validation Accuracy:
0.9815, Loss: 0.0066
Epoch  19 Batch  250/538 - Train Accuracy: 0.9924, Validation Accuracy:
0.9847, Loss: 0.0118
Epoch  19 Batch  275/538 - Train Accuracy: 0.9891, Validation Accuracy:
0.9821, Loss: 0.0098
Epoch  19 Batch  300/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9767, Loss: 0.0090
Epoch  19 Batch  325/538 - Train Accuracy: 0.9970, Validation Accuracy:
0.9812, Loss: 0.0062
Epoch  19 Batch  350/538 - Train Accuracy: 0.9957, Validation Accuracy:
0.9806, Loss: 0.0074
Epoch  19 Batch  375/538 - Train Accuracy: 0.9926, Validation Accuracy:
0.9760, Loss: 0.0045
Epoch  19 Batch  400/538 - Train Accuracy: 0.9874, Validation Accuracy:
0.9792, Loss: 0.0083
Epoch  19 Batch  425/538 - Train Accuracy: 0.9827, Validation Accuracy:
0.9794, Loss: 0.0115
Epoch  19 Batch  450/538 - Train Accuracy: 0.9872, Validation Accuracy:
0.9753, Loss: 0.0115
Epoch  19 Batch  475/538 - Train Accuracy: 0.9931, Validation Accuracy:
0.9835, Loss: 0.0092
Epoch  19 Batch  500/538 - Train Accuracy: 0.9952, Validation Accuracy:
0.9783, Loss: 0.0033
Epoch  19 Batch  525/538 - Train Accuracy: 0.9944, Validation Accuracy:
0.9755, Loss: 0.0079
Epoch  20 Batch   25/538 - Train Accuracy: 0.9922, Validation Accuracy:
0.9872, Loss: 0.0105
Epoch  20 Batch   50/538 - Train Accuracy: 0.9969, Validation Accuracy:
0.9808, Loss: 0.0072
Epoch  20 Batch   75/538 - Train Accuracy: 0.9926, Validation Accuracy:
0.9703, Loss: 0.0090
Epoch  20 Batch  100/538 - Train Accuracy: 0.9918, Validation Accuracy:
0.9824, Loss: 0.0103
Epoch  20 Batch  125/538 - Train Accuracy: 0.9950, Validation Accuracy:
0.9844, Loss: 0.0079
Epoch  20 Batch  150/538 - Train Accuracy: 0.9916, Validation Accuracy:
0.9828, Loss: 0.0073
Epoch  20 Batch  175/538 - Train Accuracy: 0.9961, Validation Accuracy:
0.9796, Loss: 0.0057
Epoch  20 Batch  200/538 - Train Accuracy: 0.9949, Validation Accuracy:
```

```
0.9863, Loss: 0.0052
Epoch  20 Batch  225/538 - Train Accuracy: 0.9900, Validation Accuracy:
0.9759, Loss: 0.0085
Epoch  20 Batch  250/538 - Train Accuracy: 0.9910, Validation Accuracy:
0.9773, Loss: 0.0089
Epoch  20 Batch  275/538 - Train Accuracy: 0.9898, Validation Accuracy:
0.9739, Loss: 0.0057
Epoch  20 Batch  300/538 - Train Accuracy: 0.9927, Validation Accuracy:
0.9826, Loss: 0.0066
Epoch  20 Batch  325/538 - Train Accuracy: 0.9959, Validation Accuracy:
0.9810, Loss: 0.0062
Epoch  20 Batch  350/538 - Train Accuracy: 0.9885, Validation Accuracy:
0.9716, Loss: 0.0098
Epoch  20 Batch  375/538 - Train Accuracy: 0.9927, Validation Accuracy:
0.9762, Loss: 0.0058
Epoch  20 Batch  400/538 - Train Accuracy: 0.9968, Validation Accuracy:
0.9753, Loss: 0.0061
Epoch  20 Batch  425/538 - Train Accuracy: 0.9885, Validation Accuracy:
0.9766, Loss: 0.0122
Epoch  20 Batch  450/538 - Train Accuracy: 0.9896, Validation Accuracy:
0.9746, Loss: 0.0092
Epoch  20 Batch  475/538 - Train Accuracy: 0.9907, Validation Accuracy:
0.9814, Loss: 0.0091
Epoch  20 Batch  500/538 - Train Accuracy: 0.9890, Validation Accuracy:
0.9805, Loss: 0.0059
Epoch  20 Batch  525/538 - Train Accuracy: 0.9901, Validation Accuracy:
0.9718, Loss: 0.0091
Epoch  21 Batch   25/538 - Train Accuracy: 0.9914, Validation Accuracy:
0.9748, Loss: 0.0059
Epoch  21 Batch   50/538 - Train Accuracy: 0.9852, Validation Accuracy:
0.9842, Loss: 0.0073
Epoch  21 Batch   75/538 - Train Accuracy: 0.9940, Validation Accuracy:
0.9730, Loss: 0.0048
Epoch  21 Batch  100/538 - Train Accuracy: 0.9984, Validation Accuracy:
0.9853, Loss: 0.0040
Epoch  21 Batch  125/538 - Train Accuracy: 0.9935, Validation Accuracy:
0.9830, Loss: 0.0077
Epoch  21 Batch  150/538 - Train Accuracy: 0.9902, Validation Accuracy:
0.9853, Loss: 0.0074
Epoch  21 Batch  175/538 - Train Accuracy: 0.9994, Validation Accuracy:
0.9755, Loss: 0.0066
Epoch  21 Batch  200/538 - Train Accuracy: 0.9965, Validation Accuracy:
0.9794, Loss: 0.0073
Epoch  21 Batch  225/538 - Train Accuracy: 0.9931, Validation Accuracy:
0.9817, Loss: 0.0097
Epoch  21 Batch  250/538 - Train Accuracy: 0.9889, Validation Accuracy:
0.9885, Loss: 0.0080
Epoch  21 Batch  275/538 - Train Accuracy: 1.0000, Validation Accuracy:
0.9812, Loss: 0.0063
Epoch  21 Batch  300/538 - Train Accuracy: 0.9922, Validation Accuracy:
0.9798, Loss: 0.0067
Epoch  21 Batch  325/538 - Train Accuracy: 0.9963, Validation Accuracy:
0.9828, Loss: 0.0059
Epoch  21 Batch  350/538 - Train Accuracy: 0.9935, Validation Accuracy:
0.9803, Loss: 0.0046
Epoch  21 Batch  375/538 - Train Accuracy: 0.9935, Validation Accuracy:
0.9806, Loss: 0.0042
```

```
Epoch  21 Batch  400/538 - Train Accuracy: 0.9978, Validation Accuracy:
0.9666, Loss: 0.0048
Epoch  21 Batch  425/538 - Train Accuracy: 0.9887, Validation Accuracy:
0.9721, Loss: 0.0103
Epoch  21 Batch  450/538 - Train Accuracy: 0.9864, Validation Accuracy:
0.9783, Loss: 0.0127
Epoch  21 Batch  475/538 - Train Accuracy: 0.9900, Validation Accuracy:
0.9854, Loss: 0.0092
Epoch  21 Batch  500/538 - Train Accuracy: 0.9945, Validation Accuracy:
0.9838, Loss: 0.0051
Epoch  21 Batch  525/538 - Train Accuracy: 0.9877, Validation Accuracy:
0.9808, Loss: 0.0083
Epoch  22 Batch   25/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9767, Loss: 0.0062
Epoch  22 Batch   50/538 - Train Accuracy: 0.9877, Validation Accuracy:
0.9767, Loss: 0.0063
Epoch  22 Batch   75/538 - Train Accuracy: 0.9967, Validation Accuracy:
0.9755, Loss: 0.0056
Epoch  22 Batch  100/538 - Train Accuracy: 1.0000, Validation Accuracy:
0.9805, Loss: 0.0036
Epoch  22 Batch  125/538 - Train Accuracy: 0.9887, Validation Accuracy:
0.9881, Loss: 0.0089
Epoch  22 Batch  150/538 - Train Accuracy: 0.9941, Validation Accuracy:
0.9890, Loss: 0.0064
Epoch  22 Batch  175/538 - Train Accuracy: 0.9977, Validation Accuracy:
0.9757, Loss: 0.0102
Epoch  22 Batch  200/538 - Train Accuracy: 0.9941, Validation Accuracy:
0.9828, Loss: 0.0061
Epoch  22 Batch  225/538 - Train Accuracy: 0.9927, Validation Accuracy:
0.9870, Loss: 0.0044
Epoch  22 Batch  250/538 - Train Accuracy: 0.9957, Validation Accuracy:
0.9888, Loss: 0.0071
Epoch  22 Batch  275/538 - Train Accuracy: 0.9910, Validation Accuracy:
0.9863, Loss: 0.0062
Epoch  22 Batch  300/538 - Train Accuracy: 0.9955, Validation Accuracy:
0.9799, Loss: 0.0044
Epoch  22 Batch  325/538 - Train Accuracy: 0.9931, Validation Accuracy:
0.9856, Loss: 0.0055
Epoch  22 Batch  350/538 - Train Accuracy: 0.9907, Validation Accuracy:
0.9748, Loss: 0.0065
Epoch  22 Batch  375/538 - Train Accuracy: 0.9913, Validation Accuracy:
0.9837, Loss: 0.0048
Epoch  22 Batch  400/538 - Train Accuracy: 0.9965, Validation Accuracy:
0.9727, Loss: 0.0053
Epoch  22 Batch  425/538 - Train Accuracy: 0.9920, Validation Accuracy:
0.9801, Loss: 0.0107
Epoch  22 Batch  450/538 - Train Accuracy: 0.9888, Validation Accuracy:
0.9764, Loss: 0.0089
Epoch  22 Batch  475/538 - Train Accuracy: 0.9920, Validation Accuracy:
0.9849, Loss: 0.0048
Epoch  22 Batch  500/538 - Train Accuracy: 0.9961, Validation Accuracy:
0.9748, Loss: 0.0047
Epoch  22 Batch  525/538 - Train Accuracy: 0.9914, Validation Accuracy:
0.9751, Loss: 0.0101
Epoch  23 Batch   25/538 - Train Accuracy: 0.9941, Validation Accuracy:
0.9801, Loss: 0.0045
Epoch  23 Batch   50/538 - Train Accuracy: 0.9926, Validation Accuracy:
```

```
0.9803, Loss: 0.0065
Epoch  23 Batch   75/538 - Train Accuracy: 0.9939, Validation Accuracy:
0.9801, Loss: 0.0037
Epoch  23 Batch  100/538 - Train Accuracy: 0.9973, Validation Accuracy:
0.9808, Loss: 0.0056
Epoch  23 Batch  125/538 - Train Accuracy: 0.9978, Validation Accuracy:
0.9821, Loss: 0.0072
Epoch  23 Batch  150/538 - Train Accuracy: 0.9910, Validation Accuracy:
0.9854, Loss: 0.0041
Epoch  23 Batch  175/538 - Train Accuracy: 0.9990, Validation Accuracy:
0.9810, Loss: 0.0039
Epoch  23 Batch  200/538 - Train Accuracy: 0.9895, Validation Accuracy:
0.9771, Loss: 0.0038
Epoch  23 Batch  225/538 - Train Accuracy: 0.9976, Validation Accuracy:
0.9803, Loss: 0.0045
Epoch  23 Batch  250/538 - Train Accuracy: 0.9934, Validation Accuracy:
0.9830, Loss: 0.0080
Epoch  23 Batch  275/538 - Train Accuracy: 0.9961, Validation Accuracy:
0.9808, Loss: 0.0085
Epoch  23 Batch  300/538 - Train Accuracy: 0.9946, Validation Accuracy:
0.9799, Loss: 0.0051
Epoch  23 Batch  325/538 - Train Accuracy: 0.9953, Validation Accuracy:
0.9869, Loss: 0.0041
Epoch  23 Batch  350/538 - Train Accuracy: 0.9950, Validation Accuracy:
0.9805, Loss: 0.0084
Epoch  23 Batch  375/538 - Train Accuracy: 0.9996, Validation Accuracy:
0.9751, Loss: 0.0046
Epoch  23 Batch  400/538 - Train Accuracy: 0.9959, Validation Accuracy:
0.9796, Loss: 0.0061
Epoch  23 Batch  425/538 - Train Accuracy: 0.9931, Validation Accuracy:
0.9757, Loss: 0.0071
Epoch  23 Batch  450/538 - Train Accuracy: 0.9942, Validation Accuracy:
0.9881, Loss: 0.0091
Epoch  23 Batch  475/538 - Train Accuracy: 0.9901, Validation Accuracy:
0.9794, Loss: 0.0086
Epoch  23 Batch  500/538 - Train Accuracy: 0.9980, Validation Accuracy:
0.9831, Loss: 0.0040
Epoch  23 Batch  525/538 - Train Accuracy: 0.9940, Validation Accuracy:
0.9755, Loss: 0.0099
Epoch  24 Batch   25/538 - Train Accuracy: 0.9893, Validation Accuracy:
0.9689, Loss: 0.0097
Epoch  24 Batch   50/538 - Train Accuracy: 0.9900, Validation Accuracy:
0.9750, Loss: 0.0049
Epoch  24 Batch   75/538 - Train Accuracy: 0.9926, Validation Accuracy:
0.9792, Loss: 0.0080
Epoch  24 Batch  100/538 - Train Accuracy: 0.9998, Validation Accuracy:
0.9849, Loss: 0.0048
Epoch  24 Batch  125/538 - Train Accuracy: 0.9968, Validation Accuracy:
0.9833, Loss: 0.0082
Epoch  24 Batch  150/538 - Train Accuracy: 0.9910, Validation Accuracy:
0.9877, Loss: 0.0060
Epoch  24 Batch  175/538 - Train Accuracy: 0.9994, Validation Accuracy:
0.9798, Loss: 0.0048
Epoch  24 Batch  200/538 - Train Accuracy: 0.9969, Validation Accuracy:
0.9776, Loss: 0.0036
Epoch  24 Batch  225/538 - Train Accuracy: 0.9987, Validation Accuracy:
0.9883, Loss: 0.0030
```

```
Epoch  24 Batch  250/538 - Train Accuracy: 0.9939, Validation Accuracy:
0.9838, Loss: 0.0082
Epoch  24 Batch  275/538 - Train Accuracy: 0.9947, Validation Accuracy:
0.9790, Loss: 0.0036
Epoch  24 Batch  300/538 - Train Accuracy: 0.9942, Validation Accuracy:
0.9771, Loss: 0.0052
Epoch  24 Batch  325/538 - Train Accuracy: 0.9998, Validation Accuracy:
0.9805, Loss: 0.0042
Epoch  24 Batch  350/538 - Train Accuracy: 0.9946, Validation Accuracy:
0.9810, Loss: 0.0062
Epoch  24 Batch  375/538 - Train Accuracy: 0.9914, Validation Accuracy:
0.9798, Loss: 0.0048
Epoch  24 Batch  400/538 - Train Accuracy: 0.9953, Validation Accuracy:
0.9789, Loss: 0.0044
Epoch  24 Batch  425/538 - Train Accuracy: 0.9965, Validation Accuracy:
0.9840, Loss: 0.0046
Epoch  24 Batch  450/538 - Train Accuracy: 0.9862, Validation Accuracy:
0.9799, Loss: 0.0094
Epoch  24 Batch  475/538 - Train Accuracy: 0.9980, Validation Accuracy:
0.9782, Loss: 0.0053
Epoch  24 Batch  500/538 - Train Accuracy: 0.9959, Validation Accuracy:
0.9767, Loss: 0.0063
Epoch  24 Batch  525/538 - Train Accuracy: 0.9885, Validation Accuracy:
0.9826, Loss: 0.0105
Model Trained and Saved
```

Save Parameters

Save the `batch_size` and `save_path` parameters for inference.

In [26]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Save parameters for checkpoint
helper.save_params(save_path)
```

Checkpoint

In [27]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import tensorflow as tf
import numpy as np
import helper
import problem_unittests as tests

_, (source_vocab_to_int, target_vocab_to_int), (source_int_to_vocab, tar
get_int_to_vocab) = helper.load_preprocess()
load_path = helper.load_params()
```

Sentence to Sequence

To feed a sentence into the model for translation, you first need to preprocess it. Implement the function `sentence_to_seq()` to preprocess new sentences.

- Convert the sentence to lowercase
- Convert words into ids using `vocab_to_int`
  - Convert words not in the vocabulary, to the `<UNK>` word id.

In [28]:
```
def sentence_to_seq(sentence, vocab_to_int):
    """
    Convert a sentence to a sequence of ids
    :param sentence: String
    :param vocab_to_int: Dictionary to go from the words to an id
    :return: List of word ids
    """
    # TODO: Implement Function
    words = [ word for word in sentence.lower().split(' ') if word]
    word_ids = [ vocab_to_int[word] if word in vocab_to_int.keys() else 
vocab_to_int['<UNK>'] for word in words ]

    return word_ids


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_sentence_to_seq(sentence_to_seq)
```

```
Tests Passed
```

Translate

This will translate `translate_sentence` from English to French.

In [34]:
```
#translate_sentence = 'he saw a old yellow truck .'
#translate_sentence = 'my favorite fruit is apple .'
translate_sentence = 'Paris is beautiful in spring .'

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
translate_sentence = sentence_to_seq(translate_sentence, source_vocab_to
_int)

loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_path + '.meta')
    loader.restore(sess, load_path)

    input_data = loaded_graph.get_tensor_by_name('input:0')
    logits = loaded_graph.get_tensor_by_name('predictions:0')
    target_sequence_length = loaded_graph.get_tensor_by_name('target_seq
uence_length:0')
    source_sequence_length = loaded_graph.get_tensor_by_name('source_seq
uence_length:0')
    keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')

    translate_logits = sess.run(logits, {input_data: [translate_sentence
]*batch_size,
                                         target_sequence_length: [len(tr
anslate_sentence)*2]*batch_size,
                                         source_sequence_length: [len(tr
anslate_sentence)]*batch_size,
                                         keep_prob: 1.0})[0]

print('Input')
print('  Word Ids:      {}'.format([i for i in translate_sentence]))
print('  English Words: {}'.format([source_int_to_vocab[i] for i in tran
slate_sentence]))

print('\nPrediction')
print('  Word Ids:      {}'.format([i for i in translate_logits]))
print('  French Words: {}'.format(" ".join([target_int_to_vocab[i] for i
in translate_logits])))
```

```
INFO:tensorflow:Restoring parameters from checkpoints/dev
Input
  Word Ids:      [4, 140, 178, 229, 142, 196]
  English Words: ['paris', 'is', 'beautiful', 'in', 'spring', '.']

Prediction
  Word Ids:      [182, 259, 48, 201, 47, 346, 1]
  French Words: paris est beau au printemps . <EOS>
```

Imperfect Translation

You might notice that some sentences translate better than others. Since the dataset you're using only has a vocabulary of 227 English words of the thousands that you use, you're only going to see good results using these words. For this project, you don't need a perfect translation. However, if you want to create a better translation model, you'll need better data.

You can train on the WMT10 French-English corpus (http://www.statmt.org/wmt10/training-giga-fren.tar). This dataset has more vocabulary and richer in topics discussed. However, this will take you days to train, so make sure you've a GPU and the neural network is performing well on dataset we provided. Just make sure you play with the WMT10 corpus after you've submitted this project.

Submitting This Project

When submitting this project, make sure to run all the cells before saving the notebook. Save the notebook file as "dlnd_language_translation.ipynb" and save it as a HTML file under "File" -> "Download as". Include the "helper.py" and "problem_unittests.py" files in your submission.