Show and Tell: An Image Caption Generator

Presenter: Hongming Shan
shanh@rpi.edu

NIC: Natural Image Caption

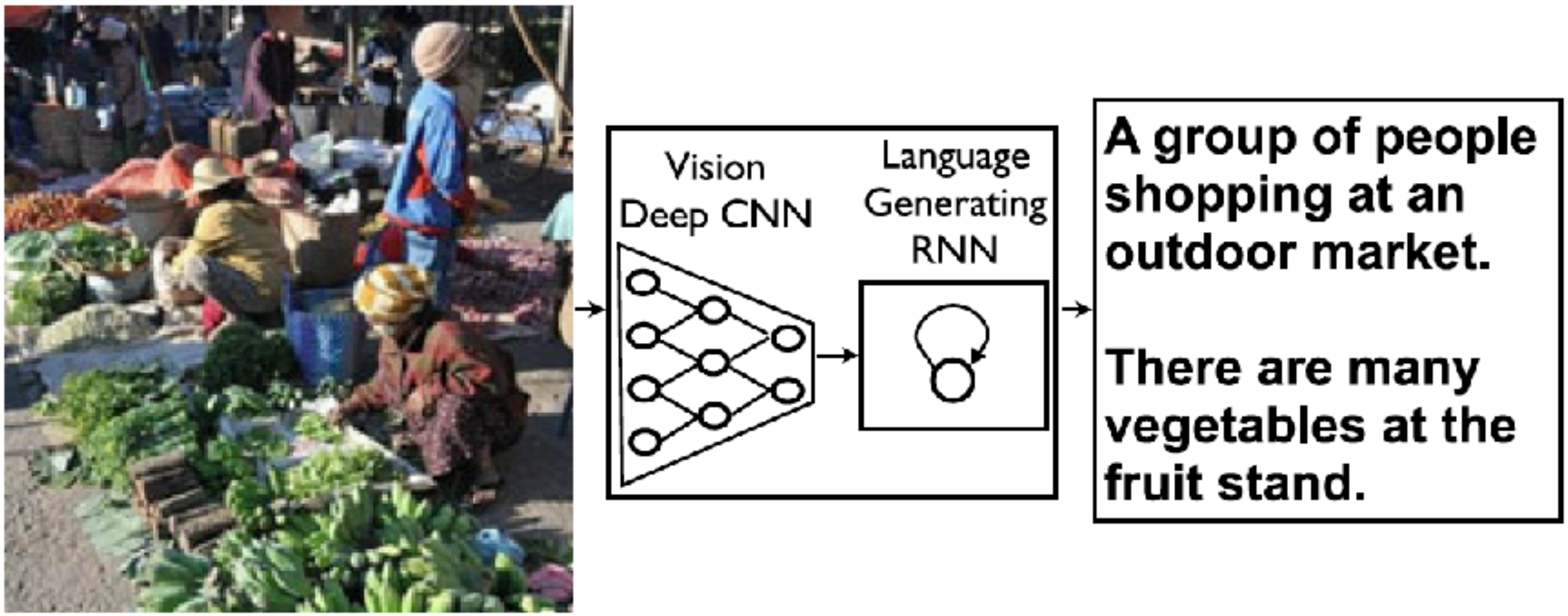


Fig. 1. NIC, our model, is based end-to-end on a neural network consisting of a vision CNN followed by a language generating RNN. It generates complete sentences in natural language from an input image, as shown on the example above.

Machine translation

"Sequence to Sequence Learning with Neural Networks"

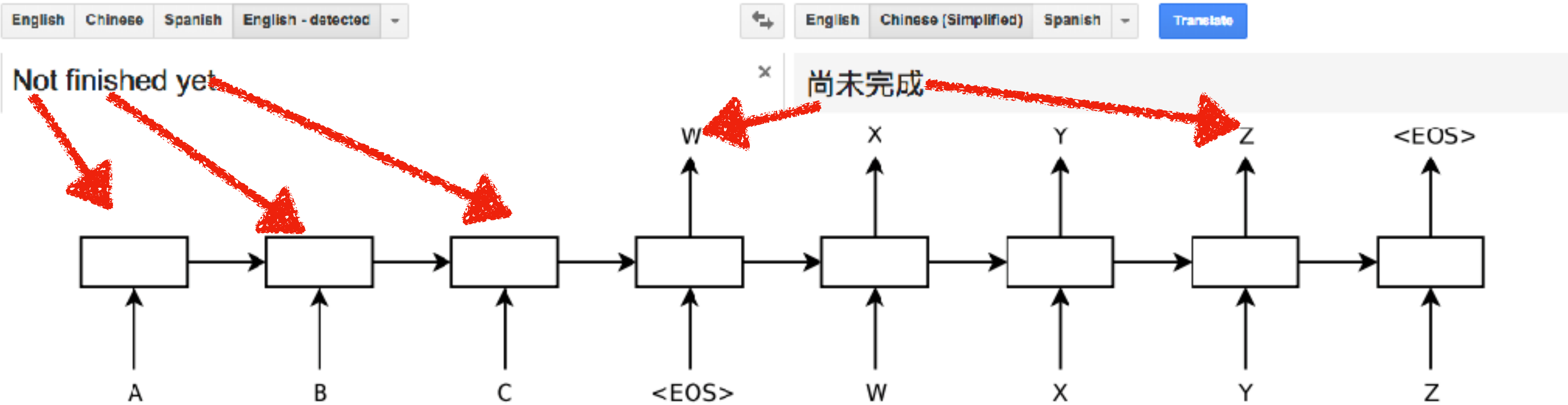


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

The goal of the LSTM is to estimate the conditional probability $p(y_1, \ldots, y_{T'}|x_1, \ldots, x_T)$ where $(x_1, \ldots, x_T)$ is an input sequence and $y_1, \ldots, y_{T'}$ is its corresponding output sequence whose length T' may differ from T. The LSTM computes this conditional probability by first obtaining the fixed-dimensional representation v of the input sequence $(x_1, \ldots, x_T)$ given by the last hidden state of the LSTM, and then computing the probability of $y_1, \ldots, y_{T'}$ with a standard LSTM-LM formulation whose initial hidden state is set to the representation v of $x_1, \ldots, x_T$:

$$p(y_1, \ldots, y_{T'}|x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t|v, y_1, \ldots, y_{t-1}) \quad (1)$$

Same idea for image caption generation

Given an image (instead of input sentence in the source language), one applies the sample principle of “translating” it into its description.

Model

- Maximize the probability of the correct description given the image using the following formulation:

$$\theta^\star = \arg\max_\theta \sum_{(I,S)} \log \; p(S|I;\theta),$$

Model

- *S* represents any sentence, its length is unbounded
- Apply the chain rule to model the joint probability over S_0, …, S_N, N is the length of this particular example

$$\log p\,(S|I) = \sum_{t=0}^{N} \log\ p(S_t|I, S_0, \ldots, S_{t-1}), \qquad (2)$$

Two crucial designs

- Long-Short Term Memory (LSTM)
- CNN: Inception V3 (pre-trained on ImageNet)

LSTM network

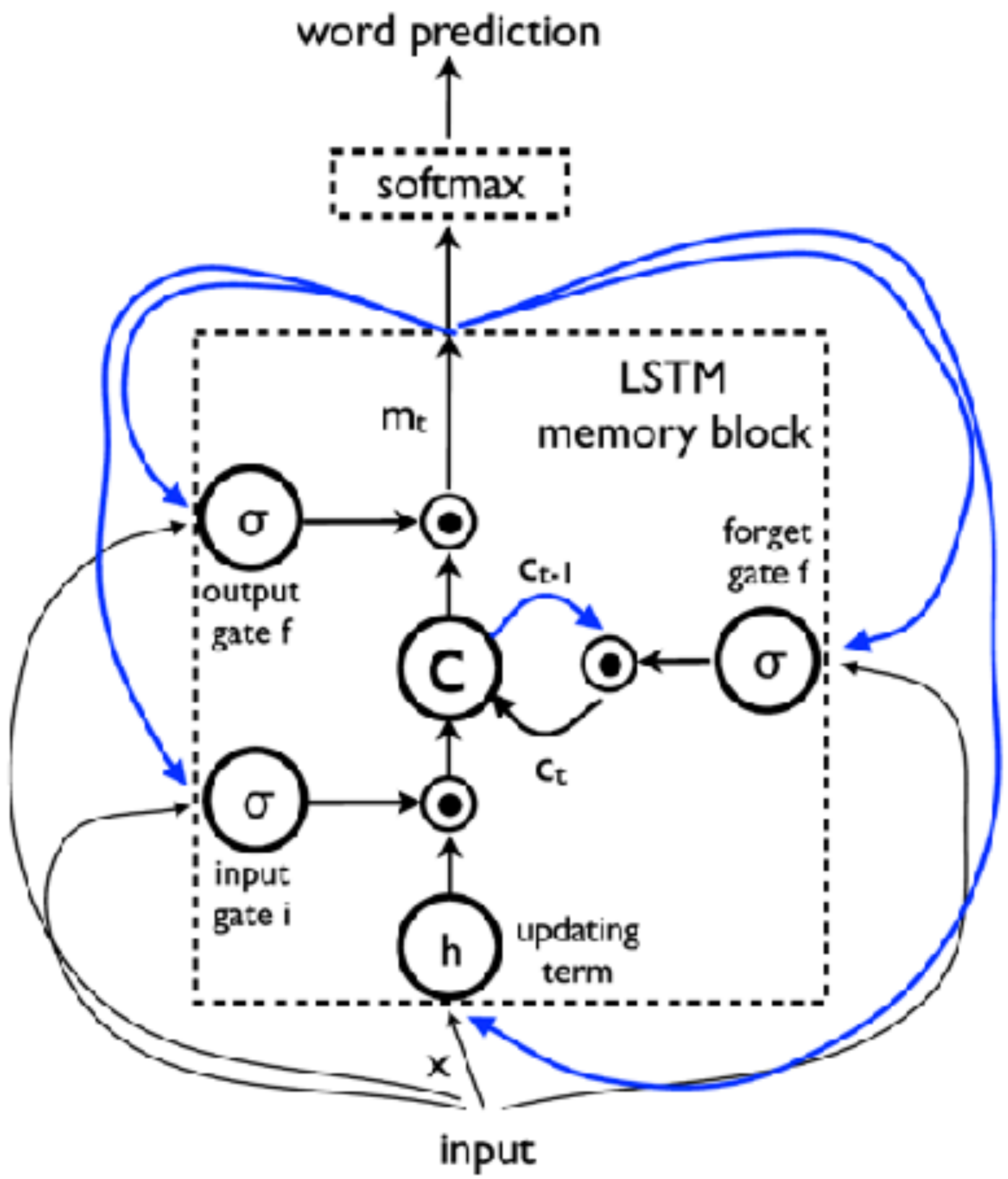


Fig. 2. LSTM: the memory block contains a cell c which is controlled by three gates. In blue we show the recurrent connections—the output m at time $t-1$ is fed back to the memory at time t via the three gates; the cell value is fed back via the forget gate; the predicted word at time $t-1$ is fed back in addition to the memory output m at time t into the Softmax for word prediction.

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1}) \tag{4}$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1}) \tag{5}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1}) \tag{6}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1}) \tag{7}$$

$$m_t = o_t \odot c_t \tag{8}$$

$$p_{t+1} = \text{Softmax}(m_t), \tag{9}$$

Training

$$x_{-1} = \mathrm{CNN}(I) \tag{10}$$

$$x_t = W_e S_t, \quad t \in \{0 \ldots N-1\} \tag{11}$$

$$p_{t+1} = \mathrm{LSTM}(x_t), \quad t \in \{0 \ldots N-1\}, \tag{12}$$

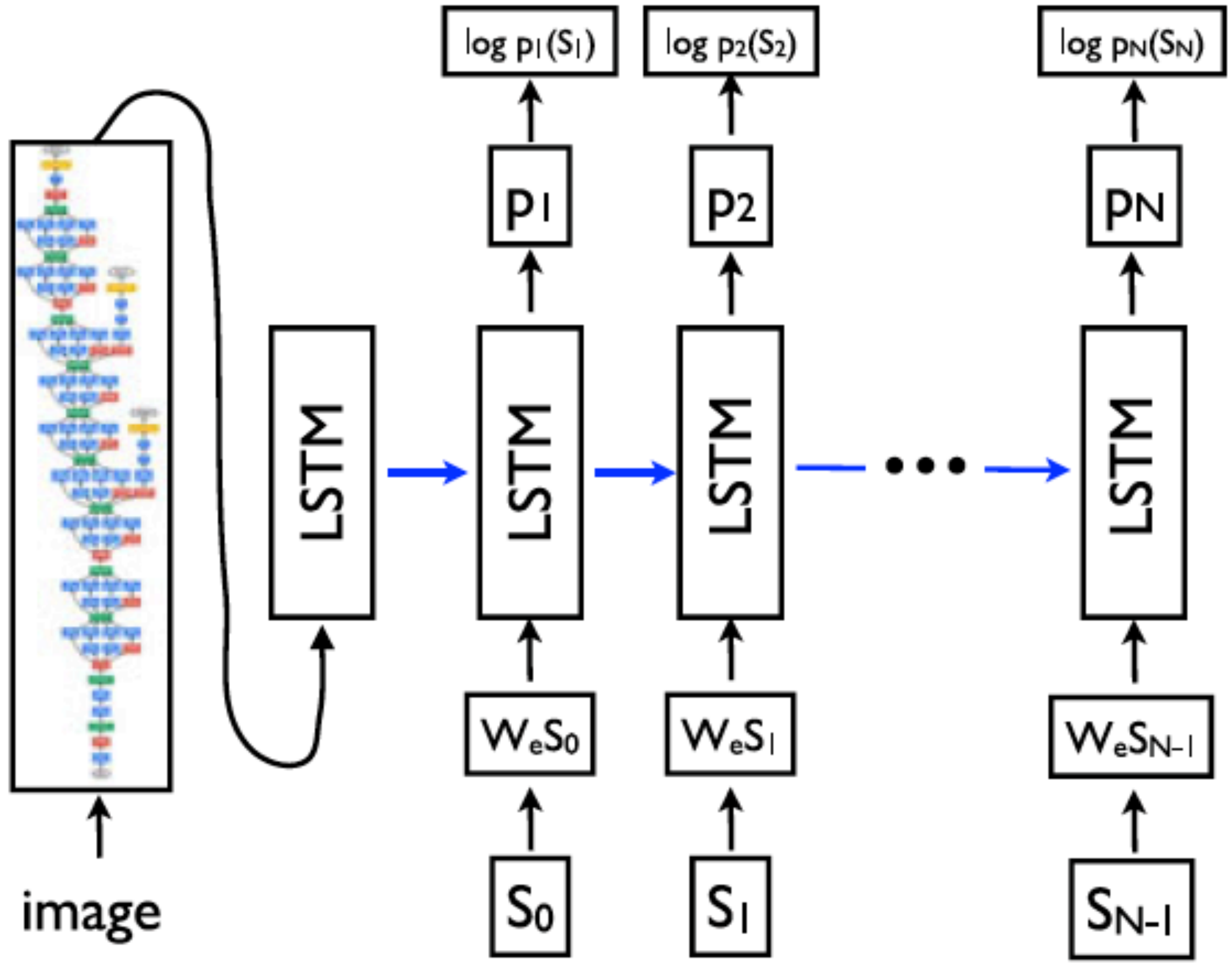


Fig. 3. LSTM model combined with a CNN image embedder (as defined in [24]) and word embeddings. The unrolled connections between the LSTM memories are in blue and they correspond to the recurrent connections in Fig. 2. All LSTMs share the same parameters.

Loss function

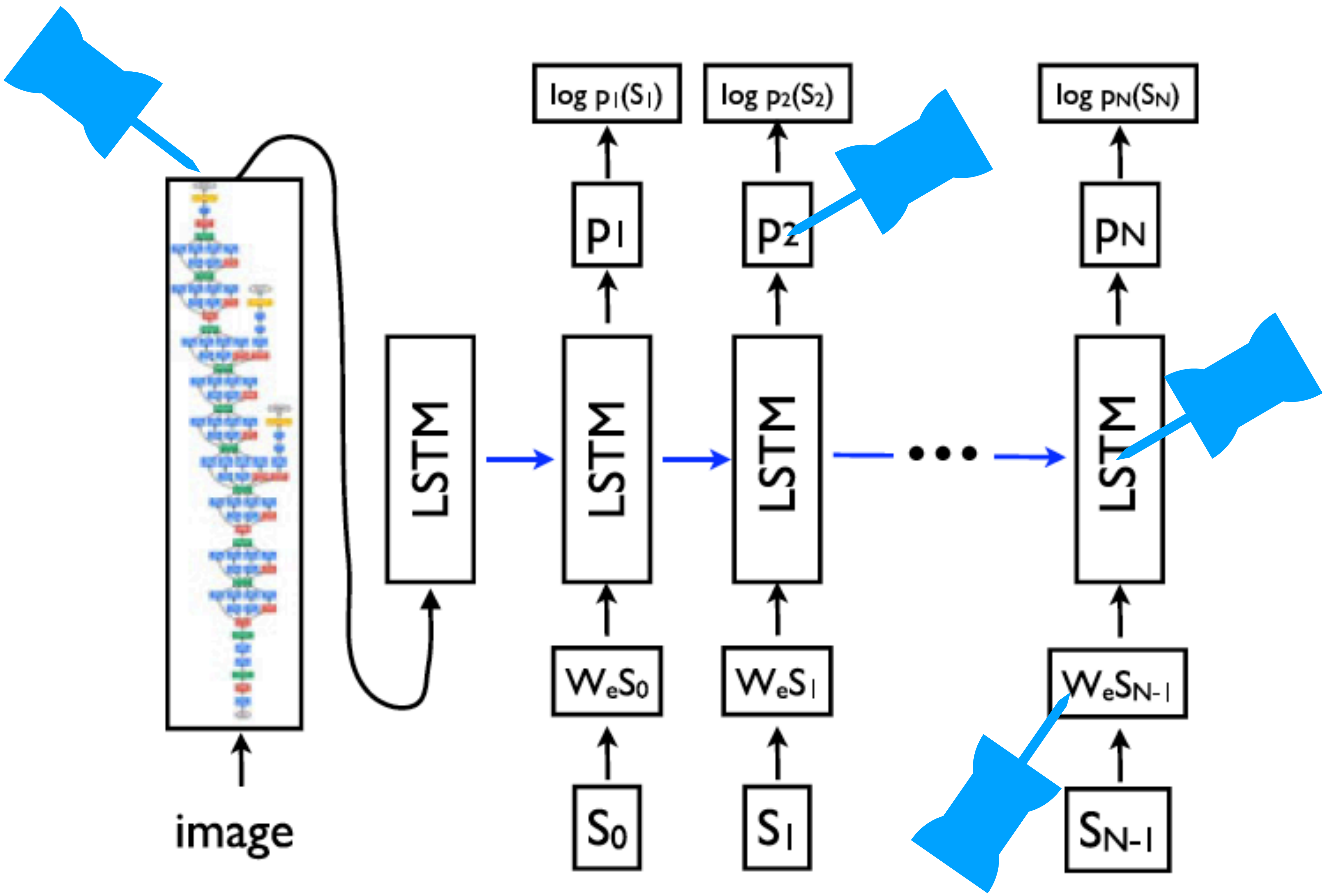
$$L(I, S) = -\sum_{t=1}^{N} \log\, p_t(S_t)\ . \qquad (13)$$

Inference

- Sampling: Sample the first word according to p_1, then provide the corresponding embedding as the input and sample p_2….

- BeamSearch: iteratively consider the set of the k best sentences up to time t as candidates to generate sentence of size t+1, and keep only the resulting best k of them.

Parameters? Dim=512



Results

Metric	BLEU-4	METEOR	CIDER
NIC	**27.7**	**23.7**	**85.5**
Random	4.6	9.0	5.1
Nearest Neighbor	9.9	15.7	36.5
Human	21.7	25.2	85.4

Table 1. Scores on the MSCOCO development set.

Approach	PASCAL (xfer)	Flickr 30k	Flickr 8k	SBU
Im2Text [24]				11
TreeTalk [18]				19
BabyTalk [16]	25			
Tri5Sem [11]			48	
m-RNN [21]		55	58	
MNLM [14][5]		56	51	
SOTA	25	56	58	19
NIC	**59**	**66**	**63**	**28**
Human	69	68	70	

Table 2. BLEU-1 scores. We only report previous work results when available. SOTA stands for the current state-of-the-art.

BLEU: https://en.wikipedia.org/wiki/BLEU


A person riding a motorcycle on a dirt road.
Two dogs play in the grass.
A skateboarder does a trick on a ramp.
A dog is jumping to catch a frisbee.
A group of young people playing a game of frisbee.
Two hockey players are fighting over the puck.
A little girl in a pink hat is blowing bubbles.
A refrigerator filled with lots of food and drinks.
A herd of elephants walking across a dry grass field.
A close up of a cat laying on a couch.
A red motorcycle parked on the side of the road.
A yellow school bus parked in a parking lot.
Describes without errors
Describes with minor errors
Somewhat related to the image
Unrelated to the image

Code:

https://github.com/tensorflow/models/tree/master/research/im2txt

```python
def process_image(self, encoded_image, thread_id=0):
    """Decodes and processes an image string.

    Args:
      encoded_image: A scalar string Tensor; the encoded image.
      thread_id: Preprocessing thread id used to select the ordering of color
        distortions.

    Returns:
      A float32 Tensor of shape [height, width, 3]; the processed image.
    """
    return image_processing.process_image(encoded_image,
                                          is_training=self.is_training(),
                                          height=self.config.image_height,
                                          width=self.config.image_width,
                                          thread_id=thread_id,
                                          image_format=self.config.image_format)
```

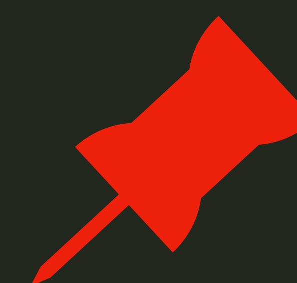
```python
def build_inputs(self):
  """Input prefetching, preprocessing and batching.

  Outputs:
    self.images
    self.input_seqs
    self.target_seqs (training and eval only)
    self.input_mask (training and eval only)
  """
  if self.mode == "inference":
    # In inference mode, images and inputs are fed via placeholders.
    image_feed = tf.placeholder(dtype=tf.string, shape=[], name="image_feed")
    input_feed = tf.placeholder(dtype=tf.int64,
                                shape=[None],  # batch_size
                                name="input_feed")

    # Process image and insert batch dimensions.
    images = tf.expand_dims(self.process_image(image_feed), 0)
    input_seqs = tf.expand_dims(input_feed, 1)

    # No target sequences or input mask in inference mode.
    target_seqs = None
    input_mask = None
```

```python
else:
  # Prefetch serialized SequenceExample protos.
  input_queue = input_ops.prefetch_input_data(
      self.reader,
      self.config.input_file_pattern,
      is_training=self.is_training(),
      batch_size=self.config.batch_size,
      values_per_shard=self.config.values_per_input_shard,
      input_queue_capacity_factor=self.config.input_queue_capacity_factor,
      num_reader_threads=self.config.num_input_reader_threads)

  # Image processing and random distortion. Split across multiple threads
  # with each thread applying a slightly different distortion.
  assert self.config.num_preprocess_threads % 2 == 0
  images_and_captions = []
  for thread_id in range(self.config.num_preprocess_threads):
    serialized_sequence_example = input_queue.dequeue()
    encoded_image, caption = input_ops.parse_sequence_example(
        serialized_sequence_example,
        image_feature=self.config.image_feature_name,
        caption_feature=self.config.caption_feature_name)
    image = self.process_image(encoded_image, thread_id=thread_id)
    images_and_captions.append([image, caption])

  # Batch inputs.
  queue_capacity = (2 * self.config.num_preprocess_threads *
                    self.config.batch_size)
  images, input_seqs, target_seqs, input_mask = (
      input_ops.batch_with_dynamic_pad(images_and_captions,
                                       batch_size=self.config.batch_size,
                                       queue_capacity=queue_capacity))

self.images = images
self.input_seqs = input_seqs
self.target_seqs = target_seqs
self.input_mask = input_mask
```

```python
def batch_with_dynamic_pad(images_and_captions,
                           batch_size,
                           queue_capacity,
                           add_summaries=True):
  """Batches input images and captions.

  This function splits the caption into an input sequence and a target sequence,
  where the target sequence is the input sequence right-shifted by 1. Input and
  target sequences are batched and padded up to the maximum length of sequences
  in the batch. A mask is created to distinguish real words from padding words.

  Example:
    Actual captions in the batch ('-' denotes padded character):
      [
        [ 1 2 3 4 5 ],
        [ 1 2 3 4 - ],
        [ 1 2 3 - - ],
      ]

    input_seqs:
      [
        [ 1 2 3 4 ],
        [ 1 2 3 - ],
        [ 1 2 - - ],
      ]

    target_seqs:
      [
        [ 2 3 4 5 ],
        [ 2 3 4 - ],
        [ 2 3 - - ],
      ]

    mask:
      [
        [ 1 1 1 1 ],
        [ 1 1 1 0 ],
        [ 1 1 0 0 ],
      ]
```

```python
def build_image_embeddings(self):
  """Builds the image model subgraph and generates image embeddings.

  Inputs:
    self.images

  Outputs:
    self.image_embeddings
  """
  inception_output = image_embedding.inception_v3(
      self.images,
      trainable=self.train_inception,
      is_training=self.is_training())
  self.inception_variables = tf.get_collection(
      tf.GraphKeys.GLOBAL_VARIABLES, scope="InceptionV3")

  # Map inception output into embedding space.
  with tf.variable_scope("image_embedding") as scope:
    image_embeddings = tf.contrib.layers.fully_connected(
        inputs=inception_output,
        num_outputs=self.config.embedding_size,
        activation_fn=None,
        weights_initializer=self.initializer,
        biases_initializer=None,
        scope=scope)

  # Save the embedding size in the graph.
  tf.constant(self.config.embedding_size, name="embedding_size")

  self.image_embeddings = image_embeddings
```

```python
def build_seq_embeddings(self):
  """Builds the input sequence embeddings.

  Inputs:
    self.input_seqs

  Outputs:
    self.seq_embeddings
  """
  with tf.variable_scope("seq_embedding"), tf.device("/cpu:0"):
    embedding_map = tf.get_variable(
        name="map",
        shape=[self.config.vocab_size, self.config.embedding_size],
        initializer=self.initializer)
    seq_embeddings = tf.nn.embedding_lookup(embedding_map, self.input_seqs)

  self.seq_embeddings = seq_embeddings
```



```python
def build_model(self):
    """Builds the model.

    Inputs:
      self.image_embeddings
      self.seq_embeddings
      self.target_seqs (training and eval only)
      self.input_mask (training and eval only)

    Outputs:
      self.total_loss (training and eval only)
      self.target_cross_entropy_losses (training and eval only)
      self.target_cross_entropy_loss_weights (training and eval only)
    """
    # This LSTM cell has biases and outputs tanh(new_c) * sigmoid(o), but the
    # modified LSTM in the "Show and Tell" paper has no biases and outputs
    # new_c * sigmoid(o).
```

```python
lstm_cell = tf.contrib.rnn.BasicLSTMCell(
    num_units=self.config.num_lstm_units, state_is_tuple=True)
if self.mode == "train":
  lstm_cell = tf.contrib.rnn.DropoutWrapper(
      lstm_cell,
      input_keep_prob=self.config.lstm_dropout_keep_prob,
      output_keep_prob=self.config.lstm_dropout_keep_prob)

with tf.variable_scope("lstm", initializer=self.initializer) as lstm_scope:
  # Feed the image embeddings to set the initial LSTM state.
  zero_state = lstm_cell.zero_state(
      batch_size=self.image_embeddings.get_shape()[0], dtype=tf.float32)
  _, initial_state = lstm_cell(self.image_embeddings, zero_state)

  # Allow the LSTM variables to be reused.
  lstm_scope.reuse_variables()
```

```python
if self.mode == "inference":
    # In inference mode, use concatenated states for convenient feeding and
    # fetching.
    tf.concat(axis=1, values=initial_state, name="initial_state")

    # Placeholder for feeding a batch of concatenated states.
    state_feed = tf.placeholder(dtype=tf.float32,
                                shape=[None, sum(lstm_cell.state_size)],
                                name="state_feed")
    state_tuple = tf.split(value=state_feed, num_or_size_splits=2, axis=1)

    # Run a single LSTM step.
    lstm_outputs, state_tuple = lstm_cell(
        inputs=tf.squeeze(self.seq_embeddings, axis=[1]),
        state=state_tuple)

    # Concatentate the resulting state.
    tf.concat(axis=1, values=state_tuple, name="state")
else:
    # Run the batch of sequence embeddings through the LSTM.
    sequence_length = tf.reduce_sum(self.input_mask, 1)
    lstm_outputs, _ = tf.nn.dynamic_rnn(cell=lstm_cell,
                                        inputs=self.seq_embeddings,
                                        sequence_length=sequence_length,
                                        initial_state=initial_state,
                                        dtype=tf.float32,
                                        scope=lstm_scope)
```

```
# Stack batches vertically.
lstm_outputs = tf.reshape(lstm_outputs, [-1, lstm_cell.output_size])

with tf.variable_scope("logits") as logits_scope:
  logits = tf.contrib.layers.fully_connected(
      inputs=lstm_outputs,
      num_outputs=self.config.vocab_size,
      activation_fn=None,
      weights_initializer=self.initializer,
      scope=logits_scope)

if self.mode == "inference":
  tf.nn.softmax(logits, name="softmax")
else:
  targets = tf.reshape(self.target_seqs, [-1])
  weights = tf.to_float(tf.reshape(self.input_mask, [-1]))

  # Compute losses.
  losses = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=targets,
                                                          logits=logits)
  batch_loss = tf.div(tf.reduce_sum(tf.multiply(losses, weights)),
                      tf.reduce_sum(weights),
                      name="batch_loss")
  tf.losses.add_loss(batch_loss)
  total_loss = tf.losses.get_total_loss()

  # Add summaries.
  tf.summary.scalar("losses/batch_loss", batch_loss)
  tf.summary.scalar("losses/total_loss", total_loss)
  for var in tf.trainable_variables():
    tf.summary.histogram("parameters/" + var.op.name, var)

  self.total_loss = total_loss
  self.target_cross_entropy_losses = losses  # Used in evaluation.
  self.target_cross_entropy_loss_weights = weights  # Used in evaluation.
```

```python
def setup_inception_initializer(self):
  """Sets up the function to restore inception variables from checkpoint."""
  if self.mode != "inference":
    # Restore inception variables only.
    saver = tf.train.Saver(self.inception_variables)

    def restore_fn(sess):
      tf.logging.info("Restoring Inception variables from checkpoint file %s",
                      self.config.inception_checkpoint_file)
      saver.restore(sess, self.config.inception_checkpoint_file)

    self.init_fn = restore_fn

def setup_global_step(self):
  """Sets up the global step Tensor."""
  global_step = tf.Variable(
      initial_value=0,
      name="global_step",
      trainable=False,
      collections=[tf.GraphKeys.GLOBAL_STEP, tf.GraphKeys.GLOBAL_VARIABLES])

  self.global_step = global_step

def build(self):
  """Creates all ops for training and evaluation."""
  self.build_inputs()
  self.build_image_embeddings()
  self.build_seq_embeddings()
  self.build_model()
  self.setup_inception_initializer()
  self.setup_global_step()
```

Thanks