Real Time American Sign Language Video Captioning using Deep Neural Networks

Syed Tousif Ahmed
BS in Computer Engineering, May 2018
Rochester Institute of Technology

Overview

- Applications
- Video Captioning Architectures
- Implementation Details
- Deployment

Applications

Research at NTID, RIT

Our Team

(clockwise from bottom left):

Anne Alepoudakis

Pamela Francis

Lars Avery

Justin Mahar

Donna Easton

Lisa Elliot

Michael Stinson (P.I.)

Applications

- Messaging app (ASR For Meetings App):
  - Hearing person replies through Automatic Speech Recognition
  - Deaf/Hard of Hearing replies through Video Captioning System
- Automated ASL Proficiency Score
  - ASL learners evaluate their ASL proficiency through the Video Captioning System

Video Captioning Architectures

Sequence to Sequence - Video to Text by Venugopalan et al.

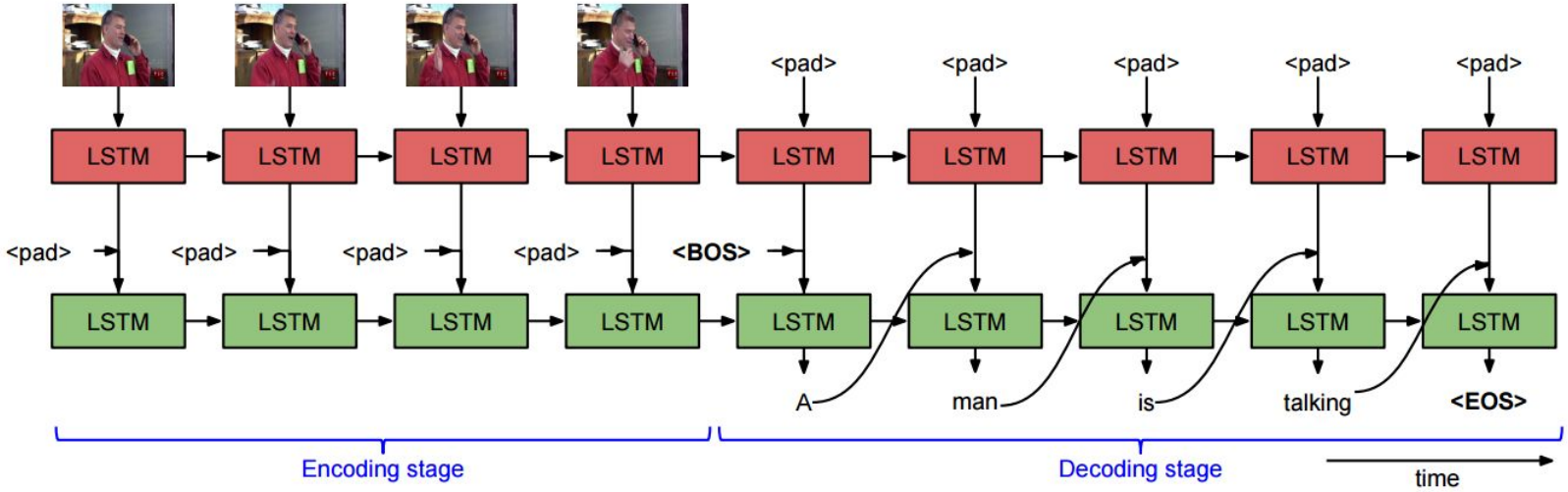


Figure 2. We propose a stack of two LSTMs that learn a representation of a sequence of frames in order to decode it into a sentence that describes the event in the video. The top LSTM layer (colored red) models visual feature inputs. The second LSTM layer (colored green) models language given the text input and the hidden representation of the video sequence. We use <BOS> to indicate begin-of-sentence and <EOS> for the end-of-sentence tag. Zeros are used as a <pad> when there is no input at the time step.

Lip Reading Sentences in the Wild by Chung et al.

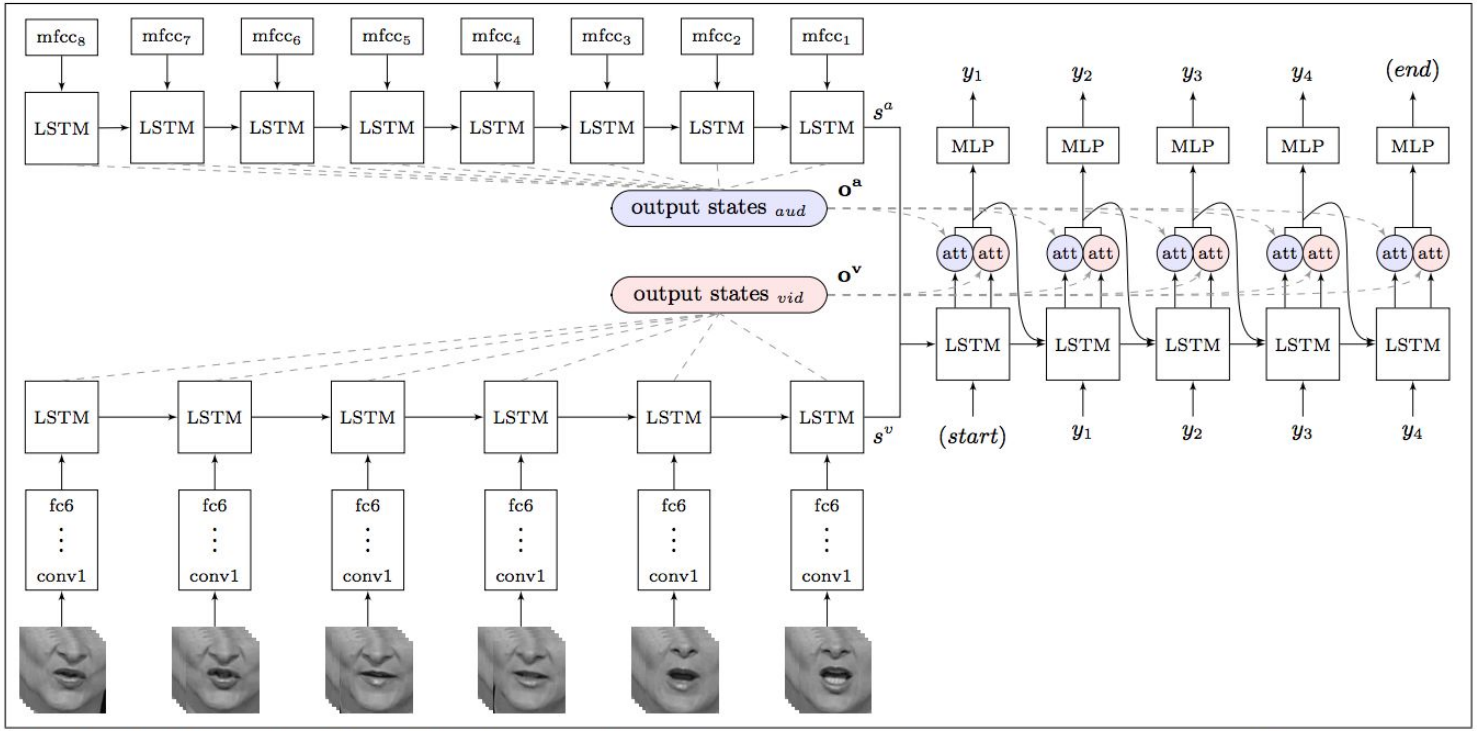


Figure 1. *Watch, Listen, Attend and Spell* architecture. At each time step, the decoder outputs a character y_i, as well as two attention vectors. The attention vectors are used to select the appropriate period of the input visual and audio sequences.

Adaptive Feature Abstraction for Translating Video to Language by Pu et al.

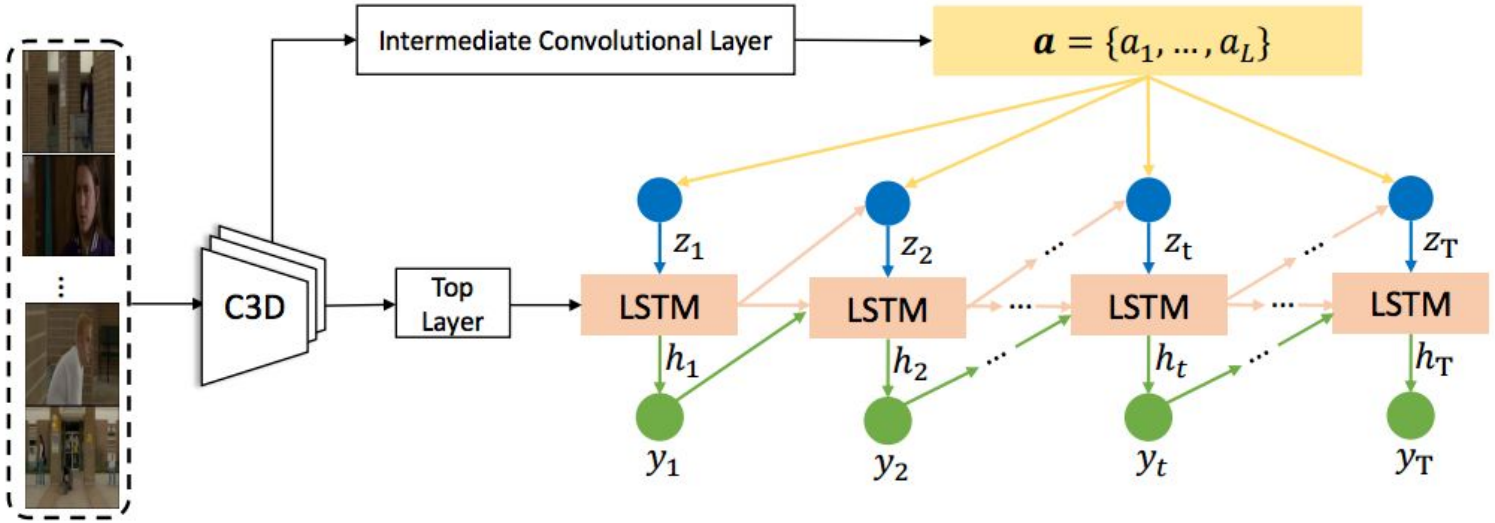


Figure 1: Illustration of our proposed caption-generation model. The model leverages a fully-connected map from the top layer as well as convolutional maps from different mid-level layers of a pretrained 3D convolutional neural network (C3D).

Similarities and Differences

- Encoder-Decoder architecture:
  - Venugopalan encodes RGB frames/Optical flow images in an LSTM layer
  - Chung encodes early fused chunks of grayscale image in an LSTM layer
  - Pu et al. uses C3D
- Using attention mechanism
  - Venugopalan doesn’t use one
- Tips and Tricks
  - Curriculum Learning
  - Scheduled Sampling

Implementation in TensorFlow

Seq2Seq framework by Denny Britz

- A general framework for implementing sequence to sequence models in TensorFlow
- Encoder, Decoder, Attention etc. in their separate modules
- Heavily software engineered
- Link: https://github.com/google/seq2seq
- Changes: https://github.com/syed-ahmed/seq2seq

ASL Text Data Set - C. Zhang and Y. Tian, CCNY

- Sentence-Video Pairs: **17,258 each video about 5 seconds.**
- Vocab with Byte Pair Encoding and 32,000 Merge Operations: **7949**
- Sentence generated from Automatic Speech Recognition in **Youtube CC**
- Data **not clean**.
- TFRecords link: https://github.com/syed-ahmed/ASL-Text-Dataset-TFRecords

6 Step Recipe

1. Tokenize captions and turn them into word vectors. (Seq2Seq)
2. Put captions and videos as sequences in SeqeunceExampleProto and create the TFRecords
3. Create the Data Input Pipeline
4. Create the Model (Seq2Seq)
5. Write the training/evaluation/inference script (Seq2Seq)
6. Deploy

6 Step Recipe

1. Tokenize captions and turn them into word vectors. (Seq2Seq)
2. Put captions and videos as sequences in SeqeunceExampleProto and create the TFRecords
3. Create the Data Input Pipeline
4. Create the Model (Seq2Seq)
5. Write the training/evaluation/inference script (Seq2Seq)
6. Deploy

Raw Video and Caption

Video

Caption

Go out of business.

Tokenizing Captions and BPE

- Tokens are individual elements in a sequence
- Character level tokens: “I love dogs” = [I, L, O, V, E, D, O, G, S, <SPACE>]
- Word level tokens: “I love dogs” = [I, LOVE, DOGS]
- Use tokenizers to split sentences into tokens
- Common tokenizers: **Moses tokenizer.perl script** or libraries such a spaCy, nltk or Stanford Tokenizer.
- Apply **Byte Pair Encoding (BPE)**

https://google.github.io/seq2seq/nmt/#neural-machine-translation-background

Tokenizing Captions and BPE

Follow the script:

https://github.com/google/seq2seq/blob/master/bin/data/wmt16_en_de.sh

```
with all the different religions and churches
or looked into the ocean i &apos; ve realized
was because i was run@@ nin project time which was training
it &apos; s important to note standard weight ranges
and i &apos; m sorry that i could n &apos;t get to you
say something i &apos; m giving up on you
and i &apos; m saying goodbye
you &apos; re the one that i love
has is really serious
say something i &apos; m giving up on you
say something i &apos; m giving up on you
and anywhere i would have followed you
there are many tax credits on page two for which you may qualify
even the causes are un@@ known
it &apos; s interesting there &apos; s no real cure for asthma
`` work incentives planning &amp; assistance &apos; &apos; or wipa project many employment networks
and there are some credits
deaf people to become sign@@ ings cheese she was very interested in that
that will help you reduce the tax on your return
they are thinking - teach deaf children how to survi@@ ve@@ - by finding a menial job
```

6 Step Recipe

1. Tokenize captions and turn them into word vectors. (Seq2Seq)
2. Put captions and videos as sequences in SeqeunceExampleProto and create the TFRecords
3. Create the Data Input Pipeline
4. Create the Model (Seq2Seq)
5. Write the training/evaluation/inference script (Seq2Seq)
6. Deploy

Encoding Video and Text in TFRecords

- SequenceExample consists of context and feature lists
- Context: width, height, channels etc.
- Feature lists: [frame1, frame2, frame3, ...]; [“What”, “does”, “the”, “fox”, “say”]
- Script:

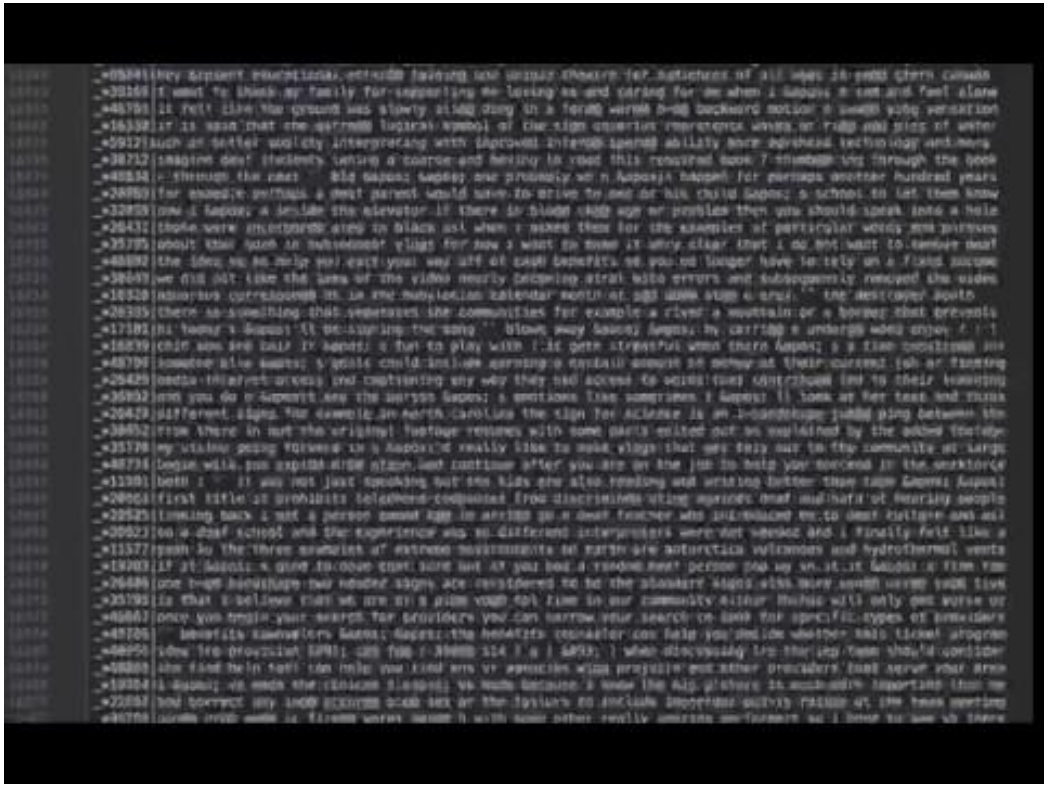
  https://github.com/syed-ahmed/ASL-Text-Dataset-TFRecords/blob/master/build_asl_data.py
- Sequence Example Proto Description:

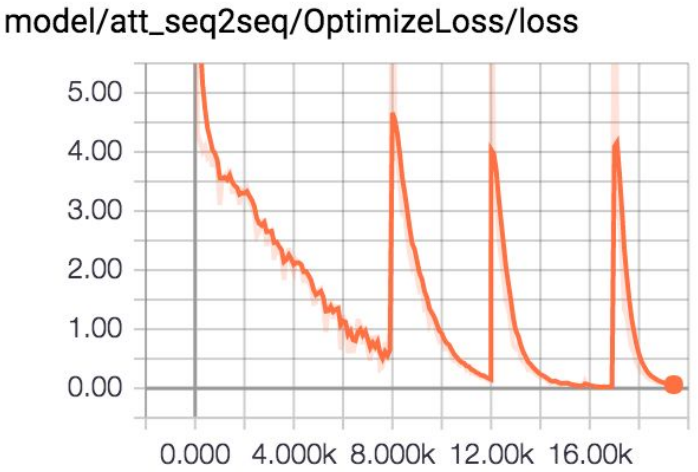
  https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/example/example.proto#L92

```
def _to_sequence_example(video, decoder):
    """Builds a SequenceExample proto for an video-caption pair.

    Args:
      video: An VideoMetadata object.
      decoder: An ImageDecoder object.

    Returns:
      A SequenceExample proto.
    """
    frames = sorted(_find_files(video.filename, "*.jpg"), key=lambda x: int(filter(str.isdigit, x.split("/")[-1])))
    feature = {
        "video/filename": _bytes_feature(video.filename.split("/")[-1]),
        "video/frame_count": _int64_feature(video.frame_count),
        "video/height": _int64_feature(video.height),
        "video/width": _int64_feature(video.width),
        "video/fps": _int64_feature(video.fps),
        "video/duration": _float_feature(video.duration),
        "video/captions": _bytes_feature(video.captions),
    }
    context = tf.train.Features(feature=feature)
    frames_encoded = []
    for idx, val in enumerate(frames):
        with tf.gfile.FastGFile(val, "r") as f:
            encoded_image = f.read()
        try:
            decoder.decode_jpeg(encoded_image)
            frames_encoded.append(encoded_image)
        except (tf.errors.InvalidArgumentError, AssertionError):
            print("Skipping file with invalid JPEG data: %s" % val)
            return

    feature_lists = tf.train.FeatureLists(feature_list={
        "video/frames": _bytes_feature_list(frames_encoded)

    })
    sequence_example = tf.train.SequenceExample(
        context=context, feature_lists=feature_lists)

    return sequence_example
```

```python
def _bytes_feature(value):
    """Wrapper for inserting a bytes Feature into a SequenceExample proto."""
    if type(value) is unicode:
        return tf.train.Feature(bytes_list=tf.train.BytesList(value=[str(value.encode('utf-8'))]))
    else:
        return tf.train.Feature(bytes_list=tf.train.BytesList(value=[str(value)]))


def _bytes_feature_list(values):
    """Wrapper for inserting a bytes FeatureList into a SequenceExample proto."""
    return tf.train.FeatureList(feature=[_bytes_feature(v) for v in values])
```

Curriculum Learning





6 Step Recipe

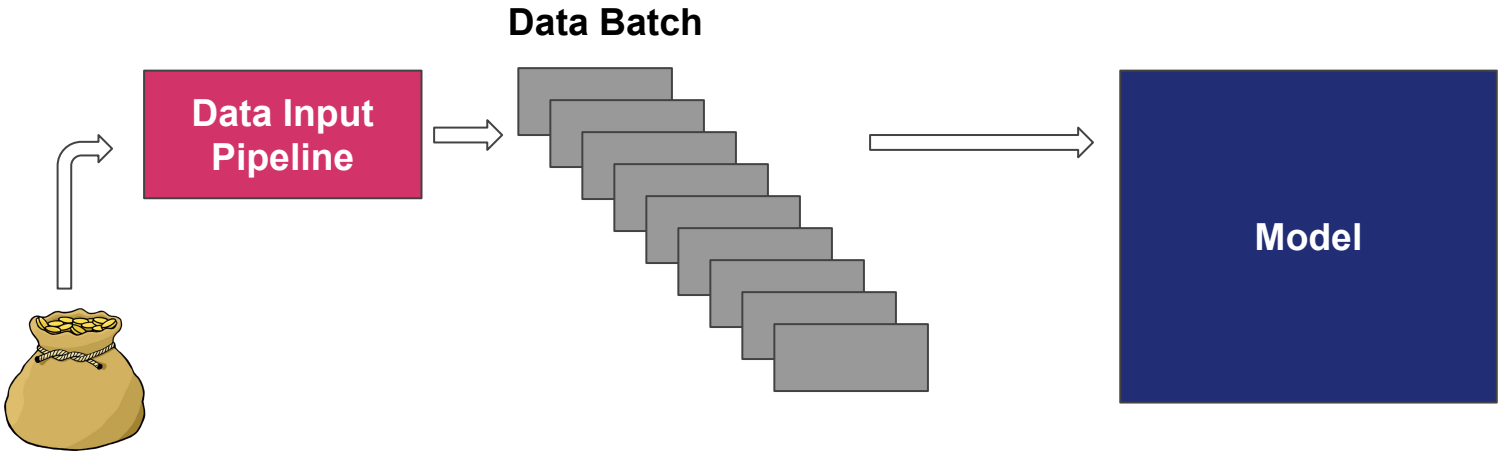
1. Tokenize captions and turn them into word vectors. (Seq2Seq)
2. Put captions and videos as sequences in SeqeunceExampleProto and create the TFRecords
3. Create the Data Input Pipeline
4. Create the Model (Seq2Seq)
5. Write the training/evaluation/inference script (Seq2Seq)
6. Deploy

TensorFlow Queues

- Keywords: Queue Runner, Producer Queue, Consumer Queue, Coordinator
- Key concepts that streamlines data fetching


```
q = tf.FIFOQueue(3, "float")
init = q.enqueue_many(([0.,0.,0.],))

x = q.dequeue()
y = x+1
q_inc = q.enqueue([y])

init.run()
q_inc.run()
q_inc.run()
q_inc.run()
q_inc.run()
```

Producer-Consumer Pattern



Parsing Data from TFRecords

1. Create a list of TFRecord file names:

```
data_files = []
for pattern in self.params["file_input_pattern"].split(","):
    data_files.extend(tf.gfile.Glob(pattern))
```

2. Create a string input producer:

```
filename_queue = tf.train.string_input_producer(
    data_files, shuffle=True, capacity=16)
```

Parsing Data from TFRecords

3. Create the Input Random Shuffle Queue

```
values_queue = tf.RandomShuffleQueue(
    capacity=capacity,
    min_after_dequeue=min_queue_examples,
    dtypes=[tf.string],
    name="random_" + value_queue_name)
```

4. Fill it with the serialized data from TFRecords

```
    _, value = reader.read(filename_queue)
    enqueue_ops.append(values_queue.enqueue([value]))
tf.train.queue_runner.add_queue_runner(tf.train.queue_runner.QueueRunner(
    values_queue, enqueue_ops))
```

Parsing Data from TFRecords

5. Parse the caption and jpeg encoded video frames

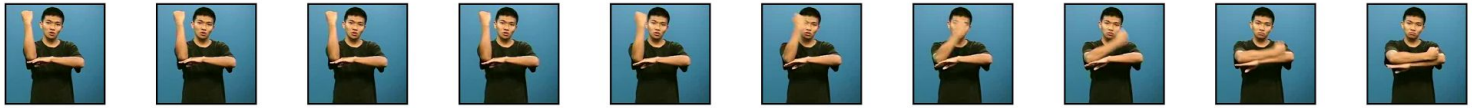
```
context, sequence = tf.parse_single_sequence_example(
    serialized,
    context_features={
        caption_feature: tf.FixedLenFeature([], dtype=tf.string)
    },
    sequence_features={
        video_feature: tf.FixedLenSequenceFeature([], dtype=tf.string),
    })

caption = context[caption_feature]
encoded_video = sequence[video_feature]
```

Using tf.map_fn for Video Processing

Raw [10x240x320x3]
Dtype Conversion

```
tf.map_fn(lambda x: tf.image.convert_image_dtype(x,
    dtype=tf.float32), video, dtype=tf.float32)
```

Crop [10x240x320x3]
Resize [10x120x120x3]



Brightness
[10x120x120x3]

Saturation
[10x120x120x3]

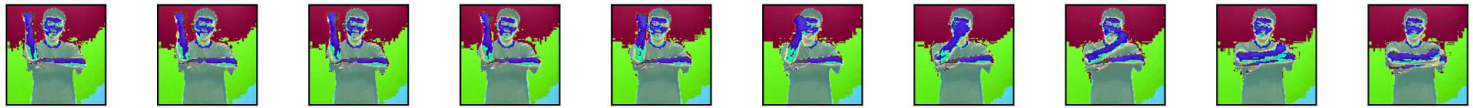
Hue
[10x120x120x3]

Data Processing, Augmentation and Early Fusion

Hue
[10x120x120x3]

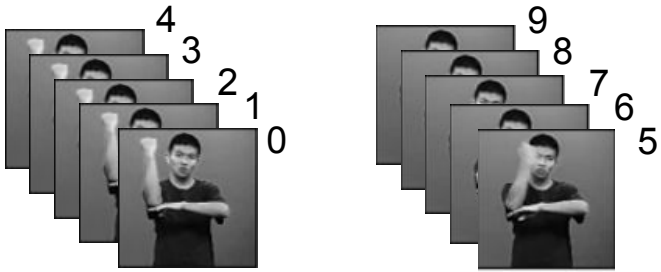
Contrast
[10x120x120x3]

Normalization
[10x120x120x3]



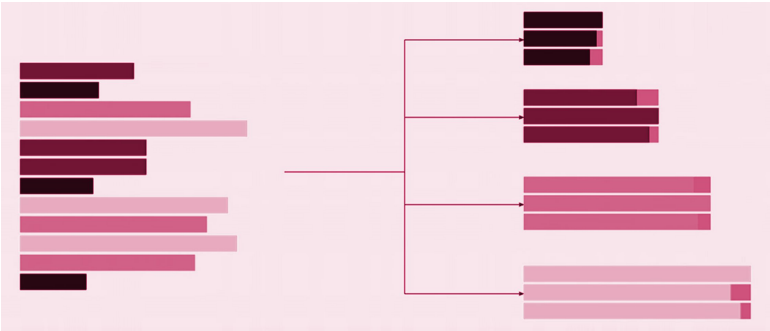
Grayscale
[10x120x120x1]

Early Fusion (reshape+concat)
[2x5x120x120x1]
[2x120x120x5]



Bucket by Sequence Length

- Sequences are of variable length
- Need to pad the sequences
- Solution: Bucketing



```
_, batch = tf.contrib.training.bucket_by_sequence_length(
    input_length=features_and_labels["source_len"],
    bucket_boundaries=bucket_boundaries,
    tensors=features_and_labels,
    batch_size=batch_size,
    keep_input=features_and_labels["source_len"] >= 1,
    dynamic_pad=True,
    capacity=5000 + 16 * batch_size,
    allow_smaller_final_batch=allow_smaller_final_batch,
    name="bucket_queue")
```

Before:

```
features_and_labels = {dict} {u'target_tokens': <tf.Tensor 'input_fn/concat_1:0' shape=(?,) dtype=string>, u'target_len': <tf.Tensor '
  __len__ = {int} 4
▶ u'source_len' (4837144592) = {Tensor} Tensor("input_fn/strided_slice:0", shape=(), dtype=int32)
▶ u'source_tokens' (4837144496) = {Tensor} Tensor("input_fn/VGG-M/logits/flatten/Reshape:0", shape=(?, 512), dtype=float32)
▶ u'target_len' (4837144160) = {Tensor} Tensor("input_fn/Size_1:0", shape=(), dtype=int32)
▶ u'target_tokens' (4837144112) = {Tensor} Tensor("input_fn/concat_1:0", shape=(?,), dtype=string)
```

After:

```
batch = {dict} {u'target_len': <tf.Tensor 'input_fn/bucket_queue/bucket/dequeue_top:4' shape=(32,) dtype=int32>, u'source_len': <tf.Tenso
  __len__ = {int} 4
▶ u'source_len' (4649515024) = {Tensor} Tensor("input_fn/bucket_queue/bucket/dequeue_top:2", shape=(32,), dtype=int32)
▶ u'source_tokens' (4649514928) = {Tensor} Tensor("input_fn/bucket_queue/bucket/dequeue_top:3", shape=(32, ?, 512), dtype=float32)
▶ u'target_len' (4649514592) = {Tensor} Tensor("input_fn/bucket_queue/bucket/dequeue_top:4", shape=(32,), dtype=int32)
▶ u'target_tokens' (4649514544) = {Tensor} Tensor("input_fn/bucket_queue/bucket/dequeue_top:5", shape=(32, ?), dtype=string)
```

6 Step Recipe

1. Tokenize captions and turn them into word vectors. (Seq2Seq)
2. Put captions and videos as seqeunces in SeqeunceExampleProto and create the TFRecords
3. Create the Data Input Pipeline
4. Create the Model (Seq2Seq)
5. Write the training/evaluation/inference script (Seq2Seq)
6. Deploy

Seq2Seq Summary

- Encoder takes an embedding as an input. For instance: our video embedding is of shape (batch size, sequence length, 512)
- Decoder takes last state of the encoder
- Attention mechanism calculates attention function on the encoder outputs

ASL Model Summary

- Encoder-Decoder Architecture
- VGG-M encodes **early fused grayscale** frames (sliding windows of 5 frames)
- **2 Layer** RNN with **512 LSTM** units in the **Encoder**
- **2 Layer** RNN with **512 LSTM** units in the **Decoder**
- Decoder uses **attention** mechanism from **Bahdanau** et al.

VGG-M/conv1/BatchNorm/beta (96, 96/96 params)
VGG-M/conv1/weights (3x3x5x96, 4.32k/4.32k params)
VGG-M/conv2/BatchNorm/beta (256, 256/256 params)
VGG-M/conv2/weights (3x3x96x256, 221.18k/221.18k params)
VGG-M/conv3/BatchNorm/beta (512, 512/512 params)
VGG-M/conv3/weights (3x3x256x512, 1.18m/1.18m params)
VGG-M/conv4/BatchNorm/beta (512, 512/512 params)
VGG-M/conv4/weights (3x3x512x512, 2.36m/2.36m params)
VGG-M/conv5/BatchNorm/beta (512, 512/512 params)
VGG-M/conv5/weights (3x3x512x512, 2.36m/2.36m params)
VGG-M/fc6/BatchNorm/beta (512, 512/512 params)
VGG-M/fc6/weights (6x6x512x512, 9.44m/9.44m params)

34.21 million parameters

model/att_seq2seq/Variable (1, 1/1 params)
model/att_seq2seq/decode/attention/att_keys/biases (512, 512/512 params)
model/att_seq2seq/decode/attention/att_keys/weights (512x512, 262.14k/262.14k params)
model/att_seq2seq/decode/attention/att_query/biases (512, 512/512 params)
model/att_seq2seq/decode/attention/att_query/weights (512x512, 262.14k/262.14k params)
model/att_seq2seq/decode/attention/v_att (512, 512/512 params)
model/att_seq2seq/decode/attention_decoder/decoder/attention_mix/biases (512, 512/512 params)
model/att_seq2seq/decode/attention_decoder/decoder/attention_mix/weights (1024x512, 524.29k/524.29k params)
model/att_seq2seq/decode/attention_decoder/decoder/extended_multi_rnn_cell/cell_0/lstm_cell/biases (2048, 2.05k/2.05k params)
model/att_seq2seq/decode/attention_decoder/decoder/extended_multi_rnn_cell/cell_0/lstm_cell/weights (1536x2048, 3.15m/3.15m params)
model/att_seq2seq/decode/attention_decoder/decoder/extended_multi_rnn_cell/cell_1/lstm_cell/biases (2048, 2.05k/2.05k params)
model/att_seq2seq/decode/attention_decoder/decoder/extended_multi_rnn_cell/cell_1/lstm_cell/weights (1024x2048, 2.10m/2.10m params)
model/att_seq2seq/decode/attention_decoder/decoder/logits/biases (7952, 7.95k/7.95k params)
model/att_seq2seq/decode/attention_decoder/decoder/logits/weights (512x7952, 4.07m/4.07m params)
model/att_seq2seq/decode/target_embedding/W (7952x512, 4.07m/4.07m params)
model/att_seq2seq/encode/forward_rnn_encoder/rnn/extended_multi_rnn_cell/cell_0/lstm_cell/biases (2048, 2.05k/2.05k params)
model/att_seq2seq/encode/forward_rnn_encoder/rnn/extended_multi_rnn_cell/cell_0/lstm_cell/weights (1024x2048, 2.10m/2.10m params)
model/att_seq2seq/encode/forward_rnn_encoder/rnn/extended_multi_rnn_cell/cell_1/lstm_cell/biases (2048, 2.05k/2.05k params)
model/att_seq2seq/encode/forward_rnn_encoder/rnn/extended_multi_rnn_cell/cell_1/lstm_cell/weights (1024x2048, 2.10m/2.10m params)

Train using tf.Estimator and tf.Experiment

```
estimator = tf.contrib.learn.Estimator(
    model_fn=model_fn,
    model_dir=output_dir,
    config=config,
    params=FLAGS.model_params)
```

```
experiment = PatchedExperiment(
    estimator=estimator,
    train_input_fn=train_input_fn,
    eval_input_fn=eval_input_fn,
    min_eval_frequency=FLAGS.eval_every_n_steps,
    train_steps=FLAGS.train_steps,
    eval_steps=None,
    eval_metrics=eval_metrics,
    train_monitors=train_hooks)
```

6 Step Recipe

1. Tokenize captions and turn them into word vectors. (Seq2Seq)
2. Put captions and videos as seqeunces in SeqeunceExampleProto and create the TFRecords
3. Create the Data Input Pipeline
4. Create the Model (Seq2Seq)
5. Write the training/evaluation/inference script (Seq2Seq)
6. Deploy

NVIDIA Jetson TX2

- Install TensorFlow: https://syed-ahmed.gitbooks.io/nvidia-jetson-tx2-recipes/content/first-question.html

- USB camera using CUDA V4L2 Driver

- Put graph in GPU

- TensorFlow XLA can potentially speed up application

Thank you!

Email: syed.ahmed.emails@gmail.com
Twitter: @tousifsays