Contents

1 Youtube-8M Starter Code

1.1 Project Structure (Core Part)

1.1.1 Train (Model):

1. Related Files

    (a) Processing Utility

    - train.py: The primary script for training models.
    - losses.py: Contains definitions for loss functions.
    - export_model.py: Provides a class to export a model during training for later use in batch prediction.
    - readers.py: Contains definitions for the Video dataset and Frame dataset readers.

    (b) Models

    i. Model Utility

    - models.py: Base class for defining a model. (common interface) - model_util.py: Must implement to define a model

    ii. Model Processing Logic

    - video_level_models.py: take whole video (agreegated features) as input
    - frame_level_models.py: take frame level features as input

2. Model Analysis

    take video_level_models as an example:

    it contains two sub models inside of it

- `LogisticModel`

```
class LogisticModel(models.BaseModel):

  def create_model(self, model_input, vocab_size, l2_penalty=1e-8,
**unused_params):
    output = slim.fully_connected(
        model_input, vocab_size, activation_fn=tf.nn.sigmoid,
        weights_regularizer=slim.l2_regularizer(l2_penalty))
    return {"predictions": output}
```

Analysis for this model

- Input: matrix of input features/ number of classes in the dataset
  - How to set up the input:
    By chaning the –train_data_pattern flag, we can specify smaller data set.
    To be more specific, using the following command
    ```
    python train.py
    --train_data_pattern='/path/to/features/train*.tfrecord'
    --model=LogisticModel --train_dir=$MODEL_DIR/video_level_logistic_model
    ```
- Output:A dictionary with a tensor containing the probability predictions of the model in the 'predictions' key.
  - How to save the output:
    By chaning the –train_dir, we can specify where to store the result
    ```
    python train.py
    --train_data_pattern='/path/to/features/train*.tfrecord'
    --model=LogisticModel --train_dir=$MODEL_DIR/video_level_logistic_model
    ```
- Processing Model: slim.fully_connected from tensorflow. A specific layer from neural network. Other layers

Layer	TF-Slim
BiasAdd	slim.bias_add
BatchNorm	slim.batch_norm
Conv2d	slim.conv2d
Conv2dInPlane	slim.conv2d_in_plane
Conv2dTranspose (Deconv)	slim.conv2d_transpose
AvgPool2D	slim.avg_pool2d
Dropout	slim.dropout

- MoeModel

```
class MoeModel(models.BaseModel):

  def create_model(self,
                   model_input,
                   vocab_size,
                   num_mixtures=None,
                   l2_penalty=1e-8,
                   **unused_params):
    num_mixtures = num_mixtures or FLAGS.moe_num_mixtures

    gate_activations = slim.fully_connected(
        model_input,
        vocab_size * (num_mixtures + 1),
        activation_fn=None,
        biases_initializer=None,
        weights_regularizer=slim.l2_regularizer(l2_penalty),
        scope="gates")
    expert_activations = slim.fully_connected(
        model_input,
        vocab_size * num_mixtures,
        activation_fn=None,
        weights_regularizer=slim.l2_regularizer(l2_penalty),
        scope="experts")

    gating_distribution = tf.nn.softmax(tf.reshape(
        gate_activations,
        [-1, num_mixtures + 1]))
    expert_distribution = tf.nn.sigmoid(tf.reshape(
        expert_activations,
        [-1, num_mixtures]))

    final_probabilities_by_class_and_batch = tf.reduce_sum(
        gating_distribution[:, :num_mixtures] * expert_distribution, 1)
    final_probabilities = tf.reshape(final_probabilities_by_class_and_batch,
                                     [-1, vocab_size])
    return {"predictions": final_probabilities}
```

How to build our own model

(a) The model should inherit `models.BaseModel`

(b) Specify Input from command

(c) Output should satisfy the format: `return {"predictions": final_probabilities}`

1.1.2 Evaluation

We can use this part directly

1. Related Files

   - eval.py: The primary script for evaluating models.
   - eval_util.py: Provides a class that calculates all evaluation metrics.
   - average_precision_calculator.py: Functions for calculating average precision.
   - mean_average_precision_calculator.py: Functions for calculating mean average precision.

2. How to use them

   Through command line:

   To evaluate the model, run

   ```
   python eval.py --eval_data_pattern='/path/to/features/validate*.tfrecord'
   --model=LogisticModel
   --train_dir=$MODEL_DIR/video_level_logistic_model --run_once=True
   ```

   As the model is training or evaluating, you can view the results on tensorboard by running

   ```
   tensorboard --logdir=$MODEL_DIR
   ```

   and navigating to `http://localhost:6006` in your web browser.

   When you are happy with your model, you can generate a csv file of predictions from it by running

   ```
   python inference.py
   --output_file=$MODEL_DIR/video_level_logistic_model/predictions.csv
   --input_data_pattern='/path/to/features/test*.tfrecord'
   --train_dir=$MODEL_DIR/video_level_logistic_model
   ```

   This will output the top 20 predicted labels from the model for every example to 'predictions.csv'.

1.1.3 Others

No need to touch other files

1.2 Set up Pycharm Development Environment