F. TensorFlow codes

F.1. Probabilistic model and loss function

```python
import tensorflow as tf

# inputs and annotators' labels
images = tf.placeholder(tf.float32, (None, image_shape))
labels = tf.placeholder(tf.int32,(None, num_annotators, num_classes))

# classifier for estimating true label distribution
logits = classifier(images)

# confusion matrices of annotators
confusion_matrices = confusion_matrix_estimators(num_annotators, num_classes)

# loss function
# 1. weighted cross-entropy
weighted_cross_entropy = cross_entropy_over_annotators(labels, logits, confusion_matrices)

# 2. trace of confusion matrices:
trace_norm = tf.reduce_mean(tf.trace(confusion_matrices))

# final loss (eq.(4))
total_loss = weighted_cross_entropy + scale * trace_norm
```

Listing 1: Implementation of the probabilistic model and the proposed loss function given in eq. (4). The final loss is minimized to learn jointly the confusion matrices of the respective annotators and the parameters of the classifier. The details of the used functions are given in Sec. F.2 & F.3

F.2. Defining confusion matrices of annotators

```python
import numpy as np
import tensorflow as tf

def confusion_matrix_estimators(num_annotators, num_classes):
    """Defines confusion matrix estimators.
    This function defines a set of confusion matrices that characterize respective annotators.
    Here (i, j)th element in the annotator confusion matrix of annotator a is given by
    P(label_annotator_a = j| label_true = i) i.e. the probability that the annotator assigns label j to
     the image when the ground truth label is i.

    Args:
        num_annotators: Number of annotators
        num_classes: Number of classes.

    Returns:
        confusion_matrices: Annotator confusion matrices. A `Tensor` of shape of shape [num_annotators,
     num_classes, num_classes]
    """
    # initialise so the confusion matrices are close to identity matrices.
    w_init = tf.constant(
        np.stack([6.0 * np.eye(num_classes) - 5.0 for j in range(num_annotators)]),
        dtype=tf.float32,
    )
    rho = tf.Variable(w_init, name='rho')

    # ensure positivity
    rho = tf.nn.softplus(rho)

    # ensure each row sums to one
    confusion_matrices = tf.divide(rho, tf.reduce_sum(rho, axis=-1, keepdims=True))
    return confusion_matrices
```

Listing 2: The confusion matrices are defined as `tf.Variable`. The positivity of the elements of confusion matrices is ensured by passing them through a soft-plus function.

F.3. Cross-entropy loss with sparse and noisy labels

```python
import numpy as np
import tensorflow as tf

def cross_entropy_over_annotators(labels, logits, confusion_matrices):
    """ Cross entropy between noisy labels from multiple annotators and their confusion matrix models.
    Args:
        labels: One-hot representation of labels from multiple annotators.
            tf.Tensor of size [batch, num_annotators, num_classes]. Missing labels are assumed to be
            represented as zero vectors.
        logits: Logits from the classifier. tf.Tensor of size [batch, num_classes]
        confusion_matrices: Confusion matrices of annotators. tf.Tensor of size
            [num annotators, num_classes, num_classes]. The (i, j) th element of the confusion matrix
            for annotator a denotes the probability P(label_annotator_a = j|label_true = i).
    Returns:
         The average cross-entropy across annotators and image examples.
    """
    # Treat one-hot labels as probability vectors
    labels = tf.cast(labels, dtype=tf.float32)

    # Sequentially compute the loss for each annotator
    losses_all_annotators = []
    for idx, labels_annotator in enumerate(tf.unstack(labels, axis=1)):
        loss = sparse_confusion_matrix_softmax_cross_entropy(
            labels=labels_annotator,
            logits=logits,
            confusion_matrix=confusion_matrices[idx, :, :],
        )
        losses_all_annotators.append(loss)

    # Stack them into a tensor of size (batch, num_annotators)
    losses_all_annotators = tf.stack(losses_all_annotators, axis=1)

    # Filter out annotator networks with no labels. This allows you train
    # annotator networks only when the labels are available.
    has_labels = tf.reduce_sum(labels, axis=2)
    losses_all_annotators = losses_all_annotators * has_labels
    return tf.reduce_mean(tf.reduce_sum(losses_all_annotators, axis=1))

def sparse_confusion_matrix_softmax_cross_entropy(labels, logits, confusion_matrix):
    """Cross entropy between noisy labels and confusion matrix based model for a single annotator.
    Args:
        labels: One-hot representation of labels. Tensor of size [batch, num_classes].
        logits: Logits from the classifier. Tensor of size [batch, num_classes]
        confusion_matrix: Confusion matrix of the annotator. Tensor of size [num_classes, num_classes].
    Returns:
         The average cross-entropy across annotators for image examples
         Returns a `Tensor` of size [batch_size].
    """
    # get the predicted label distribution
    preds_true = tf.nn.softmax(logits)

    # Map label distribution into annotator label distribution by
    # multiplying it by its confusion matrix.
    preds_annotator = tf.matmul(preds_true, confusion_matrix)

    # cross entropy
    preds_clipped = tf.clip_by_value(preds_annotator, 1e-10, 0.9999999)
    cross_entropy = tf.reduce_sum(-labels * tf.log(preds_clipped), axis=-1)
    return cross_entropy
```

Listing 3: Implementation of the cross entropy loss function. Here we use zero vectors as the "one-hot" representations of missing labels.