tf.metrics.sparse_precision_at_k

```
sparse_precision_at_k(
    labels,
    predictions,
    k,
    class_id=None,
    weights=None,
    metrics_collections=None,
    updates_collections=None,
    name=None
)
```

Defined in `tensorflow/python/ops/metrics_impl.py`.

Computes precision@k of the predictions with respect to sparse labels.

If `class_id` is specified, we calculate precision by considering only the entries in the batch for which `class_id` is in the top-k highest `predictions`, and computing the fraction of them for which `class_id` is indeed a correct label. If `class_id` is not specified, we'll calculate precision as how often on average a class among the top-k classes with the highest predicted values of a batch entry is correct and can be found in the label for that entry.

`sparse_precision_at_k` creates two local variables, `true_positive_at_<k>` and `false_positive_at_<k>`, that are used to compute the precision@k frequency. This frequency is ultimately returned as `precision_at_<k>`: an idempotent operation that simply divides `true_positive_at_<k>` by total (`true_positive_at_<k>` + `false_positive_at_<k>`).

For estimation of the metric over a stream of data, the function creates an `update_op` operation that updates these variables and returns the `precision_at_<k>`. Internally, a `top_k` operation computes a `Tensor` indicating the top `k` `predictions`. Set operations applied to `top_k` and `labels` calculate the true positives and false positives weighted by `weights`. Then `update_op` increments `true_positive_at_<k>` and `false_positive_at_<k>` using these values.

If `weights` is `None`, weights default to 1. Use weights of 0 to mask values.

Args:

- `labels`: `int64` `Tensor` or `SparseTensor` with shape [D1, ... DN, num_labels] or [D1, ... DN], where the latter implies num_labels=1. N >= 1 and num_labels is the number of target classes for the associated prediction. Commonly, N=1 and `labels` has shape [batch_size, num_labels]. [D1, ... DN] must match `predictions`. Values should be in range [0, num_classes), where num_classes is the last dimension of `predictions`. Values outside this range are ignored.
- `predictions`: Float `Tensor` with shape [D1, ... DN, num_classes] where N >= 1. Commonly, N=1 and predictions has shape [batch size, num_classes]. The final dimension contains the logit values for each class. [D1, ... DN] must match `labels`.
- `k`: Integer, k for @k metric.
- `class_id`: Integer class ID for which we want binary metrics. This should be in range [0, num_classes], where num_classes is the last dimension of `predictions`. If `class_id` is outside this range, the method returns NAN.
- `weights`: `Tensor` whose rank is either 0, or n-1, where n is the rank of `labels`. If the latter, it must be broadcastable to `labels` (i.e., all dimensions must be either `1`, or the same as the corresponding `labels` dimension).
- `metrics_collections`: An optional list of collections that values should be added to.
- `updates_collections`: An optional list of collections that updates should be added to.

- `name`: Name of new update operation, and namespace for other dependent ops.

Returns:

- `precision`: Scalar `float64` `Tensor` with the value of `true_positives` divided by the sum of `true_positives` and `false_positives`.
- `update_op`: `Operation` that increments `true_positives` and `false_positives` variables appropriately, and whose value matches `precision`.

Raises:

- `ValueError`: If `weights` is not `None` and its shape doesn't match `predictions`, or if either `metrics_collections` or `updates_collections` are not a list or tuple.
