metrics.py

```
import numpy as np
import tensorflow as tf
from sklearn.metrics import cohen_kappa_score, accuracy_score, jaccard_score
from tensorflow.keras import backend as K
from tensorflow_core.python.keras.metrics import BinaryAccuracy


def triplet_loss(y_true, y_pred, _alpha=1.0):
    positive_distance = K.mean(
        K.square(y_pred[:, 0] - y_pred[:, 1]), axis=-1
    )
    negative_distance = K.mean(
        K.square(y_pred[:, 0] - y_pred[:, 2]), axis=-1
    )
    return K.mean(K.maximum(0.0, positive_distance - negative_distance +
_alpha))


def weighted_categorical_crossentropy(weights=(1, 5, 10)):
    # Note: this is specific for 3 classes

    if isinstance(weights, list) or isinstance(weights, tuple):
        weights = np.array(weights)
    weights = K.variable(weights)

    def wcc_loss(y_true, y_pred):
        # scale predictions so that the class probs of each sample sum to 1
        ya = y_pred / (K.sum(y_pred, axis=-1, keepdims=True) + K.epsilon())
        # clip to prevent NaN's and Inf's
        yb = K.clip(ya, K.epsilon(), 1 - K.epsilon())
        # calc
        loss_a = y_true * K.log(yb) * weights
        loss_b = -K.sum(loss_a, -1)
        loss_c = K.mean(loss_b)

        return loss_c

    return wcc_loss


def jaccard_distance(y_true, y_pred, smooth=0.00001):
    intersection = K.sum(K.abs(y_true * y_pred),
axis=tuple(range(y_pred.shape.rank - 1)))
    sum_ = K.sum(K.abs(y_true) + K.abs(y_pred),
axis=tuple(range(y_pred.shape.rank - 1)))

    jac = (intersection + smooth) / (sum_ - intersection + smooth + K.epsilon())
    jd = K.mean(jac)

    return (1 - jd) * smooth

def weighted_dice_coefficient_per_class(y_true, y_pred, class_to_predict=0,
smooth=0.00001):
```

```
    axis = tuple(range(y_pred.shape.rank - 1))

    return (2. * (K.sum(y_true * y_pred,
                                axis=axis) + smooth / 2) / (K.sum(y_true,
                                                                  axis=axis) +
K.sum(y_pred,

  axis=axis) + smooth))[class_to_predict]


def weighted_dice_coefficient(y_true, y_pred, smooth=0.00001):
    axis = tuple(range(y_pred.shape.rank - 1))

    return K.mean(2. * (K.sum(y_true * y_pred,
                                axis=axis) + smooth / 2) / (K.sum(y_true,
                                                                  axis=axis) +
K.sum(y_pred,

  axis=axis) + smooth))


def weighted_dice_coefficient_loss(y_true, y_pred):
    return -weighted_dice_coefficient(y_true, y_pred)


def weighted_sum_loss(alpha=0.5, beta=0.5, weights=(1, 5, 10)):
    # Note: this is specific for 3 classes

    w_cross_entropy = weighted_categorical_crossentropy(weights)

    def loss(y_true, y_pred):
        gdl = alpha * jaccard_distance(y_true=y_true, y_pred=y_pred)
        wce = beta * w_cross_entropy(y_true=y_true, y_pred=y_pred)
        return gdl + wce

    return loss

def transform_multilabel_to_continuous(y, threshold):
    assert isinstance(y, np.ndarray), "invalid y"

    y = y > threshold
    y = y.astype(int).sum(axis=1) - 1
    return y


def score_kappa_kaggle(y, y_pred, threshold=0.5):
    y = transform_multilabel_to_continuous(y, threshold)
    y_pred = transform_multilabel_to_continuous(y_pred, threshold)
    return score_kappa(y, y_pred, labels=[0, 1, 2, 3, 4])


def score_kappa(y, y_pred, labels=None):
    if labels is not None:
        return cohen_kappa_score(y, y_pred, labels=labels, weights="quadratic")
    else:
        return cohen_kappa_score(y, y_pred, weights="quadratic")
```

```
def score_bin_acc(y, y_pred):
    m = BinaryAccuracy()
    m.update_state(y, y_pred)

    return m.result().numpy()


def score_cat_acc_kaggle(y, y_pred, threshold=0.5):
    y = transform_multilabel_to_continuous(y, threshold)
    y_pred = transform_multilabel_to_continuous(y_pred, threshold)
    return score_cat_acc(y, y_pred)


def score_cat_acc(y, y_pred):
    return accuracy_score(y, y_pred)


def score_jaccard(y, y_pred):
    y = np.argmax(y, axis=-1).flatten()
    y_pred = np.argmax(y_pred, axis=-1).flatten()

    return jaccard_score(y, y_pred, average="macro")


def score_dice(y, y_pred):
    y = np.argmax(y, axis=-1).flatten()
    y_pred = np.argmax(y_pred, axis=-1).flatten()

    j = jaccard_score(y, y_pred, average=None)

    return np.average(np.array([(2 * x) / (1 + x) for x in j]))

def score_dice_class(y, y_pred, class_to_predict):
    y = np.argmax(y, axis=-1).flatten()
    y_pred = np.argmax(y_pred, axis=-1).flatten()

    j = jaccard_score(y, y_pred, average=None)

    return np.array([(2 * x) / (1 + x) for x in j])[class_to_predict]


def _dice_hard_coe(target, output, smooth=1e-5):
    output = tf.cast(output, dtype=tf.float32)
    target = tf.cast(target, dtype=tf.float32)

    inse = tf.reduce_sum(tf.multiply(output, target))
    l = tf.reduce_sum(output)
    r = tf.reduce_sum(target)
    hard_dice = (2. * inse + smooth) / (l + r + smooth)
    return tf.reduce_mean(hard_dice)
```