```python
import tensorflow as tf
import numpy as np
import math
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, LeakyReLU
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
from sklearn.metrics import roc_auc_score
import multiprocessing
import even_samples

#num_cores = int(multiprocessing.cpu_count()/2)
#tf.config.threading.set_inter_op_parallelism_threads(num_cores)
#tf.config.threading.set_intra_op_parallelism_threads(num_cores)

# Model / data parameters
num_classes = 10
input_shape = (32, 32, 3)

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Scale images to the [0, 1] range
print("orig x_train shape:", x_train.shape)
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

def params2dict(x):
    minibatch, alpha, beta1, beta2, epochs = x
    return {
            'minibatch': minibatch,
            'alpha': alpha,
            'beta1': beta1,
            'beta2': beta2,
            'epochs': epochs,
    }

def compute_auc_loss(model, x_test, y_test):
    # Get predicted probabilities for each class
    preds = model.predict(x_test)

    # Compute AUC score for each class
    auc_scores = []
    for class_idx in range(num_classes):
        auc_score = roc_auc_score(y_test[:, class_idx], preds[:, class_idx])
        auc_scores.append(auc_score)

    return auc_scores

def compute_macro_auc(model, x_test, y_test):
    # Get predicted probabilities for each class
    preds = model.predict(x_test)

    # Compute AUC score for each class
    auc_scores = []
    for class_idx in range(num_classes):
        auc_score = roc_auc_score(y_test[:, class_idx], preds[:, class_idx])
        auc_scores.append(auc_score)

    # Compute macro-average AUC
    macro_auc = sum(auc_scores) / len(auc_scores)

    return macro_auc

def costf(x, train, test):
    x_train, y_train = train
    x_test, y_test = test
    print("params: ", params2dict(x))
    print("training data:", len(x_train))
    x_train_sub = x_train
    y_train_sub = y_train
    minibatch, alpha, beta1, beta2, epochs = x
    minibatch = math.floor(minibatch)
    epochs = math.floor(epochs)
    model = keras.Sequential()
    model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train_sub.shape[1:],activation='relu'))
    model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
    model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(num_classes, activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))

    adam_optimizer = keras.optimizers.Adam(learning_rate=alpha, beta_1=beta1, beta_2=beta2)
    model.compile(loss="categorical_crossentropy", optimizer=adam_optimizer, metrics=["accuracy"])
    model.summary()
    batch_size = minibatch
    print(x_train_sub.shape, y_train_sub.shape)
    history = model.fit(x_train_sub, y_train_sub, batch_size=batch_size, epochs=epochs, validation_split=0.1)
    test_loss, _ = model.evaluate(x_test, y_test, verbose=0)
    return test_loss
```

```
101:
102: if __name__ == "__main__":
103:     print(costf(np.array([5, 0.0001, 0.9, 0.999, 3]), n=50000))
```