Aprendizado auto-supervisionado

O aprendizado auto-supervisionado (self-supervised learning, SSL) é um paradigma de aprendizado de máquina utilizado quando a quantidade de dados rotulados é pequena. Este paradigma usa dados não-rotulados para resolver, em primeiro lugar, um problema artificial ("tarefa-pretexto" ou "pretext-task"). A tarefa-pretexto deve utilizar rótulos que podem ser obtidos automaticamente. Por exemplo, é possível rotacionar as imagens de treino aleatoriamente de 0°, 90°, 180° ou 270° e associar rótulos 0, 1, 2 ou 3 respectivamente a imagens rotacionadas por esses ângulos. A tarefa-pretexto, neste caso, é determinar o ângulo de rotação da imagem.

Resolvendo a tarefa-pretexto, o modelo aprende a extrair os atributos úteis que podem ser utilizados na tarefa verdadeira ("tarefa-jusante" ou "downstream-task"). Efetua-se o aprendizado de transferência para ajustar o modelo da tarefa-pretexto para a tarefa-jusante (usando os poucos dados rotulados disponíveis).

Assim, o pipeline SSL típico consiste em duas etapas. Primeiro, o modelo aprende a resolver uma tarefa-pretexto (tarefa de classificação auxiliar) usando os dados não-rotulados juntamente com pseudo-rótulos calculados automaticamente. Em segundo lugar, esse modelo é transferido para a tarefa real (tarefa-jusante) e é feito um ajuste fino do modelo com aprendizado supervisionado usando os poucos rótulos disponíveis. Por esse motivo, o SSL pode ser descrito como uma forma intermediária entre aprendizado não-supervisionado e supervisionado.

Vamos estudar SSL através de um exemplo bem simples, que classifica o CIFAR10 usando poucos dados de treino rotulados e muitos dados não-rotulados.

A tarefa-pretexto será reconhecer o ângulo de rotação (0°, 90°, 180° ou 270°) das imagens não-rotuladas.

I. CIFAR10 com rotação

O site abaixo traz a classificação de CIFAR10 usando SSL+SL e compara com abordagem puramente SL:

https://medium.com/analytics-vidhya/self-supervised-learning-for-image-classification-263e320fff07
https://github.com/larsh0103/SSL_Experiement
https://github.com/larsh0103/SSL_Experiement/blob/master/SSL_Experiment_CIFAR10.ipynb

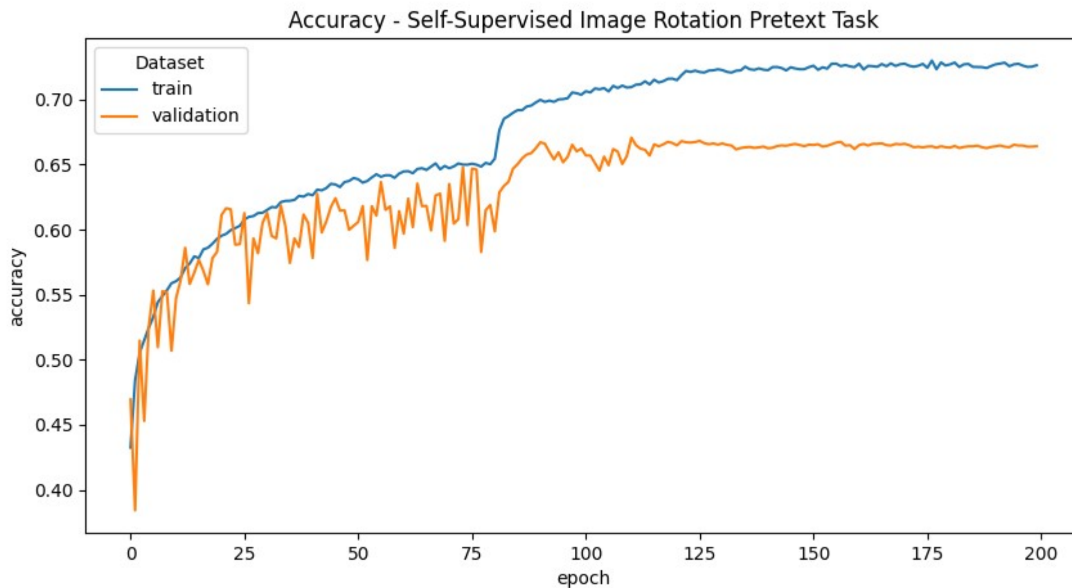
O código do site roda corretamente, fazendo apenas pequenas correções.

Executei o código parte em Colab e parte em computador local:

https://colab.research.google.com/drive/1df3lthUVqhLCLL8f4QmmUhVMaIWJCLOM?usp=share_link
~/deep/keras/ssl/cifar10

Como sabemos, o conjunto de dados CIFAR10 consiste em imagens coloridas 32×32 (50.000 imagens de treinamento e 10.000 de teste), divididas em 10 classes.

Na tarefa pretexto, cada uma das 50.000 imagens de treino é rotacionada aleatoriamente em 0°, 90º, 180° ou 270°. A tarefa pretexto é reconhecer a rotação. O treino demora um tempo considerável (algo como 12 horas). As curvas de perda e acuracidade da tarefa pretexto obtidas estão abaixo.



O programa ssl1.py abaixo é uma adaptação do programa original que só cria o modelo que resolve o problema-pretexto.

```
# -*- coding: utf-8 -*-
#ssl1.py: Roda somente SSL backbone
#https://medium.com/analytics-vidhya/self-supervised-learning-for-image-classification-263e320fff07
#https://github.com/larsh0103/SSL_Experiement

import tensorflow.keras as keras
from tensorflow.keras.layers import Dense, Conv2D, BatchNormalization, Activation
from tensorflow.keras.layers import AveragePooling2D, Input, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import cifar10
import numpy as np
import os; import sys
from sklearn.model_selection import StratifiedKFold
import seaborn as sns
import pandas as pd
import random; import pickle
import matplotlib.pyplot as plt
import tensorflow as tf

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    for gpu in gpus:
        try:
            tf.config.experimental.set_memory_growth(gpu, True)
            logical_gpus = tf.config.experimental.list_logical_devices('GPU')
            print(len(gpus), 'Physical GPUs,', len(logical_gpus), 'Logical GPUs')
        except RuntimeError as e:
            print(e)

def rotate_image(im,iterations=None):
  if not iterations:
    iterations = random.randint(0,3)
  y = [0,0,0,0]
  y[iterations]=1
  for i in range(iterations):
    im = np.rot90(im)
  return im,y

### Resnet Implementation taken from David Yang

def lr_schedule(epoch):
    lr = 1e-3
    if epoch > 180:   lr *= 0.5e-3
    elif epoch > 160: lr *= 1e-3
    elif epoch > 120: lr *= 1e-2
    elif epoch > 80:  lr *= 1e-1
    print('Learning rate: ', lr)
    return lr

def resnet_layer(inputs,
                 num_filters=16,
                 kernel_size=3,
                 strides=1,
                 activation='relu',
                 batch_normalization=True,
                 conv_first=True):
```

```
    conv = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))
    x = inputs
    if conv_first:
        x = conv(x)
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
    else:
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
        x = conv(x)
    return x

def resnet_v1(input_shape, depth, num_classes=10):
    if (depth - 2) % 6 != 0:
        raise ValueError('depth should be 6n+2 (eg 20, 32, 44 in [a])')
    # Start model definition.
    num_filters = 16
    num_res_blocks = int((depth - 2) / 6)
    inputs = Input(shape=input_shape)
    x = resnet_layer(inputs=inputs)
    # Instantiate the stack of residual units
    for stack in range(3):
        for res_block in range(num_res_blocks):
            strides = 1
            if stack > 0 and res_block == 0:  # first layer but not first stack
                strides = 2  # downsample
            y = resnet_layer(inputs=x,
                             num_filters=num_filters,
                             strides=strides)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters,
                             activation=None)
            if stack > 0 and res_block == 0:  # first layer but not first stack
                # linear projection residual shortcut connection to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=num_filters,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
            x = keras.layers.add([x, y])
            x = Activation('relu')(x)
        num_filters *= 2
    # Add classifier on top.
    # v1 does not use BN after last shortcut connection-ReLU
    x = AveragePooling2D(pool_size=8)(x)
    y = Flatten()(x)
    outputs = Dense(num_classes,
                    activation='softmax',
                    kernel_initializer='he_normal')(y)
    # Instantiate model.
    model = Model(inputs=inputs, outputs=outputs)
    return model

def make_rotated_data(Train,Test,subtract_pixel_mean=True):
    model_name='cifar100SLL'
    xy_rot_train = list(zip(*[rotate_image(im) for im in Train[0]]))
    xy_rot_test = list(zip(*[rotate_image(im) for im in Test[0]]))

    x_rot_train=np.array(xy_rot_train[0][:]).astype('float32')/255
    y_rot_train=np.array(xy_rot_train[1][:])
    x_rot_test=np.array(xy_rot_test[0][:]).astype('float32')/255
    y_rot_test=np.array(xy_rot_test[1][:])

    if subtract_pixel_mean:
        x_rot_train_mean = np.mean(x_rot_train, axis=0)
        x_rot_train -= x_rot_train_mean
        x_rot_test -= x_rot_train_mean

    return x_rot_train,y_rot_train,x_rot_test,y_rot_test

def make_ssl_backbone(Train, Test, save_dir, model_name, input_shape=(32,32,3), n=3):
    x_rot_train, y_rot_train, x_rot_test, y_rot_test = make_rotated_data(Train, Test)
    # Computed depth from supplied model parameter n
    depth = n * 6 + 2

    resnet_model= resnet_v1(input_shape=input_shape, depth=depth)
    x = Dense(4,activation='softmax')(resnet_model.layers[-2].output)
    model = keras.Model(resnet_model.inputs,x)

    filepath = os.path.join(save_dir, model_name)
    # Prepare callbacks for model saving and for learning rate adjustment.
    checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1, save_best_only=True)
    lr_scheduler = LearningRateScheduler(lr_schedule)
    lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1), cooldown=0, patience=10, min_lr=0.5e-6)
    callbacks = [checkpoint, lr_reducer, lr_scheduler]

    optimizer = keras.optimizers.Adam()
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    datagen = ImageDataGenerator(
                featurewise_center=False, # set input mean to 0 over the dataset
                samplewise_center=False, # set each sample mean to 0
                featurewise_std_normalization=False, # divide inputs by std of dataset
                samplewise_std_normalization=False, # divide each input by its std
                zca_whitening=False, # apply ZCA whitening
```

```
                 zca_epsilon=1e-06, # epsilon for ZCA whitening
                 rotation_range=0, # randomly rotate images in the range (deg 0 to 180)
                 width_shift_range=0.1, # *** randomly shift images horizontally
                 height_shift_range=0.1, # *** randomly shift images vertically
                 shear_range=0., # set range for random shear
                 zoom_range=0., # set range for random zoom
                 channel_shift_range=0., # set range for random channel shifts
                 fill_mode='nearest', # *** set mode for filling points outside the input boundaries
                 cval=0., # value used for fill_mode = "constant"
                 horizontal_flip=True, # *** randomly flip images
                 vertical_flip=False, # randomly flip images
                 rescale=None, # set rescaling factor (applied before any other transformation)
                 preprocessing_function=None, # set function that will be applied on each input
                 data_format=None, # image data format, either "channels_first" or "channels_last"
                 validation_split=0 # fraction of images reserved for validation (strictly between 0 and 1)
               )
             # Compute quantities required for featurewise normalization
             # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_rot_train)

    history_SSL = model.fit( datagen.flow(x_rot_train, y_rot_train),
                         validation_data=(x_rot_test,y_rot_test),
                         epochs=200, callbacks=callbacks)
    history_filepath = os.path.join(save_dir, model_name+"_history")
    with open(history_filepath,'wb') as fp:
        pickle.dump(history_SSL.history,fp)

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< main <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
save_dir='saved_models'
model_name='Restnetv1_SSL_Rotation.h5'
Train,Test = cifar10.load_data()
make_ssl_backbone(Train,Test,save_dir,model_name)

filepath = os.path.join(save_dir, model_name+"_history")
with open(filepath,'rb') as fp:
        history = pickle.load(fp)

ssl_df=pd.DataFrame({'loss':history['loss'] +history['val_loss'],'accuracy':history['accuracy']
+history['val_accuracy'],'Dataset':'train'})
ssl_df.loc[len(ssl_df)//2:,'Dataset']='validation'
ssl_df['epoch'] = np.concatenate((np.arange(len(history['loss'])),np.arange(len(history['loss']))))

figure = plt.figure(figsize=(10,5))
g = sns.lineplot(y='accuracy',x='epoch',hue='Dataset',data=ssl_df)
g.set_title('Accuracy - Self-Supervised Image Rotation Pretext Task')
plt.savefig("pretext_acc.png")

figure = plt.figure(figsize=(10,5))
g = sns.lineplot(y='loss',x='epoch',hue='Dataset',data=ssl_df)
g.set_title('Loss - Self-Supervised Image Rotation Pretext Task')
plt.savefig("pretext_loss.png")
```

Este programa vai gerar:
1) O modelo SSL em saved_models/Restnetv1_SSL_Rotation.h5
2) A história de progresso do treino em saved_models/Restnetv1_SSL_Rotation.h5_history
3) Os gráficos de acuracidade e loss como pretext_acc.png e pretext_loss.png.

O programa primeiro lê o BD Cifar10, obtendo variáveis Train e Test (que são tuples consistindo de imagem e rótulo):

```
Train,Test = cifar10.load_data()
# Train <class 'tuple'> 2
# Train[0] (50000, 32, 32, 3) uint8
# Train[1] (50000, 1) uint8
# Test <class 'tuple'> 2
# Test[0] (10000, 32, 32, 3) uint8
# Test[1] (10000, 1) uint8
```

Essas variáveis são convertidas em variáveis da tarefa-pretexto (rotated) com:

```
x_rot_train, y_rot_train, x_rot_test, y_rot_test = make_rotated_data(Train, Test)
# x_rot_train (50000, 32, 32, 3) float32
# y_rot_train (50000, 4) int64
# x_rot_test (10000, 32, 32, 3) float32
# y_rot_test (10000, 4) int64
```

As variáveis *y* são codificadas usando one-hot encoding.

O pré-processamento (converter para uint8 para float32 [0, 1] e subtrair média) é feito na função make_rotated_data.

Uma vez criado o modelo que resolve tarefa-pretexto, vamos resolver a tarefa-jusante, que é classificar imagens em uma das 10 categorias do Cifar-10. Tarefa-jusante é testada duas vezes: usando somente SL e usando SSL+SL.

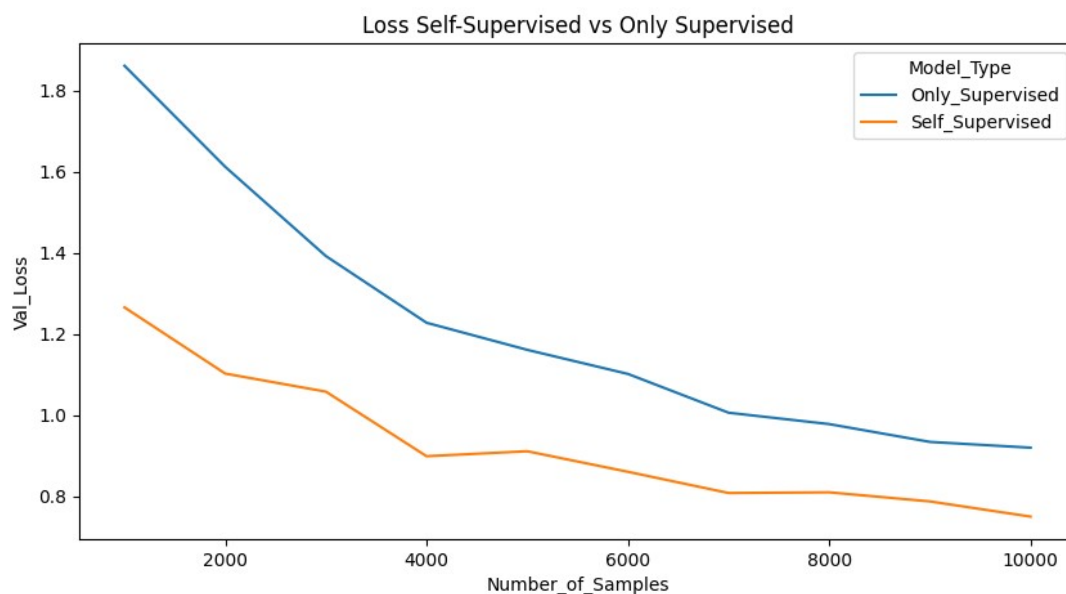
No caso somente SL, o modelo ResNet é inicializado com pesos aleatórios usando a inicialização padrão de Keras (Glorot). Este modelo é treinado usando os dados rotulados.

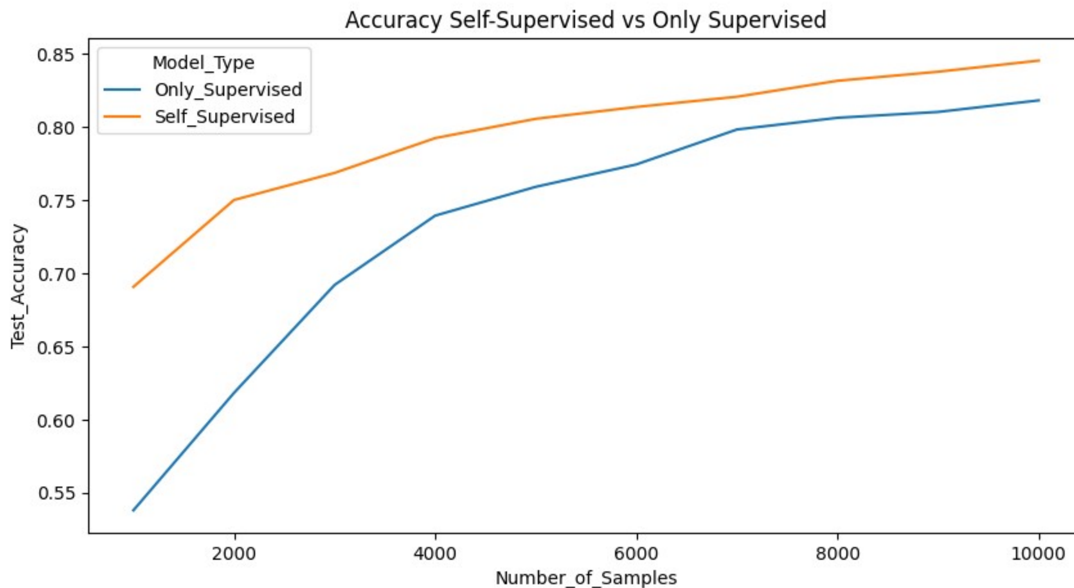
No caso de SSL+SL, os pesos iniciais são lidos do arquivo criado pelo SSL na etapa anterior:
saved_models/Restnetv1_SSL_Rotation.h5
Depois, as camadas de topo são descartados e substituídos por novas camadas de topo (uma camada densa com 10 saídas):

```
ssl_model=keras.models.load_model(ssl_path)
x = ssl_model.layers[-2].output
x = Dense(10,activation='softmax')(x)
model = keras.Model(ssl_model.inputs,x)
```

Depois, efetua-se um aprendizado de transferência usando os dados rotulados.

Rodando o programa original, obtemos os gráficos abaixo:





Como se pode ver, SSL+SL tem desempenho consideravelmente melhor do que somente SL, principalmente quando há poucos dados rotulados. *Number_of_samples* = 1000 significa que serão usadas 100 imagens de cada uma das 10 classes na aprendizagem supervisionada.

Porém, também se pode notar que a acuracidade obtida usando SSL + SL com 10.000 amostras rotuladas (≈85%) está bem abaixo das acuracidades obtidas fazendo SL padrão com 50.000 amostras rotuladas (≈92%, veja a apostila cifar-ead). Seria interessante testar qual é a maior taxa de acerto a que se chega usando 10.000 amostras rotuladas sem SSL.

O programa ssl3.py abaixo é uma versão simplificada do programa original e de execução mais rápida. Em vez de traçar o gráfico para diferentes *number_of_samples*, simplesmente imprime as acuracidades e losses de SL e SSL+SL para um dado *n_labels* (sinônimo de *number_of_samples*). Por exemplo, *n_labels* = 5.000 significa que 50.000 imagens de treino foram quebradas em *n_splits* = 10. Cada *split* tem 5.000 imagens e estas imagens foram usadas para treino.

É interessante observar a função de *sklearn*:

```
StratifiedKFold(n_splits=n_splits, random_state=random_state, shuffle=True)
```

Esta função gera amostras estratificadas (com número de amostras balanceado por classe) para teste k-fold cross validation. Esta função pode ser útil em muitas outras aplicações.

n_labels	100	250	625	1250	2500	5000
n_splits	500	200	80	40	20	10
Só SL loss	3.1759	3.0841	2.5175	2.0671	1.6122	1.2490
Só SL acc	29.39 %	38.72 %	45.95 %	54.01 %	66.60 %	76.15 %
SSL+SL loss	1.8979	1.9347	1.6211	1.3631	1.2433	1.0911
SSL+SL acc	46.61 %	55.98 %	65.22 %	70.39 %	75.06 %	79.89 %

```python
# -*- coding: utf-8 -*-
#ssl3.py: Usa SSL backbone, gerado por ssl1.py
#Testa SL e SSL+SL apenas para 1000 amostras rotuladas.
#https://medium.com/analytics-vidhya/self-supervised-learning-for-image-classification-263e320fff07
#https://github.com/larsh0103/SSL_Experiement

import tensorflow.keras as keras
from tensorflow.keras.layers import Dense, Conv2D, BatchNormalization, Activation
from tensorflow.keras.layers import AveragePooling2D, Input, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import cifar10
import numpy as np
import os; import sys
from sklearn.model_selection import StratifiedKFold
import seaborn as sns
import pandas as pd
import random; import pickle
import matplotlib.pyplot as plt
import tensorflow as tf

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    for gpu in gpus:
        try:
            tf.config.experimental.set_memory_growth(gpu, True)
            logical_gpus = tf.config.experimental.list_logical_devices('GPU')
            print(len(gpus), 'Physical GPUs,', len(logical_gpus), 'Logical GPUs')
        except RuntimeError as e:
            print(e)

def preprocess_data(x_train,y_train,x_test,y_test,subtract_pixel_mean=True):
    # Normalize data.
    x_train = x_train.astype('float32') / 255
    x_test = x_test.astype('float32') / 255
    # If subtract pixel mean is enabled
    if subtract_pixel_mean:
        x_train_mean = np.mean(x_train, axis=0)
        x_train -= x_train_mean
        x_test -= x_train_mean
    return x_train,y_train,x_test,y_test

def resnet_layer(inputs,
                 num_filters=16,
                 kernel_size=3,
                 strides=1,
                 activation='relu',
                 batch_normalization=True,
                 conv_first=True):
    conv = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))
    x = inputs
    if conv_first:
        x = conv(x)
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
    else:
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
        x = conv(x)
    return x

def resnet_v1(input_shape, depth, num_classes=10):
    if (depth - 2) % 6 != 0:
        raise ValueError('depth should be 6n+2 (eg 20, 32, 44 in [a])')
    # Start model definition.
    num_filters = 16
    num_res_blocks = int((depth - 2) / 6)
    inputs = Input(shape=input_shape)
    x = resnet_layer(inputs=inputs)
    # Instantiate the stack of residual units
    for stack in range(3):
        for res_block in range(num_res_blocks):
            strides = 1
            if stack > 0 and res_block == 0:  # first layer but not first stack
                strides = 2  # downsample
            y = resnet_layer(inputs=x,
                             num_filters=num_filters,
                             strides=strides)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters,
                             activation=None)
            if stack > 0 and res_block == 0:  # first layer but not first stack
                # linear projection residual shortcut connection to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=num_filters,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
```

```
            x = keras.layers.add([x, y])
            x = Activation('relu')(x)
        num_filters *= 2
    # Add classifier on top.
    # v1 does not use BN after last shortcut connection-ReLU
    x = AveragePooling2D(pool_size=8)(x)
    y = Flatten()(x)
    outputs = Dense(num_classes,
                    activation='softmax',
                    kernel_initializer='he_normal')(y)
    # Instantiate model.
    model = Model(inputs=inputs, outputs=outputs)
    return model

### Resnet Implementation taken from David Yang
def lr_schedule(epoch):
    lr = 1e-3
    if epoch > 180:   lr *= 0.5e-3
    elif epoch > 160: lr *= 1e-3
    elif epoch > 120: lr *= 1e-2
    elif epoch > 80:  lr *= 1e-1
    print('Learning rate: ', lr)
    return lr

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def train_models_datasplit(x_train, y_train, x_test, y_test,
                           batch_size = 128, epochs=200, data_augmentation=True,
                           num_classes=10,
                           n=3, ssl_path=None, dataset='cifar10', input_shape=(32,32,3),
                           n_labels=1000, random_state=689):

    # Convert class vectors to binary class matrices.
    y_test = keras.utils.to_categorical(y_test, num_classes)
    # Computed depth from supplied model parameter n
    depth = n * 6 + 2
    # Model name, depth and version
    # model_type = 'ResNet%d' % depth

    # Ensure that data increments consist of class balanced data
    n_splits = x_train.shape[0]//n_labels # 10000/1000=10
    print("n_labels=",n_labels,"n_splits=",n_splits)

    skf = StratifiedKFold(n_splits=n_splits, random_state=random_state, shuffle=True)
    index = next(skf.split(x_train,y_train))
    test_index=index[1] #5000
    print("Numero de amostras rotuladas:",test_index.shape[0])
    print(y_train[test_index])
    for i in range(10):
      print("Numero de %d:"%(i), np.count_nonzero(y_train[test_index]==i) )

    y_train_t = keras.utils.to_categorical(y_train, num_classes)[test_index]
    if ssl_path:
        ssl_model=keras.models.load_model(ssl_path)
        x = ssl_model.layers[-2].output
        x = Dense(10,activation='softmax')(x)
        model = keras.Model(ssl_model.inputs,x)
    else:
        model = resnet_v1(input_shape=input_shape, depth=depth)

    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(learning_rate=lr_schedule(0)),
                  metrics=['accuracy'])
    # model.summary()
    # print(model_type)
    # Prepare model model saving directory.
    save_dir = os.path.join(os.getcwd(), 'saved_models')
    #model_name = dataset + '_%s_model.h5' % model_type +"_"+ str((i/n_splits)*100)
    # if not os.path.isdir(save_dir):
        # os.makedirs(save_dir)
    # filepath = os.path.join(save_dir, model_name)

    # Prepare callbacks for model saving and for learning rate adjustment.
    #         checkpoint = ModelCheckpoint(filepath=filepath,
    #                                      monitor='val_loss',
    #                                      verbose=1,
    #                                      save_best_only=True)
    lr_scheduler = LearningRateScheduler(lr_schedule)
    lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1), cooldown=0, patience=5, min_lr=0.5e-6)
    callbacks = [ lr_reducer, lr_scheduler]

    # Run training, with or without data augmentation.
    if not data_augmentation:
        print('Not using data augmentation.')
        model.fit(x_train, y_train, batch_size=batch_size,  epochs=epochs,
                  validation_data=(x_test, y_test), shuffle=True, callbacks=callbacks)
    else:
        print('Using real-time data augmentation.')
        # This will do preprocessing and realtime data augmentation:
        datagen = ImageDataGenerator(
            featurewise_center=False, # set input mean to 0 over the dataset
            samplewise_center=False, # set each sample mean to 0
            featurewise_std_normalization=False, # divide inputs by std of dataset
            samplewise_std_normalization=False, # divide each input by its std
            zca_whitening=False, # apply ZCA whitening
            zca_epsilon=1e-06, # epsilon for ZCA whitening
            rotation_range=0, # randomly rotate images in the range (deg 0 to 180)
            width_shift_range=0.1, # *** randomly shift images horizontally
            height_shift_range=0.1, # *** # randomly shift images vertically
            shear_range=0., # set range for random shear
            zoom_range=0., # set range for random zoom
            channel_shift_range=0., # set range for random channel shifts
            fill_mode='nearest', # set mode for filling points outside the input boundaries
            cval=0., # value used for fill_mode = "constant"
            horizontal_flip=True, # *** randomly flip images
            vertical_flip=False, # randomly flip images
            rescale=None, # set rescaling factor (applied before any other transformation)
```

```
            preprocessing_function=None, # set function that will be applied on each input
            data_format=None, # image data format, either "channels_first" or "channels_last"
            validation_split=0 # fraction of images reserved for validation (strictly between 0 and 1)
            )
        # Compute quantities required for featurewise normalization
        # (std, mean, and principal components if ZCA whitening is applied).
        datagen.fit(x_train)
        history=model.fit(datagen.flow(x_train[test_index], y_train_t, batch_size=batch_size),
                          epochs=epochs, verbose=2, workers=4, validation_data=(x_test,y_test),
                          callbacks=callbacks
                         )

    score = model.evaluate(x_test, y_test, verbose=False)
    print('Test loss: %.4f'%(score[0]))
    print('Test accuracy: %.2f %%'%(100*score[1]))
    print('Test error: %.2f %%'%(100*(1-score[1])))

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< main <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
save_dir='saved_models'
model_name='Restnetv1_SSL_Rotation.h5'
#Train,Test = cifar10.load_data()
#make_ssl_backbone(Train,Test,save_dir,model_name)

n_labels=100
### Run experiment WITHOUT self-supervision
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, y_train, x_test, y_test = preprocess_data(x_train, y_train, x_test, y_test)
train_models_datasplit(x_train, y_train, x_test, y_test, batch_size=32,
                       epochs=200, n_labels=n_labels, dataset='CIFAR10_large')

### Run experiment with WITH self-supervision
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, y_train, x_test, y_test = preprocess_data(x_train, y_train, x_test, y_test)
train_models_datasplit(x_train, y_train, x_test,  y_test, batch_size=32,
                       epochs=200, n_labels=n_labels, dataset='CIFAR10_large',
                       ssl_path='saved_models/Restnetv1_SSL_Rotation.h5')
```

Outras tarefas-pretexto

Vamos tentar usar contrastive self-supervised learning.
https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html

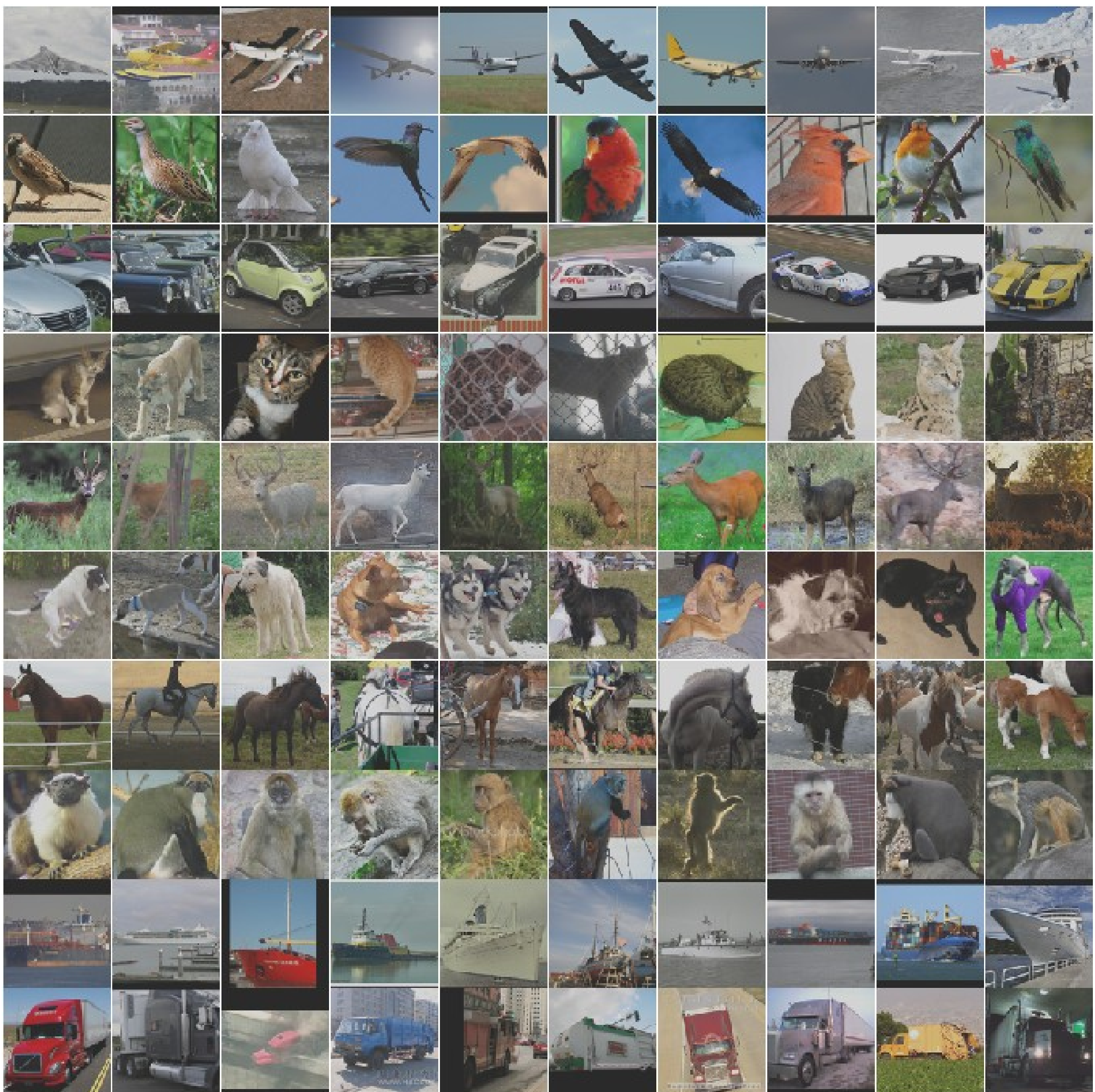
O seguinte blog faz SSL (contrastive, SimSiam) em cifar10 usando Keras:
https://keras.io/examples/vision/simsiam/
Neste blog, referencia um resnet20 projetado especifricamente para classificar cifar10.
Parece que chega a taxa de acerto de somente 23.39%.
Artigo de SimSiam (usa somente pares de imagens distorcidas, não usa exemplos negativos):
https://arxiv.org/abs/2011.10566

O seguinte blog traz supervised contrastive learning.
https://keras.io/examples/vision/supervised-contrastive-learning/
Chega a taxa de acerto de 81.62% para classificar Cifar10, o que seria uma taxa baixa se o treino for supervisionado.

Vamos usar BD STL-10, construido especificamente para testar self-supervised learning:
https://ai.stanford.edu/~acoates/stl10/

10 classes: 1=airplane, 2=bird, 3=car, 4=cat, 5=deer, 6=dog, 7=horse, 8=monkey, 9=ship, 10=truck.
Images are 96x96 pixels, color.
500 training images (10 pre-defined folds), 800 test images per class.
100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in addition to the ones in the labeled set.
Images were acquired from labeled examples on ImageNet.

Nota: Há 5000 imagens de treino e cada imagem aparece em 2 folds diferentes, de forma que em cada fold são usadas 1000 imagens de treino.
AX.shape= (5000, 96, 96, 3) AY.shape= (5000,)
QX.shape= (8000, 96, 96, 3) QY.shape= (8000,)
UX.shape= (100000, 96, 96, 3)
fold_indices.shape= (10, 1000)



Uma cópia desse BD está em ~/haebase/stl10/stl10_binary.tar.gz.
Esse BD descompactado está em ~/haebase/stl10/stl10_binary.
A rotina de download, descompactação e leitura desse BD, baseado nas rotinas disponibilizadas no site original, está em ~/haebase/stl10/stl10-master/le_stl10.py.

```python
#le_stl10.py
import os, sys, tarfile, errno
import numpy as np
import matplotlib.pyplot as plt

if sys.version_info >= (3, 0, 0):
  import urllib.request as urllib # ugly but works
else:
  import urllib

def download_and_extract(data_parent_dir,url):
  if not os.path.exists(data_parent_dir):
    os.makedirs(data_parent_dir)
  filename = url.split('/')[-1]
  filepath = os.path.join(data_parent_dir, filename)
  if not os.path.exists(filepath):
    def _progress(count, block_size, total_size):
      sys.stdout.write('\rDownloading %s %.2f%%' % (filename,
        float(count * block_size) / float(total_size) * 100.0))
      sys.stdout.flush()
    filepath, _ = urllib.urlretrieve(DATA_URL, filepath, reporthook=_progress)
    print('\nDownloaded', filename)
    tarfile.open(filepath, 'r:gz').extractall(data_parent_dir)

def read_labels(path_to_labels):
  with open(path_to_labels, 'rb') as f:
    labels = np.fromfile(f, dtype=np.uint8)
    labels = labels - 1
    return labels

def read_all_images(path_to_data):
  with open(path_to_data, 'rb') as f:
    everything = np.fromfile(f, dtype=np.uint8)
    images = np.reshape(everything, (-1, 3, 96, 96))
    images = np.transpose(images, (0, 3, 2, 1))
    return images

def read_folds(path_to_file):
  with open(path_to_file, 'rt') as f:
    lines=f.readlines()
  #lines e' list of strings
  fold_indices=np.empty((10,1000),np.int32)
  for l in range(fold_indices.shape[0]):
    linha=lines[l].split()
    for c in range(fold_indices.shape[1]):
      fold_indices[l,c]=int(linha[c])
  return fold_indices

def extract_fold(AX,AY,fold_indices,fold):
  AXF=np.empty( (fold_indices.shape[1],AX.shape[1],AX.shape[2],3), AX.dtype )
  AYF=np.empty( (fold_indices.shape[1],), AY.dtype )
  for i in range(fold_indices.shape[1]):
    AXF[i,:,:,:]=AX[fold_indices[fold,i],:,:,:]
    AYF[i]=AY[fold_indices[fold,i]]
  return AXF, AYF

def plot_image(image):
  plt.imshow(image)
  plt.show()

if __name__ == "__main__":
  DATA_PARENT_DIR = "/home/hae/haebase/stl10"
  DATA_OFFSPR_DIR = "/home/hae/haebase/stl10/stl10_binary"
  DATA_URL = "http://ai.stanford.edu/~acoates/stl10/stl10_binary.tar.gz"
  download_and_extract(DATA_PARENT_DIR, DATA_URL)

  AX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"train_X.bin"))
  AY = read_labels(os.path.join(DATA_OFFSPR_DIR,"train_y.bin"))
  print("AX.shape=",AX.shape,"AY.shape=",AY.shape) #AX.shape= (5000, 96, 96, 3) AY.shape= (5000,)
  print("AY[0]=",AY[0]); plot_image(AX[0]);
  for classe in range(10): print((AY==classe).sum(),end=" ")
  print()

  QX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"test_X.bin"))
  QY = read_labels(os.path.join(DATA_OFFSPR_DIR,"test_y.bin"))
  print("QX.shape=",QX.shape,"QY.shape=",QY.shape) #QX.shape= (8000, 96, 96, 3) QY.shape= (8000,)
  print("QY[0]=",QY[0]); plot_image(QX[0]);
  for classe in range(10): print((QY==classe).sum(),end=" ")
  print()
  #QLY=QY para testar downstream task
  #QUY para testar pretext task

  UX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"unlabeled_X.bin"))
  print("UX.shape=",UX.shape) #UX.shape= (100000, 96, 96, 3)
  for i in range(5): plot_image(UX[i])
  #UY para treinar pretext task

  fold_indices = read_folds(os.path.join(DATA_OFFSPR_DIR,"fold_indices.txt"))
  print("fold_indices.shape=",fold_indices.shape) #fold_indices.shape= (10, 1000)

  for fold in range(10):
    AXF, AYF = extract_fold(AX,AY,fold_indices,fold)
    print("AYF[0]=",AYF[0]); plot_image(AXF[0])
    for classe in range(10): print((AYF==classe).sum(),end=" ")
    print()
```

Esta rotina faz as seguintes transformações:

1) Muda os rótulos de 1 a 10 para 0 a 9:

10 classes: 0=airplane, 1=bird, 2=car, 3=cat, 4=deer, 5=dog, 6=horse, 7=monkey, 8=ship, 9=truck.

Vamos tentar resolver este problema usando aprendizagem supervisionada usando ResNet50v2 sem pesos, inspirado em (que usa Cifar10, em vez de STL10):

https://keras.io/examples/vision/supervised-contrastive-learning/

~/haelinux/haepi/deep/keras/ssl/stl10/resnet50/supervised1.py

```
#supervised1.py
import os; os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
import tensorflow as tf; tf.get_logger().setLevel('ERROR')
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import sys, tarfile, errno
import matplotlib.pyplot as plt

if sys.version_info >= (3, 0, 0): import urllib.request as urllib # ugly but works
else: import urllib

#<<<<<<<<<<<<<<<<<<<<<< Funcoes de leitura <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def download_and_extract(data_parent_dir,url):
  if not os.path.exists(data_parent_dir): os.makedirs(data_parent_dir)
  filename = url.split('/')[-1]
  filepath = os.path.join(data_parent_dir, filename)
  if not os.path.exists(filepath):
    def _progress(count, block_size, total_size):
      sys.stdout.write('\rDownloading %s %.2f%%' % (filename,
        float(count * block_size) / float(total_size) * 100.0))
      sys.stdout.flush()
    filepath, _ = urllib.urlretrieve(DATA_URL, filepath, reporthook=_progress)
    print('\nDownloaded', filename)
    tarfile.open(filepath, 'r:gz').extractall(data_parent_dir)

def read_labels(path_to_labels):
  with open(path_to_labels, 'rb') as f:
    labels = np.fromfile(f, dtype=np.uint8)
    labels = labels - 1
    return labels

def read_all_images(path_to_data):
  with open(path_to_data, 'rb') as f:
    everything = np.fromfile(f, dtype=np.uint8)
    images = np.reshape(everything, (-1, 3, 96, 96))
    images = np.transpose(images, (0, 3, 2, 1))
    return images

def read_folds(path_to_file):
  with open(path_to_file, 'rt') as f:
    lines=f.readlines()
  #lines e' list of strings
  fold_indices=np.empty((10,1000),np.int32)
  for l in range(fold_indices.shape[0]):
    linha=lines[l].split()
    for c in range(fold_indices.shape[1]):
      fold_indices[l,c]=int(linha[c])
  return fold_indices

def extract_fold(AX,AY,fold_indices,fold):
  AXF=np.empty( (fold_indices.shape[1],AX.shape[1],AX.shape[2],3), AX.dtype )
  AYF=np.empty( (fold_indices.shape[1],), AY.dtype )
  for i in range(fold_indices.shape[1]):
    AXF[i,:,:,:]=AX[fold_indices[fold,i],:,:,:]
    AYF[i]=AY[fold_indices[fold,i]]
  return AXF, AYF

def plot_image(image):
  plt.imshow(image)
  plt.show()

#<<<<<<<<<<<<<<<<<<< modelo de rede <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
data_augmentation = keras.Sequential(
#https://keras.io/examples/vision/supervised-contrastive-learning/
    [
        layers.Normalization(),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.02),
        layers.RandomWidth(0.2),
        layers.RandomHeight(0.2),
    ]
)

def create_encoder():
#https://keras.io/examples/vision/supervised-contrastive-learning/
  resnet = keras.applications.ResNet50V2(
    include_top=False, weights=None, input_shape=input_shape, pooling="avg"
  )

  inputs = keras.Input(shape=input_shape)
  augmented = data_augmentation(inputs)
  outputs = resnet(augmented)
  model = keras.Model(inputs=inputs, outputs=outputs, name="stl10-encoder")
  return model

def create_classifier(encoder, trainable=True):
  for layer in encoder.layers:
    layer.trainable = trainable
  inputs = keras.Input(shape=input_shape)
  features = encoder(inputs)
  features = layers.Dropout(dropout_rate)(features)
  features = layers.Dense(hidden_units, activation="relu")(features)
  features = layers.Dropout(dropout_rate)(features)
```

```
  outputs = layers.Dense(num_classes, activation="softmax")(features)
  model = keras.Model(inputs=inputs, outputs=outputs, name="cifar10-classifier")
  model.compile(
    optimizer=keras.optimizers.Adam(learning_rate),
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
  )
  return model

if __name__ == "__main__":
  #<<<<<<<<<<<<< Leitura <<<<<<<<<<<<<<<<<<<<<<<<<<
  DATA_PARENT_DIR = "/home/hae/haebase/stl10"
  DATA_OFFSPR_DIR = "/home/hae/haebase/stl10/stl10_binary"
  DATA_URL = "http://ai.stanford.edu/~acoates/stl10/stl10_binary.tar.gz"
  download_and_extract(DATA_PARENT_DIR, DATA_URL)
  AX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"train_X.bin"))
  AY = read_labels(os.path.join(DATA_OFFSPR_DIR,"train_y.bin"))
  QX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"test_X.bin"))
  QY = read_labels(os.path.join(DATA_OFFSPR_DIR,"test_y.bin"))
  #UX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"unlabeled_X.bin"))
  fold_indices = read_folds(os.path.join(DATA_OFFSPR_DIR,"fold_indices.txt"))

  input_shape=(AX.shape[1],AX.shape[2],AX.shape[3])
  num_classes=10
  learning_rate = 0.001
  batch_size = 100
  hidden_units = 512
  projection_units = 128
  num_epochs = 50
  dropout_rate = 0.5
  temperature = 0.05

  for fold in range(num_classes):
    print("Fold=",fold)
    ax, ay = extract_fold(AX,AY,fold_indices,fold)
    #Aqui, temos ax e ay para treinar (fold 0) e QX e QY para testar
    data_augmentation.layers[0].adapt(ax)

    encoder = create_encoder()
    classifier = create_classifier(encoder)
    #classifier.summary()
    history = classifier.fit(x=ax, y=ay, batch_size=batch_size, epochs=num_epochs, verbose=2)
    accuracy = classifier.evaluate(QX, QY)[1]
    print(f"Test accuracy: {round(accuracy * 100, 2)}%")
```

As acuracidades obtidas foram:
32.25%; 28.21%, 26.66%, 27.33%, 38.71%, 29.73% parei de executar aqui antes de atingir 10 folds.
Média = 30,48%

Vou rodar novamente, desta vez usando todas as imagens rotuladas para o treino.
~/haelinux/haepi/deep/keras/ssl/stl10/resnet50/supervised2.py

```
#supervised2.py
import os; os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
import tensorflow as tf; tf.get_logger().setLevel('ERROR')
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import sys, tarfile, errno
import matplotlib.pyplot as plt

if sys.version_info >= (3, 0, 0): import urllib.request as urllib # ugly but works
else: import urllib

#<<<<<<<<<<<<<<<<<<<<<<< Funcoes de leitura <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def download_and_extract(data_parent_dir,url):
  if not os.path.exists(data_parent_dir): os.makedirs(data_parent_dir)
  filename = url.split('/')[-1]
  filepath = os.path.join(data_parent_dir, filename)
  if not os.path.exists(filepath):
    def _progress(count, block_size, total_size):
      sys.stdout.write('\rDownloading %s %.2f%%' % (filename,
        float(count * block_size) / float(total_size) * 100.0))
      sys.stdout.flush()
    filepath, _ = urllib.urlretrieve(DATA_URL, filepath, reporthook=_progress)
    print('\nDownloaded', filename)
    tarfile.open(filepath, 'r:gz').extractall(data_parent_dir)

def read_labels(path_to_labels):
  with open(path_to_labels, 'rb') as f:
    labels = np.fromfile(f, dtype=np.uint8)
    labels = labels - 1
    return labels

def read_all_images(path_to_data):
  with open(path_to_data, 'rb') as f:
    everything = np.fromfile(f, dtype=np.uint8)
    images = np.reshape(everything, (-1, 3, 96, 96))
    images = np.transpose(images, (0, 3, 2, 1))
    return images

def read_folds(path_to_file):
  with open(path_to_file, 'rt') as f:
    lines=f.readlines()
  #lines e' list of strings
  fold_indices=np.empty((10,1000),np.int32)
  for l in range(fold_indices.shape[0]):
    linha=lines[l].split()
    for c in range(fold_indices.shape[1]):
      fold_indices[l,c]=int(linha[c])
  return fold_indices

# def extract_fold(AX,AY,fold_indices,fold):
  # AXF=np.empty( (fold_indices.shape[1],AX.shape[1],AX.shape[2],3), AX.dtype )
  # AYF=np.empty( (fold_indices.shape[1],), AY.dtype )
  # for i in range(fold_indices.shape[1]):
    # AXF[i,:,:,:]=AX[fold_indices[fold,i],:,:,:]
    # AYF[i]=AY[fold_indices[fold,i]]
  # return AXF, AYF

# def plot_image(image):
  # plt.imshow(image)
  # plt.show()

#<<<<<<<<<<<<<<<<<<<< modelo de rede <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
data_augmentation = keras.Sequential(
#https://keras.io/examples/vision/supervised-contrastive-learning/
    [
        layers.Normalization(),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.02),
        layers.RandomWidth(0.2),
        layers.RandomHeight(0.2),
    ]
)

def create_encoder():
#https://keras.io/examples/vision/supervised-contrastive-learning/
  resnet = keras.applications.ResNet50V2(
    include_top=False, weights=None, input_shape=input_shape, pooling="avg"
  )

  inputs = keras.Input(shape=input_shape)
  augmented = data_augmentation(inputs)
  outputs = resnet(augmented)
  model = keras.Model(inputs=inputs, outputs=outputs, name="stl10-encoder")
  return model

def create_classifier(encoder, trainable=True):
  for layer in encoder.layers:
    layer.trainable = trainable
  inputs = keras.Input(shape=input_shape)
  features = encoder(inputs)
  features = layers.Dropout(dropout_rate)(features)
  features = layers.Dense(hidden_units, activation="relu")(features)
  features = layers.Dropout(dropout_rate)(features)
  outputs = layers.Dense(num_classes, activation="softmax")(features)
  model = keras.Model(inputs=inputs, outputs=outputs, name="cifar10-classifier")
  model.compile(
    optimizer=keras.optimizers.Adam(learning_rate),
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
```

```
    )
    return model

if __name__ == "__main__":
  #<<<<<<<<<<<<< Leitura <<<<<<<<<<<<<<<<<<<<<<<<<<
  DATA_PARENT_DIR = "/home/hae/haebase/stl10"
  DATA_OFFSPR_DIR = "/home/hae/haebase/stl10/stl10_binary"
  DATA_URL = "http://ai.stanford.edu/~acoates/stl10/stl10_binary.tar.gz"
  download_and_extract(DATA_PARENT_DIR, DATA_URL)
  AX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"train_X.bin"))
  AY = read_labels(os.path.join(DATA_OFFSPR_DIR,"train_y.bin"))
  QX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"test_X.bin"))
  QY = read_labels(os.path.join(DATA_OFFSPR_DIR,"test_y.bin"))
  #UX = read_all_images(os.path.join(DATA_OFFSPR_DIR,"unlabeled_X.bin"))
  fold_indices = read_folds(os.path.join(DATA_OFFSPR_DIR,"fold_indices.txt"))

  input_shape=(AX.shape[1],AX.shape[2],AX.shape[3])
  num_classes=10
  learning_rate = 0.001
  batch_size = 100
  hidden_units = 512
  projection_units = 128
  num_epochs = 50
  dropout_rate = 0.5
  temperature = 0.05

  data_augmentation.layers[0].adapt(AX)
  encoder = create_encoder()
  classifier = create_classifier(encoder)
  #classifier.summary()
  history = classifier.fit(x=AX, y=AY, batch_size=batch_size, epochs=num_epochs, verbose=2)
  accuracy = classifier.evaluate(QX, QY)[1]
  print(f"Test accuracy: {round(accuracy * 100, 2)}%")
```

Foi obtida acuracidade de treino: sparse_categorical_accuracy=0.9154
E acuracidade de teste: Test accuracy=61.45%.

Agora, vamos usar a abordagem SimCLR de:
https://keras.io/examples/vision/semisupervised_simclr/
O problema da solução original é que o treino demora algo como 2 dias. Chega a acuracidade de 63%.
Supervised baseline é 61% (batch_size = 525 e num_epochs=20) que é muito parecida com 61.45% obtido no teste anterior com ResNet50v2). No meu teste, deu acuracidade 59.13%, usando batch_size de 100 e num_epochs=50. É menos do que a solução anterior (61,45%), pois aqui a estrutura da rede é mais simples do que ResNetV2.

Aqui, achar a rotação não seria um bom problema-pretexto, pois há imagens onde não é possível reconhecer a rotação.

SimCLR:
O seguinte blog mostra exemplo em Keras SimCLR usando stl10:
https://keras.io/examples/vision/semisupervised_simclr/
O problema é que o treino demora algo como 2 dias. Chega a acuracidade de 63%.
Supervised baseline é 61%.

NNCLR:
NNCLR está descrito em: https://arxiv.org/abs/2104.14548 e melhora 3,8% SimCLR.
O seguinte exemplo em Keras usa STL-10 e implementa NNCLR:
https://keras.io/examples/vision/nnclr/#:~:text=Introduction-,Self-supervised%20learning,with%20ample%20weak%20supervision%20labels.
O treino demora algo como uma hora e meia.
Chega a taxa de acerto de 59%. Diz que melhora se treinar durante mais tempo.

Conclusão:
Não é fácil melhorar rede neural usando SSL.