```
import numpy as np
import tensorflow as tf
import tarfile


!wget https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz


!tar --gunzip --extract --verbose --file=cifar-10-python.tar.gz


%cd cifar-10-batches-py/


# Code provided by https://www.cs.toronto.edu/~kriz/
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict


# Organizing data
train_b1 = unpickle('data_batch_1')
X1_train = train_b1[b'data']
y1_train = train_b1[b'labels']

train_b2 = unpickle('data_batch_2')
X2_train = train_b2[b'data']
y2_train = train_b2[b'labels']

train_b3 = unpickle('data_batch_3')
X3_train = train_b3[b'data']
y3_train = train_b3[b'labels']

train_b4 = unpickle('data_batch_4')
X4_train = train_b4[b'data']
y4_train = train_b4[b'labels']

train_b5 = unpickle('data_batch_5')
X5_train = train_b5[b'data']
y5_train = train_b5[b'labels']


test_b1 = unpickle('test_batch')
X_test = test_b1[b'data']
y_test = test_b1[b'labels']


# Splitting into train and validation set
x12 = np.vstack((X1_train, X2_train))
x123 = np.vstack((x12, X3_train))
```

```
X_train = np.vstack((X123,X4_train))
X_val = X5_train

y_train = y1_train + y2_train + y3_train + y4_train
y_val = y5_train


X_train = np.reshape(X_train, (40000,32,32,3))
X_val = np.reshape(X_val, (10000,32,32,3))
X_test = np.reshape(X_test, (10000,32,32,3))


X_train = X_train.astype('float32')
X_val = X_val.astype('float32')
X_test = X_test.astype('float32')


# Normalization values taken from https://github.com/kuangliu/pytorch-cifar/issues/19
mu = [0.49139968, 0.48215841, 0.44653091]
sigma = [0.24703223, 0.24348513, 0.26158784]


# Data normalization
def normalize(input, mu, sigma):
  input = input/255
  for i in range(0,2):
    input[:,:,i] = (input[:,:,i] - mu[i])/sigma[i]
  return input


X_train = normalize(X_train, mu, sigma)
X_val = normalize(X_val, mu, sigma)
X_test = normalize(X_test, mu, sigma)


import keras
from keras.models import Model, Sequential
from keras import optimizers, regularizers
from keras.layers import Dense, Dropout, Activation, Flatten, PReLU
from keras.layers import Conv2D, MaxPooling2D, Input, BatchNormalization
from keras.callbacks import LearningRateScheduler
from keras.preprocessing.image import ImageDataGenerator


# Make label one hot encoded
digits = 10
y_train = keras.utils.to_categorical(y_train, num_classes = digits)
y_val = keras.utils.to_categorical(y_val, num_classes = digits)
y_test = keras.utils.to_categorical(y_test, num_classes= digits)


def res_shortcut(input, residual):
  residual_channel = residual.shape[3:][0]
  shortcut = Conv2D(filters = residual_channel, kernel_size = [3,3], strides = (1,1), padding
  return shortcut
```

```
    return shortcut


def residual(input, filter):
  x = BatchNormalization()(input)
  x = Activation('relu')(x)
  x = Conv2D(filter, [3,3], strides = (1,1), padding = 'same')(x)
  x = BatchNormalization()(x)
  x = Activation('relu')(x)
  x = Conv2D(filter, [3,3], strides = (1,1), padding = 'same')(x)
  x_shortcut = res_shortcut(input, x)
  y = keras.layers.add([x, x_shortcut])
  return y




def lr_schedule(epoch):
    lrate = 0.001
    if epoch > 75:
        lrate = 0.0005
    if epoch > 100:
        lrate = 0.0003
    return lrate


wd = 1e-4
model = Sequential()
model.add(Conv2D(32, (3,3), padding='same', kernel_regularizer=regularizers.l2(wd), input_sha
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3,3), padding='same', kernel_regularizer=regularizers.l2(wd)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3,3), padding='same', kernel_regularizer=regularizers.l2(wd)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3,3), padding='same', kernel_regularizer=regularizers.l2(wd)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3,3), padding='same', kernel_regularizer=regularizers.l2(wd)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3,3), padding='same', kernel_regularizer=regularizers.l2(wd)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dense(num_classes, activation='softmax'))

    model.summary()

    #data augmentation
    datagen = ImageDataGenerator(
        rotation_range=15,
        width_shift_range=2,
        height_shift_range=2,
        horizontal_flip=True,
        )
    datagen.fit(x_train)

    #training
    batch_size = 64

    opt_rms = keras.optimizers.RMSprop(lr=0.001,decay=1e-6)
    model.compile(loss='categorical_crossentropy', optimizer=opt_rms, metrics=['accuracy'])
    model.fit_generator(datagen.flow(x_train, y_train, batch_size=batch_size),\
                        steps_per_epoch=x_train.shape[0] // batch_size,epochs=125,\
                        verbose=1,validation_data=(x_test,y_test),callbacks=[LearningRateSchedule
```

```
781/781 [==============================] - 43s 55ms/step - loss: 0.4550 - accuracy: 0.88
Epoch 96/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4576 - accuracy: 0.88
Epoch 97/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4576 - accuracy: 0.88
Epoch 98/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4560 - accuracy: 0.88
Epoch 99/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4555 - accuracy: 0.88
Epoch 100/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4545 - accuracy: 0.88
Epoch 101/125
781/781 [==============================] - 44s 56ms/step - loss: 0.4582 - accuracy: 0.88
Epoch 102/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4335 - accuracy: 0.89
Epoch 103/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4262 - accuracy: 0.89
Epoch 104/125
781/781 [==============================] - 43s 54ms/step - loss: 0.4199 - accuracy: 0.89
Epoch 105/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4216 - accuracy: 0.89
Epoch 106/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4188 - accuracy: 0.89
Epoch 107/125
781/781 [==============================] - 43s 54ms/step - loss: 0.4172 - accuracy: 0.89
Epoch 108/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4182 - accuracy: 0.89
Epoch 109/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4110 - accuracy: 0.89
Epoch 110/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4102 - accuracy: 0.89
Epoch 111/125
781/781 [==============================] - 44s 56ms/step - loss: 0.4100 - accuracy: 0.89
Epoch 112/125
781/781 [==============================] - 43s 56ms/step - loss: 0.4076 - accuracy: 0.89
Epoch 113/125
781/781 [==============================] - 44s 56ms/step - loss: 0.4011 - accuracy: 0.89
Epoch 114/125
781/781 [==============================] - 44s 56ms/step - loss: 0.4007 - accuracy: 0.89
Epoch 115/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4034 - accuracy: 0.89
Epoch 116/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4036 - accuracy: 0.89
Epoch 117/125
781/781 [==============================] - 42s 54ms/step - loss: 0.3966 - accuracy: 0.89
Epoch 118/125
781/781 [==============================] - 42s 54ms/step - loss: 0.3987 - accuracy: 0.89
Epoch 119/125
781/781 [==============================] - 42s 54ms/step - loss: 0.3988 - accuracy: 0.89
Epoch 120/125
781/781 [==============================] - 42s 54ms/step - loss: 0.3937 - accuracy: 0.89
Epoch 121/125
781/781 [==============================] - 43s 55ms/step - loss: 0.4009 - accuracy: 0.89
Epoch 122/125
781/781 [==============================] - 43s 54ms/step - loss: 0.3971 - accuracy: 0.89
Epoch 123/125
781/781 [==============================] - 43s 55ms/step - loss: 0.3907 - accuracy: 0.89
Epoch 124/125
```

```
Epoch 124/125
781/781 [==============================] - 42s 54ms/step - loss: 0.3922 - accuracy: 0.89
Epoch 125/125
781/781 [==============================] - 43s 54ms/step - loss: 0.3915 - accuracy: 0.89
<tensorflow.python.keras.callbacks.History at 0x7f41377c1da0>
```