ResNet_Keras(After_test)

August 5, 2022

0.1 Library

[ ]:
```
import keras
from keras.layers import Dense, Conv2D, BatchNormalization, Activation
from keras.layers import AveragePooling2D, Input, Flatten
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
from keras.regularizers import l2
from keras import backend as K
from keras.models import Model
from keras.datasets import cifar10
import numpy as np
import os
import math
```

0.2 Parameter

[ ]:
```
batch_size = 32  # orig paper trained all networks with batch_size=128
epochs = 5
data_augmentation = True
num_classes = 10

# Subtracting pixel mean improves accuracy
subtract_pixel_mean = True
n = 3
```

0.3 Model

[ ]:
```
# Model version
# Orig paper: version = 1 (ResNet v1), Improved ResNet: version = 2 (ResNet v2)
version = 1

# Computed depth from supplied model parameter n
if version == 1:
    depth = n * 6 + 2
```

```
elif version == 2:
    depth = n * 9 + 2

# Model name, depth and version
model_type = 'ResNet%dv%d' % (depth, version)
```

0.4 Load Data(CIFAR-10)

[ ]:
```
# Load the CIFAR10 data.
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Input image dimensions.
input_shape = x_train.shape[1:]

# 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

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print('y_train shape:', y_train.shape)

# 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)
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
y_train shape: (50000, 1)
```

[ ]:
```
def lr_schedule(epoch):
    lr = 15e-4
    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
```

0.5 Model Version1

- Version1 ; ResNet20 ~ ResNet110

[ ]:
```
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
```

0.6 Model Version2

- ResNet version

[ ]:
```
def resnet_v2(input_shape, depth, num_classes=10):
    if (depth - 2) % 9 != 0:
        raise ValueError('depth should be 9n+2 (eg 56 or 110 in [b])')
    # Start model definition.
    num_filters_in = 16
    num_res_blocks = int((depth - 2) / 9)
```

```
inputs = Input(shape=input_shape)
# v2 performs Conv2D with BN-ReLU on input before splitting into 2 paths
x = resnet_layer(inputs=inputs,
                 num_filters=num_filters_in,
                 conv_first=True)

# Instantiate the stack of residual units
for stage in range(3):
    for res_block in range(num_res_blocks):
        activation = 'relu'
        batch_normalization = True
        strides = 1
        if stage == 0:
            num_filters_out = num_filters_in * 4
            if res_block == 0:  # first layer and first stage
                activation = None
                batch_normalization = False
        else:
            num_filters_out = num_filters_in * 2
            if res_block == 0:  # first layer but not first stage
                strides = 2    # downsample

        # bottleneck residual unit
        y = resnet_layer(inputs=x,
                         num_filters=num_filters_in,
                         kernel_size=1,
                         strides=strides,
                         activation=activation,
                         batch_normalization=batch_normalization,
                         conv_first=False)
        y = resnet_layer(inputs=y,
                         num_filters=num_filters_in,
                         conv_first=False)
        y = resnet_layer(inputs=y,
                         num_filters=num_filters_out,
                         kernel_size=1,
                         conv_first=False)
        if res_block == 0:
            # linear projection residual shortcut connection to match
            # changed dims
            x = resnet_layer(inputs=x,
                             num_filters=num_filters_out,
                             kernel_size=1,
                             strides=strides,
                             activation=None,
                             batch_normalization=False)
```

```
            x = keras.layers.add([x, y])

        num_filters_in = num_filters_out

    # Add classifier on top.
    # v2 has BN-ReLU before Pooling
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    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
```

[ ]:
```
if version == 2:
    model = resnet_v2(input_shape=input_shape, depth=depth)
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 = 'cifar10_%s_model.{epoch:03d}.h5' % model_type
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_acc',
                             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 = [checkpoint, 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(
        # set input mean to 0 over the dataset
        featurewise_center=False,
        # set each sample mean to 0
        samplewise_center=False,
        # divide inputs by std of dataset
        featurewise_std_normalization=False,
        # divide each input by its std
        samplewise_std_normalization=False,
        # apply ZCA whitening
        zca_whitening=False,
        # epsilon for ZCA whitening
        zca_epsilon=1e-06,
        # randomly rotate images in the range (deg 0 to 180)
        rotation_range=0,
        # randomly shift images horizontally
        width_shift_range=0.1,
        # randomly shift images vertically
        height_shift_range=0.1,
        # set range for random shear
        shear_range=0.,
        # set range for random zoom
        zoom_range=0.,
        # set range for random channel shifts
        channel_shift_range=0.,
        # set mode for filling points outside the input boundaries
        fill_mode='nearest',
        # value used for fill_mode = "constant"
        cval=0.,
        # randomly flip images
        horizontal_flip=True,
```

```
        # randomly flip images
        vertical_flip=False,
        # set rescaling factor (applied before any other transformation)
        rescale=None,
        # set function that will be applied on each input
        preprocessing_function=None,
        # image data format, either "channels_first" or "channels_last"
        data_format=None,
        # fraction of images reserved for validation (strictly between 0 and 1)
        validation_split=0.0)

    # Compute quantities required for featurewise normalization
    # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_train)

    # Fit the model on the batches generated by datagen.flow().
    steps_per_epoch =  math.ceil(len(x_train) / batch_size)
    # fit the model on the batches generated by datagen.flow().
    model.fit(x=datagen.flow(x_train, y_train, batch_size=batch_size),
              verbose=1,
              epochs=epochs,
              validation_data=(x_test, y_test),
              steps_per_epoch=steps_per_epoch,
              callbacks=callbacks)
```

```
2022-08-05 10:57:18.768320: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.

Learning rate:  0.0015
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_1 (InputLayer)           [(None, 32, 32, 3)]  0           []

 conv2d (Conv2D)                (None, 32, 32, 16)   448
['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 32, 32, 16)  64
['conv2d[0][0]']
 alization)
```

```
 activation (Activation)        (None, 32, 32, 16)   0
['batch_normalization[0][0]']

 conv2d_1 (Conv2D)              (None, 32, 32, 16)   2320
['activation[0][0]']

 batch_normalization_1 (BatchNo  (None, 32, 32, 16)  64
['conv2d_1[0][0]']
 rmalization)

 activation_1 (Activation)      (None, 32, 32, 16)   0
['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 32, 32, 16)   2320
['activation_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 32, 32, 16)  64
['conv2d_2[0][0]']
 rmalization)

 add (Add)                      (None, 32, 32, 16)   0
['activation[0][0]',
'batch_normalization_2[0][0]']

 activation_2 (Activation)      (None, 32, 32, 16)   0           ['add[0][0]']

 conv2d_3 (Conv2D)              (None, 32, 32, 16)   2320
['activation_2[0][0]']

 batch_normalization_3 (BatchNo  (None, 32, 32, 16)  64
['conv2d_3[0][0]']
 rmalization)

 activation_3 (Activation)      (None, 32, 32, 16)   0
['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 32, 32, 16)   2320
['activation_3[0][0]']

 batch_normalization_4 (BatchNo  (None, 32, 32, 16)  64
['conv2d_4[0][0]']
 rmalization)

 add_1 (Add)                    (None, 32, 32, 16)   0
['activation_2[0][0]',
'batch_normalization_4[0][0]']

 activation_4 (Activation)      (None, 32, 32, 16)   0           ['add_1[0][0]']
```

```
 conv2d_5 (Conv2D)              (None, 32, 32, 16)   2320
['activation_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 32, 32, 16)  64
['conv2d_5[0][0]']
 rmalization)

 activation_5 (Activation)      (None, 32, 32, 16)   0
['batch_normalization_5[0][0]']

 conv2d_6 (Conv2D)              (None, 32, 32, 16)   2320
['activation_5[0][0]']

 batch_normalization_6 (BatchNo  (None, 32, 32, 16)  64
['conv2d_6[0][0]']
 rmalization)

 add_2 (Add)                    (None, 32, 32, 16)   0
['activation_4[0][0]',
'batch_normalization_6[0][0]']

 activation_6 (Activation)      (None, 32, 32, 16)   0           ['add_2[0][0]']

 conv2d_7 (Conv2D)              (None, 16, 16, 32)   4640
['activation_6[0][0]']

 batch_normalization_7 (BatchNo  (None, 16, 16, 32)  128
['conv2d_7[0][0]']
 rmalization)

 activation_7 (Activation)      (None, 16, 16, 32)   0
['batch_normalization_7[0][0]']

 conv2d_8 (Conv2D)              (None, 16, 16, 32)   9248
['activation_7[0][0]']

 conv2d_9 (Conv2D)              (None, 16, 16, 32)   544
['activation_6[0][0]']

 batch_normalization_8 (BatchNo  (None, 16, 16, 32)  128
['conv2d_8[0][0]']
 rmalization)

 add_3 (Add)                    (None, 16, 16, 32)   0
['conv2d_9[0][0]',
'batch_normalization_8[0][0]']
```

```
 activation_8 (Activation)      (None, 16, 16, 32)   0           ['add_3[0][0]']

 conv2d_10 (Conv2D)             (None, 16, 16, 32)   9248
['activation_8[0][0]']

 batch_normalization_9 (BatchNo  (None, 16, 16, 32)  128
['conv2d_10[0][0]']
 rmalization)

 activation_9 (Activation)      (None, 16, 16, 32)   0
['batch_normalization_9[0][0]']

 conv2d_11 (Conv2D)             (None, 16, 16, 32)   9248
['activation_9[0][0]']

 batch_normalization_10 (BatchN  (None, 16, 16, 32)  128
['conv2d_11[0][0]']
 ormalization)

 add_4 (Add)                    (None, 16, 16, 32)   0
['activation_8[0][0]',
'batch_normalization_10[0][0]']

 activation_10 (Activation)     (None, 16, 16, 32)   0           ['add_4[0][0]']

 conv2d_12 (Conv2D)             (None, 16, 16, 32)   9248
['activation_10[0][0]']

 batch_normalization_11 (BatchN  (None, 16, 16, 32)  128
['conv2d_12[0][0]']
 ormalization)

 activation_11 (Activation)     (None, 16, 16, 32)   0
['batch_normalization_11[0][0]']

 conv2d_13 (Conv2D)             (None, 16, 16, 32)   9248
['activation_11[0][0]']

 batch_normalization_12 (BatchN  (None, 16, 16, 32)  128
['conv2d_13[0][0]']
 ormalization)

 add_5 (Add)                    (None, 16, 16, 32)   0
['activation_10[0][0]',
'batch_normalization_12[0][0]']

 activation_12 (Activation)     (None, 16, 16, 32)   0           ['add_5[0][0]']
```

```
 conv2d_14 (Conv2D)             (None, 8, 8, 64)     18496
['activation_12[0][0]']

 batch_normalization_13 (BatchN  (None, 8, 8, 64)    256
['conv2d_14[0][0]']
 ormalization)

 activation_13 (Activation)     (None, 8, 8, 64)     0
['batch_normalization_13[0][0]']

 conv2d_15 (Conv2D)             (None, 8, 8, 64)     36928
['activation_13[0][0]']

 conv2d_16 (Conv2D)             (None, 8, 8, 64)     2112
['activation_12[0][0]']

 batch_normalization_14 (BatchN  (None, 8, 8, 64)    256
['conv2d_15[0][0]']
 ormalization)

 add_6 (Add)                    (None, 8, 8, 64)     0
['conv2d_16[0][0]',
'batch_normalization_14[0][0]']

 activation_14 (Activation)     (None, 8, 8, 64)     0           ['add_6[0][0]']

 conv2d_17 (Conv2D)             (None, 8, 8, 64)     36928
['activation_14[0][0]']

 batch_normalization_15 (BatchN  (None, 8, 8, 64)    256
['conv2d_17[0][0]']
 ormalization)

 activation_15 (Activation)     (None, 8, 8, 64)     0
['batch_normalization_15[0][0]']

 conv2d_18 (Conv2D)             (None, 8, 8, 64)     36928
['activation_15[0][0]']

 batch_normalization_16 (BatchN  (None, 8, 8, 64)    256
['conv2d_18[0][0]']
 ormalization)

 add_7 (Add)                    (None, 8, 8, 64)     0
['activation_14[0][0]',
'batch_normalization_16[0][0]']

 activation_16 (Activation)     (None, 8, 8, 64)     0           ['add_7[0][0]']
```

```
 conv2d_19 (Conv2D)             (None, 8, 8, 64)     36928
['activation_16[0][0]']

 batch_normalization_17 (BatchN  (None, 8, 8, 64)    256
['conv2d_19[0][0]']
 ormalization)

 activation_17 (Activation)     (None, 8, 8, 64)     0
['batch_normalization_17[0][0]']

 conv2d_20 (Conv2D)             (None, 8, 8, 64)     36928
['activation_17[0][0]']

 batch_normalization_18 (BatchN  (None, 8, 8, 64)    256
['conv2d_20[0][0]']
 ormalization)

 add_8 (Add)                    (None, 8, 8, 64)     0
['activation_16[0][0]',
'batch_normalization_18[0][0]']

 activation_18 (Activation)     (None, 8, 8, 64)     0           ['add_8[0][0]']

 average_pooling2d (AveragePool  (None, 1, 1, 64)    0
['activation_18[0][0]']
 ing2D)

 flatten (Flatten)              (None, 64)           0
['average_pooling2d[0][0]']

 dense (Dense)                  (None, 10)           650
['flatten[0][0]']

==================================================================================
==================
Total params: 274,442
Trainable params: 273,066
Non-trainable params: 1,376
----------------------------------------------------------------------------------
------------------
ResNet20v1
Using real-time data augmentation.
Learning rate:  0.0015
Epoch 1/5
1563/1563 [==============================] - ETA: 0s - loss: 1.5669 - accuracy:
0.4890WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
```

```
1563/1563 [==============================] - 1579s 1s/step - loss: 1.5669 -
accuracy: 0.4890 - val_loss: 1.5869 - val_accuracy: 0.5226 - lr: 0.0015
Learning rate:  0.0015
Epoch 2/5
1563/1563 [==============================] - ETA: 0s - loss: 1.1701 - accuracy:
0.6465WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
1563/1563 [==============================] - 1682s 1s/step - loss: 1.1701 -
accuracy: 0.6465 - val_loss: 1.2856 - val_accuracy: 0.6184 - lr: 0.0015
Learning rate:  0.0015
Epoch 3/5
1563/1563 [==============================] - ETA: 0s - loss: 1.0219 - accuracy:
0.7070WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
1563/1563 [==============================] - 1718s 1s/step - loss: 1.0219 -
accuracy: 0.7070 - val_loss: 1.3542 - val_accuracy: 0.6185 - lr: 0.0015
Learning rate:  0.0015
Epoch 4/5
1563/1563 [==============================] - ETA: 0s - loss: 0.9467 - accuracy:
0.7369WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
1563/1563 [==============================] - 1441s 922ms/step - loss: 0.9467 -
accuracy: 0.7369 - val_loss: 0.9992 - val_accuracy: 0.7214 - lr: 0.0015
Learning rate:  0.0015
Epoch 5/5
1563/1563 [==============================] - ETA: 0s - loss: 0.8867 - accuracy:
0.7604WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
1563/1563 [==============================] - 1332s 852ms/step - loss: 0.8867 -
accuracy: 0.7604 - val_loss: 1.0850 - val_accuracy: 0.7053 - lr: 0.0015
```

[ ]:
```python
# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
313/313 [==============================] - 45s 143ms/step - loss: 1.0850 -
accuracy: 0.7053
Test loss: 1.0849794149398804
Test accuracy: 0.705299973487854
```

[ ]:
```python
model.save("keras_resnet_test.h5")
```

[ ]:
```
```