

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



Hitchhiker's Guide to Residual Networks (ResNet) in Keras

Learn the foundations of residual networks and build a ResNet in Keras



Marco Peixeiro

Apr 8 · 4 min read ★



Photo by Andrés Canchón on Unsplash

Very deep neural networks are hard to train as they are more prone to vanishing or exploding gradients. To solve this problem, the activation unit from a layer could be fed directly to a deeper layer of the network, which is termed as a **skip connection**.

This forms the basis of **residual networks** or **ResNets**. This post will introduce the basics the residual networks before implementing one in Keras.



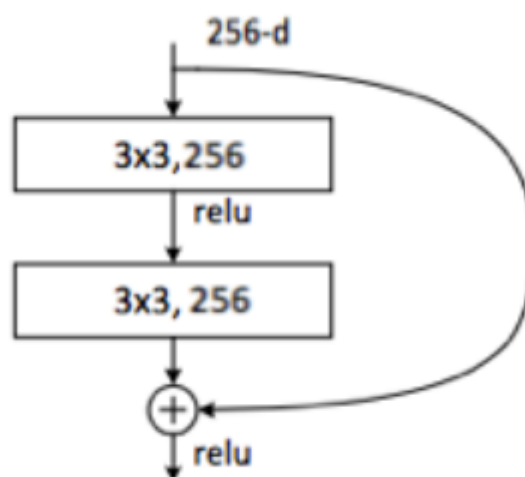
To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



With ResNets, we can build very deep neural networks

Residual block

A building block of a ResNet is called a **residual block** or **identity block**. A residual block is simply when the activation of a layer is fast-forwarded to a deeper layer in the neural network.



Example of a residual block

As you can see in the image above, the activation from a previous layer is being added to the activation of a deeper layer in the network.

This simple tweak allows training much deeper neural networks.

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



point where the training error will start increasing.

ResNets do not suffer from this problem. The training error will keep decreasing as more layers are added to the network. In fact, ResNets have made it possible to train networks with more than 100 layers, even reaching 1000 layers.

Building a ResNet for image classification

Now, let's build a ResNet with 50 layers for image classification using Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

In this case, we will use TensorFlow as the backend. Of course, feel free to grab the entire notebook and make all the necessary imports before starting.

Step 1: Define the identity block

First, we define the identity block, which will make our neural network a *residual network* as it represents the skip connection:

```
def identity_block(X, f, filters, stage, block):

    # Defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value
    X_shortcut = X

    # First component of main path
    X = Conv2D(filters = F1, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name =
conv_name_base + '2a', kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    # Second component of main path
    X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1, 1), padding = 'same', name =
conv_name_base + '2b', kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    # Third component of main path
    X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1, 1), padding = 'valid', name =
conv_name_base + '2c', kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

    # Final step: Add shortcut value to main path, and pass it through a RELU activation
    X = Add()(X, X_shortcut)
```

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



Step 2: Convolution block

Then, we build a convolution block like so:

```
def convolutional_block(X, f, filters, stage, block, s=2):

    # Defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value
    X_shortcut = X

    ##### MAIN PATH #####
    # First component of main path
    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(s, s), padding='valid', name=conv_name_base +
    '2a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    # Second component of main path
    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding='same', name=conv_name_base +
    '2b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    # Third component of main path
    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='valid', name=conv_name_base +
    '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    ##### SHORTCUT PATH #####
    X_shortcut = Conv2D(filters=F3, kernel_size=(1, 1), strides=(s, s), padding='valid',
    name=conv_name_base + '1', kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
    X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)

    # Final step: Add shortcut value to main path, and pass it through a RELU activation
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X
```

Notice how the convolution block combines both the *main* path and the *shortcut*.

Step 3: Build the model

Now, we combine both blocks into building a 50-layer residual network:

```
1 def ResNet50(input_shape = (64, 64, 3), classes = 6):
2
```


To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



```

6
7     # Zero-Padding
8     X = ZeroPadding2D((3, 3))(X_input)
9
10    # Stage 1
11    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', kernel_initializer = glorot_uniform)
12    X = BatchNormalization(axis = 3, name = 'bn_conv1')(X)
13    X = Activation('relu')(X)
14    X = MaxPooling2D((3, 3), strides=(2, 2))(X)
15
16    # Stage 2
17    X = convolutional_block(X, f = 3, filters = [64, 64, 256], stage = 2, block='a', s=2)
18    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
19    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')
20
21    # Stage 3
22    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
23    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
24    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
25    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')
26
27    # Stage 4
28    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
29    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
30    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
31    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
32    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
33    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')
34
35    # Stage 5
36    X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
37    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
38    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')
39
40    # AVGPPOOL
41    X = AveragePooling2D(pool_size=(2,2), padding='same')(X)
42
43    # Output layer
44    X = Flatten()(X)
45    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer = glorot_uniform)(X)
46
47
48    # Create model

```


To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



to assign it to a variable. Then, Keras requires us to compile the model:

```
model = ResNet50(input_shape = (64, 64, 3), classes = 6)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Once that is done, we can normalize our images and one-hot encode them:



```
X_train_orig, Y_train_orig, X_test_orig, Y_test_orig, classes = load_dataset()

# Normalize image vectors
X_train = X_train_orig/255.
X_test = X_test_orig/255.

# Convert training and test labels to one hot matrices
Y_train = convert_to_one_hot(Y_train_orig, 6).T
Y_test = convert_to_one_hot(Y_test_orig, 6).T
```

Afterwards, we can fit the model:

```
model.fit(X_train, Y_train, epochs = 2, batch_size = 32)
```

And see how it performed:

```
preds = model.evaluate(X_test, Y_test)
print ("Loss = " + str(preds[0]))
print ("Test Accuracy = " + str(preds[1]))
```

Now, you will see that we only get 16% accuracy. This is because we only trained on 2 epochs. You can train your model for longer on your own machine, but do realize that it will take a very long time, since it the network is very large.

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



code:

```
model.summary()
```

and you get a detailed summary of each layer in your network.

You can also generate a picture of the network's architecture and save it in your working directory:

```
plot_model(model, to_file='ResNet.png')  
SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

. . .

Great! You just learned the basics of a residual network and built one using Keras! Again, feel free to train the algorithm longer (~20 epochs), and you should see that the network performs very well. However, if you train only on CPU, this might take more than 1h.

In a future post, I will show how to perform neural style transfer in TensorFlow, which is a very fun way to apply convolution neural networks!

Keep learning!

Reference: deeplearning.ai

[Machine Learning](#)

[Deep Learning](#)

[Artificial Intelligence](#)

[Python](#)

[Programming](#)

[About](#) [Help](#) [Legal](#)