Image Classification with CNN on CIFAR10

MACHINE INTELLIGENCE LABORATORY

Teaching Assistants' email ID-

Darshil Shah: darshil.vs23@gmail.com

Mahim Dashora: contact.mahim@gmail.com

In this week's experiment, you will be making a convolutional neural network for image classification on the CIFAR10 dataset. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

A convolutional base consists of a stack of 2Dconvolutional, maxpooling layer to name a few.

The output tensor from the convolutional base is passed on to one or more dense layers to perform classification

You are provided with the following file:

1. **CNN.ipynb**

Important Points (Different from other labs):

1. Please do not make changes to the function definitions that are provided to you, as well as the functions that have **already been implemented**. Use the skeleton as it has been given.
2. You are free to write any helper functions that can be called in any of these predefined functions given to you. In this assignment you are required to **submit a "YOUR_SRN.h5" file** of the CNN model trained. (eg: PES1UG20CS999.h5)
3. Your model will be auto evaluated by our testing script.
4. Advisory, not mandatory: Using Google Colab is recommended for running the code

CNN.ipynb

- The **Conv Layer** parameters consist of a set of learnable filters (kernels or feature detectors). Filters are used for recognizing patterns throughout the entire input image. Convolution works by sliding the filter over the input image and along the way we take the dot product between the filter and chunks of the input image.

- **Max Pooling layer** reduces the size of feature maps by using some functions to summarize sub-regions, by taking the maximum value. Pooling works by sliding a window across the input and feeding the content of the window to a pooling function.

- **ReLU** stands for Rectified Linear Unit and is a non-linear operation. ReLU is an element-wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero.

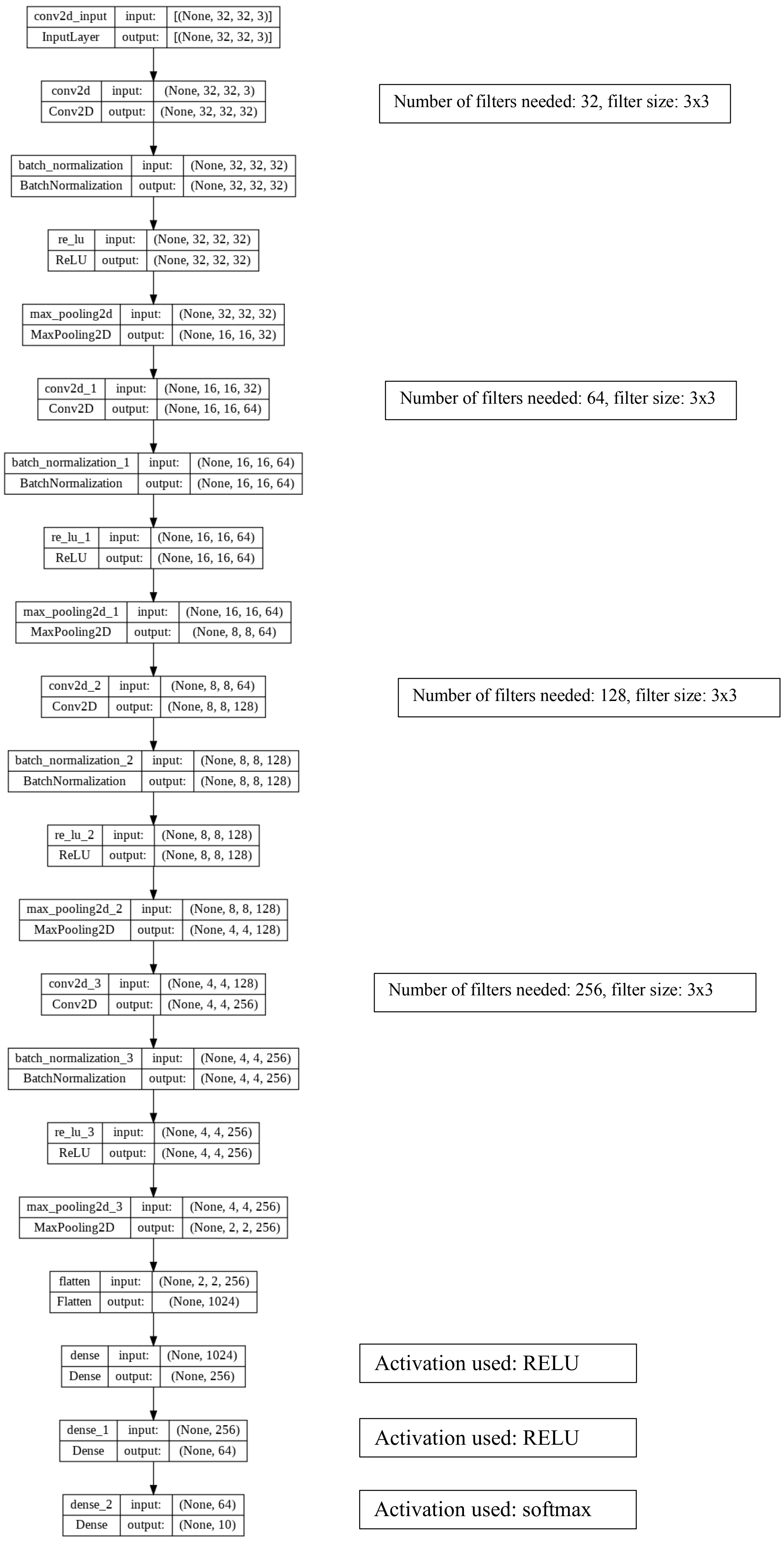
- **BatchNorm:** In **convolutions**, we have shared filters that go along the feature maps of the input (in images, the feature map is generally the height and width). These filters are the same on every feature map. It is then reasonable to normalize the output, in the same way, sharing it over the feature maps. In other words, this means that the parameters used to normalize are calculated along with each entire feature map. In a regular Batch Norm, each feature would have a different mean and standard deviation. Here, each feature map will have a single mean and standard deviation, used on all the features it contains. **Note: (Make sure to apply batchNorm only after Convolution operation)**

- **In each maxpooling operation, use the number of strides as 2 and the pool filter size as 2x2**

- **For each conv2D operation you need to figure out strides and padding to have the same input and output dimensions.**

- **There is no bias used in any convolution operation.**

- The input format in every layer is **(Batch Size, height, width, channel)** except the dense layers.


conv2d_input InputLayer input: [(None, 32, 32, 3)] output: [(None, 32, 32, 3)]
conv2d Conv2D input: (None, 32, 32, 3) output: (None, 32, 32, 32)
Number of filters needed: 32, filter size: 3x3
batch_normalization BatchNormalization input: (None, 32, 32, 32) output: (None, 32, 32, 32)
re_lu ReLU input: (None, 32, 32, 32) output: (None, 32, 32, 32)
max_pooling2d MaxPooling2D input: (None, 32, 32, 32) output: (None, 16, 16, 32)
conv2d_1 Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 64)
Number of filters needed: 64, filter size: 3x3
batch_normalization_1 BatchNormalization input: (None, 16, 16, 64) output: (None, 16, 16, 64)
re_lu_1 ReLU input: (None, 16, 16, 64) output: (None, 16, 16, 64)
max_pooling2d_1 MaxPooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
conv2d_2 Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 128)
Number of filters needed: 128, filter size: 3x3
batch_normalization_2 BatchNormalization input: (None, 8, 8, 128) output: (None, 8, 8, 128)
re_lu_2 ReLU input: (None, 8, 8, 128) output: (None, 8, 8, 128)
max_pooling2d_2 MaxPooling2D input: (None, 8, 8, 128) output: (None, 4, 4, 128)
conv2d_3 Conv2D input: (None, 4, 4, 128) output: (None, 4, 4, 256)
Number of filters needed: 256, filter size: 3x3
batch_normalization_3 BatchNormalization input: (None, 4, 4, 256) output: (None, 4, 4, 256)
re_lu_3 ReLU input: (None, 4, 4, 256) output: (None, 4, 4, 256)
max_pooling2d_3 MaxPooling2D input: (None, 4, 4, 256) output: (None, 2, 2, 256)
flatten Flatten input: (None, 2, 2, 256) output: (None, 1024)
dense Dense input: (None, 1024) output: (None, 256)
Activation used: RELU
dense_1 Dense input: (None, 256) output: (None, 64)
Activation used: RELU
dense_2 Dense input: (None, 64) output: (None, 10)
Activation used: softmax

In CNN.ipynb:

- The data has been split into test and train units.
- The image augmentation is already done and does not need to be handled by students
- You need to use the model.add() function (of tensorflow) in every operation **to complete this assignment** with variations in operations such as conv2D, MaxPooling2D etc
- Tensorflow version to be used is 2+
- Model will be trained on 20 epochs using adam optimizer and does not need to be handled by students
- **Students will need to save their model** and to refer if their model is correct they can use boilerplate code given in file.
- **Grading will be done based on number of parameters obtained at each stage, which can be viewed on doing "model.summary()" and marks will be awarded accordingly**