American University of Armenia

Graduate Program in Computer and Information Science

CS340 Machine Learning

Project Description

Academic year: 2017-2018

Project Title: Implementation of neural network described in “Deep Residual Learning for Image Recognition” paper

Supervisor: Arsen Mamikonyan

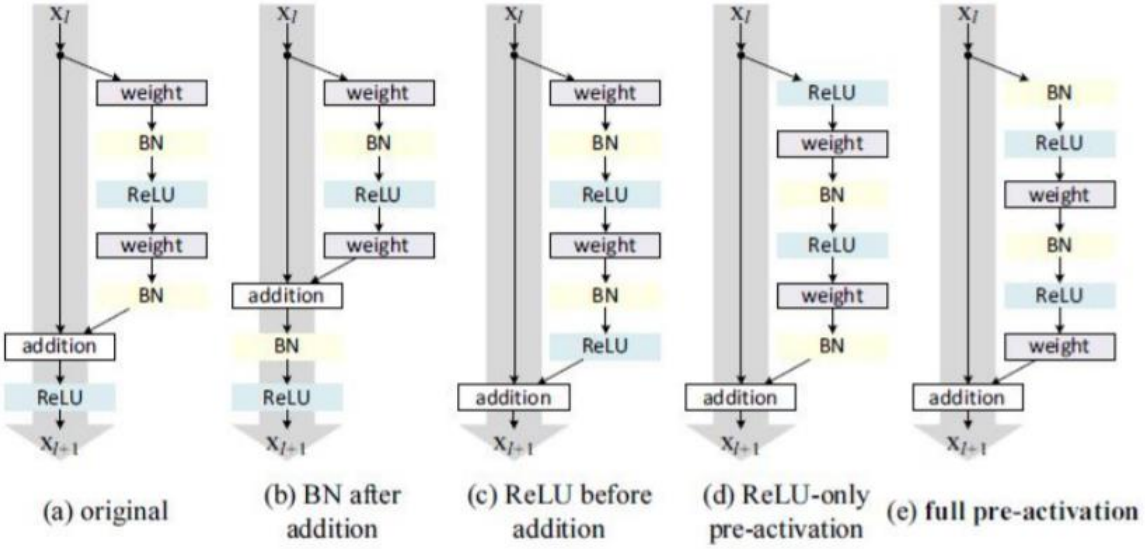
Student name: Seda Manukyan

Student N: A09175076

Student email: seda_manukyan@edu.aua.am

Introduction.

Deep convolutional neural networks have led to a series of breakthroughs for image classification. Studies showed that depth of the network is playing a major role in network's performance. But as the depth of the network increases we are facing the problem of vanishing gradient. The network that authors of the paper proposed is very simple but solves the problem. They decided to add so called "shortcut connections" to the plain network. Traditional plain networks calculates function y = f(x), where f(x) is convolution, matrix multiplication, batch normalization or something else and x is the input. In residual networks layer calculates y = f(x) + x. This connections enable the layers to learn incremental or residual representations. This is theoretical and actual implementation is a bit more complex. In reality to have better results the authors designed 5 configurations of residual blocks:



The one that is proved to be the best is configuration (e). The network is sequential attachment of this blocks.

Statement of Problem.

The purpose of this project is to implement a residual network based on the paper. To see how good it is I also implemented VGG16 network. The results can be seen in the Process section. For training and testing purposes I used CIFAR-10 dataset which has 10 different classes, 50000 training samples, 10000 testing samples, the images in dataset are 3×32×32 images (meaning the images are colored and have 32 pixel width and 32 pixel height).

Implementation.

I constructed two networks ResNet20 and VGG16 using deep learning library keras which is built on top of tensorflow library. There are deeper networks but because it is a very time consuming process to train deep neural networks I decided to train the shallower ones.

The models allow any data format of input images (there can be two ways of data formatting "channels_first" and "channels_last"). As CIFAR-10 dataset has "cannels_first" data formatting in models it is the one that is used.

There are 5 configurations of implementing residual blocks for ResNets and in my case I used 2 types of residual blocks: first block is Conv2D – Batch Normalization – ReLU and Batch Normalization – ReLU – Conv2D.

In res_net.py file you can find two main functions that are used to construct the model res_net_layer and res_net. In res_net_layer the configuration is built and in res_net the model is built.

In vgg16.py file the ordinary VGG16 model is built.

In the code you can find that you can use either data augmentation (in my case I decided not to use a lot of augmentation parameters) to train model or stay with the original datasets. keras library supports both variants.

Also as the network is starting to learn slower when iterations are increasing I used learning rate scheduler that will decrease learning rate in res_net_main.py file as the iterations grow.

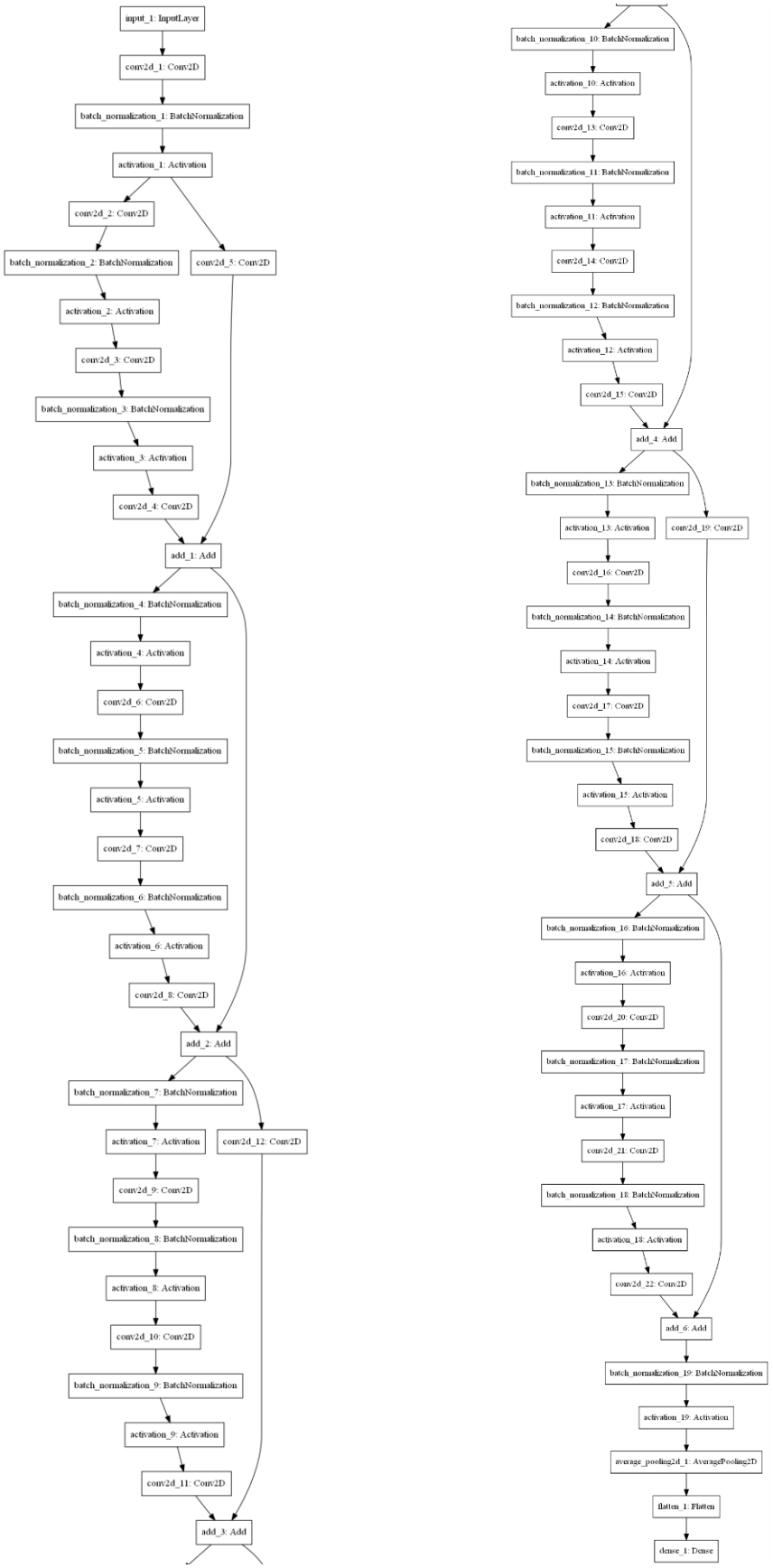
The architecture of models used in work are on the next page. As you can see both models are almost sequential but as was said before residual networks have identity additions in each block that's way there are long arrows. Both networks use same type of layers though they are connected differently.

For the ease of model reconstruction at every step that the model performs better it is saved in a .h5 file which later can be used to initialize model.

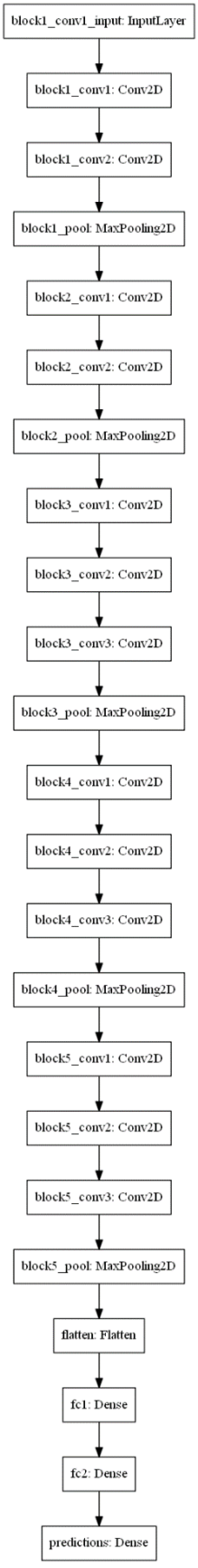
ResNet20 model has 574090 total number of parameters while VGG16 has 8423338 total number of parameters.

After training of model is finished they are evaluated with the testing data to have a fuller idea of the model performance.

ResNet


input_1: InputLayer
conv2d_1: Conv2D
batch_normalization_1: BatchNormalization
activation_1: Activation
conv2d_2: Conv2D
batch_normalization_2: BatchNormalization
conv2d_5: Conv2D
activation_2: Activation
conv2d_3: Conv2D
batch_normalization_3: BatchNormalization
activation_3: Activation
conv2d_4: Conv2D
add_1: Add
batch_normalization_4: BatchNormalization
activation_4: Activation
conv2d_6: Conv2D
batch_normalization_5: BatchNormalization
activation_5: Activation
conv2d_7: Conv2D
batch_normalization_6: BatchNormalization
activation_6: Activation
conv2d_8: Conv2D
add_2: Add
batch_normalization_7: BatchNormalization
activation_7: Activation
conv2d_12: Conv2D
conv2d_9: Conv2D
batch_normalization_8: BatchNormalization
activation_8: Activation
conv2d_10: Conv2D
batch_normalization_9: BatchNormalization
activation_9: Activation
conv2d_11: Conv2D
add_3: Add
batch_normalization_10: BatchNormalization
activation_10: Activation
conv2d_13: Conv2D
batch_normalization_11: BatchNormalization
activation_11: Activation
conv2d_14: Conv2D
batch_normalization_12: BatchNormalization
activation_12: Activation
conv2d_15: Conv2D
add_4: Add
batch_normalization_13: BatchNormalization
activation_13: Activation
conv2d_19: Conv2D
conv2d_16: Conv2D
batch_normalization_14: BatchNormalization
activation_14: Activation
conv2d_17: Conv2D
batch_normalization_15: BatchNormalization
activation_15: Activation
conv2d_18: Conv2D
add_5: Add
batch_normalization_16: BatchNormalization
activation_16: Activation
conv2d_20: Conv2D
batch_normalization_17: BatchNormalization
activation_17: Activation
conv2d_21: Conv2D
batch_normalization_18: BatchNormalization
activation_18: Activation
conv2d_22: Conv2D
add_6: Add
batch_normalization_19: BatchNormalization
activation_19: Activation
average_pooling2d_1: AveragePooling2D
flatten_1: Flatten
dense_1: Dense

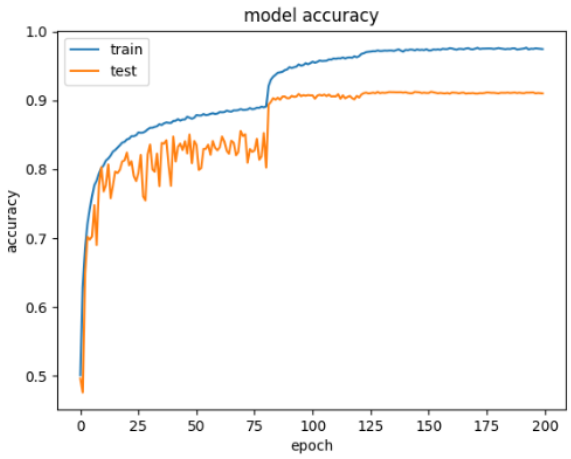
VGG

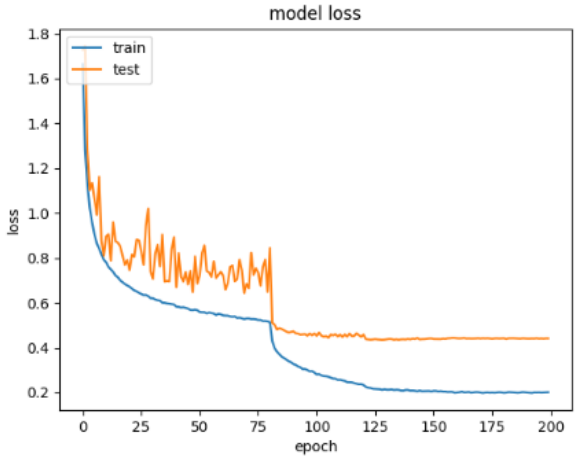

block1_conv1_input: InputLayer
block1_conv1: Conv2D
block1_conv2: Conv2D
block1_pool: MaxPooling2D
block2_conv1: Conv2D
block2_conv2: Conv2D
block2_pool: MaxPooling2D
block3_conv1: Conv2D
block3_conv2: Conv2D
block3_conv3: Conv2D
block3_pool: MaxPooling2D
block4_conv1: Conv2D
block4_conv2: Conv2D
block4_conv3: Conv2D
block4_pool: MaxPooling2D
block5_conv1: Conv2D
block5_conv2: Conv2D
block5_conv3: Conv2D
block5_pool: MaxPooling2D
flatten: Flatten
fc1: Dense
fc2: Dense
predictions: Dense

Experiments.

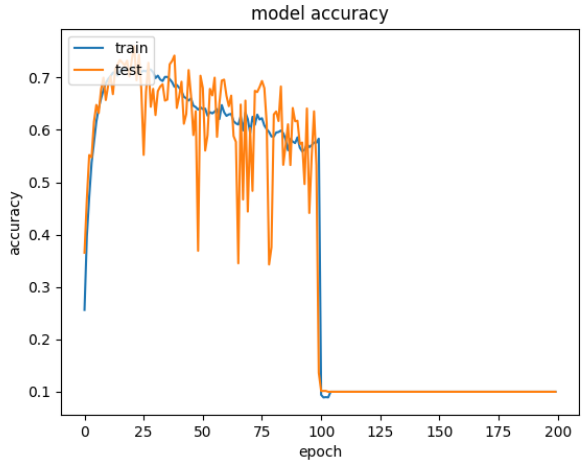
As was mentioned above the purpose of the project is to compare two models ResNet20 and VGG16. For this I created the same environment: everything used in both models was the same. Both models were trained on the same data (though I used ImageDataGenerator for image augmentation and this is how the models were trained). Both models were trained for 200 epochs, same 10 classes, same batch size.

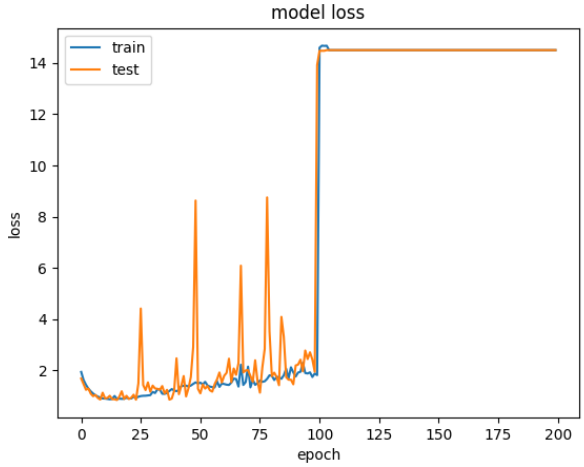
As was expected the ResNet performed very good achieving 97-98% accuracy on training data and 91% accuracy on testing data. Also you can see that at the first epochs the accuracy is growing very fast then it is stable for quite a long period which is followed by accuracy increase in 80th epoch and afterwards the accuracy increases but slowly. In the loss plot you can see the same picture only here the loss is decreasing.





The plots bellow are for VGG16. You can see that after 100 epochs accuracy and loss decrease drastically. The best result that VGG model performed was at 21st epoch where accuracy was only 75-76% on training set and 72% on testing set. First 21 epochs the model's accuracy is growing very fast but after it accuracy starts to decrease and at the 100th epoch it drops to 10%. The same is true for the loss only here it increases.





All models that had performed better than their previous ones are saved in saved_models and saved_models_vgg folders. In case you want to use them apply the model with the highest epoch number to gain the best result.

Conclusion.

This experiment proved the fact that ResNets are better classifiers than VGGs and though the number of iterations is very high the network does not tend to overfitting.

References.

1. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. Microsoft Research 2015
2. keras - https://keras.io/