Attention based Residual Network for CIFAR-10 Image Classification

Gargi Vaidya
Texas A&M University
College Station, TX, 77843

Abstract

This course project implements a novel attention mechanism based residual network configuration for the image classification of the standard CIFAR-10 dataset from Kaggle with end-to-end training.The network is built by stacking attention modules that generate attention-aware features. With deeper layers, the attention aware features learnt change adaptively. The attention residual learning mechanism is used to optimize very deep Residual Networks with hundreds of layers.

1 Introduction

Inspired by the attention mechanism techniques discussed in class and recent advances in Deep Learning beyond ResNets, this project implements mixed attention mechanism with a 'deep' structure as proposed by [2] in 'Residual Attention Networks for Image Classification'. It is observed that an attention based residual network increases the performance of the image classification model of a standard ResNet network.

2 Literature Review

Recent advances in image classification have seen various approaches to improve the discriminative ability of CNNs. Residual Learning [3] is proposed to learn residual of the identity mappings and as implemented in Assignment 2, greatly increased the depth of feed-forward network with significant improvement in classification performance. It is an elegant architecture to let information pass directly through and can be used to train thousands of layers. Batch Normalization and Dropout ensure regularization for model convergence and avoid any over-fitting and degradation. Soft attention in recent works can be trained end-to-end for convolutional networks. In the recent transformer modules, an affine transformation generated by a network module is applied to the input image to get attended region and is then fed to another deep network module.

This work includes a bottom-up top-down structure. The bottom-up feed-forward network produces low resolution feature maps while top-down network inferences dense features from each pixel. In this work, a single image is split into different sources of information and combined repeatedly. Hence, the residual learning reduces the loss of information due to repeated splitting and combining.

3 Network Architecture

The network is constructed by stacking multiple Attention Modules. Each attention module is split into a trunk and a mask branch. The trunk branch is responsible for feature processing and uses basic Residual Units to construct the Attention module. given the trunk branch output $T(x)$ with an input x, the mask branch uses a bottom-up top-down structure to learn same size mask $M(x)$ that softly

Preprint. Under review.

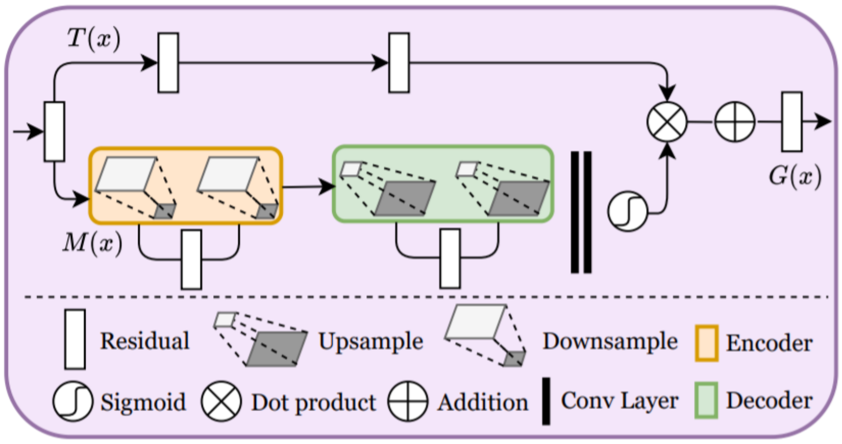


Figure 1: Attention Block

weight output features $T(x)$. The output of attention module H is :

$$H_{i,c}(x) = M_{i,c}(x) * T_{i,c}(x) \tag{1}$$

where i ranges over all spatial positions and c is the index of the channel. This structure is trained end-to-end. Refer Figure.1.

In the Attention Modules, the attention mask also acts as a gradient update filter during back propagation besides acting as a feature selector. In the soft mask branch the gradient of mask for input feature is

$$\frac{\partial M(x,\theta)T(x,\phi)}{\partial \phi} = M(x,\theta)\frac{\partial T(x,\phi)}{\partial \phi} \tag{2}$$

where θ are the mask branch parameters and ϕ are trunk branch parameters. Thus attention modules are robust to noisy labels and prevent wrong gradients to update the trunk parameters because of this feature.

Why stacked Attention Modules? Images with large variations need to be modelled by different types of attentions. Features from different layers need to be modelled by different attention masks. Using just a single mask branch would need many channels to capture all the combinations of different feature factors. Also, a single Attention Module would only modify the features once. If this modification fails there is no second chance to retrieve the lost information. In this work, the each trunk branch has its own mask branch to learn attention specialized for its features.

3.1 Attention Residual Learning

Just naive stacking of attention modules can lead to a performance drop as it involves a dot product with mask range from 0 to 1, it will degrade the value of the features repeatedly. Thus, the said paper [2] proposes a soft masknunit constructed as an identical mapping, whose performance shall be no worse than its counterpart without attention Thus, the output H of attention module modifies as per follow -

$$H_{i,c}(x) = (1 + M_{i,c}(x))) * F_{i,c}(x) \tag{3}$$

where $M_{i,c}(x)$ ranges from [0,1] with $M(x)$ trying to approximate 0 and $H(x)$ trying to approximate the original features $F(x)$ during the attention residual learning.

The original residual blocks (ResNets from Homework2), the residual learning is formulated as $H_{i,c}(x) = x + F_{i,c}(x)$, where $F_{i,c}(x)$ tries to approximate the residual function. In attention residual learning, $F_{i,c}$ indicates features approximation. The mask branches work as feature selectors that enhance the good features and suppress noises from trunk features.

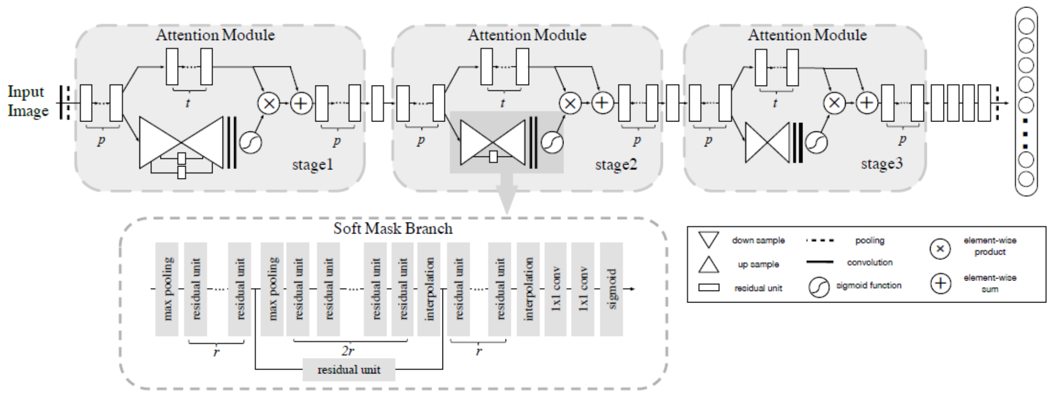


Figure 2: Residual Attention Network Architecture - Uses 3 hyper-parameters $p, tandr$ for the Attention Module. p denotes the number of pre-processing ResNet units before splitting into trunk and mask branch. t denotes the ResNet units in the trunk branch. r denotes the number of ResNet units between adjacent pooling layer in the mask branch. Best hyper-parameters in this project - $p = 1, t = 2, r = 1$

3.2 Soft Mask Branch

The mask branch contains a bottom-up feed-forward part and top-down feedback steps. The bottom-up section collects global information of the whole image and the top-down section combines the global information with original feature maps.

First, the max pooling increases the receptive field after few ResNet units. After reaching lowest resolution, global information is expanded by a symmetrical top-down architecture. Linear interpolation up-samples the output from Residual Units. The output size is kept the same as input feature map by using same number of interpolation and max-pooling layers. Lastly the sigmoid layer normalizes the output range to [0,1].

3.3 Mixed Attention

In this work, attention by the mask branch changes adaptably with the trunk branch features. Constraints to attention can be added to mask branch by changing the normalization step in activation function before soft mask output. The activation function $f(x_{i,c})$ used in this work is mixed-attention that uses simple sigmoid for each channel and spatial position.

$$f(x_{i,c}) = \frac{1}{1 + exp(-x_{i,c})} \tag{4}$$

where i ranges over the spatial positions and c ranges over all channels.

4 Experiments Results

The Residual Attention Network is evaluated on the CIFAR-10 dataset and compared with the Residual Network from Homework 2. The dataset consisted of 32x32 color images of 10 classes with 50,000 training and 10,000 public testing images. Each image is padded by 4 pixels on each side resulting in a 40x40 image. Also, a 32x32 crop is randomly sampled from the original image or its horizontal flip. The network consisted of 3 stages, with equal number of attention - modules stacked in each stage.

The baseline Residual Network architecture achieves a 90.55% accuracy as seen in Table. 1 on the public testing dataset with more number of parameters than the network implemented in this project. The Residual Attention Network achieves a % accuracy on the same public testing dataset as in Table 1. Thus, it can be inferred that the attention residual learning mechanism can efficiently reduce the number of trainable parameters in the network while improving the classification performance.

Table 1: Publi Dataset Test Accuracy Comparison

Name	Parameters (10^6)	Accuracy (%)
Residual Network	2.1	90.55
Residual Attention Network	1.9	94.96

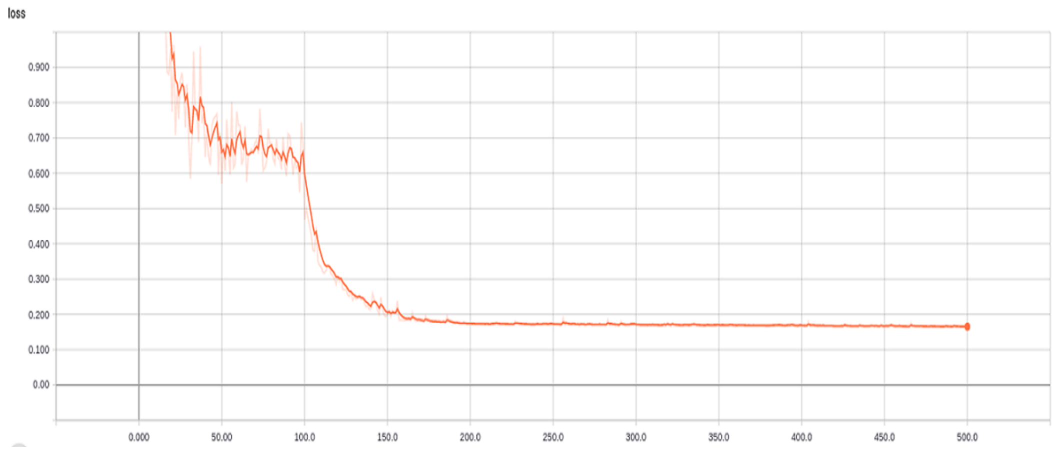


Figure 3: Training Loss vs Epochs on Baseline Residual Network

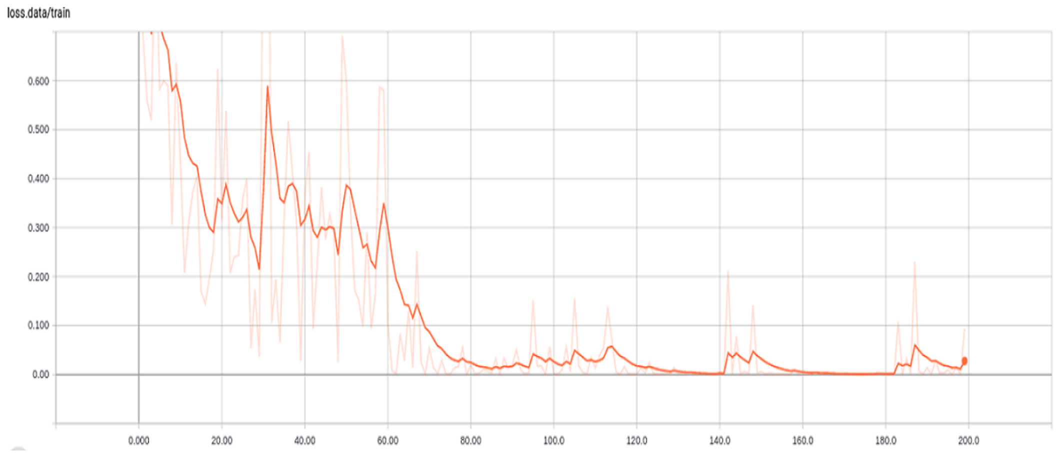


Figure 4: Training Loss vs Epochs on Residual Attention Network

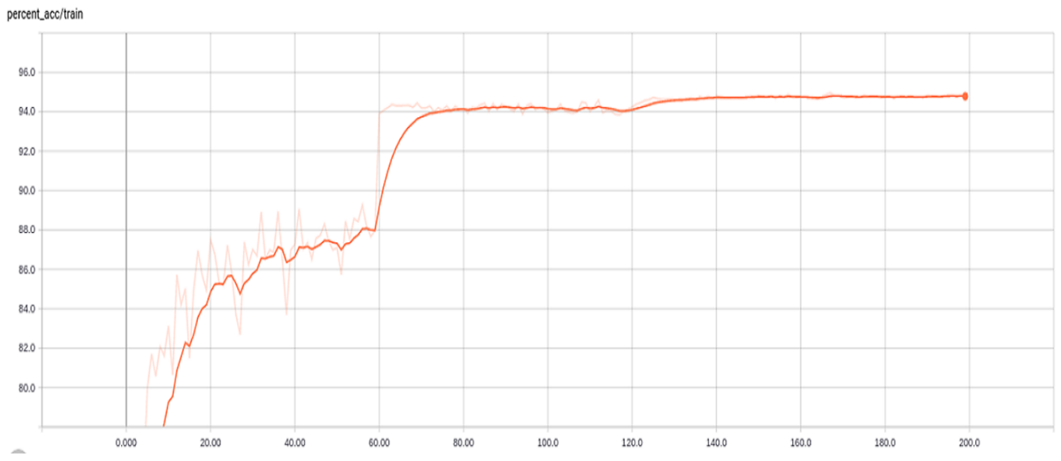


Figure 5: Training Validation Accuracy vs Epochs on Residual Attention Network

5 Conclusion

Successfully implemented an Attention based Residual Network for the CIFAR-10 dataset and compared with a state-of-the-art Residual Network and achieves an improved accuracy over the public testing dataset. The benefits of the Residual Attention Network are that the network can be modified for different kinds of attention by modifying the activation function in 3.3. The implemented mixed attention has much better overall performance compared to just spatial or channel attention. Also, the basic attention modules can be combined to form larger or deeper networks. Due to the bottom-up top-down structure this network can be extended without much degradation in performance.

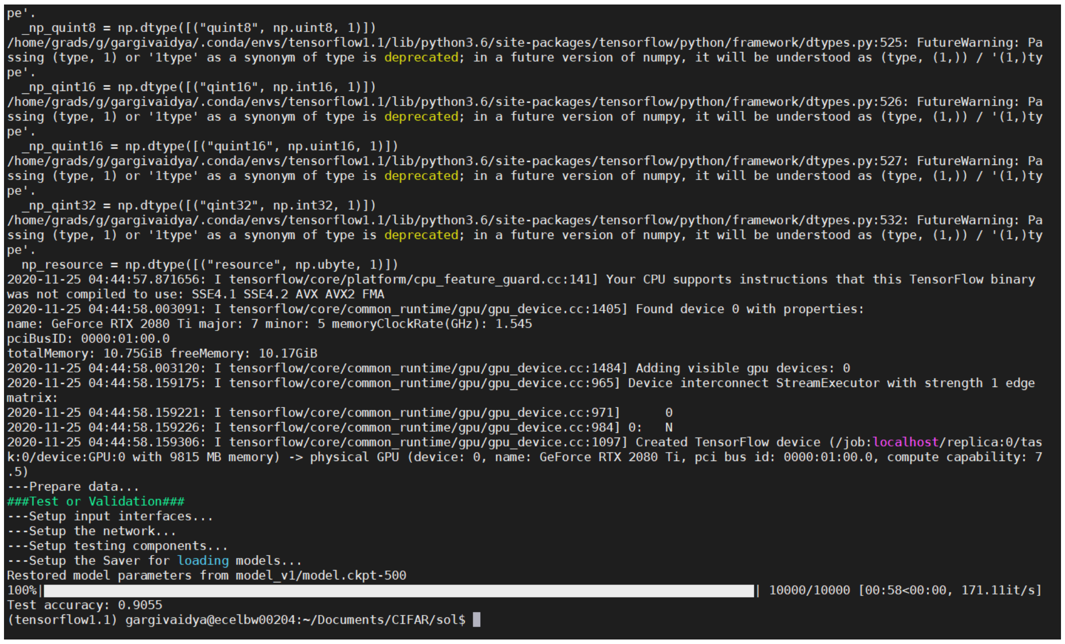
References

[1] Wang, X. & Girshick, R. & Gupta A. & He, K. (2018) Non-local Neural Networks, *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, 2018, pp. 7794-7803. doi: 10.1109/CVPR.2018.00813.

[2] Wang, F. & Jiang, M. & Qian C. & Yang, S. & Li C. & Zhang H. & Wang X. & Tang X. (2017) Residual Attention Network for Image Classification, *arXiv preprint* , arXiv:1704.06904.

[3] He, K. & Zhang, X. & Ren S. & Sun, J. (2016) Deep Residual Learning for Image Recognition, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[4] C. Cao et al., (2015) Capturing Top-Down Visual Attention with Feedback Convolutional Neural Networks, *IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 2956-2964, doi: 10.1109/ICCV.2015.338.



```
pe'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/grads/g/gargivaidya/.conda/envs/tensorflow1.1/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Pa
ssing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)ty
pe'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/grads/g/gargivaidya/.conda/envs/tensorflow1.1/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Pa
ssing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)ty
pe'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/grads/g/gargivaidya/.conda/envs/tensorflow1.1/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Pa
ssing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)ty
pe'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/grads/g/gargivaidya/.conda/envs/tensorflow1.1/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Pa
ssing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)ty
pe'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
2020-11-25 04:44:57.871656: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary
was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2020-11-25 04:44:58.003091: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1405] Found device 0 with properties:
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.545
pciBusID: 0000:01:00.0
totalMemory: 10.75GiB freeMemory: 10.17GiB
2020-11-25 04:44:58.003120: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1484] Adding visible gpu devices: 0
2020-11-25 04:44:58.159175: I tensorflow/core/common_runtime/gpu/gpu_device.cc:965] Device interconnect StreamExecutor with strength 1 edge
matrix:
2020-11-25 04:44:58.159221: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971]      0
2020-11-25 04:44:58.159226: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984] 0:   N
2020-11-25 04:44:58.159306: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1097] Created TensorFlow device (/job:localhost/replica:0/tas
k:0/device:GPU:0 with 9815 MB memory) -> physical GPU (device: 0, name: GeForce RTX 2080 Ti, pci bus id: 0000:01:00.0, compute capability: 7
.5)
---Prepare data...
###Test or Validation###
---Setup input interfaces...
---Setup the network...
---Setup testing components...
---Setup the Saver for loading models...
Restored model parameters from model_v1/model.ckpt-500
100%|████████████████████████████████████████████████████████████████████████████████████████████| 10000/10000 [00:58<00:00, 171.11it/s]
Test accuracy: 0.9055
(tensorflow1.1) gargivaidya@ecelbw00204:~/Documents/CIFAR/sol$ █
```

Figure 6: Public Dataset Test Accuracy for Baseline Residual Network

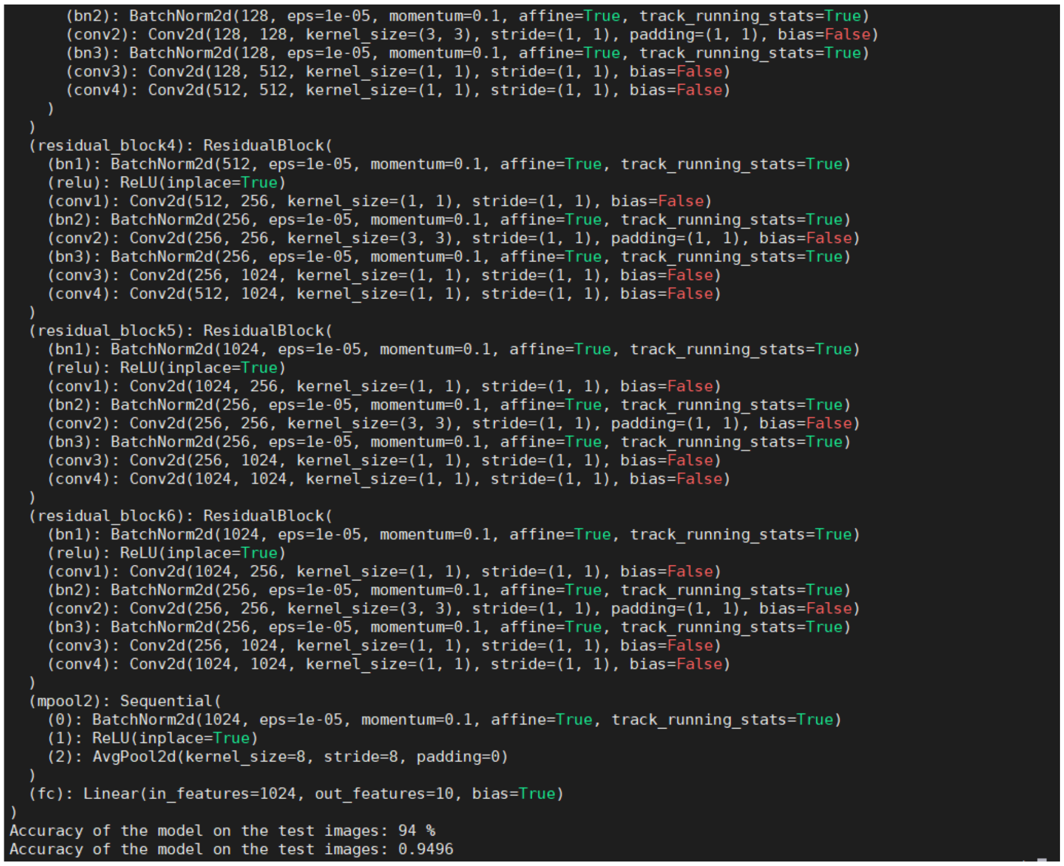
6 Appendix

Google Drive Link [here]

6.1 Code Structure

- $main.py$: Includes the code that loads the dataset and performs the training, testing and prediction.
- $DataLoader.py$: Includes the code that defines functions related to data I/O.
- $ImageUtils.py$: Includes the code that defines functions for any (pre-)processing of the images.
- $Configure.py$: Includes dictionaries that set the model configurations, hyper-parameters, training settings, etc. The dictionaries are imported to main.py
- $Model.py$: Includes the code that defines the model in a class. The class is initialized with the configuration dictionaries and should have at least the methods “train(X, Y, configs, [X_valid, Y_valid,])”, “evaluate(X, Y)”, “predict_prob(X)”. The defined model class is imported to and referenced in main.py.
- $residual_block.py$ Includes the code that defines the residual block with the convolution layers, batch normalization and RELU activaton layers.
- $attention_module.py$ Includes the code that defines the 3 stages of the Attention Module.
- $Network.py$: Includes the code that defines the network architecture with the stacking of residual blocks and attention modules.

6.2 Screenshots



```
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (conv4): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    )
  )
  (residual_block4): ResidualBlock(
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (conv4): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (residual_block5): ResidualBlock(
    (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (conv4): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (residual_block6): ResidualBlock(
    (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (conv4): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (mpool2): Sequential(
    (0): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU(inplace=True)
    (2): AvgPool2d(kernel_size=8, stride=8, padding=0)
  )
  (fc): Linear(in_features=1024, out_features=10, bias=True)
)
Accuracy of the model on the test images: 94 %
Accuracy of the model on the test images: 0.9496
```

Figure 7: Public Dataset Test Accuracy for Baseline Residual Network