Manarat International University

Department of Computer Science and Engineering
Neural Networks and Fuzzy Systems (CSE-433) &
Computer Vision & Robotics (CSE-437)

Cifar-10 Project Report (Fall 2019)

Problem Tile:	CIFAR-10 - Object Recognition in Images
Name of the Team:	**rdnasim**
Contestants Name:	Riadul Islam Nasim
	Kazi Mushfiqur Rahman
	Mahfuzur Rahman
Student ID:	1640CSE00467
	1640CSE00465
	1640CSE00519
Kaggle Account:	https://www.kaggle.com/rdnasim/
Git Repository link:	**https://github.com/rdnasim/CIFAR-10-Object-Recognition-in-Images**

Abstract—The purpose of this project is to build an object recognition system that can accurately classify images using CIFAR-10, a dataset in image recognition. We applied knowledge of machine learning to this computer vision system. Particularly, we investigated Softmax Regression, SVM and Convolutional Neural Networks to build this model, among which CNN generated the best result.

I.I INTRODUCTION

Object recognition is an important subfield in computer vision. It is easy for humans to recognize and classify objects in images, but usually not for machines. There are various obstacles in object recognition. For example, a picture only shows an object in 2D dimension but the angle of viewpoint can vary. There are also scale, color and illumination differences in a picture. Moreover, the intersection, deformation and intra-class variation in the objects themselves also make the problem difficult to handle.

However, object recognition has made great progress out of machine learning techniques. Over the past few decades, a bunch of algorithms and methods have been created to solve the problem, among which deep learning theory especially Neural Network generates the best performance. Thanks to the advancement in machine learning area, recently object recognition has thrived in a variety of commercial areas such as Automatic Focus, MobilEye and Google Goggles. It will further provide more applications in industrial and medical fields including manufacturing quality control and medical imaging.

Our project is from Kaggle competition and the dataset is publicly available. We focus object recognition particularly in color images.

I.II DATASET

Raw Data: CIFAR-10 is publicly available online. The dataset consists of 60,000 32x32 color images used for object recognition. We keep the split of train and test set in the official data. There are 50,000 images in the training set and 10,000 in the test set. In addition, there are 10 object classes in total and one image belongs to a certain class. There are no intersections among the 10 classes. The label classes are namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Both the training and test set are labeled for training and testing.

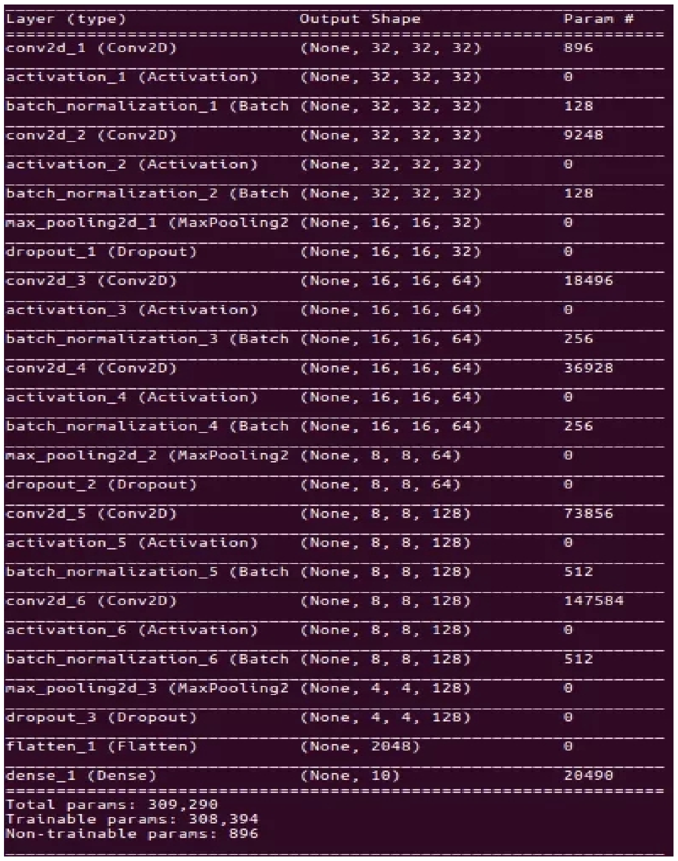
II METHOD

Reshape: The row vector for an image has the exact same number of elements if you calculate 32*32*3 == 3072. In order to reshape the row vector into (width x height x num_channel) form, there are two steps required. The first step is to use reshape function, and the second step is to use transpose function in numpy.

By definition from the numpy official web site, reshape transforms an array to a new shape without changing its data. Here, the phrase without changing its data is an important part since you don't want to hurt the data. reshape operations should be delivered in three more detailed step. The following direction is described in a logical concept.

1. Divide the row vector into 3 pieces, where each piece means each color channel.
   - the resulting array has (3 x 1024) matrix, which makes (10000 x 3 x 1024) tensor in total.
2. Divide the each 3 pieces further by 32. 32 is width and height of an image.
   - this results in (3 x 32 x 32), which makes (10000 x 3 x 32 x 32) tensor in total

In order to realize the logical concept in numpy, reshape should be called with the following arguments, (10000, 3, 32, 32). As you noticed, reshape function doesn't automatically divide further when the third value (32, width) is provided. You need to explicitly specify the value for the last value (32, height)
This is not the end of story yet. Now, one image data is represented as (num_channel, width, height) form. However, this is not the shape tensorflow and matplotlib are expecting. They are expecting different shape (width, height, num_channel) instead. You need to swap the order of each axes, and that is where transpose comes in.



```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 32)        0
________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128
________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 64)        0
________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856
________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 128)         0
________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584
________________________________________________________________
activation_6 (Activation)    (None, 8, 8, 128)         0
________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 128)         512
________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 128)         0
________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896
________________________________________________________________
```

Fig 1: CNN model summary

The process of building a Convolutional Neural Network majorly involves four major blocks show below.

Convolution layer ==> Pooling layer ==> Flattening layer ==> Dense/Output layer

The transpose can take a list of axes, and each value specifies an index of dimension it wants to move. For example, calling transpose with argument (1, 2, 0) in an numpy array of (num_channel, width, height) will return a new numpy array of (width, height, num_channel).

Deep Neural Network Architecture: The advantage of multiple layers is that they can learn features at various levels of abstraction. For example, if you train a deep CNN to classify images, you will find that the first layer will train itself to recognize very basic things like edges, the next layer will train itself to recognize collections of edges such as shapes, the next layer will train itself to recognize collections of shapes like wheels, legs, tails, faces and the next layer will learn even higher-order features like objects (truck, ships, dog, frog etc). Multiple layers are much better at generalizing because they learn all the intermediate features between the raw input and the high-level classification. At the same time, there are few important aspects which need to be taken care off to prevent over-fitting. Deep CNN are harder to train because:

a) Data requirement increases as the network becomes deeper.
b) Regularization becomes important as number of parameters (weights) increases in order to do learning of weights from memorization of features towards generalization of features.

Having said that we will build a 6 layered convolution neural network followed by flatten layer. The output layer is dense layer of 10 nodes (as there are 10 classes) with softmax activation. Fig 1: CNN is a model summary.

Normalize: Normalize function takes data, x, and returns it as a normalized Numpy array. x can be anything, and it can be N-dimensional array. In this story, it will be 3-D array for an image. Min-Max Normalization (y = (x-min) / (max-min)) technique is used, but there are other options too. By applying Min-Max normalization, the original image data is going to be transformed in range of 0 to 1 (inclusive). A simple answer to why normalization should be performed is somewhat related to activation functions.

For example, sigmoid activation function takes an input value and outputs a new value ranging from 0 to 1. When the input value is somewhat large, the output value easily reaches the max value 1. Similarly, when the input value is somewhat small, the output value easily reaches the max value 0.

For another example, ReLU activation function takes an input value and outputs a new value ranging from 0 to infinity. When the input value is somewhat large, the output value increases linearly. However, when the input value is somewhat small, the output value easily reaches the max value 0.

Now, when you think about the image data, all values originally ranges from 0 to 255. This sounds like when it is passed into sigmoid function, the output is almost always 1, and when it is passed into ReLU function, the output could be very huge. When back-propagation process is performed to optimize the networks, this could lead to an exploding/vanishing gradient problems. In order to avoid the issue, it is better let all the values be around 0 and 1.

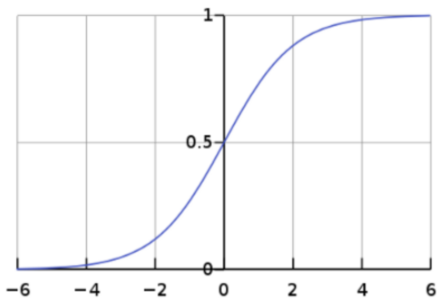


Fig 2: sigmoid function

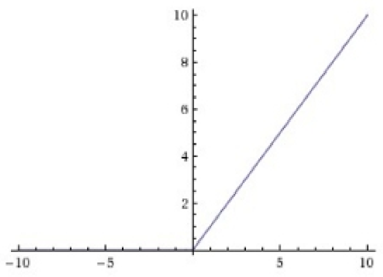
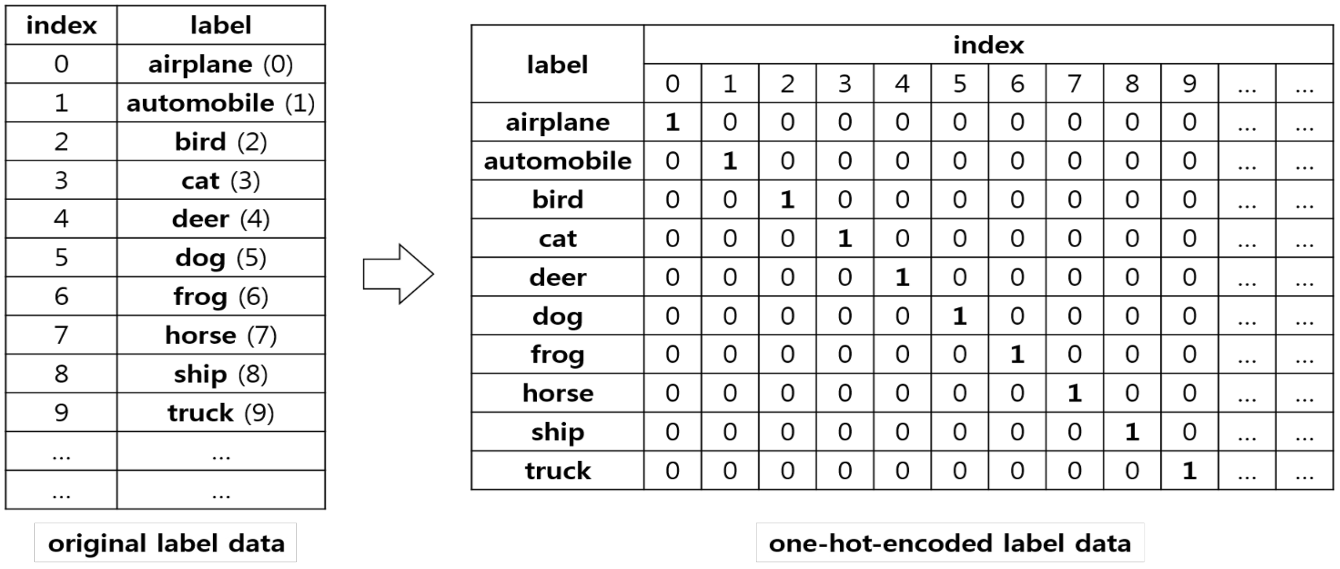


Fig 3: ReLU function

One-hot encode: Later, I will explain about the model. For now, what you need to know is the output of the model. It is a set of probabilities of each class of image based on the model's prediction result. In order to express those probabilities in code, a vector having the same number of as the number of classes of the image is needed. For instance, CIFAR-10 provides 10 different classes of the image, so you need a vector in size of 10 as well.

Also, our model should be able to compare the prediction with the ground truth label. It means the shape of the label data should also be transformed into a vector in size of 10 too. Instead, because label is the ground truth, you set the value 1 to the corresponding element.

one_hot_encode function takes the input, x, which is a list of labels(ground truth). The total number of element in the list is the total number of samples in a batch. one_hot_encode function returns a 2 dimensional tensor, where the number of row is the size of the batch, and the number of column is the number of image classes.



index	label
0	**airplane** (0)
1	**automobile** (1)
2	**bird** (2)
3	**cat** (3)
4	**deer** (4)
5	**dog** (5)
6	**frog** (6)
7	**horse** (7)
8	**ship** (8)
9	**truck** (9)
...	...
...	...

original label data

label	index											
	0	1	2	3	4	5	6	7	8	9	...	...
airplane	**1**	0	0	0	0	0	0	0	0	0	...	...
automobile	0	**1**	0	0	0	0	0	0	0	0	...	...
bird	0	0	**1**	0	0	0	0	0	0	0	...	...
cat	0	0	0	**1**	0	0	0	0	0	0	...	...
deer	0	0	0	0	**1**	0	0	0	0	0	...	...
dog	0	0	0	0	0	**1**	0	0	0	0	...	...
frog	0	0	0	0	0	0	**1**	0	0	0	...	...
horse	0	0	0	0	0	0	0	**1**	0	0	...	...
ship	0	0	0	0	0	0	0	0	**1**	0	...	...
truck	0	0	0	0	0	0	0	0	0	**1**	...	...

one-hot-encoded label data

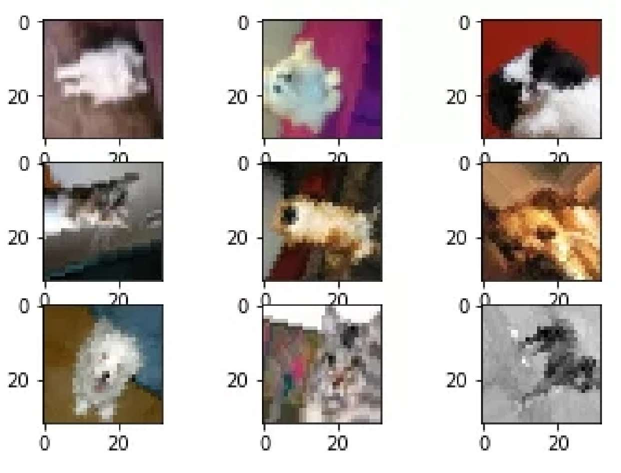
Fig 4: one-hot-encoding process

Also, our model should be able to compare the prediction with the ground truth label. It means the shape of the label data should also be transformed into a vector in size of 10 too. Instead, because label is the ground truth, you set the value 1 to the corresponding element.

one_hot_encode function takes the input, x, which is a list of labels(ground truth). The total number of element in the list is the total number of samples in a batch. one_hot_encode function returns a 2 dimensional tensor, where the number of row is the size of the batch, and the number of column is the number of image classes.

Data Augmentation: In Keras, We have a ImageDataGenerator class that is used to generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely. The image data is generated by transforming the actual training images by rotation, crop, shifts, shear, zoom, flip, reflection, normalization etc.

To know more about all the possible arguments (transformations) and methods of this class, refer to Keras documentation here. For example, one can use flow(x, y) method that takes numpy data & label arrays, and generates batches of augmented/normalized data. It yields batches indefinitely, in an infinite loop. Below is the python snippet for visualizing the images generated using flow method of ImageDataGenerator class.

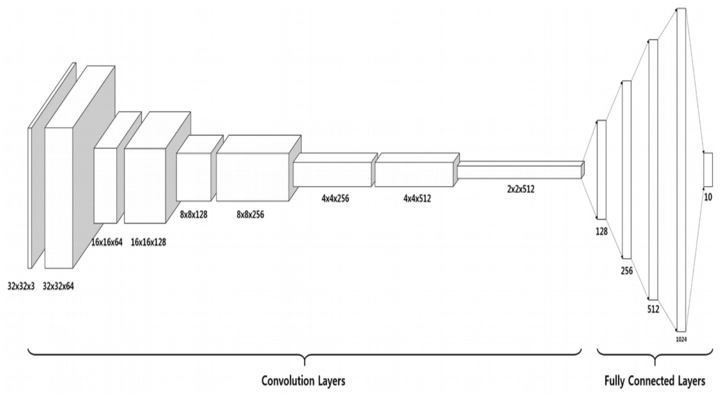


Regularization: Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Given below are few techniques which were proposed recently and has become a general norm these days in convolutional neural networks.

Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. The reduction in number of parameters in each step of training has effect of regularization. Dropout has shown improvements in the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Kernel_regularizer allows to apply penalties on layer parameters during optimization. These penalties are incorporated in the loss function that the network optimizes. This argument in convolutional layer is nothing but L2 regularisation of the weights. This penalizes peaky weights and makes sure that all the inputs are considered. During gradient descent parameter update, the above L2 regularization ultimately means that every weight is decayed linearly, that's why called weight decay.

Hyperparameters: The hyper parameters are chosen by a dozen time of experiment. One thing to note is that learning_rate has to be defined before defining the optimizer because that is where you need to put learning rate as an constructor argument.



Model	200-epoch accuracy	Original accuracy
CNN	87.01%	81.07%
DenseNet	89.80%	86.05%
ResNet20 v1	90.16 %	86.25 %
ResNet32 v1	90.46 %	86.49 %
ResNet44 v1	90.50 %	86.83 %
ResNet56 v1	91.71 %	87.01 %

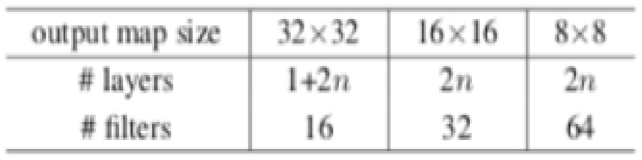
Fig 6: Accuracy Rate for Different Models

Trains a ResNet

Introduction: ImageNet dataset consist on a set of images (the authors used 1.28 million training images, 50k validation images and 100k test images) of size (224x224) belonging to 1000 different classes. However, CIFAR10 consist on a different set of images (45k training images, 5k validation images and 10k testing images) distributed into just 10 different classes.

Because the sizes of the input volumes (images) are completely different, it is easy to think that the same structure will not be suitable to train on this dataset. We cannot perform the same reductions on the dataset without having dimensionality mismatches.

We are going to follow the solution the authors give to ResNets to train on CIFAR10, which are also tricky to follow like for ImageNet dataset. CIFAR-10 and Analysis, we find the following table

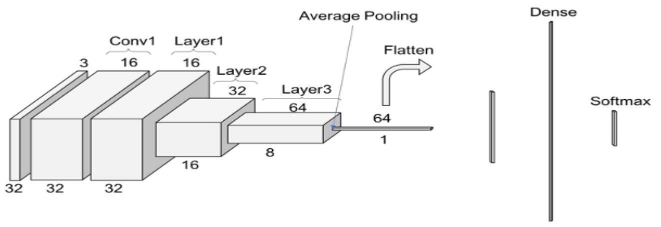


output map size	32×32	16×16	8×8
# layers	1+2n	2n	2n
# filters	16	32	64

Schema for ResNet from the paper

Let’s follow then the literal explanation they give to construct the ResNet. We will use n=1 for simplification, leading to a ResNet20.

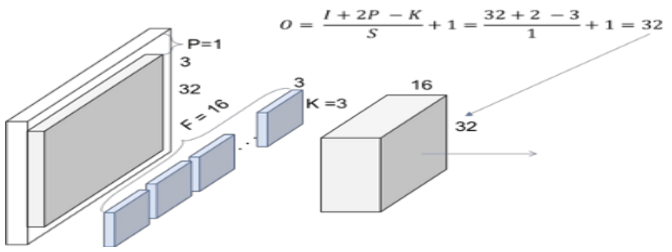
Structure: Following the same methodology of the previous work on ResNets, let’s take a look at the overall picture first, to go into the details layer by layer later.



Scheme for ResNet Structure on CIFAR10

Convolution 1: The first step on the ResNet before entering into the common layer behavior is a 3x3 convolution with a batch normalization operation. The stride is 1 and there is a padding of 1 to match the output size with the input size. Note how we have already our first big difference with ResNet for ImageNet, that we have not include here the max pooling operation in this first block.
Figure 3. Conv1

We can check with Figure 2 that the output volume of Conv1 is indeed 32x32x16.


$O = \frac{I + 2P - K}{S} + 1 = \frac{32 + 2 - 3}{1} + 1 = 32$
P=1
3
32
F = 16
3
K =3
16
32

Conv1

Layer 1: The rest of the notes from the authors to construct the ResNet are:
- Use a stack of 6 layers of 3x3 convolutions. The choice of will determine the size of our ResNet.

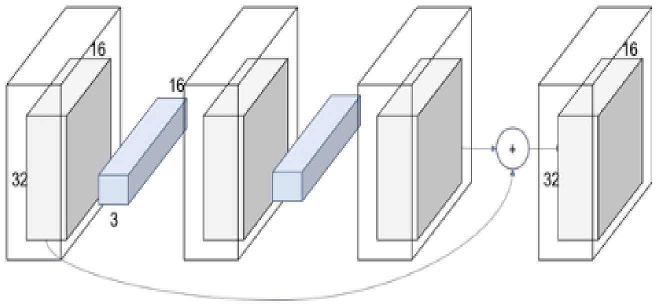
- The feature map sizes are {32, 16, 8} respectively with 2 convolutions for each feature map size. Also, the number of filters is {16, 32, 64} respectively.

- The down sampling of the volumes through the ResNet is achieved increasing the stride to 2, for the first convolution of each layer. Therefore, no pooling operations are used until right before the dense layer.

- For the bypass connections, no projections will be used. In the cases where there is a different in the shape of the volume, the input will be simply padded with zeros, so the output size matched the size of the volume before the addition.

This would leave Figure 4 as the representation of our first layer. In this case, our bypass connection is a regular Identity Shortcut because the dimensionality of the volume is constant thorough the layer operations. Since we chose n=1, 2 convolutions are applied within the layer 1.
Figure 4. Layer 1

We still can check from Figure 2 that the output volume of Layer1 is indeed 32x32x16. Let's go deeper!

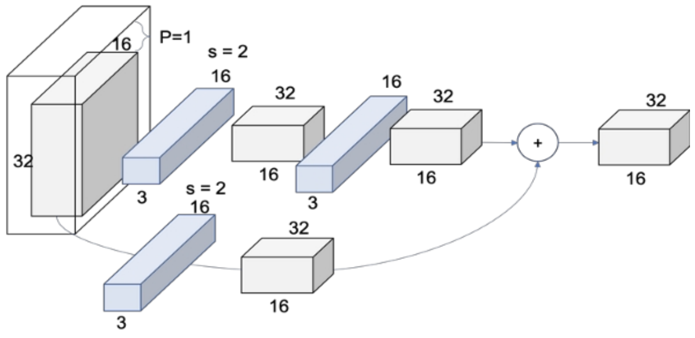


Layer 1

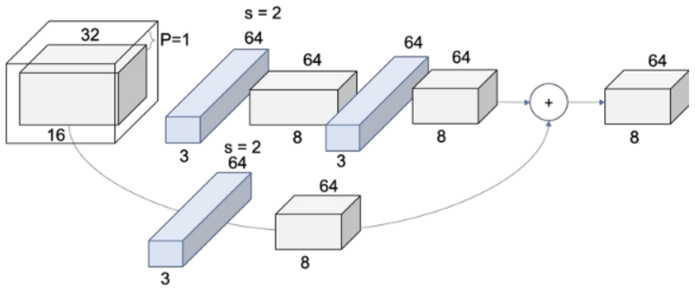
Layer 2: We are going to see now how to deal with the down sampling of the input volume. Remember we are following the same structure and notation as the work on ImageNet dataset. Make sure you take a look if not follow any step, as there is explained in more detail.

For both layer 2 and next layer 3 the behavior is equivalent to layer 1, with the exception that the first convolution uses a stride of 2, and therefore the size of the output volume is half of the input volume (with the padding of 1). This implies that also the shortcut connection will require an extra step, to adjust the volumes' sizes before the summation.

The final look of the entire layer 2 is shown in Figure 6. We can see how the convolution with stride 2 is used in the skip connection for the down sample as well as in the first convolution of the layer. Also, we can check with the table from the paper that we have indeed a 16x16x32 volume.



Layer 2



Layer 3

Summary: The ResNets following the explained rules built by the authors yield to the following structures

Number of Layers	Number of Parameters
ResNet 20	0.27M
ResNet 32	0.46M
ResNet 44	0.66M
ResNet 56	0.85M

III HARDWARE AND SOFTWARE SYSTEM

Operating System linux - Ubuntu 18.04
Intel i5 5th generation processor with 8 GB RAM.
Required memory for dataset and results: 4GB.
AMD RADEON R5 M255
Visual Studio Code
Pytorch version 1.1
Keras 2.3.0

IV RESULTS

We made learning model for CNN, Resnet and DenseNet. See Fig. 6. 4. The results of different experimental methods are listed in Tab. I showing both epoch and original accuracies of different models. Thats the best score of Accuracy 87.01%

[1] ## V CONCULUSION

We first tried out CNN and DenseNet and gained a general view of the image recognition problem. It was non-linearly separable and suffered overfitting. Then we designed our own Convolutional Neural Networks, ResNet and DenseNet and improved the performance greatly. We further implemented two well designed CNNs and obtained the high accuracy rate of 90%. We are fascinated by the great potential of Convolutional Neural Networks and we will dig more into the great variety of Neural Networks and train and test networks using GPU.

A CKNOWLEDGEMENT

Thanks for Sir Mahedi Hasan teaching this excellent course. Thanks for Classmate's suggestions on our project. Thanks for all those who gave a suggestion on our project!

REFERENCE

[1] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html
[2] https://keras.io/examples/cifar10_resnet/
[3] https://github.com/Mahedi-61/MIU_CS_411
[4]
https://appliedmachinelearning.blog/2018/03/24/achieving-90-accuracy-in-object-recognition-task-on-cifar-10-dataset-with-keras-convolutional-neural-networks/
[5] https://github.com/bamos/densenet.pytorch
[6] https://www.kaggle.com/c/cifar-10/
[7] https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c
[8] https://www.kaggle.com/c/cifar-10/discussion/83473#487186
[9] https://github.com/kuangliu/pytorch-cifar

1 Reference