Using VGG16 for Multiclass Classification on CIFAR10

Kamalnain Taank
University of Adeliade
North Terrace Adelaide SA 5000
a1805326@student.adelaide.edu.au/

Abstract

In Machine Learning or specifically in Neural networks, Convolutional neural network or CNN is most widely used for image classification or object recognition or detection. So in this paper, we will be using one of the most widely used CNN architecture known as VGG and the variant we will be using is VGG16 which was proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The well known CIFAR10 data set is used for training and validation purpose. We developed a VGG16 model from scratch and done some modifications and compared it with VGG16 present in Torchvision Models.

1. Introduction

Computer vision is booming in today's era, every industry is investing tons of resources in developing models which can perform image classification, object detection etc better and better. It is a field of artificial intelligence which helps machine to recognize things like humans do. Image classification is a key element of computer vision. In this paper we are using CIFAR10 data set which is an established computer vision dataset used for image classification. The model we used for the classification purpose is VGG16 which is a well known and famous model submitted to Imagenet Large Scale Visual Recognition Challenge 2014 (ILSVRC2014). Here, we will be modifying some of it's layers to get the best results in the least possible time.

1.1. VGG16

VGG16 is considered to be one of the most prominent computer vision model till now. It focuses on convolutional layers than the large number of hyper parameters. It consistes of 16 convolutional layers, a lot of filters and 3 fully connected layers at the end. There is one caveat with this model, that it takes a lot of time to train as it contains around 138 million parameters. So we try to reduce its converging time using some techniques which will be discussed further. We chose this architecture among other VGG variants because it is less computationally expensive than VGG19 and a tad bit more deep than VGG11 so that we can get better results. We used VGG16 with batch normalization from Torchvision models library. We also made custom VGG16 model with batch normalization with the same configuration of layers and performed some experiments to measure the difference between these two models.

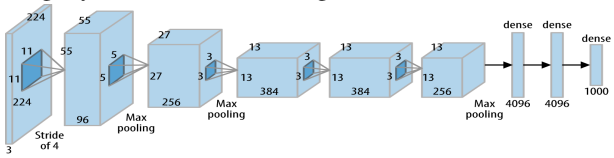
1.2. Dataset

The dataset we are using CIFAR10 which is one of the most popular dataset for object recognition. It was created in 2006 and contains 60,000 tiny images of 32x32 pixel resolution which belong to 10 different classes meaning there are 6000 images per class in this dataset. Those 10 different classes are cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Since the image size is very small i.e. 32x32 pixels, it is quite faster to train the model on and is good for evaluation purposes as well.

2. Background

2.1. AlexNet

There are an ample amount of models developed for multi class image classification. VGG was a breakthrough after AlexNet which came out in 2012. AlexNet used ReLU like VGG and was optimized for performing on multiple GPUs and overlapping pooling. To overcome the overfitting problems, it used data augmentation and dropout techniques. It was 8 layers deep which consists of 5 convolutional layers and 3 fully connected layers along with 3 max pooling layers as shown in the figure below:



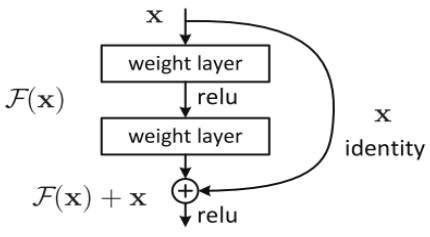
The very first convolutional layer consistes of 96 filters of 11x11. These filters have a stride of 4 and padding of 2 pixels and the remaining convolutional layers have 256,

384, 384 and 256 filters respectively of size 5x5, 3x3, 3x3 and 3x3 respectively with a stride and padding of 1 pixel. Depth of a model is an important aspect which improves performance and accuracy of a model. But AlexNet focused on small kernel sizes and strides in the very first layer. On the contrary, VGG addresses this with the increased depth i.e. more layers.

2.2. Related work

Many researches are being conducted on improving the multi class image classification nowadays and various state of the art models have been made like ResNet, RexNetX, Inception V3 by google and many more. Akwasi Darkwah Akwaboah[1] developed a model named as NET III which is inspired by AlexNet and VGG which also uses CIFAR10 for training and validation purpose. NET III contains 6 covolutional layers with channel size 96,96,128,128,128 and 128 which uses filter of size 3x3 with a stride of 1 pixel and ReLU activation function. The final layer uses softmax function. Accuracy achieved by this model on training and validation set is 73.2% and 76.60% respectively with 40 epochs. This model is not deep enough and softmax function at the final layers gives probabilities and gives results even if the image or input gives does not exists, these could be the reasons why accuracy is a bit low.

Deeper neural networks need a lot of computational power and takes a lot of time. Adding more layers in the model poses a problem of vanishing gradient, many ways were emplyed to tackle this problem but results were not upto the mark but ResNet solves this by introducing connection called "identity shortcut connection" which can skip layers. An illustration is shown below:



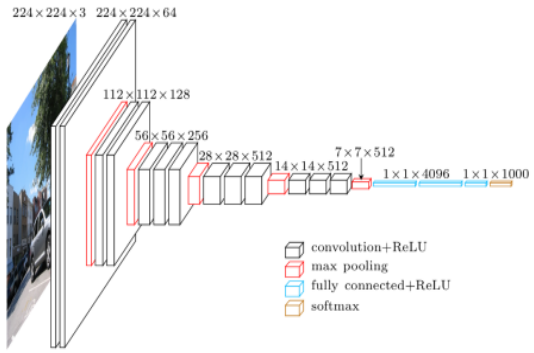
a residual block

This connection prevents the degradation of the model if layers are stacked, it does this by skipping layers which ease the training substantially. Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun[2] implemented ResNet20 ResNet32, ResNet44, ResNet56, ResNet110 upto ResNet1202 on CIFAR10 dataset and got the test error of 8.75%, 7.51%, 7.17%, 6.97%, 6.43% and 7.97% respectively. Their results outperforms many others as well.

3. Methodology

3.1. About Models

VGG model focuses on depth of the model after AlexNet. It takes the input image of 224x224 pixel resolution with 3 rgb channels. The images are loaded in the range of [0,1] then are normalized with mean of [0.485, 0.456, 0.406] and standard deviation of [0.229, 0.224, 0.225] during preprocessing steps. There are 16 weight layers in VGG16 which includes 13 convolitional layers with filters with a kernel size of 3 and 2 fully connected layers again with filters of same size and these fully connected layers have4 096 channels by default and lastly 3rd fully connected layer with 1000 channels. Every hidden layer uses ReLU as an activation function which reduces training time significantly and after every 3 hidden convolutional layers, there is a max pooling layer. The illustration is given below:



Architecture of VGG16

In the above image, activation function used at the end is softmax function. This activation funcition gives probabilities and their sum is zero so in the case of image classification, it can give an output even if the image does not exists. We used VGG16 model with batch normalization. Batch normalization helps layers to coordiante while updating. It normalizes inputs going to the layer which helps in drastically reducing the number of training epochs needed to train the model. The custom VGG16 architecture which we developed has some modifications to it. The last 2 fully connected layers have 512 channels and the last fully connected layers has 10 channels because CIFAR10 dataset have images which belongs to 10 different classes. Batch normalization is being performed after every hidden layer. Lastly the activation function employed at the final layer is SELU. There is one problem with SELU that is is relatively a new activation function so it not much explored.

4. Experiments

The very first thing we did to improve the accuracy of a model is data augmentation during the preprocessing. Tech-

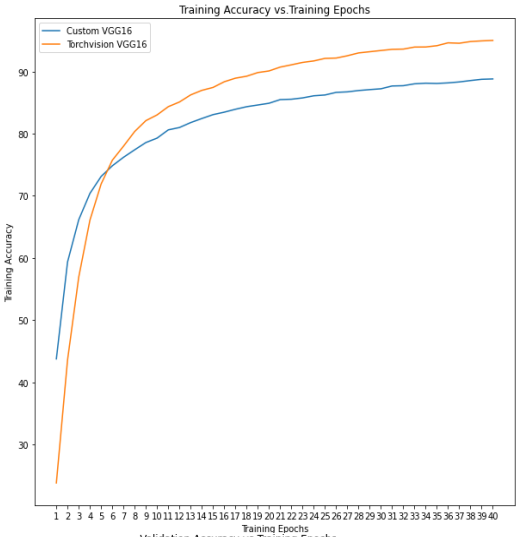
niques like Random cropping with padding and random horizontal flip are employed for both the training and testing datasets.

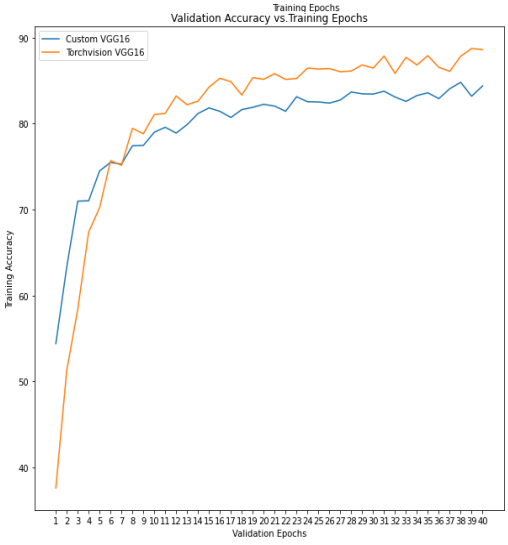
```
preprocess_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
```

We used the batch size of 128 to avoid large and very small gradient updates so the updates are of about the same size. The activation function used with hidden and final layer is SELU for the custom model and no change is made to the default torchvision VGG16 model. The default model uses ReLU activation function with hidden convolutionl layers which poses a problem of exploding gradient and DEAD ReLU where weights might not be updated to new values. So we used SELU which eliminates the problem of vanishing gradient and exploding gradient. Also normalization is a bit faster which results in increasing the speed of convergence of the model. Stochastic gradient descent us used for optimization because it is computationally fast. Because it updates faster, the steps which are taken towards minima has some oscillations which can help in getting out of the local minimas[3] and there is a good chance that we will end up in global minima. The learning rate we used here is 0.01 which is a bit high. But we are using stochastic gradient descent over mini batches, there is less chance of missing the global minima of the loss function. We are also using weight decay of 0.0005 to prevent the weights from becoming too large. Also momentum of 0.9 is used to increase the convergence of the model as it will accelerate vectors in the correct directions. Weights has been initialized using He Initialization which

$$Var(w_i) = \frac{2}{fan\_in}$$

is: where fan_in is the number of inputs. The accuracy of Custom VGG16 after 40 epochs on training and validation set is almost 89% and 84.4% and accuracy of the torchvision VGG16 with batch normalization on training and validation set is almost 95% and 88.6% respectively. Illustrations are given below potraying learning cruve of both the models on training and validation set.





5. Code

The code for this experiment is given on this Link. Please inform if you cannot access the link.

6. Conclusion

We tried various things like, using SELU activation funciotn to avoid exploding and vanishing gradient, used weight initialization to get a head start on training. Also used Stochastic gradient descent to speed up the convergence. These method helped in decreasing the time taken to train the model significantly, also some tweaks made to VGG16 custom model helped in training one epoch in arounf 30 seconds whereas for default model, tima taken was almost 1 minute which is good gap.

1. References

[1]. Convolutional Neural Network for CIFAR-10 Dataset Image Classification by Akwasi Darkwah Akwaboah See: Convolutional Neural Network for CIFAR-10 Dataset Image Classification by Akwasi Darkwah Akwaboah

[2]. Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Re nand Jian Sun See: Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Re nand Jian Sun

[3].Stochastic vs Batch Gradient Descent by Divakar Kapil See: SGD

[4]. A Gentle Introduction to Batch Normalization for Deep Neural Networks by Jason Brownlee BN