NAME – VIKRAM KUMAR

ENTRY NO. - 2016CE10237

Group - 02

Assignment2 - part2

Convolution Neural Network

Part A

Dataset

We are using Cifar-10 dataset for our assignment which represents 10 classes which are following- airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks, Our training dataset contains 50,000 RGB images of shape (32,32,3) and test dataset has 10K examples.

Network Architecture

As mentioned in assignment statement, the architecture of convolutional neural network should be following :-

i. **CONV1:** Convolutional layer with 3 RGB inputs and 64 outputs with 3×3 filter size.
ii. **POOL1:** 2×2 max pooling layer.
iii. **CONV2:** Convolutional layer with 128 outputs 3×3 filter size.
iv. **POOL2:** 2×2 max pooling layer.
v. **FC1:** Fully Connected Layer with 512 outputs
vi. **FC2:** Fully Connected Layer with 256 outputs
vii. **SM:** Soft-max layer for classification with 10 outputs.

and there should be one batch normalization layer before activation of last layer(softmax activation)

Training

For training we used Keras module and its sequential model for creating architecture and tried multiple optimization algorithms and kept default setting(according to keras) for most of those optimization algorithms to get an

idea how well different optimization techniques works without any regularization technique.

We used SGD with momentum, RMSprop and Adam optimization algorithms to train our model, RMSprop and Adam is adaptive learning-method of optimization.

1. **SGD with momentum**

   With stochastic gradient descent with momentum we actually calculate weighted gradients for our loss functions which proves better then classic Stochastic gradient descent which does not calculate exact gradient of loss function but on a small batch of training data which means we are not going always in optimal direction which leads to random fluctuation in loss, using weighted average(using past gradient history for calculating current graident) improves the optimization.

   lr=0.01,momentum=0.8,epochs=15,batch_size=100,metrics='accuracy'' loss='categorical_crossentrop”

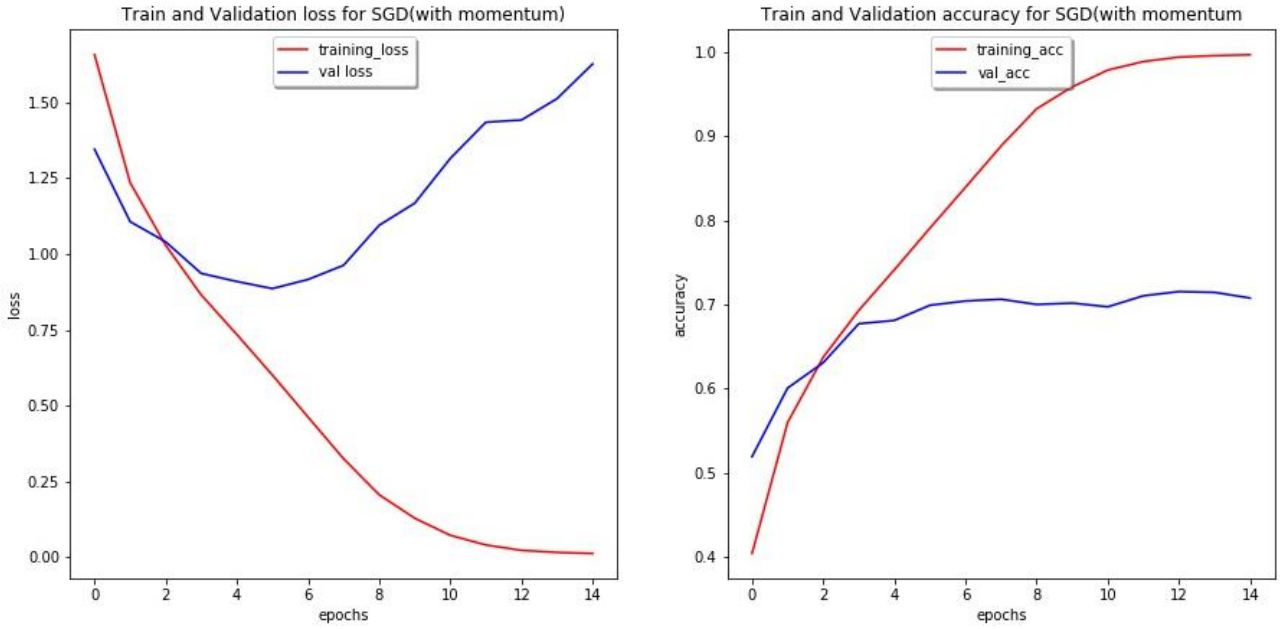

Fig. 1

2. **RMSprop**

RMSprop is also an adaptive learning-method algorithm, RMSprop is basically resolves the problem of adagrad which is radically diminshing of learning rate, RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients.
Learning rate – 0.001 and rho- 0.9, epochs – 10

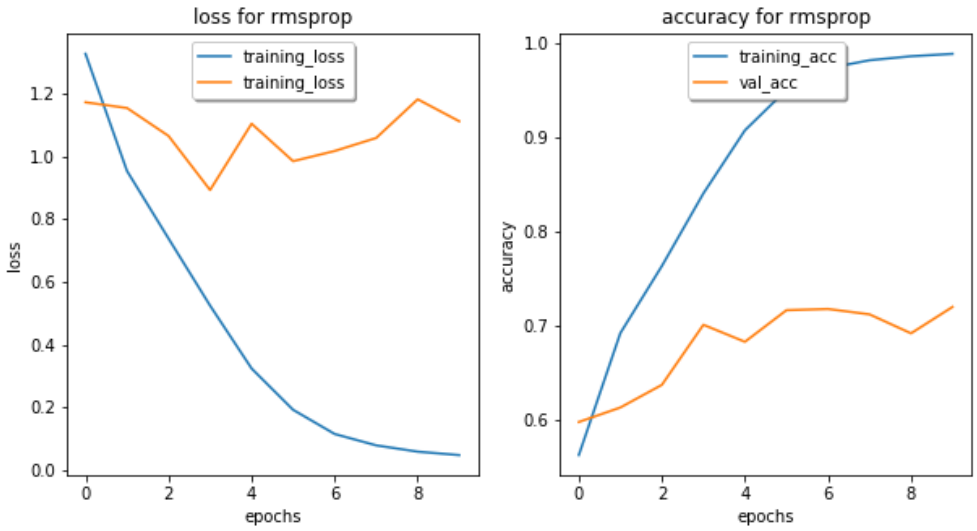

Fig. 2

3. Adam

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter,In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients similar to momentum, the decaying averages of past and past squared gradients are biased towards zero for initial time steps(due to zero initialization) therfor biased correction is applied in this method of adaptive learning. The authors propose default values of 0.9 for β1, , 0.999 for β2.

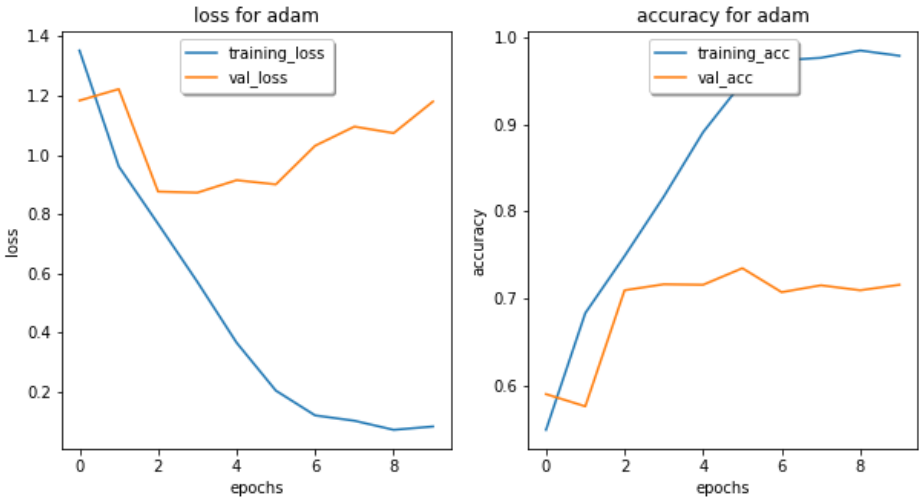
loss for adam
training_loss
val_loss
loss
epochs
accuracy for adam
training_acc
val_acc
accuracy
epochs

Fig.3

Leaning rate used for Adam is 0.001, epochs 10, batch_size =100

As can be seen from above plots that all of these algorithms gives almost same performance and lacks in generalization, we are getting validation accuracy almost 70% for our all optimization techniques which we used and can be checked for more optimization methods. various regularization techniques can be used for improving generalization such as Dropout and Data augmentation.

Finally used data augmentation to regularize the model, 25 epochs with learning rate 0.001 and batch size 300, below is the plot for accuracy and loss for the same model – as it can be seen that regularizing techniques work well for the model

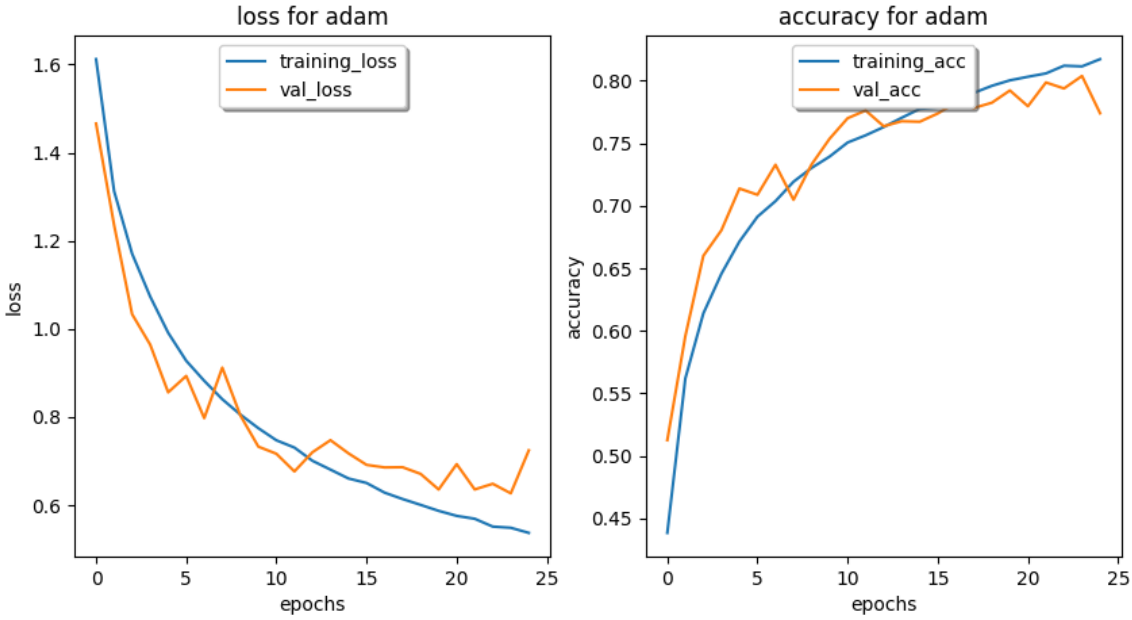

Fig 4

Part B

The problem arises with CNN that when they go deeper flow of information from input to output layer and flow of gradient in reverse direction becomes so big that they vanishes which crates problem for network to learn, therefore various SOTA networks try to solve this problem so that deep network can be possible. ResNet, Highway Networks, Fractal Networks does the same thing.

DenseNet try to solve this problem very efficiently, they keep densely connected layers which takes input from all of its previous layers. DenseNets exploit the potential of the network through feature reuse.

Problem is solved by DenseNet :

DenseNet simply overcome the problem of number of parameters which needs to be learn as compared to traditional CNN networks, ResNet try to solve the same problem of vanishing gradients in deep networks but it can be seen in ResNet that there are many layers which barely contribute to learn the network therefor those layers can be dropped, In fact number of parameters in ResNet is so big, DenseNet uses very small number of features such as 12.

Another problem which is gradient flow in deep network, since in DenseNet each layer has direct access to gradient from output layers, therefore flow of gradient becomes easy compared to traditionally deep CNN.

DenseNet

DenseNet is a new CNN architecture that reached **State-Of-The-Art** results on classification datasets such as CIFAR, SVHN and ImageNet using less parameters.

Architecture

DenseNet is composed on DenseBlocks and in those blocks the layers are densely connected together, each layer in dense block receives input all previous layers output feature maps.

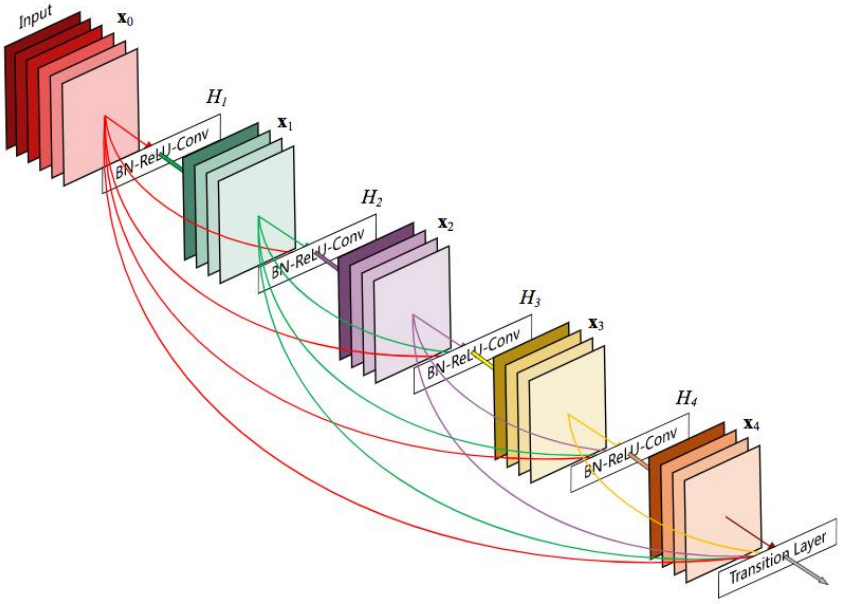

Fig.5

Dense Block – A dense block is composed layers which are connected to all its previous layers, a single layer in dense block look like this

1. Batch Normalization
2. ReLu Activation
3. 3x3 convolution

Transition Layers – These layers ensures down sampling so that features map size does not boost up, a single transition layer is made up of –

1. Batch Normalization
2. 1X1 convolution
3. Average Pooling

Bottleneck Layers – this layer ensures that the number of trainable parameters remains less. A bottleneck layer is made up of-

1. BN
2. Relu
3. 1x1 convolution with 4*K features where K is growth rate
4. BN
5. Relu
6. 3x3 convolution

Growth Rate – The number of output feature maps of a layer in dense block is defined as growth rate, authors recommend 32 value of growth rate but SOTA specified that 12 can give optimal performance.

Compression – Compression happens in transition layers and it can take values between 0 and 1. For our case we kept it equal to 0.5

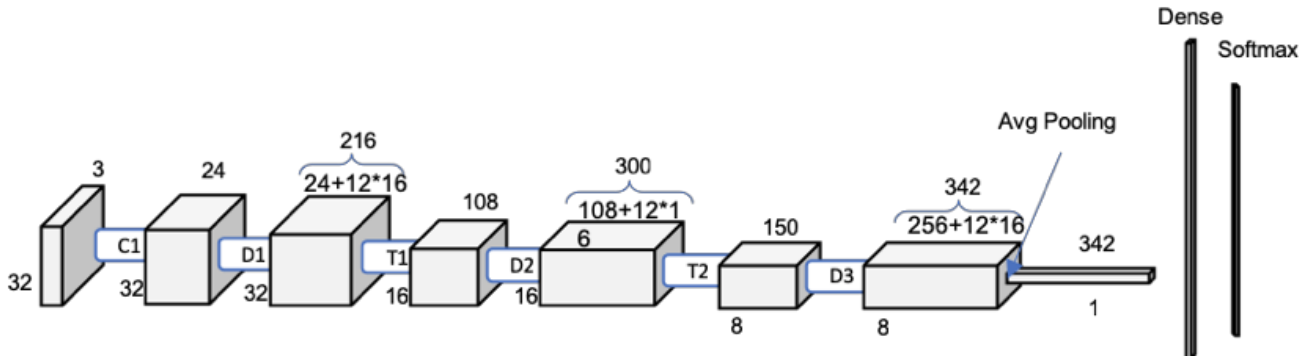

Fig. 6, DenseNet-100-12

It is also easy to train the deep network where layers are densely connected, it makes gradients to flow easily across the network.

Results for this assignment

We tried a DenseNet-BC-40-12 with depth 40 and growth rate 12, 'B' represents for bottleneck layers in architectures and 'C' represents for compression that shows that we don't keep number of features same across the transition layers.

Architecture used – we used **DenseNet-BC-40-12** with **Data Augmentation**

1. Dense blocks- 3
2. Number of layers in each dense block – 6
3. Growth rate – 12
4. Compression factor – 0.5
5. Depth of DenseNet – 40
6. Number of epochs – 23
7. Learning rate- 0.01 with Adam optimization , Batch size – 300

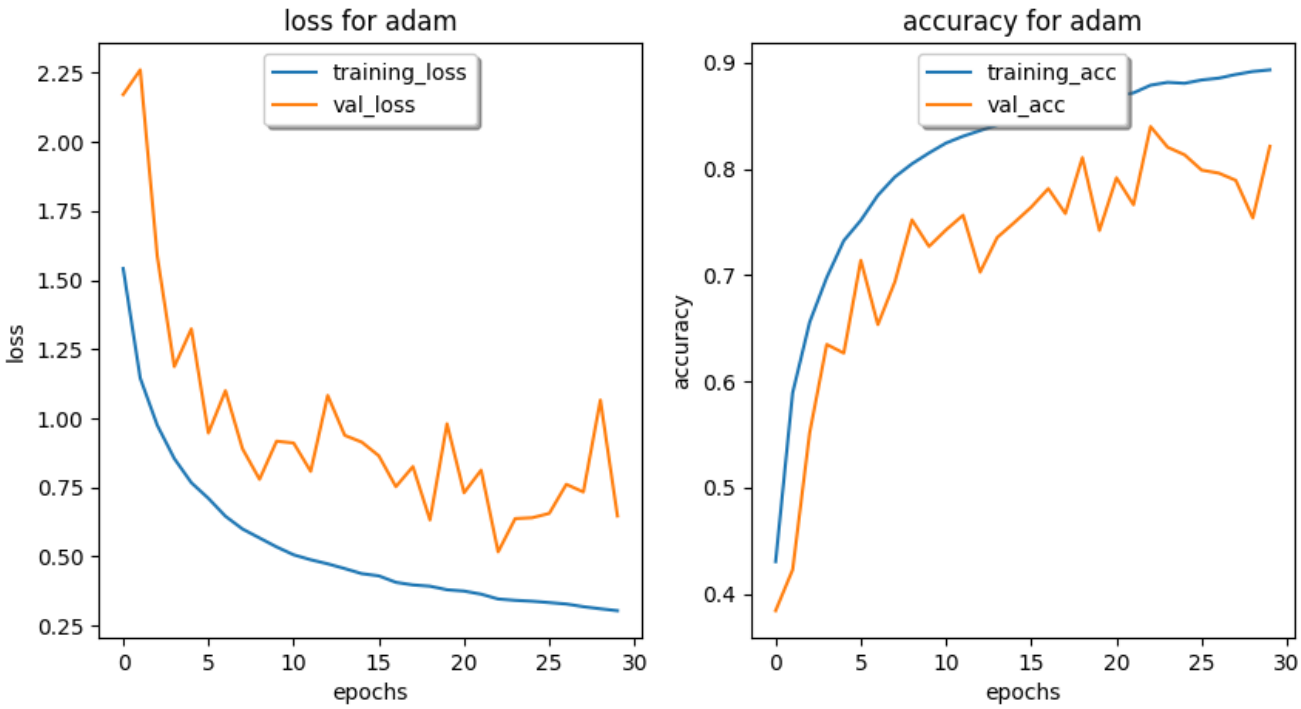

Fig. 7

As it can be seen from plots above that still there is scope for training loss to decrease and increase of train accuracy and also validation accuracy. But there is constrain of 1 hour for training for this assignment we run only 30 epochs(takes around 40 minutes) which brings us around 80% accuracy which is pretty impressive.