CIFAR-10 classification using autoencoder features

1. DATA:

I have removed 50% of training data for classes bird, deer and truck of Cifar-10 data i.e. 2500 images. All other classes have 5000 training images. Test data has 10000 images which are all used as the validation data while training the neural network in the current notebooks since the performance of the network is anway being evaluted on this test data only.

I did take a chunk of 30% from training images and used as validation set to evaluate the intermediate models but the accuracy on this validation and 10000 test data images was similar so eventually used up test dataset for validation.

2. TRIAL-1 (file : cifar_classification_v0.ipynb)

To start with I tried with a network of 3 fully connected layers. Using the features from the hidden/code layer , I tried to train 2 classifiers : (i) one with fully connected layers and (ii) 1D Convolution. Trained the network on the train data without any augmentation to just get an idea on how the network performs. The autoencoder network could reconstruct the test data with mean squared error (mse) of ~0.002.

MLP autoencoder : Test MSE = 0.0018

MLP Dense Layer classifier : Test accuracy = 50%, Train accuracy = 93 %

1D CNN classifier: Test accuracy = 44% , Train accuracy = 95%

The reconstructions and loss graph are shown in the notebook. The classifier network made from this quick trial is not able to generalize very well on the hidden dense layer features from autoencoder. The validation accuracy stuck to less than 50% , so I think doing augmentation or changing any hyperparameters for that network wont help improving the accuracy at a greater scale. So I then shifted to 2D-CNN based networks which are anyway widey used for 3D images.

Data was only scaled down by 255 – not standardized. The output of batch normalization layer after the hidden layer are extracted as autoencoder features after observing that the classifier model was not converging when trained directly on code/features from Hidden dense layer (i.e. Dense(768, activation='relu', name='features') in the model definition). Adding batch normalization helped to get about 95% training accuracy. With standardized and mean subtracted data, the MSE of autoencoder was slightly higher compared to the data just scaled by (1/255). For given number of epochs, the loss was higher. So I did not use standardization.

This was just a quick run, there is a large gap between training and validation accuracy so there is a large scope to improve both the autoencoder and classifier networks in terms of depth and architecture. So I moved to CNN based approach, given that the data is 3D images.

3. TRIAL-2 (file : cifar_classification_v1.ipynb)

This notebook has codes for independent training of autoencoder and then the classifier. I have started creating a network in simple VGG style Sequential model.

3.1 Autoencoder:

To start with, I used a 2-D CNN based autoencoder with Maxpooling layers for downsampling in encoder part and using Upsampling layers for decoding. For a network with convolutional block having Conv2d + Upsampling in the decoder, the training as well as validation MSE was stuck at 0.05 for more than 10 epochs. Although adding 2 convolutional blocks of 32 and those of 64 filters helped reducing the MSE to 0.008 for 30 epochs.

Replaced Upsampling2D layer with Conv2DTranspose layer. This helped the model to converge faster and with MSE on test data of about 0.0007. The reason can be that this layer also learns the method of interpolation instead of just upsampling while also increasing number of trainable parameters in the network.

The final decoder network was trained at batch size of 64 and 128 . With 64 batch size, the error on train as well as test data was about 0.0008. Batch size 128 resulted in test accuracy of about 0.0007 an showed earlier convergence. Graph and reconstructed images are shown in the notebook.

The autoencoder is trained on the image pixels scaled down by 255. It was observed that by subtracting mean of training data from train and test data, the reconstruction error (MSE) increased i.e. about 0.015. Similarly on using z-score i.e. standardization by mean subtraction and normalizing by standard deviation of training data, the validation MSE loss was was about 1.4. Used Adam optimizer with 0.001 learning rate for all trials.

Did not train it with noisy images or did not add noise layer in the network because the aim here is only to achieve reconstruction on Cifar-10 test data. And use the features for training a classifier.

3.2 Classifier:

Classifier network also has been made with a sequential CNN blocks. Since the layer from where the encoded features are extracted has size (8,8,64), I started with a network with 128 filters. The classifier was first trained features extracted from autoencoder on non-augmented data.

First trial was SGD with lr = .001 and 128 batch size. I observed that with higher batch size, using higher learning rate was fine. Tried with Adam 0.01 and 128 batchsize also. The network with 128 batchsize was struck for about 10 epochs at about 85% train and 72% validation accuracy.

So next I tried with 64 batch size and learning rate decay. With Adam (lr=0.001 and decay on pleateu) and 64 batch size, validation accuracy got stuck to about 75 % while training kept on increasing till 97% for about 10 epochs. Also the validation loss kept increasing which implies extreme overfitting.

Accuracy values of train and test, and the loss graph are plotted in the notebook (cifar_classification_v1.ipynb) . There is clear overfitting and the network is not able to generalize on the data. Cat accuracy is only 58%. The network is converging on train data but since the test accuracy is just 75%, performing more regularization and data-aumentation should help in improving the accuracy further.

But in this case, the features are calculated from autoencoder whose task is to reduce MSE between the input and output images. While the features have to be used by classifier whose task is to reduce error between predicted labels and actual labels. It is important that classifier gets the features that are discriminative enough. This fact is also in a way supported by TSNE graph of some data points as in following graph.

So joint training makes more sense intuitively as the encoder learns features that contribute to the reduction of cross entropy loss also.

4. TRIAL 4

4. 1 Joint training :

Notebooks:

cifar_classification_v2_no_augmentation.ipynb

cifar_classification-joint_v2_with_augmentation_Adam_89_86.ipynb

cifar_classification_v2_with_augmentation_SGD-85_82.ipynb

The accuracy on test data improved but the loss and evaluation metrics have been shown in the notebooks. Althrough for the considered network and loss function, the classification accuracy improved but the reconstruction error increased. The autoencoder and the classifier both with image as input perform better individually compared to joint training.

Without augmentation, the train accuracy was about 96% but the test accuracy remained 75%. The loss plot show the overfitting and lack of generalization in cifar_classification_v2_no_augmentation.ipynb.

Adam with class weighing gave better accuracy : Train – 89% and test 86%. Confusion matrix and class wise scores are printed in the notebook. In cifar_classification_v2_with_augmentation_SGD-85_82.ipynb, I used 2 Dense layers along with convolutional layers for classification and used SGD optimizer. The reconstruction error is higher in that network compared to the one with single Dense layer in cifar_classification-joint_v2_with_augmentation_Adam_89_86.ipynb

4.2 Data Augmentation:

The additional augmentation functions have been added in keras preprocessing file like Contrast, Histogram quilixatio, Sharpness etc. In affine_transformations.py file of keras_preprocessing. The file has been included and the methods have been written using PIL (pillow) library. These augmentations need to be included in image.py file in keras/preprocessing/image.py. And then should be used to get custom augmentations in the data.

5. OBSERVATIONS:

From the current set of trials, as of now the joint training of autoencoder or even the independent feature extraction is not more accurate than training a CNN classifier on images directly. But using

an autoencoder can be a good way to detect an anomaly in the input image even if the classifier classifies the data into one of the categories.

End-to-end training of autoencoder and classifier, batch size of 64 with initial 0.001 learning rate gave stable training and an improvement in training accuracy at lesser epochs compared to 32. Batch size 128 reached higher accuracy in lowest epochs and the accuracy of model improved at initial epochs itself especially with 0.01 learning rate, but then it was stuck at about 78% accuracy for a long time.

Weight tying in deep autoencoders have been used to gain accurate semantic segmentation of smaller objects too e.g. W-Net, U-Net. Somehow weight tieing in autoencoder while performing joint (end-to-end) training with classifier did not make any significant improvement in reconstruction error.

During joint training of the multi-output model, autoencoder loss is more compared to it was when autoencoder was trained individually. Same applies with the classifier loss when individually trained on images. A classifier needs to learn class specific features that generalize well for objects representing the considered classes. While autoencoder learns a mapping function , to reconstruct an image based on what is given in input and output in this case.

I cannot strongly conclude the hypothesis of extracting features out of autoencoder and using them for classification to be less accurate for every kind of application. More training and loss computation methods can be explored to see how the accuracy can be improved for a certain scenario.

6. NEXT STEPS:

6.1 Hyper parameter tuning:

- Use hyperparameter optimization (Bayesian / grid or random search) and see how much accuracy improvement is achieved with the considered model architecture.
- Use tuning methods to see if number of layers can be tuned using keras-tuner. https://github.com/keras-team/keras-tuner
- Need to add a custom routine for dynamic learning rate and batch size change while training

6.2 Network architecture:

- Can make the network deeper and check how much Resnet or inception layers help in accuracy improvement and if they also help in reducing training time. The block of NASNet architecture can also be tried.
- For a given model and data , different data augmentation techniques that contribute to better performance can be checked using following https://arxiv.org/pdf/1805.09501.pdf

6.3 Loss function:

- Check what happens if the reconstruction error is added with the classifier loss. Currently they are being trained individually but the loss of reconstruction can be added to tune the classifier

- Currently both the models are being trained for individual tasks with different cost functions. Calculating a cost function that combines both the tasks might help in calculating features that are discriminative enough for classification as well as image reconstruction.