ELEC 576: Assignment 2

Xiaoqian Chen

Nov 06, 2019

Contents

1 Visualizing a CNN with CIFAR10

We train a CNN on a modified version of the CIFAR10 dataset, we visualize the weights of the first convolutional layer, and we visualize the activations using the test images.

1.1 CIFAR10 Dataset

CIFAR10 is a dataset of natural images of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images of CIFAR10 are originally of size 32×32 and have three channels (RGB). We use a modified version of the CIFAR10 dataset whose images are of size 28×28 and have one channel (grayscale). Before training, we divide our images by 255 and use one-hot encoding for labels. The following are the variables that we use for training and testing:

```
In[1]: x_train.shape, y_train.shape, x_test.shape, y_test.shape
Out[1]: ((10000, 28, 28, 1), (10000, 1), (1000, 28, 28, 1), (1000, 1))
```

To verify that the dataset looks correct, let's plot the 10 images from the training set and display the class name below each image.

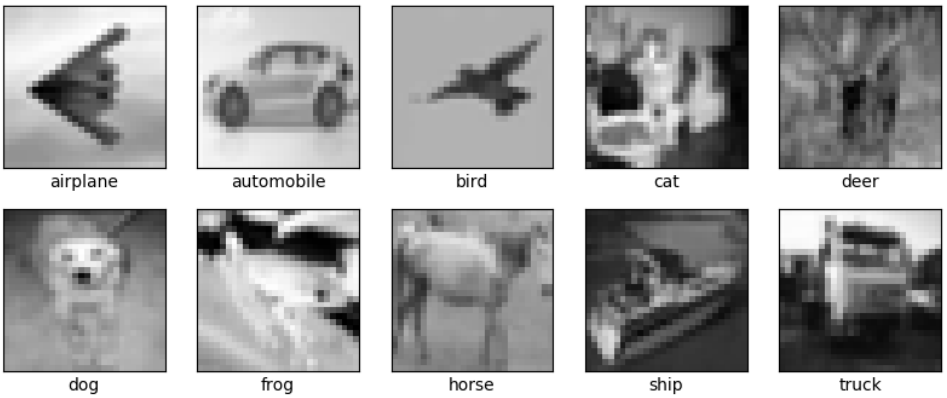


Figure 1: CIFAR10 samples

1.2 Train LeNet5 on CIFAR10

We implement a LeNet5 and train it on the modified version of the CIFAR10 dataset. The following is the configuration of LeNet5:

- convolutional layer with 5×5 kernel and 32 filter maps followed by ReLU
- max-pooling layer sub-sampling by 2
- convolutional layer with 5×5 kernel and 64 filter maps followed by ReLU
- max-pooling layer sub-sampling by 2
- fully-connected layer with input size $7 \cdot 7 \cdot 64$ and output size 1024
- fully-connected layer with input size 1024 and output size 10
- softmax layer

Let's display the architecture of our model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1049600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250
=================================================================
Total params: 1,111,946
Trainable params: 1,111,946
Non-trainable params: 0
_________________________________________________________________
Train on 10000 samples, validate on 1000 samples
```

During training, weights in the neural networks are updated so that the model performs better on the training data. For a while, improvements on the training set correlate positively with improvements on the test set. However, there comes a point where you begin to overfit on the training data and further "improvements" will result in lower generalization performance. This is known as overfitting. Early stopping is a technique used to terminate the training before overfitting occurs. In this case, we require that the accuracy should at least improve 0.0001 and 5 epochs with no improvement after which training will be stopped.

```
earlystop = callbacks.EarlyStopping( monitor='val_acc', min_delta=0.0001, patience=5)
```

We tried 3 training methods: SGD, RMSprop and Adam.
SGD optimizer: is Stochastic gradient descent and momentum optimizer.

$$\begin{aligned} v_t &= momentum * v_{t-1} - lr * dx \\ x_t &= x_{t-1} + v_t \end{aligned} \tag{1}$$

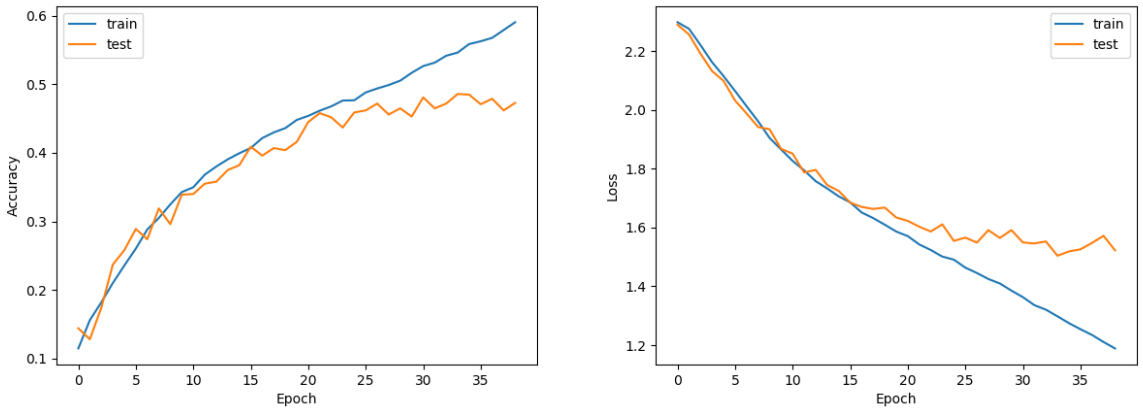
RMSprop optimizer: maintain a moving (discounted) average of the square of gradients; and divide gradient by the root of this average.

$$\begin{aligned} v_t &= \beta 1 * v_{t-1} + (1 - \beta 1) * dx^2 \\ mom_t &= momentum * mom_{t-1} + lr * dx / (\sqrt{v_t} + \epsilon) \\ x_t &= x_{t-1} - mom_t \end{aligned} \tag{2}$$

Adam optimizer: is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. It looks a bit like RMSprop with momentum, using the "smooth" version of the gradient dx instead of the raw gradient vector .

$$\begin{aligned} lr_t &= lr_0 * \sqrt{1 - \beta_2^t} / (1 - \beta_1^t) \\ m_t &= \beta 1 * m_{t-1} + (1 - \beta 1) * dx \\ v_t &= \beta 2 * v_{t-1} + (1 - \beta 2) * dx^2 \\ x_t &= x_{t-1} - lr_t * m_t / (\sqrt{v_t} + \epsilon) \end{aligned} \tag{3}$$

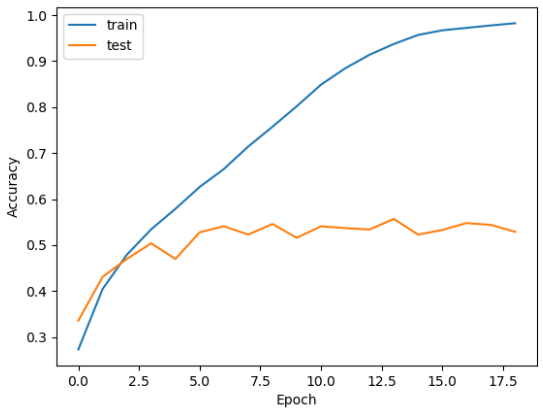
optimizer=SGD, test accuracy= 46.7%;



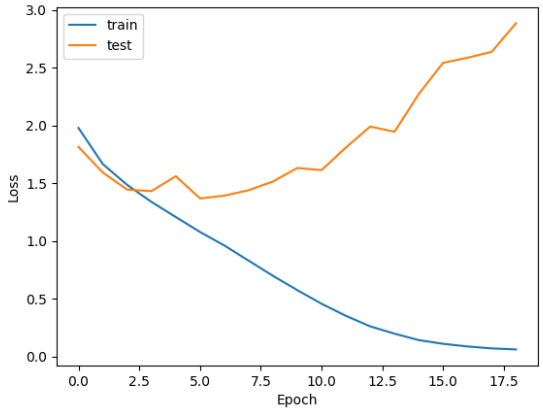
(a) Accurancy (b) Loss

Figure 2: LeNet5 on CIFAR10 with SGD optimizer

optimizer=RMSprop, test accuracy= 52.3%;



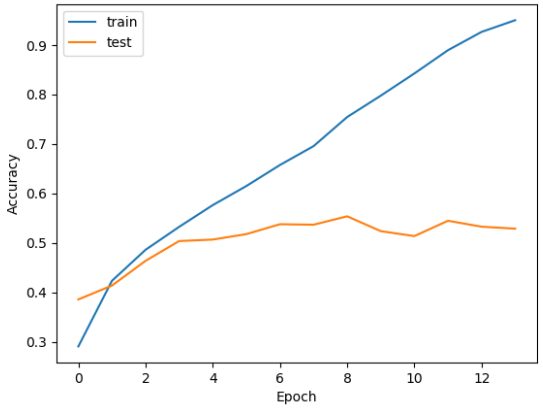
(a) Accurancy



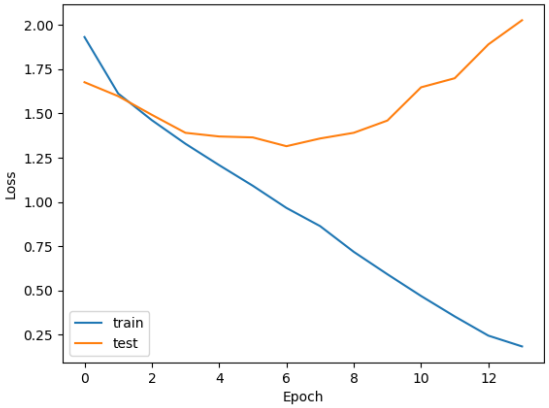
(b) Loss

Figure 3: LeNet5 on CIFAR10 with RMSprop optimizer

optimizer=Adam, test accuracy= 52.9%;



(a) Accurancy



(b) Loss

Figure 4: LeNet5 on CIFAR10 with Adam optimizer

I could not get LeNet5 to yield a test accuracy of at least 60% despite it often yielding a train accuracy of 100%. For efficiency, Adam finished train with the least epochs and $Adam \geq RMSprop \geq SGD$.

1.3 Visualize the Trained Network

We visualize the weights of the first convolutional layer and we show the statistics of the activations in the convolutional layers on test images. Note that the weights of the first convolutional layer look like Gabor filters (edge detectors).

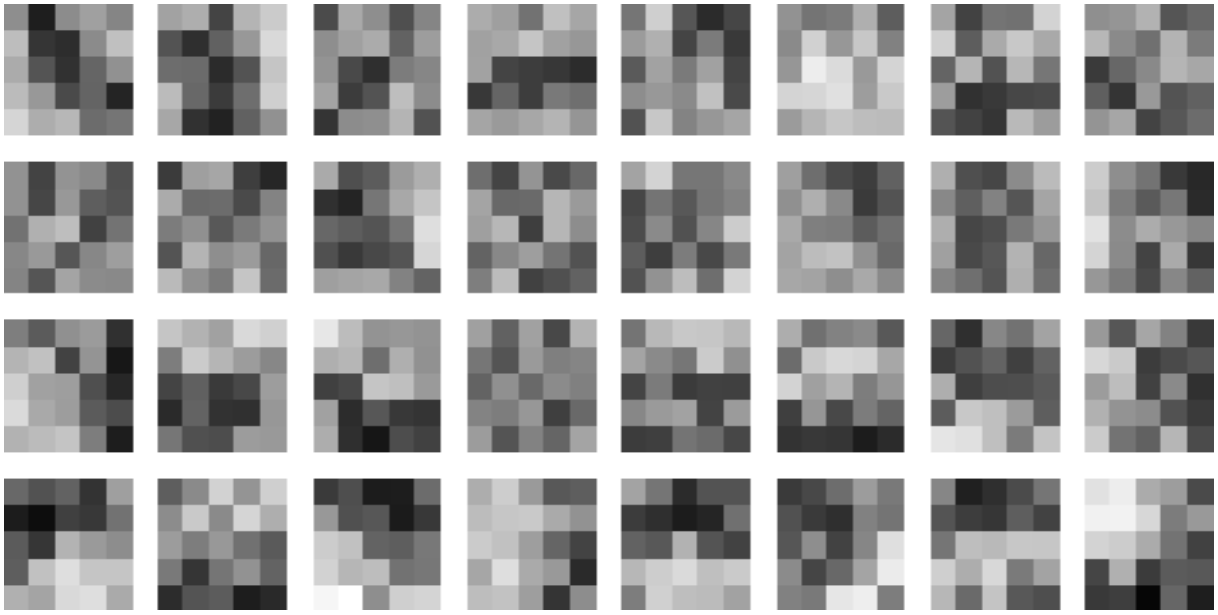


Figure 5: Weights of First Convolutional Layer Filters

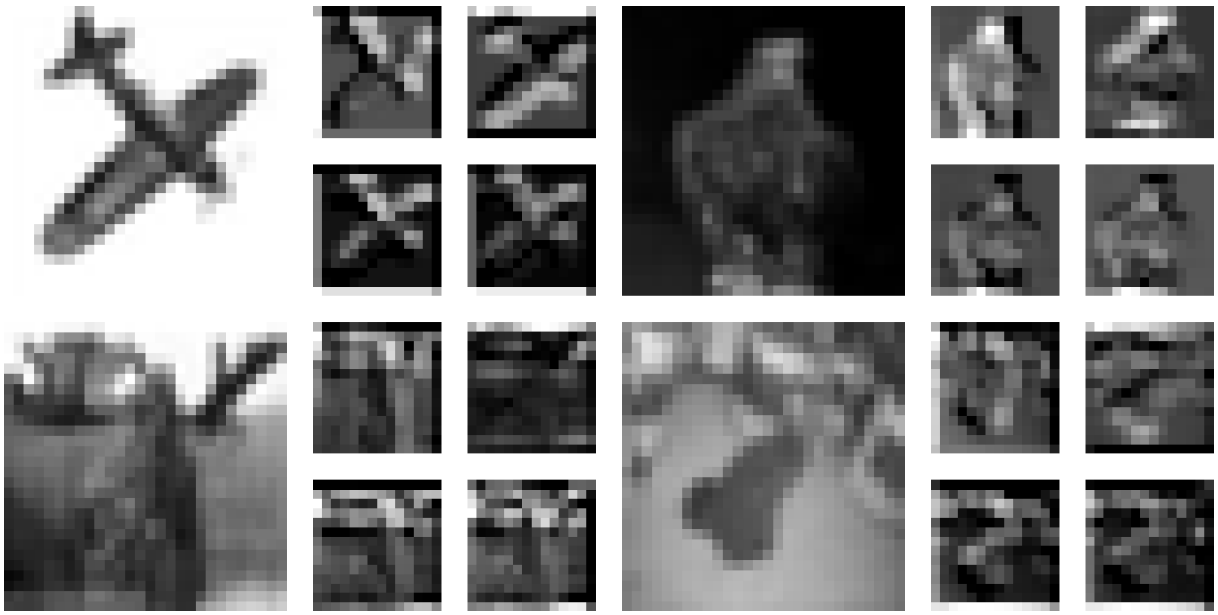


Figure 6: Statistics of Activations in Convolutional Layers on Test Images

2 Visualizing and Understanding Convolutional Networks

In the paper "Visualizing and Understanding Convolutional Networks" by Matthew D. Zeiler and Rob Fergus, the authors propose a framework for visualizing the hidden layers of a deep convolutional neural network, which later helps them infer the roles of said layers in the task of object recognition. The method is based on the framework proposed in the paper "Deconvolutional Networks" by Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus, which is basically a pseudo-inverse of a convolutional neural network. The authors later used the insight they gained from said framework to design an architecture for the task of object recognition that yielded the best performance on the 2012 ImageNet dataset.

3 Build and Train an RNN on MNIST

An RNN is well-suited for tasks on time series or sequential data. However, in this task we use an RNN for object recognition and compare it to the CNN of Assignment 1.

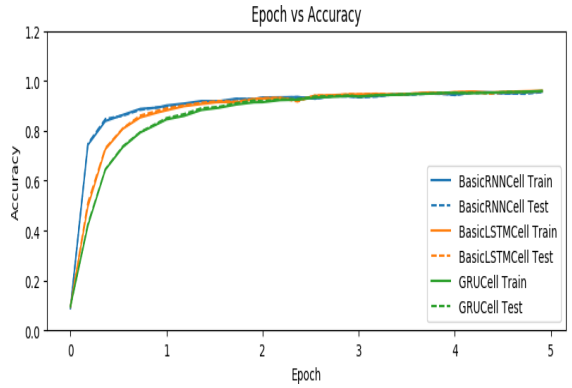
3.1 Setup an RNN

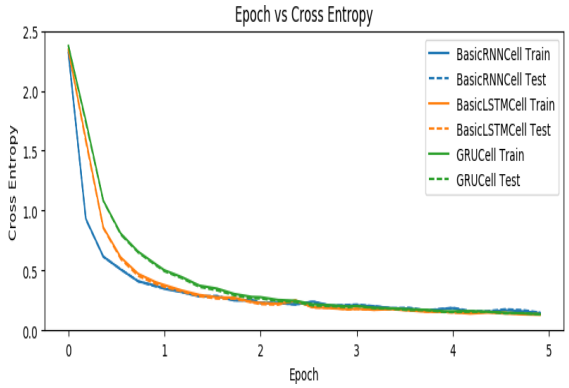
The input to a CNN is an image. However, the input to our RNN will be 28-pixel chunks of an image, i.e., each row of the image will be sent to the RNN at each iteration. The following is the result of training the RNN over 50 epochs with the Adam Optimizer, a learning rate of 10^{-4}, and a batch size of 100:

```
In[2]: evaluation_train, evaluation_test
Out[2]: (0.9993, 0.9886)
```

3.2 How about using an LSTM or GRU

Note that an RNN, an LSTM, and a GRU yield comparable results for this task.





3.3 Compare against the CNN

The RNN and the CNN of Assignment 1 yield comparable results. Nevertheless, training the CNN of Assignment 1 was much faster.