# Deep Learning Practice: Homework1

张驰　　Academy for Advanced Interdisciplinary Studies

Mar 21, 2019

## 1 Q1: How a smaller dataset affects test accuracy?

- a. Feed all the data from train32x32.mat.
- b. Feed 30000 images from train32x32.mat.
- c. Feed 10000 images from train32x32.mat.

### 1.1 Results analysis

In this experiment, there was no cloud server，so I set "use_extra_data" flag to False in common.py. However, in practice, it did not actually take 60 steps to converge, so I changed some of the Config so that the program can run faster and save more time. Therefore, I remodified the configuration conditions as follows:

```
nr_epoch = 40
boundaries = [xx * 15,xx * 25]
```

Note that the code to be modified should be in the load() function, not in the instance_generator() function. Because the latter function will process the samples of each batch and has a great impact on training process, while the former function operates on the dataset without affecting the training process. After the above processing, I selected the results with the best test accuracy as baseline.

表 1: Best Results Of Q1

| Name | Method | test−accuracy |
|---|---|---|
| q1.a | Feed all the data | 94.6% |
| q1.b | Feed 30000 images | 92.7% |
| q1.c | Feed 10000 images | 89.6% |

It can be concluded that the size of training set has a great influence on the test accuracy.

# 2 Q2: How the distribution of data affects test accuracy?

- a.reduce the amount of images labelled with'8' '9' and '0' to 500 and get Dataset A
- b. reduce the amount of images labelled with '6', '7', '8','9' and '0' to 1000 and get Dataset B
- c. reduce the amount of images labelled with '1','2','3','4'and '5' to 6000 and get Dataset C

## 2.1 Results analysis

In this experiment, We will use the different data sets in Q1 to verify the impact of distribution on test accuracy. Here is the key code:

```
def reduce_size(self, data, labels, List, num):
  for j in range(data.shape[3]):
    if self.index < self.instances:
      if labels[j,:][0] == 10:
        labels[j,:][0] = 0
      if labels[j,:][0] in List:
        for i in range(len(List)):
          if labels[j,:][0]  == List[i] and self.count[i] < num:
            self.count[i] += 1
            data[:, :, :, self.index] = data[:, :, :, j]
            labels[self.index, :] = labels[j, :][0]
            self.index += 1
            break
      else:
        data[:, :, :, self.index] = data[:, :, :, j]
        labels[self.index, :] = labels[j, :]
        self.index += 1
    else:
      print(data[:, :, :, :self.instances].shape, labels[:self.instances, :].shape)
      return data[:, :, :, :self.instances], labels[:self.instances, :]
```

After the above processing, I selected the results with the best test accuracy.

表 2: Best Results Of Q2

| Name | Method | test−accuracy |
|---|---|---|
| q1.a | Feed all the data | 94.6% |
| q1.b | Feed 30000 images | 92.7% |
| q1.c | Feed 10000 images | 89.6% |
| q2.a | datasetA with all the data | 92.0% |
| q2.a.30000 | datasetA with 30000 images | 91.5% |
| q2.a.10000 | datasetA with 10000 images | 87.0% |
| q2.b | datasetB with all the data | 92.6% |
| q2.b.30000 | datasetB with 30000 images | 92.2% |
| q2.b.10000 | datasetB with 10000 images | 89.7% |
| q2.c | datasetC with all the data | 93.8% |
| q2.c.30000 | datasetC with 30000 images | 93.1% |
| q2.c.10000 | datasetC with 10000 imagess | 90.3% |

As is shown in the experiment, we can see that the distribution of datasets has a slight effect on test accuracy. By comparing the results of q2.a, q2.b, and q2.c with q1.a, we can find that the closer the data distribution is to the distribution of the source dataset, the higher the test accuracy will be. However, the distribution limitation of dataset C actually improves the test accuracy, indicating that changing the distribution of dataset does not necessarily reduce the test accuracy.

# 3 Q3: How augmentation helps when training dataset is small?

- a.color inversion: sets a pixel value from v to 255-v.
- b.affine transformation: affine transformation is usually adopted for expressing rotations, translations and scale operations.
- c.adding salt and pepper noise: sets a pixel value to 255(make it white-ish as ”salt”) or to 0( make it black-ish as ”pepper”)

## 3.1 Results analysis for color inversion

In this experiment, We will implement some of the augmentation techniques. Here is the key code for color inversion:

```
def color_inversion(self, data, label):
  for i in range(self.instances):
    if label[i, :][0] == 10:
      label[i, :][0] == 0
```

```
        rows, cols, dims, n = data.shape
        for k in range(dims):
            a = np.random.randint(0, 2)
            if a:
                data[:, :, k, i] = 255 - data[:, :, k, i]
    return data, label
```

I invert pixels from each channel individually with a probability with the code. The results are as follows:

表 3: Best Results Of Color Inversion

| Name | Method | test−accuracy |
|---|---|---|
| q1.a | Feed all the data | 94.6% |
| q1.b | Feed 30000 images | 92.7% |
| q1.c | Feed 10000 images | 89.6% |
| q3.a | all the data | 94.4% |
| q3.a.30000 | 30000 images | 92.6% |
| q3.a.10000 | 10000 images | 88.4% |

Color inversion is a good technique for augmentation. It basically does not change the accuracy of the baseline. We can expand the sample size, increase the dataset and improve the test accuracy through this technology

## 3.2 Results analysis for affine transformation

In this experiment, we will test the result of affine transformation. Here is the key code for it:

```
def affine_transformation(self, img):
    rows, cols, dims = img.shape
    choice = np.random.choice(['scale', 'rotate', 'shift', 'affine'])
    if choice == 'scale':
        # 放缩
        scale = np.random.choice([0.8, 0.9, 1.0, 1.1, 1.2])
        img = cv2.resize(img, dsize=(int(rows * scale), int(cols * scale)), interpolation=cv2.
                                                    INTER_LINEAR)
    elif choice == 'rotate':
        # 旋转
        RotateMatrix = cv2.getRotationMatrix2D(center=(cols / 2, rows / 2), angle=90, scale=1.2)
        img = cv2.warpAffine(img, RotateMatrix, (rows * 2, cols * 2))
    elif choice == 'shift':
        # 平移
        TranslationMatrix = np.float32([[1, 0, 5], [0, 1, 2]])
        img = cv2.warpAffine(img, TranslationMatrix, (rows, cols))
    elif choice == 'affine':
    # 仿射变换
        pts1 = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])
```

```
    pts2 = np.float32([[cols * 0.2, rows * 0.1], [cols * 0.9, rows * 0.2], [cols * 0.1, rows * 0
                                                .9]])
    M_affine = cv2.getAffineTransform(pts1, pts2)
    img = cv2.warpAffine(img, M_affine, (cols, rows))
  return img
```

I randomly select an affine transformation to test the results. The results are as follows:

表 4: Best Results Of Affine Transformation

| Name | Method | test−accuracy |
|---|---|---|
| q1.a | Feed all the data | 94.6% |
| q1.b | Feed 30000 images | 92.7% |
| q1.c | Feed 10000 images | 89.6% |
| q3.b | all the dat | 93.4% |
| q3.b.30000 | 30000 image | 94.8% |
| q3.b.10000 | 10000 images | 90.8% |

Affine transformation is the best augmentation technique in the experiment. It is even more precise than baseline. So, it can be concluded that after the affine transformation of the image, the image is greatly changed, but the learning of the neural network still works. So it can be used as a technical method to expand the dataset.

### 3.3 Results analysis for salt and pepper noise

In this experiment, we will test the result of salt and pepper noise. Here is the key code for it:

```
  def salt_pepper_noise(self, data, labels):
  rows, cols, dims, n = data.shape
  for j in range(data.shape[3]):
    if j < self.instances:
      if labels[j, :][0] == 10:
        labels[j, :][0] = 0
      for i in range(40):
        x = np.random.randint(0, rows)
        y = np.random.randint(0, cols)
        a = np.random.randint(0, 2)
        data[x, y, :,j] = 255 if a else 0
    else:
      return data[:, :, :, :j], labels[:j, :]
```

I randomly generate 40 noise points that are half pepper and half salt. The results are as follows:

表 5: Best Results Of Salt Pepper Noise

| Name | Method | test−accuracy |
|---|---|---|
| q1.a | Feed all the data | 94.6% |
| q1.b | Feed 30000 images | 92.7% |
| q1.c | Feed 10000 images | 89.6% |
| q3.c | all the data | 94.3% |
| q3.c.30000 | 30000 images | 92.2% |
| q3.c.10000 | 10000 images | 89.5% |

The result of salt and pepper noise is close to the baseline. So, I think it is also a good way to expand the dataset for augmentation.

### 3.4 Results analysis

In this experiment, we can see that affine transformation is very effective augmentation technique. We can combine these methods to train small dataset.

## 4 Q4: Whether Mixup(a data-agnostic augmentation technique) helps

- Apply mixup technique when you feed all/30000/10000 images from train32x32.mat in the classification tasks and see whether the test accuracy increases compared with that from Q1.)

Mixup is a special technology that works like this:

$$\begin{aligned} x &= x_1 * weight + x_2 * (1 - weight) \\ y &= y_1 * weight + y_2 * (1 - weight) \end{aligned} \tag{1}$$

where x1 and x2 are images from the dataset, y1 and y2 are the corresponding labels. The weight is taken from the beta distribution. Here is the key code:

```
def mix_up(self, data, labels):
  for j in range(data.shape[3]):
    if j < self.instances:
      if labels[j, :][0] == 10:
        labels[j, :][0] = 0
      weight = np.random.beta(1.2, 0.2)
      rand = np.random.randint(0, self.instances)
      img2 = data[:, :, :, rand]
      data[:, :, :, j] = data[:, :, :, j] * weight + img2 * (1 - weight)
    else:
      return data[:, :, :, :j], labels[:j, :]
```

I randomly select another image from the dataset, and calculate the mixup value. In the version I submitted earlier, the labels became float due to weight. So I changed the type of the label returned in the corresponding function from int32 to float, which led to a very bad result. However, I changed my method by referring to the website https://www.inference.vc/mixup-data-dependent-data-augmentation/. Here are the final results:

表 6: Best Results Of affine transformation

| Name | Methody | test−accuracy |
|---|---|---|
| q1.a | Feed all the data | 94.6% |
| q1.b | Feed 30000 images | 92.7% |
| q1.c | Feed 10000 images | 89.6% |
| q4.all | all the data | 92.2% |
| q4.30000 | 30000 images | 90.5% |
| q4.10000 | 10000 images | 86.5% |

From the above results, we can see that the mixup method is also effective. However, the running result is getting better and better as the amount of data increases.

# 5 Tips

In the version I submitted earlier, I made a very serious mistake that I changed the learning rate from 15-40 to 3-5 and epoch from 60 to 20. The change for epoch did not affect too much, however, the change of learning rate made me premature convergence to a local optimal solution, and that's why my test accuray was lower than baseline. Be careful in the future.