实验任务

在本实验中，我们将指导大家利用 TensorFlow 完成猫与狗的图片的任务；同时作为练习，要在每个步骤时，完成 cifar10 的分类任务。

实验预备：

1. 阅读材料 http://yann.lecun.com/exdb/mnist/，了解 MNIST 数据集
2. 阅读材料 https://www.cs.toronto.edu/~kriz/cifar.html，了解 CIFAR 数据集
3. 为方便数据集的管理，请 TensorFlow 数据集管理模块 tensorflow_datasets：
   ```
   C:\Users\MSUser>pip install tensorflow_datasets
   ```
4. 请在 D 盘下或其它目录下新建文件夹 lab-3

准备训练数据

子任务：

由 MNIST 数据生成特定格式的文件，称为 TensorFlow Record（TFR）文件，方便训练时使用。TFR 文件是二进制的结构化文件，具有较高的处理效率，可以避免当训练数据过多时，不能加载进内存的困境，又可避免利用一般磁盘文件时的低效率。

步骤：

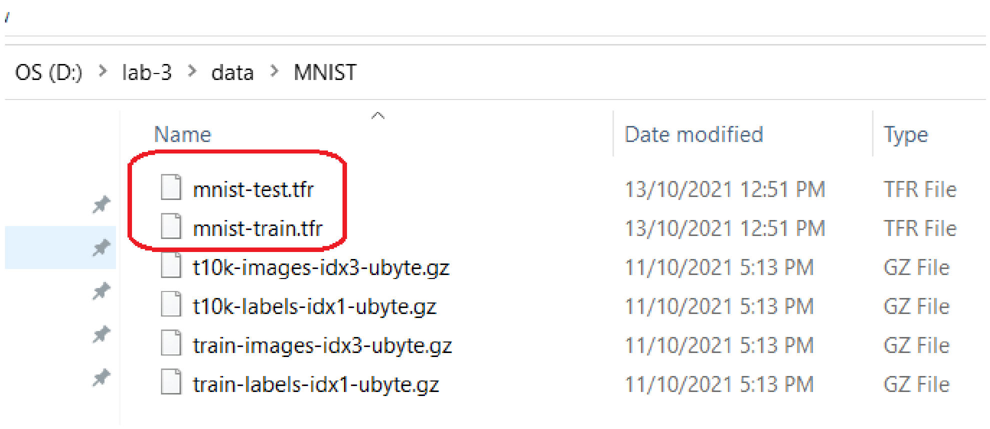
1. 在 lab-3 文件夹下新建文件 ic_prepare.py
2. 输入以下内容：
   ```
   import os
   import numpy as np
   import tensorflow as tf

   from tensorflow.examples.tutorials.mnist import input_data

   def float_feature(value):
      return tf.train.Feature(float_list = tf.train.FloatList(value = value))

   def bytes_feature(value):
      return tf.train.Feature(bytes_list = tf.train.BytesList(value = [value]))

   def int64_feature(value):
      return tf.train.Feature(int64_list = tf.train.Int64List(value = [value]))

   save_path = 'data/mnist'
   ```

```
print("Loading MNIST dataset")
data = input_data.read_data_sets("data/MNIST/", one_hot=False)

print('beginning prepare MNIST tfrecords for training')
writer = tf.io.TFRecordWriter(os.path.join(save_path, 'mnist-train.tfr'))

num_train_records = 0

for image, label in zip(data.train.images, data.train.labels):
    feature = \
    {
        'image': float_feature(image),
        'label': int64_feature(label)
    }
    example = tf.train.Example(features = tf.train.Features(feature = feature))
    writer.write(example.SerializeToString())

    num_train_records = num_train_records + 1

writer.close()
print('end of tfrecords preparation for training')

print('beginning prepare MNIST tfrecords for testing')
writer = tf.io.TFRecordWriter(os.path.join(save_path, 'mnist-test.tfr'))

num_test_records = 0

for image, label in zip(data.test.images, data.test.labels):
    feature = {
        'image': float_feature(image),
        'label': int64_feature(label)
    }
    example = tf.train.Example(features = tf.train.Features(feature = feature))
    writer.write(example.SerializeToString())

    num_test_records = num_test_records + 1

writer.close()

print('end of tfrecords preparation for testing')

print('#tfrecords for training: {}'.format(num_train_records))
print('#tfrecords for testing: {}'.format(num_test_records))
```

为方便起见，文件 ic_prepare.py 附后： ic_prepare.py

3. 查看输出结果，如果程序无误，则会生成如下图所示文件：

```
D:\lab-3>python ic_prepare.py
```



OS (D:) › lab-3 › data › MNIST

Name	Date modified	Type
mnist-test.tfr	13/10/2021 12:51 PM	TFR File
mnist-train.tfr	13/10/2021 12:51 PM	TFR File
t10k-images-idx3-ubyte.gz	11/10/2021 5:13 PM	GZ File
t10k-labels-idx1-ubyte.gz	11/10/2021 5:13 PM	GZ File
train-images-idx3-ubyte.gz	11/10/2021 5:13 PM	GZ File
train-labels-idx1-ubyte.gz	11/10/2021 5:13 PM	GZ File

实作：

生成 CIFAR-10 样本的 TensorFlow Record 文件，示例代码已经提供，但请读者先自行实验。

观察待分类的图片

子任务：

我们将首先观察待分类的 mnist 图片，以便对我们的任务有一个认识。

步骤：

1. 在 lab-3 文件夹下新建文件 ic_data.py
2. 输入以下内容：

```
import os
import numpy as np
import tensorflow as tf
import sonnet as snt

class Input(snt.AbstractModule):
  def __init__(self, batch_size, image_dims, num_epochs = -1, name = 'input'):
    super(Input, self).__init__(name = name)
```

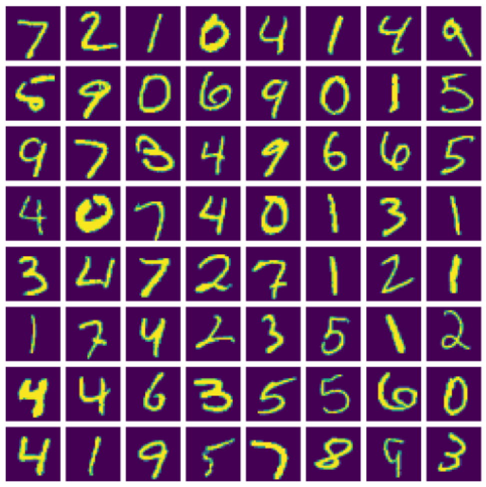
```
        self._batch_size = batch_size
        self._image_dims = image_dims
        self._num_epochs = num_epochs

    def _parse_function(self, example):
        dims = np.prod(self._image_dims)

        features = {
            "image": tf.FixedLenFeature([dims], dtype = tf.float32),
            "label": tf.FixedLenFeature([], dtype = tf.int64)
        }

        example_parsed = tf.parse_single_example(serialized = example, features = 
features)
        value = tf.reshape(example_parsed['image'], self._image_dims)

        label = example_parsed['label']

        return value, label

    def _build(self, filename):
        assert os.path.isfile(filename), "invalid file name: {}".format(filename)

        dataset = tf.data.TFRecordDataset([filename])
        dataset = dataset.map(self._parse_function)

        dataset = dataset.batch(self._batch_size)
        dataset = dataset.repeat(self._num_epochs)

        it = dataset.make_one_shot_iterator()
        images, labels = it.get_next()

        return images, labels


def draw_image(images, rows, cols, tensor_name = "images"):
    import tfmpl

    @tfmpl.figure_tensor
    def draw(images):
        num_figs = len(images)
        fig = tfmpl.create_figures(1, figsize= (12.8, 12.8))[0]

        # pdb.set_trace()
        for i in range(rows):
            for j in range(cols):
                seq = i * cols + j + 1
                if seq > num_figs:
                    fig.tight_layout()
                    return fig

                if num_figs == 1:
                    ax = fig.add_subplot(1, 1, 1)
                else:
                    ax = fig.add_subplot(rows, cols, seq)

                ax.axis('off')
                ax.imshow(images[seq-1, ...])

        fig.tight_layout()
        return fig

    image_tensor = draw(images)
    image_summary = tf.summary.image(tensor_name, image_tensor)
    sess = tf.get_default_session()
    assert sess != None, "Invalid session"
    image_str = sess.run(image_summary)

    return image_str


if __name__ == '__main__':
    num_images = 64
    image_width = 28
    image_height = 28

    input_ = Input(num_images, [image_height, image_width, 1])
    images, labels = input_('data/mnist/mnist-train.tfr')

    if not os.path.exists("output-data"):
        os.makedirs("output-data")

    writer = tf.summary.FileWriter("output-data", tf.get_default_graph())

    with tf.Session() as sess:
        sess.run([tf.global_variables_initializer(), tf.local_variables_initializer()])

        image_str = draw_image(images, 8, 8)
```

```
writer.add_summary(image_str, global_step = 0)
```

为方便起见，文件 ic_data.py 附后： ic_data.py

3. 查看输出结果，如果程序无误，则输出如下图所示：

```
D:\lab-3>python ic_data.py
tensorboard --logdir=D:\\lab-3\\output-data --port=8008
http://localhost:8008/#images
```



实作：

观察 CIFAR-10 的样本，示例代码，已经提供，但请读者先自行实验。

小结：

可以通过考察真实数据的分布特征，决定所选取模型的复杂度。

建立模型

子任务:

我们用如下配置来建立网络模型:

层 ID	层类型	Filter 数目	Filter 大小	激活函数	Padding 方式
1	Conv2D	32	5	ReLU	SAME
2	Pooling	-	2		-
3	Conv2D	64	5	ReLU	SAME
4	Pooling	-	2		-
5	FC	-	256	ReLU	-
6	FC	-	#class	Softmax	-

步骤:

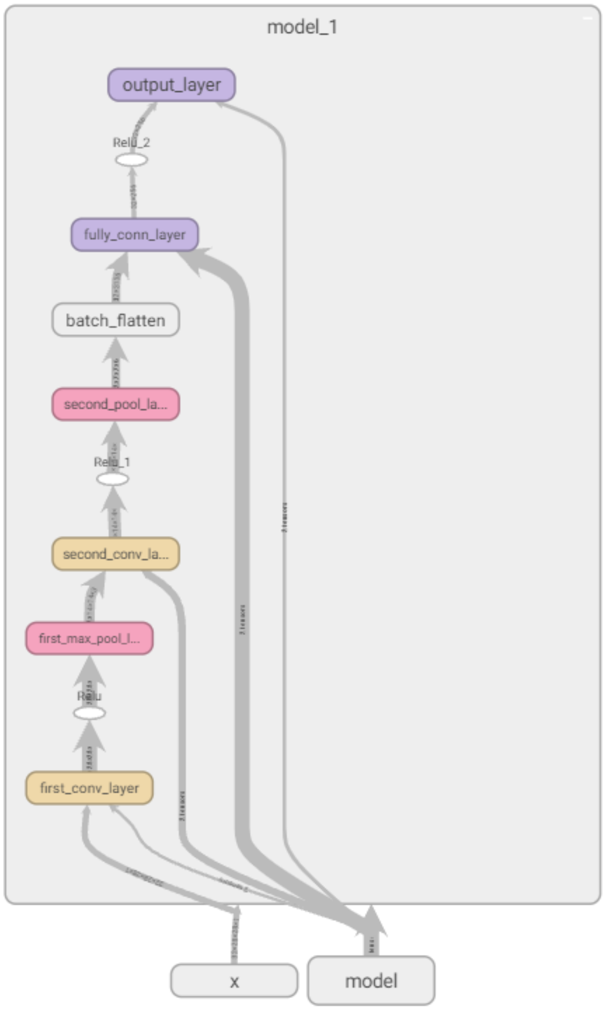
1. 在 lab-3 文件夹下新建文件 ic_model.py
2. 输入以下内容:

```
import os
import numpy as np
import tensorflow as tf
import sonnet as snt

class Pooling(snt.AbstractModule):
  def __init__(self, pool = None, k = 2, padding = 'SAME', name = "pooling"):
    super(Pooling, self).__init__(name = name)
    self._pool = pool
    self._k = k
    self._padding = padding

  def _build(self, x):
    if self._pool == 'max':
      return tf.nn.max_pool2d(x, self._k, self._k, self._padding)
    elif pool == 'avg':
      return tf.nn.avg_pool2d(x, self._k, self._k, self._padding)
    else:
      return lambda x: x

class Model(snt.AbstractModule):
  def __init__(self, num_classes, filter_size = 5, name = "model"):
    super(Model, self).__init__(name = name)

    with self._enter_variable_scope():
      self._conv1 = snt.Conv2D(32, filter_size, name = "first_conv_layer")
      self._pool1 = Pooling('max', name = "first_max_pool_layer")
```

```
        self._conv2 = snt.Conv2D(64, filter_size, name = "second_conv_layer")
        self._pool2 = Pooling('max', name = "second_pool_layer")

        self._lin = snt.Linear(256, name = "fully_conn_layer")
        self._output = snt.Linear(num_classes, name = "output_layer")

    def _build(self, x):
        y = tf.nn.relu(self._conv1(x))
        y = self._pool1(y)

        y = tf.nn.relu(self._conv2(y))
        y = self._pool2(y)

        y = snt.BatchFlatten()(y)

        y = tf.nn.relu(self._lin(y))

        return self._output(y)

def test():
    x = tf.random_normal([32, 28, 28, 1], name="x")

    model = Model(10)
    y = model(x)

    if not os.path.exists("output-model"):
        os.makedirs("output-model")

    writer = tf.summary.FileWriter("output-model")

    with tf.Session() as sess:
        writer.add_graph(sess.graph)

        sess.run([tf.global_variables_initializer(),
                tf.local_variables_initializer()])

        sess.run(y)

        writer.close()

if __name__ == "__main__":
    test()
```

为方便起见，文件 ic_model.py 附后：ic_model.py

3. 查看输出结果，如果程序无误，则输出如下图所示：
   D:\lab-3>python ic_model.py
   tensorboard --logdir=D:\\lab-3\\output-model --port=8008
   http://localhost:8008/#graphs&run=.



实作：

请以如下配置构建网络，通过 TensorBoard 观察所建立模型是否正确。

层 ID	层类型	Filter 数目	Filter 大小	激活函数	Padding 方式
1	Conv2D	32	3	ReLU	SAME

2	Pooling	-	2		-
3	Conv2D	64	3	ReLU	SAME
4	Pooling	-	2		-
5	Conv2D	128	3	ReLU	SAME
6	Pooling	-	2		-
7	FC	-	256	ReLU	-
8	FC	-	#class	Softmax	-

训练模型

子任务:

我们用 mini-batch 方法，训练新建模型。

步骤:

1. 在 lab-3 文件夹下新建文件 ic_train.py
2. 输入以下内容

```
import os
import numpy as np
import tensorflow as tf

from ic_data import Input
from ic_model import Model

batch_size = 32

image_width = 28
image_height = 28
num_classes = 10

# 执行自动微分算法（或反向传播算法）的优化器参数
learning_rate = 0.001
lr_decay_steps = 100
lr_decay_factor = 0.9

# 训练时迭代次数
iterations = 1000

if not os.path.exists("output-train"):
   os.makedirs("output-train")
```

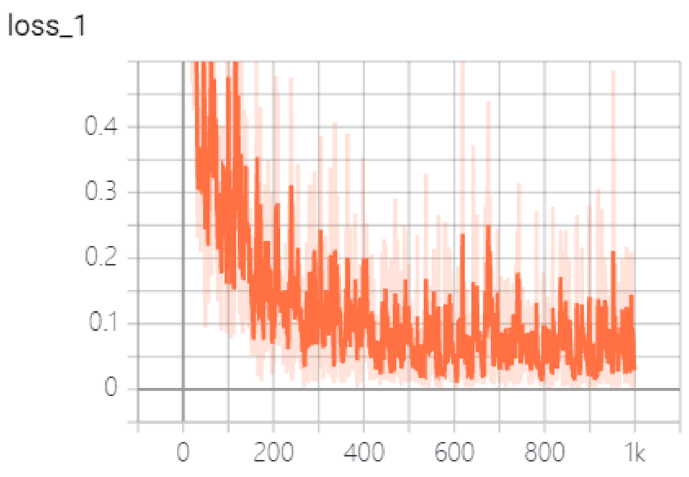
```
# 训练好的模型的保存路径
checkpoint_path = os.path.join("output-train", "model.ckpt")

# 以数据集方式加载样本，方便进行 mini-batch 训练
input_ = Input(batch_size, [image_height, image_width, 1])
images, labels = input_("data/mnist/mnist-train.tfr")

# 由输入得到模型的输出
net = Model(num_classes)
logits = net(images)

# 计算损失函数
labels = tf.one_hot(labels, num_classes, axis = -1)

with tf.control_dependencies([tf.assert_equal(tf.rank(labels), tf.rank(logits))]):
   loss = tf.nn.softmax_cross_entropy_with_logits_v2(labels = labels, logits = logits)

loss = tf.reduce_mean(loss, name = 'loss')
loss_summary = tf.summary.scalar('loss', loss)

# 设置学习率
global_step = tf.train.get_or_create_global_step()
lr = tf.train.exponential_decay(learning_rate, global_step,
   lr_decay_steps, lr_decay_factor, staircase = True)

lr_summary = tf.summary.scalar('lr', lr)

# 创建优化器
opt = tf.train.AdamOptimizer(lr)

# 进行优化
train_op = opt.minimize(loss, global_step = global_step, var_list =
tf.trainable_variables())

# 合并所有 summary 信息
all_summaries = tf.summary.merge_all()

writer = tf.summary.FileWriter('output-train', tf.get_default_graph())

# 保存训练好的模型
saver = tf.train.Saver(tf.trainable_variables())

# 创建 Session，进行训练
```

```
with tf.Session() as sess:
    sess.run([tf.global_variables_initializer(), tf.local_variables_initializer()])

    for i in range(iterations):
        loss_val, _, summ_str = sess.run([loss, train_op, all_summaries])
        print("{}-th iteration with loss {}".format(i, loss_val))
        writer.add_summary(summ_str, global_step = i)


    # 训练完成，保存模型
    print('Saving model.')
    saver.save(sess, checkpoint_path)
    print('Training complete')

    writer.close()
```

为方便起见，文件 ic_train.py 附后：ic_train.py

3. 查看输出结果，如果程序无误，则输出如下图所示：

```
D:\lab-3>python ic_train.py
tensorboard --logdir=D:\\lab-3\\output-train --port=8008
http://localhost:8008/#scalars
```



实作：

依据上节中所建网络，训练非线性函数对应的网络模型

测试模型

子任务:

测试我们用 mini-batch 方法所训练模型的表现。

步骤:

1. 在 lab-3 文件夹下新建文件 ic_test.py
2. 输入以下内容

```
import os
import numpy as np
import tensorflow as tf
import tfmpl

from ic_data import Input
from ic_model import Model

from tensorflow.contrib.tensorboard.plugins import projector

image_width = 28
image_height = 28
num_classes = 10

LOG_DIR = "output-test"

if not os.path.exists(LOG_DIR):
   os.makedirs(LOG_DIR)

if not os.path.exists("output-train"):
   raise ValueError("Non-existing output-train folder")

# 训练好的模型的保存路径
checkpoint_dir = "output-train"

graph_test = tf.Graph()
with graph_test.as_default():
   # 加载测试数据
   input_ = Input(1, [image_height, image_width, 1], 1)
   image, label = input_("data/mnist/mnist-test.tfr")

   # 由输入得到模型的输出
   net = Model(num_classes)
   logit = net(image)
```

```
    logit = tf.squeeze(logit, axis = 0)
    id = tf.argmax(logit, axis = -1)

    restorer = tf.train.Saver()

embeddings = []
metadata = os.path.join(LOG_DIR, 'metadata.tsv')
metadata_file = open(metadata, 'w')
count = 0

with tf.Session(graph = graph_test) as sess:
    sess.run([tf.global_variables_initializer(), tf.local_variables_initializer()])

    ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
    if ckpt and ckpt.model_checkpoint_path:
        # Restores from checkpoint.
        restorer.restore(sess, ckpt.model_checkpoint_path)

        print('Successfully loaded model from %s.' % ckpt.model_checkpoint_path)

    else:
        print('No checkpoint file found')
        exit

    while True:
        try:
            logit_val, id_val = sess.run([logit, id])
            embeddings.append(logit_val)
            metadata_file.write('%d\n' % id_val)
            # sess.run(metric_update)
            count = count + 1
        except tf.errors.OutOfRangeError:
            break

    metadata_file.flush()
    metadata_file.close()


graph_visualize = tf.Graph()
with graph_visualize.as_default():
    embedding_var = tf.Variable(np.zeros([count, num_classes]), dtype = tf.float32,
name="embedding_var")
```

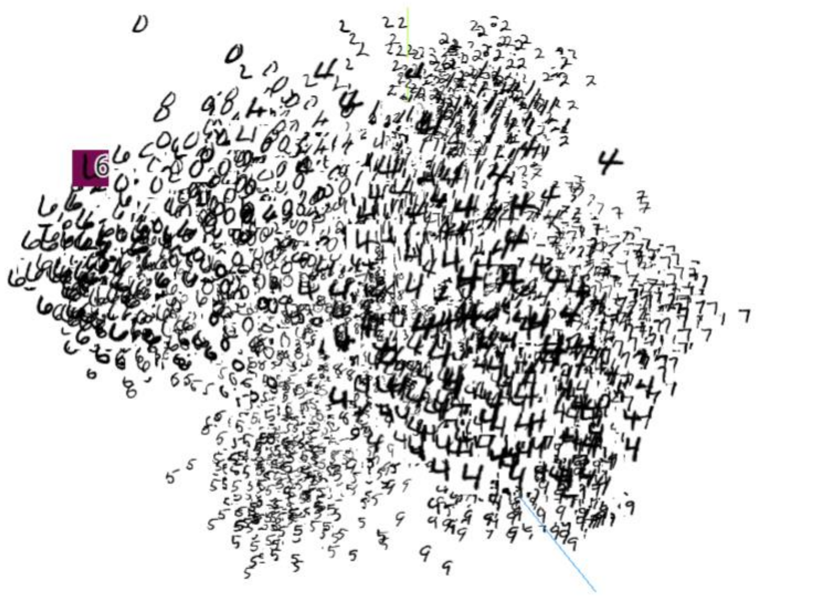
```
    embedding_op = embedding_var.assign(tf.convert_to_tensor(np.array(embeddings),
dtype = tf.float32))

    saver = tf.train.Saver([embedding_var])

    writer = tf.summary.FileWriter(LOG_DIR, graph_visualize)
    config = projector.ProjectorConfig()

    embedding = config.embeddings.add()
    embedding.tensor_name = embedding_var.name
    embedding.sprite.image_path = 'mnist_10k_sprite.png' # Path relative to writer's log
directory
    embedding.sprite.single_image_dim.extend([28, 28])
    embedding.metadata_path = 'metadata.tsv'           # Path relative to writer's log
directory

    projector.visualize_embeddings(writer, config)

with tf.Session(graph = graph_visualize) as sess:
    sess.run([tf.global_variables_initializer(), tf.local_variables_initializer()])
    sess.run(embedding_op)

    saver.save(sess, os.path.join(LOG_DIR, 'model.ckpt'), global_step = 1)
```

为方便起见，文件 ic_test.py 附后： ic_test.py

3. 查看输出结果，如果程序无误，则输出如下图所示：

```
D:\lab-3>python ic_test.py
tensorboard --logdir=D:\\lab-3\\output-test --port=8008
```

http://localhost:8008/#projector&run=.



实作:

依据上节训练结果，测试 CIFAR-10 的模型的表现。