实验任务

在本实验中，我们将指导大家利用 TensorFlow 完成 MNIST 手写数字识别的任务；同时作为练习，要在每个步骤时，完成 cifar10 的分类任务，在此基础上，完成与本文档结构类似的实验报告。

零、实验预备：

1. 确保按照实作零中步骤，已经完成三个库 TensorFlow，Sonnet， tfmpl 的安装；
2. 阅读材料 http://yann.lecun.com/exdb/mnist/，了解 MNIST 数据集；
3. 阅读材料 https://www.cs.toronto.edu/~kriz/cifar.html，了解 CIFAR 数据集；
4. 请在 D 盘下或其它目录下新建文件夹 assignment-2，文件夹路径之间尽量不要有中文字符；切换至新建的文件夹；同时打开命令行，切换至新建文件夹。

一、准备训练数据

子任务：

尽管 TensorFlow 提供了对象，可以由 MNIST 数据直接构建数据输入管线，供 TensorFlow 框架在训练模型时使用，但一种更为高效的使用方式是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. 在当前目录下新建文件 mnist_prepare.py
2. 输入以下内容：

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

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 = r'data\mnist'
if not os.path.exists(save_path):
    os.makedirs(save_path)

print("Loading MNIST dataset")
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

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

for i, (image, label) in enumerate(zip(x_train, y_train)):
    print(f"Processing the {i}-th sample ...")
    feature = \
    {
        'image': float_feature(image.flatten()),
        'label': int64_feature(tf.squeeze(label))
    }
    example = tf.train.Example(features = tf.train.Features(feature=feature))
    writer.write(example.SerializeToString())

num_train_records = i + 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'))

for i, (image, label) in enumerate(zip(x_test, y_test)):
    print(f"Processing the {i}-th sample ...")
    feature = {
        'image': float_feature(image.flatten()),
        'label': int64_feature(tf.squeeze(label))
    }
    example = tf.train.Example(features = tf.train.Features(feature=feature))
    writer.write(example.SerializeToString())

num_test_records = i + 1
```

当处理不同的数据集时，需要更改为对应的保存路径，以防准备好的数据集被覆盖。

当更换数据集时，要用不同数据集的 load_data 函数，具体的数据集在实作部分会讲。

当处理不同的数据集时，更改为更为合适的名称，尽量不要沿袭原来数据集的名称。

```
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 mnist_prepare.py 附后：

mnist_prepare.py

3. 在命令行执行文件：python mnist_prepare.py

4. 查看输出结果，如果程序无误，则会生成如下图所示的文件：

实作：

生成 CIFAR-10 样本的 TensorFlow Record 文件，在完成的过程中可以参考以下步骤：

1. 查阅什么是 CIFAR-10 数据集；
2. 阅读下面网站，了解 TensorFlow 内置 Keras 模块的支持：
   https://keras.io/api/datasets/cifar10/
3. 参照上面示例程序，编写程序，如 cifar10_prepare.py，将 CIFAR-10 数据集准备为 TF Record 形式的数据集；
4. 将数据准备的代码，生成的文件的截图附在新的实验报告中。

二、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. 在当前文件夹下新建文件 mnist_data.py
2. 输入以下内容:

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

class Dataset(snt.Module):
    def __init__(self, batch_size, image_dims, num_epochs=-1, name='dataset'):
        super(Dataset, 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.io.FixedLenFeature([dims], dtype=tf.float32),
            "label": tf.io.FixedLenFeature([], dtype=tf.int64)
        }

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

        return value, label

    def __call__(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)
```

```
        return dataset

@tfmpl.figure_tensor
def draw_image(images, rows, cols):
    num_figs = len(images)
    fig = tfmpl.create_figure(figsize= (12.8, 12.8))

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

    fig.tight_layout()
    return fig


if __name__ == '__main__':
    num_images = 64
    image_width = 28
    image_height = 28
    image_channels = 1

    ds = Dataset(num_images, [image_height, image_width, image_channels])
    images, labels = next(iter(ds(r'data\mnist\mnist-train.tfr')))

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

当用不同的数据集时，针对数据集的图片的大小，这些值要做适当的调整。具体要看数据集的说明。

针对不同数据集准备的 TF Records 格式文件的存放位置作相应调整。

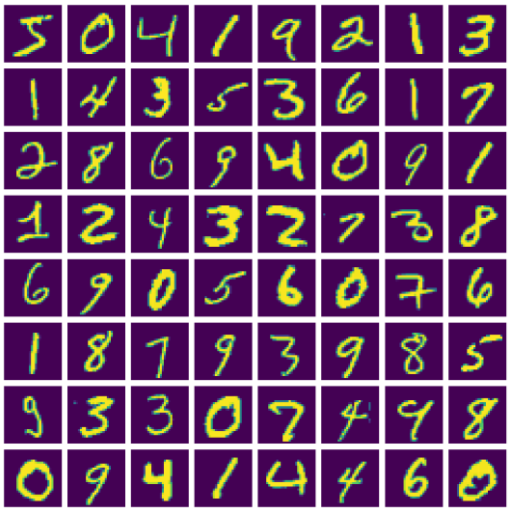
```
summary_writer = tf.summary.create_file_writer(log_dir)
with summary_writer.as_default():
    image_tensor = draw_image(images, 8, 8)
    image_summary = tf.summary.image("images", image_tensor, step=0)
summary_writer.close()
```

为方便起见，文件 mnist_data.py 附后：

mnist_data.py

3. 在命令行执行文件：python mnist_data.py

4. 查看输出结果，如果程序无误，则通过如下命令在命令行启动 TensorBoard 后，在浏览器键入地址 http://127.0.0.1:6006/，会在浏览器中看到如下图形：
   tensorboard --logdir=output-data



实作：

观察 CIFAR-10 的样本，根据示例代码的提示，创建相关脚本文件，如 cifar10_data.py，展示数据集内相关图片，将相关代码与 TensorBoard 展现在浏览器中的图片附在新的实验报告中。

三、建立模型

子任务：

我们用如下配置来建立卷积神经网络模型：

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	-	-

步骤:

1. 在当前文件夹下新建文件 mnist_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.Module):
    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 __call__(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.Module):
    def __init__(self, num_classes, filter_size=5, name="model"):
        super(Model, self).__init__(name=name)
        self._num_classes = num_classes
        self._filter_size = filter_size

    @snt.once
    def _initialize(self):
```

```
        self._conv1 = snt.Conv2D(32, self._filter_size, name="first_conv_layer")
        self._pool1 = Pooling('max', name="first_max_pool_layer")

        self._conv2 = snt.Conv2D(64, self._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(self._num_classes, name="output_layer")

    def __call__(self, x):
        self._initialize()

        y = tf.nn.relu(self._conv1(x))
        y = self._pool1(y)

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


        y = snt.Flatten()(y)

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

        return self._output(y)

def test():
    log_dir = "output-model"
    if not os.path.exists(log_dir):
        os.makedirs(log_dir)

    x = tf.random.normal([32, 28, 28, 1])
    model = tf.function(Model(10))

    summary_writer = tf.summary.create_file_writer(log_dir)
    tf.summary.trace_on(graph=True, profiler=False)
    y = model(x)
    with summary_writer.as_default():
        tf.summary.trace_export(name="model_trace", step=0, profiler_outdir=log_dir)
    tf.summary.trace_off()
```

如果要增加新的卷积层与池化层，请在这里定义，如：

self._conv3 = snt.Conv2D(128, self._filter_size, name="third_conv_layer")

如果定义了新的层，请在这里建构在一起，如 y = tf.nn.relu(self._conv3(y))等等

注意根据批量大小、数据集中图片的高度、宽度、通道数目，对四个些数值作相应的调整。

根据数据集中类别或分类的数目，对此数值作相应的调整。

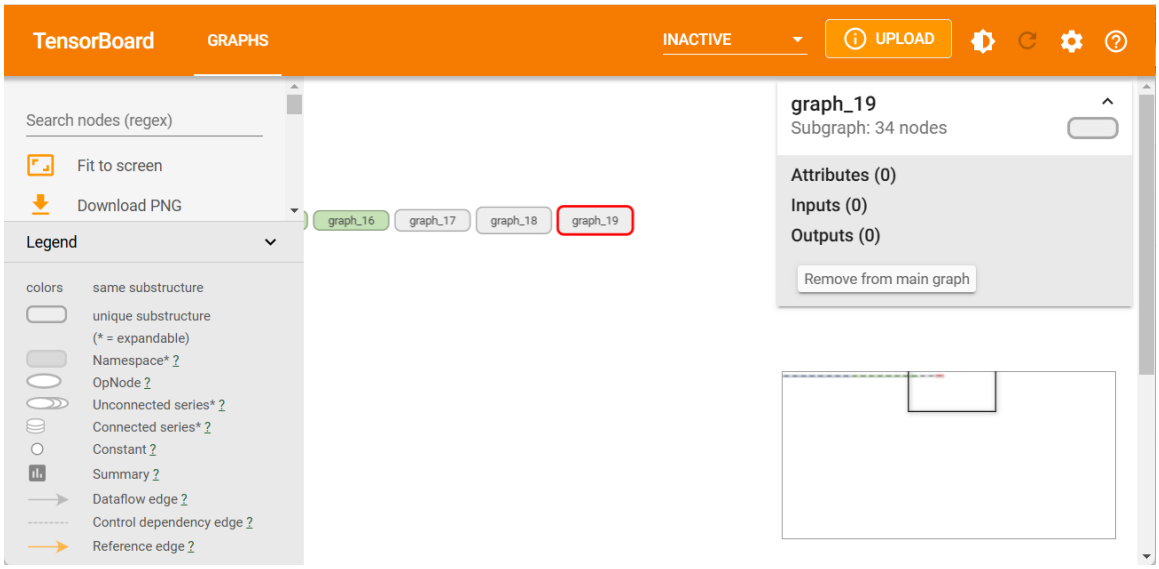
```
if __name__ == "__main__":
    test()
```

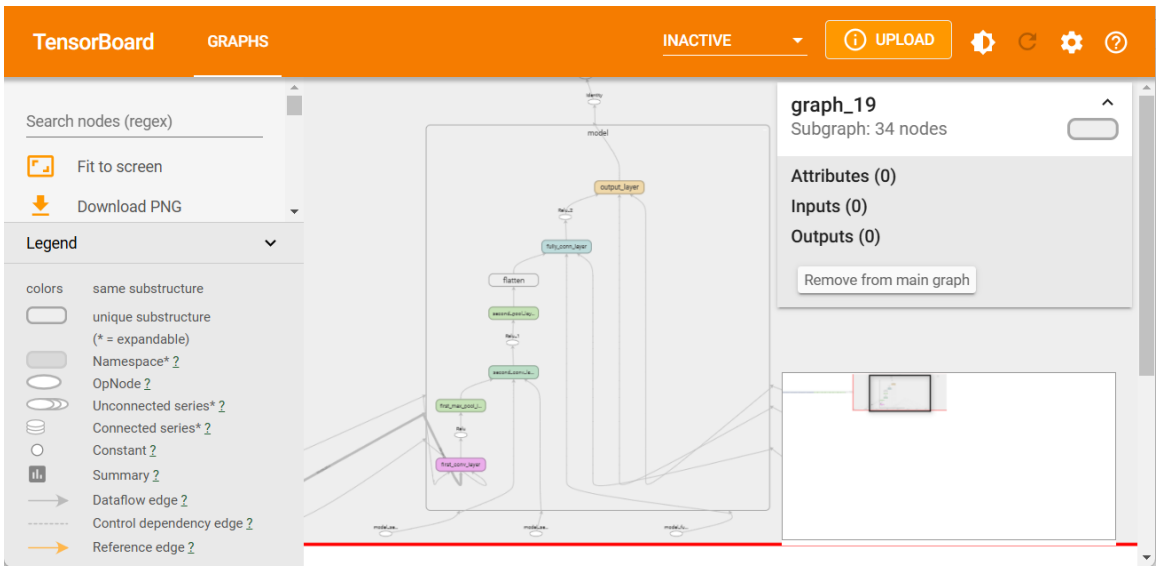
为方便起见，文件 mnist_model.py 附后：

mnist_model.py

3. 在命令行执行程序：python mnist_model.py
4. 若程序执行无误，则可以通过如下命令在命令行中启动 TensorBoard，在浏览器中键入 http://127.0.0.1:6006/，查看浏览器中的模型结构展示（注意把最后一个节点打开）：

   tensorboard --logdir=output-model





实作：

请参考示例代码，参考如下配置构建网络，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	-	-

请按以上步骤，将构建模型的代码与 TensorBoard 呈现的模型结构输出附在新的实验报告中。

四、训练模型

子任务：

根据基于监督学习利用神经网络模型解决分类任务的理论，利用 mini-batch 方法，结合准备的 TF Records 格式的数据，训练构建的模型。

步骤：

1. 在当前文件夹下新建文件 mnist_train.py
2. 输入以下内容

```
import os
import numpy as np
import tensorflow as tf
import sonnet as snt
from mnist_data import Dataset
from mnist_model import Model

batch_size = 32
image_height = 28
image_width = 28
image_channels = 1
num_classes = 10
```

如果用 cifar-10，则图片是 32x32x3 的，这里也要作对应更改。如果分类的类别不是 10 类的话，对类别数目也要进行相应更改。

```
# 建立模型与优化器
model = Model(num_classes)
learning_rate = 0.01
optimizer = snt.optimizers.Adam(learning_rate)

# 准备存储运行时信息的文件夹与对象
log_dir = "output-train"
if not os.path.exists(log_dir):
    os.makedirs(log_dir)

summary_writer = tf.summary.create_file_writer(log_dir)

# 定义单步训练
def train(images, labels, step):
    # 记录前向传播信息
    with tf.GradientTape() as tape:
        logits = model(images)
        labels = tf.one_hot(labels, depth=num_classes)
        with tf.control_dependencies([tf.assert_equal(tf.shape(logits), tf.shape(labels))]):
            loss = tf.math.reduce_mean(
                tf.nn.softmax_cross_entropy_with_logits(labels, logits))

    # 计算梯度，并更新权重
    variables = model.trainable_variables
    gradients = tape.gradient(loss, variables)
    optimizer.apply(gradients, variables)

    with summary_writer.as_default():
        tf.summary.scalar('loss', loss, step=step)

    return loss


# 验证模型的超参
def validate(dataset_iter, step):
    images, labels = next(dataset_iter)
    logits = model(images)
    logits = tf.math.argmax(logits, -1)
    with tf.control_dependencies([tf.assert_equal(tf.shape(logits), tf.shape(labels))]):
```

如果观察到训练不收敛，可能学习率设置过大（或过小），请调整学习率

```
        prediction = tf.equal(labels, logits)
        accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))

    with summary_writer.as_default():
        tf.summary.scalar('accuracy', accuracy, step=step)

    return accuracy

# 保存模型
def save_model(module):
    @tf.function(input_signature=[tf.TensorSpec([None, image_height, image_width,
image_channels])])
    def inference(x):
        return module(x)

    save_path = r"saved_model"
    if not os.path.exists(save_path):
        os.makedirs(save_path)

    to_save = snt.Module()
    to_save.inference = inference
    to_save.all_variables = list(module.variables)
    tf.saved_model.save(to_save, save_path)


def main():
    # 加载准备好的数据
    train_data_path = r"data\mnist\mnist-train.tfr"
    val_data_path = r"data\mnist\mnist-test.tfr"

    train_dataset = Dataset(batch_size,
        [image_height, image_width, image_channels], 1)(train_data_path)
    val_dataset = Dataset(batch_size,
        [image_height, image_width, image_channels])(val_data_path)

    @tf.function
    def loop():
        num_epochs = 2
        step = np.int64(0)
        val_dataset_iter = iter(val_dataset)
```

如果用其它数据集，请其更改为对应的的准备的 TFR 文件所在的路径。

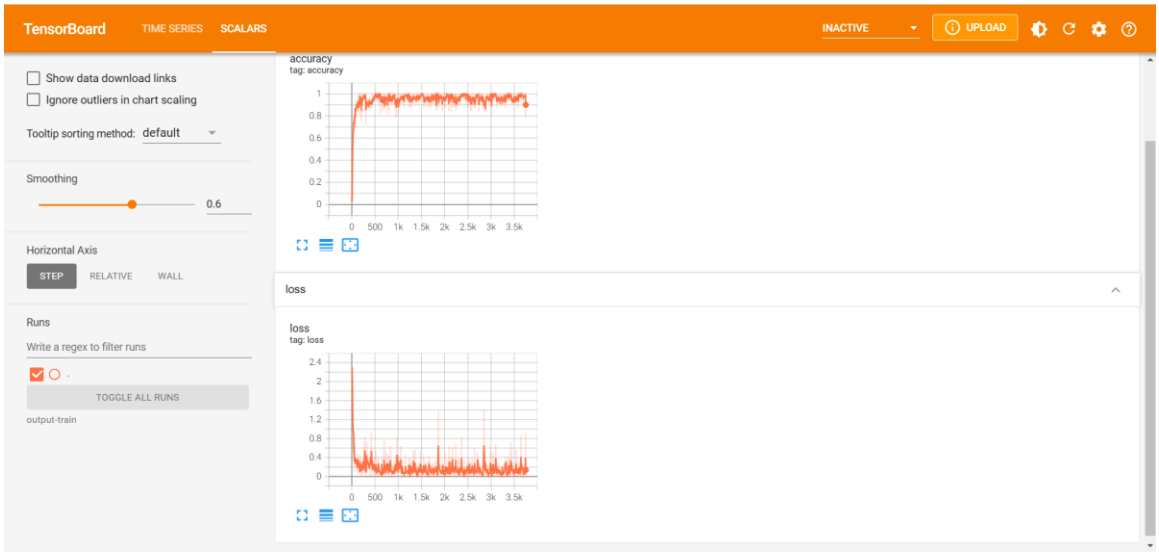
可以根据自己所用的计算机性能调整该值，2 到 10 之间均可。但较大值会用较长时间训练。

```
        for _ in range(num_epochs):
            for features, labels in train_dataset:
                step = step + 1
                loss = train(features, labels, step, )
                accu = validate(val_dataset_iter, step)
                tf.print("iteration:", step, " loss - ", loss, " accuracy - ", accu)

    # 训练模型
    loop()

    # 保存模型
    save_model(model)

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

为方便起见，文件 mnist_train.py 附后：

mnist_train.py

3. 在命令行执行文件：python mnist_train.py
4. 如果程序无误，在命令行启动 TensorBoard 查看训练过程：
   tensorboard --logdir=output-train
   启动 TensorBoard 之后，在浏览器键入 http://localhost:6006/#scalars，则原则上应该可以看到输出如下图所示：



实作:

依据上节中所建网络，训练识别 CIFAR-10 图片所用的网络模型。请将各步所做的工作，包括训练过程的代码，与反映训练过程的指标的图像，如上面所示的 TensorBoard 中所展示的图案，均附在实验报告里。

五、测试模型

子任务:

测试所训练模型的在测试集样本上的表现。可以根据自己的情况选择测试集中的全部样本或部分样本。

步骤:

1. 在当前文件夹下新建文件 mnist_test.py
2. 输入以下内容:

```
import os
import numpy as np
import tensorflow as tf
from mnist_data import Dataset
from tensorboard.plugins import projector

image_width = 28
image_height = 28
image_channels = 1
num_classes = 10

log_dir = "output-test"
if not os.path.exists(log_dir):
    os.makedirs(log_dir)

# 模型加载函数
def load_model(save_path):
    assert os.path.exists(save_path), "模型路径不存在"

    loaded = tf.saved_model.load(save_path)
    assert len(loaded.all_variables) > 0, "加载模型失败"

    return loaded
```

如果用 cifar-10，则图片是 32x32x3 的，这里需要更

```
# 加载测试数据与模型
ds = Dataset(1, [image_height, image_width, image_channels], 1)
model = load_model("saved_model")

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

accuracy = 0
count = 1000
for i, (image, label) in enumerate(ds(r"data\mnist\mnist-test.tfr")):
    logit = model.inference(image)
    logit = tf.squeeze(logit, axis=0)
    embeddings.append(logit)
    num = tf.argmax(logit, axis=-1)
    metadata_file.write('%d\n' % label[0])
    if num == label:
        accuracy = accuracy + 1
    if (count == 1):
        break
    else:
        count = count - 1

metadata_file.close()

accuracy = accuracy / (i + 1)
print("The accuracy of inference on test set is {}".format(accuracy))

embedding_var = tf.Variable(embeddings, dtype=tf.float32)
checkpoint = tf.train.Checkpoint(embedding=embedding_var)
checkpoint.save(os.path.join(log_dir, "embedding.ckpt"))

config = projector.ProjectorConfig()
embedding = config.embeddings.add()
embedding.tensor_name = "embedding/.ATTRIBUTES/VARIABLE_VALUE" # embedding_var.name
embedding.sprite.image_path = 'mnist_10k_sprite.png' # Path relative to the log directory
embedding.sprite.single_image_dim.extend([image_height, image_width])
```

如果需要不同数目的测试样本，可以进行调整。

如果训练 CIFAR-10，请更改为 CIFAR-10 的 TFR 文件所在的正确的路径

更改为所需要的 sprite 文件的路径，并注意将该文件置于输出文件夹，如 output-test 之下。

```
embedding.metadata_path = 'metadata.tsv'          # Path relative to the log directory

projector.visualize_embeddings(log_dir, config)
```

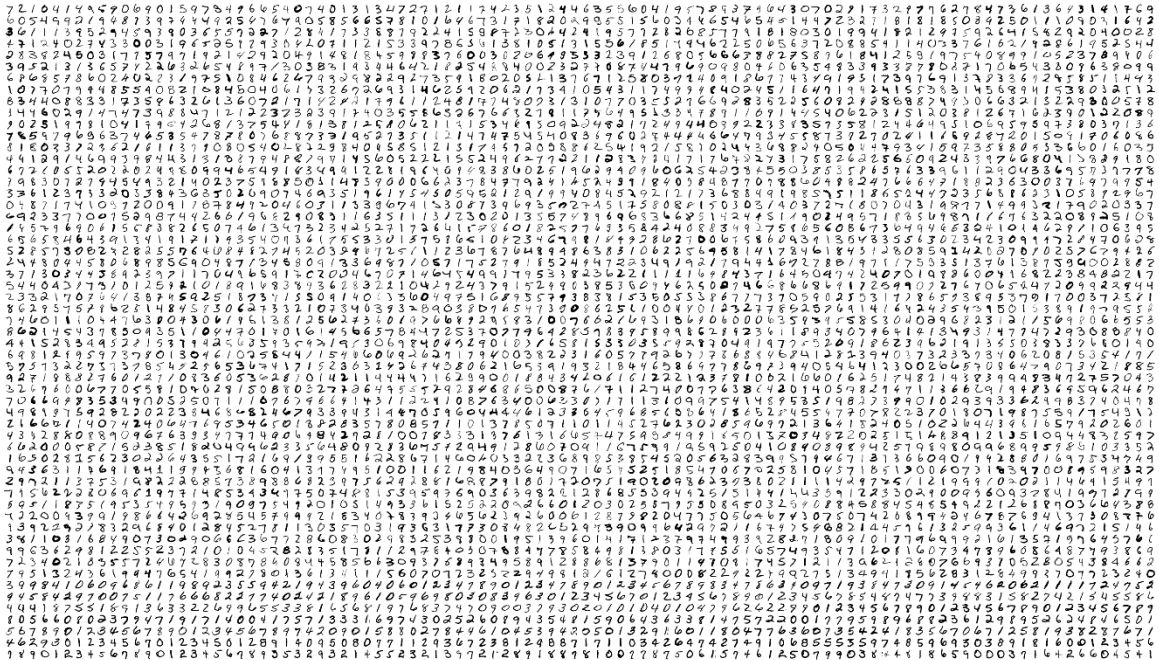
为方便起见，文件 ic_test.py 附后：

mnist_test.py

3. 在命令行执行文件：python mnist_test.py，如果无误，则能看到类似如下输出：

```
The accuracy of inference on test set is 0.936
```

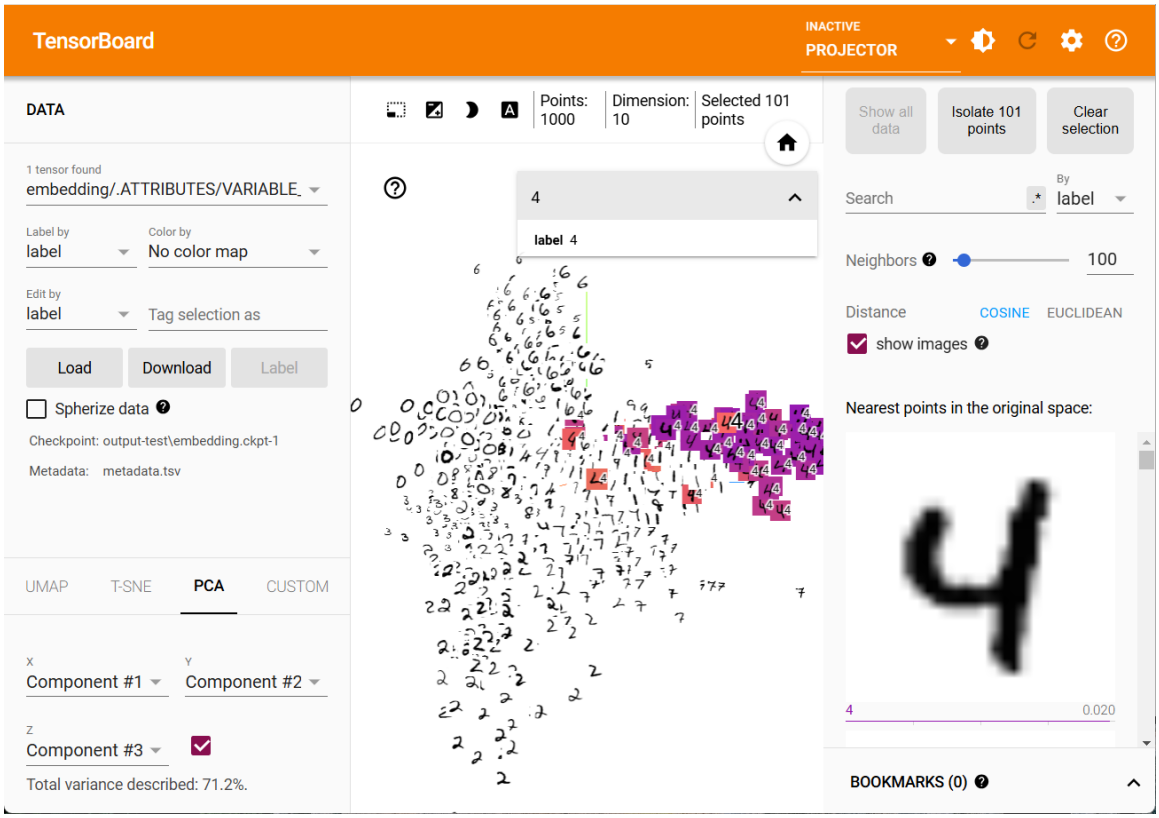
4. 将如下类似如下的 sprite 图片放入目录 output-test 中，注意图片的名称必须和代码中 embedding.sprite.image_path = 'mnist_10k_sprite.png'的值一致：



5. 通过如下命令在命令行启动 TensorBoard：

   tensorboard --logdir=output-test

   然后在浏览器中键入 http://localhost:6006/#projector，查看输出结果，如果程序无误，则输出如下图所示：



小结：

从输出可以看出，在两层卷积层与仅经过 2 个 epoch 训练，模型在 1000 个测试样本上的准确率为 93.6%；从 TensorBoard 的 PCM 图中可以看到，不同类别的数字图片被基本分到了不同的流形上，模型基本达到了要求。

实作：

仿照上面步骤，验证所设计的神经神经网络经训练之后，在 CIFAR-10 的测试集上的表现。将各个步骤的实验结果体现在新的实验报告中。

注意，要将给的类似如下形式的图片放入 output-test 文件夹中，如提供的 cifar10_sprite.png：