CSE4019 - Image Processing

Project Review 2 -
Person Re-identification with CUHK03 Dataset

SANJIT C K S
18BCE0715
SLOT - C2

Overview

Image re-identification with multiple cameras has been a major area of interest in the past 5 years. A network of cameras are situated at different angles and more or less monitor the same geographical area. In such a case, it is of great functional interest to identify/ label the same object/person in from all different cameras.

The best way to do object detection with images is via a Convolutional Neural Networks. Convolutional Neural Networks work take the approach of machine learning to learn the best convolutions that highlight the correct features to improve image identification.

Objects, in this case people have defining features that can be highlighted with the correct mask/filter/kernel. By using convolutional neural networks these can be learned. Convolution neural networks' efficiency are highly dependent on architecture.

General Approach

This has game changing applications in security and surveillance. The process of re-identification, in a multi CCTV camera surveillance network can be explained as follows,

1. A Person walks into the area of coverage of single camera (that is a part of a network of cameras).
2. The images of the person (footage is broken down into images) and are processed for feature extraction and object detection - which is usually done with Convolution Neural Networks.
3. The person leave the area of coverage of the first CCTV camera and enters the coverage area of another cabers (which is also a part of the network)
4. The Neural Network now knows enough features about the original person's image (object) to re-identify it from the 2nd camera's feed/pictures.

Applications

Person re-identification is a major step towards automated and AI-powered security and surveillance. Manually having to go through video footages to go over or constantly monitor video feed be it live or after an incident is highly time consuming and also error-prone. By automating this process with high accuracy and reliability, this takes improves video surveillance trust and security.

Dataset Description

The dataset that has been chosen is the CUHK03 dataset.

It was published by The Chinese University of Honk Kong. It is the 3rd edition of a series of 3 datasets (as of now). It has a considerably good amount of research usage and has been in common use among Chinese researchers.

The dataset has

1. 1,360 identities (People/Persons)
2. 13,164 images (Multiple images of the same person for feature learning and testing)
3. The images are a mixture of manually cropped and automatically detected

This in a considerable upgrade to the previous version of CUHK datasets (CUHK01 and CUHK02) which had

1. CUHK01 - 971 identities, 3884 images, manually cropped (Published in 2012)
2. CUHK02 - 1816 identities, 7264 images, manually cropped (Published in 2013)

And therefore, CUHK03 was the first person re-identification dataset that is large enough for deep learning.

CUHK03 dataset was published in 2014. It provides the bounding boxes detected from DPM and manually labelling.

Proposed Methodology and Architecture

This implementation is based on the paper - ***An Improved Deep Learning Architecture for Person Re-Identification*** *by Ejaz Ahmed, Michael Jones and Tim K. Marks.*

In all fairness, this methodology is no longer state of the art. But it was a successful approach in improving the efficiency of feature extraction and object detection in 2015. This was one of the first few papers that attempted to use (the proposed methodology) - **deep learning for Person Re-identification as binary classification**.

Incorporating deep learning in image re-identification means

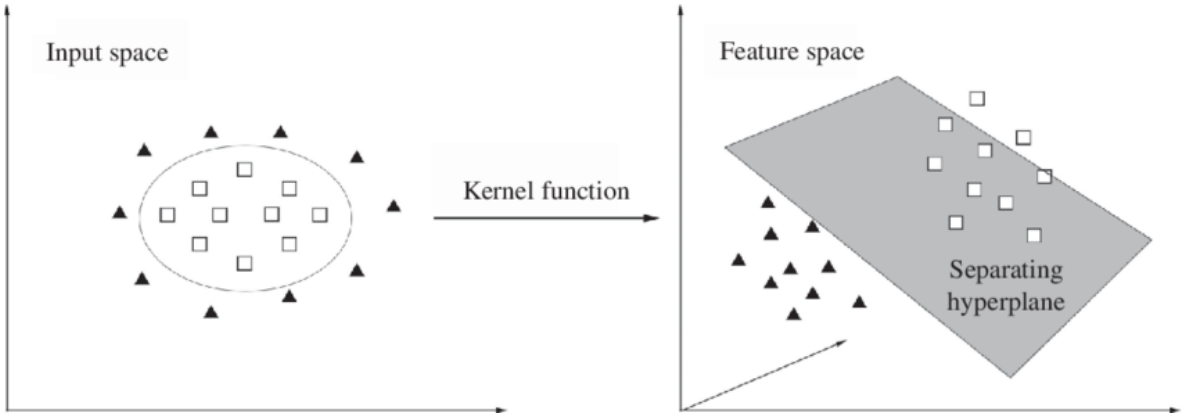
1. Inputting 2 images both of which contain a person's full body
2. Output a Similarity Score between the 2 images or
3. Classification of the pair of images as *same* or *different* (based on whether or not its the same person in the 2 pictures).

Typically image re-identification methodology is of 2 main components -

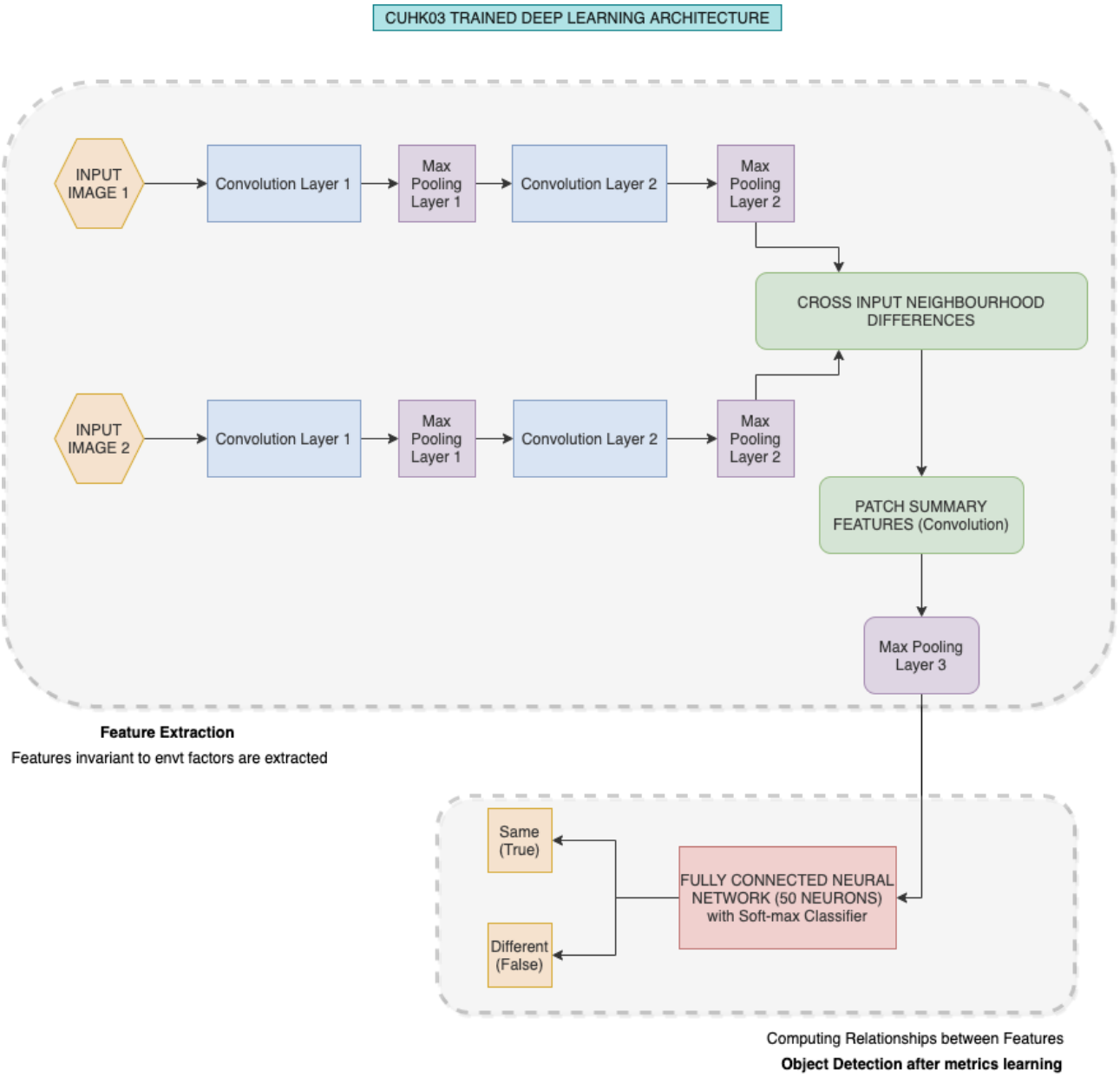
A. A method for extracting features from input images

B. A metric for comparing the features

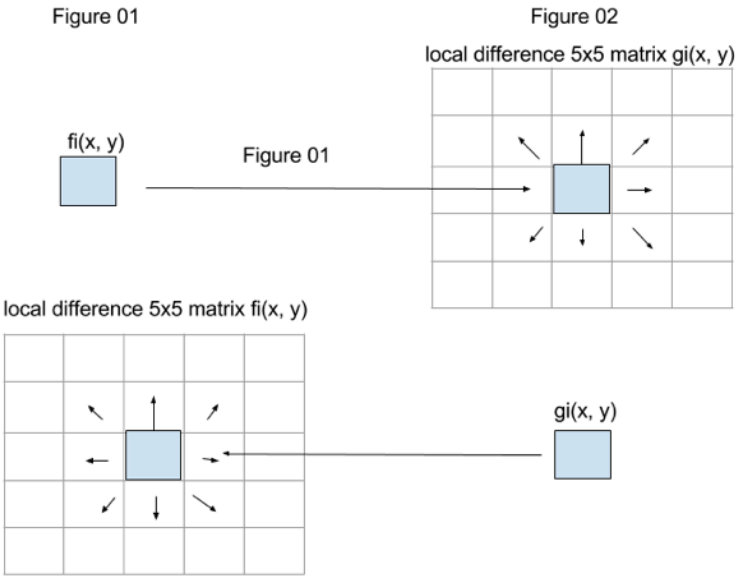
Most research and innovation focusses on finding better features that are at least partially invariant to light, pose and view-point changes. As for metric learning, learning approaches to find a mapping from feature space to a new space in which feature vectors from same image are closer than vectors from different image pairs.



The proposed deep learning architecture has the following layers



1. 2 Tied Convolution Layers - For feature extraction
2. Each of this is followed by a Max Pooling layer (1 & 2) - reduces the size of the image by a factor of 2.
3. Cross input neighbourhood difference - The aim is computed a rough relationship among features from the two input images in the form of neighbourhood difference maps. It is done to add robustness to positional differences in corresponding features of the two input images. ReLu Activation Function is used to add non-linearity.



4. Patch summary features - Another convolution layer summaries these neighbourhood difference maps by producing a summary representation of the differences.
5. Max Pooling Layer 3 - reduces the dimensions to 18x5 pixels in the end.
6. Fully connected neural network - This is where the relationship is found with ReLu and Softmax activation function.

Algorithm

The approach/algorithm used is Convolution Neural Networks with Cross Input Neighbourhood Difference.

A step by step algorithm for classification:

1. A neural network with the above architecture is created with Tensor-flow.
2. The CUHK03 dataset is used to train the model (with the cuhk04.mat file) against 13,164 images and 1,360 identities.
3. Images are convolved and max pooled to simplify and reduce the un-wanted features and retain the core features that help identify the person
4. This common knowledge about ‘what makes a person a person’ is learned during the training process
5. 2 Input images are inputted in the program
6. They are convolved and max-pooled like the training images. The 2 images are passed into cross input neighbourhood difference layer - position invariance is improved.
7. The fully connected ANN detects the image and if the features are identical then the classifier returns true. Else it return false.

Experimental Set Up

Main Environment and Dependencies:

I. Mac OSX 10.15.6 - 4 Cores CPU - i5 Processor - 1600 MHz DDR3

II. Python 3.6.4

III. TensorFlow 1.8

IV. opencv-python 4.4.0.42

V. numpy 1.19.2

VI. h5py 2.10.0

Tensor-flow Co-dependencies:

- absl-py 0.10.0
- astor 0.8.1
- bleach 1.5.0
- gast 0.4.0
- grpcio 1.32.0
- protobuf3.13.0
- six1.15.0
- termcolor1.1.0
- Werkzeug1.0.1
- zipp3.1.0

Implementation Progress

Status: the data preparation utilities from the dataset and architecture definition is in completed. Training happens but number of epochs is high and the hardware environment is enough as of now.

Data Preparation Utilities

def prepare_data(path):

```
f = h5py.File('%s/cuhk-03.mat' % path)
labeled = [f['labeled'][0][i] for i in range(len(f['labeled'][0]))]
labeled = [f[labeled[0]][i] for i in range(len(f[labeled[0]]))]
detected = [f['detected'][0][i] for i in range(len(f['detected'][0]))]
detected = [f[detected[0]][i] for i in range(len(f[detected[0]]))]
datasets = [['labeled', labeled], ['detected', detected]]
prev_id = 0
```

```
    for dataset in datasets:
        if not os.path.exists('%s/%s/train' % (path, dataset[0])):
            os.makedirs('%s/%s/train' % (path, dataset[0]))
        if not os.path.exists('%s/%s/val' % (path, dataset[0])):
            os.makedirs('%s/%s/val' % (path, dataset[0]))

        for i in range(0, len(dataset[1])):
            for j in range(len(dataset[1][0])):
                try:
                    image = np.array(f[dataset[1][i][j]]).transpose((2, 1, 0))
                    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                    image = cv2.imencode('.jpg', image)[1].tostring()
                    if len(dataset[1][0]) - j <= 100:
                        filepath = '%s/%s/val/%04d_%02d.jpg' % (path, dataset[0], j - prev_id - 1,
i)
                    else:
                        filepath = '%s/%s/train/%04d_%02d.jpg' % (path, dataset[0], j, i)
                        prev_id = j
                    with open(filepath, 'wb') as image_file:
                        image_file.write(image)
                except Exception as e:
                    continue



def get_num_id(path, set):
    files = os.listdir('%s/labeled/%s' % (path, set))
    files.sort()
    return int(files[-1].split('_')[0]) - int(files[0].split('_')[0]) + 1

def read_data(path, set, num_id, image_width, image_height, batch_size):
    batch_images = []
    labels = []
    for i in range(batch_size // 2):
```

```
            pairs = [get_pair(path, set, num_id, True), get_pair(path, set, num_id, False)]
            for pair in pairs:
                images = []
                for p in pair:
                    image = cv2.imread(p)
                    image = cv2.resize(image, (image_width, image_height))
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                    images.append(image)
                batch_images.append(images)
            labels.append([1., 0.])
            labels.append([0., 1.])

        '''
        for pair in batch_images:
            for p in pair:
                cv2.imshow('img', p)
                key = cv2.waitKey(0)
                if key == 1048603:
                    exit()
        '''
        return np.transpose(batch_images, (1, 0, 2, 3, 4)), np.array(labels)

if __name__ == '__main__':
    prepare_data(sys.argv[1])
```

Network Architecture Definitions

```
def network(images1, images2, weight_decay):
    with tf.variable_scope('network'):
        # Tied Convolution
        conv1_1 = tf.layers.conv2d(images1, 20, [5, 5], activation=tf.nn.relu,
            kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='conv1_1')
        pool1_1 = tf.layers.max_pooling2d(conv1_1, [2, 2], [2, 2], name='pool1_1')
```

```
    conv1_2 = tf.layers.conv2d(pool1_1, 25, [5, 5], activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='conv1_2')
    pool1_2 = tf.layers.max_pooling2d(conv1_2, [2, 2], [2, 2], name='pool1_2')
    conv2_1 = tf.layers.conv2d(images2, 20, [5, 5], activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='conv2_1')
    pool2_1 = tf.layers.max_pooling2d(conv2_1, [2, 2], [2, 2], name='pool2_1')
    conv2_2 = tf.layers.conv2d(pool2_1, 25, [5, 5], activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='conv2_2')
    pool2_2 = tf.layers.max_pooling2d(conv2_2, [2, 2], [2, 2], name='pool2_2')
# Cross-Input Neighborhood Differences
    trans = tf.transpose(pool1_2, [0, 3, 1, 2])
    shape = trans.get_shape().as_list()
    m1s = tf.ones([shape[0], shape[1], shape[2], shape[3], 5, 5])
    reshape = tf.reshape(trans, [shape[0], shape[1], shape[2], shape[3], 1, 1])
    f = tf.multiply(reshape, m1s)

    trans = tf.transpose(pool2_2, [0, 3, 1, 2])
    reshape = tf.reshape(trans, [1, shape[0], shape[1], shape[2], shape[3]])
    g = []
    pad = tf.pad(reshape, [[0, 0], [0, 0], [0, 0], [2, 2], [2, 2]])
    for i in range(shape[2]):
        for j in range(shape[3]):
            g.append(pad[:,:,:,i:i+5,j:j+5])

    concat = tf.concat(g, axis=0)
    reshape = tf.reshape(concat, [shape[2], shape[3], shape[0], shape[1], 5, 5])
    g = tf.transpose(reshape, [2, 3, 0, 1, 4, 5])
    reshape1 = tf.reshape(tf.subtract(f, g), [shape[0], shape[1], shape[2] * 5, shape[3] * 5])
    reshape2 = tf.reshape(tf.subtract(g, f), [shape[0], shape[1], shape[2] * 5, shape[3] * 5])
    k1 = tf.nn.relu(tf.transpose(reshape1, [0, 2, 3, 1]), name='k1')
    k2 = tf.nn.relu(tf.transpose(reshape2, [0, 2, 3, 1]), name='k2')
# Patch Summary Features
    l1 = tf.layers.conv2d(k1, 25, [5, 5], (5, 5), activation=tf.nn.relu,
```

```
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='l1')
    l2 = tf.layers.conv2d(k2, 25, [5, 5], (5, 5), activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='l2')

    # Across-Patch Features
    m1 = tf.layers.conv2d(l1, 25, [3, 3], activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='m1')
    pool_m1 = tf.layers.max_pooling2d(m1, [2, 2], [2, 2], padding='same',
name='pool_m1')
    m2 = tf.layers.conv2d(l2, 25, [3, 3], activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(weight_decay), name='m2')
    pool_m2 = tf.layers.max_pooling2d(m2, [2, 2], [2, 2], padding='same',
name='pool_m2')

    # Higher-Order Relationships
    concat = tf.concat([pool_m1, pool_m2], axis=3)
    reshape = tf.reshape(concat, [FLAGS.batch_size, -1])
    fc1 = tf.layers.dense(reshape, 500, tf.nn.relu, name='fc1')
    fc2 = tf.layers.dense(fc1, 2, name='fc2')

    return fc2

def main(argv=None):
  if FLAGS.mode == 'test':
    FLAGS.batch_size = 1

  learning_rate = tf.placeholder(tf.float32, name='learning_rate')
  images = tf.placeholder(tf.float32, [2, FLAGS.batch_size, IMAGE_HEIGHT,
IMAGE_WIDTH, 3], name='images')
  labels = tf.placeholder(tf.float32, [FLAGS.batch_size, 2], name='labels')
  is_train = tf.placeholder(tf.bool, name='is_train')
  global_step = tf.Variable(0, name='global_step', trainable=False)
  weight_decay = 0.0005
  tarin_num_id = 0
```

```
    val_num_id = 0

    if FLAGS.mode == 'train':
        tarin_num_id = cuhk03_dataset.get_num_id(FLAGS.data_dir, 'train')
    images1, images2 = preprocess(images, is_train)

    print('Build network')
    logits = network(images1, images2, weight_decay)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=labels,
logits=logits))
    inference = tf.nn.softmax(logits)

    optimizer = tf.train.MomentumOptimizer(learning_rate, momentum=0.9)
    train = optimizer.minimize(loss, global_step=global_step)
    lr = FLAGS.learning_rate

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        saver = tf.train.Saver()

        ckpt = tf.train.get_checkpoint_state(FLAGS.logs_dir)
        if ckpt and ckpt.model_checkpoint_path:
            print('Restore model')
            saver.restore(sess, ckpt.model_checkpoint_path)

        if FLAGS.mode == 'train':
            step = sess.run(global_step)
            for i in range(step, FLAGS.max_steps + 1):
                batch_images, batch_labels = cuhk03_dataset.read_data(FLAGS.data_dir,
'train', tarin_num_id,
                    IMAGE_WIDTH, IMAGE_HEIGHT, FLAGS.batch_size)
                feed_dict = {learning_rate: lr, images: batch_images,
                    labels: batch_labels, is_train: True}
                sess.run(train, feed_dict=feed_dict)
```

```
            train_loss = sess.run(loss, feed_dict=feed_dict)
            print('Step: %d, Learning rate: %f, Train loss: %f' % (i, lr, train_loss))

            lr = FLAGS.learning_rate * ((0.0001 * i + 1) ** -0.75)
            if i % 1000 == 0:
                saver.save(sess, FLAGS.logs_dir + 'model.ckpt', i)

    elif FLAGS.mode == 'test':
        image1 = cv2.imread(FLAGS.image1)
        image1 = cv2.resize(image1, (IMAGE_WIDTH, IMAGE_HEIGHT))
        image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
        image1 = np.reshape(image1, (1, IMAGE_HEIGHT, IMAGE_WIDTH,
3)).astype(float)
        image2 = cv2.imread(FLAGS.image2)
        image2 = cv2.resize(image2, (IMAGE_WIDTH, IMAGE_HEIGHT))
        image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
        image2 = np.reshape(image2, (1, IMAGE_HEIGHT, IMAGE_WIDTH,
3)).astype(float)
        test_images = np.array([image1, image2])

        feed_dict = {images: test_images, is_train: False}
        prediction = sess.run(inference, feed_dict=feed_dict)
        print(bool(not np.argmax(prediction[0])))
```

Training

```
(image-processing-reid) sanjitkumar@Sanjits-MacBook-Air person-reid % python run.py --data_dir=../Person-Re-identification/Dataset/cuhk03_release
/Users/sanjitkumar/.pyenv/versions/image-processing-reid/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passin
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/Users/sanjitkumar/.pyenv/versions/image-processing-reid/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passin
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/Users/sanjitkumar/.pyenv/versions/image-processing-reid/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:521: FutureWarning: Passin
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/Users/sanjitkumar/.pyenv/versions/image-processing-reid/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:522: FutureWarning: Passin
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/Users/sanjitkumar/.pyenv/versions/image-processing-reid/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passin
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/Users/sanjitkumar/.pyenv/versions/image-processing-reid/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:528: FutureWarning: Passin
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
Build network
WARNING:tensorflow:From run.py:134: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed in a futu
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See @{tf.nn.softmax_cross_entropy_with_logits_v2}.

2020-09-21 16:00:16.785251: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not c
Restore model
Step: 1, Learning rate: 0.080000, Train loss: 0.743854
Step: 2, Learning rate: 0.079994, Train loss: 0.811584
Step: 3, Learning rate: 0.079988, Train loss: 0.696699
Step: 4, Learning rate: 0.079982, Train loss: 0.709823
Step: 5, Learning rate: 0.079976, Train loss: 0.694080
Step: 6, Learning rate: 0.079970, Train loss: 0.693196
Step: 7, Learning rate: 0.079964, Train loss: 0.698665
Step: 8, Learning rate: 0.079958, Train loss: 0.696844
Step: 9, Learning rate: 0.079952, Train loss: 0.693672
Step: 10, Learning rate: 0.079946, Train loss: 0.698551
Step: 11, Learning rate: 0.079940, Train loss: 0.696277
Step: 12, Learning rate: 0.079934, Train loss: 0.692606
Step: 13, Learning rate: 0.079928, Train loss: 0.691726
Step: 14, Learning rate: 0.079922, Train loss: 0.692093
Step: 15, Learning rate: 0.079916, Train loss: 0.694674
Step: 16, Learning rate: 0.079910, Train loss: 0.695065
Step: 17, Learning rate: 0.079904, Train loss: 0.692614
Step: 18, Learning rate: 0.079898, Train loss: 0.691507
Step: 19, Learning rate: 0.079892, Train loss: 0.693488
Step: 20, Learning rate: 0.079886, Train loss: 0.691745
Step: 21, Learning rate: 0.079880, Train loss: 0.692064
```