1 FlowNet Definition

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
import load_dataset as ld

weights_dict = dict()
def load_weights_from_file(weight_file):
    try:
        weights_dict = np.load(weight_file, allow_pickle=True).item()
    except:
        weights_dict = np.load(weight_file, allow_pickle=True, encoding='bytes
            ').item()
    return weights_dict

def set_layer_weights(model, weights_dict):
    for layer in model.layers:
        if layer.name in weights_dict:
            print(layer.name)
            cur_dict = weights_dict[layer.name]
            current_layer_parameters = list()
            if layer.__class__.__name__ == "BatchNormalization":
                if 'scale' in cur_dict:
                    current_layer_parameters.append(cur_dict['scale'])
                if 'bias' in cur_dict:
                    current_layer_parameters.append(cur_dict['bias'])
                current_layer_parameters.extend([cur_dict['mean'], cur_dict['
                    var']])
            elif layer.__class__.__name__ == "Scale":
                if 'scale' in cur_dict:
                    current_layer_parameters.append(cur_dict['scale'])
                if 'bias' in cur_dict:
                    current_layer_parameters.append(cur_dict['bias'])
            elif layer.__class__.__name__ == "SeparableConv2D":
                current_layer_parameters = [cur_dict['depthwise_filter'],
                    cur_dict['pointwise_filter']]
                if 'bias' in cur_dict:
                    current_layer_parameters.append(cur_dict['bias'])
            elif layer.__class__.__name__ == "Embedding":
                current_layer_parameters.append(cur_dict['weights'])
            elif layer.__class__.__name__ == "PReLU":
                gamma =  np.ones(list(layer.input_shape[1:]))*cur_dict['gamma'
                    ]
                current_layer_parameters.append(gamma)
            else:
                # rot
                if 'weights' in cur_dict:
```

```
                    current_layer_parameters = [cur_dict['weights']]
                if 'bias' in cur_dict:
                    current_layer_parameters.append(np.squeeze(cur_dict['bias'
                        ]))
            model.get_layer(layer.name).set_weights(current_layer_parameters)
    return model

def step_schedule(epoch):
    #this learning rate may actually be incorrect
    #base_lr = 1e-5 in solver prototype
    #step interval is also off...
    l = 1e-4
    #this is supposed to happen after epochs or batches?
    n = epoch - 300000
    if n > 0:
        return l/(2 << n//100000)
    else:
        return l

def EPE(y_true, y_pred):
    y_true = y_true * 0.05
    dim = y_pred.shape.as_list()[1:-1]
    #print(dim)
    if y_true.shape != y_pred.shape:
        #lets hope batching works correctly
        y_true = tf.image.resize(y_true, size=dim, method=tf.image.
            ResizeMethod.BILINEAR)
    dist = tf.norm(y_pred - y_true, ord='euclidean', axis=-1)
    return tf.reduce_mean(dist)

def EPE_Accuracy(y_true, y_pred):
    y_true = y_true * 0.5
    dist = tf.norm(y_pred - y_true, ord='euclidean', axis=-1)
    return tf.reduce_mean(dist)

def FlowNetS_deployed(weight_file = None, trainable = False):
    weights_dict = load_weights_from_file(weight_file) if not weight_file ==
        None else None
    #stride of 2 for each layer
    #relu after each layer
    #inputs = keras.Input(shape=(384,512,6))
    img1 = keras.Input(shape=(384,512,3))
    img2 = keras.Input(shape=(384,512,3))
    catimg = layers.Concatenate(axis=3)([img1,img2])
    #inputs = layers.Concatenate(axis=3)(inputs['img0'], inputs['img1'])
    #perform concaction in network
    x = layers.Conv2D(64, 7, 2, padding='same', name='conv1', activation='relu
        ')(catimg)
    c2out = layers.Conv2D(128, 5, 2, padding='same', name='conv2', activation=
        'relu')(x)
    x = layers.Conv2D(256, 5, 2, padding='same', name='conv3', activation='
```

```
    relu')(c2out)
c31out = layers.Conv2D(256, 3, 1, padding='same', name='conv3_1',
    activation='relu')(x)
x = layers.Conv2D(512, 3, 2, padding='same', name='conv4', activation='
    relu')(c31out)
c41out = layers.Conv2D(512, 3, 1, padding='same', name='conv4_1',
    activation='relu')(x)
x = layers.Conv2D(512, 3, 2, padding='same', name='conv5', activation='
    relu')(c41out)
c51out = layers.Conv2D(512, 3, 1, padding='same', name='conv5_1',
    activation='relu')(x)
x = layers.Conv2D(1024, 3, 2, padding='same', name='conv6', activation='
    relu')(c51out)
#add in extra c6_1 layer from release model
c61out = layers.Conv2D(1024, 3, 1, padding='same', name='conv6_1',
    activation='relu')(x)

#Refinement section
#kernel size 4 instead of 5?
decon5 = layers.Conv2DTranspose(512, 4, 2, padding='same', name='deconv5',
     activation='relu')(c61out)
flow6 = layers.Conv2D(2, 3, 1, padding='same', name='Convolution1')(c61out
    )
flow6cup = layers.Conv2DTranspose(2, 4, 2, padding='same', name='
    upsample_flow6to5')(flow6)
#make sure to check the order on those concats
cat2 = layers.Concatenate(axis=3)([c51out,decon5,flow6cup])

decon4 = layers.Conv2DTranspose(256, 4, 2, padding='same', name='deconv4',
     activation='relu')(cat2)
flow5 = layers.Conv2D(2, 3, padding='same', name='Convolution2')(cat2)
flow5up = layers.Conv2DTranspose(2, 4, 2, padding='same', name='
    upsample_flow5to4')(flow5)
#it may be worth building a custom layer for this as it repeats a few
    times
cat3 = layers.Concatenate(axis=3)([c41out, decon4, flow5up])

decon3 = layers.Conv2DTranspose(128, 4, 2, padding='same', name='deconv3',
     activation='relu')(cat3)
flow4 = layers.Conv2D(2, 3, padding='same', name='Convolution3')(cat3)
flow4up = layers.Conv2DTranspose(2, 4, 2, padding='same', name='
    upsample_flow4to3')(flow4)
cat4 = layers.Concatenate(axis=3)([c31out, decon3, flow4up])

decon2 = layers.Conv2DTranspose(64, 4, 2, padding='same', name='deconv2',
    activation='relu')(cat4)
flow3 = layers.Conv2D(2, 3, padding='same', name='Convolution4')(cat4)
flow3up = layers.Conv2DTranspose(2, 4, 2, padding='same', name='
    upsample_flow3to2')(flow3)
cat5 = layers.Concatenate(axis=3)([c2out, decon2, flow3up])
```

```
    flow2 = layers.Conv2D(2, 3, 1, padding='same', name='Convolution5')(cat5)
    x = flow2*20; #why? because!
    #some more bullshit here
    #padding does nothing here right?
    #some magic interpolation here
    #convolution with constants for scaling purposes see actual model wtf
    #x = layers.experimental.preprocessing.Resizing(384, 512, interpolation="
        bilinear", name='resample4')(x)
    #hardcoded values bad
    flow_full = tf.image.resize(x, size=(384,512), method=tf.image.
        ResizeMethod.BILINEAR, name='flow_full')
    #I don't think this convolution does much
    #outputs = layers.Conv2D(2, 1, 1, padding='valid', name='Convolution6')(x)
    #384, 512 output

    #flow = keras.Input(shape=(384,512,2))
    #why is this
    #flow = flow * 0.05


    model = [];
    if trainable:
        model = Model(inputs = [img1,img2], outputs = [flow_full, flow2, flow3
            , flow4, flow5, flow6])
    else:
        model = Model(inputs = [img1,img2], outputs = [flow_full])

    if weights_dict != None:
        set_layer_weights(model, weights_dict)
    return model

def test(model):
    import flowiz as fz
    path = 'testfiles/0000000-'
    img1 = tf.expand_dims(plt.imread(path+'img0.ppm'), 0)
    img2 = tf.expand_dims(plt.imread(path+'img1.ppm'), 0)
    img1 = tf.cast(img1, tf.float32)/255.0
    img2 = tf.cast(img2, tf.float32)/255.0

    print(img1.shape, img2.shape)
    flow = model.predict([tf.reverse(img1,[-1]),tf.reverse(img2,[-1])])

    plt.figure(1)
    plt.subplot(1,3,1)
    plt.imshow(img1.numpy().squeeze())
    plt.subplot(1,3,2)
    plt.imshow(img2.numpy().squeeze())
    plt.subplot(1,3,3)
    print(flow.shape)
    plt.imshow(fz.convert_from_flow(flow.squeeze()))
    plt.show()
```

```
if __name__ == '__main__':

    #model = FlowNetS_deployed()
    model = FlowNetS_deployed('checkpoints/trained_weights.npy', trainable=
        False)
    model.summary()
    keras.utils.plot_model(model, "FlowNetS_model.png", show_shapes=True)

    #data_valid = ld.get_dataset('FlyingChairs_release/tfrecord/fc_val.
        tfrecords', 4)
    #data_train = ld.get_dataset('FlyingChairs_release/tfrecord/fc_train.
        tfrecords', 4)

    test(model)
```

2 Dataset Creator

```
import tensorflow as tf
from tensorflow import keras
import FlowNet as fn
import load_dataset as ld
import argparse
import random

#script to train the model

def fast_schedule(epoch):
    l = 1e-5
    return l/(2 << epoch//10)

def train(opts):
    #model = fn.FlowNetS_deployed('checkpoints/trained_weights.npy', trainable
        =True)
    model = []
    if opts.starting_weights==None:
        model = fn.FlowNetS_deployed('checkpoints/trained_weights.npy',
            trainable=True)
    else:
        model = fn.FlowNetS_deployed(trainable=True)
        model.load_weights

    model.summary()
    #keras.utils.plot_model(model, "FlowNetS_model.png", show_shapes=True)

    optimizer = tf.keras.optimizers.Adam(1e-4)
    loss_weights = [0.32, 0.08, 0.02, 0.01, 0.005]
    loss = {
        'Convolution5': fn.EPE,
        'Convolution4': fn.EPE,
        'Convolution3': fn.EPE,
        'Convolution2': fn.EPE,
        'Convolution1': fn.EPE
    }
    metrics = {'tf_op_layer_ResizeBilinear': fn.EPE}
    model.compile(optimizer=optimizer, loss=loss, loss_weights=loss_weights,
        metrics=metrics)

    SAVE_PERIOD = 1
    #SESSION_ID = random.randint(0,9999)
    rate_callback = keras.callbacks.LearningRateScheduler(fast_schedule)
    checkpoint_callback = keras.callbacks.ModelCheckpoint(
        #filepath='checkpoints/model-2-{SESSION_ID:04d}-{epoch:04d}.hdf5',
        filepath='checkpoints/model-2-{epoch:04d}.hdf5',
        save_freq='epoch',
        #     period='SAVE_PERIOD',
        save_weights_only=True)
```

```
    history_callback = keras.callbacks.CSVLogger(opts.stats_output, append=
        True)

    callbacks = [rate_callback, checkpoint_callback, history_callback]
    batch_size = 4

    data_valid = ld.get_dataset('FlyingChairs_release/tfrecord/fc_val.
        tfrecords', batch_size)
    data_train = ld.get_dataset('FlyingChairs_release/tfrecord/fc_train.
        tfrecords', batch_size)

    #history = model.fit(x, y, batch_size=8, epochs=1, callbacks=[
        rate_callback])

    model.fit(data_train, batch_size=4, epochs=opts.epochs, validation_data=
        data_valid, callbacks=callbacks, verbose=2)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--starting-weights',
        type=str,
        required=False,
        help='File of starting weights, default is pretrained from paper'
    )
    parser.add_argument(
        '--epochs',
        type=int,
        required=True,
        help='Number of training epochs to take'
    )
    parser.add_argument(
        '--stats-output',
        type=str,
        required=True,
        help='Output file for stats'
    )
    flags = parser.parse_args()
    print(flags)
    train(flags)
```

3 Dataset Loader

```
import tensorflow as tf
#chair_train_dataset = tf.data.TFRecordDataset('FlyingChairs_release/tfrecord/
    fc_train.tfrecords')
#chair_val_dataset = tf.data.TFRecordDataset('FlyingChairs_release/tfrecord/
    fc_val.tfrecords')

# Custom dataset reader class
# https://www.tensorflow.org/tutorials/load_data/tfrecord

AUTOTUNE = tf.data.experimental.AUTOTUNE

def _parse_record(example):
    tfrecord_format = {
        'height': tf.io.FixedLenFeature((), tf.int64),
        'width': tf.io.FixedLenFeature((), tf.int64),
        'img1': tf.io.FixedLenFeature((), tf.string),
        'img2': tf.io.FixedLenFeature((), tf.string),
        'flow': tf.io.FixedLenFeature((393216), tf.float32),
    }
    example = tf.io.parse_single_example(example, tfrecord_format)
    #print(example.keys())
    #these don't work for some fucking reason
    h = example['height']
    w = example['width']
    h = 384
    w = 512
    img1 = tf.io.parse_tensor(example['img1'], tf.uint8)
    img1 = tf.reshape(img1, [h,w,3])
    img1 = tf.cast(img1, tf.float32)/255.0
    img1 = tf.reverse(img1, [-1])
    #reverse order to BGR as in Caffe

    img2 = tf.io.parse_tensor(example['img2'], tf.uint8)
    img2 = tf.reshape(img2, [h,w,3])
    img2 = tf.cast(img2, tf.float32)/255.0
    img2 = tf.reverse(img2, [-1])

    #print(img1.shape, img2.shape)
    flow = example['flow']
    flow = tf.reshape(flow, [h,w,2])
    #is there way to auto generate dict?
    return (img1, img2), flow
    #return {'img1':img1, 'img2':img2, 'flow':flow}

def load_dataset(filename):
    dataset = tf.data.TFRecordDataset(filename, compression_type='ZLIB')
    dataset = dataset.map(_parse_record, num_parallel_calls=1)
    return dataset
```

```
def get_dataset(filename, batch_size):
    dataset = load_dataset(filename)
    dataset = dataset.prefetch(buffer_size=AUTOTUNE)
    dataset = dataset.batch(batch_size)
    return dataset

def test_dataset(filename):
    import matplotlib.pyplot as plt
    import flowiz as fz
    data = get_dataset(filename, 1)
    (img1, img2), flow = next(iter(data))

    plt.figure(1)
    plt.subplot(1,3,1)
    plt.imshow(img1.numpy().squeeze())
    plt.subplot(1,3,2)
    plt.imshow(img2.numpy().squeeze())
    plt.subplot(1,3,3)
    plt.imshow(fz.convert_from_flow(flow.numpy().squeeze()))
    plt.show()

'''
import tensorflow as tf
import load_dataset as ld
ld.test_dataset('FlyingChairs_release/tfrecord/fc_val.tfrecords')
'''
```

4 Dataset Creator

```
import os
# Disable GPU to avoid conflicting with other TF sessons
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import argparse
import glob
import IO
import sys

# Based on code by Sam Pepose

def _bytes_feature(value):
  """Returns a bytes_list from a string / byte."""
  if isinstance(value, type(tf.constant(0))):
    value = value.numpy() # BytesList won't unpack a string from an
        EagerTensor.
  return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _float_feature(value):
  """Returns a float_list from a float / double."""
  return tf.train.Feature(float_list=tf.train.FloatList(value=value))

def _int64_feature(value):
  """Returns an int64_list from a bool / enum / int / uint."""
  return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))

def image_example(img1, img2, flow):
    #assume all are same shape, checks done
    shape = img1.shape
    feature = {
        'height': _int64_feature(shape[0]),
        'width': _int64_feature(shape[1]),
        'img1': _bytes_feature(tf.io.serialize_tensor(img1.flatten())),
        'img2': _bytes_feature(tf.io.serialize_tensor(img2.flatten())),
        'flow': _float_feature(flow)
    }
    #haha what is this line
    return tf.train.Example(features=tf.train.Features(feature=feature))

# Flo reader based on
# https://stackoverflow.com/questions/28013200/reading-middlebury-flow-files-
    with-python-bytes-array-numpy
def open_flo_file(filename):
    with open(filename, 'rb') as f:
        magic, = np.fromfile(f, np.float32, count=1)
        if 202021.25 != magic:
```

```
            print('Magic number incorrect. Invalid .flo file')
        else:
            w,h = np.fromfile(f, np.int32, count=2)
            #h = np.fromfile(f, np.int32, count=1)
            data = np.fromfile(f, np.float32, count=2*w*h)
            # Reshape data into 3D array (columns, rows, bands)
            #return np.resize(data, (w[0], h[0], 2))
            #print(data.size)
            return w, h, data

def open_ppm_file(filename):
    return np.asarray(Image.open(filename))


path = 'FlyingChairs_release/'

def convert_dataset(indices, name):
  filename = os.path.join(FLAGS.out, name + '.tfrecords')
  writer_opt = tf.io.TFRecordOptions(compression_type='ZLIB')
  writer = tf.io.TFRecordWriter(filename, options=writer_opt)
  for i in indices:
    img1_file = os.path.join(FLAGS.data_dir, '%05d_img1.ppm' % (i + 1))
    img2_file = os.path.join(FLAGS.data_dir, '%05d_img2.ppm' % (i + 1))
    flow_file = os.path.join(FLAGS.data_dir, '%05d_flow.flo' % (i + 1))

    flo_w, flo_h, flo_dat = open_flo_file(flow_file.strip())
    img1_dat = open_ppm_file(img1_file)
    img2_dat = open_ppm_file(img2_file)

    if img1_dat.shape != img2_dat.shape:
      print('error, image shape mismatch')
      print(i)
    if img1_dat.shape[0:2] != (flo_h,flo_w):
      print('error, flo shape mismatch')
      print(i)
      print(img1_dat.shape, (flo_h,flo_w))

    tf_example = image_example(img1_dat, img2_dat, flo_dat)
    writer.write(tf_example.SerializeToString())
  writer.close()

TRAIN = 1
VAL = 2
def main():
    # Load train/val split into arrays
    train_val_split = np.loadtxt(FLAGS.train_val_split)
    train_idxs = np.flatnonzero(train_val_split == TRAIN)
    val_idxs = np.flatnonzero(train_val_split == VAL)

    # Convert the train and val datasets into .tfrecords format
    print('Converting validation set')
```

```
    convert_dataset(val_idxs, 'fc_val')
    print('Converting training set')
    convert_dataset(train_idxs, 'fc_train')


if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
    '--data_dir',
    type=str,
    required=True,
    help='Directory that includes all .ppm and .flo files in the dataset'
  )
  parser.add_argument(
    '--train_val_split',
    type=str,
    required=True,
    help='Path to text file with train-validation split (1-train, 2-validation
        )'
  )
  parser.add_argument(
    '--out',
    type=str,
    required=True,
    help='Directory for output .tfrecords files'
  )
  FLAGS = parser.parse_args()

  # Verify arguments are valid
  if not os.path.isdir(FLAGS.data_dir):
    raise ValueError('data_dir must exist and be a directory')
  if not os.path.isdir(FLAGS.out):
    raise ValueError('out must exist and be a directory')
  if not os.path.exists(FLAGS.train_val_split):
    raise ValueError('train_val_split must exist')
  main()
```

5 IO Routines

```
#!/usr/bin/env python3.4

import os
import re
import numpy as np
import uuid
from scipy import misc
import numpy as np
from PIL import Image
import sys


def read(file):
    if file.endswith('.float3'): return readFloat(file)
    elif file.endswith('.flo'): return readFlow(file)
    elif file.endswith('.ppm'): return readImage(file)
    elif file.endswith('.pgm'): return readImage(file)
    elif file.endswith('.png'): return readImage(file)
    elif file.endswith('.jpg'): return readImage(file)
    elif file.endswith('.pfm'): return readPFM(file)[0]
    else: raise Exception('don\'t know how to read %s' % file)

def write(file, data):
    if file.endswith('.float3'): return writeFloat(file, data)
    elif file.endswith('.flo'): return writeFlow(file, data)
    elif file.endswith('.ppm'): return writeImage(file, data)
    elif file.endswith('.pgm'): return writeImage(file, data)
    elif file.endswith('.png'): return writeImage(file, data)
    elif file.endswith('.jpg'): return writeImage(file, data)
    elif file.endswith('.pfm'): return writePFM(file, data)
    else: raise Exception('don\'t know how to write %s' % file)

def readPFM(file):
    file = open(file, 'rb')

    color = None
    width = None
    height = None
    scale = None
    endian = None

    header = file.readline().rstrip()
    if header.decode("ascii") == 'PF':
        color = True
    elif header.decode("ascii") == 'Pf':
        color = False
    else:
        raise Exception('Not a PFM file.')
```

```
    dim_match = re.match(r'^(\d+)\s(\d+)\s$', file.readline().decode("ascii"))
    if dim_match:
        width, height = list(map(int, dim_match.groups()))
    else:
        raise Exception('Malformed PFM header.')

    scale = float(file.readline().decode("ascii").rstrip())
    if scale < 0: # little-endian
        endian = '<'
        scale = -scale
    else:
        endian = '>' # big-endian

    data = np.fromfile(file, endian + 'f')
    shape = (height, width, 3) if color else (height, width)

    data = np.reshape(data, shape)
    data = np.flipud(data)
    return data, scale

def writePFM(file, image, scale=1):
    file = open(file, 'wb')

    color = None

    if image.dtype.name != 'float32':
        raise Exception('Image dtype must be float32.')

    image = np.flipud(image)

    if len(image.shape) == 3 and image.shape[2] == 3: # color image
        color = True
    elif len(image.shape) == 2 or len(image.shape) == 3 and image.shape[2] ==
        1: # greyscale
        color = False
    else:
        raise Exception('Image must have H x W x 3, H x W x 1 or H x W
            dimensions.')

    file.write('PF\n' if color else 'Pf\n'.encode())
    file.write('%d %d\n'.encode() % (image.shape[1], image.shape[0]))

    endian = image.dtype.byteorder

    if endian == '<' or endian == '=' and sys.byteorder == 'little':
        scale = -scale

    file.write('%f\n'.encode() % scale)

    image.tofile(file)
```

```
def readFlow(name):
    if name.endswith('.pfm') or name.endswith('.PFM'):
        return readPFM(name)[0][:,:,0:2]

    f = open(name, 'rb')

    header = f.read(4)
    if header.decode("utf-8") != 'PIEH':
        raise Exception('Flow file header does not contain PIEH')

    width = np.fromfile(f, np.int32, 1).squeeze()
    height = np.fromfile(f, np.int32, 1).squeeze()

    flow = np.fromfile(f, np.float32, width * height * 2).reshape((height,
        width, 2))

    return flow.astype(np.float32)

def readImage(name):
    if name.endswith('.pfm') or name.endswith('.PFM'):
        data = readPFM(name)[0]
        if len(data.shape)==3:
            return data[:,:,0:3]
        else:
            return data

    return misc.imread(name)

def writeImage(name, data):
    if name.endswith('.pfm') or name.endswith('.PFM'):
        return writePFM(name, data, 1)

    return misc.imsave(name, data)

def writeFlow(name, flow):
    f = open(name, 'wb')
    f.write('PIEH'.encode('utf-8'))
    np.array([flow.shape[1], flow.shape[0]], dtype=np.int32).tofile(f)
    flow = flow.astype(np.float32)
    flow.tofile(f)

def readFloat(name):
    f = open(name, 'rb')

    if(f.readline().decode("utf-8")) != 'float\n':
        raise Exception('float file %s did not contain <float> keyword' % name
            )

    dim = int(f.readline())

    dims = []
```

```
    count = 1
    for i in range(0, dim):
        d = int(f.readline())
        dims.append(d)
        count *= d

    dims = list(reversed(dims))

    data = np.fromfile(f, np.float32, count).reshape(dims)
    if dim > 2:
        data = np.transpose(data, (2, 1, 0))
        data = np.transpose(data, (1, 0, 2))

    return data

def writeFloat(name, data):
    f = open(name, 'wb')

    dim=len(data.shape)
    if dim>3:
        raise Exception('bad float file dimension: %d' % dim)

    f.write(('float\n').encode('ascii'))
    f.write(('%d\n' % dim).encode('ascii'))

    if dim == 1:
        f.write(('%d\n' % data.shape[0]).encode('ascii'))
    else:
        f.write(('%d\n' % data.shape[1]).encode('ascii'))
        f.write(('%d\n' % data.shape[0]).encode('ascii'))
        for i in range(2, dim):
            f.write(('%d\n' % data.shape[i]).encode('ascii'))

    data = data.astype(np.float32)
    if dim==2:
        data.tofile(f)

    else:
        np.transpose(data, (2, 0, 1)).tofile(f)
```

6 TFRecord Tester

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
import tensorflow as tf
import glob
import flowiz as fz
import matplotlib.pyplot as plt
import IO
import numpy as np

# Test validity of tfrecord conversion

path = 'FlyingChairs_release/tfrecord/fc_val.tfrecords'
#path = 'FlyingChairs_release/images.tfrecords'
feature = {
    'height': tf.io.FixedLenFeature((), tf.int64),
    'width': tf.io.FixedLenFeature((), tf.int64),
    'img1': tf.io.FixedLenFeature((), tf.string),
    'img2': tf.io.FixedLenFeature((), tf.string),
    #'flow': tf.io.FixedLenFeature((), tf.string),
    'flow': tf.io.FixedLenFeature((393216), tf.float32),
    }

def _parse_record(proto):
    return tf.io.parse_single_example(proto, feature)

data = tf.data.TFRecordDataset(path, compression_type='ZLIB')

y = []

for x in  data.take(1):
    y = _parse_record(x)
    print(y.keys())

    img1 = tf.io.parse_tensor(y['img1'], tf.uint8).numpy()
    plt.imshow(img1.reshape(384,512,3))
    plt.show()
    img2 = tf.io.parse_tensor(y['img2'], tf.uint8).numpy()
    plt.imshow(img2.reshape(384,512,3))
    plt.show()
    print(img1.shape, img2.shape)

    flow = np.frombuffer(y['flow'].numpy(), dtype=np.float32)
    print(flow.shape)
    plt.imshow(fz.convert_from_flow(flow.reshape(384,512,2)))
    plt.show()
```

7 Flow File Visualizer

```
import glob
import flowiz as fz
import matplotlib.pyplot as plt
import IO

# Demo flo file viewer

path = 'FlyingChairs_release/tfrecord/fc_train.tfrecords'

num = 1000

img1 = sorted(glob.glob('FlyingChairs_release/data/*img1.ppm'))
img2 = sorted(glob.glob('FlyingChairs_release/data/*img2.ppm'))
flow = sorted(glob.glob('FlyingChairs_release/data/*.flo'))

plt.figure(1)
plt.subplot(1,3,1)
plt.imshow(plt.imread(img1[num]))
plt.subplot(1,3,2)
plt.imshow(plt.imread(img2[num]))

plt.subplot(1,3,3)
flow_img = fz.convert_from_file(flow[num])
plt.imshow(flow_img)
plt.show()
```

8 Converted Test Model

```
import keras
from keras.models import Model
from keras import layers
import keras.backend as K
import numpy as np
from keras.layers.core import Lambda
import tensorflow as tf

# This code was automatically generated by MMDNN from FlowNet Caffe model
# modified to load weights, otherwise non-functional

weights_dict = dict()
def load_weights_from_file(weight_file):
    try:
        weights_dict = np.load(weight_file, allow_pickle=True).item()
    except:
        weights_dict = np.load(weight_file, allow_pickle=True, encoding='bytes
            ').item()

    return weights_dict


def set_layer_weights(model, weights_dict):
    for layer in model.layers:
        if layer.name in weights_dict:
            print(layer.name)
            cur_dict = weights_dict[layer.name]
            current_layer_parameters = list()
            if layer.__class__.__name__ == "BatchNormalization":
                if 'scale' in cur_dict:
                    current_layer_parameters.append(cur_dict['scale'])
                if 'bias' in cur_dict:
                    current_layer_parameters.append(cur_dict['bias'])
                current_layer_parameters.extend([cur_dict['mean'], cur_dict['
                    var']])
            elif layer.__class__.__name__ == "Scale":
                if 'scale' in cur_dict:
                    current_layer_parameters.append(cur_dict['scale'])
                if 'bias' in cur_dict:
                    current_layer_parameters.append(cur_dict['bias'])
            elif layer.__class__.__name__ == "SeparableConv2D":
                current_layer_parameters = [cur_dict['depthwise_filter'],
                    cur_dict['pointwise_filter']]
                if 'bias' in cur_dict:
                    current_layer_parameters.append(cur_dict['bias'])
            elif layer.__class__.__name__ == "Embedding":
                current_layer_parameters.append(cur_dict['weights'])
            elif layer.__class__.__name__ == "PReLU":
                gamma =  np.ones(list(layer.input_shape[1:]))*cur_dict['gamma'
```

```
                ]
                current_layer_parameters.append(gamma)
            else:
                # rot
                if 'weights' in cur_dict:
                    current_layer_parameters = [cur_dict['weights']]
                if 'bias' in cur_dict:
                    current_layer_parameters.append(np.squeeze(cur_dict['bias'
                        ]))
            model.get_layer(layer.name).set_weights(current_layer_parameters)
    return model


def KitModel(weight_file = None):
    global weights_dict
    weights_dict = load_weights_from_file(weight_file) if not weight_file ==
        None else None

    data            = layers.Input(name = 'data', shape = (384, 512, 6,) )
    conv1_input     = layers.ZeroPadding2D(padding = ((3, 3), (3, 3)))(data)
    conv1           = convolution(weights_dict, name='conv1', input=
        conv1_input, group=1, conv_type='layers.Conv2D', filters=64,
        kernel_size=(7, 7), strides=(2, 2), dilation_rate=(1, 1), padding='
        valid', use_bias=True)
    ReLU1           = layers.Activation(name='ReLU1', activation='relu')(conv1
        )
    conv2_input     = layers.ZeroPadding2D(padding = ((2, 2), (2, 2)))(ReLU1)
    conv2           = convolution(weights_dict, name='conv2', input=
        conv2_input, group=1, conv_type='layers.Conv2D', filters=128,
        kernel_size=(5, 5), strides=(2, 2), dilation_rate=(1, 1), padding='
        valid', use_bias=True)
    ReLU2           = layers.Activation(name='ReLU2', activation='relu')(conv2
        )
    conv3_input     = layers.ZeroPadding2D(padding = ((2, 2), (2, 2)))(ReLU2)
    conv3           = convolution(weights_dict, name='conv3', input=
        conv3_input, group=1, conv_type='layers.Conv2D', filters=256,
        kernel_size=(5, 5), strides=(2, 2), dilation_rate=(1, 1), padding='
        valid', use_bias=True)
    ReLU3           = layers.Activation(name='ReLU3', activation='relu')(conv3
        )
    conv3_1_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU3)
    conv3_1         = convolution(weights_dict, name='conv3_1', input=
        conv3_1_input, group=1, conv_type='layers.Conv2D', filters=256,
        kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
        valid', use_bias=True)
    ReLU4           = layers.Activation(name='ReLU4', activation='relu')(
        conv3_1)
    conv4_input     = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU4)
    conv4           = convolution(weights_dict, name='conv4', input=
        conv4_input, group=1, conv_type='layers.Conv2D', filters=512,
        kernel_size=(3, 3), strides=(2, 2), dilation_rate=(1, 1), padding='
```

```
    valid', use_bias=True)
ReLU5           = layers.Activation(name='ReLU5', activation='relu')(conv4
    )
conv4_1_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU5)
conv4_1         = convolution(weights_dict, name='conv4_1', input=
    conv4_1_input, group=1, conv_type='layers.Conv2D', filters=512,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
ReLU6           = layers.Activation(name='ReLU6', activation='relu')(
    conv4_1)
conv5_input     = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU6)
conv5           = convolution(weights_dict, name='conv5', input=
    conv5_input, group=1, conv_type='layers.Conv2D', filters=512,
    kernel_size=(3, 3), strides=(2, 2), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
ReLU7           = layers.Activation(name='ReLU7', activation='relu')(conv5
    )
conv5_1_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU7)
conv5_1         = convolution(weights_dict, name='conv5_1', input=
    conv5_1_input, group=1, conv_type='layers.Conv2D', filters=512,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
ReLU8           = layers.Activation(name='ReLU8', activation='relu')(
    conv5_1)
conv6_input     = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU8)
conv6           = convolution(weights_dict, name='conv6', input=
    conv6_input, group=1, conv_type='layers.Conv2D', filters=1024,
    kernel_size=(3, 3), strides=(2, 2), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
ReLU9           = layers.Activation(name='ReLU9', activation='relu')(conv6
    )
conv6_1_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU9)
conv6_1         = convolution(weights_dict, name='conv6_1', input=
    conv6_1_input, group=1, conv_type='layers.Conv2D', filters=1024,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
ReLU10          = layers.Activation(name='ReLU10', activation='relu')(
    conv6_1)
Convolution1_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    ReLU10)
Convolution1    = convolution(weights_dict, name='Convolution1', input=
    Convolution1_input, group=1, conv_type='layers.Conv2D', filters=2,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
#deconv5_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(ReLU10
    )
deconv5_input   = layers.ZeroPadding2D(padding = (0))(ReLU10)
deconv5         = convolution(weights_dict, name='deconv5', input=
    deconv5_input, group=1, conv_type='layers.Conv2DTranspose', filters
    =512, kernel_size=(4, 4), strides=(2, 2), dilation_rate=(1, 1),
    padding='same', use_bias=True)
```

```
#upsample_flow6to5_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1))
    )(Convolution1)
upsample_flow6to5_input = layers.ZeroPadding2D(padding = (0))(Convolution1
    )
upsample_flow6to5 = convolution(weights_dict, name='upsample_flow6to5',
    input=upsample_flow6to5_input, group=1, conv_type='layers.
    Conv2DTranspose', filters=2, kernel_size=(4, 4), strides=(2, 2),
    dilation_rate=(1, 1), padding='same', use_bias=True)
ReLU11          = layers.Activation(name='ReLU11', activation='relu')(
    deconv5)

Concat2         = layers.concatenate(name = 'Concat2', inputs = [ReLU8,
    ReLU11, upsample_flow6to5])
Convolution2_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat2)
Convolution2    = convolution(weights_dict, name='Convolution2', input=
    Convolution2_input, group=1, conv_type='layers.Conv2D', filters=2,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    valid', use_bias=True)
#deconv4_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat2)
deconv4_input   = layers.ZeroPadding2D(padding = (0))(Concat2)
deconv4         = convolution(weights_dict, name='deconv4', input=
    deconv4_input, group=1, conv_type='layers.Conv2DTranspose', filters
    =256, kernel_size=(4, 4), strides=(2, 2), dilation_rate=(1, 1),
    padding='same', use_bias=True)
#upsample_flow5to4_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1))
    )(Convolution2)
upsample_flow5to4_input = layers.ZeroPadding2D(padding = (0))(Convolution2
    )
upsample_flow5to4 = convolution(weights_dict, name='upsample_flow5to4',
    input=upsample_flow5to4_input, group=1, conv_type='layers.
    Conv2DTranspose', filters=2, kernel_size=(4, 4), strides=(2, 2),
    dilation_rate=(1, 1), padding='same', use_bias=True)
ReLU12          = layers.Activation(name='ReLU12', activation='relu')(
    deconv4)
Concat3         = layers.concatenate(name = 'Concat3', inputs = [ReLU6,
    ReLU12, upsample_flow5to4])
#Convolution3_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat3)
Convolution3_input = layers.ZeroPadding2D(padding = (0))(Concat3)
Convolution3    = convolution(weights_dict, name='Convolution3', input=
    Convolution3_input, group=1, conv_type='layers.Conv2D', filters=2,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    same', use_bias=True)
#deconv3_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat3)
deconv3_input   = layers.ZeroPadding2D(padding = (0))(Concat3)
deconv3         = convolution(weights_dict, name='deconv3', input=
    deconv3_input, group=1, conv_type='layers.Conv2DTranspose', filters
    =128, kernel_size=(4, 4), strides=(2, 2), dilation_rate=(1, 1),
```

```
    padding='same', use_bias=True)
#upsample_flow4to3_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1))
    )(Convolution3)
upsample_flow4to3_input = layers.ZeroPadding2D(padding = (0))(Convolution3
    )
upsample_flow4to3 = convolution(weights_dict, name='upsample_flow4to3',
    input=upsample_flow4to3_input, group=1, conv_type='layers.
    Conv2DTranspose', filters=2, kernel_size=(4, 4), strides=(2, 2),
    dilation_rate=(1, 1), padding='same', use_bias=True)
ReLU13          = layers.Activation(name='ReLU13', activation='relu')(
    deconv3)
Concat4         = layers.concatenate(name = 'Concat4', inputs = [ReLU4,
    ReLU13, upsample_flow4to3])
#Convolution4_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat4)
Convolution4_input = layers.ZeroPadding2D(padding = (0))(Concat4)
Convolution4    = convolution(weights_dict, name='Convolution4', input=
    Convolution4_input, group=1, conv_type='layers.Conv2D', filters=2,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    same', use_bias=True)

#deconv2_input   = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat4)
deconv2_input   = layers.ZeroPadding2D(padding = (0))(Concat4)
deconv2         = convolution(weights_dict, name='deconv2', input=
    deconv2_input, group=1, conv_type='layers.Conv2DTranspose', filters
    =64, kernel_size=(4, 4), strides=(2, 2), dilation_rate=(1, 1), padding
    ='same', use_bias=True)
#upsample_flow3to2_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1))
    )(Convolution4)
upsample_flow3to2_input = layers.ZeroPadding2D(padding = (0))(Convolution4
    )
upsample_flow3to2 = convolution(weights_dict, name='upsample_flow3to2',
    input=upsample_flow3to2_input, group=1, conv_type='layers.
    Conv2DTranspose', filters=2, kernel_size=(4, 4), strides=(2, 2),
    dilation_rate=(1, 1), padding='same', use_bias=True)
ReLU14          = layers.Activation(name='ReLU14', activation='relu')(
    deconv2)
Concat5         = layers.concatenate(name = 'Concat5', inputs = [ReLU2,
    ReLU14, upsample_flow3to2])
#Convolution5_input = layers.ZeroPadding2D(padding = ((1, 1), (1, 1)))(
    Concat5)
Convolution5_input = layers.ZeroPadding2D(padding = (0))(Concat5)
Convolution5    = convolution(weights_dict, name='Convolution5', input=
    Convolution5_input, group=1, conv_type='layers.Conv2D', filters=2,
    kernel_size=(3, 3), strides=(1, 1), dilation_rate=(1, 1), padding='
    same', use_bias=True)
#Eltwise4        = my_add()([Convolution5, None])
Eltwise4 = Convolution5 + 20
model           = Model(inputs = [data], outputs = [Eltwise4])
#model           = Model(inputs = [data], outputs = [Convolution5])
```

```
    if weights_dict != None:
        set_layer_weights(model, weights_dict)
    return model

class my_add(keras.layers.Layer):
    def __init__(self, **kwargs):
        super(my_add, self).__init__(**kwargs)
    def call(self, inputs):
        res = inputs[0] + inputs[1]
        self.output_shapes = K.int_shape(res)
        return res

    def compute_output_shape(self, input_shape):
        return self.output_shapes


def convolution(weights_dict, name, input, group, conv_type, filters=None, **
    kwargs):
    if not conv_type.startswith('layer'):
        layer = keras.applications.mobilenet.DepthwiseConv2D(name=name, **
            kwargs)(input)
        return layer
    elif conv_type == 'layers.DepthwiseConv2D':
        layer = layers.DepthwiseConv2D(name=name, **kwargs)(input)
        return layer

    inp_filters = K.int_shape(input)[-1]
    inp_grouped_channels = int(inp_filters / group)
    out_grouped_channels = int(filters / group)
    group_list = []
    if group == 1:
        func = getattr(layers, conv_type.split('.')[-1])
        layer = func(name = name, filters = filters, **kwargs)(input)
        return layer
    weight_groups = list()
    if not weights_dict == None:
        w = np.array(weights_dict[name]['weights'])
        weight_groups = np.split(w, indices_or_sections=group, axis=-1)
    for c in range(group):
        x = layers.Lambda(lambda z: z[..., c * inp_grouped_channels:(c + 1) *
            inp_grouped_channels])(input)
        x = layers.Conv2D(name=name + "_" + str(c), filters=
            out_grouped_channels, **kwargs)(x)
        weights_dict[name + "_" + str(c)] = dict()
        weights_dict[name + "_" + str(c)]['weights'] = weight_groups[c]
        group_list.append(x)
    layer = layers.concatenate(group_list, axis = -1)
    if 'bias' in weights_dict[name]:
        b = K.variable(weights_dict[name]['bias'], name = name + "_bias")
        layer = layer + b
    return layer
```

```
if __name__ == '__main__':
    #load_weights_from_file('50351a4ff04c469e8db58de103982ac7.pb')
    mymodel = KitModel('checkpoints/trained_weights.npy')
    mymodel.summary()
```