Classify laterality (left or right sided knee) of the OAI AKOA knee data set

Name： Chen Xu

```
Student ID ： 45474563
```


In this project, I will use a variant VGG model to train and predict the OAI Acelerated Osteoarthritis knee data set. The model contains two VGG blocks and one output block. The final result showed that the accuracy of the model was as high as 0.99%.

In [1]:

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Select the Runtime > "Change runtime type" menu to enable a GPU acceler
ator, ')
  print('and then re-execute this cell.')
else:
  print(gpu_info)
```

```
Sat Nov  7 02:01:04 2020
+---------------------------------------------------------------------
----------+
| NVIDIA-SMI 455.32.00    Driver Version: 418.67       CUDA Version:
10.1     |
|-------------------------------+----------------------+------------
----------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Un
corr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  C
ompute M. |
|                               |                      |
MIG M. |
|===============================+======================+============
==========|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:04.0 Off |
0 |
| N/A   35C    P0    24W / 300W |      0MiB / 16130MiB |      0%
Default |
|                               |                      |
ERR! |
+-------------------------------+----------------------+------------
----------+

+---------------------------------------------------------------------
----------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  G
PU Memory |
|        ID   ID                                                   U
sage      |
|=====================================================================
==========|
|  No running processes found
|
+---------------------------------------------------------------------
----------+
```

First of all, the running environment is based on Google's Colab, the graphics card used is Tesla V100 and has 16GB of video memory, which greatly improves the training speed of the model.

Setup

Import the necessary runtime files, including path recognition and drawing and tensorflow libraries.

In [2]:

```
from pathlib import Path
from zipfile import ZipFile
from matplotlib import pyplot as plt
import IPython.display as display
import random
import tensorflow as tf
```

Data processing

In this section, the data will be sorted and stored in the left and right folders from the zip file.

OAI Acelerated Osteoarthritis knee data set

OAI Acelerated Osteoarthritis knee data set (18K images) - This is part of the Osteoarthritis Initiative and comes with only labelled laterality (left/right knee labelling) in the filename. The preprocessed version of this data set can be found on the course Blackboard site (under Course Help/Resources).

In [3]:

```
path = '/content/drive/My Drive/colab/P4/AKOA_Analysis.zip'
raw_file = ZipFile(path)
raw_list = raw_file.namelist()
right_list = filter(lambda x: 'right' in x.lower() or 'r_i_g_h_t' in x.lower() ,
raw_list)
left_list = filter(lambda x: 'left' in x.lower() or 'l_e_f_t' in x.lower() ,raw_
list)

for vowel in right_list:
  raw_file.extract(vowel, 'data/right')
for vowel in left_list:
  raw_file.extract(vowel, 'data/left')
```

After finishing the data cleaning, use the following code to show the total number of pictures.

In [4]:

```
def check_photo(data_root):
  data_root = Path(data_root)
  all_image_paths = list(data_root.glob('*/*'))
  all_image_paths = [str(path) for path in all_image_paths]
  random.shuffle(all_image_paths)

  image_count = len(all_image_paths)
  print(image_count)
print('Total right photo has:')
check_photo('data/right')
print('Total left photo has:',)
check_photo('data/left')
```

```
Total right photo has:
10920
Total left photo has:
7760
```

Extract one of the pictures to get an intuitive impression of the data.

In [5]:

```
image_path = '/content/data/left/AKOA_Analysis/OAI9036287_BaseLine_3_de3d1_SAG_3
D_DESS_WE_Left.nii.gz_0.png'
display.display(display.Image(image_path))
```

Create a dataset

In the previous section, the data and were cleaned and classified into two folders on the left and right. In this chapter, I built two classes and some functions to generate tf data streams. The final output is training set, validation set and test set.

Define some parameters for the loader:

In [6]:

```
batch_size = 32
img_height = 224
img_width = 224
```

In [7]:

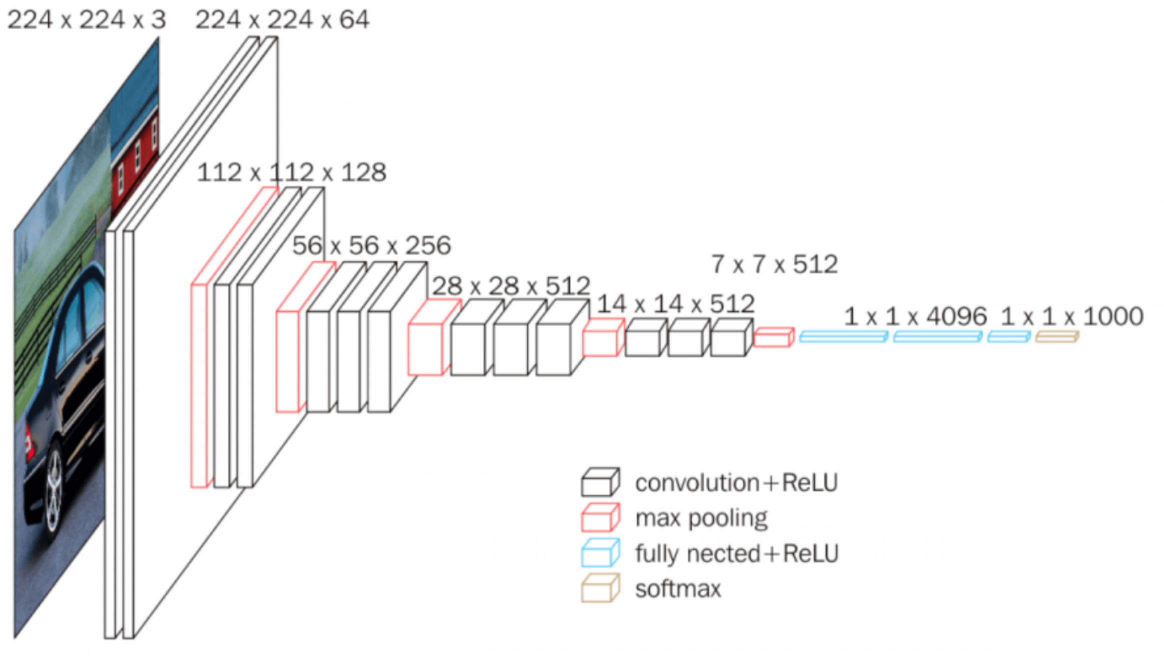
```
AUTOTUNE = tf.data.experimental.AUTOTUNE


class Img_dir:
    def __init__(self, path_list):
        self.path_dir = tf.data.Dataset.from_tensor_slices(path_list)

    def switch_dir(self):
        batch_size = round(len(self.path_dir) / 3)
        path_dir = self.path_dir.shuffle(len(self.path_dir))
        return path_dir.batch(batch_size)


class TotalDataset:
    def __init__(self, left_dir, right_dir):
        self.left_path_dir = [ds for ds in Img_dir(left_dir).switch_dir()]
        self.right_path_dir = [ds for ds in Img_dir(right_dir).switch_dir()]

    def get_path_label(self, index):
        left_xs = self.left_path_dir[index]
        left_ys = tf.zeros(len(left_xs), dtype=tf.int64)

        right_xs = self.right_path_dir[index]
        right_ys = tf.ones(len(right_xs), dtype=tf.int64)

        xs = tf.concat((left_xs, right_xs), axis=0)
        ys = tf.concat((left_ys, right_ys), axis=0)

        ys = tf.one_hot(ys, depth=2)

        img_paths = tf.data.Dataset.from_tensor_slices(xs)

        image_dir = img_paths.map(load_and_preprocess_image)
        label_dir = tf.data.Dataset.from_tensor_slices(ys)
        image_label_dir = tf.data.Dataset.zip((image_dir, label_dir))
        return image_label_dir

    def shuffle_buffer(self, image_label_dir, batch_size):
        # 设置一个和数据集大小一致的 shuffle buffer size（随机缓冲区大小）以保证数据
        # 被充分打乱。
        n = len(image_label_dir)
        ds = image_label_dir.shuffle(buffer_size=n)
        # ds = ds.repeat()
        ds = ds.batch(batch_size)
        # 当模型在训练的时候，`prefetch` 使数据集在后台取得 batch。
        ds = ds.prefetch(buffer_size=AUTOTUNE)
        return ds

    def get_train(self, batch_size):
        train_set = self.get_path_label(0)
        return self.shuffle_buffer(train_set, batch_size)

    def get_val(self, batch_size):
        val_set = self.get_path_label(1)
        return self.shuffle_buffer(val_set, batch_size)

    def get_test(self, batch_size):
        test_set = self.get_path_label(2)
        return self.shuffle_buffer(test_set, batch_size)
```

```python
def preprocess_image(image, img_width=224, img_height=224):
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [img_width, img_height])
    image /= 255.0  # normalize to [0,1] range
    return image


def load_and_preprocess_image(path):
    image = tf.io.read_file(path)
    return preprocess_image(image)

def get_dir(keyword):
    dir = [path.as_posix()
            for path in Path(data_root).rglob('**/'+keyword+'/**/*png')]

def get_dataset(data_root = 'data', batch_size = 32):
    left_dir = [path.as_posix()
                for path in Path(data_root).rglob('**/left/**/*png')]
    right_dir = [path.as_posix()
                 for path in Path(data_root).rglob('**/right/**/*png')]
    data_set = TotalDataset(left_dir, right_dir)

    trainset = data_set.get_train(batch_size)
    valset = data_set.get_val(batch_size)
    testset = data_set.get_test(batch_size)
    return trainset, valset, testset
```

Model building

In this chapter, I constructed a simplified version of VGG that can provide faster training and prediction speeds with the same accuracy.

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman of Oxford University in the paper "Very Deep Convolutional Networks for Large-scale Image Recognition". It is one of the famous models submitted to ILSVRC-2014. By replacing the large kernel size filters (11 and 5 in the first and second convolutional layers respectively) with multiple 3×3 kernel size filters,



In [8]:

```
trainset, valset, testset = get_dataset(data_root = 'data/')
```

In [15]:

```
model = tf.keras.models.Sequential()

# 1 VGG Blocks
model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3, activation='relu', pa
dding='same', input_shape=(224, 224, 3)))
model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=2,activation='relu'))
# model.add(tf.keras.layers.MaxPooling2D((2, 2), strides=2, padding='same'))
model.add(tf.keras.layers.Dropout(0.2))

# 2 VGG Blocks
model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3, activation='relu', pa
dding='same', input_shape=(224, 224, 3)))
model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=2,activation='relu'))
# model.add(tf.keras.layers.MaxPooling2D((2, 2), strides=2, padding='same'))
model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.7))
model.add(tf.keras.layers.Dense(2, activation='softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 224, 224, 64)      1792      
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 223, 223, 64)      16448     
_________________________________________________________________
dropout_2 (Dropout)          (None, 223, 223, 64)      0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 3182656)           0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 3182656)           0         
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 6365314   
=================================================================
Total params: 6,383,554
Trainable params: 6,383,554
Non-trainable params: 0
_________________________________________________________________
```

The input of the cov1 layer has a fixed size 224 x 224 image. The image passes through a stack of convolution (convolution) layers. In one configuration, it also uses 64 convolution filters). The convolution stride is fixed at 1 pixel; the converted space filling layer input preserves the spatial resolution after convolution. All hidden layers have rectification (ReLU) nonlinear characteristics. This structure will improve the performance of the ILSVRC data set, and also save memory consumption and calculation time.

In [26]:

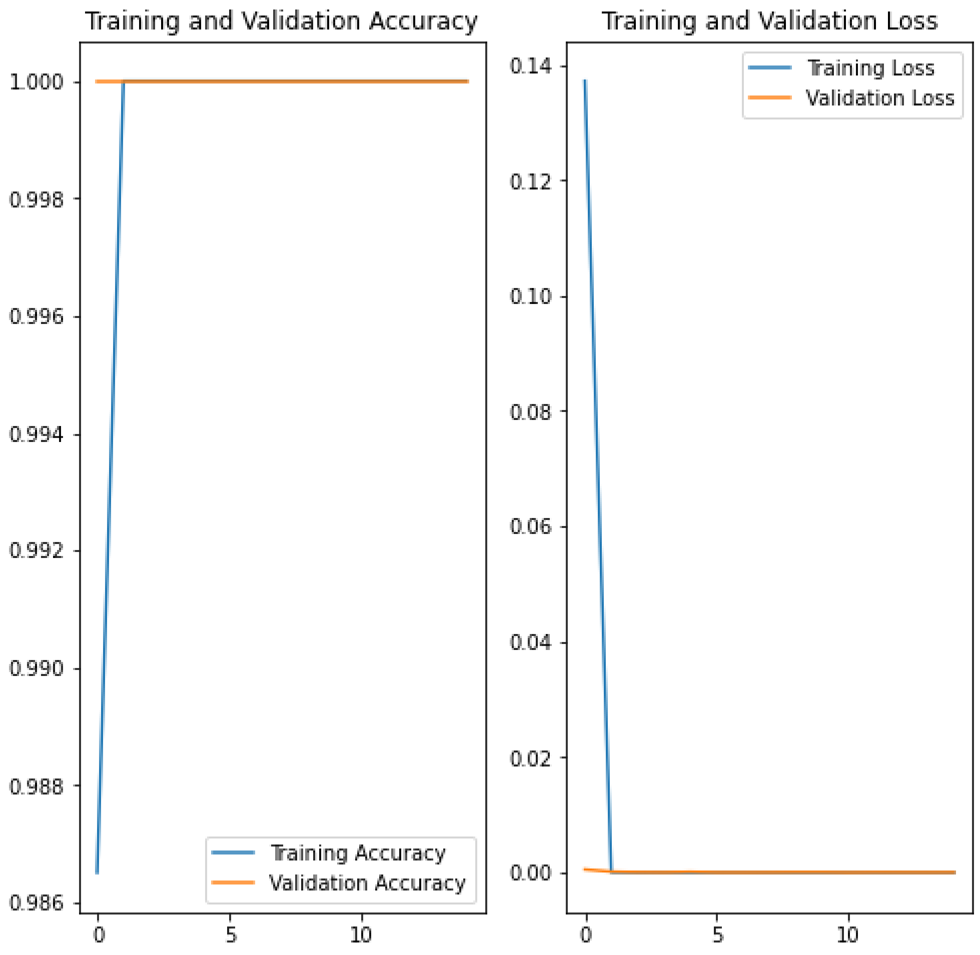
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    trainset,
    epochs=15,
    validation_data=valset,
    # validation_steps=10
)
```

```
Epoch 1/15
195/195 [==============================] - 47s 239ms/step - loss: 0.
1372 - accuracy: 0.9865 - val_loss: 5.4412e-04 - val_accuracy: 1.000
0
Epoch 2/15
195/195 [==============================] - 46s 238ms/step - loss: 4.
9412e-05 - accuracy: 1.0000 - val_loss: 1.5971e-04 - val_accuracy:
1.0000
Epoch 3/15
195/195 [==============================] - 46s 236ms/step - loss: 1.
5178e-05 - accuracy: 1.0000 - val_loss: 6.8459e-05 - val_accuracy:
1.0000
Epoch 4/15
195/195 [==============================] - 47s 241ms/step - loss: 5.
5823e-06 - accuracy: 1.0000 - val_loss: 3.2114e-05 - val_accuracy:
1.0000
Epoch 5/15
195/195 [==============================] - 47s 239ms/step - loss: 4.
1083e-06 - accuracy: 1.0000 - val_loss: 1.3409e-04 - val_accuracy:
1.0000
Epoch 6/15
195/195 [==============================] - 47s 240ms/step - loss: 4.
3576e-06 - accuracy: 1.0000 - val_loss: 2.9248e-05 - val_accuracy:
1.0000
Epoch 7/15
195/195 [==============================] - 46s 238ms/step - loss: 1.
3120e-06 - accuracy: 1.0000 - val_loss: 2.1438e-05 - val_accuracy:
1.0000
Epoch 8/15
195/195 [==============================] - 46s 235ms/step - loss: 1.
6847e-06 - accuracy: 1.0000 - val_loss: 3.4339e-05 - val_accuracy:
1.0000
Epoch 9/15
195/195 [==============================] - 41s 210ms/step - loss: 1.
3197e-06 - accuracy: 1.0000 - val_loss: 6.3285e-05 - val_accuracy:
1.0000
Epoch 10/15
195/195 [==============================] - 41s 208ms/step - loss: 1.
0510e-06 - accuracy: 1.0000 - val_loss: 6.2886e-05 - val_accuracy:
1.0000
Epoch 11/15
195/195 [==============================] - 38s 197ms/step - loss: 6.
9197e-07 - accuracy: 1.0000 - val_loss: 1.2744e-05 - val_accuracy:
1.0000
Epoch 12/15
195/195 [==============================] - 38s 197ms/step - loss: 4.
9604e-06 - accuracy: 1.0000 - val_loss: 4.7944e-05 - val_accuracy:
1.0000
Epoch 13/15
195/195 [==============================] - 39s 198ms/step - loss: 2.
0919e-06 - accuracy: 1.0000 - val_loss: 3.8072e-05 - val_accuracy:
1.0000
Epoch 14/15
195/195 [==============================] - 39s 198ms/step - loss: 8.
1314e-07 - accuracy: 1.0000 - val_loss: 3.0530e-05 - val_accuracy:
1.0000
Epoch 15/15
195/195 [==============================] - 39s 198ms/step - loss: 6.
6851e-07 - accuracy: 1.0000 - val_loss: 4.2685e-05 - val_accuracy:
1.0000
```

Evaluation and prediction

In [30]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(15)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



The Test accuracy is：

In [31]:

```
loss, accuracy = model.evaluate(testset)
```

```
195/195 [==============================] - 3s 14ms/step - loss: 2.49
36e-05 - accuracy: 1.0000
```

Conclusion

From the above training structure, it can be found that the variant VGG model has a high accuracy (infinitely close to one) for the classification of Classify laterality (left or right sided knee) of the OAI AKOA knee data set. Performance in the test set and validation set Consistently, no overfitting occurred. However, like VGGNet, it has two main disadvantages: despite the optimization and streamlining of the model, the training time is still very slow, and the network architecture weight itself is very large (with regard to disk/bandwidth).