

Deep Learning Workshop Using Python and Jupyter Notebook

Magnetic Resonance Journal Club (MRJC)

Dr. Alexander Weber

Assistant Professor, Department of Pediatrics,
Division of Neurology, Faculty of Medicine
Imaging Staff Scientist, BC Children's Hospital Research Institute
University of British Columbia

February 25, 2020

Introduction

Today we will be applying several concepts that we learned in the last journal club meeting, such as:

- Machine Learning
- Convolutional Neural Networks
- Medical image classification

In order to do so, we will be running a tutorial that uses:

- Jupyter Notebook
- Python programming language
- TensorFlow - currently the most actively used machine learning library and framework (Tensorflow 1.4, Google LLC, Mountain View, CA)
- Keras Library (Keras v 2.12, <http://keras.io/>), which is a high-level application programming interface that simplifies working with Tensorflow

Introduction

Today we will be applying several concepts that we learned in the last journal club meeting, such as:

- Machine Learning
- Convolutional Neural Networks
- Medical image classification

In order to do so, we will be running a tutorial that uses:

- Jupyter Notebook
- Python programming language
- TensorFlow - currently the most actively used machine learning library and framework (Tensorflow 1.4, Google LLC, Mountain View, CA)
- Keras Library (Keras v 2.12, <http://keras.io/>), which is a high-level application programming interface that simplifies working with Tensorflow

TensorFlow and Keras



- free and open-source software library
- used for machine learning applications such as neural networks
- developed by the Google Brain team

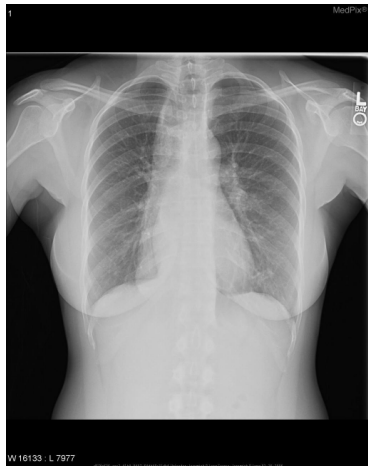


- open-source neural-network library
- runs on top of TensorFlow (or other frameworks)
- designed to enable fast experimentation with deep neural networks, by being user-friendly, modular, and extensible.

Workshop Goal

The goal of this workshop will be to build a Deep Learning neural network in order to help us differentiate and classify two different x-ray images:

- Chest Radiographs



- Abdominal Radiographs



Dataset

- Our dataset consists of 75 images, split roughly in half, with 37 of the abdomen and 38 of the chest
- These images are in PNG (Portable Network Graphics) format, and derive from OpenI, a searchable online repository of medical images from published PubMed Central articles (<https://openi.nlm.nih.gov>)
- For handling DICOM images, a Python library such as PyDicom (<https://pydicom.github.io/pydicom/stable/>) may be used to import and convert the images to be used in TensorFlow



1.4.2

[Docs](#) » pydicom documentation

pydicom documentation

Getting Started

If you're new to *pydicom* then start here:

- [Installation](#) | [Introduction to pydicom](#)
- **Basics:** [Read](#), [access](#), [modify](#), [write](#)

Dataset

- Our dataset consists of 75 images, split roughly in half, with 37 of the abdomen and 38 of the chest
- These images are in PNG (Portable Network Graphics) format, and derive from OpenI, a searchable online repository of medical images from published PubMed Central articles (<https://openi.nlm.nih.gov>)
- For handling DICOM images, a Python library such as PyDicom (<https://pydicom.github.io/pydicom/stable/>) may be used to import and convert the images to be used in TensorFlow



 pydicom

1.4.2

[Docs](#) » pydicom documentation

pydicom documentation

Getting Started

If you're new to *pydicom* then start here:

- [Installation](#) | [Introduction to pydicom](#)
- **Basics:** [Read](#), [access](#), [modify](#), [write](#)

Dataset

- Our dataset consists of 75 images, split roughly in half, with 37 of the abdomen and 38 of the chest
- These images are in PNG (Portable Network Graphics) format, and derive from OpenI, a searchable online repository of medical images from published PubMed Central articles (<https://openi.nlm.nih.gov>)
- For handling DICOM images, a Python library such as PyDicom (<https://pydicom.github.io/pydicom/stable/>) may be used to import and convert the images to be used in TensorFlow



1.4.2

[Docs](#) » pydicom documentation

pydicom documentation

Getting Started

If you're new to *pydicom* then start here:

- [Installation](#) | [Introduction to pydicom](#)
- **Basics:** [Read](#), [access](#), [modify](#), [write](#)

Dataset

- Our first step is to randomly divide our images into training and validation sets
- In this example, our data has already been separated into 65 training cases and 10 validation cases

```
data/  
  train/  
    abd/  
      abd001.png  
      abd002.png  
      ...  
    chst/  
      chst001.png  
      chst002.png  
      ...  
  val/  
    abd/  
      abd_val_001.png  
      abd_val_002.png  
      ...  
    chst/  
      chst_val_001.png  
      chst_val_002.png  
      ...
```

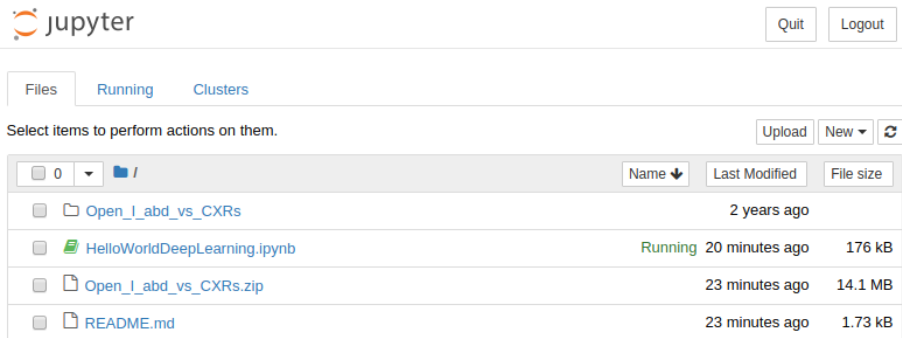
Fig. 1 Directory structure for the data

Jupyter Notebook

Next, let's go ahead and open up Jupyter Notebook

```
weberam2:~$ jupyter notebook &
```

Which will open up a browser and look like this



jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them. Upload New ↕

<input type="checkbox"/> 0 ▾	📁 /	Name ▾	Last Modified	File size
<input type="checkbox"/>	📁	Open_I_abd_vs_CXRs	2 years ago	
<input type="checkbox"/>	📄	HelloWorldDeepLearning.ipynb	Running 20 minutes ago	176 kB
<input type="checkbox"/>	📄	Open_I_abd_vs_CXRs.zip	23 minutes ago	14.1 MB
<input type="checkbox"/>	📄	README.md	23 minutes ago	1.73 kB

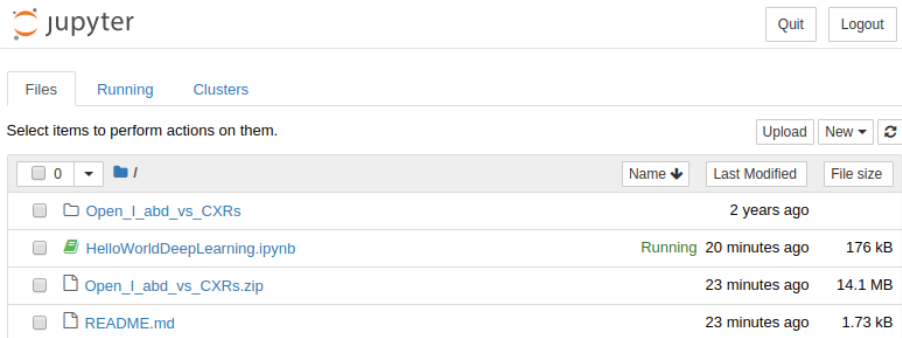
Click on the HelloWorldDeepLearning.ipynb file

Jupyter Notebook

Next, let's go ahead and open up Jupyter Notebook

```
weberam2:~$ jupyter notebook &
```

Which will open up a browser and look like this



Jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them. Upload New ↕

<input type="checkbox"/> 0 ▾	📁 /	Name ▾	Last Modified	File size
<input type="checkbox"/>	📁	Open_I_abd_vs_CXRs	2 years ago	
<input type="checkbox"/>	📄	HelloWorldDeepLearning.ipynb	Running 20 minutes ago	176 kB
<input type="checkbox"/>	📄	Open_I_abd_vs_CXRs.zip	23 minutes ago	14.1 MB
<input type="checkbox"/>	📄	README.md	23 minutes ago	1.73 kB

Click on the HelloWorldDeepLearning.ipynb file

Importing Keras

For the first cell, I had an error:

```
In [1]: #! ##This notebook is built around using tensorflow as the backend for keras  
      !KERAS_BACKEND=tensorflow python -c "from keras import backend"  
  
Traceback (most recent call last):  
  File "<string>", line 1, in <module>  
ImportError: No module named keras
```

This is likely to do with how tensorflow and keras were installed on my machine. Instead, I ran the following:

```
In [27]: #! from tensorflow.python import keras
```

From the tensorflow library, we are importing keras.

Then, from keras, we will import more functions that we plan to use:

```
In [28]: #! from keras import applications  
      from keras.preprocessing.image import ImageDataGenerator  
      from keras import optimizers  
      from keras.models import Sequential  
      from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D  
      from keras.models import Model  
      from keras.optimizers import Adam
```

Importing Keras

For the first cell, I had an error:

```
In [1]: ! ##This notebook is built around using tensorflow as the backend for keras  
      !KERAS_BACKEND=tensorflow python -c "from keras import backend"  
  
Traceback (most recent call last):  
  File "<string>", line 1, in <module>  
ImportError: No module named keras
```

This is likely to do with how tensorflow and keras were installed on my machine. Instead, I ran the following:

```
In [27]: ! from tensorflow.python import keras
```

From the tensorflow library, we are importing keras.

Then, from keras, we will import more functions that we plan to use:

```
In [28]: ! from keras import applications  
      from keras.preprocessing.image import ImageDataGenerator  
      from keras import optimizers  
      from keras.models import Sequential  
      from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D  
      from keras.models import Model  
      from keras.optimizers import Adam
```

Importing Keras

For the first cell, I had an error:

```
In [1]: #! ##This notebook is built around using tensorflow as the backend for keras  
!KERAS_BACKEND=tensorflow python -c "from keras import backend"  
  
Traceback (most recent call last):  
  File "<string>", line 1, in <module>  
ImportError: No module named keras
```

This is likely to do with how tensorflow and keras were installed on my machine. Instead, I ran the following:

```
In [27]: #! from tensorflow.python import keras
```

From the tensorflow library, we are importing keras.

Then, from keras, we will import more functions that we plan to use:

```
In [28]: #! from keras import applications  
from keras.preprocessing.image import ImageDataGenerator  
from keras import optimizers  
from keras.models import Sequential  
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D  
from keras.models import Model  
from keras.optimizers import Adam
```

Specify Image Information

Next we will specify the image information, the number of 'epochs', and the number of 'batches'

```
# dimensions of our images.
img_width, img_height = 299, 299

train_data_dir = '/home/weberam2/Dropbox/AssistantProf_BCCHRI/Teaching/MRI_JournalClub/Feb25_DeepLearningTutoria
validation_data_dir = '/home/weberam2/Dropbox/AssistantProf_BCCHRI/Teaching/MRI_JournalClub/Feb25_DeepLearningTu

# number of samples used for determining the samples_per_epoch
nb_train_samples = 65
nb_validation_samples = 10
epochs = 20
batch_size = 5
```

An epoch is the number of passes through the training data.
A batch is the number of images processed at the same time.

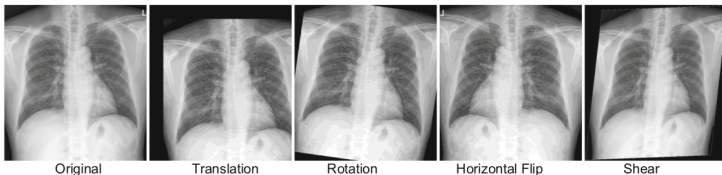
Image Preprocessing and Augmentation

Next we preprocess and specify augmentation options

```
train_datagen = ImageDataGenerator(  
    rescale=1./255,           # normalize pixel values to [0,1]  
    shear_range=0.2,  
    zoom_range=0.2,  
    rotation_range=20,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    horizontal_flip=True)  
  
val_datagen = ImageDataGenerator(  
    rescale=1./255)           # normalize pixel values to [0,1]
```

E.g. shear, zoom, rotation, width and height shift, and horizontal flip.
Help preempt overfitting or “memorization”; leads to greater accuracy and generalization of CCNs.

Example images:



Defining the Training and Validation Generator

Next we give more information, such as training directory, size of images, and batch size:

```
train_generator = train_datagen.flow_from_directory(  
    train_data_dir,  
    target_size=(img_height, img_width),  
    batch_size=batch_size,  
    class_mode='binary')  
  
validation_generator = train_datagen.flow_from_directory(  
    validation_data_dir,  
    target_size=(img_height, img_width),  
    batch_size=batch_size,  
    class_mode='binary')
```

Found 65 images belonging to 2 classes.

Found 10 images belonging to 2 classes.

Build the Model

Next we want to build a pretrained network, in this case called Inception V3

```
In [4]: base_model = applications.InceptionV3(weights='imagenet', include_top=False,  
                                              input_shape=(img_width, img_height, 3))
```

We are removing the top or fully connected layers from the original network as well as using pretrained weights from ImageNet

ImageNet

Formally a project aimed at (manually) labeling and categorizing images into almost 22,000 separate object categories for the purpose of computer vision research (e.g. species of dogs, various household objects, etc.)

Build a Classifier Model

Next we add new layers on top of the original model.

This consists of a global average pooling layer and a fully connected layer with 256 nodes.

We apply dropout and sigmoid activation.

Then we compile a model using Adam optimizer with common values, and a low learning rate (lr) for transfer learning

```
In [5]: model_top = Sequential()
model_top.add(GlobalAveragePooling2D(input_shape=base_model.output_shape[1:], data_format=None)),
model_top.add(Dense(256, activation='relu'))
model_top.add(Dropout(0.5))
model_top.add(Dense(1, activation='sigmoid'))

model = Model(inputs=base_model.input, outputs=model_top(base_model.output))

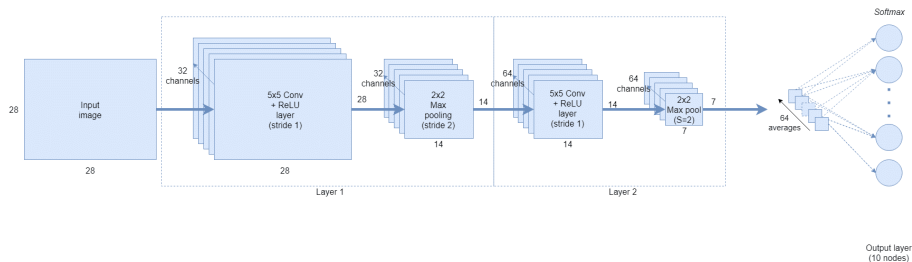
model.compile(optimizer=Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0),
              loss='binary_crossentropy', metrics=['accuracy'])
```

As there are two categories (abdominal or chest radiograph), we compile the model using binary cross-entropy loss

Global Average Pooling Layer

Global Average Pooling Layer

An operation that calculates the average output of each feature map in the previous layer. This fairly simple operation reduces the data significantly and prepares the model for the final classification layer.



Dropout

Dropout Regularization

A technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

If neurons are randomly dropped out of the network during training, other neurons will step in to handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

Network becomes less sensitive to the specific weights of neurons, resulting in better generalization and less likely to overfit training data.

Adam

Adam

An adaptive learning rate method: it computes individual learning rates for different parameters. Its name is derived from adaptive moment estimation: Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.

Combines the advantages of two other extensions of stochastic gradient descent:

- **Adaptive Gradient Algorithm (AdaGrad)** that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
- **Root Mean Square Propagation (RMSProp)** that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Transfer Learning

Transfer Learning

Application of a process suited for one specific task to a different problem

Essentially, all images share similar features, such as edges and blobs, which aids transfer learning.

In this example, we remove the final (top) fully connected layers of the pretrained Inception V3 model that is intended for a 1000-class problem of ImageNet, and insert additional layers with random initialization.

We fine-tune the entire model using a very low learning rate (0.0001), as not to rapidly perturb the weights that are already relatively well optimized.

Entropy Aside

Cross-Entropy

In information theory, the cross entropy between two probability distributions p and q over the same underlying set of events, measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q , rather than the true distribution p .

Cross-Entropy Loss

$$CE = - \sum_i^C p_i \log(q_i)$$

In Machine Learning, cross-entropy can be used to define a loss function, where the true probability p_i is the true label, and the given distribution q_i is the predicted value of the current model.

Binary Cross-Entropy

In a binary classification problem, where $C = 2$, the Cross Entropy Loss can be defined as:

$$CE = - \sum_{i=1}^{C=2} p_i \log(q_i) = -p_1 \log(q_1) - (1 - p_1) \log(1 - q_1)$$

Where it is assumed there are two classes: C_1 and C_2 .

p_1 [0,1] and q_1 are the groundtruth and the score for C_1 , and $p_2 = 1 - p_1$ and $q_2 = 1 - q_1$ are the groundtruth and the score for C_2

Fitting the Model

Finally, we fit the model into the generator:

```
history = model.fit_generator(  
    train_generator,  
    steps_per_epoch=nb_train_samples // batch_size,  
    epochs=epochs,  
    validation_data=validation_generator,  
    validation_steps=nb_validation_samples // batch_size)
```

```
Epoch 1/20  
13/13 [=====] - 68s 5s/step - loss: 0.4510 - accuracy: 0.8154 - val_loss: 0.5501 - v  
al_accuracy: 0.7000  
Epoch 2/20  
13/13 [=====] - 77s 6s/step - loss: 0.1289 - accuracy: 0.9846 - val_loss: 0.3488 - v  
al_accuracy: 0.8000  
Epoch 3/20  
13/13 [=====] - 123s 9s/step - loss: 0.1368 - accuracy: 0.9692 - val_loss: 0.2683 -  
val_accuracy: 0.7000  
Epoch 4/20  
13/13 [=====] - 138s 11s/step - loss: 0.2973 - accuracy: 0.9077 - val_loss: 0.0463 -  
val_accuracy: 0.8000  
Epoch 5/20  
13/13 [=====] - 134s 10s/step - loss: 0.0522 - accuracy: 0.9846 - val_loss: 0.0537 -  
val_accuracy: 0.8000  
Epoch 6/20  
13/13 [=====] - 140s 11s/step - loss: 0.1352 - accuracy: 0.9385 - val_loss: 0.5047 -  
val_accuracy: 0.8000  
Epoch 7/20
```

Plot Training Data

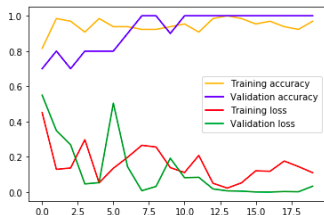
The loss and accuracy values are stored in arrays, and can be plotted using Matplotlib:

```
import matplotlib.pyplot as plt

print(history.history.keys())

plt.figure()
plt.plot(history.history['accuracy'], 'orange', label='Training accuracy')
plt.plot(history.history['val_accuracy'], 'blue', label='Validation accuracy')
plt.plot(history.history['loss'], 'red', label='Training loss')
plt.plot(history.history['val_loss'], 'green', label='Validation loss')
plt.legend()
plt.show()
```

dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])



Note: I needed to change 'acc' and 'val_acc' to 'accuracy' and 'val_accuracy'

Evaluating the Training Model

It is common practice to evaluate the performance of the trained model on independent test-cases

```
import numpy as np
from keras.preprocessing import image

img_path='/home/weberam2/Dropbox/AssistantProf_BCCHRI/Teaching/MRI_JournalClub/Feb25_DeepLearningTutorial/Hello
img_path2='/home/weberam2/Dropbox/AssistantProf_BCCHRI/Teaching/MRI_JournalClub/Feb25_DeepLearningTutorial/Hello
img = image.load_img(img_path, target_size=(img_width, img_height))
img2 = image.load_img(img_path2, target_size=(img_width, img_height))
plt.imshow(img)
plt.show()

img = image.img_to_array(img)
x = np.expand_dims(img, axis=0) * 1./255
score = model.predict(x)
print('Predicted:', score, 'Chest X-ray' if score < 0.5 else 'Abd X-ray')

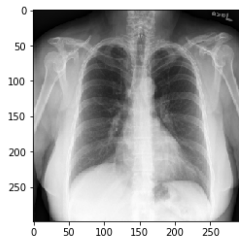
plt.imshow(img2)
plt.show()

img2 = image.img_to_array(img2)
x = np.expand_dims(img2, axis=0) * 1./255
score2 = model.predict(x)
print('Predicted:', score2, 'Chest X-ray' if score2 < 0.5 else 'Abd X-ray')
```

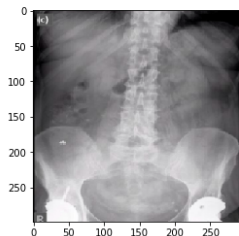
Note: you will have to change the `img_path` and `img_path2` directories to the test cases

Results

Here we have the
predictions of our
two test cases:



Predicted: `[[0.00293497]]` Chest X-ray



Predicted: `[[0.9991743]]` Abd X-ray

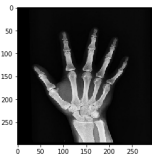
Diving Deeper

I tried testing the model on some random images:

```
In [ ]: img_path3='/home/weberam2/Dropbox/AssistantProf_BCCHRI/Teaching/MRI_JournalClub/Feb25_DeepLearningTu
img3 = image.load_img(img_path3, target_size=(img_width, img_height))

plt.imshow(img3)
plt.show()

img3 = image.img_to_array(img3)
x = np.expand_dims(img3, axis=0) * 1./255
score = model.predict(x)
print('Predicted:', score, 'Chest X-ray' if score < 0.5 else 'Abd X-ray')
```



Predicted: [[0.9847786]] Abd X-ray

```
In [ ]: img_path4='/home/weberam2/Dropbox/AssistantProf_BCCHRI/Teaching/MRI_JournalClub/Feb25_DeepLearningTu
img4 = image.load_img(img_path4, target_size=(img_width, img_height))

plt.imshow(img4)
plt.show()

img4 = image.img_to_array(img4)
x = np.expand_dims(img4, axis=0) * 1./255
score = model.predict(x)
print('Predicted:', score, 'Chest X-ray' if score < 0.5 else 'Abd X-ray')
```

