Fracture detection using Modified and pre-trained VGG19

Note:

We originally developed the algorithm to detect fractures from well log data / seismic discontinuity (seismic contrasts) from 2D and 3D seismic volumes for the geothermal and oil/gas industry. However, given the issue of data confidentiality, we only present the application of the algorithm to a different set of data that is available in the public domain. Furthermore, we only share glimpses of the algorithm (to learn more please contact yesser@sges.cloud)

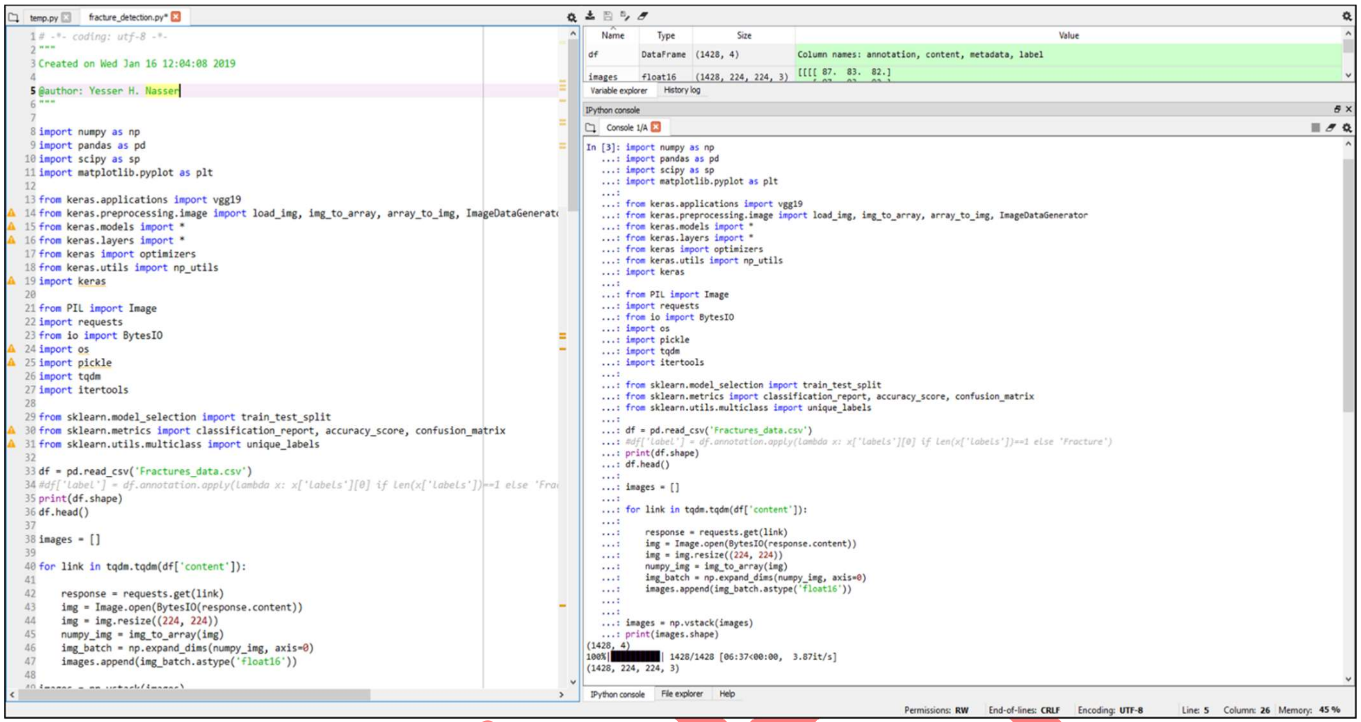
Abstract

Machine learning algorithm and new architecture of Deep learning neural network has proven to be valuable in detecting anomalies and features. Here, we present the application of a state-of-the-art modified and pre-trained VGG19 to identify fractures. In this work we show the application of modified Deep learning neural network that was previously trained on a large data set on a much smaller dataset. This prior learning provides calibrated weights to the hidden layers of the network to be able to detect features and learn from the new dataset. This transfer of learning of feature-detection from the large dataset to this much smaller dataset allowed the network to learn faster and hence be able to detect anomalies/fractures in the images. In this application we reached an accuracy of 87% after only 20 epochs. Furthermore, the computation time to achieve such results with pre-trained weights is considerably faster compare to running the DNN without prior learning.

- **Input Data:**

The input data used for this project consists of images of free-surface fractures/cracks and images of random objects. Additionally, images of objects that could appear like fractures are added to the dataset during the training process to boost the DNN learning process and validate the robustness of the DNN in detecting true fractures.

The dataset consists of 1142 images for training and 286 for testing. Given the architecture, padding (p), strides (s), and number of filters used in this DNN the input data is resized to 224x224x3.



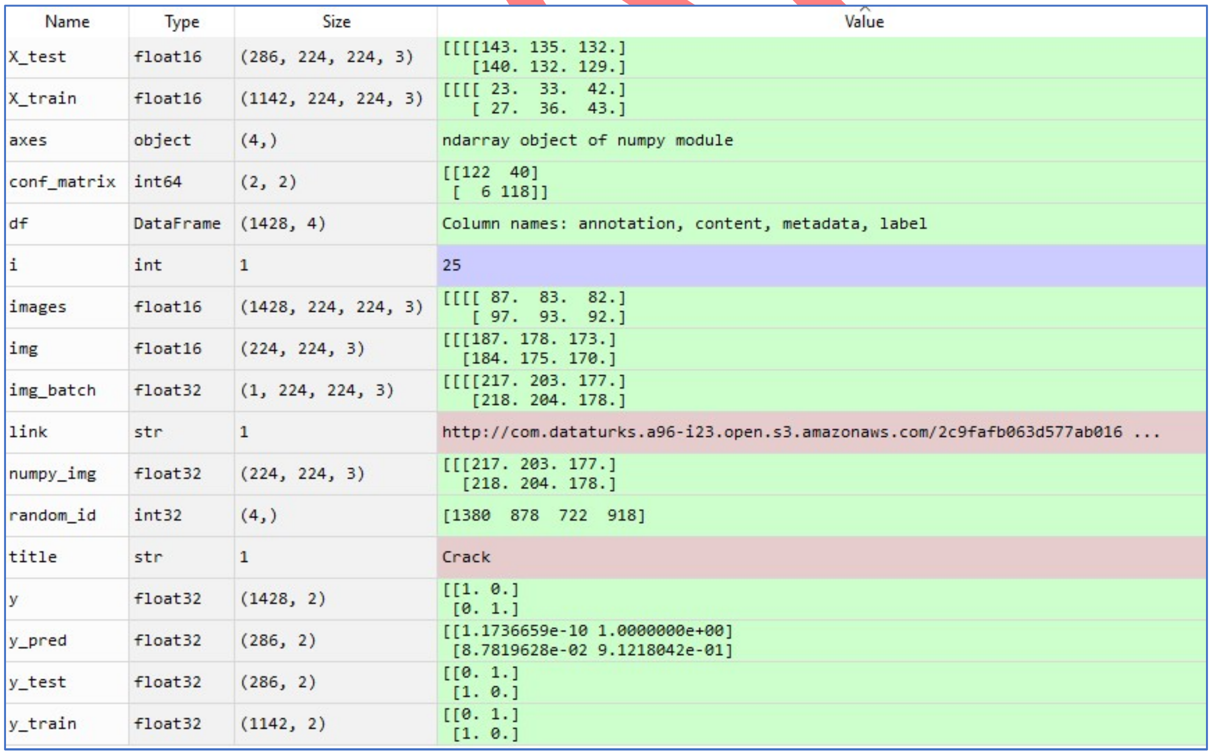
```
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 16 12:04:08 2019

@author: Yesser H. Nasser
"""

import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt

from keras.applications import vgg19
from keras.preprocessing.image import load_img, img_to_array, array_to_img, ImageDataGenerat
from keras.models import *
from keras.layers import *
from keras import optimizers
from keras.utils import np_utils
import keras

from PIL import Image
import requests
from io import BytesIO
import os
import pickle
import tqdm
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.utils.multiclass import unique_labels

df = pd.read_csv('Fractures_data.csv')
#df['label'] = df.annotation.apply(lambda x: x['labels'][0] if len(x['labels'])==1 else 'Fra
print(df.shape)
df.head()

images = []

for link in tqdm.tqdm(df['content']):

    response = requests.get(link)
    img = Image.open(BytesIO(response.content))
    img = img.resize((224, 224))
    numpy_img = img_to_array(img)
    img_batch = np.expand_dims(numpy_img, axis=0)
    images.append(img_batch.astype('float16'))

```

Name	Type	Size	Value
df	DataFrame	(1428, 4)	Column names: annotation, content, metadata, label
images	float16	(1428, 224, 224, 3)	[[[[87. 83. 82.]

```
In [3]: import numpy as np
   ...: import pandas as pd
   ...: import scipy as sp
   ...: import matplotlib.pyplot as plt
   ...:
   ...: from keras.applications import vgg19
   ...: from keras.preprocessing.image import load_img, img_to_array, array_to_img, ImageDataGenerator
   ...: from keras.models import *
   ...: from keras.layers import *
   ...: from keras import optimizers
   ...: from keras.utils import np_utils
   ...: import keras
   ...:
   ...: from PIL import Image
   ...: import requests
   ...: from io import BytesIO
   ...: import os
   ...: import pickle
   ...: import tqdm
   ...: import itertools
   ...:
   ...: from sklearn.model_selection import train_test_split
   ...: from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
   ...: from sklearn.utils.multiclass import unique_labels
   ...:
   ...: df = pd.read_csv('Fractures_data.csv')
   ...: #df['label'] = df.annotation.apply(lambda x: x['labels'][0] if len(x['labels'])==1 else 'Fracture')
   ...: print(df.shape)
   ...: df.head()
   ...:
   ...: images = []
   ...:
   ...: for link in tqdm.tqdm(df['content']):
   ...:
   ...:     response = requests.get(link)
   ...:     img = Image.open(BytesIO(response.content))
   ...:     img = img.resize((224, 224))
   ...:     numpy_img = img_to_array(img)
   ...:     img_batch = np.expand_dims(numpy_img, axis=0)
   ...:     images.append(img_batch.astype('float16'))
   ...:
   ...:
   ...: images = np.vstack(images)
   ...: print(images.shape)
(1428, 4)
100%|██████████| 1428/1428 [06:37<00:00,  3.87it/s]
(1428, 224, 224, 3)
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 5 Column: 26 Memory: 45 %

Figure 1: overview of the algorithm to load the data



Name	Type	Size	Value
X_test	float16	(286, 224, 224, 3)	[[[[143. 135. 132.] [140. 132. 129.]
X_train	float16	(1142, 224, 224, 3)	[[[[23. 33. 42.] [27. 36. 43.]
axes	object	(4,)	ndarray object of numpy module
conf_matrix	int64	(2, 2)	[[122 40] [6 118]]
df	DataFrame	(1428, 4)	Column names: annotation, content, metadata, label
i	int	1	25
images	float16	(1428, 224, 224, 3)	[[[[87. 83. 82.] [97. 93. 92.]
img	float16	(224, 224, 3)	[[[187. 178. 173.] [184. 175. 170.]
img_batch	float32	(1, 224, 224, 3)	[[[[217. 203. 177.] [218. 204. 178.]
link	str	1	http://com.dataturks.a96-i23.open.s3.amazonaws.com/2c9fafb063d577ab016 ...
numpy_img	float32	(224, 224, 3)	[[[217. 203. 177.] [218. 204. 178.]
random_id	int32	(4,)	[1380 878 722 918]
title	str	1	Crack
y	float32	(1428, 2)	[[1. 0.] [0. 1.]
y_pred	float32	(286, 2)	[[1.1736659e-10 1.0000000e+00] [8.7819628e-02 9.1218042e-01]
y_test	float32	(286, 2)	[[0. 1.] [1. 0.]
y_train	float32	(1142, 2)	[[0. 1.] [1. 0.]

Figure 2: Data dimensions

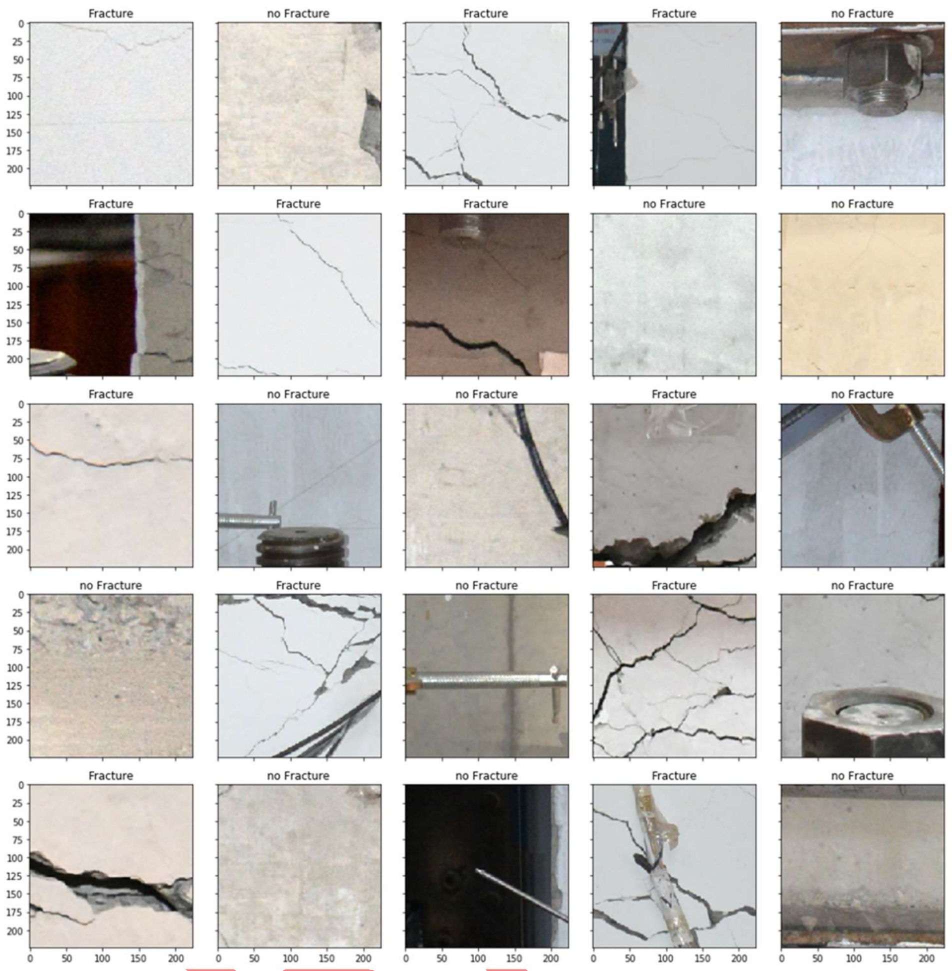
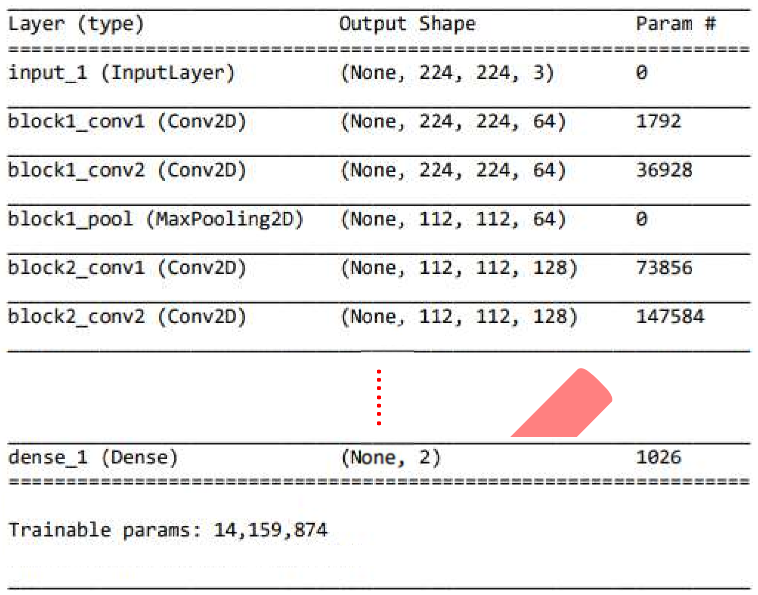


Figure 3: Random display of 25 images from the dataset. The training dataset consist of fractures and random objects.

Figure 2 summarizes the data dimension and structure. Figure 3 shows random display of 25 images of fractures and other random objects. We include 'fracture-like' object to increase the accuracy of the DNN. Given the size of the data we applied data augmentation techniques (horizontal flip, width and height shift, and rotation).

- **DNN - Structure**

In this work we use VGG Neural Network structure with modification and prior training from larger dataset. Here, the prior training is introduced in order to transfer the learning and 'trained-weights' on how to detect features/objects (figure 4). These pre-trained weights are used at early layers of the network. Additionally the network will have 14M new parameters (weights and hyperparameters) to calibrate during the training process. To achieve better accuracy and reduce losses we introduce modifications to the network structure.

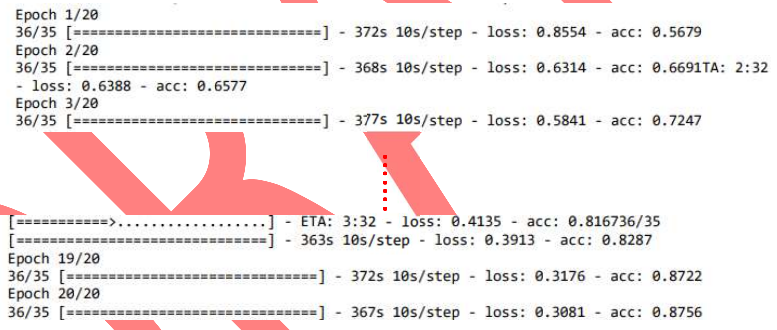


```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
                                   :
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Trainable params: 14,159,874
_________________________________________________________________
```

Figure 4: Overview of the Network architecture, the full details are not shared in this document.

- **Training**

To evaluate the training process, we use ‘categorical cross-entropy’ as a loss function and ‘Stochastic Gradient Descent’ as an optimizer with a learning rate of 0.0001 and a momentum of 0.9. The metric for our model is ‘accuracy’. We run 20 iterations, Figure 5 is a snapshots of loss and accuracy with iterations.



```
Epoch 1/20
36/35 [==============================] - 372s 10s/step - loss: 0.8554 - acc: 0.5679
Epoch 2/20
36/35 [==============================] - 368s 10s/step - loss: 0.6314 - acc: 0.6691TA: 2:32
- loss: 0.6388 - acc: 0.6577
Epoch 3/20
36/35 [==============================] - 377s 10s/step - loss: 0.5841 - acc: 0.7247
                                        :
[===========>..................] - ETA: 3:32 - loss: 0.4135 - acc: 0.816736/35
[==============================] - 363s 10s/step - loss: 0.3913 - acc: 0.8287
Epoch 19/20
36/35 [==============================] - 372s 10s/step - loss: 0.3176 - acc: 0.8722
Epoch 20/20
36/35 [==============================] - 367s 10s/step - loss: 0.3081 - acc: 0.8756
```

Figure 5: Training Iterations (model achieves accuracy of 0.87 after 20 iterations)

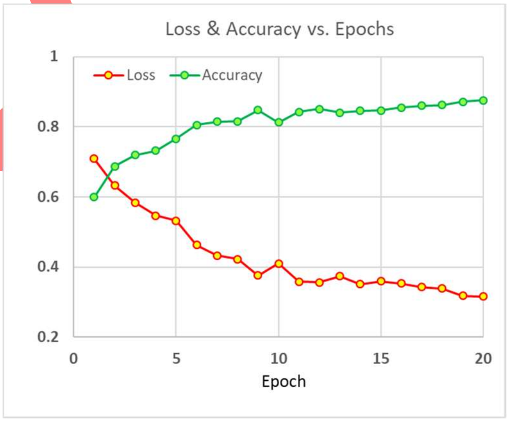


Figure 6: Loss and Accuracy during training.

- **Testing /Evolution: results (classification report)**

To evaluate the model we run the prediction on the test data. In this section we present the high-level results of the test process.

Table 1 presents a summary of the DNN prediction. %95 of the predicted fractures are true fractures. and 77% of the prediction are True negative. Overall the F1-score is 85%.

	Precision	**Recall**	**F1-score**	**Support**
0 : Fracture	0.95	0.77	0.85	162
1: No fracture	0.76	0.95	0.84	124
Micro Avg	0.85	0.85	0.85	286
Macro Avg	0.86	0.86	0.85	286
Weighted Avg	0.87	0.85	0.85	286

Table 1: Classification report

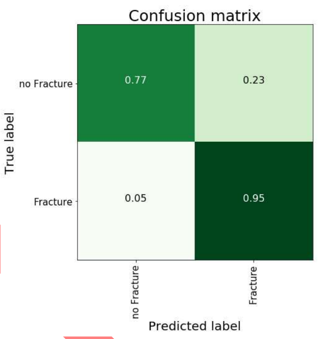


Figure 7: Confusion Matrix

- **Evaluation:**

To illustrate the findings of the DNN in identifying fractures form images. We import intermediate weights form the network to highlight the learning and the features used identify fractures.

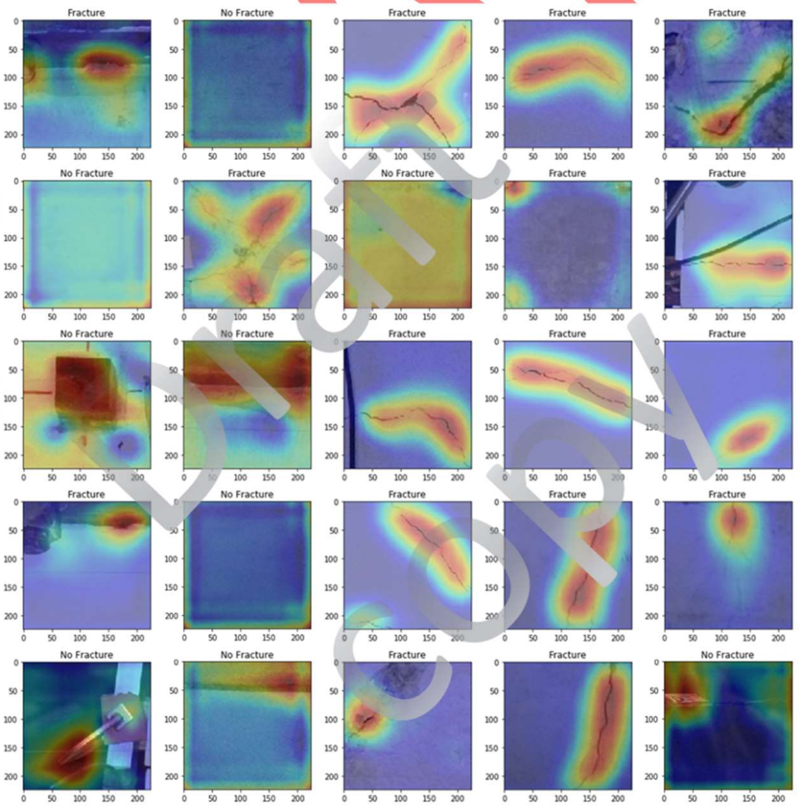
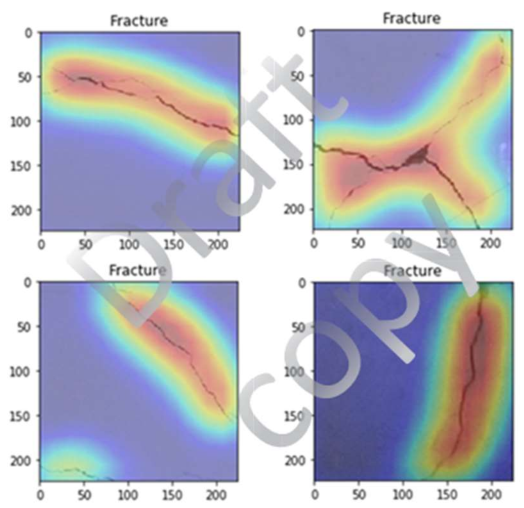
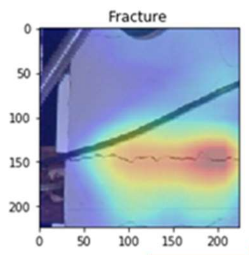
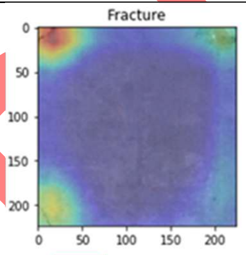
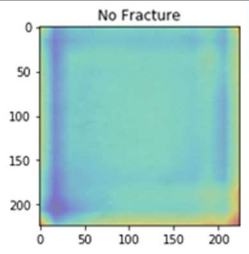


Figure 8: DNN prediction with intermediate weights overlapped on top of the images to highlight the model attention to fractures.

 Figure 9: Model predictions – clear fractures	Figure 8 presents clear example of fractures identified by the model. In this application, the model was able to identify 100% of the clear fractures. The learning weights for certain hidden layers are overlapped on top of the images to highlight the fractures and model attention. Clearly the model was able to identify the exact location of the fracture as well as the orientation. The intensity of the weights / attention is strongly correlated with the density/width of the fracture. These findings could be very valuable when characterizing fractures from well log images (or seismic discontinuity from seismic volumes). It provides a quantitative evaluation of fracture/anomaly identification.
 Figure 10: Model predictions : fracture with additional objects in the image	Figure 9 highlights a case where we add additional objects, a dark wire, next the fracture. the model was successful at identify the 'actual fracture' and ignore the 'fracture-lookalike' object. The overlapped attention/weights clearly highlights the fracture location. The attention intensity correlate with the fracture width.
 Figure 11: Model prediction - fracture at the corner of the image	Figure 10 highlights fractures identified at the corner of the image. Despite the odd location of the fracture, the model was able to successfully identify the fractures.
 Figure 12: Model prediction - no fracture	Figure 11 presents a case where there is no fracture. Clearly the model was able to recognize the absence of fractures.