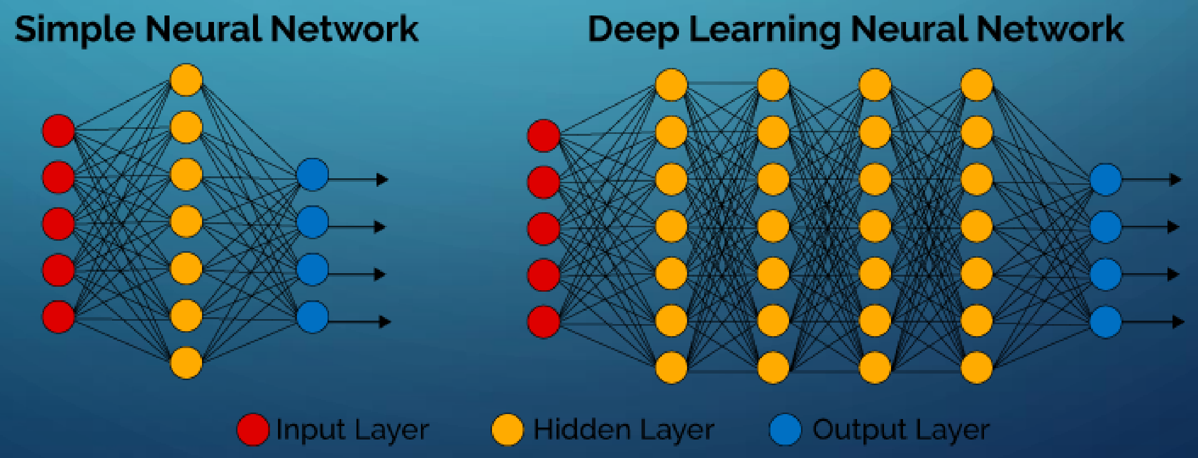
# SEISMIC IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

ALEX HOWELL

VISTAS IN ADVANCED COMPUTING

# DEEP MACHINE LEARNING

- Imitates how the human brain works by using neural networks
- Multiple nodes used in each layer throughout the process
- Each neuron has a weight associated with it which is a number that is changed while learning
- Has the ability to predict based of previous data

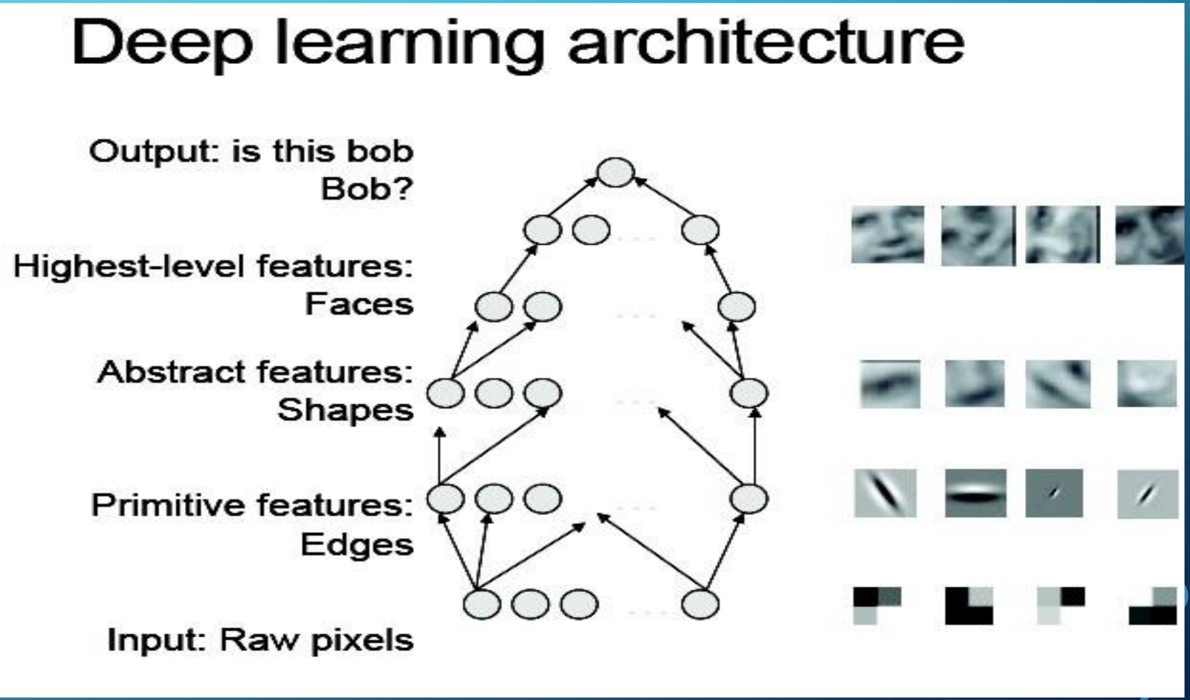


# HIERARCHICAL REPRESENTATIONS

"Deep learning methods aim at **learning feature hierarchies** with features from higher levels of the hierarchy formed by the composition of lower level features.

**Automatically learning features** at multiple levels of abstraction allows a system to learn **complex functions** mapping the input to the output directly from data, without depending completely on human-crafted features."

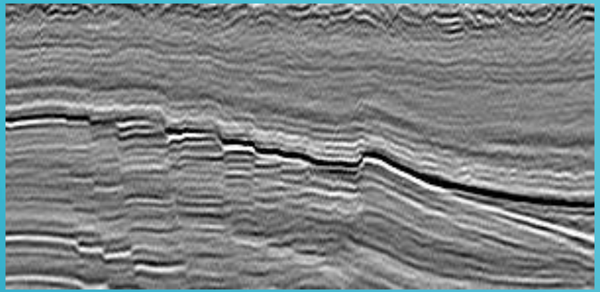
— Yoshua Bengio




[Bengio, "On the expressive power of deep architectures", *Talk at ALT*, 2011]
[Bengio, *Learning Deep Architectures for AI*, 2009]

# DEEP LEARNING - NEURAL NETWORK

How to capture representation of objects in computational/mathematical model?



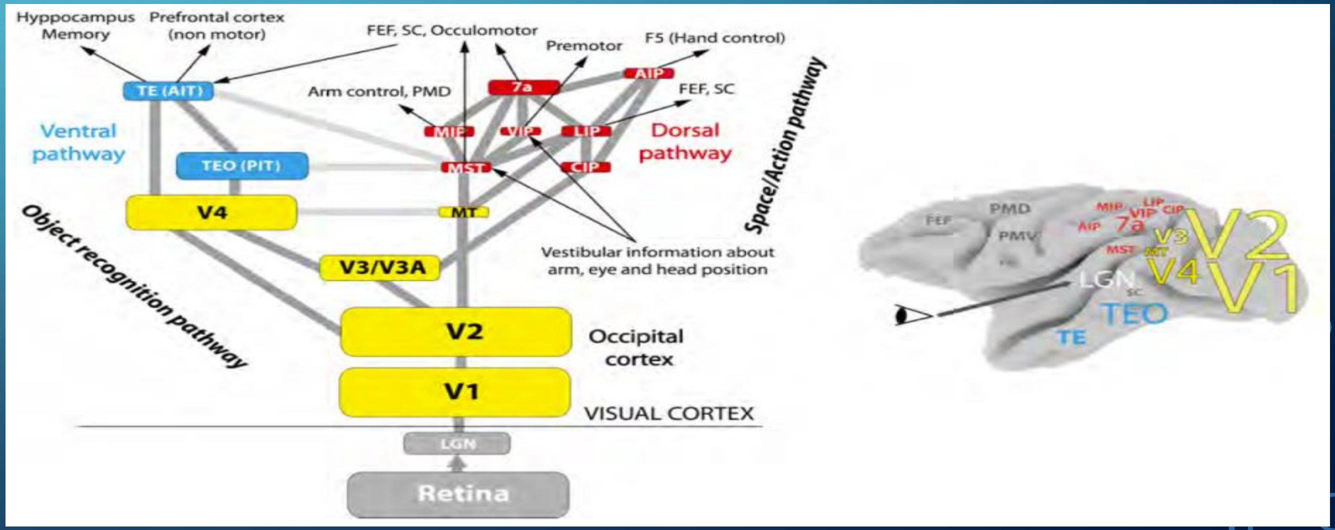
Objects: a category [position or segmentation]

**Let machine learn the model from samples**

Inspiration from biological (human/primate) visual system

Key element: *a hierarchy*

- Biological plausibility
- Part sharing between objects/categories
  ➔ Efficient representation
- Object/part as a composition of other parts
  ➔ Compositional interpretation

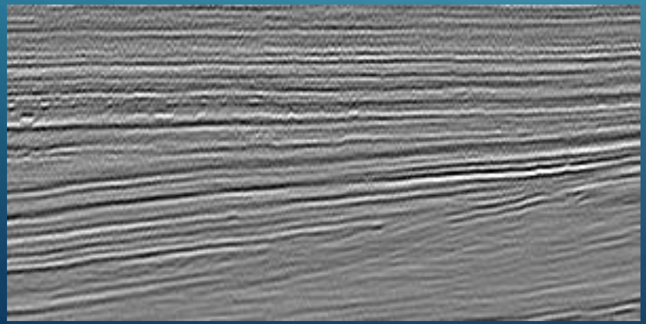



Kruger et al, Deep Hierarchies in the Primate Visual Cortex : What Can We Learn For Computer Vision ?, PAMI 2012

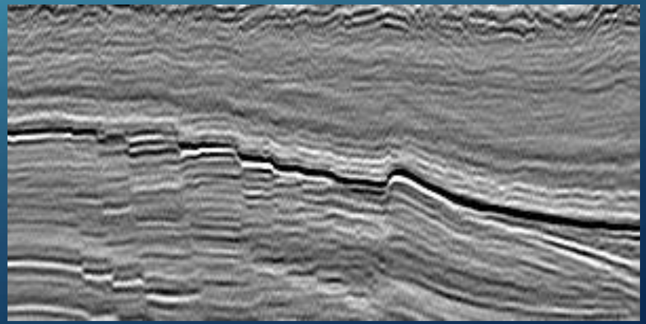
# IMPORTANCE OF SEISMIC ANALYSIS

- Detects underground features and distinguishes them from each other
- Faults and other features are important to find before working on the ground
- For example, finding the area where oil, gas or certain types of rocks are
- The algorithms can highlight faults for trying to locate petroleum in a reservoir
- The work would be conducted on a specific spot because of the data extracted

No Faults

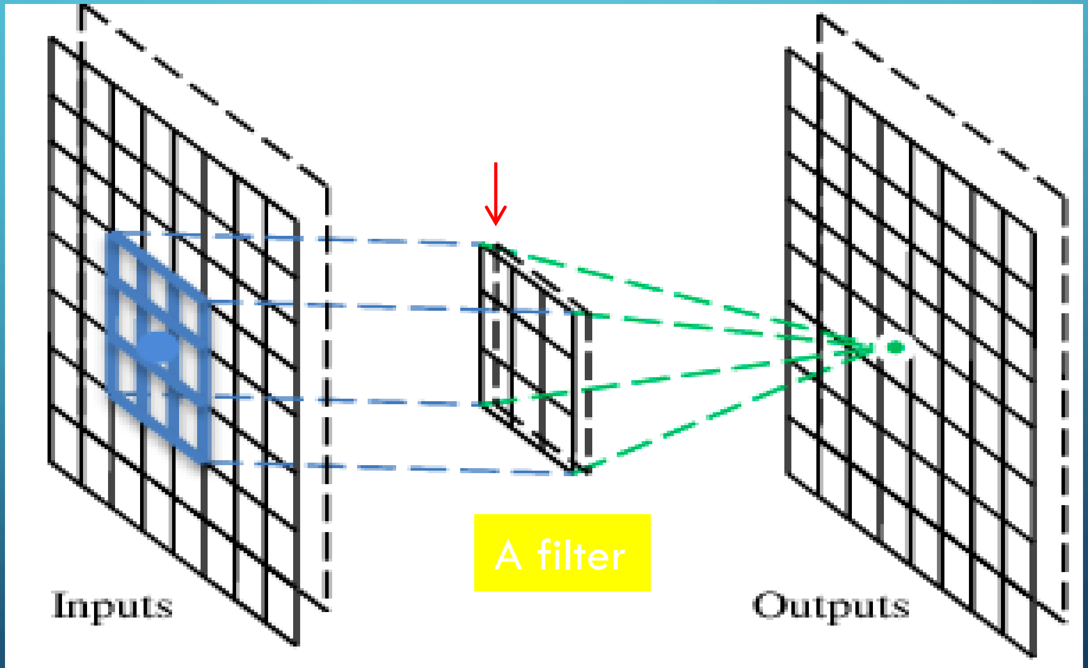


Faults



# Convolutional Neural Networks

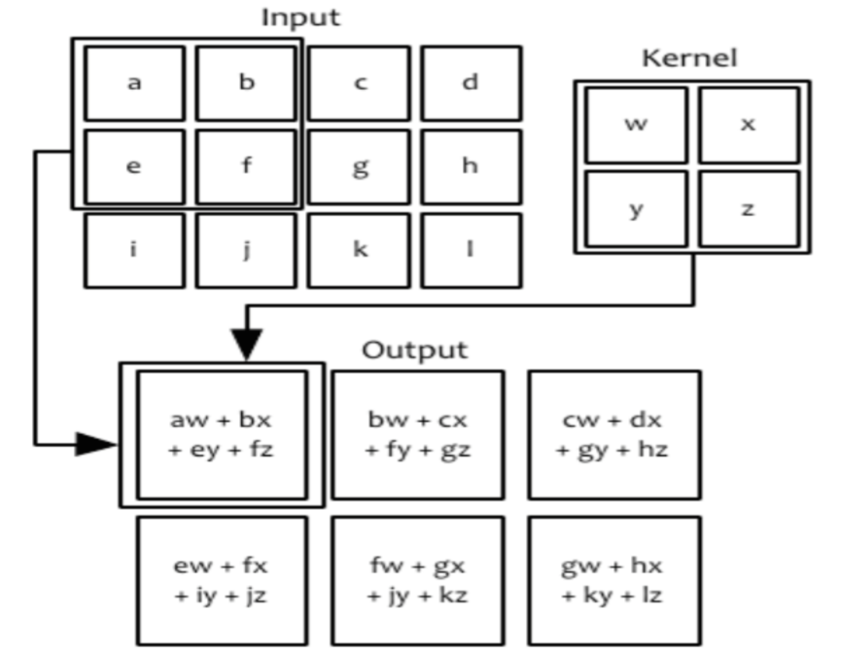
A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



# CONVOLUTION OPERATION IN CNN

- Input: an image (2-D array) x
- Convolution kernel/operator(2-D array of learnable parameters): w

$$s[i,j] = (x * w)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} x[i+m, j+n]\, w[m,n]$$



# DATA USED

- Total: 2000 pictures
  - 150x300 pixels
- Class1 – no-fault: 1000 images
- Class2 – fault: 1000 images
- Training: 980 images for Class1, and 980 images for Class2
- Testing: 20 images for Class1, and 20 images for Class2
- Dataset: Large North-Sea Dataset of Migrated Aggregated Seismic Structures (Landmass)
- Available at: http://cegp.ece.gatech.edu/codedata/landmass/

# IMPORTS

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Lambda
from keras.layers import Dense
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import train_test_split
import cv2
import scipy
import os
%matplotlib inline
import matplotlib.pyplot as pet
```

Keras – Neural Network API used for processing pictures
Sklearn – Used mainly for classification in this project
os, scipy, cv2 – Used for imported pictures into usable object

# GETTING PICTURES INTO THE PROGRAM

```python
from keras.preprocessing import image
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []

    for seismic_type in os.listdir(folder):
        if not seismic_type.startswith('.'):
            if seismic_type in ['Class1']:
                label = '0'
            else:
                label = '1'
            for image_filename in os.listdir(folder + seismic_type):
                img_file = cv2.imread(folder + seismic_type + '/' + image_filename)
                if img_file is not None:
                    # Downsample the image to 120, 160, 3
                    #img_file = scipy.misc.imresize(arr=img_file, size=(120, 160, 3))
                    img_arr = np.asarray(img_file)
                    X.append(img_arr)
                    y.append(label)
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
```

- “folder” is the directory
- Separates the pictures into two classes
- X holds the pictures
- y holds labels corresponding to X

# SETTING UP FOR DEEP LEARNING

```
X_train, y_train = get_data(BASE_DIR + 'images/Train/')
X_test, y_test = get_data(BASE_DIR + 'images/Test/')
X_train = X_train*1./255.
X_test = X_test*1./255.
encoder = LabelEncoder()
encoder.fit(y_train)
y_train = encoder.transform(y_train)
y_test = encoder.transform(y_test)
```

Function Call

Normalizing Data

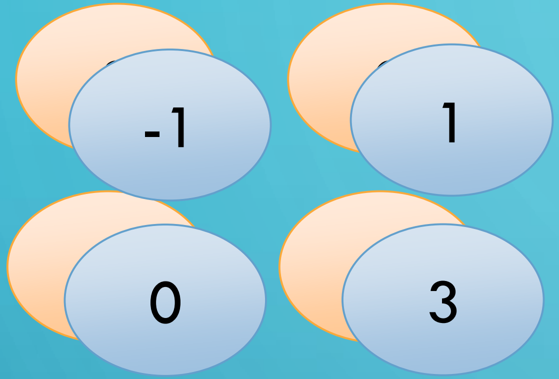
Sets the classes as 0 and 1

# ONE CONVOLUTION ALGORITHM

```
def get_model():
    model = Sequential()
    model.add(Conv2D(32, (3,3), input_shape=(150, 300, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(units=128,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return model
```

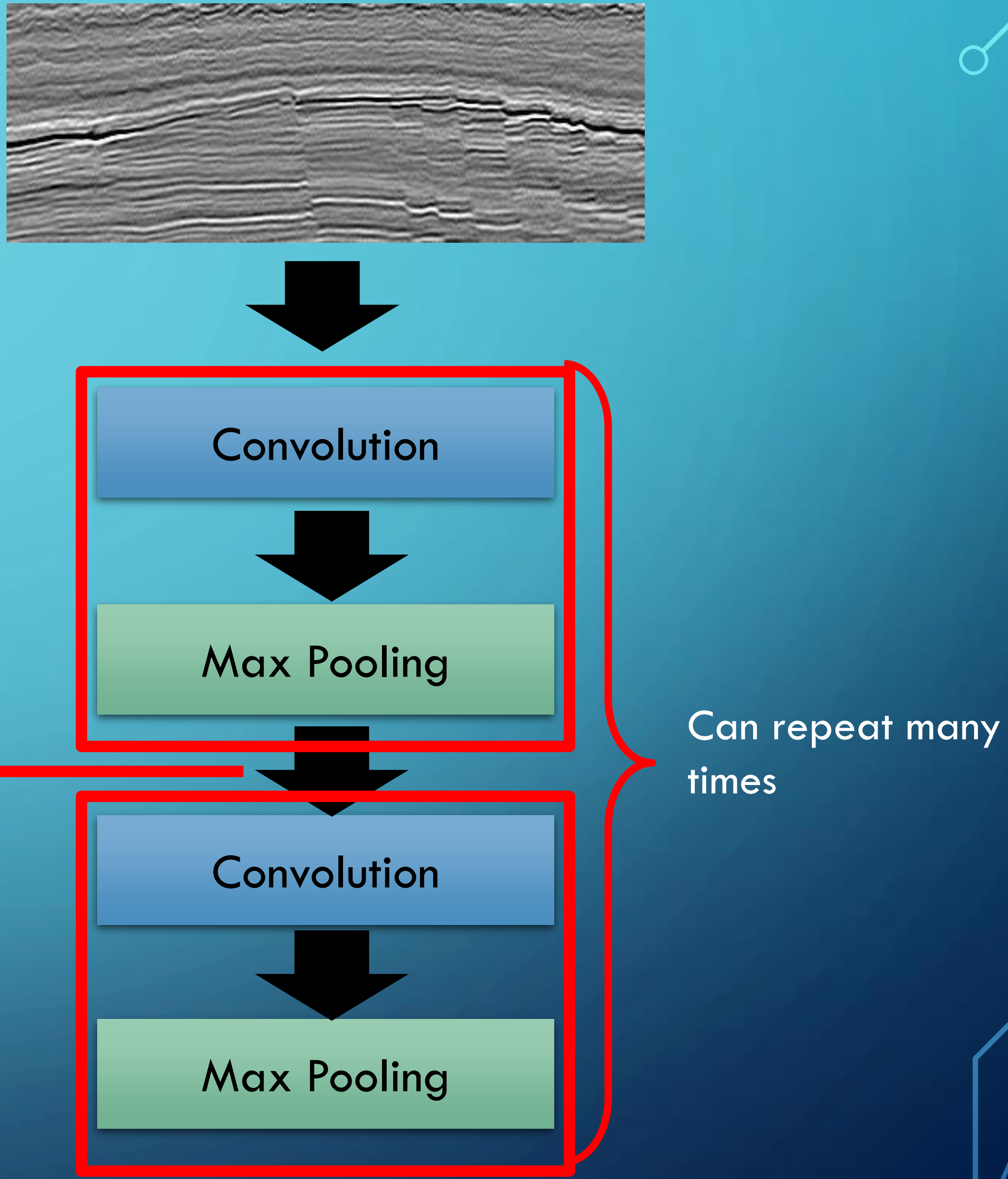
- Conv2D – Number of output operations
- MaxPooling2D – Clears the noise
- Flatten – Changes the shape of the input
- Dropout – Takes out a percentage of neurons to use
- Dense – Creates a hidden layer of neurons
- Activation – Applies an activation function
- Compile – Puts all the layers together

# The Whole CNN



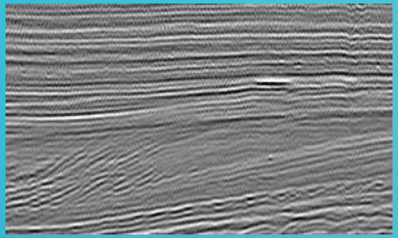

A new image

Smaller than the original image

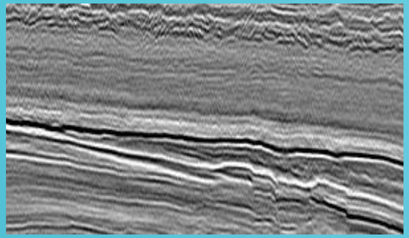


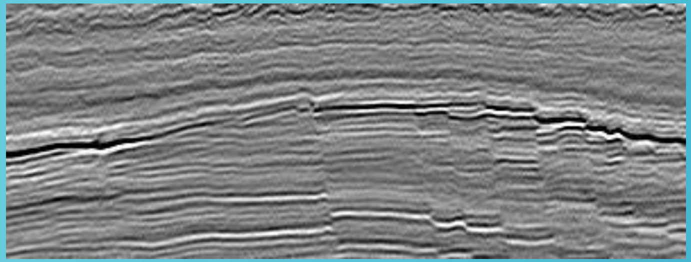
# The Whole CNN

no-fault



fault





Convolution

Max Pooling

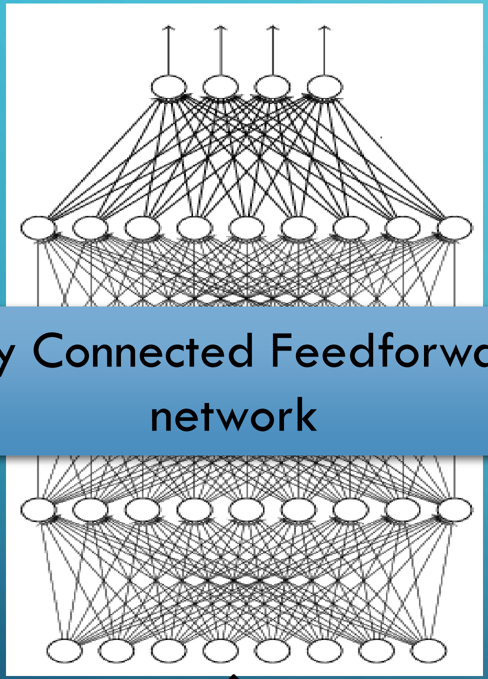
A new image

Convolution

A new image

Max Pooling

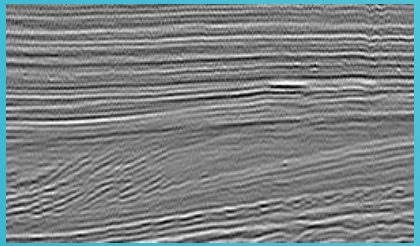
Flattened



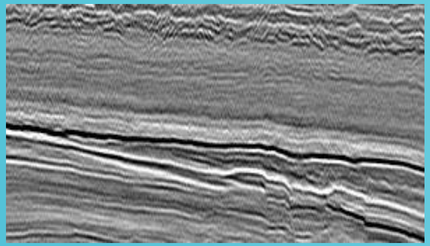
Fully Connected Feedforward network

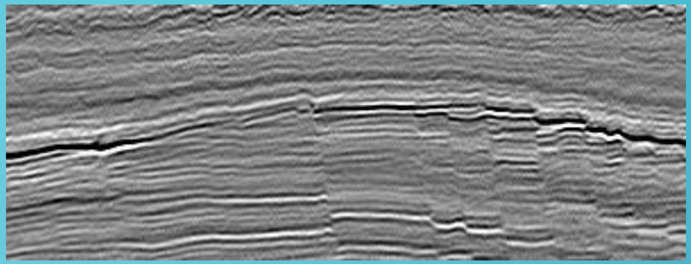
# THE WHOLE CNN

no-fault



fault





Convolution

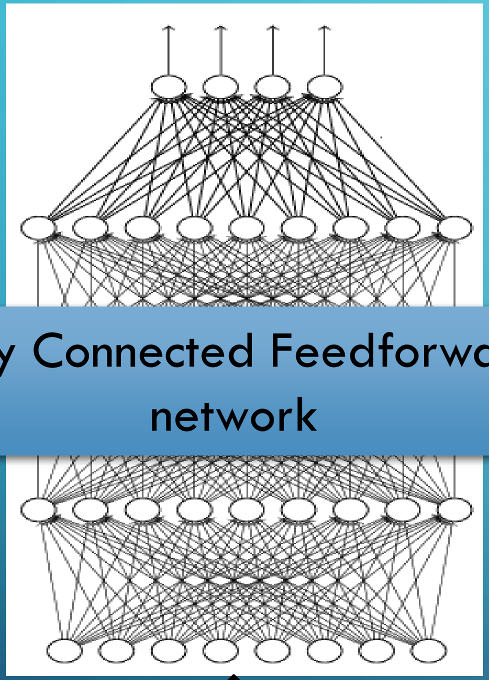
Max Pooling

Convolution

Max Pooling

Can repeat many times

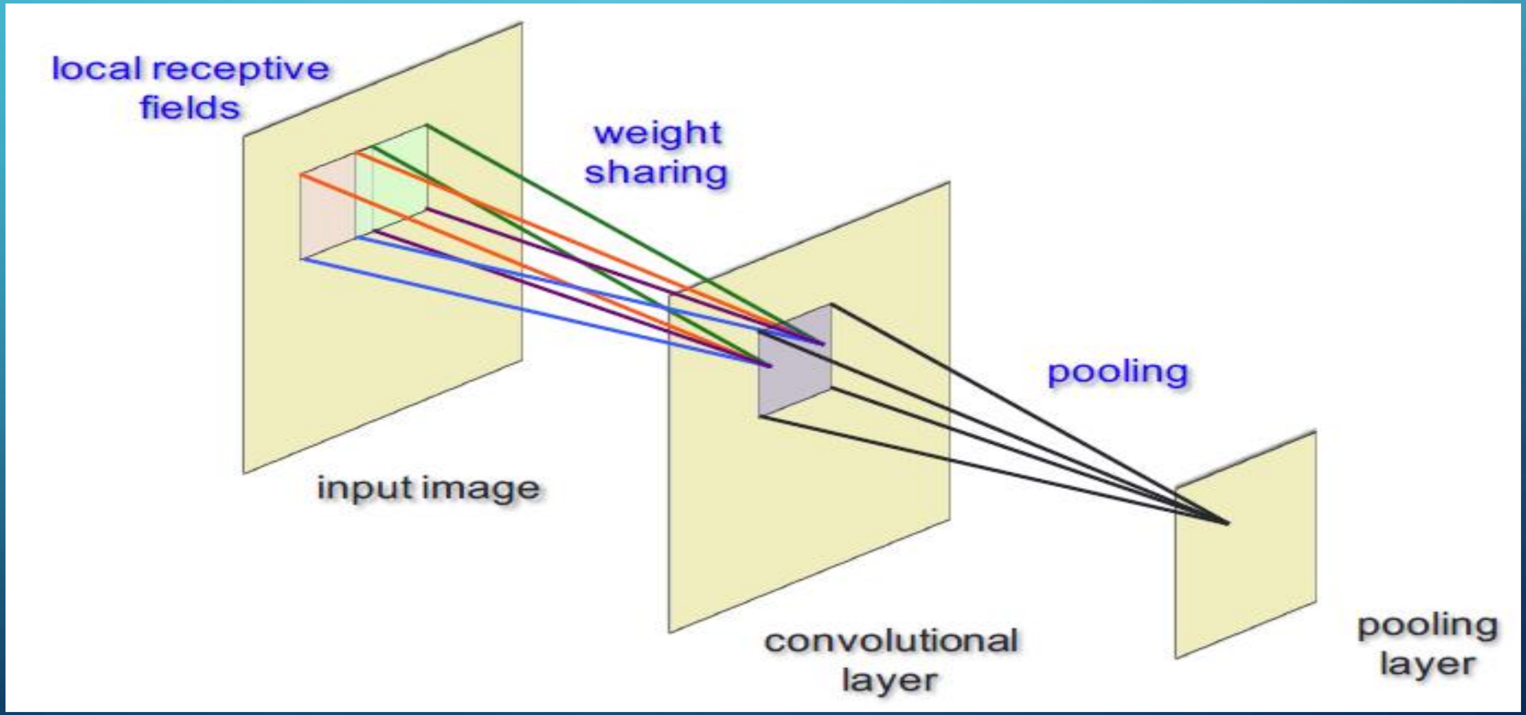
Flattened



Fully Connected Feedforward network

# POOLING

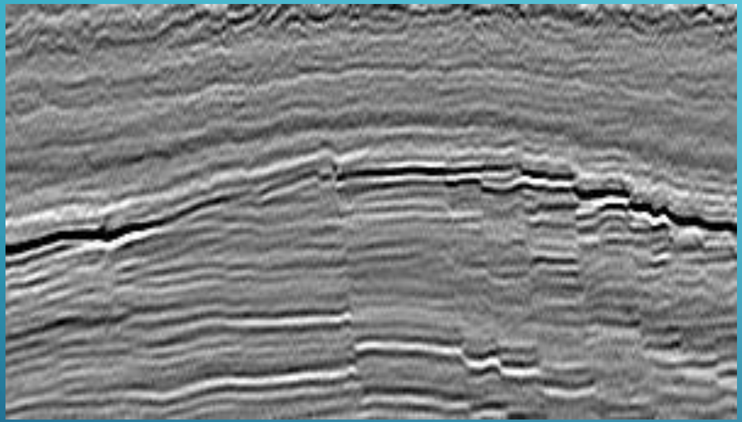
- Common pooling operations:
  - Max pooling: reports the maximum output within a rectangular neighborhood.
  - Average pooling: reports the average output of a rectangular neighborhood (possibly weighted by the distance from the central pixel).



# WHY POOLING

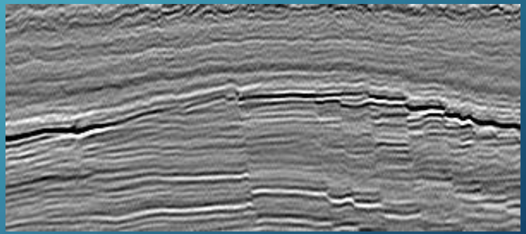
- Subsampling pixels will not change the object

fault



Subsampling

fault



We can subsample the pixels to make image smaller → fewer parameters to characterize the image

# MAX POOLING

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

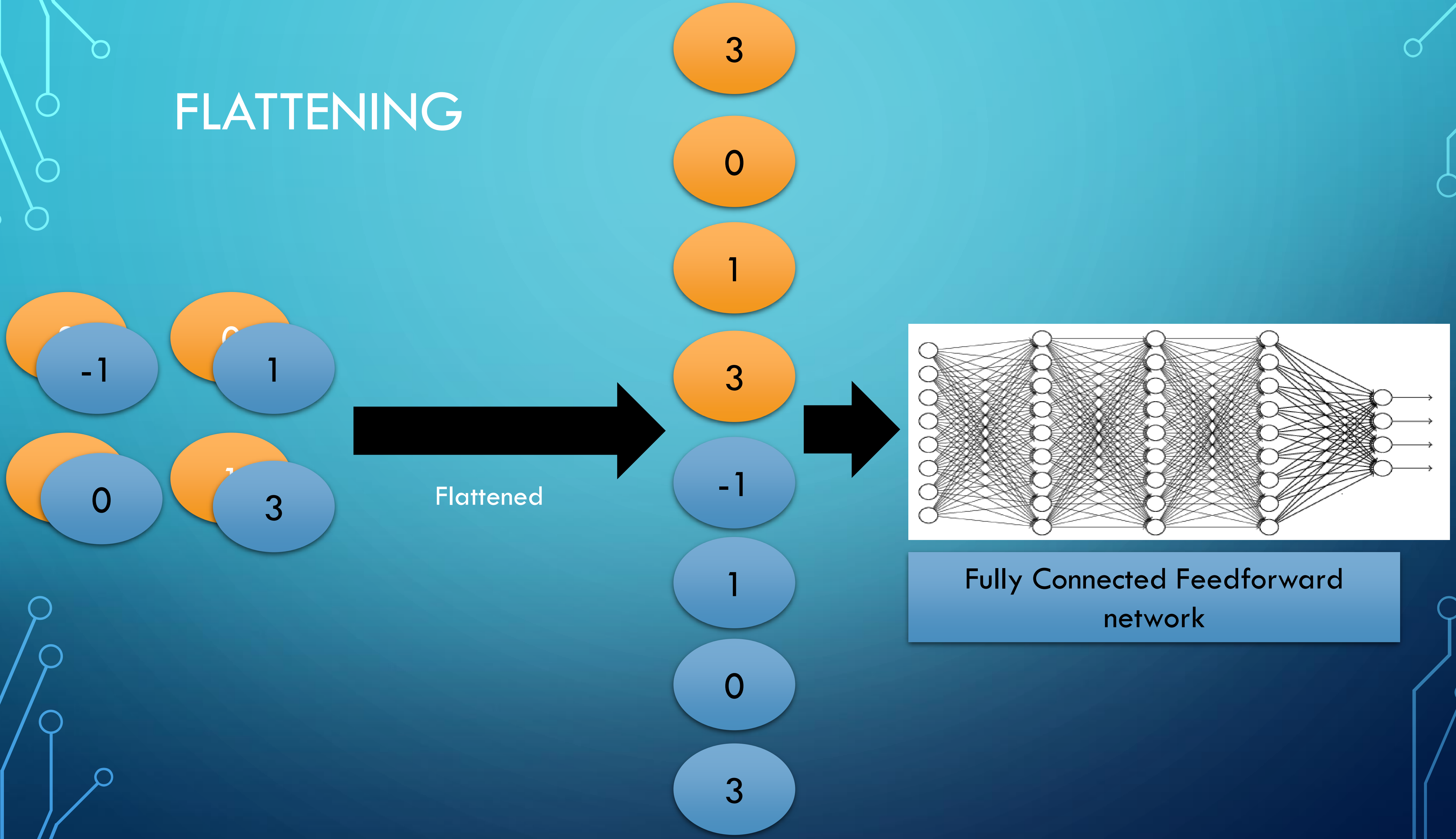
| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

| | | | |
|---|---|---|---|
| 3 | -1 | -3 | -1 |
| -3 | 1 | 0 | -3 |
| -3 | -3 | 0 | 1 |
| 3 | -2 | -2 | -1 |

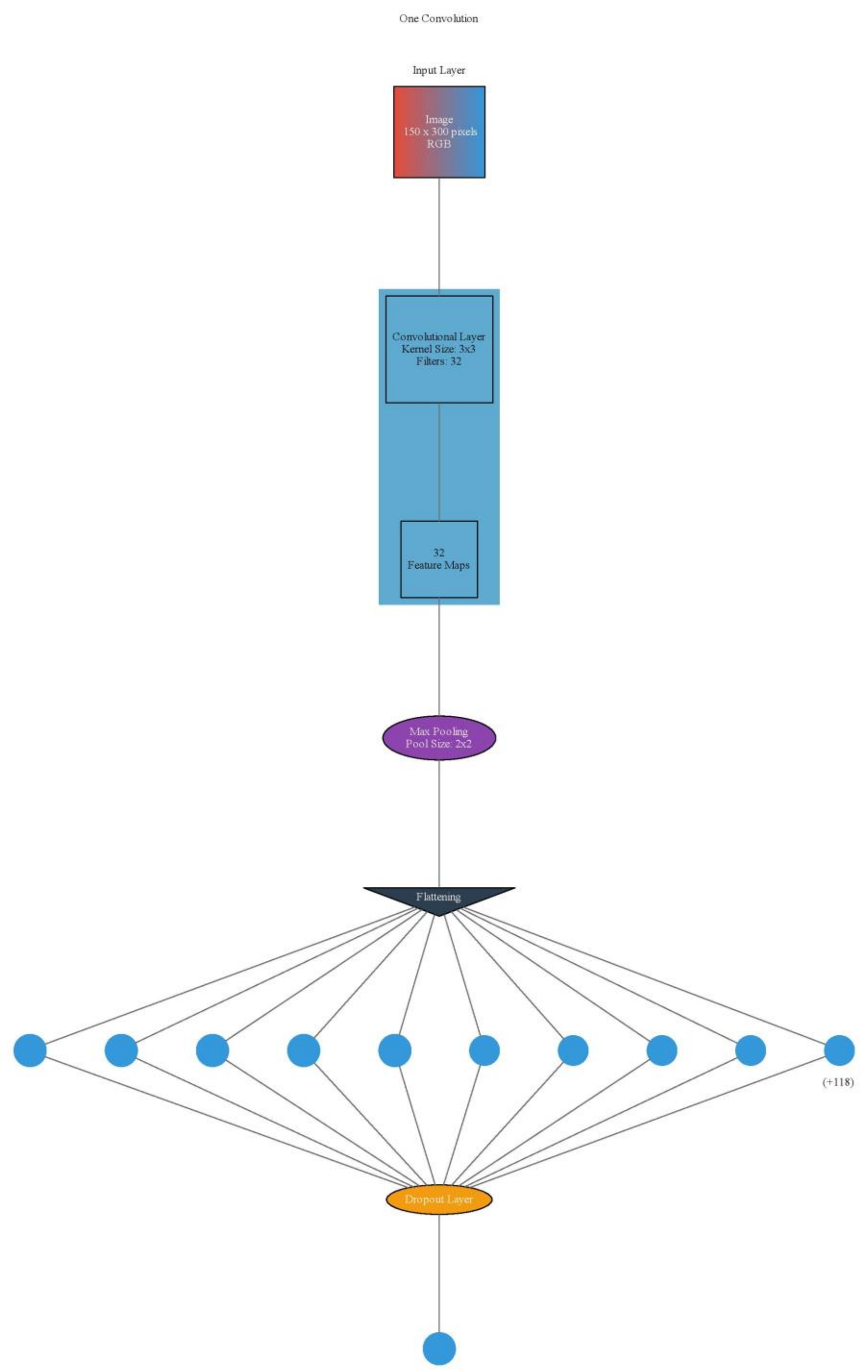
| | | | |
|---|---|---|---|
| -1 | -1 | -1 | -1 |
| -1 | -1 | -2 | 1 |
| -1 | -1 | -2 | 1 |
| -1 | 0 | -4 | 3 |

# FLATTENING


-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected Feedforward network

# Visual Representation of Model Layering

```python
from ann_visualizer.visualize import ann_viz;

ann_viz(model, title="One Convolution")
```




Input of 300x150 image

Conv2D Layer
Kernel Size: 3x3
Filters: 32

Max Pooling
Pool Size: 2x2

Flattening

Dense 128
Neuron Network

Dropout

Dense 1 Neuron
(Binary Classification)

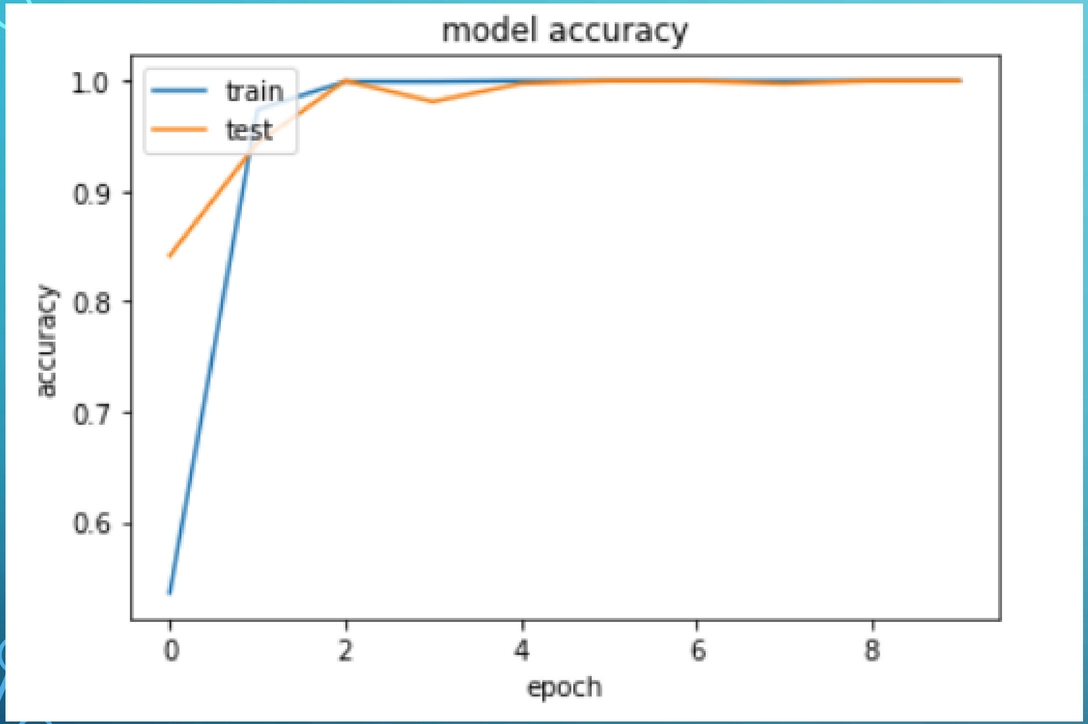
# THE LEARNING PART

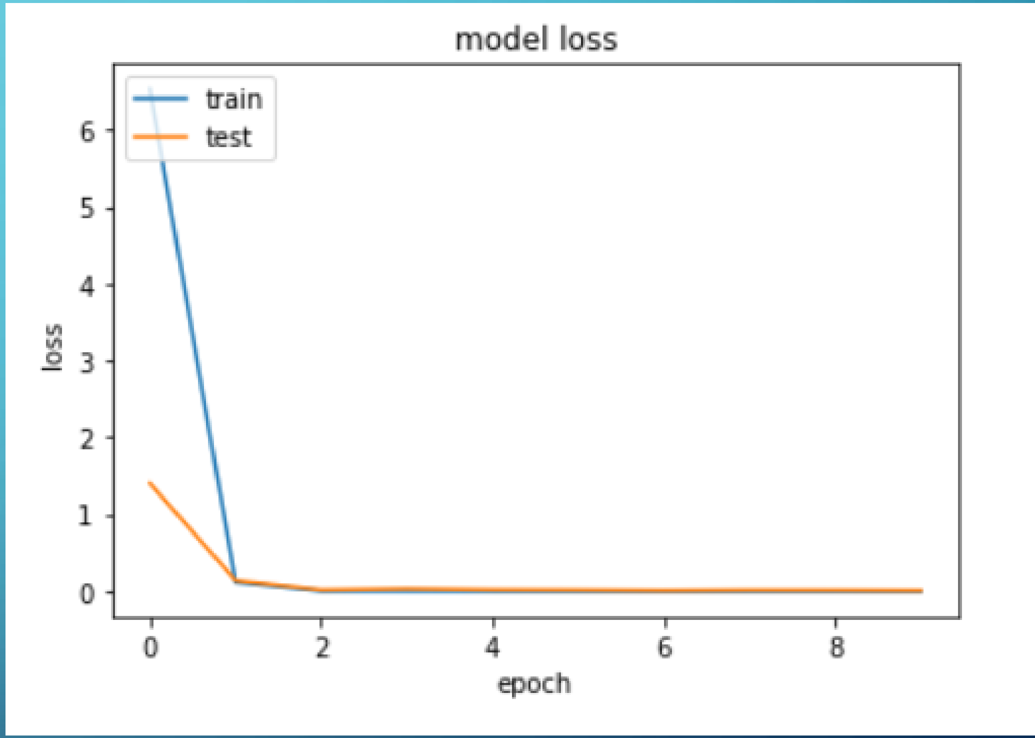
```
model = get_model()
# fits the model on batches
history = model.fit(X_train,y_train,validation_split=0.2,epochs=epochs,shuffle=True,batch_size=batch_size)
```

```
Train on 1488 samples, validate on 372 samples
Epoch 1/10
1488/1488 [==============================] - 122s 82ms/step - loss: 6.5331 - acc: 0.5356 - val_loss: 1.4028 - val_acc: 0.8414
Epoch 2/10
1488/1488 [==============================] - 124s 83ms/step - loss: 0.1156 - acc: 0.9738 - val_loss: 0.1381 - val_acc: 0.9435
Epoch 3/10
1488/1488 [==============================] - 124s 83ms/step - loss: 0.0019 - acc: 0.9993 - val_loss: 0.0150 - val_acc: 1.0000
Epoch 4/10
1488/1488 [==============================] - 126s 85ms/step - loss: 8.8079e-04 - acc: 0.9993 - val_loss: 0.0288 - val_acc: 0.98
12
Epoch 5/10
1488/1488 [==============================] - 124s 83ms/step - loss: 1.3041e-04 - acc: 1.0000 - val_loss: 0.0171 - val_acc: 0.99
73
Epoch 6/10
1488/1488 [==============================] - 123s 83ms/step - loss: 1.0224e-04 - acc: 1.0000 - val_loss: 0.0138 - val_acc: 1.00
00
Epoch 7/10
1488/1488 [==============================] - 124s 83ms/step - loss: 2.8502e-05 - acc: 1.0000 - val_loss: 0.0075 - val_acc: 1.00
00
Epoch 8/10
1488/1488 [==============================] - 126s 85ms/step - loss: 4.1331e-05 - acc: 1.0000 - val_loss: 0.0123 - val_acc: 0.99
73
Epoch 9/10
1488/1488 [==============================] - 127s 86ms/step - loss: 1.9421e-05 - acc: 1.0000 - val_loss: 0.0112 - val_acc: 1.00
00
Epoch 10/10
1488/1488 [==============================] - 126s 85ms/step - loss: 1.4042e-06 - acc: 1.0000 - val_loss: 0.0073 - val_acc: 1.00
00
```

- fit – uses the model on the dataset to be used in the machine learning
- epoch – iterations to learn the entire set
- validation_split – sets aside part of training data to be checked for better accuracy

# VISUALIZATION OF RESULTS




- Train – train set data
- Test – validation set data
- Loss – Error in learning

# PREDICTIONS ON ONE CONVOLUTION

```
from sklearn.metrics import accuracy_score

print('Predicting on test data')
y_pred = np.rint(model.predict(X_test))
print(accuracy_score(y_test,y_pred))
```

```
Predicting on test data
1.0
```

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[20  0]
 [ 0 20]]
```

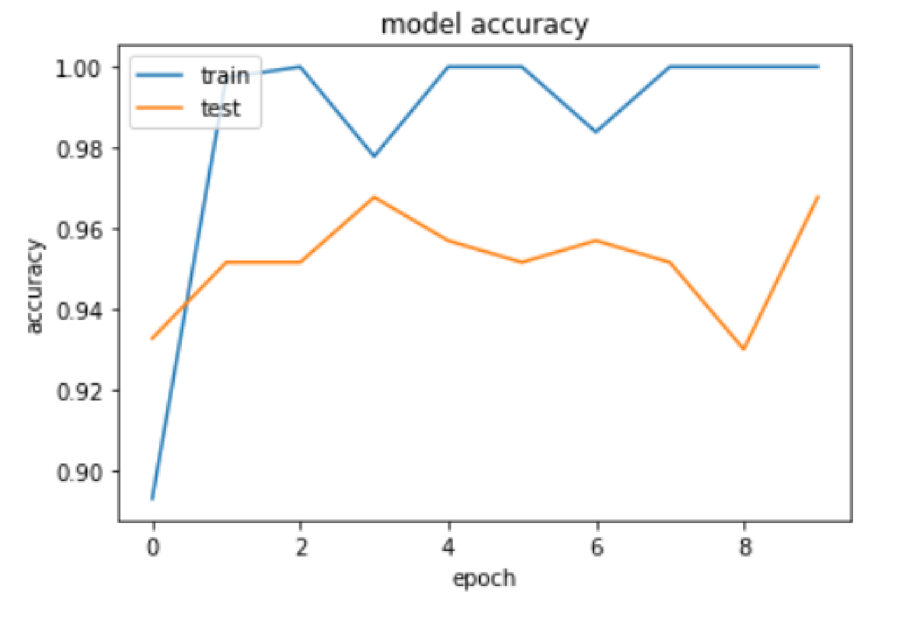
100% Accuracy

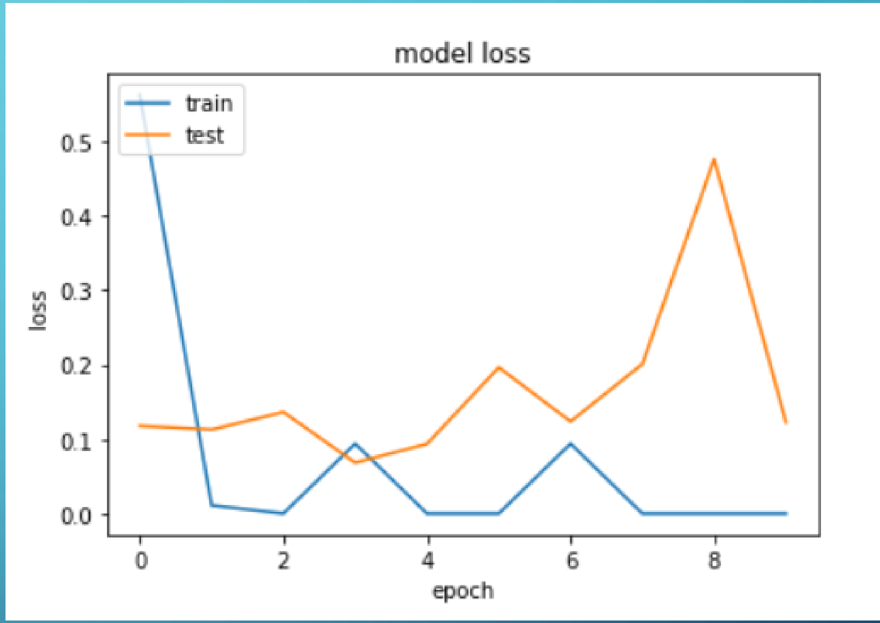
# TWO CONVOLUTIONS ALGORITHM

```
def get_model():

    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(150, 300, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])

    return model
```

- Conv2D – Number of output operations
- MaxPooling2D – Clears the noise
- Flatten – Changes the shape of the input
- Dropout – Takes out a percentage of neurons to use
- Dense – Creates a hidden layer of neurons
- Activation – Applies an activation function
- Compile – Puts all the layers together

# TWO CONVOLUTION RESULTS




- Train – train set data
- Test – validation set data
- Loss – Error in learning

# PREDICTIONS ON TWO CONVOLUTIONS

```
from sklearn.metrics import accuracy_score

print('Predicting on test data')
y_pred = np.rint(model.predict(X_test))
print(accuracy_score(y_test,y_pred))
```

```
Predicting on test data
0.975
```

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[20  0]
 [ 1 19]]
```

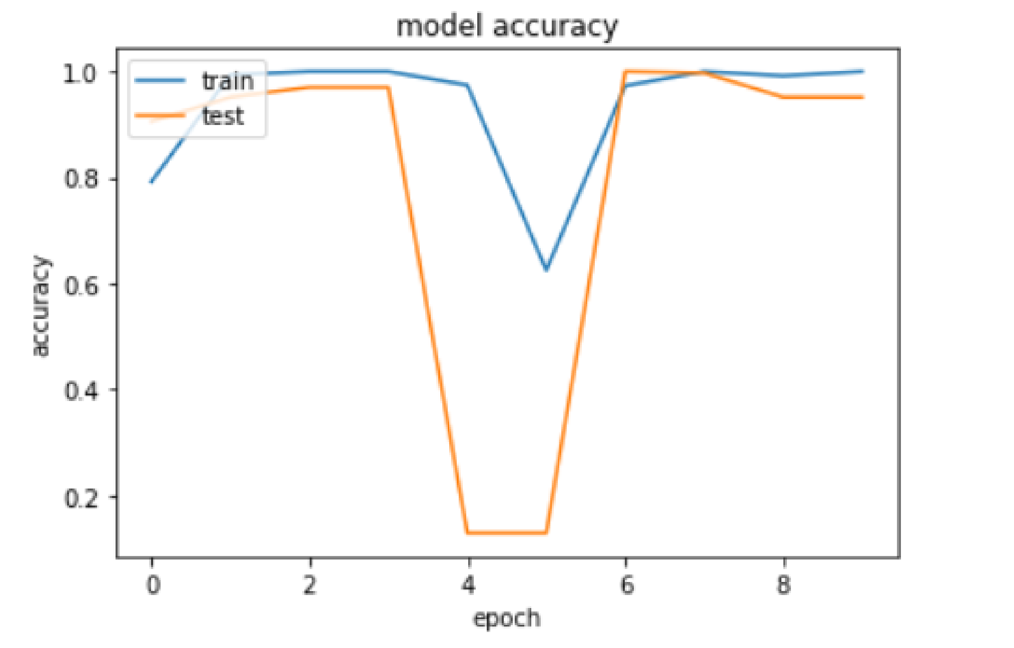
97.5% Accuracy

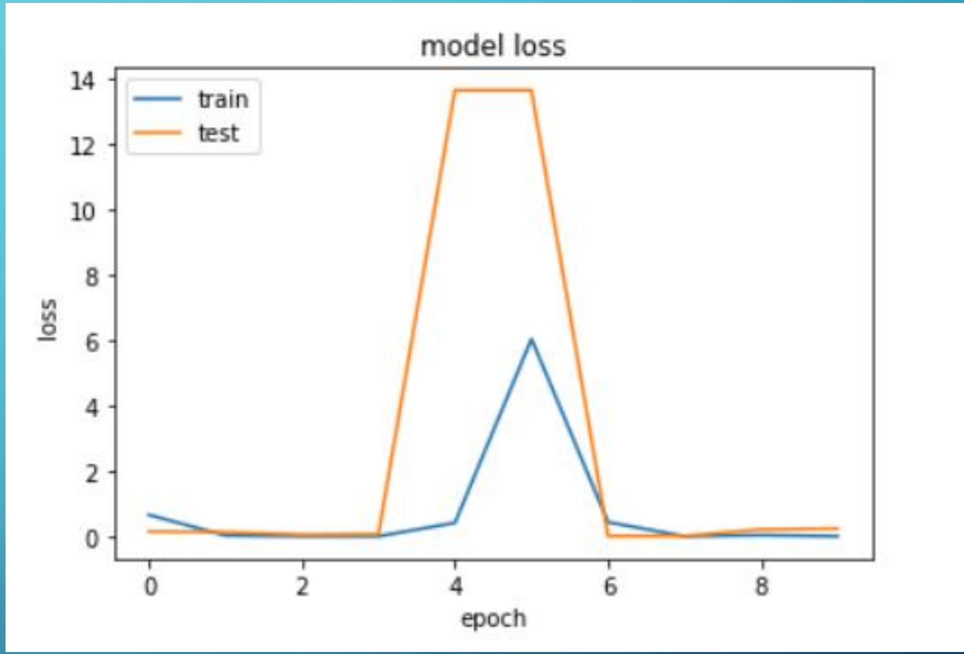
# THREE CONVOLUTION ALGORITHM

```
def get_model():

    model = Sequential()
    model.add(Conv2D(32, (3,3), input_shape=(150, 300, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3,3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3,3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(units=128,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])

    return model
```

- Conv2D – Number of output operations
- MaxPooling2D – Clears the noise
- Flatten – Changes the shape of the input
- Dropout – Takes out a percentage of neurons to use
- Dense – Creates a hidden layer of neurons
- Activation – Applies an activation function
- Compile – Puts all the layers together

# THREE CONVOLUTION RESULTS




- Train – train set data
- Test – validation set data
- Loss – Error in learning

# PREDICTIONS ON THREE CONVOLUTIONS

```
from sklearn.metrics import accuracy_score

print('Predicting on test data')
y_pred = np.rint(model.predict(X_test))
print(accuracy_score(y_test,y_pred))
```

```
Predicting on test data
0.95
```

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[20  0]
 [ 2 18]]
```

95% Accuracy

# CONCLUSION

- The multilayered convolution algorithm is most likely better suited for more complicated data
- For the simpler faults, less convolutions show better results because there is less to analysis
- Having other seismic features would require a more complex algorithm than just one convolution

# CITATIONS


Brownlee, J. (2018). *Display Deep Learning Model Training History in Keras*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/ [Accessed 8 Aug. 2018].

Di, Haibin & Shafiq, Amir & Alregib, Ghassan. (2017). Seismic-fault detection based on multiattribute support vector machine analysis. 2039-2044. 10.1190/segam2017-17748277.1.

Gill, J. (2018). *Automatic Log Analysis using Deep learning and AI forMicroservices*. [online] Xenostack. Available at: https://www.xenonstack.com/blog/data-science/log-analytics-deep-machine-learning-ai/ [Accessed 8 Aug. 2018].

Investopedia. (2018). *Deep Learning*. [online] Available at: https://www.investopedia.com/terms/d/deep-learning.asp [Accessed 8 Aug. 2018].

Keras.io. (2018). *Keras Documentation*. [online] Available at: https://keras.io/ [Accessed 8 Aug. 2018].

Shah, A. (2018). *Visualizing Artificial Neural Networks (ANNs) with just One Line of Code*. [online] Towards Data Science. Available at: https://towardsdatascience.com/visualizing-artificial-neural-networks-anns-with-just-one-line-of-code-b4233607209e [Accessed 8 Aug. 2018].

Scikit-learn.org. (2018). *sklearn.preprocessing.LabelEncoder — scikit-learn 0.19.2 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html [Accessed 9 Aug. 2018].