CLASSIFICATION AND LOCALIZATION OF FRACTURE-HIT EVENTS IN LOW-FREQUENCY DAS STRAIN RATE WITH CONVOLUTIONAL NEURAL NETWORKS

A Thesis

by

MENGYUAN CHEN

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Ding Zhu
Committee Members,	Alfred Daniel Hill
	Zenon Medina-Cetina
Head of Department,	Jeff Spath

August 2021

Major Subject: Petroleum Engineering

ABSTRACT

Distributed acoustic sensing (DAS) has been used in the oil and gas industry as an advanced technology for surveillance and diagnostics. Operators use DAS to monitor hydraulic fracturing activities, to examine well stimulation efficacy, and to estimate complex fracture system geometries. Particularly, low-frequency DAS can detect geomechanical events such as fracture-hits as hydraulic fractures propagate and create strain rate variations. Analysis of DAS data today is mostly done post-job and subject to interpretation methods. However, the continuous and dense data stream generated live by DAS offers the opportunity for more efficient and accurate real-time data-driven analysis.

The objective of this study is to develop a machine learning-based workflow that can identify and locate fracture-hit events in low-frequency DAS data. The study is conducted in two phases. In phase one, a fracture propagation model is used to produce strain rate patterns observed at a hypothetical monitoring well. Using this model, two sets of strain rate responses are generated with one set containing fracture-hit events. The simulated data are then used to train a custom convolutional neural network (CNN) model for identifying the presence of fracture-hit events. The same model is trained again for locating the event with the output layer of the model replaced with linear units. The models achieved near-perfect predictions for both event classification and localization. In phase two, the same workflow is applied to field data, which includes 8.4 days of DAS monitoring data while two offset wells are hydraulically stimulated. A more complex model (AlexNet) is used to train for classifying events and for localizing fracture-hits. Using AlexNet, we achieved f1 score of 0.9 for identifying fracture-hits and R^2 of 0.93 for localizing fracture-hits.

Additionally, edge detection techniques are used for recognizing fracture-hit event patterns in the simulated strain rate images. The accuracy is also plausible, but edge detection is more dependent on image quality and shape, hence less robust compared to CNN models. It can only be applied to simulated data since field data often shows irregular fracture-hit patterns. This comparison further supports the need for CNN applications in image-based real-time fiber optic sensing event detection.

DEDICATION

To my parents, Youzhuan Zhang and Yibin Chen.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Ding Zhu, and Dr. A. Daniel Hill for taking me in as a graduate student, for guiding me on my study and research, and for allowing me the opportunity to explore the subject. Pursuing graduate study had been my dream for years. They have immensely helped me make this dream come true.

I thank Dr. Zenon Medina-Cetina for being my committee member and for his valuable input to my research.

Dr. Siddarth Misra's Machine Learning course was of great help as it built the foundation of my machine learning knowledge and solidified my programming skill in Python.

I would also like to thank my friends in the research group – Gongsheng Li, Smith Leggett, Jin Tang, Julia Pakhotina, Shohei Sakaida, Tohoko Tajima, Gabriel Tatman, and Yuhai Zhou. Despite my short-spent time in College Station due to the pandemic, I enjoyed all the inspiring conversations.

My gratitude is also due to Dr. David Castineira, Dr. Sebastien Matringe, and Dr. Nansen Saleri. Their constant encouragement has helped me grow as a young professional.

Thanks to Dr. Larry Lake for believing in me.

Lastly, I would like to thank my beloved husband, Rui Li, who has always supported my life choices.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Dr. Ding Zhu, Dr. A. Daniel Hill of Harold Vance Department of Petroleum Engineering, and Dr. Zenon Medina-Cetina of Zachry Department of Civil and Environmental Engineering.

The strain rate simulation model used to produce input data for the synthetic case was developed by Dr. Jin Tang. Smith Leggett shared the method to process field DAS data, which was provided by Marathon Oil.

All other work described in this thesis was conducted by the student independently.

Funding Sources

This research work was fully funded by the United Stated Department of Energy Office of Fossil Energy under grant number DE-FE0031579 (The Eagle Ford Shale Laboratory: A Field Study of the Stimulated Reservoir Volume, Detailed Fracture Characteristics and EOR Potential). Contents of this study are solely the responsibility of the author and do not necessarily represent the official views of the Department of Energy.

NOMENCLATURE

A	Activation function result
b	Bias unit
f	Filter size
G	Magnitude of image intensity gradient from Sobel filter
G_x, G_y	Horizontal and vertical image intensity gradients from Sobel filter
g	Activation function
$h_{w,b}$	Perceptron as a function of weight and bias
J	Loss function
m	Sample size
n_{ave}	Average of the numbers of neurons of the previous and current layers
$n_{h,w}^{[l]}$	Shape of output (height and width) of a convolutional layer l
p	Padding size
p^i	Probability of sample i belonging to the target class
p_k	Probability of sample belonging to class k
s	Stride size
s_b, s_W, s_θ	RMSprop terms
t	Time step
v	Exponentially weighted moving average
W, w	Weight parameter in matrix and vector forms
x	Neural networks input

$y^{(i)}$	True value for sample i
$\hat{y}^{(i)}$	Predicted value for sample i
Z, z	Weighted sum of input plus bias in matrix and vector forms
α	Learning rate
β	Momentum hyperparameter
ϵ	A tiny number that avoids division by zero
θ	Model parameters
σ	Sigmoid function
σ^2	Variance

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1. Distributed Fiber Optic Sensing

Most well-known for the telecommunication industry, fiber optics as a means of reliable data transmission is not new to us. It is widely used in industries including defense, security, and transportation (Molenaar et al., 2011). The oil and gas industry also has adopted fiber optic cables as an instrument for detecting pipeline leakage (Tanimola and Hill, 2009) and monitoring hydraulically stimulated (HFS) wells (Molenaar et al., 2011).

Completing long horizontal wells with hydraulic fracturing is capital intensive. Operating companies strive to optimize completion designs to get the most value of the investment. Distributed fiber optic sensing gives operators the ability to monitor activities during fracturing, flow-back, and to investigate cross-well communication. Standard fiber optic cables are usually installed outside of the casing or inside the tubing, with a surface interrogation unit to receive and process the monitored data. When laser pulses are sent from the interrogation unit, the system measures the disturbance in the backscattered light caused by changes in temperature, strain, and acoustic intensity where the disturbance occurs along the fiber.

Fiber optic sensing works based on the properties of light scattering. There are three types of fiber optic sensing based on the frequency of the signal and the discrete peaks within the electromagnetic spectrum. They are Rayleigh, Raman, and Brillouin backscattering, as illustrated in Figure 1.1 (Tanimola and Hill, 2009).

Distributed temperature sensing (DTS) uses the difference in the Stokes and anti-Stokes components of Raman backscattering to infer local temperature changes in the fiber. Distributed acoustic sensing (DAS) measures Rayleigh backscattering to infer vibration-induced disturbance (Molenaar et al., 2012). Distributed strain sensing (DSS) uses strain sensitive Brillouin scattering to provide strain measurements (Tanimola and Hill, 2009).

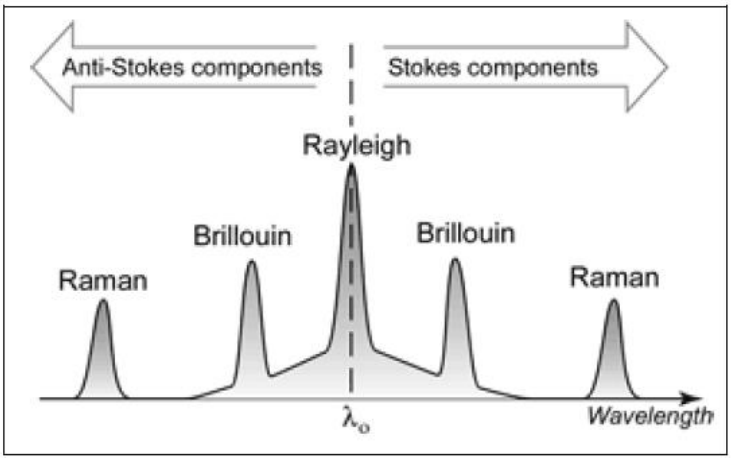

Figure 1.1 Different components of light scattering (reprinted from Molenaar et al., 2012).

Due to natural random inhomogeneity of the fiber optic cable, when light pulses travel in the fiber, a small fraction of the light pulses is reflected back by imperfections much smaller than the wavelength of the light. This is referred as Rayleigh backscattering. The DAS system then uses optical time domain reflectometry (OTDR) to measure changes in the backscattering. The measurement is made in the form of intensity changes or phase changes. The system is illustrated in Figure 1.2.

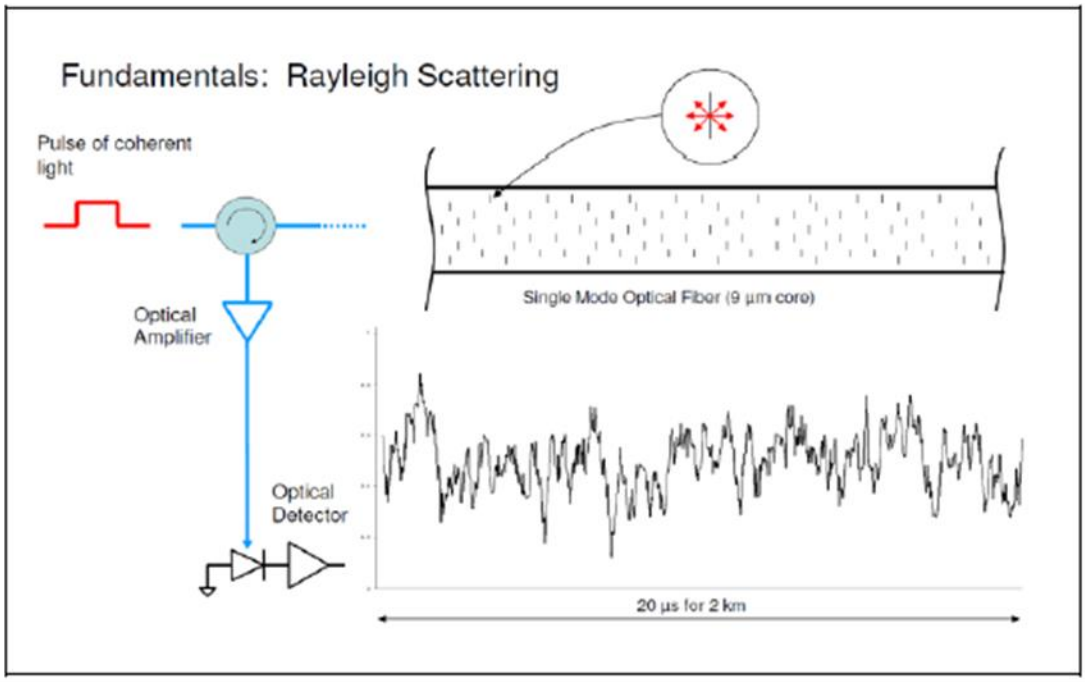

Figure 1.2 Schematic of Rayleigh scattering used in the DAS system (reprinted from Molenaar et al., 2012).

Many authors have demonstrated various applications of fiber optic sensing during or after hydraulic fracturing operation. The real-time sound vibration from DAS reflects activities such as tools traveling up and down the wellbore, ball seating, packer setting, and perforation shots (Molenaar et al., 2012). Such capability allows operators to change the course of action on the fly if needed. In the past few years, efforts have been made to interpret DTS and DAS data for post-job well stimulation diagnostics. By cross-examining treatment information, operators use DAS and DTS data to analyze fluid distributions and cluster efficiency for diagnosing and optimizing completion designs (Molenaar and Cox, 2013 and Ugueto et al., 2016). In addition to observation-based analyses, model-based interpretations are also achieved. Pakhotina et al. (2020)

presented a joint interpretation of DAS and DTS for multi-fracture fluid distribution based on experimental and numerical models. For cross-well analysis, the low-frequency band (<0.05Hz) of DAS reflects the discernable deformation of the cable caused by propagating fractures from hydraulically stimulated wells, hence providing a basis for fracture geometry estimation (Jin and Roy, 2017 and Haustveit et al., 2020).

As the applications grow, the need for effective and timely analysis of data generated by fiber optic sensing has become imperative. Furthermore, the technology produces enormous amounts of data with high spatial and temporal resolutions. This dense coverage of data allows for machine learning techniques to be used to enhance the value of data interpretation. Jin et al. (2019) demonstrated using a multi-layer perceptron neural network to predict fracture-hits from low-frequency DAS data. In their work, DAS data were divided into sections according to completion stages and pumping time windows. Input features for the neural network were deliberately designed variables based on the strain rates of each section. Then channels were selected to create a labeled training data set for neural network classification. Their model scored f1 scores of 0.86 on the validation data, and 0.79 on the test data. Stork et al. (2020) used a convolutional neural network (CNN) object detection algorithm called YOLO (you only look once) to identify and locate microseismic events in images of DAS data sampled at 2000 Hz. These studies demonstrated the feasibility of using machine learning to detect events from fiber optic sensing data.

1.2. Problem Statement and Research Objectives

1.2.1. Problem Statement

Low-frequency DAS strain rate measurement has gained attention in recent years for its ability to provide information about hydraulic fracture geometries (Ugueto et al., 2019). In developing unconventional resources, it is crucial to understand the extent of fracture propagation in order to optimize well completion and maximize capital efficiency. Fracture-hit is a direct indication of fracture-induced cross well communication. Surface or downhole pressure gauges can detect fluid communication caused by hydraulic fracturing in real time, but pressure monitoring is unable to provide precise location of the hit. Fluid communication can also result from fracture-to-fracture connection, rather than fracture-to-well connection.

Jin and Roy (2017), Ugueto et al. (2019) and Haustveit et al. (2020) demonstrated the ability of low-frequency DAS to detect a propagating fracture intersecting a monitoring well equipped with fiber optic sensing cable. When a transverse fracture reaches the monitoring well, the fracturing opening causes axial deformation of the fiber optic cable that can be reflected on low frequency DAS data. The data is essentially the phase changes of the backscattered light plotted over time and over a depth range. On the 2-dimensional representation, low frequency DAS signal shows either extension or compression as the fiber optic cable slightly deforms. When the fracture arrives, the data reflects the cable deformation by showing extension at the fracture-hit location. Once the pumping stops, the fracture starts to close, and the fiber responds with relaxation (Ugueto et al., 2019). This process is illustrated by Figure 1.3.

Moreover, the arrival of the fracture is indicated by a cone-shaped extension zone right before the fracture hits the monitoring well. Researchers have been using this cone-shaped strain rate front pattern to identify fracture-hits. For example, in Figure 1.4, the pattern can be seen as a new fracture arrives (Jin and Roy, 2017). However, the identification of the fracture-hit patterns from low-frequency DAS is mostly done through visual inspection after the stimulation operation has been completed. It would be advantageous to be able to identify fracture-hits automatically during the operation so that operators can respond more promptly.

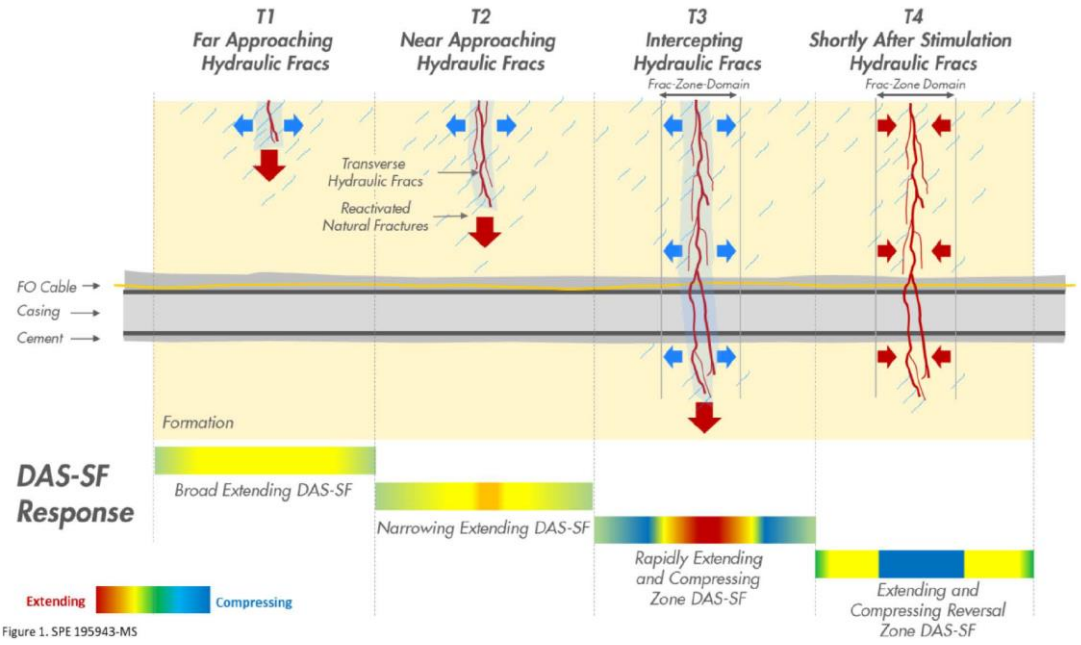

Figure 1.3 Illustration of low-frequency DAS response to fracture propagation (reprinted from Ugueto et al., 2019).

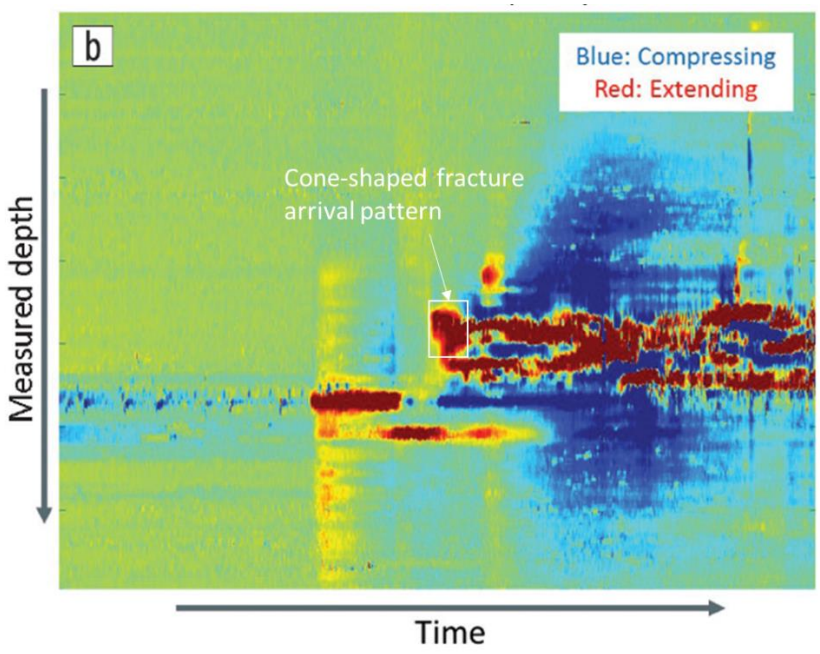

Figure 1.4 Example of low-frequency DAS signal showing fracturing arrival pattern (modified from Jin and Roy, 2017).

1.2.2. Research Objectives

Based on the characteristic pattern of fracture arrival, in this study, we investigate the feasibility of training convolutional neural network models to recognize the presence of fracture-hit events from low-frequency DAS strain rate data, as well as the location of fracture-hit events in terms of time and depth. A trained CNN model can be deployed to identify fracture-hit events automatically and systematically in real time, allowing operators to take actions more promptly with confidence.

The study is conducted in two phases. In phase one, we use data generated by a fracture propagation strain response model to train CNN models. With the proof of concept provided by phase one, we proceed to phase two, where field data is used to produce samples to train CNN models for the same purpose – recognizing and locating

fracture-hit events. The following procedure is executed to accomplish this goal. Figure 1.5 illustrates the workflow of this study.

- Run fracture propagation strain response models with various completion configurations to obtain sufficient data samples. Data samples should be labeled with classes (fracture-hit events or non-fracture-hit events). Fracture-hit samples should be annotated with locations for training purpose.
- Develop CNN models with the objectives of classifying and localizing events. Start with simpler model architecture but also investigate well-known CNN models.
- Process field DAS data to produce training samples. The data set should be analogous to the simulated data in the sense that each sample should be a 2D segment of DAS data in which a fracture-hit event could appear. All samples should be properly labeled and annotated.
- Develop CNN models to achieve the same classification and localization objectives of fracture-hit events with the expectation that the model architecture used for the field data should be more complex than the one used for the simulated data.

From data processing to model development, the code is completely written in Python, with the CNN models developed in Keras framework. Since training CNN models requires high computer performance, GPU (graphics processing unit) enabled Google Colaboratory (https://colab.research.google.com/) is used as the model development environment. Google Colaboratory is an online Python platform.

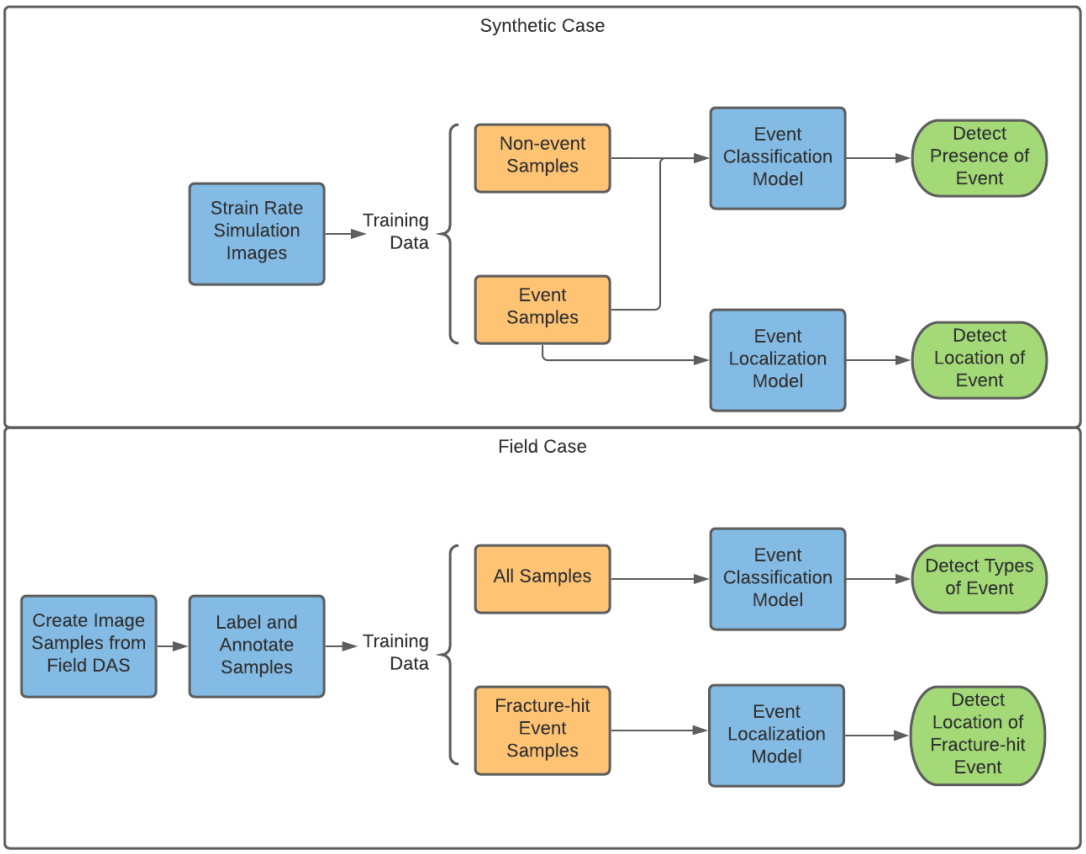

Figure 1.5 Workflow of using CNN to classify and localize fracture-hit events in low-frequency DAS.

2. METHODOLOGIES

2.1. Artificial Neural Networks and Convolutional Neural Networks

This section explains the key principles of artificial neural networks and convolutional neural networks. It is necessary to understand how artificial neural networks learn to perform a task. Elements of convolutional neural networks will be described, and why they excel at visual tasks will be explained. This section also covers some key techniques in training convolutional neural networks.

In machine learning, we teach the model what we want it to learn with relevant data so that the model can make a recognition when it sees new unidentified data. A model trained to identify categories (discrete values) is called a classifier. A model trained to make a continuous prediction is called a regressor. This is referred as supervised learning in the sense that the model has been trained with data that has corresponding responses. Among the numerous machine learning algorithms, artificial neural networks (ANN) excel at more complex tasks because they mimic the brain's cognitive system. Examples of these complex tasks include autonomous driving, speech recognition, and natural language processing.

2.1.1. Artificial Neural Networks

Inspired by the network of biological neurons in the brain, artificial neural networks are powerful machine learning models that have a wide range of applications, including search engines, recommender systems, fraud detection, medical diagnostics

and so on. Although ubiquitous today, research and development of ANNs have gone through some up and downs. The earliest ANN model was introduced by neurophysiologist Warrant McCulloch and mathematician Walter Pitts in 1943 (McCulloch and Pitts, 1943). However, due to the lack of computational power, the promises of ANNs were deemed unrealistic. In the 1980s, new developments sparked a wave of revival, but it was not until the 1990s did ANNs see substantial advancements thanks to the abundant digital data available today and the exponential growth in computing power (Géron, 2019).

2.1.1.1. Fundamentals of Artificial Neural Networks

In a nutshell, an artificial neural network (ANN) can be viewed as a structured and layered system composed of many linear regression units with the extension of some activation functions. The system parameters (weights and biases) are to be fitted through minimizing a certain cost function.

The simplest one-variable linear regression model can be represented by:

$$z = wx + b, \tag{2.1}$$

where x is the predictor variable, z is the response variable, and w and b represent the parameters (slope and intercept) of the straight line that fits the data represented by x. An example of this simple linear regression problem is predicting the price of a house based on a data set that contains the square footage (x) of m number of houses and their corresponding prices (y). Running linear regression will yield a pair of parameters w and

b that best describe the linear relationship between the square footage and price of a house.

In more generalized multi-variable linear regression, equation 2.1 can be represented as:

$$z(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots b, \tag{2.2}$$

or in vectorized form:

$$z(x) = w^T x + b. \tag{2.3}$$

In linear regression, finding the best parameters is done by minimizing the cost function (J), the squared distance between the actual value (y) and the corresponding predicted value ($\hat{y}$), through Gradient Descent. The cost function is defined as:

$$J(w, b) = \sum_{i=1}^{m} \left[y^{(i)} - \hat{y}^{(i)} \right]^2. \tag{2.4}$$

Superscript i indicates the ith sample.

In Gradient Descent, the parameters are updated iteratively in the direction of the steepest descent until the reduction of the cost function reaches a tolerance level.

In an ANN model, linear regression is realized through the fundamental element, the neuron (or unit), which computes the weighted sum of its input then applies an activation function to produce an output. The output is passed to neurons in the next layer as their input. The neuron can be viewed as an extension of linear regression in the sense that the output from the linear part needs to be activated or de-activated. Whether a neuron is activated depends on the neuron value and the specific activation function

used. More details on activation functions will be discussed in sections 2.1.1.2 and 2.1.3.1.

The earliest artificial neural network model is the perceptron model introduced by Frank Rosenblatt in 1958 (Rosenblatt, 1958). The perceptron model is a single layer ANN composed of linear threshold units (LTU). The LTU uses a step function as the activation function applied to the linear combination of its inputs ($z = w^T x$), as shown in Figure 2.1. The step function outputs 1 if z is above or equal to a threshold value (typically 0), and it outputs 0 if z is below the threshold. A perceptron has several LTUs and every LTU is connected to all the input units. Additionally, a bias unit is added. The bias is analogous to the intercept in linear regression. Equation 2.5 is the mathematical representation of a perceptron:

$$h_{w,b}(x) = g(w^T x + b). \qquad 2.5$$

In the equation above, x represents the input vector, w represents the weights applied to the input, and b is the bias unit. Function g is the activation function (step function in an LTU).

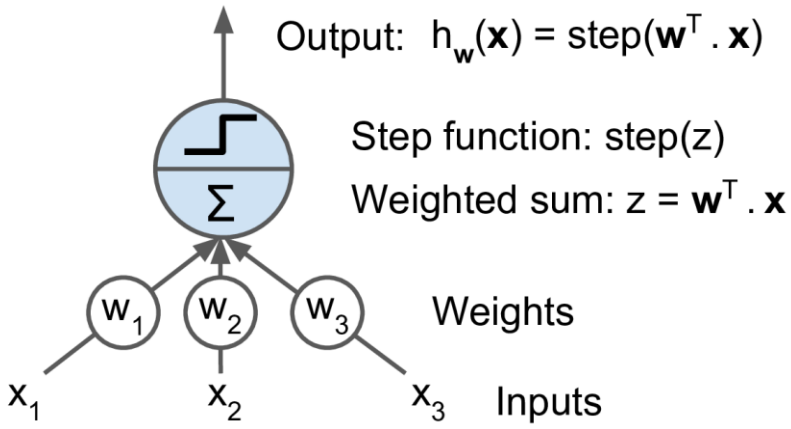

Figure 2.1 Schematic of a linear threshold unit, an artificial neuron (reprinted from Géron, 2019).

A perceptron by itself is not particularly useful. To utilize its power, researchers developed multilayer perceptron (MLP) models, which perhaps are the most represented ANN model in the literature. A MLP model contains more than one layer of neurons, and they are referred as the hidden layers. With stacked layers of neurons, MLPs can be trained to make predictions for complex systems. In general, more complex problems require more layers of neurons. Instead of using a step function, MLPs use activation functions. Activation functions give the network system nonlinearity, otherwise a multi-layer network is equivalent to a single layer network. In theory, activation functions also need to be continuous with non-zero gradients defined everywhere to make Gradient Descent work.

A MLP model contains an input layer (layer 0), several hidden layers, and an output layer. In each layer, every unit (or neuron) is connected to every single unit in the next layer to indicate their impact on units in the next layer. The number of units in the hidden layers generally depend on the dimensions of the data and the complexity of the

problem. The number of units in the input layer is the same as the number of predictor variables (features) of the input data. The number of units in the output layer depends on the objective of the task. For example, a binary classifier and a single variable regressor need one unit in the output layer. For a multi-class classifier, the number of units of the output layer needs to be the same as the number of classes.

In the real world, most problems are nonlinear. For example, instead of using simple linear regression, MLP can be used to better predict the price of a house. A house has attributes such as the size, number of rooms, year built, and location. A MLP model can be constructed with four input units representing the four attributes and one output unit representing the price, as illustrated in Figure 2.2. The hidden units can be viewed as secondary or implicit attributes that indicate the value of the house. The arrows indicate the connections of units from the previous layer to units in the next layer. They represent the impact each unit has on units in the next layer. The levels of impact are characterized by the weight parameters.

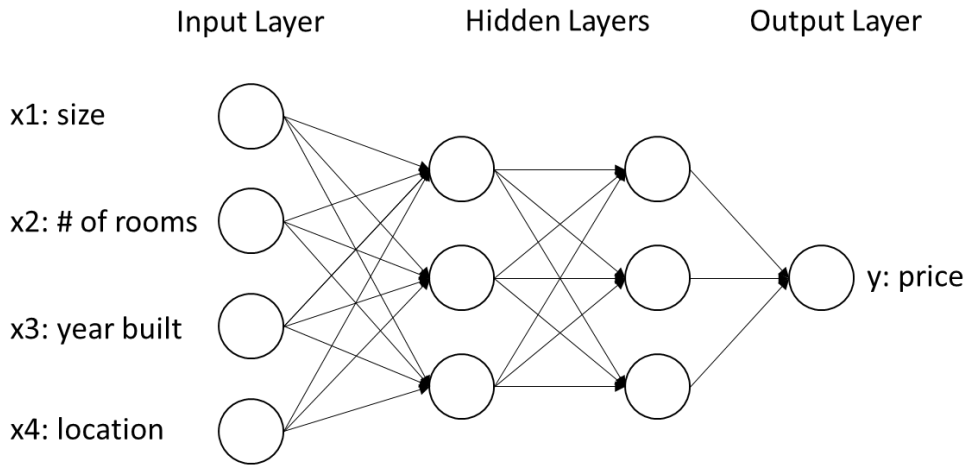

Figure 2.2 Architecture of a MLP model to predict the price of a house.

In a neural network model, the weights and biases are the trainable parameters that make the model produce the best prediction. The training process is done through the *forward pass and back propagation* algorithm, the core of training artificial neural networks. The algorithm is described below.

- The input data is passed to the network's input layer, and the neurons calculate the corresponding output layer by layer till the output layer of the network is computed (a prediction is made).
- Once the output has been computed, the error of the prediction defined by a certain cost function can be measured.
- Error gradients (gradients of the cost function) with respect to each model parameter are calculated layer by layer backwards using the chain rule.
- Parameters are updated by Gradient Descent using the calculated error gradients.

In summary, the calculations (weighted sum and activation) are carried forward layer by layer, and the error gradients determined by the cost function are carried backward layer by layer to update the system parameters until the parameters are fitted.

The mathematical formulation of the forward pass calculation of layer l is represented by equation 2.6 and equation 2.7.

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]} \tag{2.6}$$

$$A^{[l]} = g^{[l]}\left(Z^{[l]}\right) \tag{2.7}$$

In the equations above, $A^{[l-1]}$ is the input matrix of layer l (or output from layer $l-1$), and W is the weight matrix in the shape of (n, m) where n is the number of neurons in the current layer and m is the number of neurons in the previous layer. The weight parameters make the connections between neurons in each layer. Z represents the result of the linear combination of the input matrix of layer l ($A^{[l-1]}$) plus the bias vector b. For the input layer, $A^{[0]} = X$, where X represents the model's input matrix. Figure 2.3 demonstrates a concrete example.

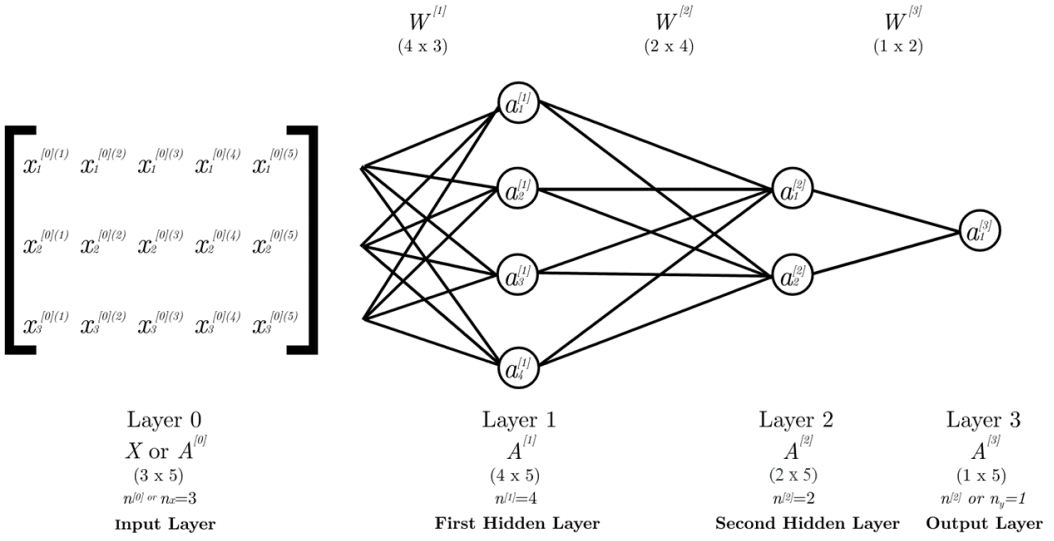

Figure 2.3 Example of a multilayer perceptron with corresponding notations (modified from Script Reference, 2021).

In the example shown in Figure 2.3, there are two hidden layers with four units and two units, respectively. The input data has three variables and five training samples. The output layer has one unit. The notation can be described as follows.

- The superscripts in square brackets indicate layer indices, superscripts in parenthesis indicate sample indices, and subscripts indicate unit indices in the layer. For instance, $x_1^{[0](2)}$ represents the second sample (2) of the first input variable (1) in the input layer (0).
- $n^{[l]}$ indicates the number of units in layer l. For the input and output layers, they can also be represented as n_x and n_y.
- In this network, there are three input variables and five training samples, hence $n^{[0]} = 3$, and the shape of X is 3×5. In layer 1 (the first hidden layer), there are four units, hence $W^{[1]}$ has the shape of (4×3). As a result, the output of layer 1, $A^{[1]}$, has the shape of (4×5). $A^{[1]}$ is then fed into layer 2 to obtain $A^{[2]}$. Eventually, the output of the network, $A^{[3]}$, is calculated.

After the forward pass, the error can be measured by the cost function, $J(Y, A)$, which is a function of the true value Y and the model output A. The specific cost function depends on the type of the learning task.

In backpropagation, because gradients are needed in order to update the model parameters in Gradient Descent, the partial derivatives of J w.r.t W and b for each layer are calculated using the chain rule, as shown by equations 2.8 and 2.9.

$$\nabla_W J = \frac{\partial J}{\partial A}\frac{\partial A}{\partial Z}\frac{\partial Z}{\partial W} \tag{2.8}$$

$$\nabla_b J = \frac{\partial J}{\partial A}\frac{\partial A}{\partial Z}\frac{\partial Z}{\partial b} \tag{2.9}$$

The exact expressions for the partial derivatives depend on the activation function and cost function used.

Finally, the weights and biases can be updated by:

$$W \leftarrow W - \alpha \nabla_W J, \tag{2.10}$$

and

$$b \leftarrow b - \alpha \nabla_b J, \tag{2.11}$$

where α is the learning rate. The entire process is repeated iteratively until model parameters W and b have been fitted (error no longer reduces).

It is important to note that for each layer, the weight parameters must be initialized randomly so that the units bear different weights. If the units have the same weight, then having many units is meaningless. Initializing weights randomly is referred as "breaking the symmetry".

2.1.1.2. Activation Functions

To provide the network nonlinearity, and to predict a discrete output for the final layer (for classification), ANN must use activation functions. Without activation function, stacked layers are equivalent to a single layer. The most common activation function is the sigmoid function (or the logistic function), which is defined as:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{2.12}$$

with

$$z = w^T x + b. \tag{2.13}$$

The sigmoid function outputs a probability between 0 and 1, as shown in Figure 2.4. For binary classification, the final output is 1 if the probability is greater or equal to 0.5, 0 if the probability is less than 0.5.

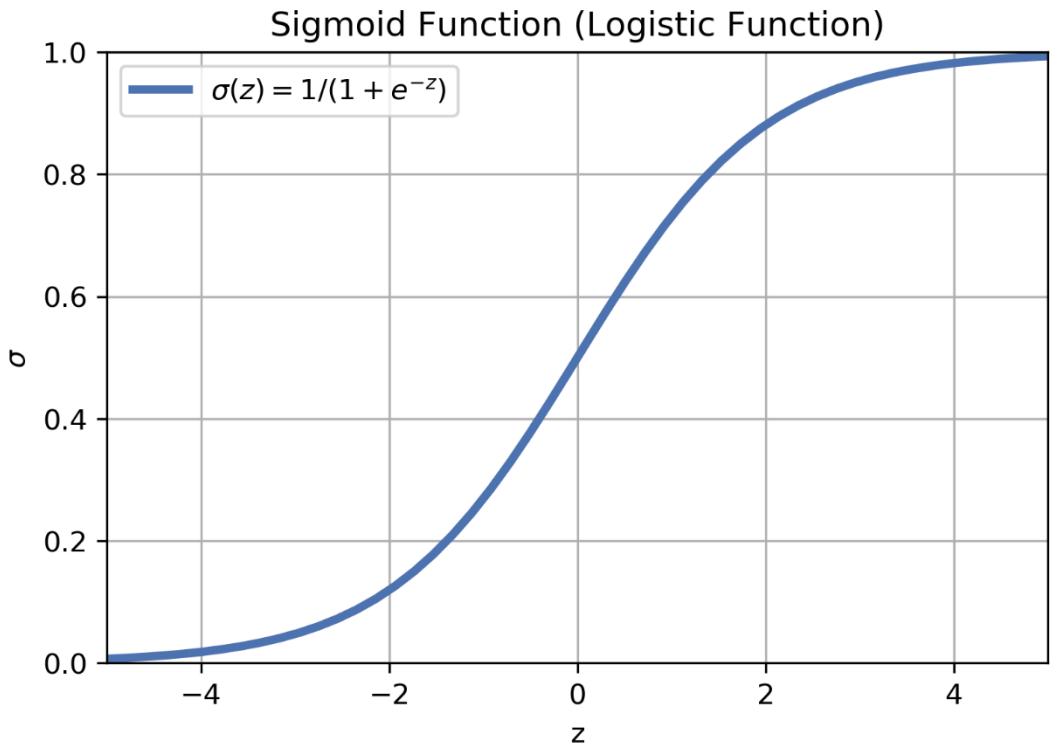

Figure 2.4 The sigmoid activation function (logistic function).

The hyperbolic tangent function (tanh) is also s-shaped like the sigmoid function, but its value ranges from -1 to 1 instead of 0 to 1 (Figure 2.5). As a result, the output of this function is usually centered around 0, making model converge more easily. The tanh function is defined as:

$$\tanh(z) = 2\sigma(2z) - 1. \qquad 2.14$$

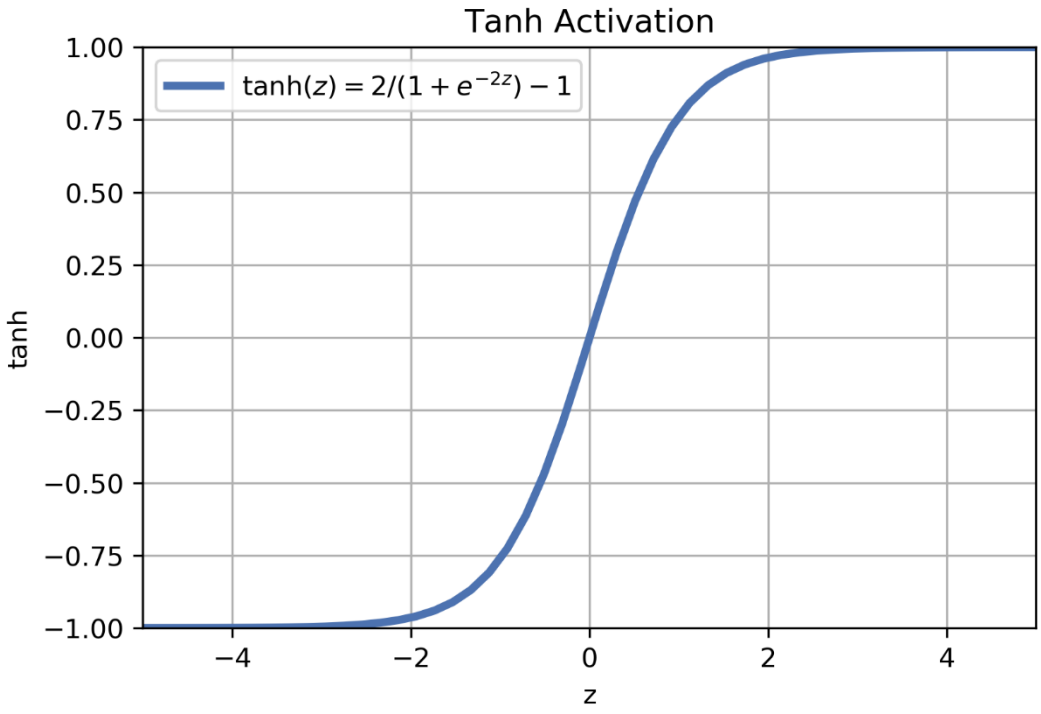

Figure 2.5 The hyperbolic tangent (tanh) activation function.

When a model is trained to recognize multiple classes, the output layer is activated by the softmax activation function. The softmax activation computes the probability of a sample belonging to each class. As usual, the linear combination is computed first:

$$z = w^T x + b. \tag{2.15}$$

Then the softmax function computes the probabilities according to:

$$p_k = \sigma\big(z(x)\big)_k = \frac{\exp\big(z_k(x)\big)}{\sum_{j=1}^{K} \exp\big(z_j(x)\big)}. \tag{2.16}$$

The final class prediction is made by:

$$\hat{y} = argmax\ (p_k). \qquad 2.17$$

In the equations above, K is the number of classes, p_k is the probability of sample x belonging to class k, and $\hat{y}$ is the predicted class, class k that maximizes the probability p_k.

2.1.1.3. An Image Classifier with MLP

To demonstrate how MLP works, this section details the process of developing an MLP model to recognize clothing items in the Fashion MNIST data set. Fashion MINST is one of the built-in data sets in Keras. Figure 2.6 shows some examples of items in the data set. In the Fashion MNIST data set, each sample is a grayscale image of size 28×28 and contains one fashion item. There are 60,000 training images and 10,000 testing images. Each image is labeled with one of the ten clothing item classes: “T-shirt/top”, “Trouser”, “Pullover”, “Dress”, “Coat”, “Sandal”, “Shirt”, “Sneaker”, “Bag”, and “Ankle boot”. In the data set, the classes are coded as numbers from 0 to 9.

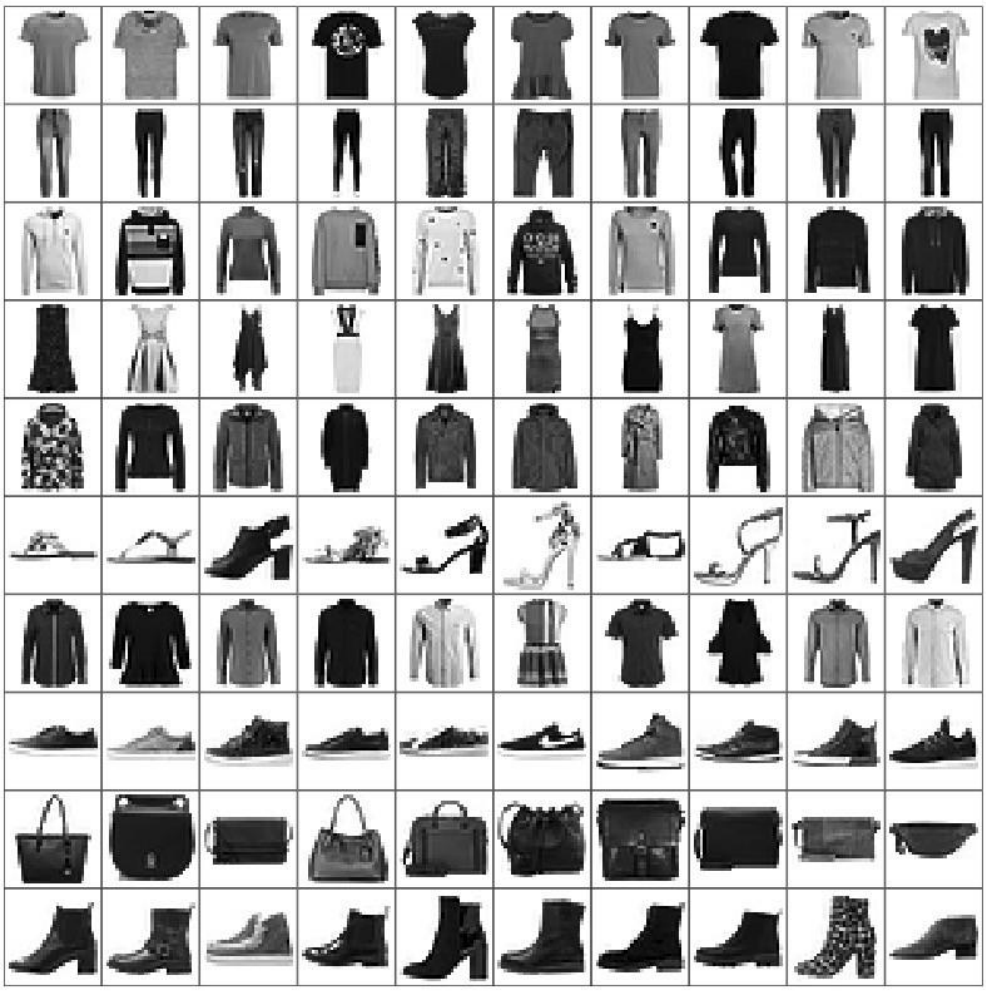

Figure 2.6 Examples in the Fashion MNIST data set. Each image example contains one fashion item.

Because each layer in an MLP is a 1D array of neurons (flat layers), the image needs to be converted to a 1D vector of size 784. Hence the input layer will have 784 neurons. The output layer will have ten neurons, one for each class. For this example, we use two hidden layers with 200 and 100 neurons, respectively. The model architecture is displayed in Table 2.1.

The total number of parameters in an MLP model is the sum of total number of neuron connections and the number of all bias units. For each layer, the number of parameters is $n \times m + n$, where n is the number of neurons in the current layer and m is the number of neurons in the previous layer. Hence in this network, the total number of parameters to be trained is:

$$(200 * 784 + 200) + (100 * 200 + 100) + (10 * 100 + 10) = 178110.$$

Table 2.1 Architecture of the MLP model used for the Fashion MNSIT data classification.

Layer (type)	Output Shape	Param #
input (Flatten)	(None, 784)	0
dense0 (Dense)	(None, 200)	157000
dense1 (Dense)	(None, 100)	20100
output (Dense)	(None, 10)	1010
Total params: 178,110		
Trainable params: 178,110		
Non-trainable params: 0		

After the model has been constructed, it can be then used to fit the training data. The training process is an iterative optimization process with the objective of minimizing the loss. Hence the history of loss should generally decrease and the history of model performance (some accuracy measure) should increase. An optimal model should be one that does not overfit nor underfit the training data. An overfitted model tends to "memorize" the training data and performs poorly on the validation data. An underfitted model is too general and does not capture the characteristics of the data. Keeping track of model performance on both training and validation data allows the data scientist to tune the model to the optimal state.

Figure 2.7 shows the training history of the MLP model on the Fashion MNSIT data set. As expected, the loss decreases for both training and validation data whereas the accuracy increases for both training and validation data.

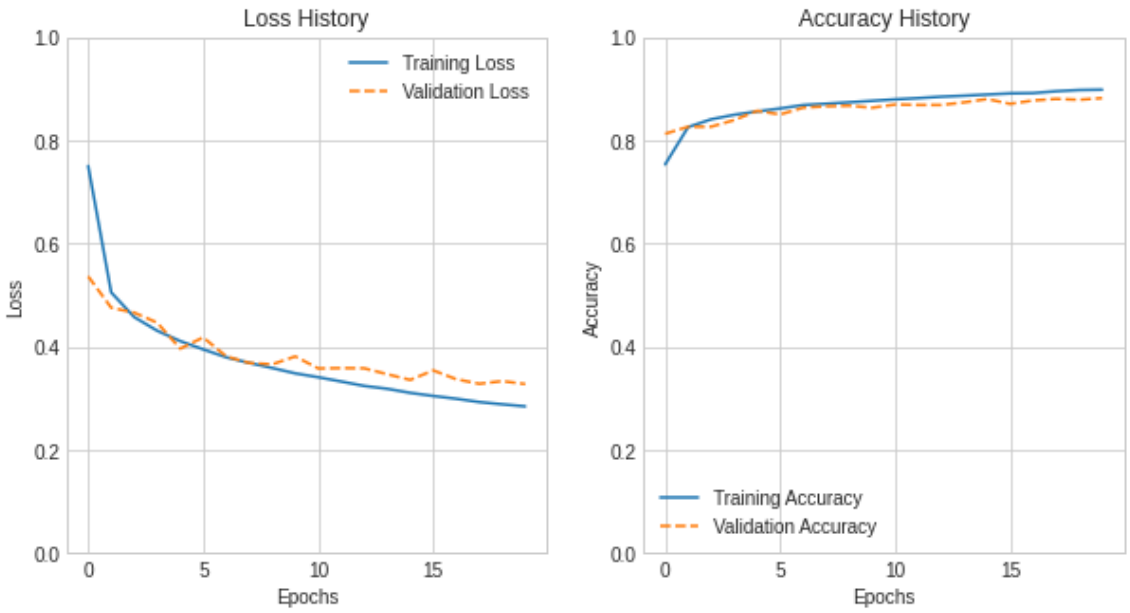

Figure 2.7 Training history of the example MLP model.

The trained model can then be used to make a prediction on the testing data. Figure 2.8 shows the first five samples of the testing data, the true label of each sample and the predicted label of each sample. The model correctly identifies all the five samples.

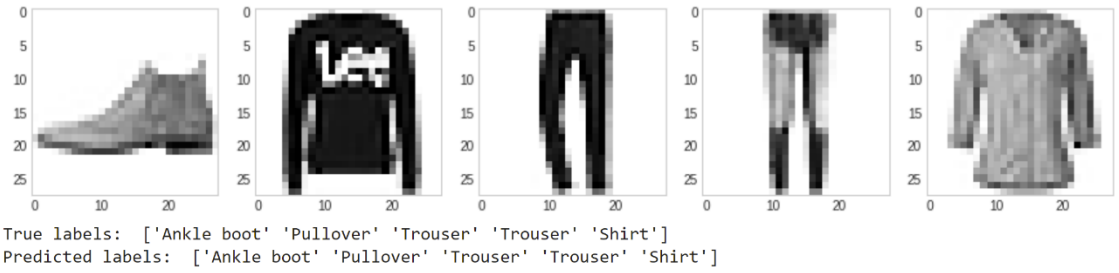

Figure 2.8 Samples from the Fashion MNIST testing data with corresponding true and predicted classes.

2.1.2. Convolutional Neural Networks

Emerged from studies of the brain's visual cortex, convolutional neural networks (CNN) are the state-of-the-art technology used in computer vision. Regular MLP models can perform some visual tasks with small images. However, digital images can easily have millions of pixels. With fully connected neurons of each layer, the computational need is simply infeasible and inefficient. CNNs are developed based on the idea that the cognitive neuron of a layer does not need to be connected to every single neuron before and after. It only needs to be connected to a small receptive field of the previous layer.

The earliest inspiration was from studying of the structure of cats' visual cortex by neurophysiologists David H. Hubel and Torsten Wiesel in 1958 and 1959 (Géron, 2019). Their studies showed that many neurons in the visual cortex have a small local receptive field, and the neurons only react to visual stimuli located in this receptive field. The neurons with the receptive fields may react to different visual patterns (e.g., lines with different orientations). Moreover, the authors showed that some neurons have larger receptive fields that react to more complex patterns. These studies inspired the neocognitron model introduced by Kunihiko Fukushima in 1980 (Fukushima, 1980).

The neocognitron is a hierarchical multilayer artificial neural network. It is the predecessor of convolutional neural networks. The earliest well-known CNN is the modeled called LeNet-5 developed by Yann LeCun et al. (1989) after many years of research. LeNet-5 popularized CNN applications and was widely used to recognize handwritten characters by banks and postal offices.

2.1.2.1. The Convolutional Operation

The core of a CNN model is the convolutional operation. The operation takes a small 2D filter (or convolution kernel) and convolves it with an input image to produce an output image, called a feature map. The convolutional operation performs elementwise multiplication with an area in the input of the same size as the filter. The output is a single number that is the summation of the products. The operation continues to the next receptive field horizontally then vertically until the entire input has been covered. This process produces a 2D feature map as the output and it is usually smaller in size compared to the input. For example, convolving a 6×6 input with a 3×3 filter produces an output of size 4×4, as illustrated by Figure 2.9. In Figure 2.10, the filter has predefined values. It is a type of vertical edge detector. While training CNN models, the values of the filter are the weight parameters that are to be determined by the training process.

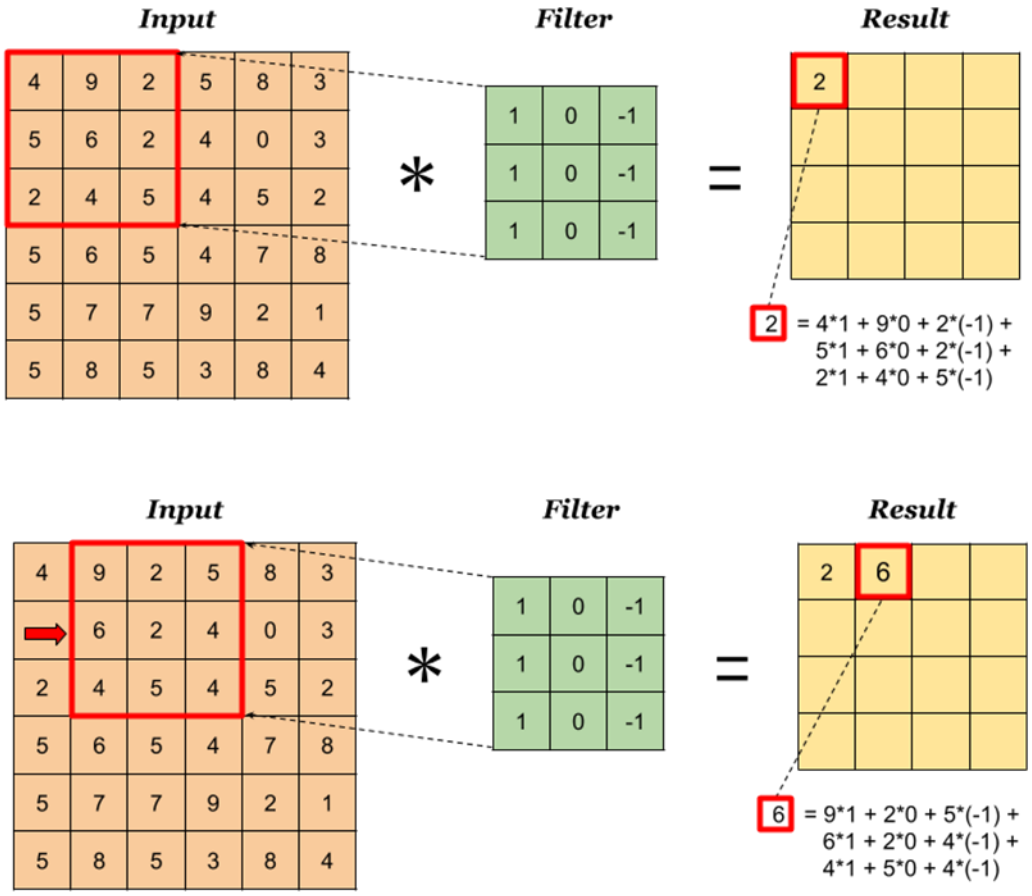

Figure 2.9 Illustration of the convolutional operation (reprinted from IndoML, 2021).

2.1.2.2. Stride and Padding

The filter can move pixel by pixel or skip some pixels. The step size by which the filter moves is called *stride*. Bigger strides result in smaller feature maps. It is possible to produce a feature map the same size as the input. This is done by *zero padding* – adding zeros around the input. In a convolutional layer l, the shape of the

output ($n_{h,w}^{[l]}$) is determined by the following expression with the shape of the input ($n_{h,w}^{[l-1]}$), filter size (f), strides (s), and padding (p):

$$n_{h,w}^{[l]} = \left\lfloor \left(\frac{n_{h,w}^{[l-1]} + 2p - f}{s} \right) + 1 \right\rfloor, \tag{2.18}$$

where subscripts h and w denote height and width. The brackets $\lfloor\ \rfloor$ indicate the mathematical *floor* operation. In practice, filters are square matrices and stride is the same in horizontal and vertical directions.

2.1.2.3. Convolution Over Volume

The convolutional operation is typically done over a 3D tensor since images are usually represented in RGB (red, green, and blue) channels. Correspondingly, the filter must also be 3D with the third dimension equal to the third dimension of the input. If only one filter is used, the output is still 2D because each output number is the sum of the results from all channels. Figure 2.10 illustrates the convolution over volume concept.

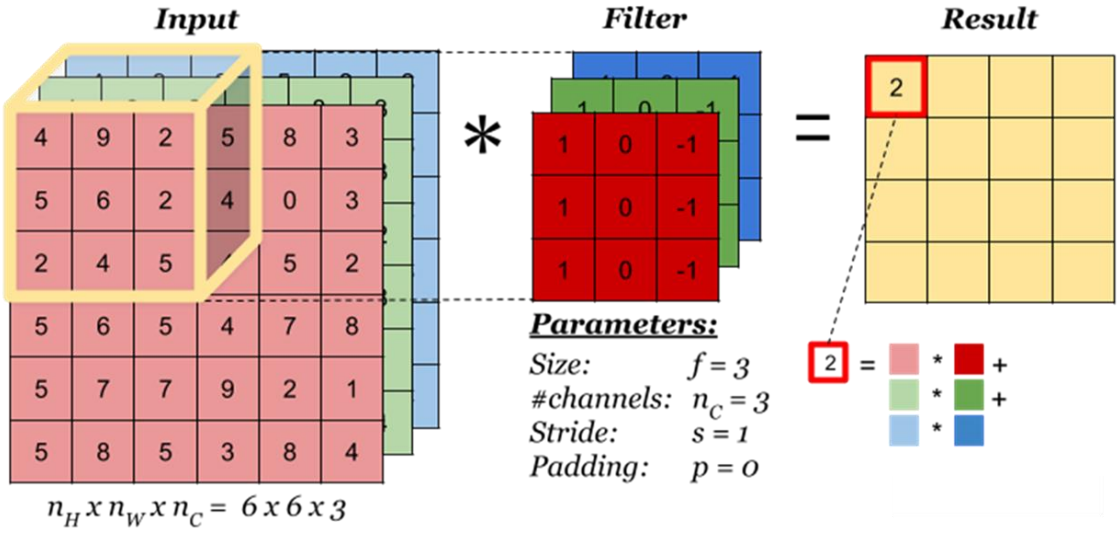

Figure 2.10 Illustration of convolution over volume (reprinted from IndoML, 2021).

2.1.2.4. One Convolutional Layer

In summary, a convolutional layer l has the following components:

- Filter size: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
- Padding size: $p^{[l]}$
- Stride: $s^{[l]}$
- Number of filters: $n_c^{[l]}$
- Input: $n_{\mathrm{h}}^{[\mathrm{l}-1]} \times n_{\mathrm{w}}^{[\mathrm{l}-1]} \times n_c^{[l-1]}$
- Output: $n_{\mathrm{h}}^{[\mathrm{l}]} \times n_{\mathrm{w}}^{[\mathrm{l}]} \times n_c^{[l]}$

Notice that the number of filters is the same as the number of channels of the output, and the third dimension of the filter size is the same as the third dimension of the input size (output size from previous layer).

A convolutional layer often also contains a *pooling layer*. A pooling layer operates similarly to convolution, but instead of computing the sum of the element-wise products, pooling layer computes the average or the maximal values within the pooling kernel. In modern applications, max pooling is more commonly used because it has the effect of reducing data dimensionality while preserving the most prominent feature.

A CNN model is usually a stack of several convolutional layers followed by a few flat layers. In a convolutional layer, each filter learns a specific feature across an image. For example, an early layer in the network can learn low-level features such as horizontal and vertical lines in the image. Compared to a classic fully connected multi-layer perceptron network with the same scale, a CNN model can extract relevant features of an image with fewer parameters to be trained. This is because neurons (pixels) in a feature map share the same trainable parameters (parameter sharing) and each output is only connected to a small number of inputs (sparsity of connection) (Géron, 2019). A typical CNN architecture is shown in Figure 2.11. In this example, input images containing single objects are fed into the network. The end goal is for the model to recognize the type of the object (i.e., car, truck, van, bicycle etc.)

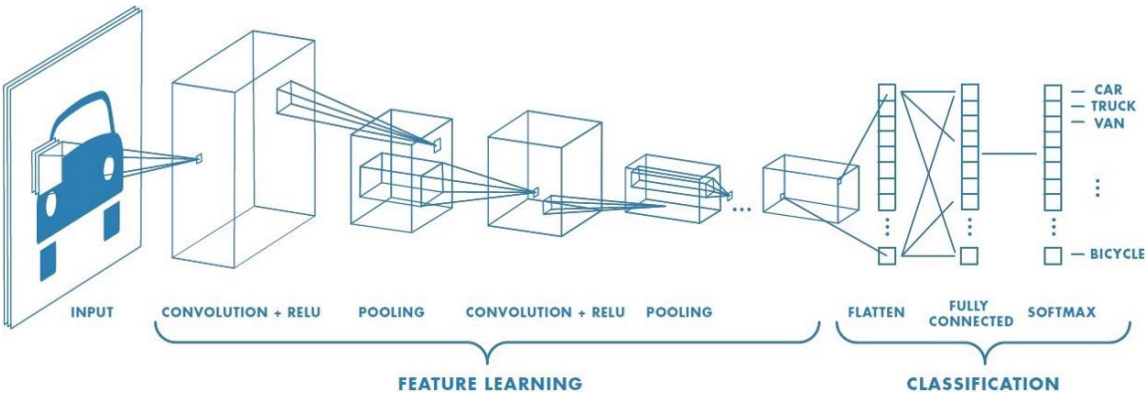

Figure 2.11 Example of a typical CNN architecture (reprinted from Prabhu, 2018).

2.1.3. Training Deep Neural Networks

Along with the powerfulness and freedom deep neural networks have, challenges arise in training a deep neural network models. This section describes the prevailing vanishing/exploding gradients problem and several techniques researchers developed to address it.

In backpropagation, the error gradients of each layer are calculated sequentially from later layers. Subsequently, the error gradients of the early layers can be exponentially small or exponentially large, depending on the initial values of the weight parameters. As a result, the model may never converge. This is referred as the vanishing or exploding gradients problem. There are a few techniques that tackle this challenge, such as using non-saturating activation functions, initializing weights with a certain variance, and normalizing data for each layer.

2.1.3.1. Non-saturating Activation Functions

Unlike the saturating sigmoid activation function, the rectified linear unit (ReLU) activation function has become the most popular activation function for training ANN models. The ReLU function is defined as:

$$ReLU(z) = \max(0, z) \tag{2.19}$$

As can be seen in Figure 2.12, the ReLU function does not have a gradient at $z = 0$ and the gradient is zero for $z < 0$. This turns out to work fine in practice. The key is that the ReLU function does not saturate for positive values.

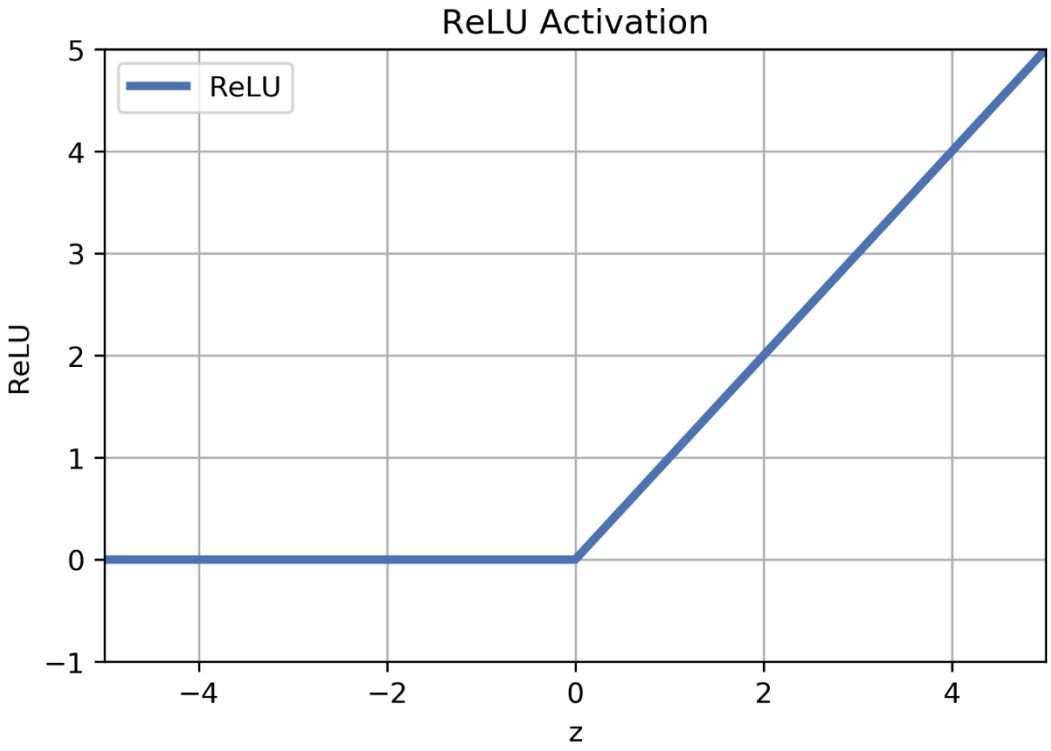

Figure 2.12 The ReLU activation function.

Although ReLU works well in practice, there are some improved variants of ReLU. Leaky ReLU (Figure 2.13) modifies the negative portion of ReLU so that it does not have a zero gradient. Leaky ReLU is defined as:

$$Leaky\ ReLU = \max(\alpha z, z), \tag{2.20}$$

where α is the slope of the negative portion, usually a very small number.

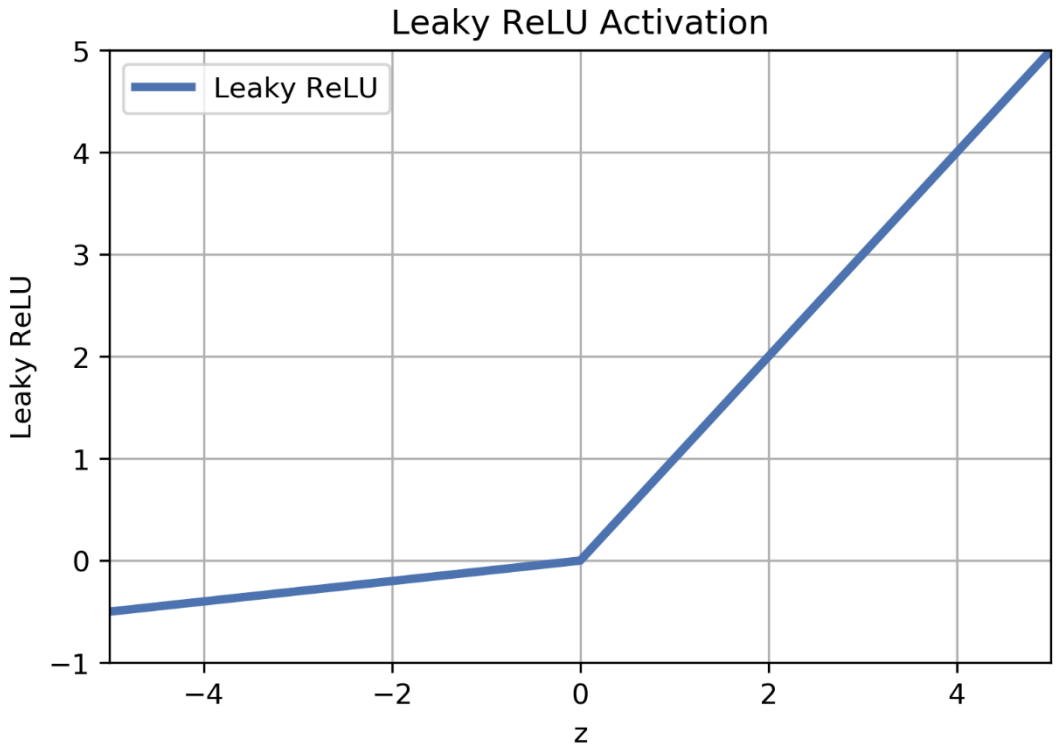

Figure 2.13 The Leaky ReLU activation function.

Exponential linear unit (ELU) also addresses the zero-gradient problem. It is defined as:

$$ELU_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1), & if\ z < 0 \\ z, & if\ z \geq 0' \end{cases} \tag{2.21}$$

with α typically set to 1. Figure 2.14 shows the ELU activation function.

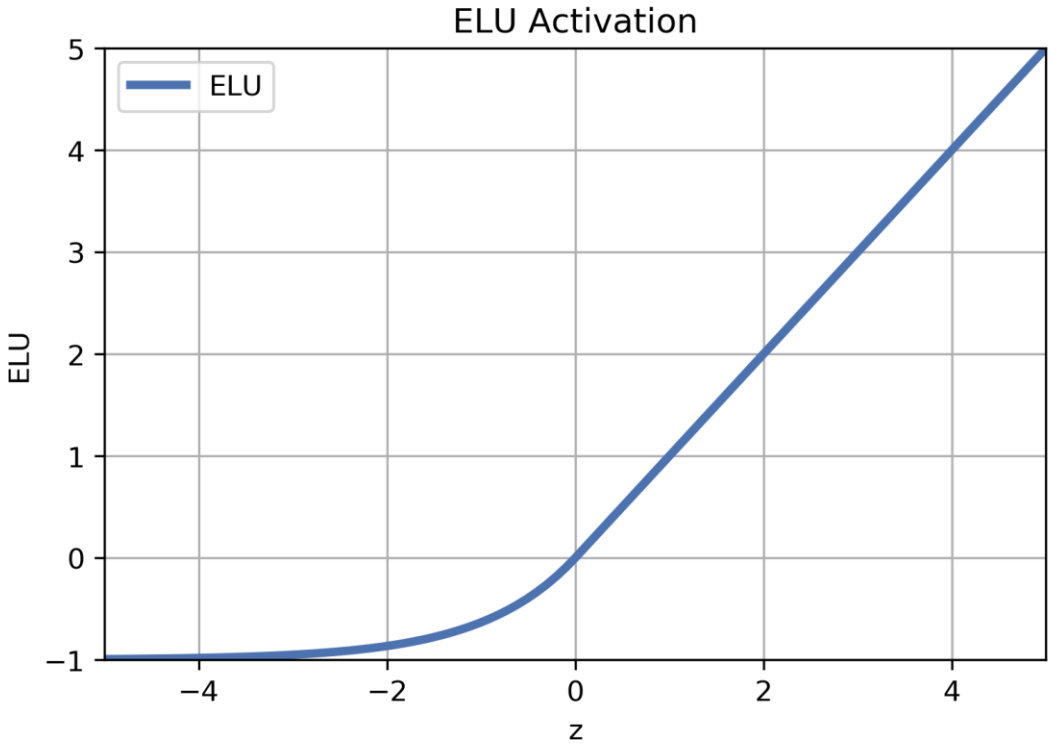

Figure 2.14 The exponential linear unit activation function with α set to 1.

2.1.3.2. Initializations

The vanishing/exploding gradients problem was first addressed by Xavier Glorot and Yoshua Bengio in 2010 (Glorot and Bengio, 2010). The authors pointed out that if the variance of the output of each layer is much bigger than the variance of the input, the sigmoid function will eventually saturate. When the inputs of the sigmoid function are very large or small, the gradients become extremely small, practically leaving nothing to be learned for the earlier layers. Glorot and Bengio (2010) stated that to alleviate this problem, the variance of the input and the variance of the output of each layer need to be

the same. Although this cannot be guaranteed since different layers usually have different numbers of neurons, the authors proposed a practical solution described below.

- Normal distribution with mean of 0 and $\sigma^2 = \frac{1}{n_{ave}}$ where n_{ave} is the average of the numbers of neurons of the previous and current layers.
- Uniform distribution between $-r$ and $+r$ where $r = \sqrt{\frac{3}{n_{ave}}}$

This is referred as the *Glorot initialization*. The Glorot initialization is suitable for sigmoid, tanh, and softmax activations. For ReLU, He et al.(2015b) suggest that using a variance of $\frac{2}{n_{ave}}$ with a normal distribution reduces model errors faster.

2.1.3.3. Batch Normalization

Another technique that drastically reduces the vanishing/exploding gradients problem is called *batch normalization*, introduced by Sergey Ioffe and Christian Szegedy in 2015 (Ioffe and Szegedy, 2015). This operation transforms the input so that it is zero centered and normalized, then scales and shifts each input. This can be done before or after each convolutional layer. It is called batch normalization because in practice, the operation is done using the current *mini-batch*. A mini-batch is simply a subset of the training data to be used to train the model. As opposed to using the whole data set, dividing the data into several mini batches allows the model to update parameters more frequently, and reduces the memory requirement. During testing, Keras's implementation of batch normalization estimates the means and standard deviations needed for the transformation using moving averages of the input layer's

mean and standard deviations during training (Géron, 2019). Batch normalization also has the effect of regularization.

2.1.3.4. Optimization Techniques

Although Gradient Descent is the standard way for optimization, it is not the most suitable for training deep neural networks because it can be painfully slow. In standard Gradient Descent, the parameters are updated by subtracting the gradient of the cost function multiplied a learning rate. If the gradient is small, then the model learns slowly. *Optimization with momentum* (Polyak, 1962) addresses this problem by keeping track of the history the gradient. This is achieved by using the exponentially weighted moving average of the gradients and using this averaged gradient to update the parameters. For step t, the exponentially weighted moving average (v) of a variable x is defined as:

$$v_t = \beta v_{t-1} + (1-\beta)x_t, \tag{2.22}$$

where β is the momentum hyperparameter usually set to 0.9 and v is initialized as 0 ($v_0 = 0$). The weight and bias parameters are updated as follows:

$$v_W \leftarrow \beta v_W + (1-\beta)\nabla_W J, \tag{2.23}$$

$$W \leftarrow W - \alpha v_W, \tag{2.24}$$

$$v_b \leftarrow \beta v_b + (1-\beta)\nabla_b J, \tag{2.25}$$

$$b \leftarrow b - \alpha v_b. \tag{2.26}$$

Optimization can be even faster with *RMSProp*, a non-published technique developed for a machine learning course on Coursera (https://www.coursera.org/). The RMSProp algorithm is similar to optimization with momentum, but it scales the gradient terms so that gradient decent moves faster along dimensions with flatter slopes. The weight and bias parameters are updated as follows:

$$s_W \leftarrow \beta s_W + (1-\beta)(\nabla_W J \otimes \nabla_W J), \tag{2.27}$$

$$W \leftarrow W - \alpha(\nabla_W J \oslash \sqrt{s_W + \epsilon}) \tag{2.28}$$

$$s_b \leftarrow \beta s_b + (1-\beta)(\nabla_b J \otimes \nabla_b J) \tag{2.29}$$

$$b \leftarrow b - \alpha(\nabla_b J \oslash \sqrt{s_b + \epsilon}) \tag{2.30}$$

In the expressions above, $\otimes$ and $\oslash$ represent element wise multiplication and division. ϵ is a tiny number that prevents dividing by zero. If the original gradient terms are steep, then adjusted gradients will be the result of dividing the original gradients by relatively large numbers. On the other hand, if the original gradients are small, then they are adjusted by dividing relatively small numbers. The combined effect is preventing the path of optimization from zig-zagging too much in the wrong directions. Figure 2.15 illustrates the idea of RMSProp with a two-parameter example.

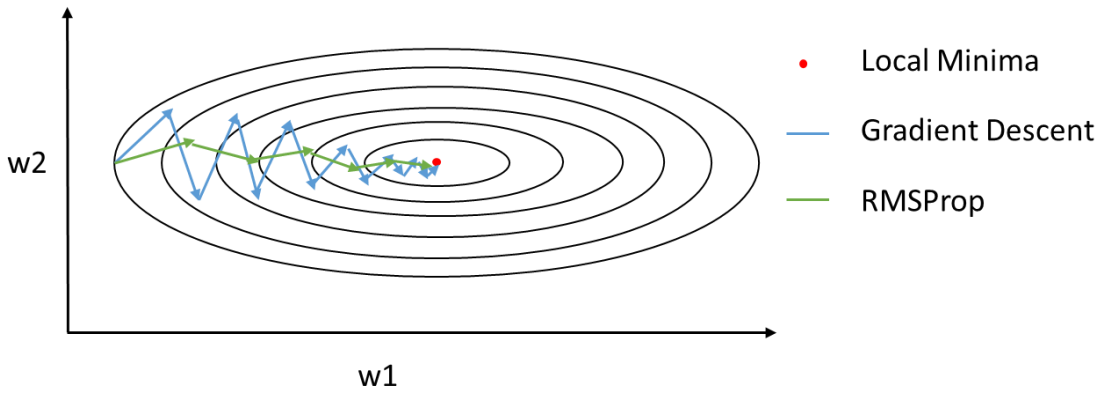

Figure 2.15 A two-parameter example demonstrating the difference between regular Gradient Descent and and RMSprop.

Adaptive moment estimation (Adam) combines the benefits of both Momentum Optimization and RMSProp (Kingma and Ba, 2015). The Adam algorithm uses the exponentially weighted averages calculated by both MomentumOoptimization and RMSProp. It then applies bias correction, and the parameters are updated using a combination of the corrected exponentially weighted averages. The update process is described by the expressions below. For brevity, θ is used to represent both weight and bias.

For iteration t:

$$v_\theta \leftarrow \beta_1 v_\theta + (1-\beta_1)\nabla_\theta J(\theta), \tag{2.31}$$

$$s_\theta \leftarrow \beta_2 s_\theta + (1-\beta_2)(\nabla_\theta J \otimes \nabla_\theta J), \tag{2.32}$$

$$v_\theta^{corrected} = \frac{v_\theta}{1-\beta_1^t}, \tag{2.33}$$

$$s_\theta^{corrected} = \frac{s_\theta}{1-\beta_2^t}, \tag{2.34}$$

$$\theta \leftarrow \theta - \frac{\alpha v_\theta^{corrected}}{\sqrt{s_\theta^{corrected} + \epsilon}}. \tag{2.35}$$

Because the momentum terms ($\boldsymbol{v_\theta}$) and the RMSProp terms ($\boldsymbol{s_\theta}$) are initialized at zero, they are biased towards zero. The purpose of the bias correction (equation 2.33 and equation 2.34) is to boost these terms at the beginning of the training.

2.2. Edge Detection

The simulated strain rate of a fracture-hit exhibits a well-defined pattern. This pattern is distinguished by a cone-shaped extension. Based on this characteristic, edge detection techniques can be applied to identify the cone-shaped strain rate front. The edge detection workflow consists of Sobel filtering, morphological transformation, and connected component labeling. This workflow may not be applicable to field data because the strain rate front is far less regular in shape compared to the simulated strain rate front. The purpose of using edge detection workflow is to compare the results with CNN models for fracture-hit location identification in simulated strain rate data.

2.2.1. Sobel Filter

In image processing, edge detection techniques can identify distinctive lines of an image. It is achieved by convolving predefined filters with an input image. This operation approximates the image intensity gradients at each pixel location. In this study, Sobel filter is chosen as the edge detection filter to find the strain rate front. The Sobel filter consists of one horizontal filter and one vertical filter that detect horizontal edges and vertical edges, respectively. The approximated gradient results G_x and G_y are obtained by convolving these two 3×3 filters with an input image A:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \tag{2.36}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A. \tag{2.37}$$

The results are combined to produce the magnitude of the gradient defined as:

$$G = \sqrt{G_x^2 + G_y^2}. \tag{2.38}$$

2.2.2. Morphological Transformation

If the input image is noisy, the output from the Sobel filter can have undesired spotty edges that are not part of the strain rate front. To remove them, morphological transformation, opening, is performed. Opening operation is the combination of two basic transformations: erosion and dilation. Morphological transformations have a similar process as convolution. A transformation is usually performed over a binary

image made of ones and zeros. The image is scanned with a structuring element analogous to the convolutional filter to produce an output image. The structuring element is a small matrix also made of ones and zeros. The transformation determines whether each pixel location is *fit* or *hit* by the structuring element based on the following rules.

- A structuring element fits the image if, for every pixel set to one in the structuring element, its corresponding position in the image is also one.
- A structuring element hits the image if, at least one pixel set to one in the structuring element, its corresponding position in the image is also one.

Figure 2.16 illustrates the concept of fit and hit. Hitting is more tolerant than fitting to produce a positive result at a pixel location. Erosion is the result of a structuring element fitting the input image. In contrast, dilation is the result of a structuring element hitting the input image. Figure 2.17 illustrates the effect of erosion, dilation, and opening.

		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Figure 2.16 Demonstration of fitting and hitting a binary image with two structuring elements Structuring element S1 fits region A only, and hits regions A, B, and C. Structuring element S2 fits regions A and B, and hits regions A and B. (reprinted from School of Computer Science at the University of Auckland, 2021).

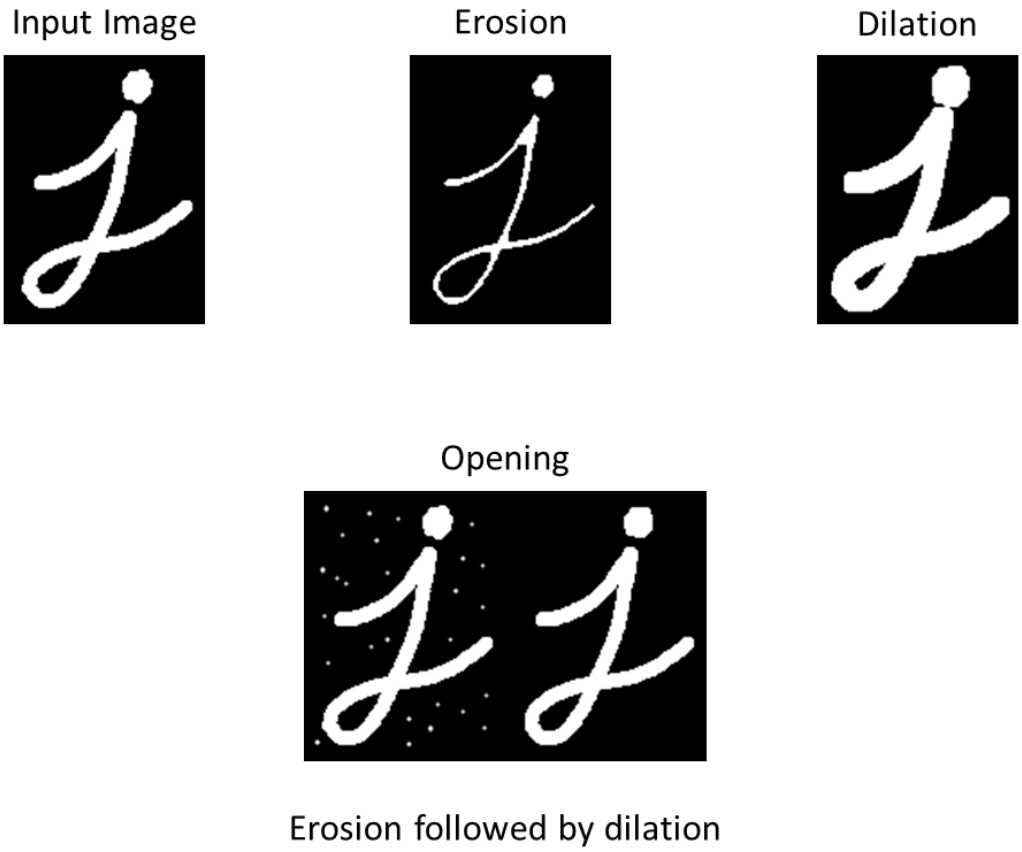

Figure 2.17 Effects of erosion, dilation, and opening operations (modified from OpenCV, 2021).

2.2.3. Connected Component Labeling

To further enhance the identification of the strain rate front, connected component labeling can be used to identify unique pixel blobs in the image. The strain rate front should be only two blobs of connected pixels separated by a small gap at the location of the fracture-hit. In the image, they should be the two largest blobs. This assumption ensures that only the strain rate front is retained in the image. The fracture-hit location can then be identified easily assuming it is the apex of this strain rate front.

Connected component labeling scans the image and finds unique groups of pixels defined by connectivity with adjacent pixels. Figure 2.18 illustrates the two types of connectivity with a center pixel: four-connectivity and eight-connectivity. Four-connectivity checks top, bottom, left, and right pixels whereas eight-connectivity checks all the eight surrounding pixels. Adjacent pixels share the same value are identified to be in the same group. For this study, eight-connectivity is used to find the pixel blobs.

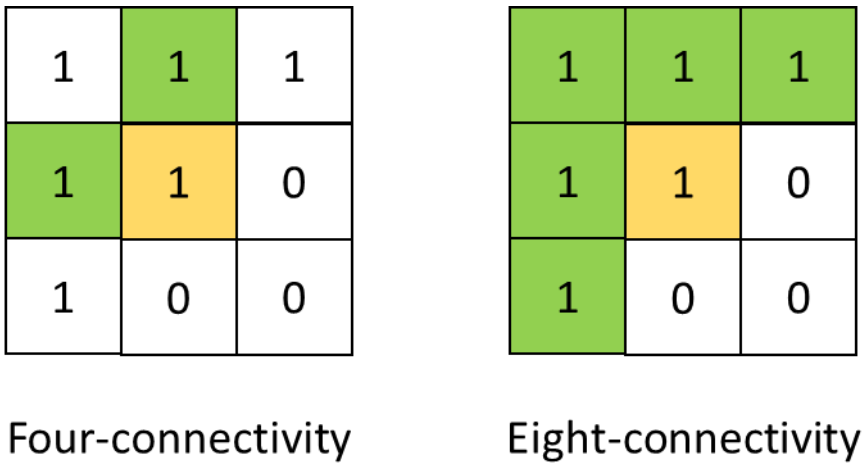

Figure 2.18 Four-connectivity (left) and eight-connectivity (right) in connected component labeling. Connected pixels that share the same value as the center pixel are grouped together.

3. MODEL DEVELOPMENT AND RESULTS

3.1. Synthetic Case

To test the feasibility of using CNN to identify and locate fracture-hit events in low-frequency DAS data, we start with using data generated by a fracture propagation model developed by Tang and Zhu (2021) that simulates the axial displacement of a fiber as a fracture approaches. Because the displacement is linearly proportional to the phase change in DAS signal associated with dynamic strain change, the displacement change can be used to approximate the strain rate inside the observation window. This simulation model provides the training data for the synthetic case.

3.1.1. Input Data

Using the strain rate simulation model to produce training data for the CNN models involves running many simulated observations at a hypothetical monitoring well over a 90-minute by 366-ft observation window. The output of the simulation is a 2D matrix representing the strain rate of the observation window. Rows of the matrix correspond to depth and columns correspond to time. Figure 3.1 shows an example of the strain rate simulation.

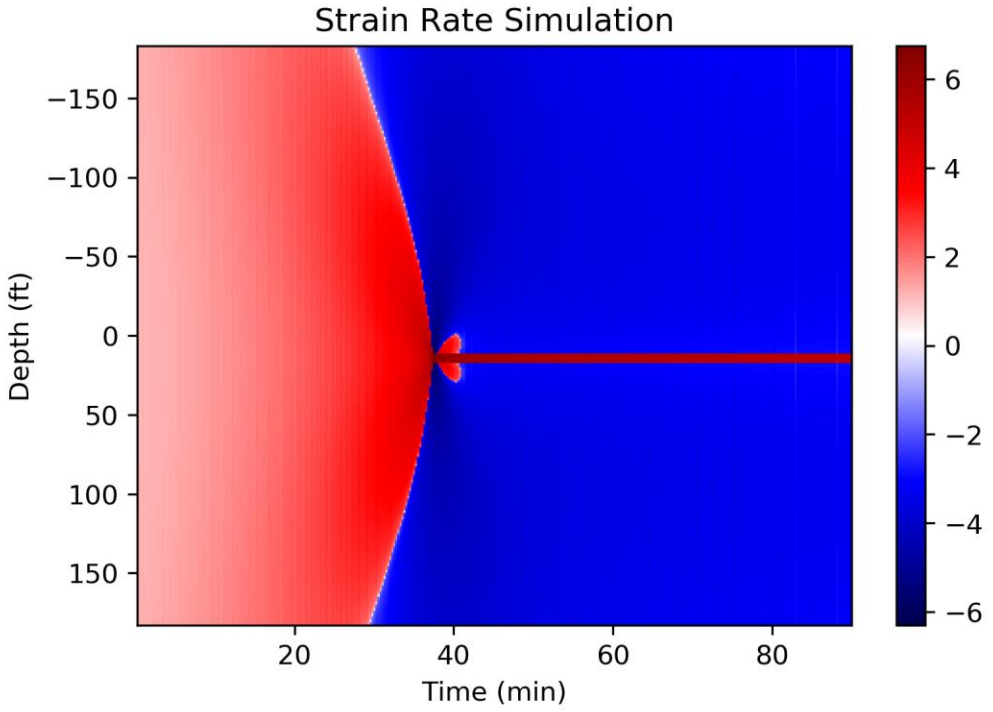

Figure 3.1 Simulation of fracture-hit strain rate at a monitoring well.

Because we intend to train a binary classifier, two sets of simulations are generated. One set contains fracture-hit patterns, labeled as positive events, and the other set does not contain fracture-hit patterns, labeled as negative events. The process of generating simulated data is described below.

- Define distributions of input parameters including injection rates, fluid viscosity, Young's modulus, Poisson's ratio, injection location, fracture height, and well spacing.
- Randomly draw sufficient samples from the input parameter distributions and run strain rate simulations.

- Save simulation results as images and ensure the validity of positive and negative events of each set.
- For the positive event set, obtain fracture-hit locations (XY-coordinates) analytically and label each sample with the location.

The input parameters for the strain rate simulations have either uniform or normal distributions. Table 3.1 and Table 3.2 list the distributions of these parameters used to generate the two sets of data. Common ranges of values are used for operational parameters. Mechanical property distributions are estimated from a study done by Kwabi (2013) for the Eagle Ford shale formation. Kwabi found the southwest and northeast regions of Eagle Ford have different distributions of Young's modulus and Poisson's ratio. In the northeastern region, Young's modulus and Poisson's ratio exhibit a negative correlation. A correlation coefficient of -0.7 is used to associate the two parameters for northeastern Eagle Ford. The correlation coefficient represents the linear correlation between two variables. It is their covariance divided by the product of their standard deviation. The fact that the two regions were used to provide input for the strain rate simulation was rather arbitrary. An additional artificial parameter, injection location, was also used to determine the hypothetical location of the fracture-hit in depth. For the positive events, injection locations are within the 366-ft observation window, whereas for the negative events, injection locations are outside the 366-ft observation window. In image representation, only positive events have the apex of the cone-shaped strain rate front. In contrast, the negative events only have a partial extension zone as if the fracture hit at a location outside the observation zone. Gaussian white noise was added to each

sample to make model training more robust. Mathematically, Gaussian white noise is generated from a normal random variable with zero mean. In this application, the noise was added to each channel (depth) and scaled to 20% of the channel's standard deviation. Figure 3.2 shows an example of a positive event (left) and a negative event (right).

Table 3.1 Strain rate simulation input parameter distributions. Mechanical properties represent SW Eagle Ford.

	Min/Mu	Max/Sigma	Distribution
Injection (bbl/min)	5	40	Uniform
Young's Modulus (GPa)	14	56	Uniform
Poisson's ratio	0.18	0.3	Uniform
Viscosity (cp)	5	100	Uniform
Fracture height(ft)	100	200	Uniform
Well Spacing (ft)	400	1000	Uniform

Table 3.2 Strain rate simulation input parameter distributions. Mechanical properties represent NE Eagle Ford. Young's modulus and Poisson's ratio for NE Eagle Ford exhibit a negative correlation as discovered by Kwabi (2013).

	Min/Mu	Max/Sigma	Distribution	Correlation Coeff
Injection (bbl/min)	5	40	Uniform	
Young's Modulus (GPa)	19	3	Normal	-0.7
Poisson's ratio	0.35	0.02	Normal	-0.7
Viscosity (cp)	5	100	Uniform	
Fracture height(ft)	100	200	Uniform	
Well Spacing (ft)	400	1000	Uniform	

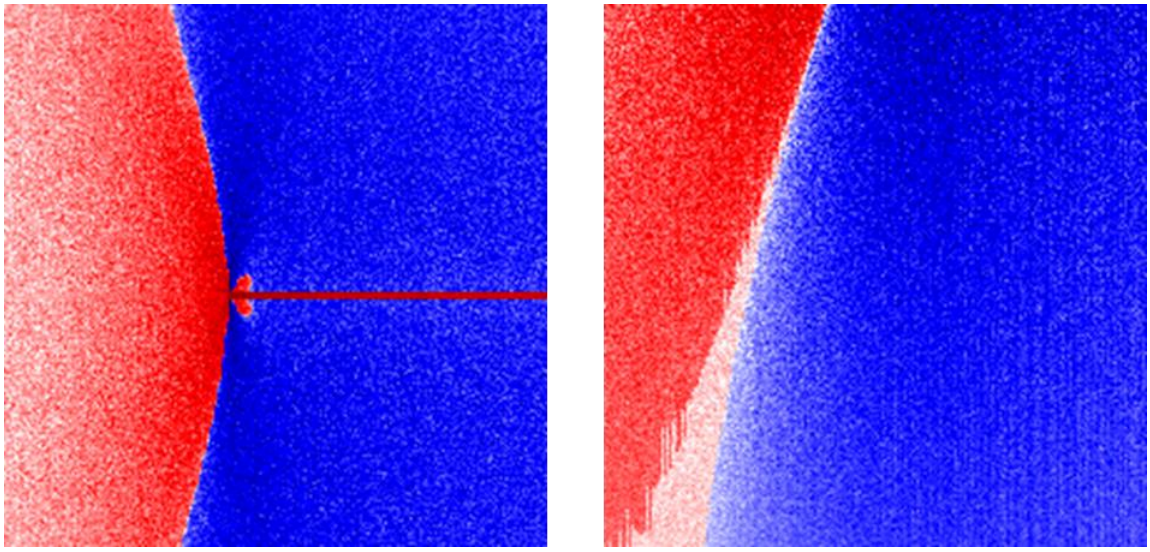

Figure 3.2 Examples of noise added strain rate patterns for synthetic fracture-hit event classification (left: positive event, right: negative event). 1000 samples of positive events and 1000 samples of negative events are generated. Red color indicates extension, and blue color indicates compression.

The goal of generating input samples for the CNN models was to obtain images with various strain rate patterns, regardless of whether the event is considered existent or not. Hence input parameters were randomly drawn from the distributions. Initially 1000 samples were generated for each set, however 60 of the positive samples did not show the apex of the strain rate front, hence they were removed from the data set. Accordingly, 60 negative samples were also removed to make the data set balanced. For both classification and localization, the images were divided into training and testing sets (80% training and 20% testing). The testing set is set apart and is only used for reporting model performance. During each training run, a random 25% of the training set is held out as the validation set. Table 3.3 lists the numbers of samples for training, validation, and testing. Only positive event samples were used for the localization task.

Table 3.3 Numbers of training, validation, and testing data sets for the synthetic case.

	Classification	Localization
Total	1880	940
Training	1128	564
Validation	376	188
Testing	376	188

3.1.2. Model Architecture and Configuration

Many image classification applications utilize well-known pre-trained models such as ResNet (He et al., 2015a) and VGGNet (Simonyan and Zisserman, 2014). These models are sophisticated and are trained to identify hundreds of types of objects. In this synthetic case, since the patterns to be identified differ drastically from objects in our daily life, using complex object detection models would require retraining the weights from the beginning, which could take much computational time. In fact, a simple CNN architecture is sufficient for the synthetic case. The model consists of two convolutional layers followed by two fully connected flat layers. Each convolutional layer includes convolution, batch normalization, ReLU activation, and max pooling. The output layer contains one unit with sigmoid activation for event classification, or two linear units for event localization. Figure 3.3 depicts the model architecture.

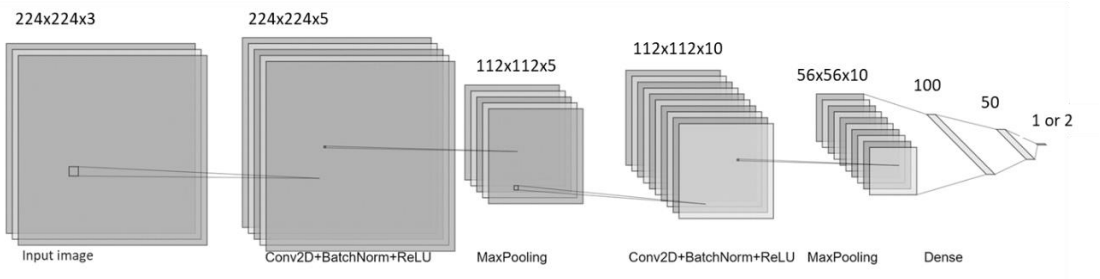

Figure 3.3 CNN model architecture for the synthetic case (model visualized by NN-SVG).

The objective of a classification problem is to maximize the probability of the target class and to minimize the probability of the non-target classes. For binary classification, the objective is to minimize the binary cross-entropy loss, J, defined as:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(p^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-p^{(i)}\right)\right] \quad 3.1$$

where θ is the model parameter vector, p is the probability of sample i belonging to the target class, obtained from the sigmoid function. m is the total number of samples, and y is the true probability (Géron, 2019).

Adam optimization is adopted here since it is a proven efficient optimization algorithm used in deep learning. As explained in section 2.1.3.4, Adam optimization allows the model to converge much faster along the right dimension with adaptive moments applied to the gradients. Additionally, tuning on hyperparameters, including the learning rate, is almost not needed. f1 score is used to evaluate model performance. The metric is defined as the harmonic mean of precision and recall. Precision is the ratio of the number of true positive events over the number of all predicted positive events.

Recall is the number of true positive events over the number of all actual positive events. Formula of f1 score is expressed in the equations below. f1 score is high when both false negative and false positive rates are low.

$$f1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{3.2}$$

where

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{3.3}$$

and

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{3.4}$$

Plug equations 3.3 and 3.4 into equation 3.2, f1 score can be expressed as:

$$f1 = \frac{true\ positive}{true\ positive + \frac{false\ negative + false\ positive}{2}} \tag{3.5}$$

In addition to identifying whether an image contains a fracture-hit event, the model is also trained to identify the location of the event. This is considered an object localization problem, practically a regression problem with the task of identifying where in the image the object of interest is. The location is typically characterized by a 2D coordinate, and relative width and height that define a bounding box around the object. In this application, we are only concerned with the 2D coordinate, hence the model has two outputs: location in X-axis (time) and in Y-axis (depth). The architecture of the localization model is the same the classification model trained to identify the presence of

the event, except that the sigmoid output is replaced with two linear activations (essentially no activations). Correspondingly, mean squared error (MSE) is used as the loss function since the training data are generated with well-defined distributions. MSE is defined as:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} - \hat{y}^{(i)}\right]^2 \tag{3.6}$$

where $y^{(i)}$ and $\hat{y}^{(i)}$ represent the true and predicted values for sample i, respectively.

3.1.3. Fracture-hit Event Classification

When selecting the filter shapes for a CNN model, the principle is to have the total number of trainable parameters small while maintaining stable predictive capability. After a few trials, the following set-up is determined. The simulation domain from the strain model for each sample is 360 time increments by 360 depth increments. Without losing much resolution, the input images for the CNN are read as 224×224 pixels in RGB (red, green, and blue) channels. The first convolution layer includes five filters of size 11×11. The second convolution layer includes ten filters of size 5×5. In general, the pooling layer filters should not be too big to avoid over destruction. Both pooling layers are of size 2×2. With the fully connected layers, the number of parameters to be trained totals 3,144,311 for the classification model, significantly smaller than that of all well-known image classification CNN models.

To ensure model robustness, the model is trained five times as a way for validation. Each time Keras's fitting function randomly selects 25% of the training data

as validation data and keeps track of the performance on both data sets. Training is quite fast with GPU enabled on Google Colaboratory. With 40 epochs and a batch size of 50, training the model five times takes less than three minutes. One *epoch* refers to that the training process having gone through all the training data once. Increasing epochs leads to bettering fitting the data but increasing it too much can result in overfitting. Figure 3.4 illustrates the history of binary cross-entropy loss and f1 score on training data and validation data for the five runs. As shown, the model stabilized on both training and validation sets for all five runs. Deployed on the testing set, the models achieved superior f1 scores (Table 3.4). Confusion matrices from the five models are shown in Table 3.5. A *confusion matrix* lists the numbers of actual and predicted classes. The rows represent the actual numbers of each class, and the columns represent the predicted numbers of each class. Table 3.5 shows that all models correctly classified positive events. Two of the models only misclassified two negative events as positive events.

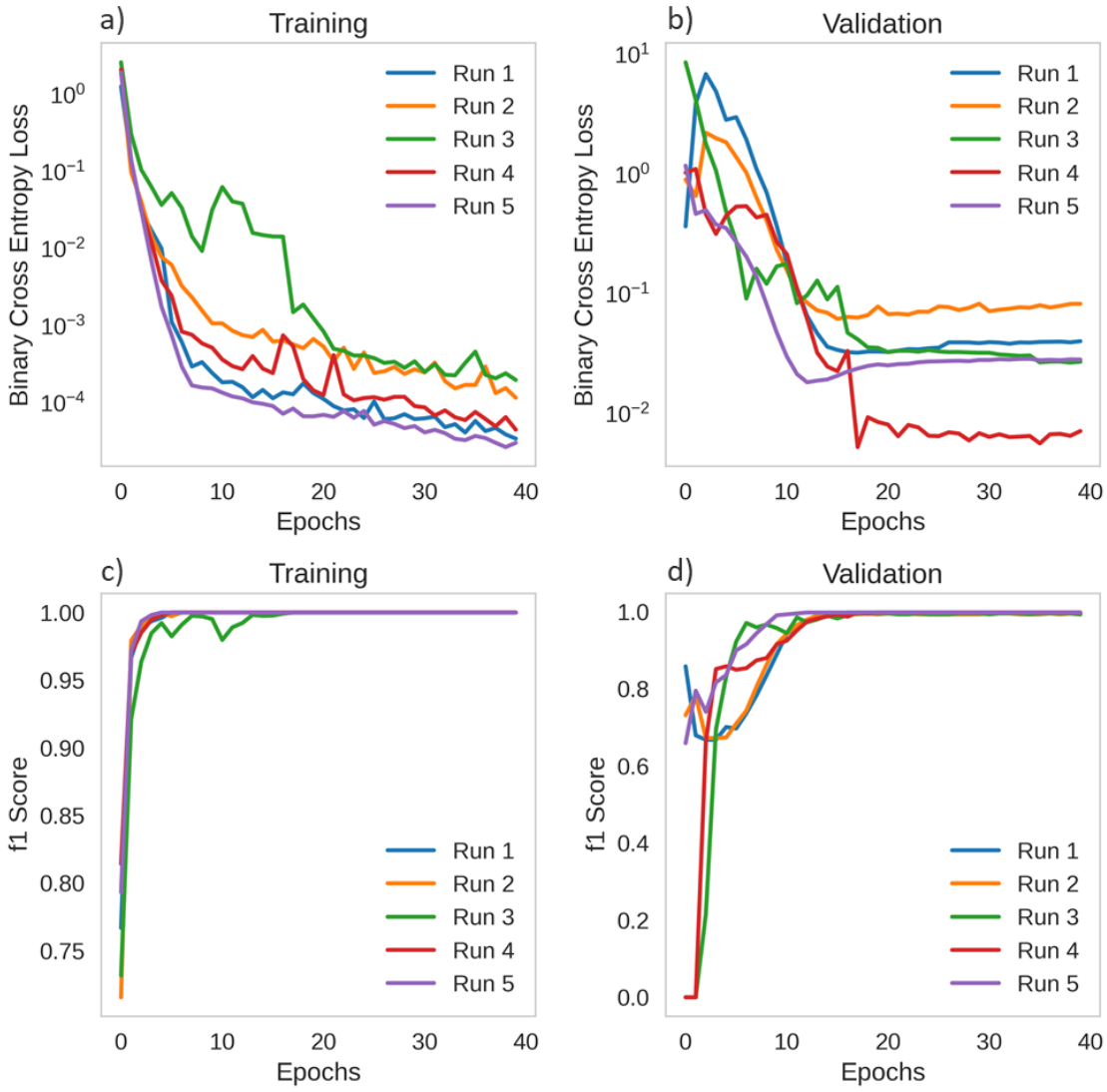

Figure 3.4 Synthetic case: history of classification model performance on training and validation sets for all five runs. a) history of binary cross-entropy loss on training data; b) history of binary cross-entropy loss on validation data; c) history of f1 score on training data; d) history of f1 score on validation data.

Table 3.4 Testing f1 scores from predictions by the five classification models (synthetic case).

	Run 1	Run 2	Run 3	Run 4	Run 5
Test F1 Score	1	0.9945	0.9944	1	1

Table 3.5 Confusion matrices from the five classification models (synthetic case).

		Run 1		Run 2		Run 3		Run 4		Run 5	
		Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive
Actual	Negative	188	0	186	2	186	2	188	0	188	0
	Positive	0	188	0	188	0	188	0	188	0	188

3.1.4. Fracture-hit Event Localization

As mentioned earlier, for locating fracture-hit events, the same model was trained with two linear outputs: one for location in time and one for location in depth. The only difference from the classification model is the number of parameters in the output layer, which is doubled. Tuning on batch size was needed to avoid overfitting. Using a batch size of 25 achieved high performance on both training and validation sets. The history of the MSE loss and coefficient of determination (R^2) are shown in Figure 3.5 for fitting the model five times. The performance is near perfection on the testing set (Table 3.6). No weights are imposed for MSE or R^2, hence the values are the average R^2 scores for the two outputs.

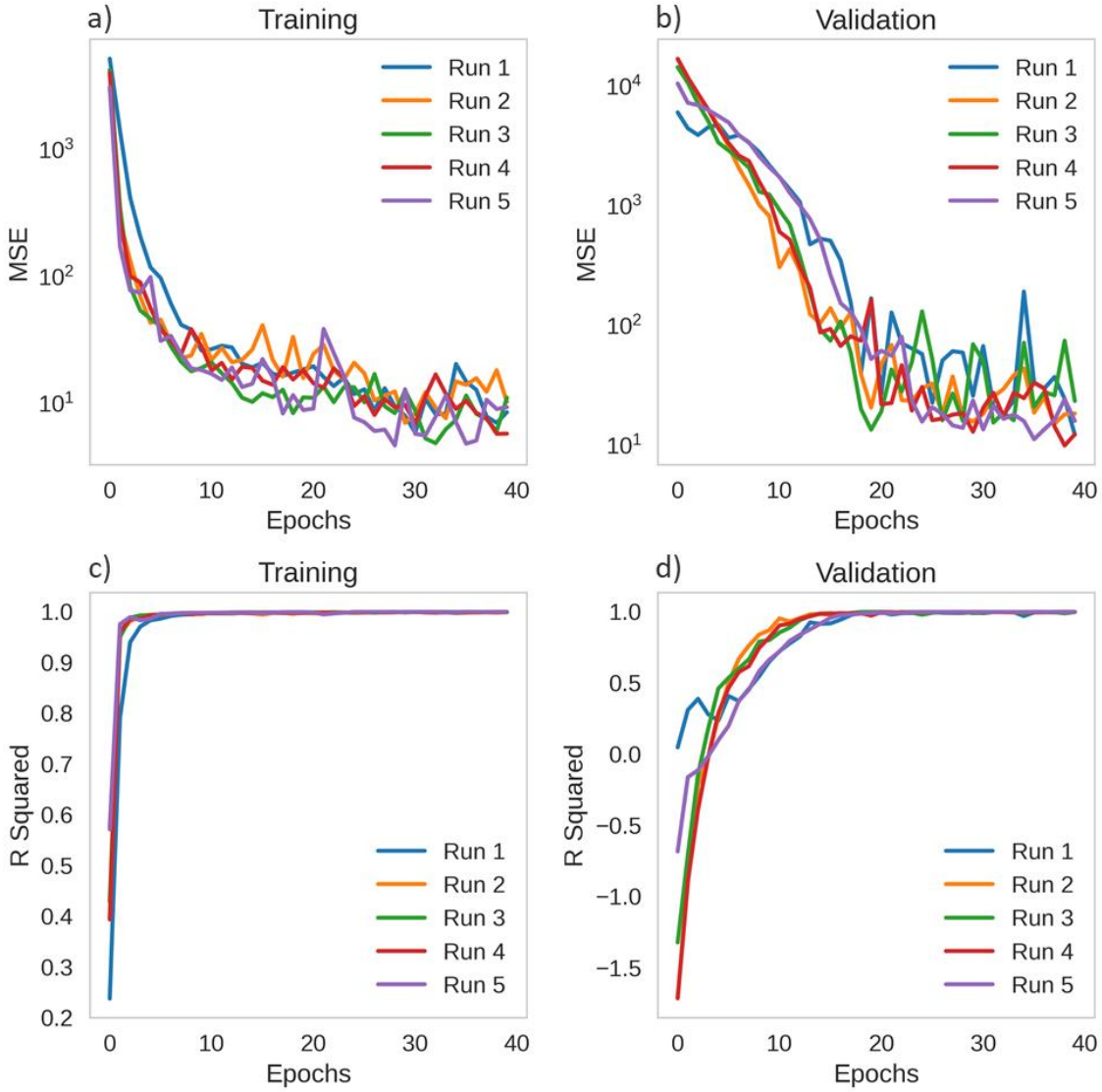

Figure 3.5 Synthetic case: history of localization model performance on training and validation sets for all five runs. a) history of mean squared error loss on training data; b) history of mean squared error loss on validation data; c) history of R^2 on training data; d) history of R^2 on validation data.

Table 3.6 R^2 of testing set predictions from the five localization models (numbers are averages of depth and time R^2).

	Run 1	Run 2	Run 3	Run 4	Run 5
Test R^2	0.9974	0.997	0.9964	0.998	0.9978

3.1.5. Event Localization with Edge Detection

The edge detection algorithm is implemented with the following Python modules:

`scipy.ndimage.sobel`

`cv2.morphologyEx`

`cv2.connectedComponents`

Each fracture-hit image goes through Sobel filtering, morphological transformation (opening), and connected component labeling. Figure 3.6 shows three examples of the edge detection process. As the examples show, the opening operation is able to retain only the strain rate front. Connected component labeling effectively does not change the result.

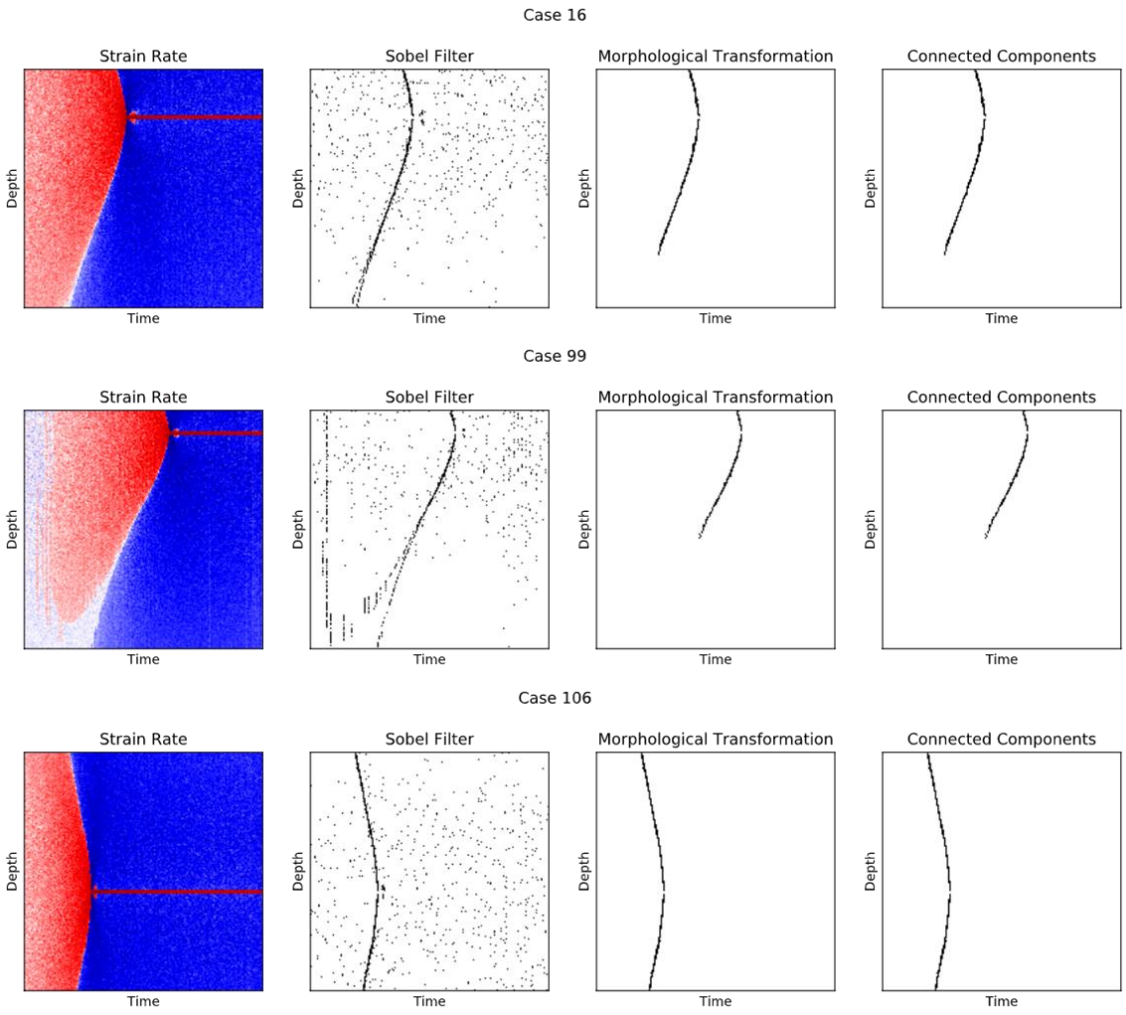

Figure 3.6 Examples results from the edge detection workflow to identify strain rate fronts.

Using the strain rate front pixels, the fracture-hit location is easily identified by selecting the rightmost pixel(s). If several rightmost pixels are identified, their average location in depth is taken. Figure 3.7 shows the cross plots of detected locations and true locations for all the 940 fracture-hit images. The process produces perfect detection in time. There are a few outliers in depth detection. This is because when several pixels sharing the same time location are identified as fracture-hit locations, the average depth

is taken, hence it can be less accurate. To compare CNN models, the trained CNN localization models are used to predict locations in depth and time on all 940 positive event samples. Figure 3.8 and Figure 3.9 show the prediction from all five models (left column: depth, right column: time). CNN models show slight variability in prediction with near-perfect correlations. Figure 3.10 shows the absolute error distributions from the five CNN localization models and edge detection in a boxplot for a more quantitative comparison. As expected, CNN models have more consistent and tighter error distributions, whereas depth location by edge detection has a wider error distribution.

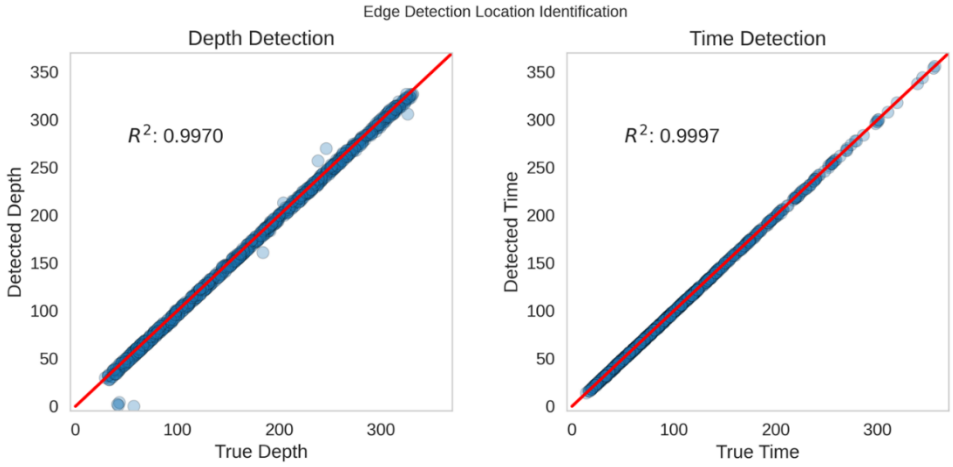

Figure 3.7 Cross plot of detected depth versus true depth (left) and cross plot of detected time versus true time (right) from edge detection workflow. Red solid line represents equality.

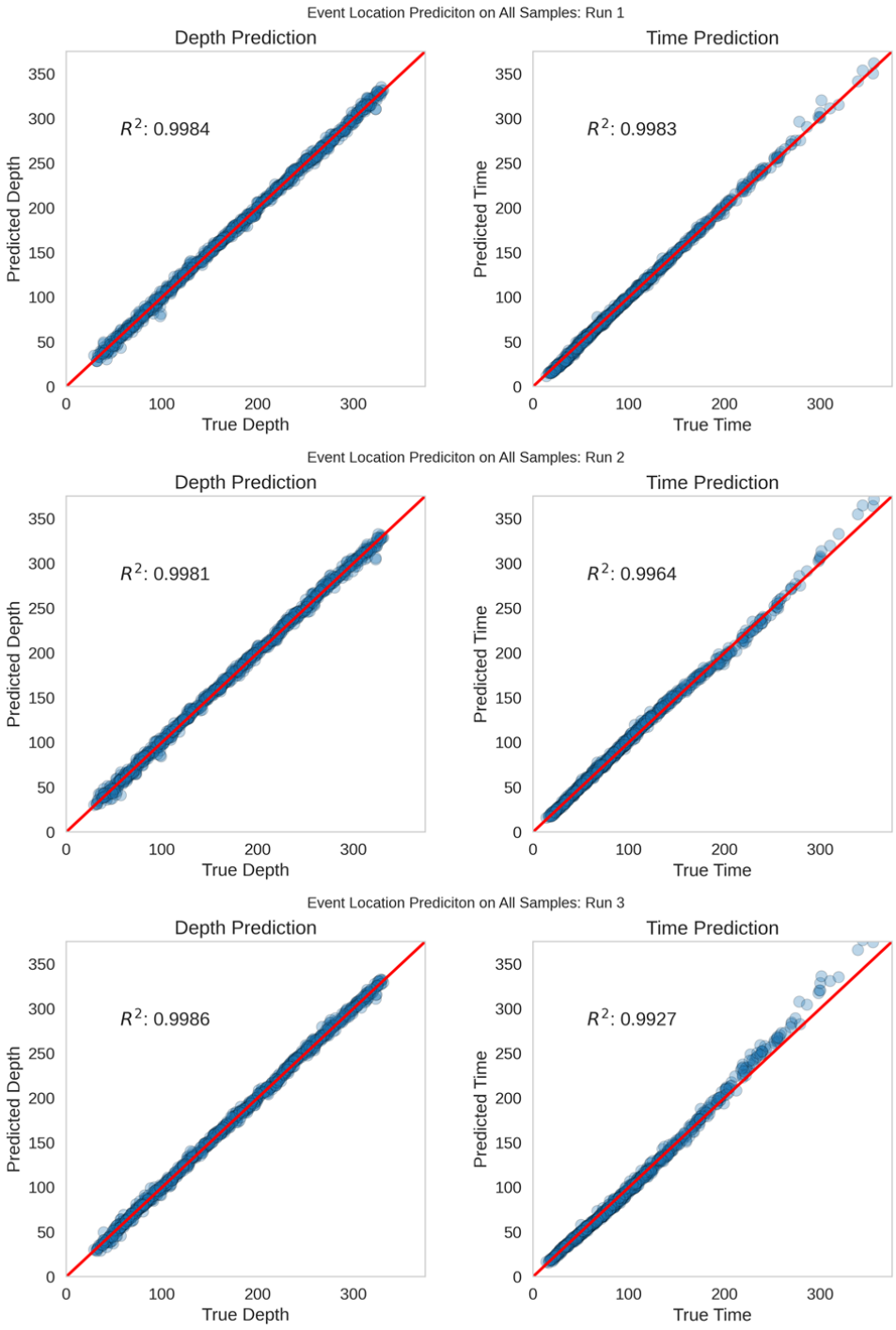

Figure 3.8 Synthetic case: cross plots of predicted depth and time of all data versus true depth and time from localization models 1-3. Red solid line represents equality (left column shows depth predictions; right column shows time predictions).

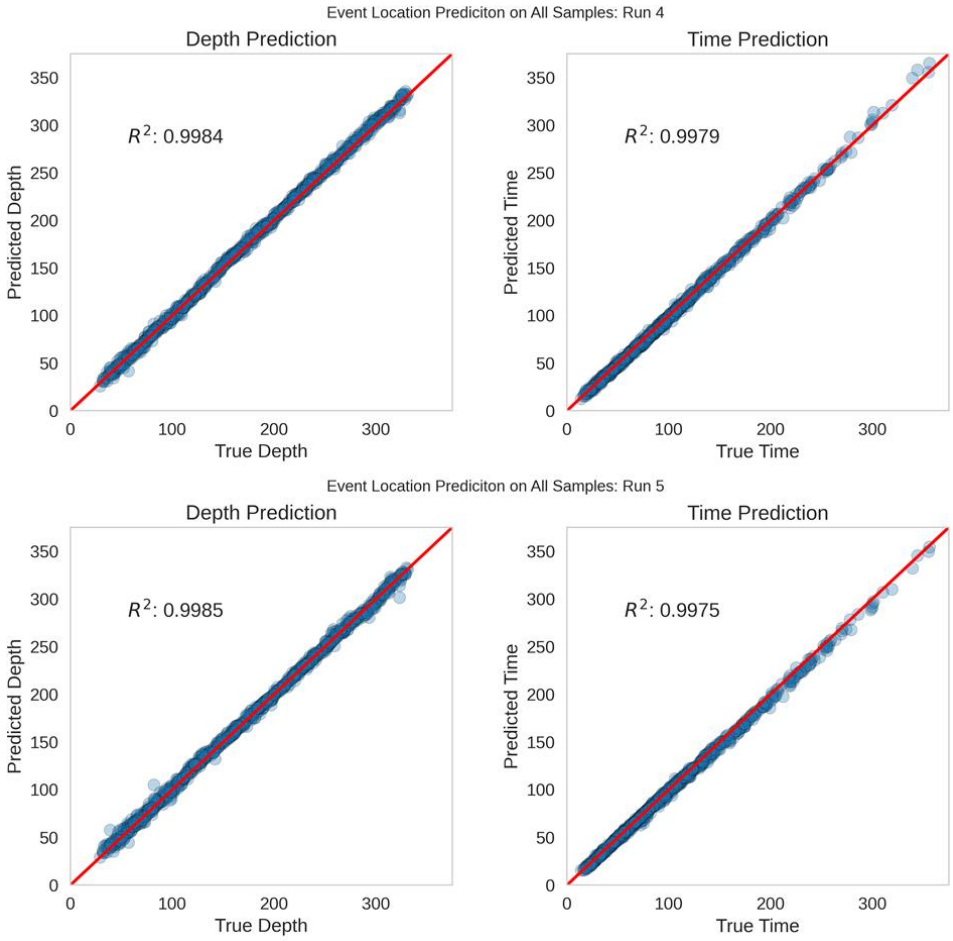

Figure 3.9 Synthetic case: cross plots of predicted depth and time of all data versus true depth and time from localization models 4 and 5. Red solid line represents equality (left column shows depth predictions; right column shows time predictions).

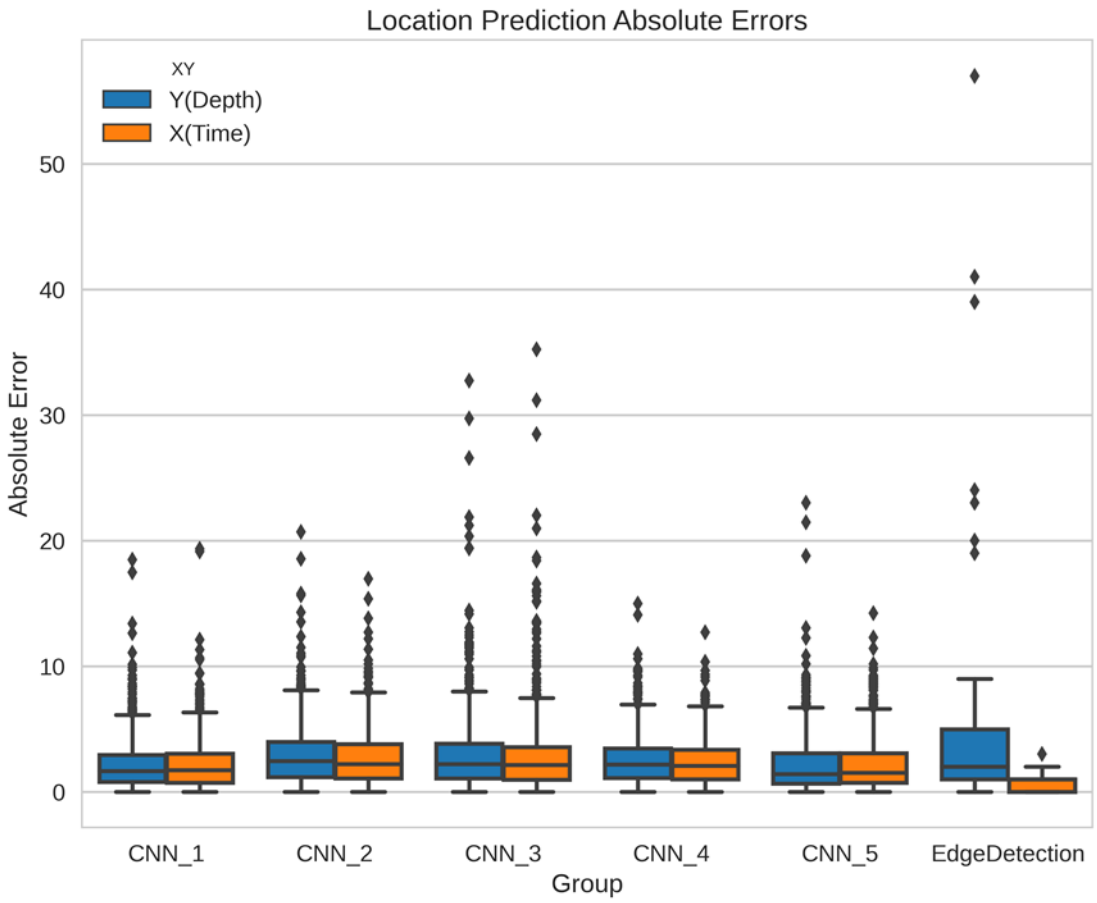

Figure 3.10 Boxplot of localization absolute error distributions for the CNN models and edge detection. The boxes represent interquartile ranges of the data, and the whisks cover 99% of the data. Diamonds represent outliers.

Although using edge detection is a simple, straightforward way to locate the apex of the strain rate front, it has some disadvantages. It assumes that the fracture-hit location is the apex of the cone-shaped strain rate front, which is valid for synthetic data, but field data may not show such regularly shaped patterns. It is also more dependent on the quality of the image, making the workflow less robust. The edge detection workflow was manually tuned to ensure edges were clearly identified. Color intensity, resolution, and level of noise in the image all affect how well defined the edges are, eventually

affecting the end result accuracy. Additionally, every image needs to go through the edge detection algorithm, making the identification slower than the CNN model prediction.

3.2. Field Case

The same training framework is applied to a field case following the successful implementation of CNN with simulated strain rate data. The field data used in this research comes from a well located in the STACK play in Oklahoma. The equipped study well (7HX) is an observation well with DAS monitoring commenced during the hydraulic fracturing treatment of two offset wells (6HX and 8HX). Figure 3.11 displays the gun barrel view of the layout of the three wells.

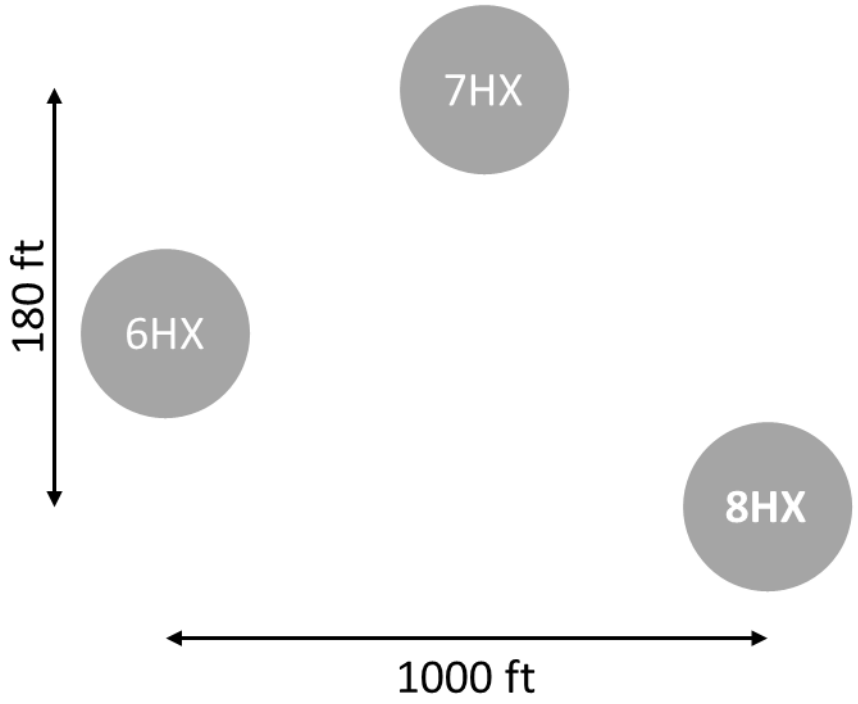

Figure 3.11 Gun barrel view of the instrumented monitoring well (7HX) and two treatment wells (6HX and 8HX).

Distributed fiber optic sensing was operating for 202 hours (8.4 days) and low-frequency (1Hz) DAS phase change was recorded every 10 seconds at a one-meter depth resolution. The raw data needs to go through log transformation for the extension/compression polarity to be visible. Firstly, the data is differentiated in time to represent strain rate. Then all zeros are changed to a tiny number (0.0001). Negative values are converted to positive then back to negative after the logarithm is taken. Figure 3.12 shows a portion of the data before and after the transformation. The processed data clearly shows several fracture-hit patterns as desired. Figure 3.13 shows the entire processed DAS data from the study well.

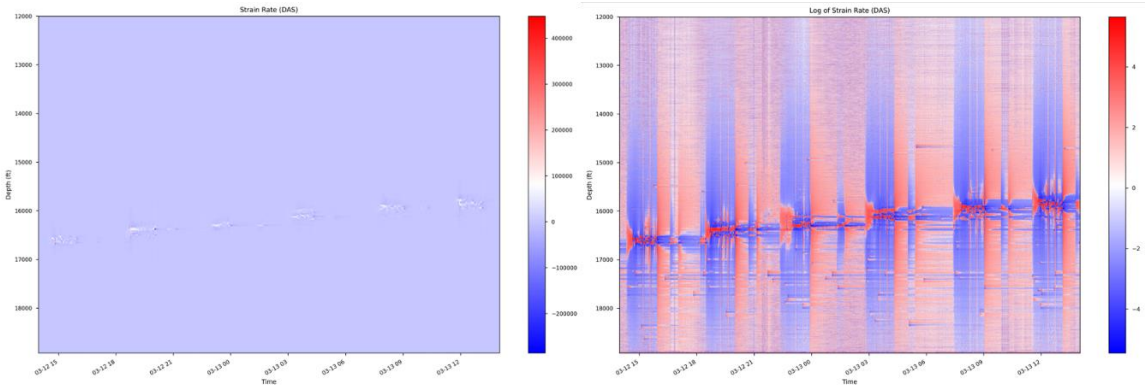

Figure 3.12 A portion of unprocessed (left) and processed (right) low-frequency DAS data from the study well.

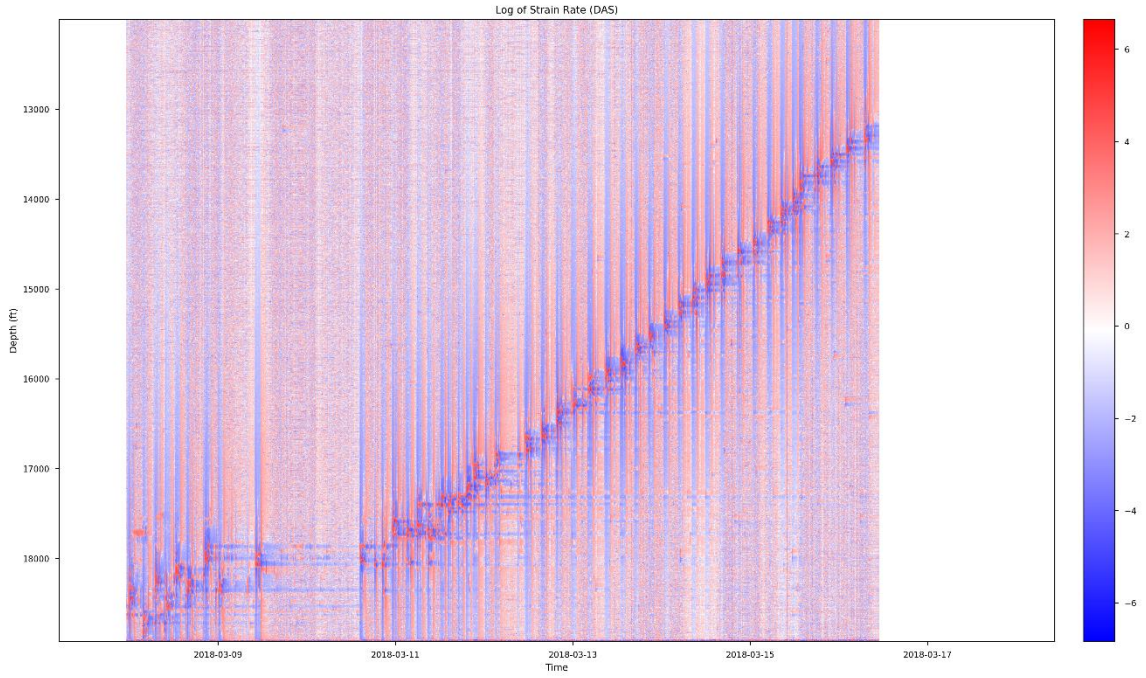

Figure 3.13 Entire processed low-frequency DAS data from the study well.

3.2.1. Input Data

For the data to be usable by the CNN models, image samples must be produced in the size that can contain one fracture-hit event. This is done by sliding a rectangular

window over the data set horizontally and vertically. The sliding window covers three hours and about 300 meters. It moves by one fourth of its size both horizontally and vertically. Figure 3.14 demonstrates the sliding window process. As a result, 5754 samples are produced from the study well. Because of how the sliding window moves, the same fracture-hit event appears in several image samples at various relative locations. This process has the effect of *Data Augmentation*, producing more available training data. Data augmentation refers to slightly modifying input images to increase the number of training data. Some data augmentation techniques include changing the color of the image, rotating or flipping the image, or slightly twisting the image. In this case, the input data is augmented in the sense that the same event appears in different locations of several input images.

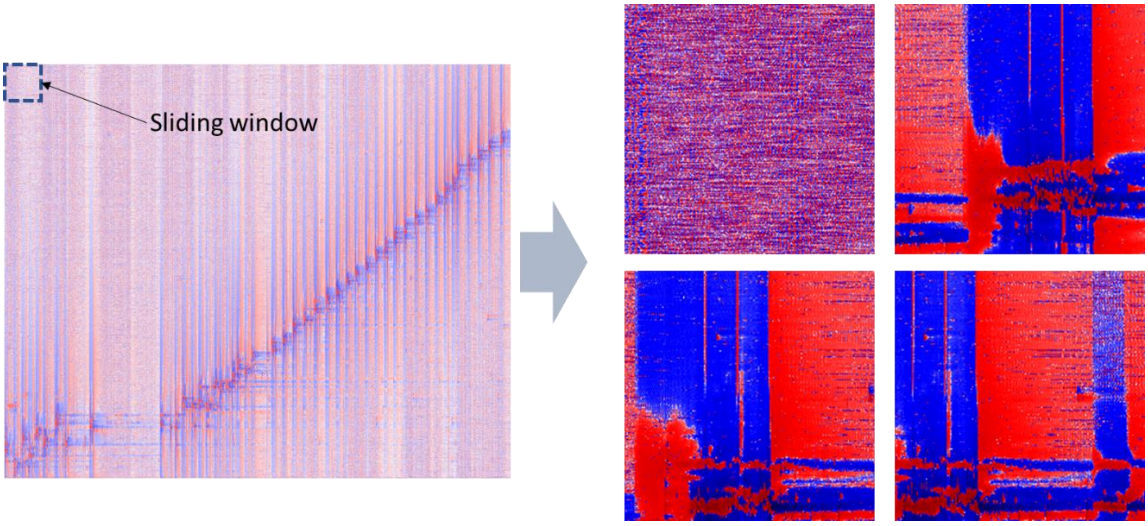

Figure 3.14 Sliding a rectangular window to produce field DAS image samples. Left: the entire DAS data. Right: four examples of subsets of the data generated by the sliding window.

Although the interest of this study is to recognize fracture-hit events, it is reasonable to create three classes rather than a binary class. The samples are manually labeled with “noise”, “shadow”, or “fracture-hit” event classes. The fracture-hit samples are further annotated with hit locations using an online open-source image labeling and annotation tool, MakeSense.AI. Examples of each class are shown in Figure 3.15. Table 3.7 lists the numbers of samples of each class. This data set is imbalanced. Noise and shadow samples significantly outweigh fracture-hit samples. To handle imbalanced data, random down sampling is performed during the classification model training. This means that noise and shadow samples are randomly removed so that their quantities equal to the number of fracture-hit events. Although the number of samples are significantly reduced, it is still sufficient to train models with reasonably high performance as demonstrated later. The fracture-hit event samples are further annotated with XY coordinates to indicate the hit locations in time and depth. Now the data set is prepared for training both classification and localization models. The same train-validation-test split proportion is used, and the sample numbers are listed in Table 3.8.

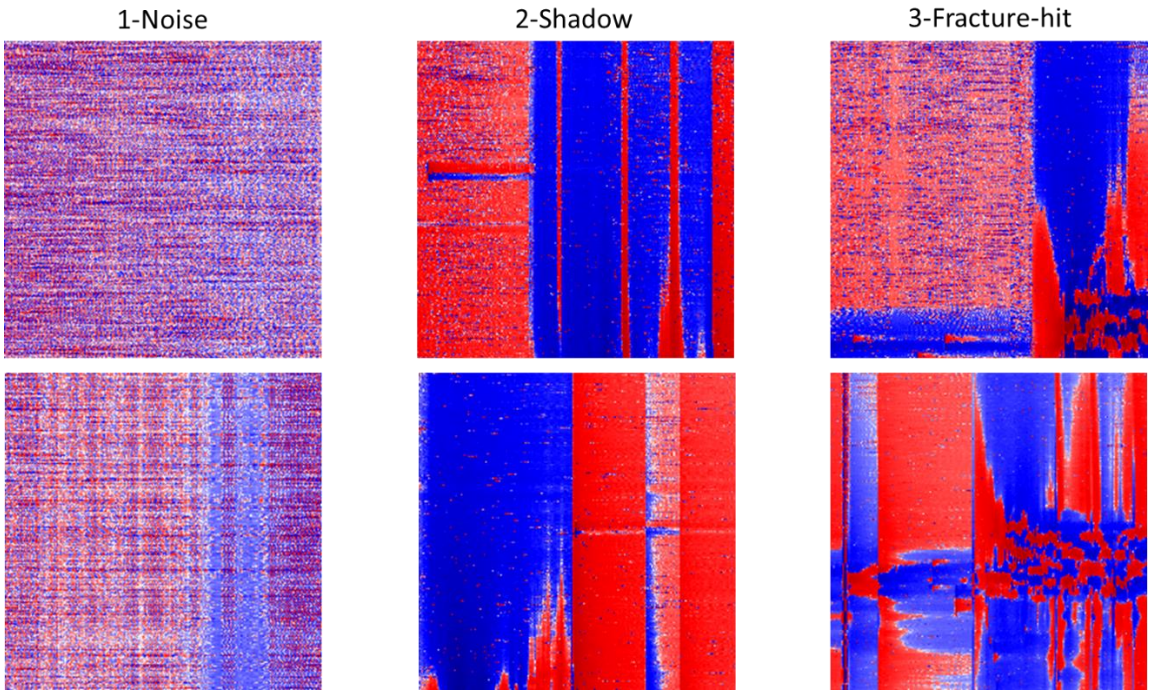

Figure 3.15 Examples of noise, shadow and fracture-hit samples generated from field DAS data.

Table 3.7 Field case sample counts.

	Noise	Shadow	Fracture-hit
Number of samples	2845	2485	424

Table 3.8 Numbers of samples for training, validation, and testing sets for the field case.

	Classification	Localization
Total	1272	424
Training	762	254
Validation	255	85
Testing	255	85

3.2.2. Model Architecture and Configuration

Since the field case images are much more complex than the simulated data images, the simple architecture used for the simulated data is no longer adequate. Generally, more complex visual tasks require deeper neural networks. The common path is to utilize well known convolutional neural networks architecture that is proven to work. One of the relatively simple architectures is the AlexNet, which won the 2012 ImageNet ILSVRC challenge (ImageNet Large Scale Visual Recognition Challenge). The AlexNet resembles the architecture used for the synthetic case. The major difference is that it has three more convolutional layers which contain many small 3×3 filters. The architecture is illustrated in Figure 3.16. Using AlexNet, we train the classification and localization models with field DAS data samples. The input and output sizes of AlexNet are modified to fit this case. The input size is changed to DAS sample image size (233×233) and the output layer is changed to three units for classification and two units for localization. For classification, the output layer is activated by the softmax function since there are three classes.

In the two flat layers of AlexNet, there is a 50% drop out rate. *Drop out* is randomly removing the connections between neurons while training the model. This means the model must not rely on every single neuron in the corresponding layer, hence the learning process makes the model more robust and avoids overfitting.

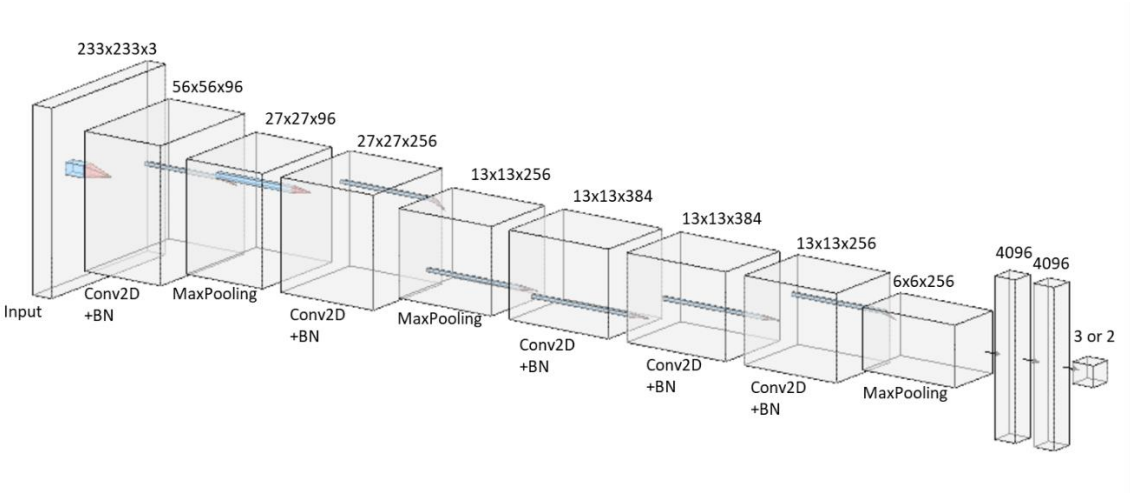

Figure 3.16 Architecture of the modified AlexNet used for field case (model visualized by NN-SVG).

For multi-class classification with softmax activation, the objective is to minimize the cross-entropy cost function defined as:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right) \tag{3.7}$$

where K is the total number of class, and $\hat{p}_k^{(i)}$ is the predicted probability of sample i belonging to class k, and $y_k^{(i)}$ is the true probability of sample i belonging to class k. When K is two, this equation becomes the binary cross-entropy cost function. f1 score is still used as the evaluation metric. Since now there are three classes, global f1 is calculated during evaluation. However, the metric can be calculated for each class.

For fracture-hit event localization with CNN, the configuration is the same as the synthetic case. MSE is the cost function, and R^2 is the evaluation metric.

3.2.3. Fracture-hit Event Classification

There are numerous factors that affect how a model performs. Here we focus on the activation function, the initializer, epochs, and batch size as they are more impactful factors. The training process involves using different combinations of these parameters. Table 3.9 lists the mean f1 scores for training and validation using various combinations of the four parameters. f1 score on validation data is color coded with the darkest green indicating the best model. Based on validation performance, the best model is given by ELU activation, He initialization with normal distribution, 20 epochs and a batch size of 25. Table 3.10 shows the f1 scores of the best model for the testing data set for all three classes. For noise and events, the model achieved 0.9075 and 0.9021 f1 scores. f1 score for shadow is much lower (0.7705). This is expected because among the three classes, shadow is the least well defined. It is essentially anything that is neither noise nor fracture-hit.

Table 3.9 Classification model training configuration and model performance (field case).

No.	Activation	Initializer	Epoch	Batch Size	Training Mean f1	Validation Mean f1	Testing Mean f1
1	relu	glorot_uniform	30	50	0.9177	0.7310	0.7414
2	relu	glorot_uniform	30	40	0.8697	0.7880	0.8128
3	relu	glorot_uniform	30	20	0.8836	0.7588	0.7947
4	relu	glorot_uniform	30	30	0.8872	0.7363	0.7922
5	relu	glorot_uniform	20	20	0.8498	0.7483	0.7776
6	relu	glorot_uniform	20	30	0.9152	0.8359	0.8630
7	elu	he_normal	30	50	0.9248	0.8393	0.8919
8	elu	he_normal	30	40	0.8773	0.7810	0.8169
9	elu	he_normal	30	20	0.8457	0.7523	0.7678
10	elu	he_normal	30	30	0.9108	0.7570	0.8025
11	elu	he_normal	20	20	0.8779	0.8510	0.8728
12	elu	he_normal	20	30	0.8934	0.8500	0.8949
13	elu	he_normal	20	25	0.9065	0.8655	0.8661

Table 3.10 Testing f1 scores for each event type (field case).

Testing Mean f1 (Noise)	Testing mean f1 (Shadow)	Testing mean f1 (Event)
0.9075	0.7705	0.9021

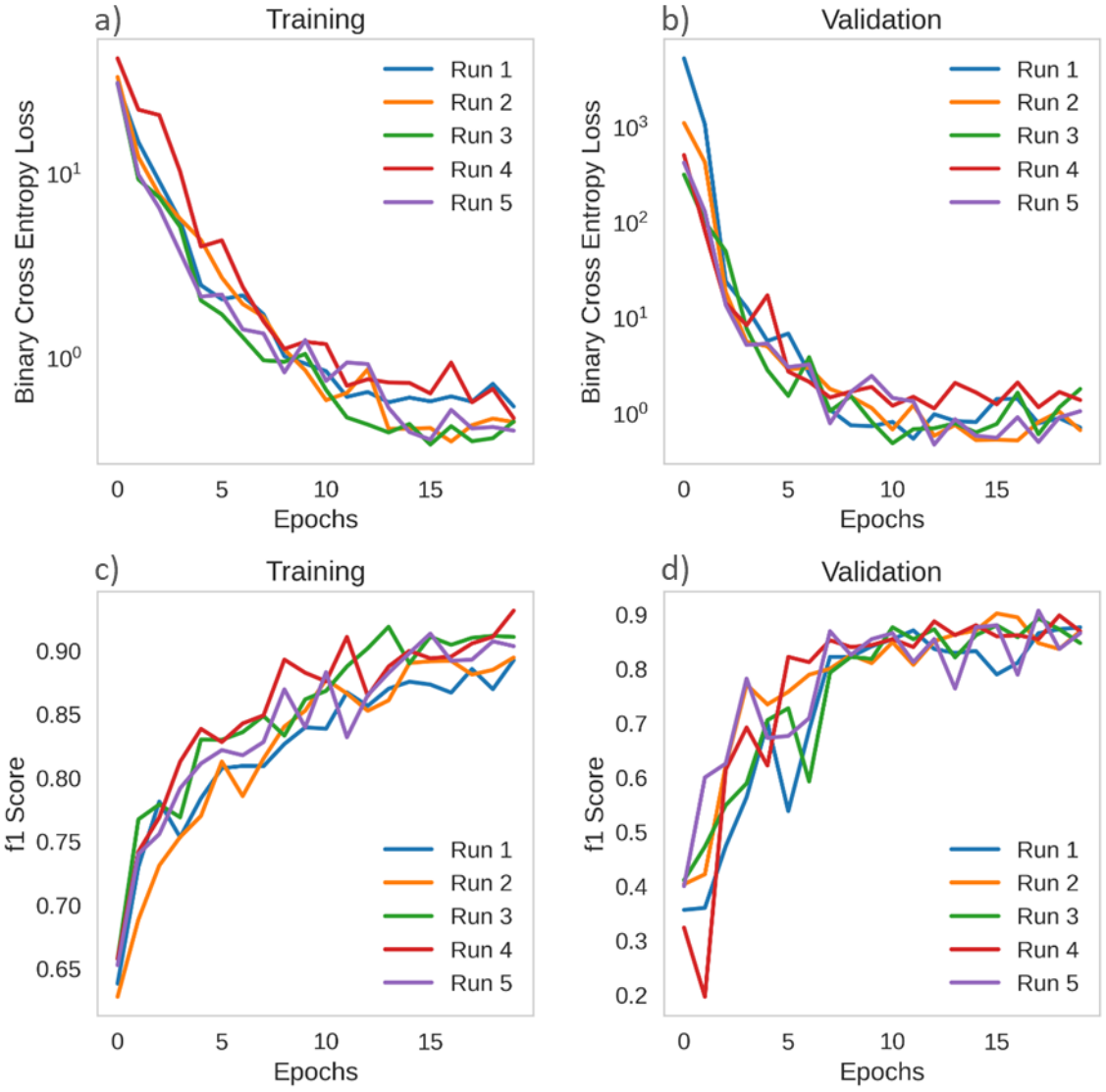

Figure 3.17 Field case: history of classification model performance on training and validation sets for all five runs. a) history of binary cross-entropy loss on training data; b) history of binary cross-entropy loss on validation data; c) history of f1 score on training data; d) history of f1 score on validation data.

3.2.4. Fracture-hit Event Localization

Similar to the synthetic case, the same model architecture (AlexNet) is trained again for the localization task, with the output layer replaced with two linear units. Again, various combinations of the activation function, initializer, epochs, and batch size are used (Table 3.11). Based on validation R^2, the best model is given by ELU activation, uniform Glorot initialization, 50 epochs and a batch size of 15. This model achieved an R^2 of 0.9349 on the validation data, and 0.9565 on the testing data. He initialization is designed for ELU activation. However, it appears that for the same epochs and batch size, the combination of ELU activation and uniform Glorot initialization performs slightly better on the validation data set. It is unclear to the author why that is the case. Figure 3.18 shows the comparison of R^2. For the training data (light blue), uniform Glorot initialization and normal He initialization do not show much difference in R^2, but the difference is obvious on the validation data (dark blue). The training history of the best model configuration is shown in Figure 3.19. Figure 3.20 and Figure 3.21 show the cross plots of the predicted fracture-hit locations from the five localization models versus true locations for the testing data. The accuracy is less perfect compared to the synthetic case because the manually labeled "true" locations bears some level of subjectivity and uncertainty. Nonetheless, an average R^2 of 0.9565 is considered effective validation.

Table 3.11 Localization model training configuration and model performance (field case).

No.	Activation	Initializer	Epoch	BatchSize	TrainingMeanR^2	ValidationMeanR^2	TestingMeanR^2
1	relu	glorot_uniform	40	20	0.8769	0.8835	0.8775
2	relu	glorot_uniform	40	30	0.8935	0.7832	0.7691
3	relu	glorot_uniform	50	20	0.8787	0.8798	0.8670
4	relu	glorot_uniform	30	20	0.8793	0.8087	0.7833
5	elu	he_normal	40	20	0.9549	0.9145	0.9280
6	elu	he_normal	40	30	0.9502	0.8825	0.8928
7	elu	he_normal	50	20	0.9573	0.9184	0.9392
8	elu	he_normal	30	20	0.9417	0.9037	0.9171
9	elu	glorot_uniform	40	20	0.9539	0.9184	0.9392
10	elu	glorot_uniform	50	20	0.9597	0.9293	0.9426
11	elu	glorot_uniform	50	30	0.9568	0.8956	0.9175
12	elu	glorot_uniform	50	10	0.9497	0.8966	0.9388
13	elu	glorot_uniform	50	15	0.9592	0.9349	0.9565
14	elu	he_normal	50	15	0.9590	0.9033	0.9351

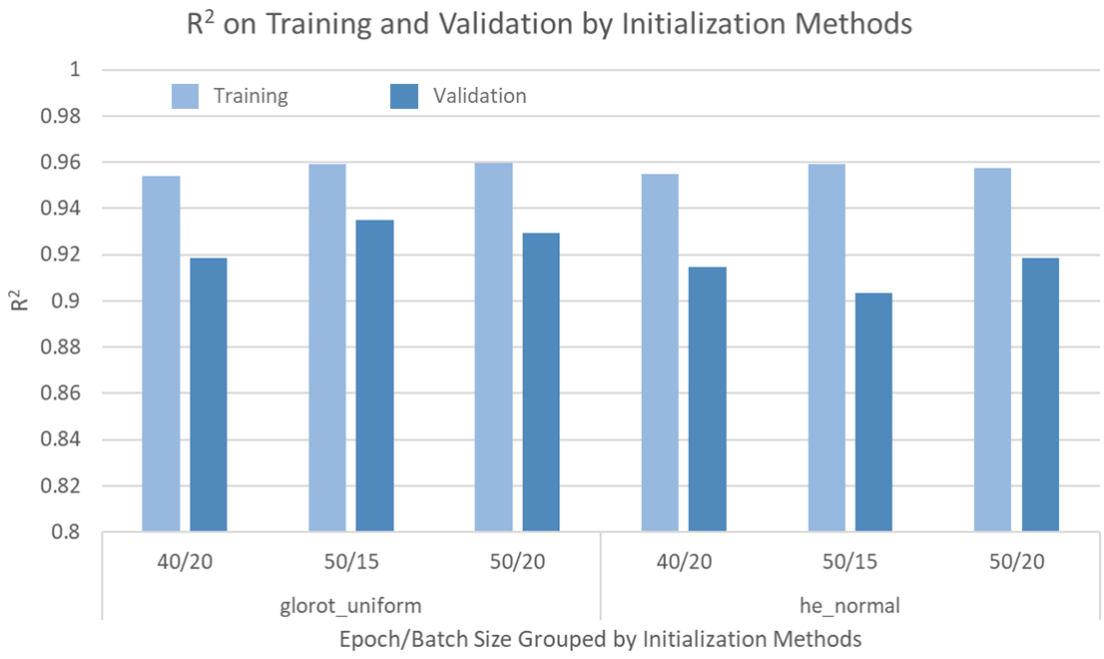

Figure 3.18 Comparison of R^2 on training and validation data with the same activation function but different initialization methods.

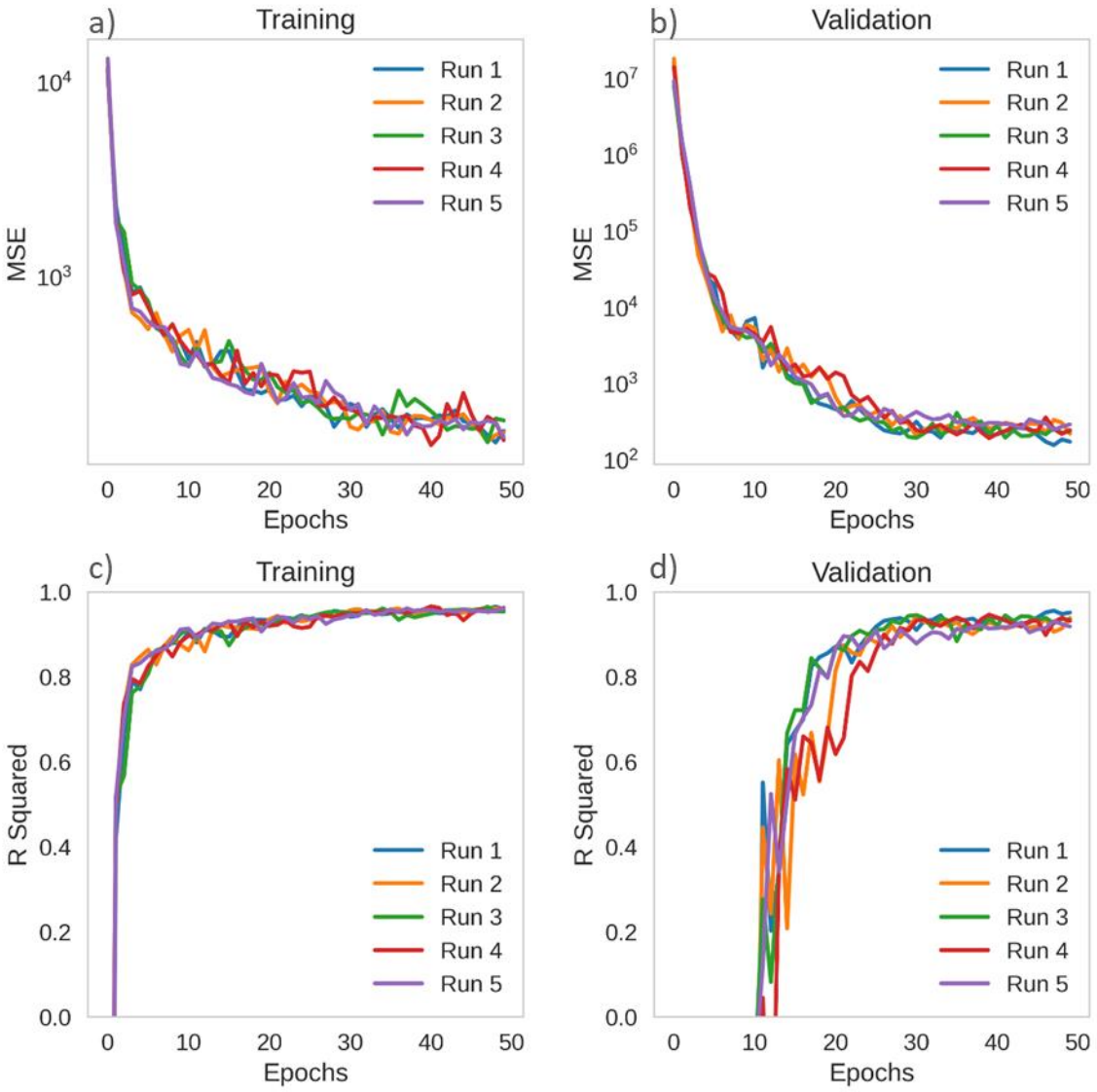

Figure 3.19 Field case: history of localization model performance on training and validation sets for all five runs. a) history of mean squared error loss on training data; b) history of mean squared error loss on validation data; c) history of R^2 on training data; d) history of R^2 on validation data, early epochs not shown due to large negative values.

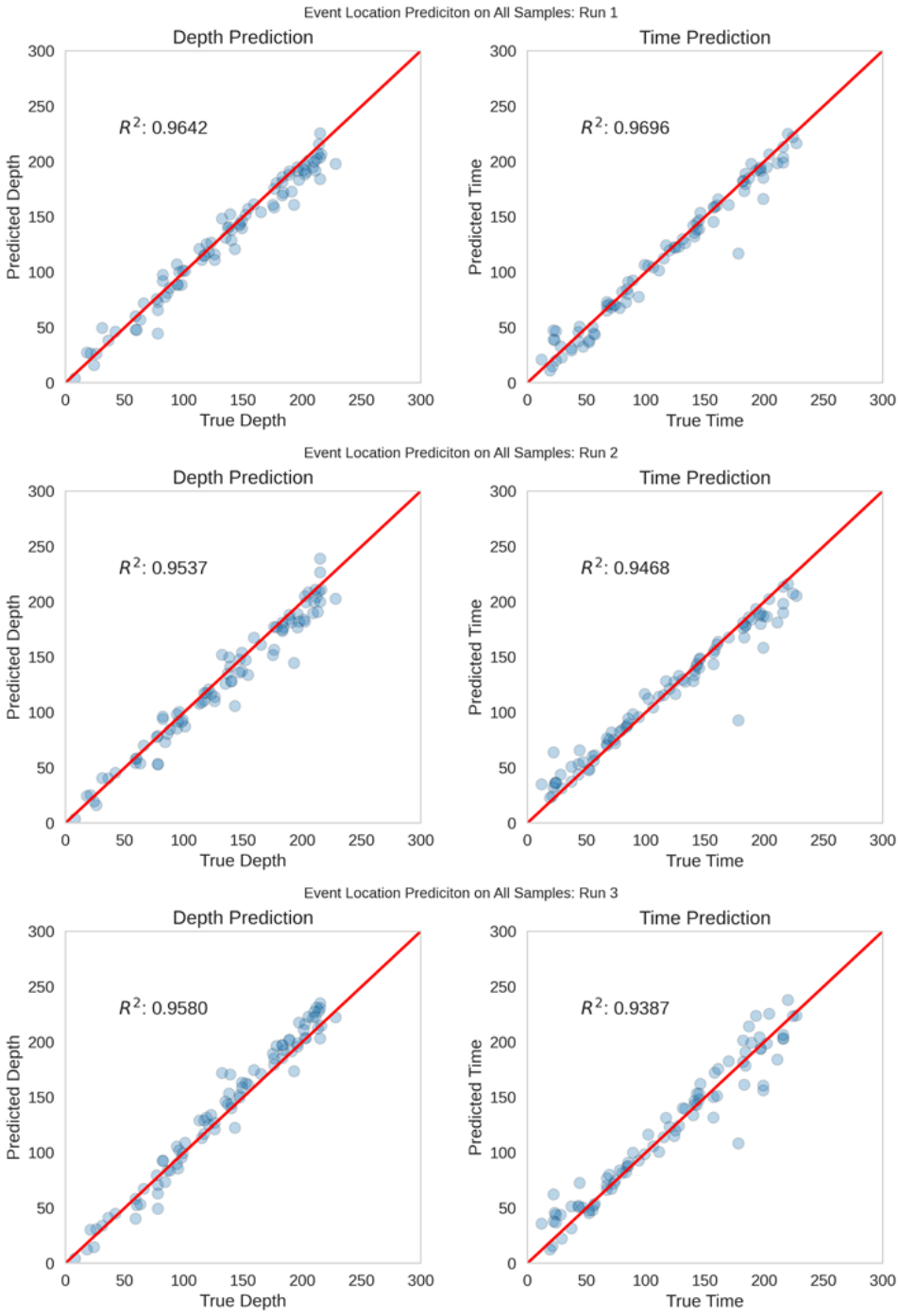

Figure 3.20 Field case: cross plots of predicted depth and time of the testing data versus true depth and time from localization models 1-3. Red solid line represents equality (left column shows depth predictions; right column shows time predictions).

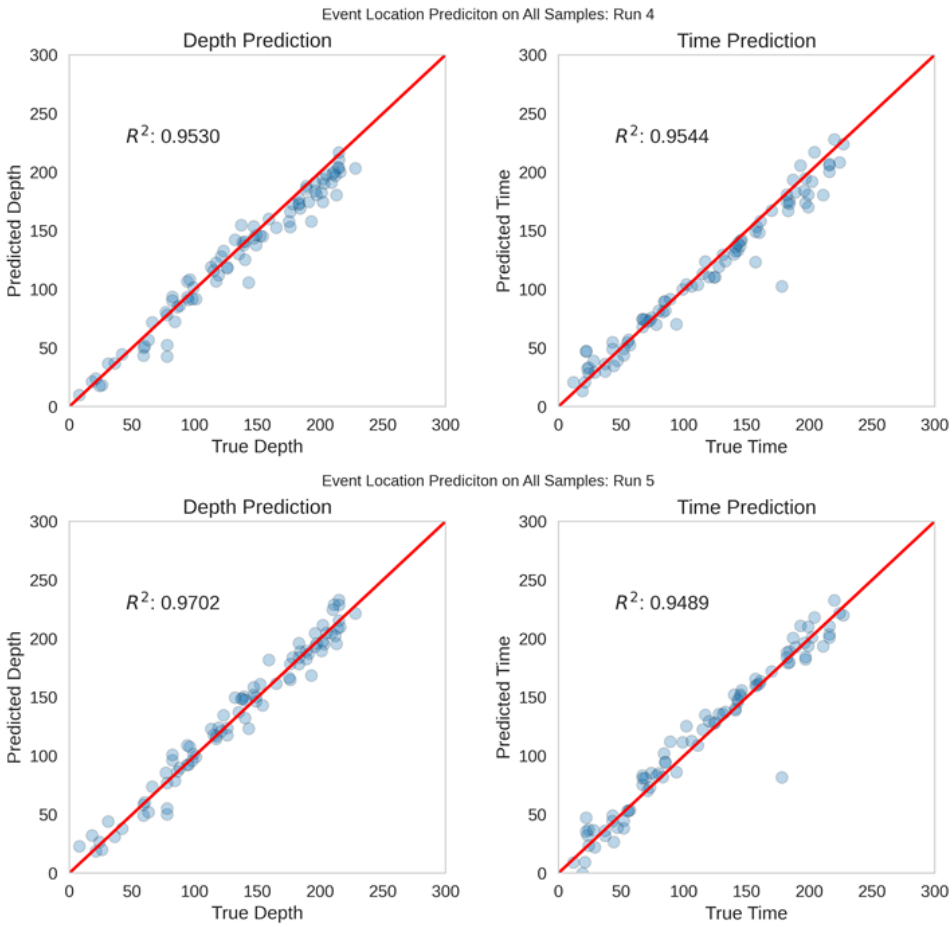

Figure 3.21 Field case: cross plots of predicted depth and time of the testing data versus true depth and time from localization models 4 and 5. Red solid line represents equality (left column shows depth predictions; right column shows time predictions).

3.2.5. Field Case Summary

It is exciting to see how CNN can predict classes of events and fracture-hit locations with field data. In the field case application, the entire low-frequency DAS data from a monitoring well is divided into small segments to serve as training data samples. The labeled and annotated samples are used to train the modified AlexNet for recognizing fracture-hits and identifying the location of fracture-hits. A trained model

can be deployed in real-time to identify fracture-hits events. In practice, the model would need to be able to recognize events from the entire live stream of DAS signals, rather than small rectangular segments. However, since the model is trained with fracture-hit events located anywhere inside the rectangle, it should be able to recognize the pattern as soon as it appears on a 2D screen. With the model developed into a software package, operators can be notified of fracture-hits instantly rather than having to wait for the operation to be completed.

4. CONCLUSIONS

To the author's best knowledge, this study is the first to demonstrate the efficiency and accuracy of using convolutional neural networks models to classify and localize fracture-hit events in low-frequency DAS strain rate, for both simulated and field data. For the simulated data, both classification and localization models achieved near-perfect predictions. The results provide a proof of concept of using CNN for real-time event detection from low-frequency DAS data. The success was replicated with field data. For the field case, the model architecture needed to be more complex to accommodate the nature of field data, which was expected. AlexNet was chosen because it is a proven architecture similar to the one used for the synthetic case but deeper. We successfully trained AlexNet to recognize field DAS images grouped into three classes, and to identify locations of fracture-hit events.

Training the models for the synthetic case was relatively easy compared to training for the field case. Because simulated strain rate images have regularly shaped strain rate patterns, simple model architecture and configuration were sufficient. Regarding training the models for the field case, using ELU activation with normal He initialization generally yielded better performance than using ReLU activation and uniform Glorot initialization.

The functionality of using CNN models to identify and locate fracture-hit events can be further developed as software packages for fiber optic sensing deployment, enabling this technology with automated real-time event detection capability. With real-

time fracture-hit event identification, operators can obtain instant accurate response of fracture propagation in terms of time and space. With prompt understanding of fracture propagation geometry, operators can have the capability of optimizing stimulation designs with quicker responses, potentially saving lots of time and capital on fracking operation.

Furthermore, because of CNN's flexibility and trainable nature, its applications extend beyond identifying strain rate patterns in low-frequency DAS data. For example, if events associated with treatment well activities are of interest, CNN models can be trained to recognize various in-well activities.

Edge detection is another plausible technique for locating fracture-hits, but it relies on assumptions about the image. The cone-shaped pattern must be smoothly and sharply defined. These assumptions often do not hold for field DAS data. The comparison of edge detection and CNN further supports the need for using CNN for image-based real-time event detection applications for fiber optic sensing data.

REFERENCES

Fukushima, K. 1980. Neocognitron: A Self-organizing Neural Network Model for A Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biol. Cybernetics* 36, 193–202 https://doi.org/10.1007/BF00344251.

Géron, A, 2019. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, second edition. Sebastopol, California: O'Reilly.

Glorot, X. & Bengio, Y. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. *Machine Learning Research* 9:249-256. http://proceedings.mlr.press/v9/glorot10a.html.

Google Colaboratory. Web-based Python Development Environment. https://colab.research.google.com/.

Haustveit, K., Elliiot, B., Haffener, J. et al. 2020. Monitoring the Pulse of a Well Through Sealed Wellbore Pressure Monitoring, a Breakthrough Diagnostic with a Multi-Basin Case Study. Paper presented at the SPE Hydraulic Fracturing Technology Conference and Exhibition, The Woodlands, Texas, USA, February. SPE-199731-MS. https://doi.org/10.2118/199731-MS.

He, K., Zhang, X., Ren, S., Sun, J., 2015a. Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385.

He, K., Zhang, X., Ren, S., Sun, J. 2015b. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1026-1034. https://openaccess.thecvf.com/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html.

IndoML, 2021. Student Notes: Convolutional Neural Networks (CNN) Introduction. https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction (accessed 12 February 2021).

Ioffe, S. and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167.

Jin, G. and Roy, B. 2017. Hydraulic-fracture Geometry Characterization Using Low-frequency DAS Signal. *The Leading Edge* 36 (12): 975-980. https://doi.org/10.1190/tle36120975.1.

Jin, G., Mendoza, K., Roy, B. et al. 2019. Machine Learning-based Fracture-hit Detection Algorithm Using LFDAS signal. *The Leading Edge* 38(7): 520-524. https://doi.org/10.1190/tle38070520.1.

Kingma, D., P., and Ba., J. 2015. Adam: A Method for Stochastic Optimization. https://arxiv.org/abs/1412.6980.

Kwabi,E., 2013. *Mineral, Fluid, and Elastic Property Quantification from Well Logs and Core Data in the Eagle Ford Shale Play: A Comparative Study*. Master of Science thesis, The University of Texas at Austin. Austin, Texas (August 2013).

LeCun, Y., Bottou L., Bengio Y., and Haffner, P. 1998. Gradient-based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11), 2278-2324, November. https://doi.org/10.1109/5.726791.

MakeSense.AI. Online Open-source Image Labeling Tool. https://www.makesense.ai/.

McCulloch, W.S., Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* 5, 115–133. https://doi.org/10.1007/BF02478259.

Molenaar, M. M., Fidan, E., and Hill, D. J. 2012. Real-Time Downhole Monitoring of Hydraulic Fracturing Treatments Using Fibre Optic Distributed Temperature and Acoustic Sensing. Paper presented at the SPE/EAGE European Unconventional Resources Conference and Exhibition, Vienna, Austria, March 20-22. SPE-152981-MS. https://doi.org/10.2118/152981-MS.

Molenaar, M. M. and Cox, B. E. 2013. Field Cases of Hydraulic Fracture Stimulation Diagnostics Using Fiber Optic Distribution Acoustic Sensing (DAS) Measurements and Analyses. Paper presented at the SPE Unconventional Gas Conference and Exhibition, Muscat, Oman, January 28-30. SPE-164030-MS. https://doi.org/10.2118/164030-MS.

NN-SVG. Neural Networks Architecture Schematics. https://alexlenail.me/NN-SVG/.

OpenCV, 2021. Morphological Transformations. https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html (accessed 15 January 2021).

Pakhotina, I., Sakaida, S., Zhu, D. et al. 2020. Diagnosing Multistage Fracture Treatments with Distributed Fiber-Optic Sensors. *SPE Prod & Oper* 35 (2020): 0852–0864. SPE-199723-PA. https://doi.org/10.2118/199723-PA.

Polyak, B. T., 1964. Some Methods of Speeding Up the Convergence of Iteration

Methods. *USSR Computational Mathematics and Mathematical Physics*. 4(5), 1-17. https://doi.org/10.1016/0041-5553(64)90137-5.

Prabhu, 2018. Understanding of Convolutional Neural Network (CNN) — Deep Learning. Medium. (published online 4 March 2018). https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (accessed 19 January 2021).

Rosenblatt, F. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6), 386–408. https://doi.org/10.1037/h0042519.

School of Computer Science at the University of Auckland, 2021. Morphological Image Processing. https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm#compound (accessed 5 March 2021).

Script Reference, 2021. Neural Networks from Scratch. https://scriptreference.com/neural-networks-from-scratch/#neural-networks-backpropagation-summary (accessed 21 March 2021).

Simonyan, K. and Zisserman, A. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556.

Stork, A. L., Baird, A., F., Horne, S. A. et al. 2020. Application of Machine Learning to Microseismic Event Detection in Distributed Acoustic Sensing Data. *Geophysics* 85: KS149-KS160. https://doi.org/10.1190/geo2019-0774.1.

Tang, J. and Zhu, D. *In press*. Characterize Fracture Network Development Through Strain Rate Measurements by Distributed Acoustic Sensor (DAS). Paper submitted to 2021 SPE International Hydraulic Fracturing Technology Conference and Exhibition (IHFTC).

Tanimola, F. and Hill, D., 2009. Distributed Fibre Optic Sensors for Pipeline Protection. *Journal of Natural Gas Science and Engineering*. 1(4-5), 134-143, November. https://doi.org/10.1016/j.jngse.2009.08.002.

Ugueto C., G., A., Huckabee, P., T., Molenaar, M., M. et al. 2016. Perforation Cluster Efficiency of Cemented Plug and Perf Limited Entry Completions; Insights from Fiber Optics Diagnostics. Paper presented at the SPE Hydraulic Fracturing Technology Conference, The Woodlands, Texas, USA, February. SPE-179124-MS. https://doi.org/10.2118/179124-MS.

APPENDIX A

MODEL ARCHITECTURE SPECIFICATIONS

The four tables listed below summarize the model architecture specifications for the classification and localization models used in the synthetic case and the field case.

Table A.4.1 Synthetic case classification model architecture specifications.

Layer Name	Type	Output Shape	Kernel Size	Stride	Padding	Param #
input_1	InputLayer	224, 224, 3	-	-	-	0
conv0	Conv2D	224, 224, 5	11×11	1, 1	same	1820
bn0	BatchNormalization	224, 224, 5	-	-	-	20
act0	Activation	224, 224, 5	-	-	-	0
max_pool0	MaxPooling2D	112, 112, 5	2×2	2, 2	valid	0
conv1	Conv2D	112, 112, 10	5×5	1, 1	same	1260
bn1	BatchNormalization	112, 112, 10	-	-	-	40
act1	Activation	112, 112, 10	-	-	-	0
max_pool1	MaxPooling2D	56, 56, 10	2×2	2, 2	valid	0
flat	Flatten	31360	-	-	-	0
fc0	Dense	100	-	-	-	3136100
fc1	Dense	50	-	-	-	5050
class	Dense	1	-	-	-	51
Total params: 3,144,341 Trainable params: 3,144,311 Non-trainable params: 30						

Table A.4.2 Synthetic case localization model architecture specifications.

Layer Name	Type	Output Shape	Kernel Size	Stride	Padding	Param #
input_1	InputLayer	224, 224, 3	-	-	-	0
conv0	Conv2D	224, 224, 5	11×11	1, 1	same	1820
bn0	BatchNormalization	224, 224, 5	-	-	-	20
act0	Activation	224, 224, 5	-	-	-	0
max_pool0	MaxPooling2D	112, 112, 5	2×2	2, 2	valid	0
conv1	Conv2D	112, 112, 10	5×5	1, 1	same	1260
bn1	BatchNormalization	112, 112, 10	-	-	-	40
act1	Activation	112, 112, 10	-	-	-	0
max_pool1	MaxPooling2D	56, 56, 10	2×2	2, 2	valid	0
flat	Flatten	31360	-	-	-	0
fc0	Dense	100	-	-	-	3136100
fc1	Dense	50	-	-	-	5050
regress	Dense	2	-	-	-	102
Total params: 3,144,392 Trainable params: 3,144,362 Non-trainable params: 30						

Table A.4.3 Field case classification model architecture specifications.

Layer Name	Type	Output Shape	Kernel Size	Stride	Padding	Param #
conv0	Conv2D	56,56,96	11,11	4,4	valid	34944
bn0	BatchNormalization	56,56,96	-	-	-	384
max_pool0	MaxPooling2D	27,27,96	3,3	2,2	valid	0
conv1	Conv2D	27,27,256	5,5	1,1	same	614656
bn1	BatchNormalization	27,27,256	-	-	-	1024
max_pool1	MaxPooling2D	13,13,256	3,3	2,2	valid	0
conv2	Conv2D	13,13,384	3,3	1,1	same	885120
bn2	BatchNormalization	13,13,384	-	-	-	1536
conv3	Conv2D	13,13,384	3,3	1,1	same	1327488
bn3	BatchNormalization	13,13,384	-	-	-	1536
conv4	Conv2D	13,13,256	3,3	1,1	same	884992
bn4	BatchNormalization	13,13,256	-	-	-	1024
max_pool2	MaxPooling2D	6,6,256	3,3	2,2	valid	0
flat	Flatten	9216	-	-	-	0
dense0	Dense	4096	-	-	-	37752832
dropout0	Dropout	4096	-	-	-	0
dense1	Dense	4096	-	-	-	16781312
dropout1	Dropout	4096	-	-	-	0
class	Dense	3	-	-	-	12291
Total params: 58,299,139 Trainable params: 58,296,387 Non-trainable params: 2,752						

Table A.4.4 Field case localization model architecture specifications.

Layer Name	Type	Output Shape	Kernel Size	Stride	Padding	Param #
conv0	Conv2D	56,56,96	11,11	4,4	valid	34944
bn0	BatchNormalization	56,56,96	-	-	-	384
max_pool0	MaxPooling2D	27,27,96	3,3	2,2	valid	0
conv1	Conv2D	27,27,256	5,5	1,1	same	614656
bn1	BatchNormalization	27,27,256	-	-	-	1024
max_pool1	MaxPooling2D	13,13,256	3,3	2,2	valid	0
conv2	Conv2D	13,13,384	3,3	1,1	same	885120
bn2	BatchNormalization	13,13,384	-	-	-	1536
conv3	Conv2D	13,13,384	3,3	1,1	same	1327488
bn3	BatchNormalization	13,13,384	-	-	-	1536
conv4	Conv2D	13,13,256	3,3	1,1	same	884992
bn4	BatchNormalization	13,13,256	-	-	-	1024
max_pool2	MaxPooling2D	6,6,256	3,3	2,2	valid	0
flat	Flatten	9216	-	-	-	0
dense0	Dense	4096	-	-	-	37752832
dropout0	Dropout	4096	-	-	-	0
dense1	Dense	4096	-	-	-	16781312
dropout1	Dropout	4096	-	-	-	0
regress	Dense	2	-	-	-	8194
Total params: 58,295,042 Trainable params: 58,292,290 Non-trainable params: 2,752						