12-2021

Deep Learning Strategies for Pool Boiling Heat Flux Prediction Using Image Sequences

Connor Heo
University of Arkansas, Fayetteville

Deep Learning Strategies for Pool Boiling Heat Flux Prediction Using Image Sequences

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering

by

Connor Heo
University of Arkansas
Bachelor of Science in Mechanical Engineering, 2019

December 2021
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Han Hu, Ph.D.
Thesis Director

_____
Darin W. Nutter, Ph.D.
Committee Member

_____
Paul Millett, Ph.D.
Committee Member

Abstract

The understanding of bubble dynamics during boiling is critical to the design of advanced heater surfaces to improve the boiling heat transfer. The stochastic bubble nucleation, growth, and coalescence processes have made it challenging to obtain mechanistic models that can predict boiling heat flux based on the bubble dynamics. Traditional boiling image analysis relies on the extraction of the dominant physical quantities from the images and is thus limited to the existing knowledge of these quantities. Recently, machine-learning-aided analysis has shown success in boiling crisis detection, heat flux prediction, real-time image analysis, etc., whereas most of the existing studies are focused on static boiling images, failing to capture the dynamic behaviors of the bubbles. To address this issue, in the present work, a convolutional long short-term memory (ConvLSTM) model is developed to enable quantitative prediction of heat flux based on sequences of boiling images, where the convolutional layers are used to extract the features of the boiling images and the LSTM layers to identify the temporal features of the sequences. A convolutional neural network (CNN) model that is based on the classification of static images is also developed as a reference. Both models are trained with images of HFE-7100 boiling on silicon micropillar arrays at different steady-state heat fluxes. The results show that both CNN and ConvLSTM models have led to accurate predictions of heat flux based on the boiling images. In particular, the ConvLSTM model is shown to yield higher accuracy for heat flux predictions of completely unseen data, indicating a higher level of generality. Another focus of the present work is the forecasting capability of data-driven models using boiling images under transient heat loads. A CNN regression model is coupled with a one-dimensional LSTM model to enable a quantitative forecast of heat flux during boiling. The model is trained using image sequences of water boiling on planar copper surfaces with power ramp-up and has demonstrated a reliable forecasting capability.

Acknowledgments

My sincerest gratitude first goes to Dr. Hu, for guiding me throughout my graduate school career and for always being available and willing to help. None of this would have been possible without your hard work. Many thanks also go to our collaborators, Justin Weibel, Manohar Bongarala, Todd Kingston, and Hari Pandey. I would also like to thank committee members Darin Nutter and Paul Millett for their flexibility and willingness to assist. Finally, I must say thanks to my parents and friends, both local and abroad, for cheering me on and supporting me throughout this journey. Again, thank you.

Table of Contents

Chapter 1: Introduction

With the steady march of progress follows a steady increase in power, efficiency, and other metrics to which we prescribe value. In the development of computer chips, for example, engineers are constantly innovating new ways to pack a higher number of transistors into a smaller area. In the world of transportation, progress is measured by higher efficiencies and velocities. Despite the near-infinite diversity of such technologies, increases in output often come with an increase in power, and as such, an increase in heat. So, it follows that cooling technologies need to follow suit in order to keep new technologies feasible.

Narrowing our gaze onto the field of power electronics, we see a rise in liquid cooling technologies, as the heat transfer coefficients (HTC) of open-air cooling methods proved insufficient as power outputs rose. To gain further increases in cooling performance, engineers and researchers employ pool boiling to their cooling solutions, as it has been observed that as a liquid is boiled, there is a notable increase in its HTC, as long as the liquid is still within the nucleate boiling regime. Unfortunately, there is a limit to the benefits boiling can provide, as there is a phenomenon called critical heat flux (CHF), where a failure within the liquid occurs, drastically decreasing its HTC, and therefore its ability to transfer heat away from the surface to cool it. It is clear that controlling the heat flux in a liquid to avoid CHF is an essential problem.

To meet this challenge, researchers have begun to implement machine learning algorithms in pool boiling applications. The primary benefits of machine learning include automation and deep insight into behaviors that human analysis might overlook. Machining learning as a concept has been discussed since the 1960s [1], with researchers noting how much more useful a computer could be if it could learn on its own. In due time, computer scientists

were able to develop such algorithms, allowing computers to improve and learn independently of human input, given the proper environment and target. With the rising popularity of software packages such as TensorFlow and PyTorch, such powerful technology has become much more accessible to any aspiring user. Naturally, researchers have begun implementing machine learning into their work.

One of the most powerful implementations of machine learning is that of machine vision. With the proper set of algorithms, one can train computers to recognize features in images and perform complex tasks, such as classification and quantity regression. Researchers have already implemented machine vision for pool boiling applications. Hobold and da Silva trained machine learning models both to classify boiling regimes [2] and quantify heat flux [3]. Other work has focused on analyzing the boiling situation in real-time [4] and on the specific detection of CHF [5]. Machine learning has also found use in modelling and building a deeper understanding of the underlying physical behaviors [6]. A common thread in these papers is that they focus on the current time-step. This limits the ability of the algorithms to look ahead and prevent CHF in advance, necessitating faster sampling and more processing power to match.

In this thesis, the analysis of pool boiling with machine learning algorithms is extended by implementing the prediction of future behaviors. With this capability, any undesirable behaviors, such as CHF, can be predicted ahead of time by the algorithm, and the appropriate measures can be taken to prevent failure. In short, predictive models can enable prevention, instead of focusing on alleviating any issues. In the first chapter, the groundwork code is developed to classify pool boiling images and videos in a steady-state environment using Convolutional Neural Networks (CNN) and Convolutional Long Term Short Memory

(ConvLSTM). Then, further work involving classification-based boiling heat transfer prediction and steady-state heat flux prediction are discussed. In the second chapter, CNN regression is implemented, alongside Multilayer Perceptron (MLP) model architectures and Long Short Term Memory (LSTM) in order to implement transient heat flux prediction.

Chapter 2: Steady-State Boiling Heat Transfer Prediction Using CNN and ConvLSTM

2.1: Background and Motivation

While the history of CNNs and ConvLSTMs is not as long as that of other pillars of computer science, the popularity of these algorithms has only grown exponentially in recent times, due to their ability to simplify the path to implementing machine learning, most especially computer vision. Programming computers to recognize objects in images and videos, a task once thought to be no less than sci-fi, is now so commonplace as to be found in applications ranging from research and military to social media and entertainment.

With software packages such as Tensorflow and PyTorch, the implementation of machine learning, and more specifically CNNs/ConvLSTMs has never been easier. This has led to a rise in researchers implementing computer vision into their work. As mentioned in the introduction, many research groups have begun implementing neural networks in work concerning pool boiling in cooling applications.

Our work is motivated for similar reasons, namely the relative ease in using neural networks to both analyze and generate results. More specifically with regards to computer vision, a well-trained CNN could, in theory, replace expensive measuring devices that need to be placed within the boiling chamber, as the CNN could recognize boiling regimes and even measure quantities based on visual cues, such as bubble size [3].

However, because CNN models are feedforward networks, there is no relationship between different data points, which would be images in this work. During training, a model is shown each image and its specified class, but no information from other images is stored. For simple classification cases, this poses little to no issue. However, when the images are taken

from videos, by isolating each frame, information stored in the time dimension is lost. For example, the growth of a bubble over time would be impossible to track using a CNN model. To overcome this deficiency, a ConvLSTM model was trained on the same images as the CNN model. However, several images were combined into each input array, with the hope that the ConvLSTM model would learn not only to identify qualities of each image but also attributes that change over time and depend on previous timesteps. With regards to this specific work, the aim is that ConvLSTM models would be able to capture inherently time-dependent bubble dynamics and show an improvement in comparison to traditional static image CNN analysis.

Another knowledge gap identified during the outlining of this work is that there is no strong consensus on the dominant transport mechanism causing the CHF condition, despite the development of several theories. Two popular models are the hydrodynamic model, first proposed by Zuber [7], and the force-balance model, proposed by Kandlikar [8]. In the hydrodynamic model, the theory is that CHF is caused by a hydrodynamic instability of vapor columns. This would mark said vapor columns as key features in any pool boiling image analysis. In contrast, the force-balance model theorizes that CHF is caused by forces from evaporation momentum overcoming the surface force. In this case, key features to observe would be the contour and contact line of the macrobubble.

2.2: Literature Review

Despite its relatively simple premise, pool boiling is constantly being iterated upon to match the cooling demand that higher power outputs necessarily demand. Whether it is by increasing the overall heat flux with better fluids and materials, increasing the margin before

experiencing undesirable behaviors (critical heat flux), or other methods, researchers are constantly working to improve the effectiveness of pool boiling.

A review of pool boiling CHF done by Liang and Mudawar first compared different models for CHF mechanisms, then assessed these models and their correlations. The author identified 5 CHF mechanisms present in the literature: bubble interference, hydrodynamic instability, macro-layer dryout, hot/dry spot, and interfacial lift-off. Of these 5, Zuber's hydrodynamic instability theory was found to be most popular due to its "mechanistic formulation and theoretical appeal". However, recently the interfacial lift-off mechanism has received significant experimental validation. The authors end the review by concluding that despite the large body of work done to understand and study CHF, "major data gaps" were found with regards to some of the major parameters. Specific studies are recommended, including microphotographic analysis of near-wall interfacial features.

One of the more recent innovations in pool boiling is the implementation of nanofluids and nanostructures in boiling setups. For example, Lee et. al enhanced critical heat flux by boiling both AL_2O_3/water nanofluid and SiC/water nanofluid [9]. An enhancement of critical heat flux in this context refers to an increase in the usable heat flux value before experiencing critical failure in the boiling crisis. The researchers noted that the CHF enhancements also improved the wettability of the liquid film on the surface "due to nanoparticle deposition". More recently, Hwang et. al performed a similar study to implement nanofluids to enhance CHF. The researchers note that the primary barrier to implementing nanofluids in the industry is their instability over long periods "due to sedimentation and aggregation". To combat this, the researchers use "lightweight negatively charged TEMPO-oxidized cellulose nanofibers (CNFs)"

in water and perform experiments. CNFs have low density and the negative charge creates repulsion between particles, which results in the elimination of both sedimentation and aggregation while maintaining the enhanced CHF properties of nanofluids.

Another avenue of study for pool boiling is to examine the boiling setup itself. Geometric parameters can be optimized to maximize the cooling effectiveness of pool boiling setups. Ghaffari et. al performed such a study, studying the limits of heat transfer for the immersion cooling of a microprocessor while changing various design parameters [10]. The researchers note that the main barriers to optimized immersion cooling are the boiling crisis and the decreased reliability of immersed components, which are usually designed for air cooling applications. Ultimately, the researchers found an optimal geometry for an internal heat sink, showing the viability of more advanced two-phase immersion cooling solutions for modern electronics. Aside from incremental improvements to pool boiling setups and materials, much more novel methods have been proposed and studied to improve cooling effectiveness. For example, Sinha-Ray et. al demonstrated a novel method of cooling high-power microelectronics in microgravity situations [11]. The method involves introducing a "swing-like" motion using the heat removed during pool boiling. This swinging motion would potentially be beneficial in shedding large vapor bubbles that encapsulate the heaters in microgravity situations, due to the lack of a buoyancy force to remove the bubbles. On top of this, the researchers have also experimented with nano-textured surfaces, another common avenue of research in the field of pool boiling.

One of the most common applications of machine learning algorithms, especially CNN and other convolution-based models, is computer vision. Convolutions allow models to identify and differentiate between different features in images, such as curves, lines, and unique shapes.

A common beginner's case study for such an application is that of text recognition in written documents. In 1998, Yann et. all published a seminal paper describing the use of a multilayer neural network, trained with a gradient-based learning technique, to identify handwritten characters on physical mediums [12]. Following this work, researchers increased the complexity of the task given to the models by classifying images, both with single and multiple labels. Wei et. al introduced a new deep CNN infrastructure called Hypotheses-CNN-Pooling [13]. Using "object segment hypotheses" as input, the researchers were able to train the model to classify images with multiple objects of interest with multiple assigned labels. Wang et. al performed a similar study, aiming to train a neural network to identify multi-label images [14]. However, instead of a deep CNN model, a recurrent neural network (RNN) was combined with a CNN. The CNN-RNN architecture was found to perform better on public benchmarks than other state-of-the-art models. Of course, CNN architectures are not the only option for computer vision applications. Jiang et. al compared the image classification performance of several model architectures, namely CNN, CapsNet, and Fully Convolutional Network (FCN) [15]. Evaluating the models' performance classifying two public datasets, the researchers found that both the CNN and FCN models performed similarly, while CapsNet, which lacks a pooling method to preserve data, did perform to a high level. However, due to the larger amount of data, CapsNet needed more time to train. It was found that all 3 models performed adequately on single-class datasets but struggled with multi-class datasets.

CNNs and hybrid models have been applied to more focused applications, such as high-resolution aerial images [16] and hyperspectral image classification using deep CNNs [17] and region-based CNNs [18]. Another powerful application is that of video classification. For example, Ye et. al implemented two-stream CNN, with one input stream for static frames and

another for motion optical flows to classify videos [19]. The model architecture was found to be competitive with state-of-the-art video classification methods, especially considering that the two-stream CNN was less data-intensive. Zha et. al also implemented CNN for unconstrained video classification, using only images to train the model. The researchers found the CNN to be a "black-box feature extractor", concluding that any improvements to the image classifier would also improve the video classifier.

However, aside from specializing and adapting CNN architectures, another application of CNNs and convolution is in the ConvLSTM model architecture. The Long Short Term Memory (LSTM) portion of the model implements special logic gates that can recall and forget temporally spaced information, such as how values change over time. By adding convolutional layers to the model, not only values but also features can be understood through time. For example, Khan et. al implemented ConvLSTM models to assess water quality in thermal images, as the thermal images were sequentially organized [20]. This allowed the researchers to take advantage of the LSTM portion of the architecture. One interesting paper published by Song et. al documents the use of a ConvLSTM model for object detection in videos [21]. The model architecture stacks several modules with specific purposes to achieve this task, with the model performing as well as other state-of-the-art methods on public benchmarks.

Classification is certainly an important task for machine learning models. Still, with how powerful the technology is, researchers have tasked models with a wider range of tasks. For example, another common task for machine learning algorithms is to make a prediction based on input data. Now, instead of simply identifying data at a given timestep, models are being trained to not only learn the underlying patterns in data but also to use that information to simulate or

predict continuations of the data. For such a task, it is much more common to see ConvLSTM model architectures, due to the LSTM's ability to understand changes over time. Kim et. al used a ConvLSTM network to predict rainfall information using radar reflectivity data [22]. Comparing the predictions to the actual data, the ConvLSTM model had a root mean squared error (RMSE) 23% less than predictions using linear regression. Showing the flexibility of the ConvLSTM architecture, Kelotra et. al used a Deep-ConvLSTM model to make stock market predictions [23]. The mean squared error (MSE) and RMSE dropped to 7% and 2% respectively, showing the power of the model architecture for making predictions. Combining both special and temporal data, a "Hetero-ConvLSTM" model was proposed by Yuan et. al to use a wide range of data, including weather, environment, road conditions, and traffic volume, to predict accidents [24]. The researchers concluded that the predictions were "reasonably accurate", and significantly more accurate than other traditional methods.

2.3: Methodology

Previous work by Hobold and da Silva [2] has shown that it is possible to train neural networks to identify and classify different boiling regimes, based on visualization alone. This being a major inspiration and springboard for the work in this thesis, it was deemed necessary to first reach this capability. With this in mind, a CNN was trained to classify images from various boiling regimes. Additional code and post-processing allowed for evaluation of model performance and further insights into the phenomena.

Once this base capability was reached, the next step was to train both a CNN and a ConvLSTM model to classify boiling images, not for their boiling regimes, but rather their associated heat flux value. The data used for both models came from high-speed video taken of

HFE-7100 boiling on silicon micropillar arrays at different steady-state heat fluxes. These high-speed videos were originally saved in CINE format and were saved to the necessary data formats using Phantom Camera Control [25]. For the CNN model, each frame was saved as an image in JPG format and placed in a corresponding class folder based on heat flux value. Once all the images were saved, a software package called "split-folders was used to separate the images into training, testing, and validation datasets with a 64/20/16 split [26]. For experiment design, this dataset was designated as DS1.

When testing the generality of the CNN and ConvLSTM models, two new datasets, DS2 and DS3, were generated. All three datasets came from the same boiling setup, with DS2 using the same working fluid and surface as DS1 and being collected during a different test. DS3 used a different working fluid and surface, those being deionized water and a copper flat surface. This would later enable the testing of the generality of the two models, by having them predict on a similar image dataset and a completely different image dataset. For the generality tests, instead of classifying the images by heat flux, they were classified into 3 classes: onset of nucleate boiling, bubble coalescence and interference, and CHF. This could theoretically train the models to detect CHF.

While the ConvLSTM model used the same source for data, the format was AVI videos instead of individual JPG images. To do this, the full CINE file for each heat flux was saved to AVI. Then, AVI Splitter was used to split the full video into several shorter videos of equal length [27]. Later, the code to train the model would take a specified number of frames from each of these shorter videos to create the necessary data. For both datasets, videos corresponding

to heat flux values of 36.14 W/cm^2 and 36.73 W/cm^2 were left to the side to be used as unseen testing data once both models were trained.

The heat fluxes for each dataset were measured using Fourier's Law for a one-dimensional heat transfer case. The temperature gradient was generated by heaters placed within a copper block, and the temperatures were measured using thermocouples placed at specific locations. Primary sources of uncertainty were the thermal conductivity of the copper block, the temperature measurements from the thermocouples, and the measurements of the positions of the thermocouples. Taking all this into account, the estimated uncertainty of the heat fluxes was calculated to be 0.69 W/cm^2.

2.3.1: Algorithms and Model Architecture

The core mathematical function in this work is that of convolution. Convolution is a mathematical function performed on a multi-dimensional array, wherein a filter, or kernel, is passed over the array and matrix multiplied with the values underneath that filter. Assuming the original array is a square N by N grid, and the filter is a square filter with size m by m, the output of the convolution function will be a square grid with size (N – m + 1) by (N – m + 1). For example, given a simple 4 by 4 array and a basic 3 by 3 kernel, one can perform convolution to output a new array of values with size 2 by 2, as seen below:

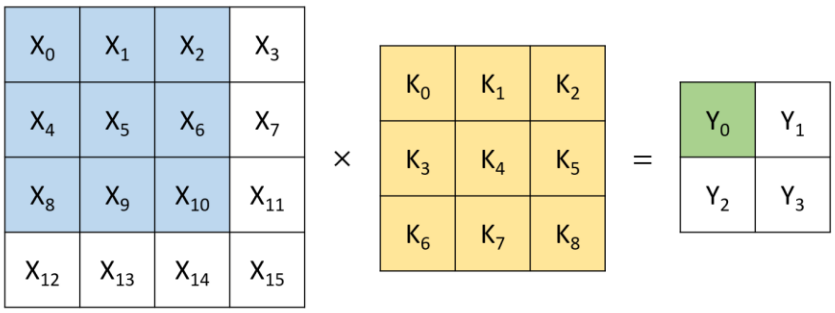


Figure 1: Simple schematic of a single convolution operation. The blue region of the original array is multiplied by the filter, in yellow, resulting in the upper left corner of the output array, in green. To complete the operation, the yellow filter would be multiplied by all 4 possible blue regions of size 3 by 3.

The convolution operation is a key component in the aptly named Convolutional Neural Network (CNN) and the Convolutional Long Short Term Memory network (ConvLSTM), both of which were examined and implemented throughout this research. This is because convolutions have been found to be key in image and pattern recognition and are an essential component in computer vision. Over time, specialized filters have been found that are able to extract specific features. For example, a filter can be specifically designed to capture vertical lines, while another is equipped to discover circles in an image.

However, machine learning models rely on more algorithms than just convolutions. A wide variety of algorithms often referred to as model layers, are combined into single models in order to achieve the desired performance. In this chapter, the additional layers include max pooling, dropout, flatten, dense, and LSTM layers.

Max pooling is a relatively simple function performed on arrays. For 2-D arrays, the original array is divided into non-overlapping regions of size A by B, and the maximum value in each region is used to populate the output array, with each region's position corresponding to the

position of their output in the new array. Below is an example of a max-pooling function performed on a 4 by 4 array with sampling size 2 by 2:

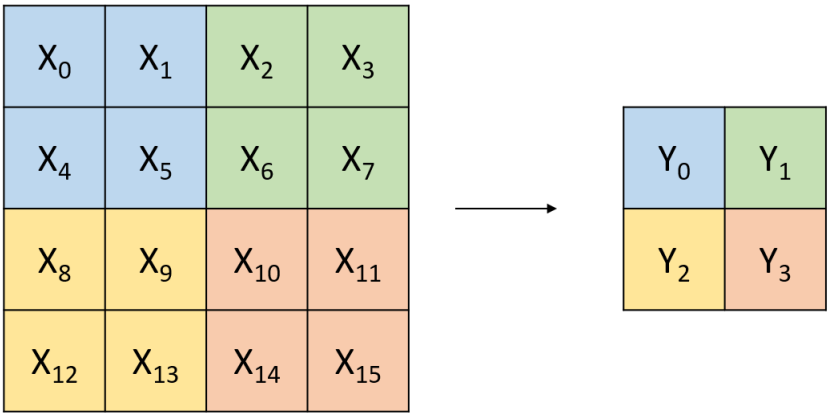


Figure 2: Simple schematic of max-pooling operation. The highest value in the blue region is output to the blue value in the output array. This is repeated for each region, represented by the other colors.

The goal of pooling layers in general is to "achieve spatial invariance by reducing the resolution of the feature maps" [28]. Max pooling layers have been found to help models converge faster "by selecting superior invariant features" [29]. Dropout layers remove a given percentage, represented by a float variable between 0.0 and 1.0, from the given dataset, usually the training data. Changing the dropout value can have a significant impact on model performance, as setting the value too high will sacrifice overall performance, while setting the value too low will increase the risk of overfitting, reducing the model's generality. Flatten layers are an important component of neural networks, as they connect multi-dimensional image data arrays to dense layers, discussed further below. To accomplish this, the layer will take one value at a time in the original array and feed it into a new one-dimensional array. The order of the data is preserved during this process and the data is then ready to be processed by the dense layer.

The dense layer (also often called the fully connected layer) consists of nodes or neurons that are all connected to one another. These connections are each given weights during training and are what allow the model to make its final decisions/predictions. The number of neurons usually corresponds with the number of desired outcomes. For a classification case, such as the work described in this chapter, we would have as many neurons as there are boiling regimes or heat flux values, and we would interpret the dense layer's outputs by examining the node with the highest value, understanding this to be the class label with the highest probability of matching the input data according to the model.

On the right is the model architecture used for the CNN model used for the work described in this chapter. The input is an array of shapes (128, 128, 3), which refers to the height and width of the image and 3 channels of RGB. The model consists of 5 sections: 3 convolutional layers, 1 flattening layer, and a fully connected layer. The 3 convolutional layers are identical and are stacked to provide higher performance, with the tradeoff of more trainable parameters and longer training time.

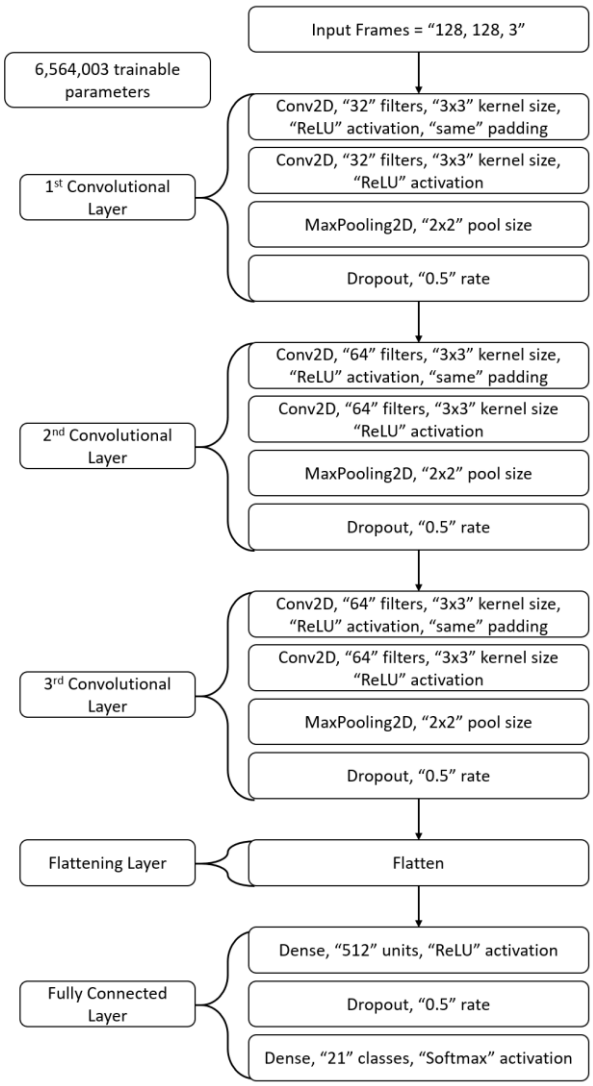


Figure 3: Model architecture and parameters of the CNN model with 3 convolutional layer stacks and 2 processing layer stacks.

Each convolutional layer contains 2 Conv2D layers with a 3 by 3 kernel size and ReLU

activation. The first convolutional layer uses 32 filters while the following 2 layers have 64 filters. The first Conv2D layer in each convolutional layer also contains the same padding.

ReLU activation refers to rectified linear units. Activation functions as a concept date back to 1980, where researchers noted pattern recognition could be modeled using specific activation functions, including the ReLU activation function [30]. The function can be defined as $f(x) = \max\{0, x\}$. By implementing same padding, the shape of the kernel and the shape of the output of each convolution is kept the same by adding zeros to the array [31].

After another Conv2D layer with no padding, the resulting array is processed through a max-pooling function with a pool size of 2 by 2. The convolutional layer finishes processing the data by dropping out 50% of the data to prevent overfitting. This is repeated twice more before the data is flattened and processed through the fully connected layer. The first dense layer contains 512 neurons with ReLU activation. This first dense layer is often called a "hidden layer", as this layer is essentially a black box, where the inputs are related to the outputs using weights determined by the model during training. It has been shown that any continuous function can be approximated using finite linear combinations of fixed, single variable functions [32]. After dropping out 50% of the data, a final dense layer is used with as many neurons as classes to determine from, and a Softmax activation is employed. Since the dataset, in this case, contained images from 21 different heat flux values, the number of classes was set to 21. Softmax activation "converts a vector of values to a probability distribution" [33]. This final dense layer is where the model decides on what class the input data belongs to, which is based on whichever class is given the highest probability.

Before examining the architecture of the ConvLSTM model, it is important to first understand the concept of LSTM. In contrast to CNNs, which are a feedforward neural network,

LSTM is a recurrent neural network. This means that each LSTM unit is able to also obtain feedback and information from other units. Below is a simplified schematic of how LSTM cells connect.

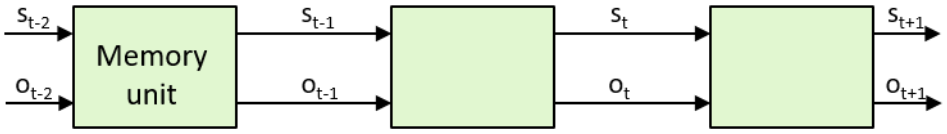


Figure 4: Simplified schematic of LSTM memory units. Between cells at different timesteps, information regarding the system state, s, and model output, o, are saved and transferred between cells.

While many of the post-processing layers are similar to that in the CNN model (dropout, flatten, dense), the key difference is that the ConvLSTM model has a single ConvLSTM2D layer, which also takes in a different shape of input data. Instead of a single image data array, the ConvLSTM2D layer takes 10 combined arrays of image data of specified height and width with 3 channels of RGB.

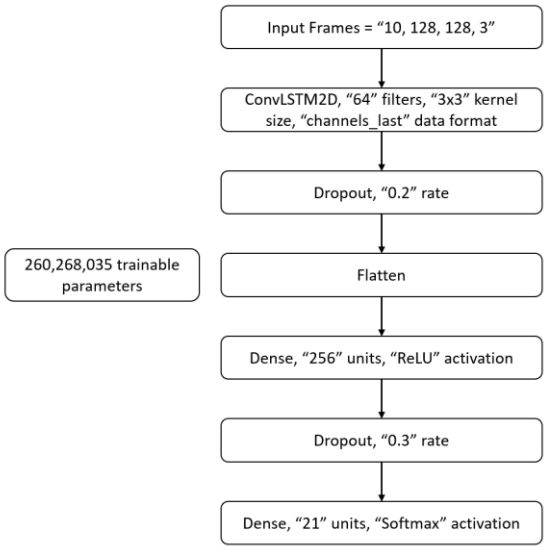


Figure 5: Model schematic and parameters for the ConvLSTM model.

The ConvLSTM2D layer uses 64 filters with a 3 by 3 kernel size and the "channels_last" data format, which refers to the location of the number storing the amount of RGB channels. To remain consistent with the input data shape, the last index is set as the RGB channels. This layer performs all the functions of a normal LSTM, but the key transformations are now convolutions instead of normal arithmetic functions [34].

2.3.2: Code Structure

All the codes in this thesis use Keras and Tensorflow as the backend for machine learning functions. Models were built sequentially, meaning that each layer was added on one at a time, with their parameters set as they were added. The CNN model was compiled with the RMSprop optimizer [35]. The loss was represented using categorical cross-entropy, which acts in the same way that Softmax activation does, with the goal of using a probability distribution to determine the predicted class. Accuracy was also added on as a metric and was plotted after training to assess the efficacy of the training process. Both accuracy and loss for training and validation per epoch were saved through the history object, which is output from Keras' fit function, which trains the model. The ConvLSTM model was compiled using the SGD optimizer with a learning rate of 0.001 [36]. Loss is represented in the same way as the CNN model, with categorical cross-entropy, and accuracy is also added as a metric.

Once the images for the CNN model were prepared as described earlier, the training, testing, and validation images were passed through "ImageDataGenerator" objects through the use of "flow_from_directory". The 15,229 images were split into training, testing, and validation sets with a 64/20/16 percent split. The image data were also downscaled to fit in the range between 0 and 1. The batch size was set to 25 and the image width and height were both set to 128. While the training and validation data were shuffled, the test data was left unshuffled in order to compare model predictions to real heat flux values. Before finally training the model, three callbacks were defined for the model: ModelCheckpoint, TimeHistory, and EarlyStopping.

The purpose of ModelCheckpoint is to monitor a chosen metric at the end of each epoch (in this case, loss) and to save the model if there is an improvement in the metric. This prevents unruly end behavior from negatively affecting model performance if a well-performing set of

weights was determined earlier in the training process. The TimeHistory callback serves to simply monitor how long each epoch took and save it, much like how accuracy and loss are saved into a history object. Finally, the EarlyStopping callback is meant to stop the training process if the performance of the model doesn't improve for a set number of epochs. Here, the patience (number of epochs without improvement) was set to 7. The model was trained using "fit_generator" for 50 epochs on the GPU partitions on Bridges,

Once training was completed, the resulting history object was called to save the loss, validation loss, accuracy, and validation accuracy per epoch. These statistics were then written to an output text file, along with the total training time and average training time per epoch. All models, regardless of architecture, were saved to the ".h5" format, so that both the layers and weights could be loaded in one step during model evaluation. The trained CNN model was tested with the test data set using Keras' "predict_generator". Since the output from the generator was a prediction distribution, the maximum value per entry was taken, as that would be the model's indication of a predicted heat flux value. The final comparison between the model's predicted heat flux versus the actual heat flux was done using scikit learn's "confusion_matrix" function [37] and visualized with seaborn [38].

To create the attention maps representing the CNN model's areas of interest during the prediction process, the following process was utilized. First, a layer of the model's architecture was chosen to be visualized. In most cases, this would be the final dense layer, but in rare cases, clearer visualizations were found in the hidden layers. Then, the activation of the chosen layer was changed from softmax, which would be used for classification, to linear, allowing each point of the image array to have a given value from the model. After loading the image through its file path and having the model predict its class, a function called "visualize_cam" from Keras' "io"

library was used to generate the attention map, assigning values to the activation of each point in the image array [39]. By applying a color gradient to the activation and laying the activation over the original image, an attention map was then created.

While the overall structure of the code to train the ConvLSTM models was similar to that of the CNN models, there were a few major differences, most visibly in the I/O. The CNN code took in single images arranged in training, testing, and validation folders in advance. In contrast, the ConvLSTM code required that all the images in single folders corresponding to their classes. Originally, the data for the ConvLSTM model was prepared with sequences in an end-to-end arrangement, meaning that no frames were found in common between different sequences. However, to increase the amount of data provided to the model, rolling sampling was implemented instead. Below in Figure 6b, a visual schematic of the rolling sampling method can be found. As it was observed that for subsequent sequences, all but the head and tail frames were the same, the code was optimized to copy the identical frames into the next sequence. This absolved a memory issue that occurred originally when implementing the rolling sampling, due to calling OpenCV to read video data too many times.

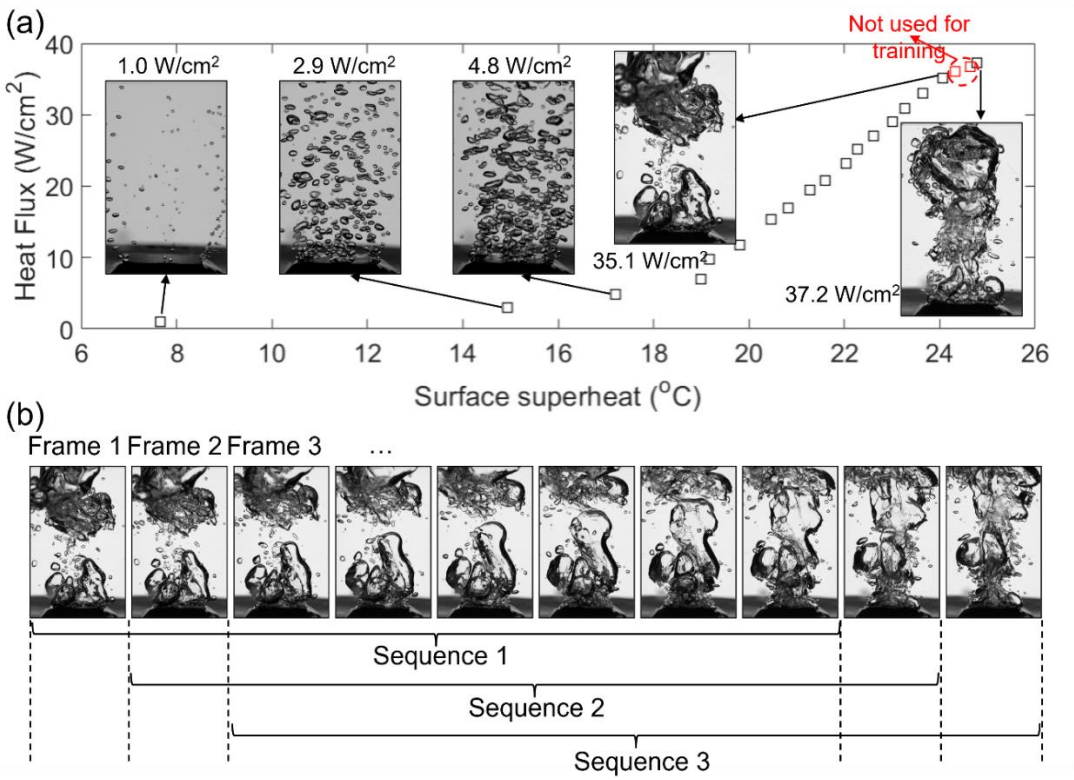


Figure 6: (a) Visual representation of the classification of images based on heat flux, including the 2 unseen classes used later to test interpolation abilities of the two models, (b) Schematic showing how rolling sampling is performed.

Before settling on a final sequence length, which would be 16 frames, parametric testing was performed to determine the benefits and costs of changing sequence length. While increasing sequence length generally had a positive effect on the accuracy of the ConvLSTM model, it was observed that increasing the length after 12 frames led to diminishing returns. A length of 16 frames was ultimately chosen, as it was the maximum number of frames that could be sampled from each video before instabilities and memory issues were posted by the code. In the following results and discussion section, a deeper parametric study, exploring both the sequence length and temporal gaps between images will be discussed.

The rolling sampling was implemented inside a function called “create_data”. As inputs, the function took in a data directory and an array listing all the classes present in the data. To perform the sampling, a series of nested loops were used. These loops took advantage of Python’s ability to abstract loop parameters, meaning that instead of defining a loop variable and iterating, the code could instead know to loop over filenames or classes without needing to index. The first and outermost loop served to loop over each class within the list of classes. For any single class, the code would then compile the full file directory of each image file and append the directories to a list, making sure to sort the list numerically as well. Then an inner for loop was initiated, iterating a counter variable i over the length of the list of directories, and served to append an array of consecutive image directories with length equal to the desired sequence length. A simple if statement was used to ensure that only full-length sequences were added, by stopping the loop once the last possible full sequence was appended to the list. Finally, a final nested for loop was defined, looping over each sequence of consecutive image directories in the full rolling sampled list. As the code looped over each directory, it would use OpenCV to convert each image to numerical image data, along with resizing the images. This would then result in a list of arrays of image data sequences. Barring the very first sequence, all subsequent sequences took advantage of the fact that in rolling sampling, all but the first and last images are the same in the next sequence. This optimization, as mentioned above, made it possible to run the code without running into any memory issues due to calling OpenCV too many times. An array of output values was also created within the main loop. First, an array of zeros with length equal to the number of classes was created for each video, mirroring the format of the probability distribution. Then, for any processed video from the nth class, the nth value in the array of 0’s

was flipped from 0 to 1. This represented the “right answer” to be output by the model during testing.

Finally, scikit learn’s “train_test_split” was used to split the data into training and testing with an 80/20 split, and a further 20% split for validation data was defined in the fit function, resulting in a 64/20/16 split, just like in the CNN code. Using the same set of callbacks, the ConvLSTM model was trained for 20 epochs. The evaluation of the ConvLSTM model was done in the same manner as that of the CNN model.

2.4: Results and Discussion

The work in this chapter can be divided into two major sections: Heat Flux Prediction Using ConvLSTM Model Trained on Image Sequences, and CNN and ConvLSTM for Boiling Crisis Detection – Generality & Interpretability.

2.4.1: Heat Flux Prediction Using ConvLSTM Model Trained on Image Sequences

Firstly, the accuracy and loss per epoch were examined to ensure that the models trained in a positive manner. In Figure 7, both training and validation accuracy for the CNN model quickly rise in the early epochs before plateauing at high accuracy. However, the validation accuracy is consistently lower than the training accuracy. This indicates the effectiveness of the dropout layers in the model architecture in preventing overfitting. A similar trend is mirrored in the training and validation loss, further confirming that the model behaved properly during training.

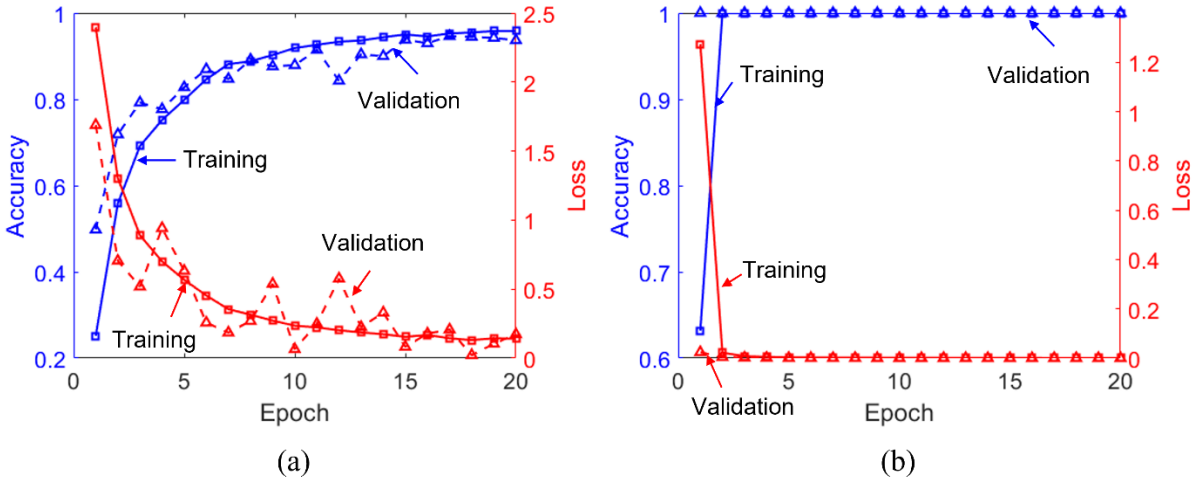


Figure 7: (a) Accuracy and loss per epoch for CNN model during training on steady-state heat flux images and (b) accuracy and loss per epoch for ConvLSTM model during training on steady-state heat flux images

After confirming that there was no overfitting or other undesirable behaviors during training, both the CNN and ConvLSTM models were tasked with predicting on their respective test datasets. To compare the predictions with the true results, a confusion matrix was generated for each model. For each image or video in the testing set, it was plotted on the matrix with its horizontal position determined by the model's predicted heat flux and its vertical position determined by its true heat flux. Ideally, all points would lie along the diagonal, indicating that all the predictions were correct. In Figure 8, one can see that both the CNN and ConvLSTM model generally predicted correctly on their testing sets, with most of the points lying along the diagonal. With the rolling sampling, the ConvLSTM model predicted with a higher accuracy than the CNN model. During training and validation, it was observed that accuracy and validation accuracy would jump to 100%, causing concerns of overfitting. The ConvLSTM model's perfect performance on its testing set is also somewhat abnormal but could be explained by again observing that subsequent sequences in the rolling sampling method are mostly

identical. Further work on the model design, including a higher dropout rate, could potentially keep the model's performance within a more realistic range. Still, considering the ConvLSTM model's high performance also on the testing set, which is excluded from training, there is still confidence in the model's performance. Testing on the unseen data would provide another strong metric to judge the models by.

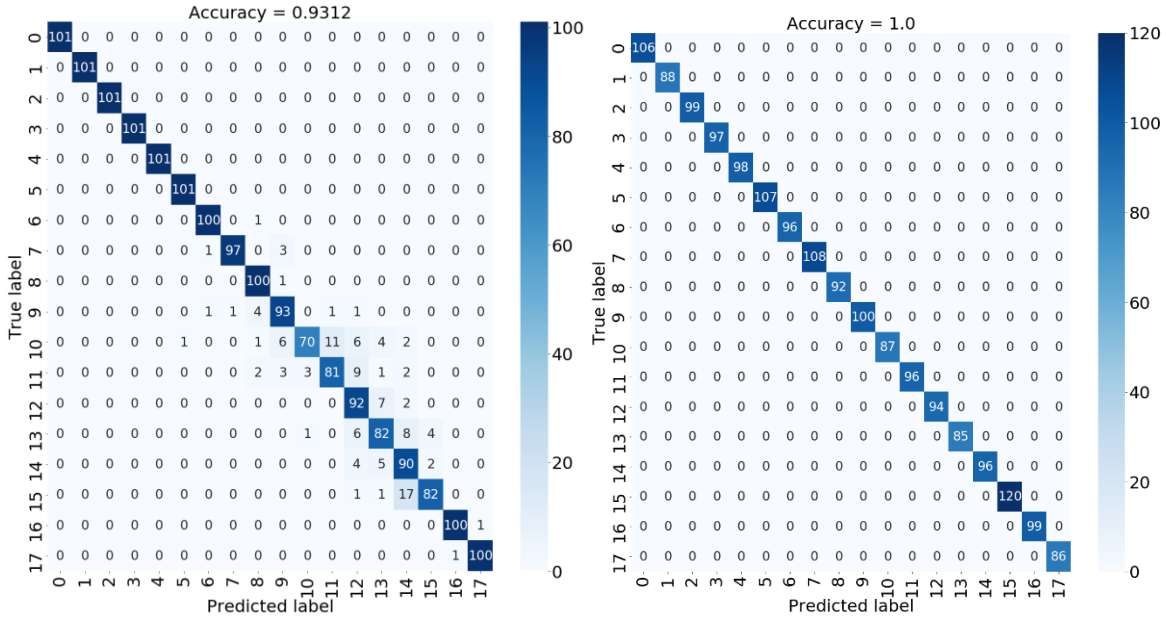


Figure 8: Confusion matrices for CNN and ConvLSTM models after predicting on their respective testing sets.

However, since the goal of the models is to determine a specific quantity, as opposed to a qualitative attribute, it was found that a confusion matrix might not provide the most accurate assessment of the models' performance. This is because a confusion matrix will consider any prediction that isn't exactly correct as incorrect, no matter how close the prediction was to the true value. With this in mind, the predicted values and true values were used to calculate a weighted average, by which the model's performance could more accurately be assessed.

During the data preparation phase, images/videos from certain heat flux values were kept aside to be used as "unseen" data to test the models' ability to interpolate predictions. Due to the nature of the models, no prediction would be perfectly accurate for individual data points, as it would be impossible for the models to predict a class (quantity) that they were not trained on. However, by taking the weighted average for an entire group of images, the heat flux value that the models predicted could be interpolated.

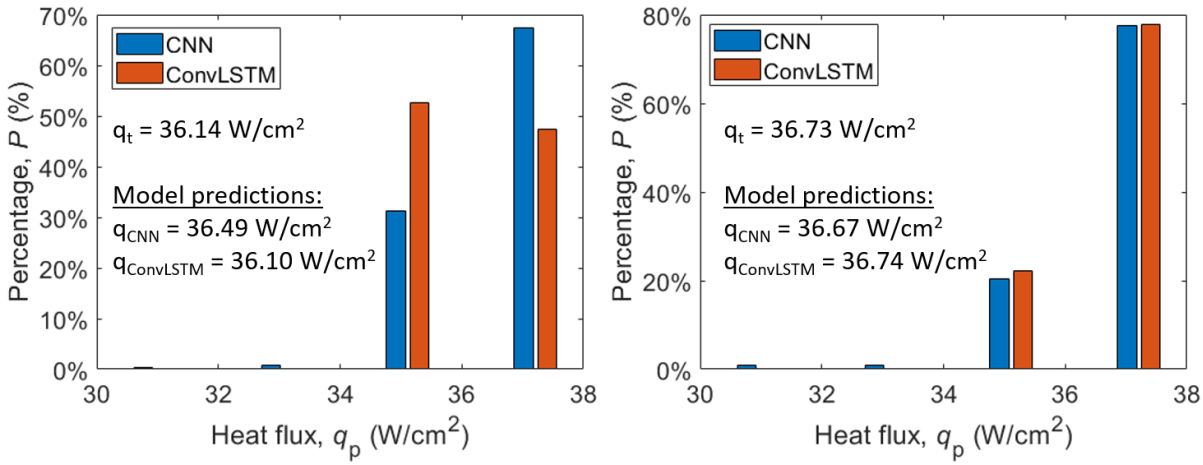


Figure 9: (a) CNN and ConvLSTM classification of images/videos from experiment with steady-state heat flux q_t = 36.14 W/cm^2, (b) CNN and ConvLSTM classification of images/videos from experiment with steady-state heat flux q_t = 36.73 W/cm^2. The estimated uncertainty of the heat fluxes was calculated to be 0.69 W/cm^2.

In Figure 9 above, the predictions by the two models on the unseen data are presented in a histogram. Both models were trained with a dataset that included steady-state heat flux values equal to 35 and 37 W/cm^2 with no intermediate values. For the unseen data with the lower heat flux value, q_t = 36.14 W/cm^2, both models predict 35 and 37 W/cm^2 at about the same rate, which draws the weighted average close to 36. With the unseen heat flux value set to q_t = 36.73 W/cm^2, the models both show bias towards the higher heat flux value, pushing the weighted

average higher as well. Looking at the results in a qualitative manner, both models show promise in interpolating on unseen data that is positioned between previously seen data.

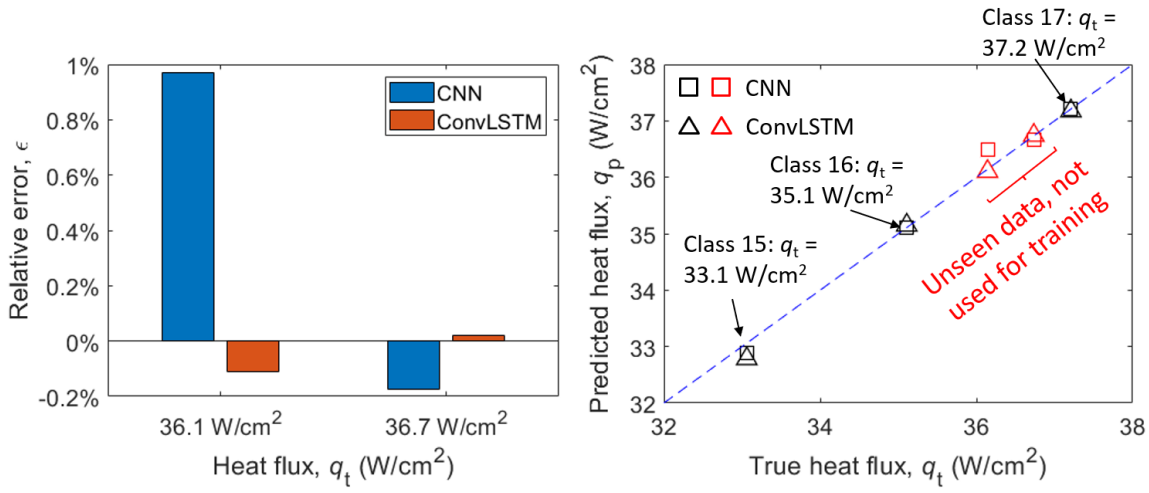


Figure 10: (a) Relative error between true steady-state heat flux values of unseen data and weighted average of CNN and ConvLSTM predictions on unseen data. (b) Demonstration of the interpolation performed by both the CNN and ConvLSTM models with the unseen data. The estimated uncertainty of the heat fluxes was calculated to be 0.69 W/cm^2.

Calculating the relative error between the true heat flux values of the unseen data and the weighted averages of the two models' predictions, we have the results in part (a) of Figure 10. In both the confusion matrix and weighted average analyses, the ConvLSTM model with rolling sampling displayed a higher performance, with a higher classification accuracy on testing data and a lower relative error on the unseen dataset. This shows the potential for ConvLSTM in improving machine vision performance in applications where data is arranged with respect to time. As opposed to single image classifications, when working with videos or otherwise subsequent images, the LSTM portion of the ConvLSTM architecture shows promise in capturing behaviors and changing features over time.

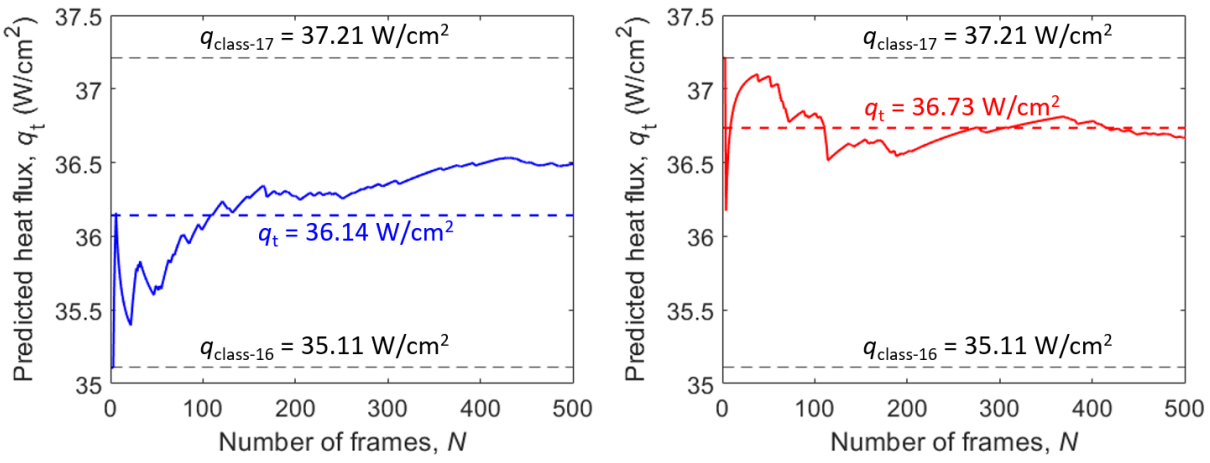


Figure 11: Plotting the predicted heat flux for the CNN model when given images from (a) q_t = 36.14 W/cm^2 and (b) q_t = 36.73 W/cm^2 versus the number of frames used to populate the testing dataset. While the predictions on both unseen datasets display a trend towards a steady-state value, the CNN model's predictions on q_t = 36.14 W/cm^2 show some over-estimation. The estimated uncertainty of the heat fluxes was calculated to be 0.69 W/cm^2.

Figure 11 above shows the results of feeding the CNN model one image from each unseen dataset at a time, maintaining the images' temporal sequence. As more images are fed into the model, the predicted heat flux is calculated by taking the difference between the weighted average heat flux and the true heat flux. For both unseen datasets, as the CNN model is given more frames to predict the steady-state heat flux, the average prediction trends a single value. However, the CNN shows some over-estimation on the dataset with q_t = 36.14 W/cm^2, as the line trends towards a higher value. While both graphs show that the method used here is viable with a CNN model, the inaccuracy of the prediction trend shows some improvements could be made to the model.

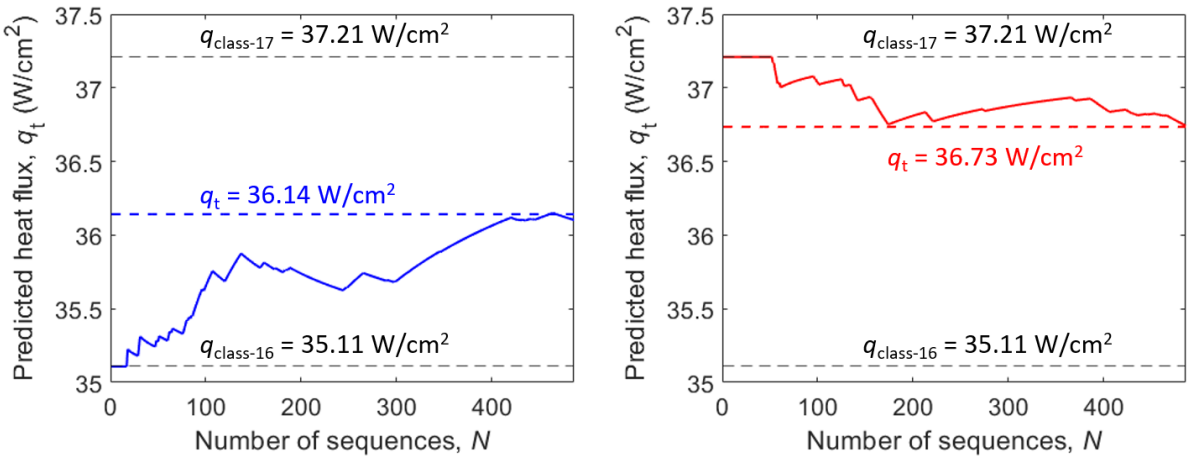


Figure 12: Plotting the predicted heat flux for the ConvLSTM model when given images from (a) q_t = 36.14 W/cm^2 and (b) q_t = 36.73 W/cm^2 versus the number of frames used to populate the testing dataset. On both unseen datasets, the ConvLSTM model's predictions trend towards the true steady-state heat flux value. The estimated uncertainty of the heat fluxes was calculated to be 0.69 W/cm^2.

For comparison, Figure 12 above shows the results after performing the same analysis with the ConvLSTM model. In contrast with the prediction line trend of the CNN model, the trend of the ConvLSTM model is much stronger towards the true heat flux values of the unseen datasets. Again, this can be attributed to the ConvLSTM model's LSTM cells preserving information between subsequent timesteps, allowing the model to appreciate changing features over time. This could, for example, capture trends in bubble dynamics, which could guide the model's predictions. Another possible explanation for the ConvLSTM's performance is the overall higher density of each unit of data. While the CNN model predicts using data from a single image, the ConvLSTM model uses a wide berth of information, from the entire sequence of image data to the relationships between features in subsequent images.

To complete the comparison between the two models, the relative error of the predictions for both models was plotted against the number of frames or sequences. Figure 13 shows the

results of the analysis. As expected, as the number of data points increases for both models, the relative error saturates, trending towards a constant value. However, due to the CNN's over-estimation on the unseen data, the relative error trends towards a non-zero value. In contrast, the ConvLSTM model's relative error on both datasets trends towards 0, confirming the higher performance of this model architecture for this situation in comparison to the CNN model architecture.

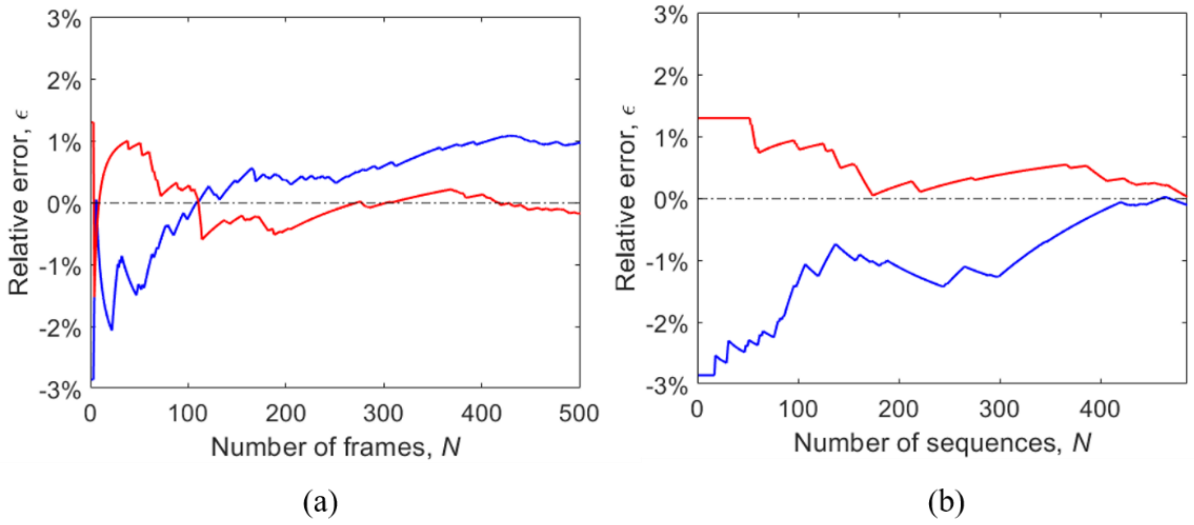


Figure 13: Plot of the relative prediction error versus the number of frames/sequences on the unseen datasets for (a) the CNN model and (b) the ConvLSTM model. Note the saturation of the relative error as the number of data points increases, and the effect of CNN's overestimation on the unseen dataset resulting in a non-zero relative error.

Due to the sequential and temporal nature of the ConvLSTM data, both the sequence length and temporal gap between images were deemed as important parameters to explore. The source videos were taken at 3000 frames per second, and sampled into a group of 5001 images per class, each representing a steady-state heat flux. In order to calculate the temporal length, we used the equation $t_{seq} = (N_{seq} - 1) \cdot dt$, where t_{seq} represents the total length in time of each

sequence, N_{seq} represents the number of frames in each sequence, and dt represents the actual time elapsed between consecutive images in a sequence. By downsampling the original image sets, we reduced the framerates to 1500, 600, and 300 frames per second, resulting in dt values of 0.67, 1.67, and 3.33 ms respectively.

Downsampling the image sets to lower framerates necessarily changes the amount of images used for training, validation, and testing, although the percentage splits remain the same. In order to compare the different parameters more fairly, the percentage splits were modified so that the same amount of training and validation data were given to each ConvLSTM model, with the testing dataset changing to adjust accordingly.

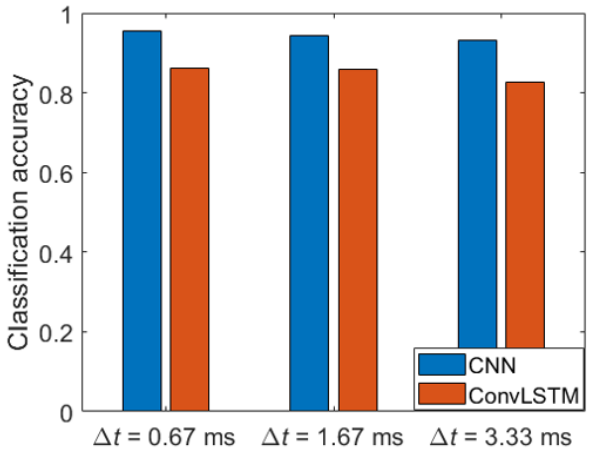


Figure 14: Comparison of CNN classification accuracy and ConvLSTM classification accuracy, with a sequence length of 1, while varying time between consecutive frames in a sequence.

In Figure 14 above, for all values of dt, the ConvLSTM model architecture shows slightly worse performance than the CNN model architecture when the sequence length is 1. This can be explained by examining the respective model architectures. While the ConvLSTM model balances convolutions and LSTM cells to preserve temporal information about image features,

the CNN model contains 3 convolutional layers, each performing several operations in sequence to further process individual images with great detail. The advantage of ConvLSTM is visible when the model is able to work with a sequence of images, resulting in a deeper understanding of dynamic features in the data. Another result from this portion of the study is that the sampling rate, dt, shows no large effect on the accuracy of the CNN models' predictions. This again follows the understanding that CNN models are not equipped to deal with any temporal or dynamic information in the data.

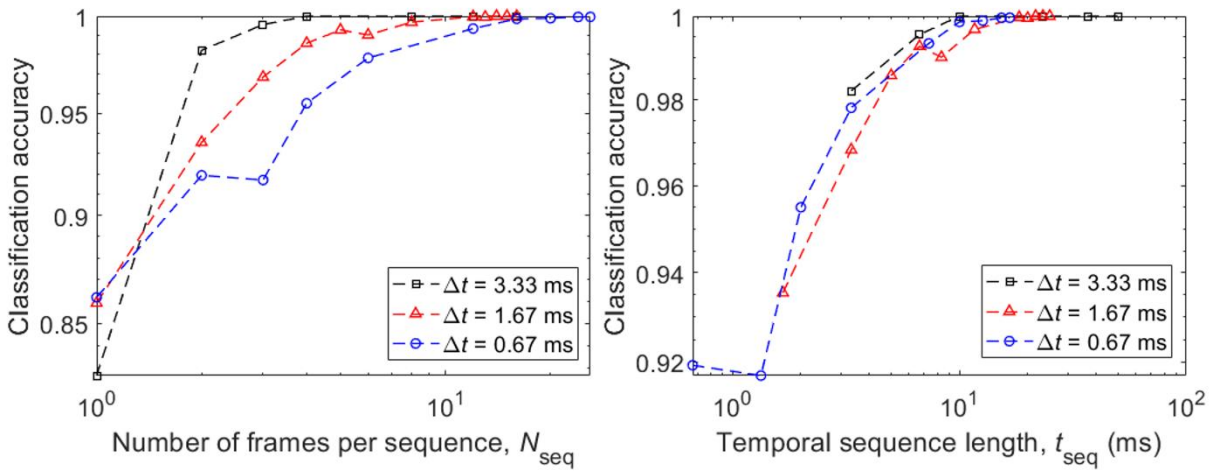


Figure 15: Plotting classification accuracy of ConvLSTM models versus (a) sequence length and (b) temporal lengths for different sampling rates.

In Figure 15 above, the results of the training of several ConvLSTM models on data at different sampling rates are shown. Varying both the sequence length in terms of number of frames and in terms of temporal length, we see accuracy is most dictated by the temporal sequence length, t_{seq}, as opposed to the number of frames per sequence, N_{seq}. As mentioned previously, the temporal length is what most determines the information that the ConvLSTM

model will have access to, as opposed to the length of the sequence. Also worth noting is that dt has no significant effect on the accuracy in comparison to the other parameters, as seen with how close the lines are in each plot. Generally speaking, accuracy saturates when t_{seq} is greater than 15 ms.

2.4.2: CNN and ConvLSTM for Boiling Crisis Detection – Generality & Interpretability

An important metric for any machine learning model is its generality. Similar to concerns of overfitting, measuring generality is a great way to ensure the usefulness and robustness of a trained model, by assessing its performance on datasets both similar and different to the dataset it was originally trained on. To perform this study, the CNN and ConvLSTM models were tasked with predicted on the testing dataset of DS1 and the full DS2, DS3 datasets. Then, confusion matrices were generated from these predictions.

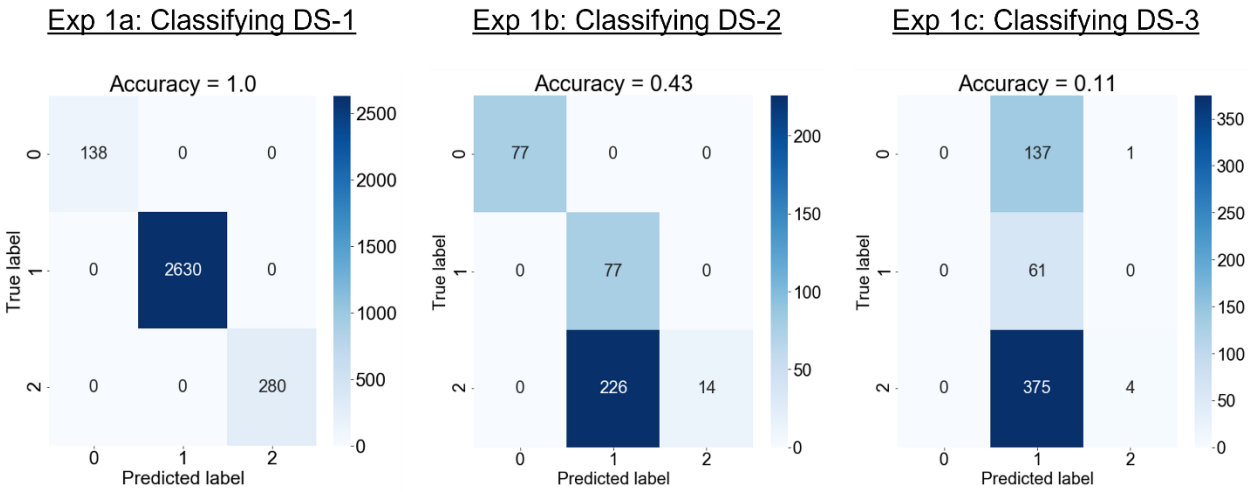


Figure 16: Confusion matrices of CNN testing on DS1, DS2, and DS3 to test model generality.

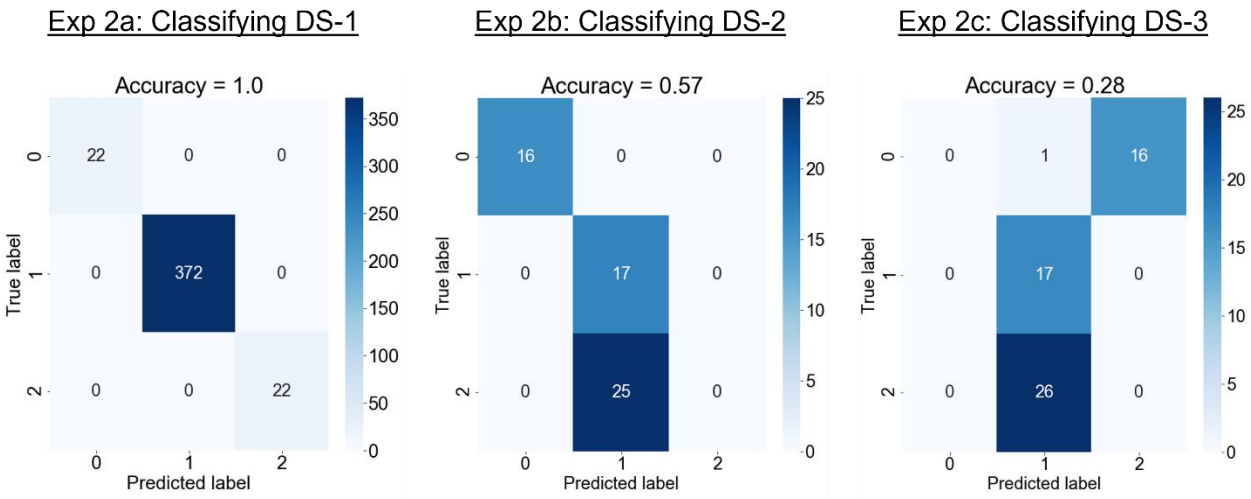


Figure 17: Confusion matrices of ConvLSTM testing on DS1, DS2, and DS3 to test model generality.

In Figures 16 and 17 above, it is clear that on the original dataset, DS1, both models perform the classification task with no issues. In other words, the models' interpolation abilities are strong. However, in terms of extrapolation i.e., classifying on completely unknown datasets, both models struggled, with very low classification accuracy. However, despite the mutual drop in accuracy, the ConvLSTM model still shows a higher accuracy compared to the CNN model. Figure 18 below compares the accuracies more directly, to highlight this difference. With this in mind, it can be said that the ConvLSTM model shows stronger extrapolation abilities in comparison to the CNN model.

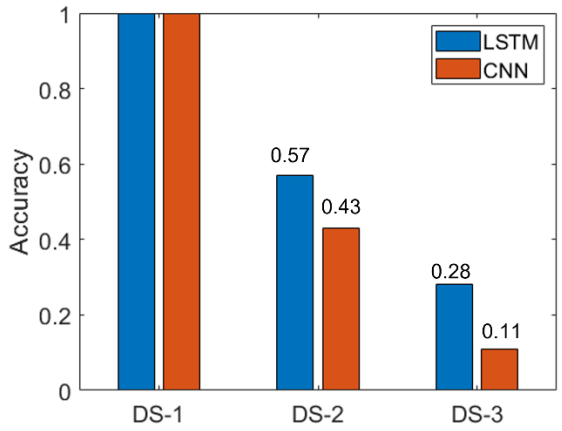


Figure 18: Testing accuracy comparison between CNN and ConvLSTM (labeled as "LSTM") models. Note the higher extrapolation accuracies of the ConvLSTM model on DS2 and DS3.

The final avenue of study with the CNN classification model specifically was an examination of the interpretability of the results. By visualizing the attention maps of the CNN model during prediction, we hoped to obtain new insights into the primary mechanisms causing the CHF condition. Although physical models do exist and can be used to observe and identify such mechanisms, machine learning models operate as a black box, meaning that by interpreting its attention during prediction, one could potentially find completely new results or confirm current models with a completely independent model. Below in Figure 19, we see two attention maps generated by the CNN classification model. The bright red areas represent the image features that were most important for the model to predict its class. From these two examples, we see particular attention given to the vapor columns near the surface, along with a lack of major focus on the macrobubble. This indicates that hydrodynamic instability dominates the CHF condition in this particular instance, where water is boiled on a flat copper surface.

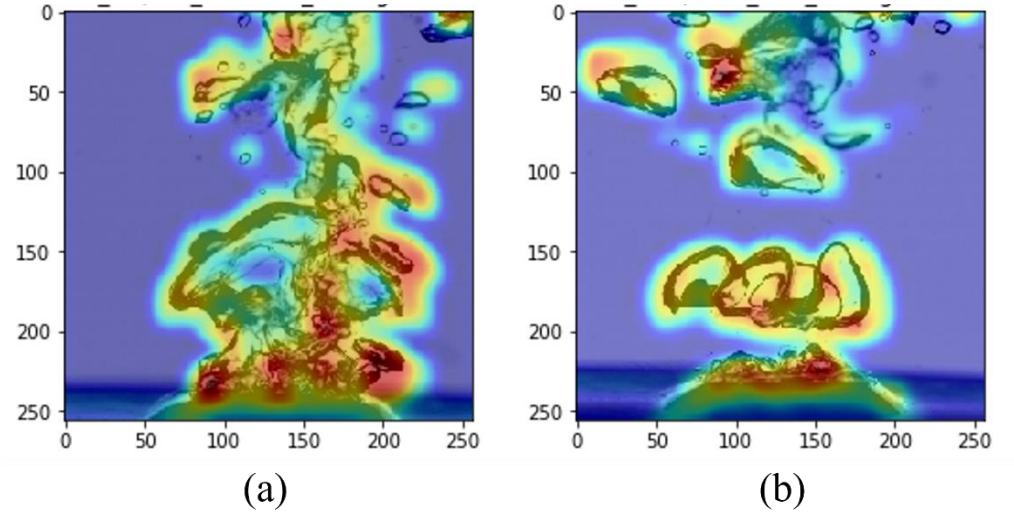


Figure 19: Two attention maps generated by the CNN model when classifying on an image from (a) the nucleate boiling regime and (b) the film boiling regime.

2.5: Conclusion

Two neural networks, a CNN and a ConvLSTM model, were trained to classify images and videos taken from high-speed video of HFE-7100 boiling on silicon micropillar arrays at various steady-state heat fluxes. As opposed to traditional boiling image analysis, focusing on static images and dominant features unique to each image, the analysis here aimed to capture the dynamic behaviors of bubbles between subsequent images by implementing the ConvLSTM model. Each class that the models were tasked with identifying corresponded to a single heat flux value, allowing the models to effectively quantify the steady-state heat flux in a given image or video. Both the CNN and ConvLSTM model predicted the heat flux of the testing dataset with accuracy greater than 90%. Both models can also interpolate heat fluxes on unseen images and videos representing heat flux values between the trained values. Comparing both model's performance on testing data from already-seen classes and unseen classes, the ConvLSTM model predicts with a higher accuracy due to the LSTM portion of the model being able to recognize

relationships between image features over time. Another factor in the ConvLSTM model's performance is its higher information density per unit of data, due to the rolling sampling.

By varying sequence length, framerate, and temporal sequence length, a deeper understanding of ConvLTSM's advantages was obtained. Although dt had no strong effect on the accuracy of ConvLSTM models' predictions, the temporal sequence length was found to have the most impact on model prediction accuracy. This points to the intuitive advantage of ConvLSTM models, in that the model architecture is capable of capturing dyanamic features and changing information to make stronger, more accurate predictions.

Both models were examined for their generality as well. By having the CNN and ConvLSTM models classify on DS1, DS2, and DS3, their ability to interpolate and extrapolate on new data was tested. Both models showed strong interpolation abilities, but the ConvLSTM model showed higher performance during extrapolation.

To cap off the work in this chapter, the CNN model was tasked to predict the class of testing images. The results were then visualized using an attention map to highlight the image features deemed most important by the model to predict its class. Interpreting the results, we concluded that for water boiling on a flat copper surface, the model's focus on the vapor columns, along with its lack of focus on the macrobubble, hydrodynamic instability is the primary mechanism for CHF.

Chapter 3: Transient Signal Prediction Using LSTM and CNN Regression

3.1: Background and Motivation

Machine learning has shown immense promise in data analysis. Much work has already been done implementing machine learning models of all forms in performing analysis such as stock market prediction [23]. A review performed by Yoo, et. al summarized several machine learning methods of big data analysis, just in the field of medicine [40]. With a large body of established work already in place, the next natural step was to apply such knowledge to our work.

With the process described in chapter 2, both CNN and ConvLSTM models were able to estimate steady-state heat fluxes using only images. However, this process involved a large amount of post-analysis, and per the suggestion of a collaborator, a new direction was settled on: regression. As opposed to classification problems, regression problems involve the prediction or designation of a specific value. For example, instead of classifying an image as representing the class of images representing a temperature of 97 degrees Celsius, a regression-based model would instead use the image as input and then output an estimated value, say, 96.5 degrees Celsius. With regards to pool boiling applications, the benefits are clear. While classification-based quantification generally is simpler to implement, regression is the most natural fit for any problem that aims to quantify values.

However, a good portion of machine learning regression work focuses on numerical estimations. The work described in this chapter instead aims to take in the same boiling images as input and give a heat flux estimation as output. This requires a different training data pipeline, with a different model architecture to be able to not only capture key features in each image but

to be able to associate these key features with numerical values. This chapter will focus on such analysis, implementing regression for more accurate predictions of transient data.

3.2: Literature Review

In "An Introduction to Regression Analysis", Sykes defines regression analysis as "a statistical tool for the investigation of relationships between variables" [41]. The goal of regression analysis is to model (and oftentimes simplify) the relationship between input variables and output variables which originally have complex, unknown relationships. An extremely basic example of this is the creation of a line of best fit for a scatter plot. By minimizing the error between the line of best fit and the data, one can claim that the line approximates the data and allows for general predictions outside of the original input domain.

While originally a method for statistical analysis and traditional modeling, with the rise of machine learning, researchers have begun to assign machine learning models the same task of correlating inputs and outputs with minimal error. In fact, machine learning inherently lends itself to regression problems, due to its "black box" nature. While the weights and connections formed within a model during training can remain unknown, the model will still predict and simulate functions with high degrees of accuracy. In papers dating back to the early 2000s, researchers implemented machine learning to regression problems. In Segal's paper addressing random forest regression with machine learning [42], the author cites Breiman's work in 2001, where an "ensemble classification and regression approach" was developed [43]. With this in mind, Segal revisits and refines the method for the same random forest problem, further exemplifying the power that machine learning has in regression problems. In another paper, Kapelner and Bleich further optimize the implementation of machine learning applied to specific regression problems, showing that the theoretical side of the field is still ever-evolving [44].

More recent work shows that machine learning can be applied to real-life analytic problems. For example, Goldstein et. al tackle the problem of cardiovascular risk prediction by training a machine learning model on 13 “regularly measured laboratory markers” to predict mortality after specific diagnoses [45]. Here, the authors specifically cite the black box approach as a major boon in solving regression problems that traditional methods cannot. Another example of applied regression analysis using machine learning is found in a paper by Shah et. al. Here, several complex models for water quality are developed using various methods and are compared for performance [46]. The authors cite rising water pollution and the limitations of current models as motivation for this work. Using the same 30-year dataset of water quality of the upper Indus River basin, 4 models were developed using linear and non-linear regression methods, artificial neural networks (ANN), and gene expression programming (GEP). Comparing the performance of the 4 models, it was found that the GEP model outperformed all other models, with the results of the GEP model sent to authorities to help with decision-making with regards to water quality. Narrowing the scope of review further to pool boiling, researchers have applied regression analysis to analyze data, both with traditional numerical analysis and with machine learning. For example, Sarangi et. al use a quadratic polynomial fit to create a fit between the coating characteristics of sintered copper particles of differing morphologies and their respective boiling performance [47]. In this application, traditional regression analysis was sufficient in modeling the relationships, allowing the authors to also determine the most important variables with regards to the boiling performance. In contrast to traditional regression analysis methods, Alic et. al used machine learning (or as they write, computational intelligence) methods to estimate heat flux at pool boiling processes [48]. Citing the difficulty in manually processing and analyzing large amounts of data, the authors turn to machine learning algorithms

to optimize the process. They estimated pool boiling heat flux “in the isolated bubble regime” using 3 machine learning algorithms: decision tree, ANN, and support vector machine. Similar to the cardiovascular work cited above, the authors here aimed at a black-box approach to analyze the data.

While the most classic CNN case studies involve some form of image classification, researchers have long been developing CNN models for the purpose of regression analysis. This is an interesting avenue of research, considering that the goal is to extract or extrapolate data, typically numerical data, from image data. Wang et. al published a paper documenting their work in developing a regression model to transfer medical images to continuous clinical variables [49]. Although a CNN is not used in this specific case, it does highlight the potential use of feature extraction and specifically designed machine learning models to move from image data to continuous values. Niu et. al specifically applied multiple output CNN to apply regression for age estimation [50]. Here, it is specifically shown that the image processing capabilities of CNN models can be used to output, not just a classified value, but rather a continuous, specific value. The researchers achieve this by breaking down the regression into a “series of binary classification sub-problems”.

Another large body of work related to CNN regression is that of 2D and 3D reconstruction and registration. In theory, the regression capabilities of models are not limited to just value extraction but can even be used for object identification and extrapolation, as found in a myriad of papers. For example, Cao et. al developed a CNN regression model to map input images to a corresponding deformation field [51]. The authors found the model’s performance to be “promising”, even across different datasets, although they note that the performance varied

highly with new data. A similar study, dealing with biomedical image segmentation, was performed by Meng et. al [52]. Combining CNN, Attention Refinement Module (ARM), and Graph Convolution Network (GCN) architectures, the authors prepared a machine learning model tasked with identifying and locating the instance boundary in medical images. The ARM and GCN portions of the model aided in identifying the local boundaries, while the CNN extracted "semantic information from the input image". Much like boundary identification, object identification in images is a common task assigned to CNN models. Nakahara et. al developed such a model, using a CNN with "Parallel Support Vector Regression" to find and classify objects in images [53]. The CNN portion of the model was tasked with classifying the objects with boundaries defined by the support vector regression portion of the model. The researchers found the developed model suitable for embedded vision systems. Yuan et. al noted that classification-based models, while common and powerful in their own right, have limitations including a required uniform input size and classifier design, which introduces human intervention in the learning process. As such, they converted a CNN classifier into a regression model by replacing the fully connected layer with continuous feature maps [54]. In their paper, they lay out a general procedure for implementing CNN regression models for image segmentation and object detection.

Even more complex of a task than boundary searching and object detection is 3D reconstruction and registration. Aside from the added spatial dimension, oftentimes the model is tasked not only with finding objects in space using images, but also with maintaining that object in a theoretical space or extending the object and its movements. Considering how complex of a task this is, with many layers of abstraction, it is not without its limitations. Sattler et. al write about such limitations in their paper, taking camera pose regression and estimation as a specific

application [55]. While the newer pose regression methods do function, the authors found through several benchmarks that they needed more development before outperforming standard 3D structure-based methods. Chinaev et. al proposed MobileFace, a CNN regression-based model for facial reconstruction on mobile devices [56]. To prepare the model, the researchers first took slower, more established face models with the goal of preparing training data for MobileNet-based CNN's tasked with shape regression. On 3 different datasets, the new model maintained state-of-the-art accuracy while also showing improvements in speed and size of the model. Similarly, Jackson et. al prepared a CNN using both 2D images and 3D facial scans [57]. The resulting model requires only a 2D image to reconstruct the geometry of the face in 3D space. While the CNN architecture is kept simple, the model is then tasked with performing a regression on the volumetric representation of the geometry. The authors also note the current limitations of similar technology, where the methods require "complex and inefficient pipelines for model building and fitting". Work has also been done to not only reconstruct facial geometries, but also entire poses of humans in images. For example, Kolotouros et. al prepare a machine learning model to estimate 3D poses using single 2D images. The authors build off of a previous approach, where a parametric model of the body, called SMPL, is given to a model to regress the model parameters to match the pose as closely as possible with a given image. By introducing regression much earlier in the model, the authors are able to instead "directly regress the 3D location of the mesh vertices". The authors claim that this approach outperforms comparable methods. Mahendran et. al performed a similar study, regressing poses in a continuous 3D space using CNN models [58]. This contrasts with most current approaches, where poses are instead classified. A related avenue of research is 2D/3D registration. An example of such work is proposed by Miao et. al, wherein a CNN regression approach is used for

real-time registration [59]. The goal of registration is to bring "pre-operative 3-D data and intra-operative 2-D data into the same coordinate system," with the goal of facilitating diagnosis and analysis. The ability of CNN regression models to identify objects in both 2D and 3D space makes them a powerful tool for this specific application. Compared to traditional intensity-based methods, the authors found the CNN regression method just as accurate with an advantage in computational efficiency.

3.3: Methodology

There are several algorithms/model architectures explored in this chapter, using a variety of datasets. For the data-driven nowcasting/forecasting, models were trained and tested using vapor fraction data related to boiling setups, thermodynamic data provided from Kingston Group, and also hydrology data from NASA's Earthdata. Because this work focused on simple numerical data, the input data required little to no preprocessing, only requiring that the data be either in an Excel spreadsheet or text file, such that it could be read using Panda's "read_csv" function.

For the CNN-LSTM/Multi-Modal work, a combination of image and numerical data was used to train and test the associated models. The main data used in this study were provided by Dr. Todd Kingston at Iowa State University, such as the pressure drop sequences and boiling images. The pressure data were sampled at a frequency of 2,500 Hz, while the frame rate of the source videos was 30,000 fps. With this in mind, the videos were down sampled to 2,500 fps to line up the images with corresponding data points. Each image was resized to a resolution of 24 by 128 to keep training times low while maintaining prominent visual features in each image.

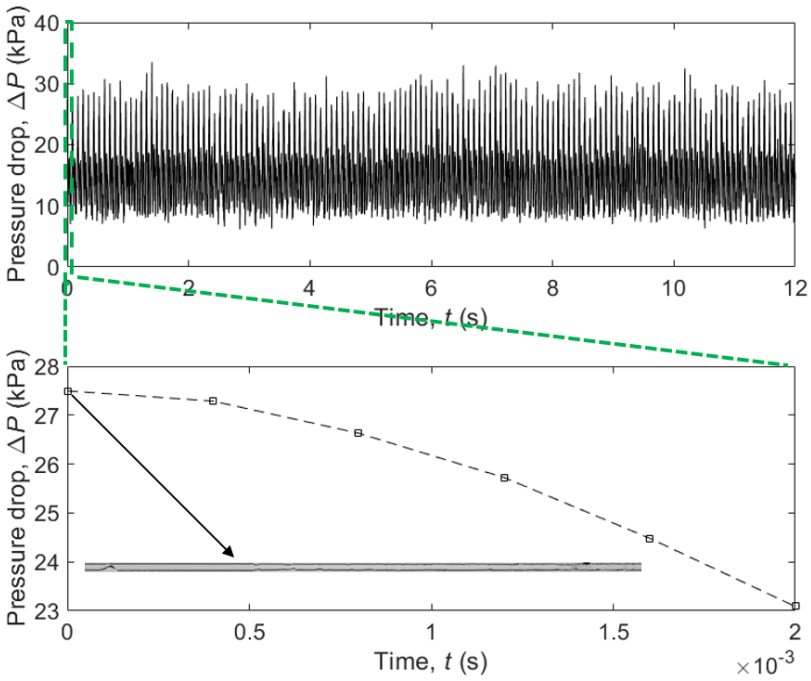


Figure 20: Close-up view of Kingston pressure drop data, and an example of an associated image for a single pressure drop data point.

In both model architectures, the LSTM portion handles the numerical data with vectors, as opposed to single values. What this means is that given a defined number of values as input, the LSTM model will predict the desired number of values into the future. As such, once the data has been read from the source file, it is arranged into vectors in the code. This process will be discussed in more detail in the discussion of the code structure.

3.3.1: Algorithms and Model Architecture

Before discussing any algorithms relevant to each model (that previously have not been discussed), it will be helpful to first cover the overall structure of the models, as the models in this chapter combine several smaller models. For both nowcasting/forecasting and multi-modal analysis, an LSTM model is the final link between input and output. For the multi-modal work, a

CNN regression model is first linked to the original input data, then connected to the LSTM model to perform the forecasting. Below in Figure 21 is a schematic showing the connection between input, CNN regression model, LSTM model, and output.

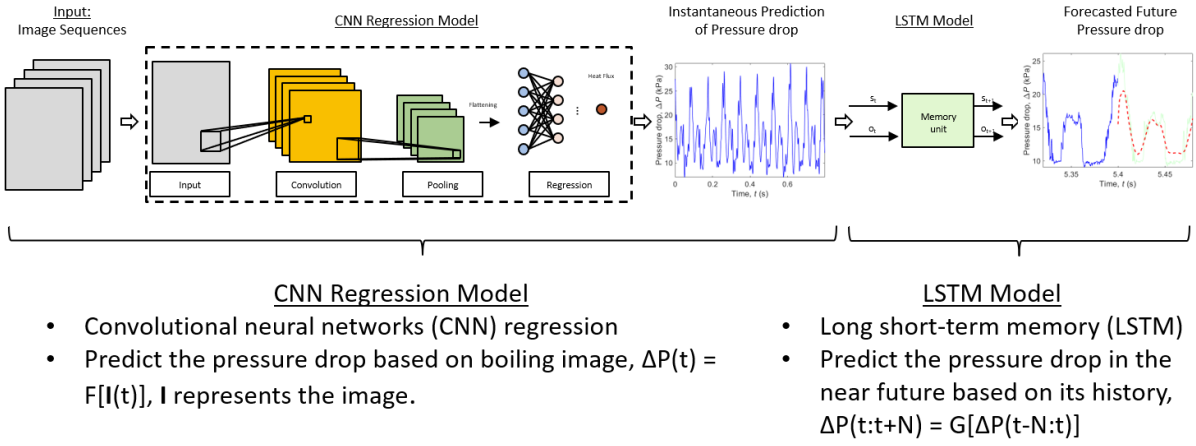


Figure 21: Schematic of the combined CNN regression and LSTM models, connecting boiling images to forecasted future pressure drop data.

For the most part, the layers in the CNN regression model architecture have already been explained in the previous chapter. However, a key difference, one that will be expanded upon in the following section, is that the final dense layer uses a linear activation, as opposed to a Softmax activation. Instead of outputting a probability distribution for the most likely class, a dense layer with linear activation will simply output a value. In its most simple form, a linear activation can be understood as a function of the form f(x) = x. Changing the activation of the dense layer is what will convert a CNN classification model to a regression model and is what drives the work described in this chapter. Another layer not found in other model architectures in this thesis is the "BatchNormalization" layer. This layer serves a similar purpose to scaling the numerical data to a 0 to 1 scale. By normalizing the data in each batch, the training is both faster and more stable.

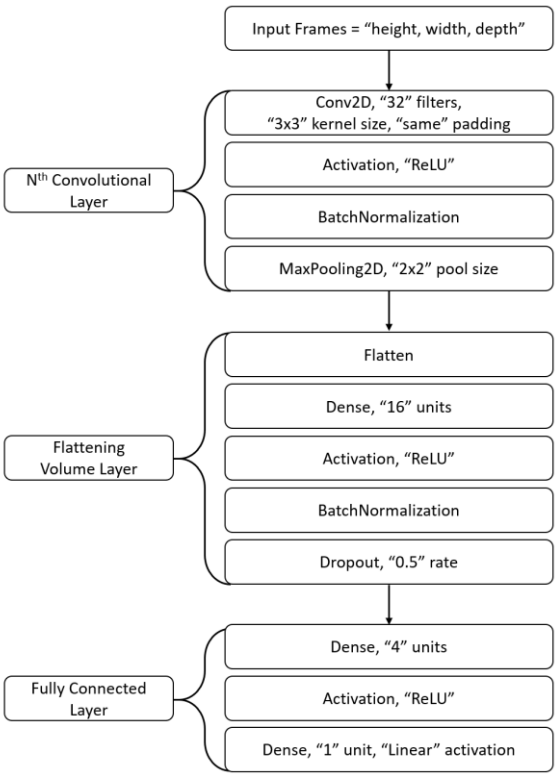


Figure 22: CNN regression model architecture with parameters.

Many of the layers used in the CNN regression model are similar, if not identical, to the CNN model found in chapter 2. There are, however, minor differences, such as the ReLU activation being relegated to its own layer as opposed to as a parameter within the convolutional layer. The first convolutional layer stack is repeated for however many filters are defined in the CNN regression code before the data is flattened, sent through another dense layer, and finally

sent to the final dense layer for regression analysis. The flattening volume layer is necessary to convert the image data into linear data that the model can then predict a value from.

With the instantaneous outputs from the CNN regression model, vectors can be constructed to be used as inputs for an LSTM model. The LSTM portion of the CNN-LSTM model is relatively simple, consisting of 2 stacked LSTM layers and a dense layer with linear activation. Despite its simplicity, the LSTM architecture is what enables the forecasting and prediction of any number of subsequent future timesteps. In a simple parametric study, a single LSTM layer was found to show some instabilities during training, while 3 stacked LSTM layers increased training time with no noticeable benefits. As such, it was decided that 2 stacked LSTM layers were sufficient. Note that many of the parameters in the LSTM model are dependent on other parameters. The vector sizes and the number of variables, translated to the number of features, all change the details of the LSTM model, although the overall architecture remains the same.

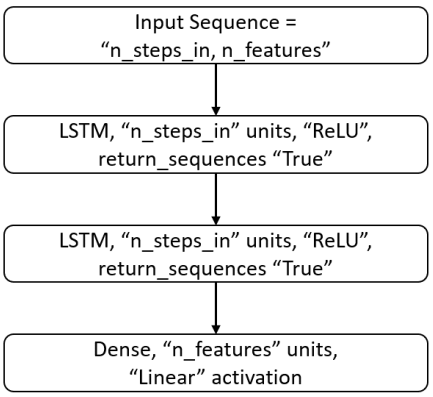


Figure 23: LSTM model architecture with parameters

3.3.2: Code Structure

First, we will discuss the structure of the LSTM forecasting code, as it is also found in the multi-modal analysis code later on. The list of imported packages is similar to the CNN and ConvLSTM codes in this chapter, with Keras, Numpy, and Matplotlib being key software packages in preparing the models and analyzing the results. One key function for the LSTM codes is scikit learn's "MinMaxScaler". The purpose of this function is to scale a given input

array of data to any desired range. All data were scaled from 0 to 1 in order to handle different units and scales during training. For each unique variable, the corresponding scaler was saved in order to undo the scaling during post-analysis to assess the model's performance more accurately.

To prepare the vectors used for the training, validation, and testing data, a new function, "split_sequence", was created. Given any 1-dimensional array of values, the desired number of steps in, and desired number of steps out, the function will output two arrays, labeled X and Y. Array X contains each input vector, while array Y contains each output vector. For any timestep, one can visualize the input vector as the values leading up to the current timestep and the output vector as the values following the current timestep. By preparing the data in this manner, an LSTM model can be trained to take a sequence of values as input, then output the most "correct" sequence of values.

For single-variable cases, such as vapor fraction data, pressure drop data with CNN regression, and singled out variables in multivariate cases, only one array is needed to contain all the training, testing, and validation data. However, once multiple variables are to be analyzed, the dataset needs to be processed further before training and testing the model. Whereas with single variable cases, each vector consists of a sequence of single values, multivariate datasets consist of vectors, where each vector is a sequence of multi-dimensional arrays with length 1. In other words, each entry in a given vector consists of a horizontal stack of data, with each column in the stack corresponding to a specific variable. This was achieved by first reading each variable's data to respective arrays, then horizontally stacking each data point one entry at a

time. Once the data was compiled in this manner, the trained LSTM would require the same dimension arrays for input and output data.

With the data stacked properly and scaled according to each variable, the "split_sequence" function was called to vectorize the data. Then, the LSTM model architecture was defined (remember to go through that architecture in the previous section). As the final layer was a dense layer, it was important to ensure that the number of units corresponded to the number of variables/features. This was handled automatically by defining the number of features based on the dimensions of the dataset. Then, the model was trained on the data and saved for testing.

The testing of the LSTM models involved a similar I/O to that of the training process. Data was loaded using "read_csv" and scaled according to each variable. Then, the data was vectorized and saved to these 2 arrays, one for input vectors and one for output vectors. Finally, the trained model was loaded and the "predict" function was called in order to obtain a prediction output vector. 2 different methods were used to analyze given prediction vectors. The simplest form of analysis involved simply comparing a single prediction vector with the true output vector. This method is useful in assessing the model's performance at a smaller scale. Before calculating error, one could qualitatively compare the prediction vector to the output vector. If there is a strong pattern in the output vector, does the prediction vector follow suit? However, in order to visualize the overall performance of the model across the entire dataset, a different method of analysis was used.

Instead of looking at a single prediction vector, for this method, the last point of each prediction vector was appended to an array. Then, the stitched final predicted points could be

compared to the original dataset, providing an overall, albeit imperfect, assessment of the model's overall performance. The decision to use the final point of each prediction vector was guided by the thinking that for any prediction vector, the error will most likely increase as one strays further from the original starting point. As such, the stitched final points provide the "worst-case scenario" for the model's predictions, giving us the most honest view of the overall performance. For both of these methods, the error was calculated using mean absolute percentage error, and the input, output, and prediction vectors were plotted altogether. Below is an example of such a plot.

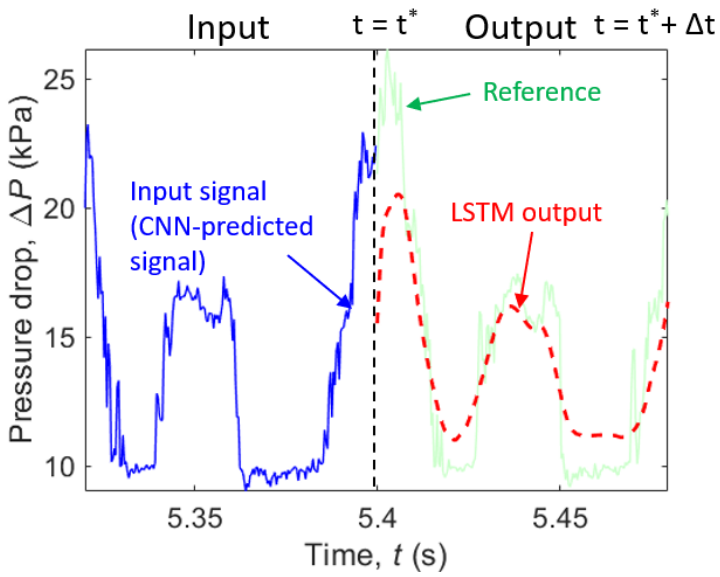


Figure 24: LSTM prediction of pressure drop during two-phase cooling, demonstrating the comparison between predicted vector and true output vector.

The CNN regression code uses a different overall code structure, in that it calls to several files referenced in the "__init__.py" file for functions, to keep the main code uncluttered. The first step to preparing the data for the model was to create a text file containing the associated

heat flux values for each image. Since all the images were sorted in advance, the creation of the text file involved simply adding the consecutive values in a loop. Then, the function "load_frame_attributes" was called from the file "datasets.py". This function serves to simply read the text file containing the associated values that enable the regression analysis. After converting the array to a Pandas dataframe object, allowing for reference of data using variable keys, the image directories are defined and used in the "load_frame_images" function, also from the "datasets.py" file. The purpose of this function is to loop over each image file and load the image data to an array, resizing each image to the desired dimensions. The array of image data was scaled down by a factor of 255 to suit the regression analysis, which requires values to be on a 0 to 1 scale. Training and testing data were then defined using "train_test_split", with 20% of the data going to testing. Later during training, the validation split was defined, leading to a 64/20/16 split of training/testing/validation. The image data was defined as the input data and the associated flux values as the output data. The flux values were also scaled for the same reasons as the image data.

Next, the callbacks for training were defined. Much like the CNN and ConvLSTM codes in chapter 2, ModelCheckpoint was used to monitor the loss, and only save the model weights that yielded the minimum validation loss value, saving space and ensuring that strange end behavior during training did not invalidate the entire training process. The model architecture was defined using "create_cnn" from "models.py". This function takes in the desired image width and height, along with the RGB depth and filter size. The final parameter for the function is a Boolean value, determining whether to compile a CNN model for regression or classification. For this code, the regression option was chosen, and the image height and width were set to 64. Based on the number of convolution filters defined in the function call,

“create_cnn” will then create the same number of convolutional sections in the model architecture, with each section consisting of Conv2D, Activation, BatchNormalization, and MaxPooling2D layers. More details on the layers and their parameters can be found in the previous section dedicated to model architecture. In order to prepare the model architecture for regression, an extra dense layer is added at the end of the model with a linear activation.

With the model layers defined with a final dense layer with linear activation, the code then compiled the model with Adam as the optimizer, with a learning rate of 1E-3 and decay set to 1E-3/200, and loss defined using mean absolute percentage error. The model was then trained using the input images and output flux values for X epochs with a batch size of 8. Finally, the model was evaluated on the testing set. The evaluation of the model began with predicting on the testing dataset. Then, the predictions were scaled up with the original max flux value to match the scale of the original heat flux values. With the fully scaled predictions and the original heat flux values prepared, the performance could then be evaluated using statistical methods.

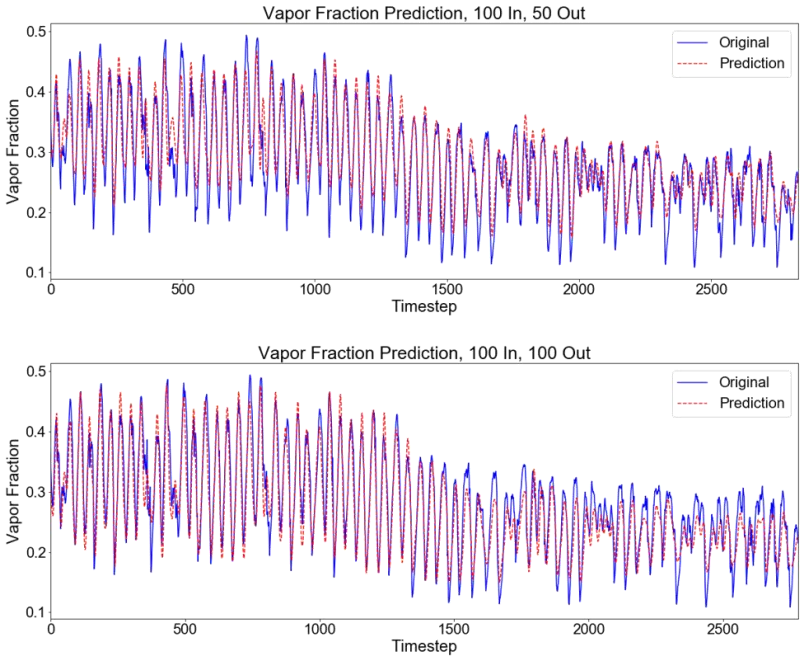
3.4: Results and Discussion

The results in this section will be divided into 2 sections, as the LSTM nowcasting/forecasting work was preliminary and separate from the CNN regression work that followed.

3.4.1: Nowcasting and Forecasting of Data using LSTM

One of the main parameters in the LSTM code is the length of the vector sizes. The code structure allows for the adjustment of both input and output vector lengths. With this in mind, it was deemed important to first establish the effect of vector lengths on model performance. Below are two examples of stitched predictions overlaid with the original data. The upper figure

represents an input vector size of 100 and an output vector size of 50, while the lower figure represents an equal input and output vector size of 100.



Predicting 50 data points using 100 data points

Predicting 100 data points using 100 data points

Figure 25: Two examples of vapor fraction LSTM predictions overlaid the original dataset with varying input and output vector sizes. The top figure shows predictions with vector sizes 100 in, 50 out, and the bottom figure shows predictions with vector sizes 100 in, 100 out.

Looking at the figure above, it is hard to make a quantitative assessment of the effect of vector size on the LSTM model's performance. With this in mind, a larger parametric study was performed by plotting the square root of root mean square error (RMSE) versus the vector period size. For various values of N, LSTM models were trained on the vapor fraction data with input vector sizes of form N in, N out, and 2N in, N out. Below are the results of the parametric study plotted on a graph.

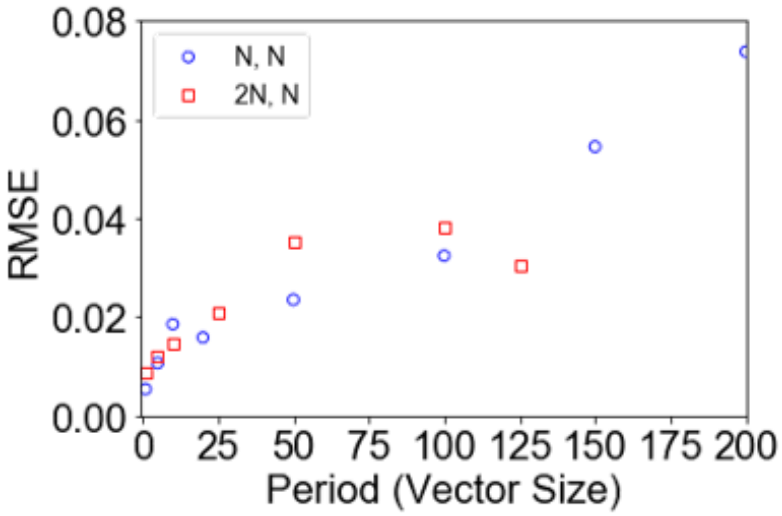


Figure 26: Square root of RMSE plotted against vector size N, comparing models trained with vector size N in, N out and 2N in, N out.

In Figure 26 above, the RMSE increases with increasing period size. This is expected behavior, considering that a larger prediction window means that there is a higher likelihood of error propagation. However, more interesting to note is that despite doubling the input vector size, no appreciable improvement is observed in the RMSE. This is somewhat counterintuitive, considering that by doubling the input vector size, the model is given twice the data to train on and extrapolate from. This is most likely a result of diminishing returns, although a much denser parametric study would have to be performed in order to fully understand this behavior.

3.4.2: CNN Regression and Multimodal Analysis

Figure 27 below contains a full view of the true experimental signal data overlaid with the instantaneous predictions from the CNN regression model and the predictions from the

combined CNN regression to the LSTM model. On a qualitative level, it is clear that both models accurately capture the periodic nature of the pressure drop data. Calculating the mean absolute percentage error over a randomly chosen 200 timestep range, the error for the instantaneous model is 9.12%, while the error for the CNN-LSTM model is 13.99%. This can be explained by understanding that the CNN regression model's predictions are done one at a time for each timestep, meaning that there is less possibility for error propagation over time. In contrast, the CNN-LSTM model predicts an entire set of points at once, taking into account the relationships between data points learned in the LSTM portion of the model. As such, there is more possibility for error propagation, which is reflected in the higher error value. However, the benefits of being able to predict several timesteps into the future cannot be exaggerated, as it would allow for the prediction of unfavorable behaviors and improve control systems.

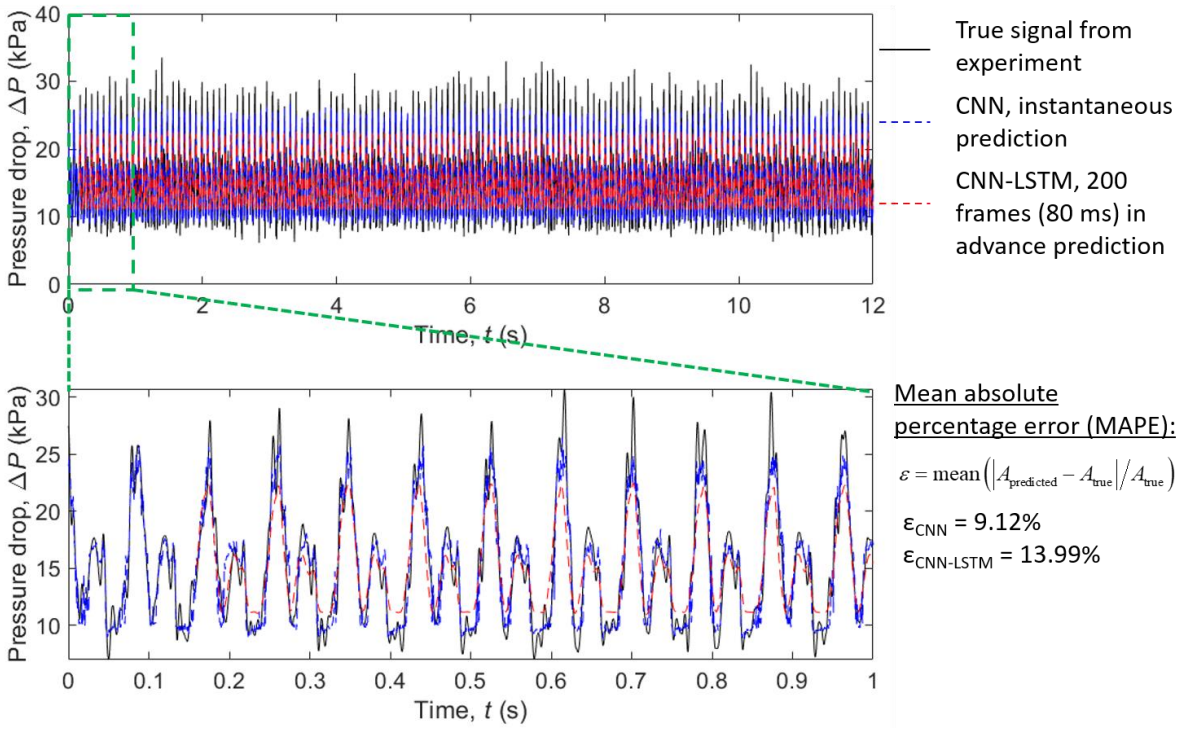


Figure 27: Zoomed-in view of comparison between true signal, CNN regression model's instantaneous prediction, and CNN-LSTM model's prediction 200 frames in advance.

3.5: Conclusion

Initial development of code to train LSTM models on vectorized data was successful. A simple parametric study of vector sizes showed that increasing vector sizes had an adverse effect on error, as expected. However, doubling the input vector size in relation to the output vector size, while increasing the amount of information used per prediction, had no appreciable effect on prediction error, most likely due to diminishing returns. Due to equivalent performance and shorter training times, equal vectors in and out were found to be most efficient.

Then, a CNN regression model was developed to predict heat flux values directly from boiling images. While the data preparation was more involved than in the work in chapter 2, because the model directly predicted heat flux values, we were able to eliminate the need for extensive analysis post-training to understand the model's predicted values.

Finally, with both codes developed, the CNN and LSTM models were combined, wherein the CNN regression model's instantaneous predictions were used to build vectors with which the LSTM model could predict future sequences. Comparing the relative error of the CNN-LSTM model to that of the CNN model's instantaneous predictions, there was a higher error observed in the combined model. However, the combined model grants the ability to use only boiling images to predict future sequences of desired variables, such as pressure drop. This capability shows much promise and can be worked on in the future.

Chapter 4: Summary and Future Work

To summarize the work in chapter 2, CNN and ConvLSTM models were developed, trained, and tested on boiling images, with each image corresponding to a specific steady-state heat flux value. After being given images from unseen heat flux values, the models' predicted classes were averaged to determine the predicted heat flux value. The ConvLSTM model outperformed the CNN model in both seen and unseen data, showing promise in interpolation by doing so. The ConvLSTM model's performance can be attributed to its LSTM cells, which allow the model to track changing image features and information over time. Another possible factor in the ConvLSTM model's higher accuracy is the higher density per unit of data, as the model predicts on sequences of images, as opposed to single images.

In the work described in chapter 3, a CNN regression model and stacked LSTM model were combined and integrated in order to forecast thermal properties using visual data i.e., boiling images. To achieve this, code had to be developed to train LSTM models to work with long sequences of data, also known as vectors. It was vital that the LSTM model would be able to accept sequences of data as input and predict an entire sequence of data as output. Then, code was developed to implement CNN regression on boiling images. By training the CNN regression model with boiling images and corresponding pressure drop, the model was then able to predict a single instantaneous heat flux value. By combining the instantaneous predictions, we were able to then provide the sequences of data needed as input for the LSTM model, thus integrating both model architectures into one pipeline.

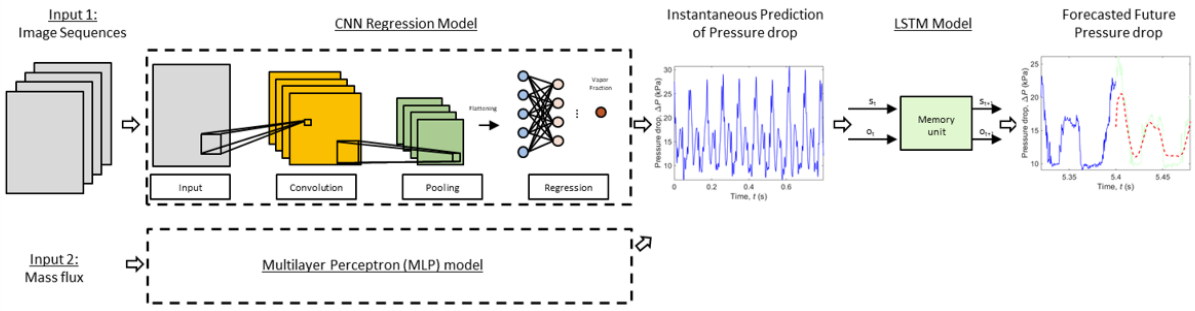


Figure 28: Schematic of multimodal fusion network, connecting CNN regression and MLP model predictions as inputs into a single LSTM model for forecasting of data.

Future work will mostly focus on building off the work in chapter 3. One potential avenue of work is multimodal fusion networks. Much like how a CNN regression model and LSTM model were interfaced together, this work would involve the combination of a CNN regression model, multilayer perceptron (MLP) model, and LSTM model. In so doing, we could supplement the input image data with extra numerical data to provide the LSTM model with more information to make more robust predictions. Figure 28 above shows how the models would connect.

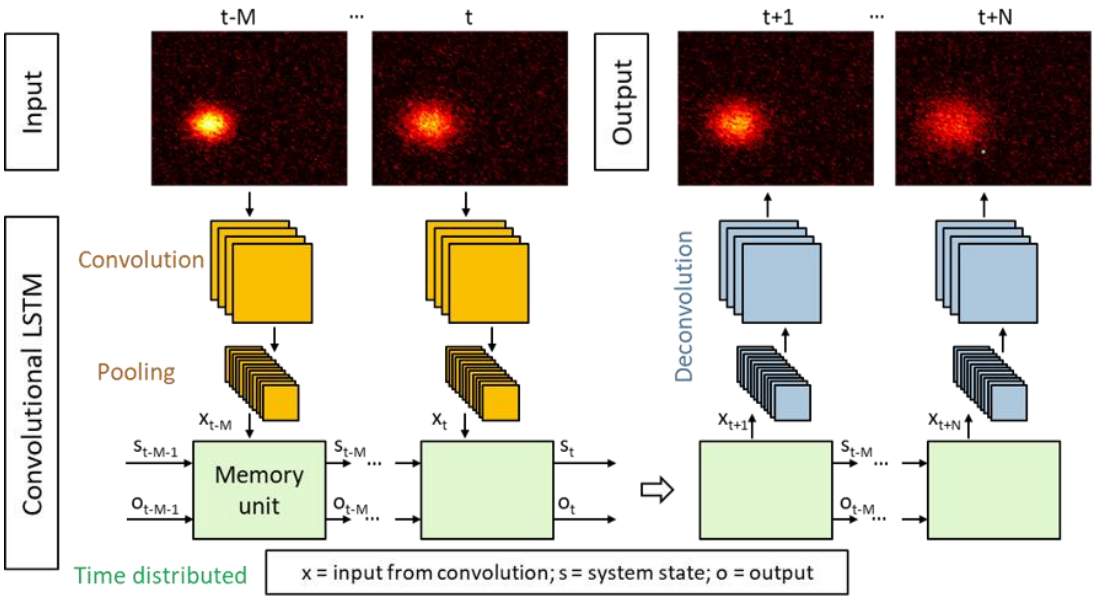


Figure 29: Schematic of high-dimensional sequence prediction, wherein 2D heat flux maps are input into ConvLSTM models to predict the future state of the heat flux map.

Another promising direction is high-dimensional sequence prediction. Figure 29 shows the basic pipeline of data that this work would require. Much like the nowcasting/forecasting work with LSTM, the goal of this work would be to expand the dimensionality of the forecasting, allowing for 2-dimensional heat flux prediction using heat flux maps, as opposed to 1-dimensional data. This would require a combination of the convolutional capabilities of a ConvLSTM model and the forecasting capabilities of the LSTM code.

References

[1] D. Michie, "'Memo' Functions and Machine Learning," *J. Phys. A Math. Theor.*, vol. 218, pp. 1–8, 1968.

[2] G. M. Hobold and A. K. da Silva, "Machine learning classification of boiling regimes with low speed, direct and indirect visualization," *Int. J. Heat Mass Transf.*, vol. 125, pp. 1296–1309, 2018.

[3] G. M. Hobold and A. K. da Silva, "Visualization-based nucleate boiling heat flux quantification using machine learning," *Int. J. Heat Mass Transf.*, vol. 134, pp. 511–520, 2019.

[4] M. Ravichandran and M. Bucci, "Online, quasi-real-time analysis of high-resolution, infrared, boiling heat transfer investigations using artificial neural networks," *Appl. Therm. Eng.*, vol. 163, no. September, p. 114357, 2019.

[5] M. He and Y. Lee, "Application of machine learning for prediction of critical heat flux: Support vector machine for data-driven CHF look-up table construction based on sparingly distributed training data points," *Nucl. Eng. Des.*, vol. 338, no. August, pp. 189–198, 2018.

[6] Y. Suh, R. Bostanabad, and Y. Won, "Deep learning predicts boiling heat transfer," *Sci. Rep.*, vol. 11, no. 1, pp. 1–10, 2021.

[7] N. Zuber, "Hydrodynamic Aspects of Boiling Heat Transfer," 1959.

[8] S. G. Kandlikar, "A theoretical model to predict pool boiling CHF incorporating effects of contact angle and orientation," *J. Heat Transfer*, vol. 123, no. 6, pp. 1071–1079, 2001.

[9] S. W. Lee *et al.*, "Study on critical heat flux enhancement in flow boiling of SiC nanofluids under low pressure and low flow conditions," *Int. Congr. Adv. Nucl. Power Plants 2012, ICAPP 2012*, vol. 4, pp. 2182–2190, 2012.

[10] O. Ghaffari, F. Grenier, J. F. Morissette, M. Bolduc, S. Jasmin, and J. Sylvestre, "Pool boiling experiment of dielectric liquids and numerical study for cooling a microprocessor," *Intersoc. Conf. Therm. Thermomechanical Phenom. Electron. Syst. ITHERM*, vol. 2019-May, pp. 540–545, 2019.

[11] S. Sinha-Ray, W. Zhang, B. Stoltz, R. P. Sahu, S. Sinha-Ray, and A. L. Yarin, "Swing-like pool boiling on nano-textured surfaces for microgravity applications related to cooling of high-power microelectronics," *npj Microgravity*, vol. 3, no. 1, pp. 1–8, 2017.

[12] L. Yann, B. Léon, B. Yoshua, and H. Patrick, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE*, vol. 15, no. 1, pp. 19–24, 1998.

[13] Y. Wei *et al.*, "HCP: A Flexible CNN Framework for Multi-Label Image Classificatio," vol. 22, no. 7, pp. 719–725, 2000.

[14] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "CNN-RNN: A Unified Framework for Multi-label Image Classification," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 2285–2294, 2016.

[15] X. Jiang, Y. Wang, W. Liu, S. Li, and J. Liu, "CapsNet, CNN, FCN: Comparative performance evaluation for image classification," *Int. J. Mach. Learn. Comput.*, vol. 9, no. 6, pp. 840–848, 2019.

[16] C. Zhang *et al.*, "A hybrid MLP-CNN classifier for very fine resolution remotely sensed image classification," *ISPRS J. Photogramm. Remote Sens.*, vol. 140, pp. 133–144, 2018.

[17] H. Lee and H. Kwon, "Going Deeper with Contextual CNN for Hyperspectral Image Classification," *IEEE Trans. Image Process.*, vol. 26, no. 10, pp. 4843–4855, 2017.

[18] M. Zhang, W. Li, and Q. Du, "Diverse region-based CNN for hyperspectral image classification," *IEEE Trans. Image Process.*, vol. 27, no. 6, pp. 2623–2634, 2018.

[19] H. Ye, Z. Wu, R. W. Zhao, X. Wang, Y. G. Jiang, and X. Xue, "Evaluating two-stream CNN for video classification," *ICMR 2015 - Proc. 2015 ACM Int. Conf. Multimed. Retr.*, pp. 435–442, 2015.

[20] N. Khan and N. Roy, "Water Quality Assessment with Thermal Images," *Proc. - 2020 IEEE Int. Conf. Smart Comput. SMARTCOMP 2020*, pp. 164–171, 2020.

[21] H. Song, W. Wang, S. Zhao, J. Shen, and K. M. Lam, "Pyramid Dilated Deeper ConvLSTM for Video Salient Object Detection," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11215 LNCS, pp. 744–760, 2018.

[22] S. Kim, S. Hong, M. Joh, and S. Song, "DeepRain: ConvLSTM Network for Precipitation Prediction using Multichannel Radar Data," pp. 3–6, 2017.

[23] A. Kelotra and P. Pandey, "Stock Market Prediction Using Optimized Deep-ConvLSTM Model," *Big Data*, vol. 8, no. 1, pp. 5–24, 2020.

[24] Z. Yuan, X. Zhou, and T. Yang, "Hetero-ConvLSTM: A deep learning approach to traffic accident prediction on heterogeneous spatio-temporal data," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 984–992, 2018.

[25] "PCC Software." [Online]. Available: https://www.phantomhighspeed.com/resourcesandsupport/phantomresources/pccsoftware. [Accessed: 04-Jul-2021].

[26] "split-folders · PyPI." [Online]. Available: https://pypi.org/project/split-folders/. [Accessed: 04-Jul-2021].

[27] "AVI Splitter: How to Split AVI files into Multiple Equal Files." [Online]. Available: https://www.bandicam.com/avi-splitter/. [Accessed: 04-Jul-2021].

[28] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6354 LNCS, no. PART 3, pp. 92–101, 2010.

[29] J. Nagi *et al.*, “Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition,” *Proc. 2019 IEEE Int. Conf. Signal Image Process. Appl. ICSIPA 2019*, pp. 342–347, 2019.

[30] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, 1980.

[31] “Conv2D layer.” [Online]. Available: https://keras.io/api/layers/convolution_layers/convolution2d/. [Accessed: 04-Jul-2021].

[32] C. Debao, “Degree of approximation by superpositions of a sigmoidal function,” *Approx. Theory its Appl.*, vol. 9, no. 3, pp. 17–28, 1989.

[33] “Layer activation functions.” [Online]. Available: https://keras.io/api/layers/activations/. [Accessed: 04-Jul-2021].

[34] “ConvLSTM2D layer.” [Online]. Available: https://keras.io/api/layers/recurrent_layers/conv_lstm2d/. [Accessed: 04-Jul-2021].

[35] “RMSprop.” [Online]. Available: https://keras.io/api/optimizers/rmsprop/. [Accessed: 04-Jul-2021].

[36] “SGD.” [Online]. Available: https://keras.io/api/optimizers/sgd/. [Accessed: 04-Jul-2021].

[37] “sklearn.metrics.confusion_matrix — scikit-learn 0.24.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. [Accessed: 04-Jul-2021].

[38] “seaborn: statistical data visualization — seaborn 0.11.1 documentation.” [Online]. Available: https://seaborn.pydata.org/index.html. [Accessed: 04-Jul-2021].

[39] “keras-io/grad_cam.py at master · keras-team/keras-io · GitHub.” [Online]. Available: https://github.com/keras-team/keras-io/blob/master/examples/vision/grad_cam.py. [Accessed: 15-Jul-2021].

[40] C. Yoo, L. Ramirez, and J. Liuzzi, “Big data analysis using modern statistical and machine learning methods in medicine,” *Int. Neurourol. J.*, vol. 18, no. 2, pp. 50–57, 2014.

[41] A. O. Sykes, “An introduction to regression analysis,” *Sensors (Peterborough, NH)*, vol. 17, no. 9, pp. 68–74, 2000.

[42] M. R. Segal, “Machine Learning Benchmarks and Random Forest Regression,” *Biostatistics*, pp. 1–14, 2004.

[43] L. Breiman, “Random forests,” *Random For.*, pp. 1–122, 2001.

[44] A. Kapelner and J. Bleich, “bartMachine: Machine learning with bayesian additive regression trees,” *J. Stat. Softw.*, vol. 70, no. 2013, 2016.

[45] B. A. Goldstein, A. M. Navar, and R. E. Carter, “Moving beyond regression techniques in cardiovascular risk prediction: Applying machine learning to address analytic challenges,” *Eur. Heart J.*, vol. 38, no. 23, pp. 1805–1814, 2017.

[46] M. I. Shah, M. F. Javed, and T. Abunama, “Proposed formulation of surface water quality and modelling using gene expression, machine learning, and regression techniques,” *Environ. Sci. Pollut. Res.*, vol. 28, no. 11, pp. 13202–13220, 2021.

[47] S. Sarangi, J. A. Weibel, and S. V. Garimella, “Quantitative evaluation of the dependence of pool boiling heat transfer enhancement on sintered particle coating characteristics,” *J. Heat Transfer*, vol. 139, no. 2, 2017.

[48] E. Alic, M. Das, and O. Kaska, “Heat flux estimation at pool boiling processes with computational intelligence methods,” *Processes*, vol. 7, no. 5, pp. 1–16, 2019.

[49] Y. Wang, “High-Dimensional Pattern Regression Using Machine Learning: From Medical Images to Continuous Clinical Variables,” *Image (Rochester, N.Y.)*, vol. 50, no. 4, pp. 1519–1535, 2011.

[50] Z. Niu, M. Zhou, L. Wang, X. Gao, and G. Hua, “Ordinal regression with multiple output CNN for age estimation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 4920–4928, 2016.

[51] X. Cao *et al.*, “Deformable Image Registration based on Similarity-Steered CNN Regression,” *Physiol. Behav.*, vol. 176, no. 5, pp. 139–148, 2017.

[52] Y. Meng *et al.*, “Regression of Instance Boundary by Aggregated CNN and GCN,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12353 LNCS, pp. 190–207, 2020.

[53] H. Nakahara, T. Fujii, H. Yonekawa, and S. Sato, “A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA,” *FPGA 2018 - Proc. 2018 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, vol. 2018-Febru, pp. 31–40, 2018.

[54] J. Yuan, B. Ni, and A. A. Kassim, “Yuan et al. - Half-CNN A General Framework for Whole-Image Regr,” pp. 1–9.

[55] T. Sattler, Q. Zhou, M. Pollefeys, and L. Leal-taix, “Understanding the Limitations of CNN-based Absolute Camera Pose Regression Chalmers University of Technology,” *Cvpr*, pp. 3302–3312, 2019.

[56] N. Chinaev, A. Chigorin, and I. Laptev, “MobileFace: 3D face reconstruction with efficient CNN regression,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11132 LNCS, pp. 15–30, 2019.

[57] A. S. Jackson, A. Bulat, V. Argyriou, and G. Tzimiropoulos, “Large Pose 3D Face Reconstruction from a Single Image via Direct Volumetric CNN Regression,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 1031–1039, 2017.

[58] S. Mahendran, H. Ali, and R. Vidal, “3D Pose Regression using Convolutional Neural Networks 6D Task : given a single 2D image , estimate 6D object pose.”

[59] S. Miao, Z. J. Wang, Y. Zheng, and R. Liao, “Real-time 2D/3D registration via CNN regression,” *Proc. - Int. Symp. Biomed. Imaging*, vol. 2016-June, pp. 1430–1434, 2016.