Data Science: Process Discovery and Analysis

Primary Topic: Process Mining, Secondary Topic: Data Mining
Course: 2022-1B – Group: 82 – Submission Date: 2023-02-05

Sara Miotto
University of Twente
s.miotto@student.utwente.nl

Victor Dibbets
University of Twente
v.l.dibbets@student.utwente.nl

ABSTRACT

The purpose of the paper is to highlight alternative approaches meant to optimize a business process. First of all, the phases of process discovery and analysis are explored with the aim of *ProM*. In a second moment, the collaboration between process mining and deep learning has been taken into account. A convolutional neural network has been built and trained on historical data in order to predict the most suitable outcomes for the process studied.

KEYWORDS

Process Mining, Data Mining, Deep-learning, Neural Networks, Business Process

1 INTRODUCTION

The latest advancements in technology have allowed companies to look deeply into their operations utilizing data from information systems. One such technique to deal with it is process mining. It essentially extracts insights from the event logs which in turn are captured by process-aware information systems (PAIS). PAIS monitor all the necessary steps involved in executing a particular workflow.

The objective of this project is to analyze and optimize the performances and conformance of a job-shop manufacturing facility. Improving the process means to find out how to achieve higher performance while using fewer resources as well as evaluating whether certain processes could be more efficiently handled if outsourced to an external party, rather than remaining in-house. So, the main questions are:

- How can we identify possible bottlenecks within an industrial process?
- How could we make this same process more streamlined and efficient?
- *ProM* is definitely an essential tool, but are there any other interesting paths?

Throughout the project we will try to provide answers to these questions in such a way to be helpful to any managers who want to improve the productivity of their enterprises.

The most straightforward way to tackle this problem, as previously anticipated, is by means of *ProM Software*. The exploitation of plugins such as *Inductive Visual Miner*, *Replay a Log on Petri net for performance/conformance*, and Conformance Checking for DPN already allow us to point out some ways to obtain progress.

Regarding the alternative approach, we were very curious in the idea of combining process mining with deep learning. In fact deep learning is nowadays a leader in each single field, so what prevents us from using it also for this scope?

The general study-case scenario, the *job-shop scheduling problem*, aims at finding the most efficient way to combine a set of industrial operations so as to minimize the total amount of time required to complete the whole process. Solutions are essentially made of permutations of the orders in which activities should be executed as well as their assignment to one machine rather than another. Of course factors such as the prioritization of jobs and the actual ability of the different agents has to be taken into account.

The paper has the following structure: *Background and Related Work* deals with the prior knowledge recommended for a better understanding of the experiments. In *Approach* the two alternative strategies are described in details. In *Experiments* a more quantitative analysis is carried out so as to prove empirically the approaches. It also provides a description of the data set and an overview of the experimental design. *Discussion* presents a reflection upon the findings. Finally *Conclusion* takes stock of the situation and discusses possible future developments.

2 BACKGROUND AND RELATED WORK

Useful prerequisites to better understand the experiments are for sure a certain knowledge on how to use *ProM* software and a smattering of how neural networks work.

PM techniques can be applied to various scenario, such as economic activities or production processes, and can assist managers with the aim of reducing expenses, augmenting efficiency and enhancing the overall performance. A peculiar typology of data is considered, namely the log files, i.e. collections of events recording ordered operations. They can come from a wide variety of sources, including transaction processing systems, wireless sensor networks, and social media. Thanks to these information, process mining is able to combine data mining and business process management in order to understand properly the event logs.

In particular, data mining is very useful in identifying patterns and trends within the data, and this can be used to improve the workflow, delete bottlenecks and increase the overall efficiency. Instead business process management methodologies aims at designing, analyzing, and enhancing business processes.

Another interesting collaboration is the one between process mining and multi-agent systems. When these two areas are combined, process mining tries to examine the data produced by multi-agent systems, leading to an improved coordination of the agents. Additionally, multi-agent systems can be employed to carry out process mining tasks, such as discovering processes and conformance checking, in a distributed and decentralized way. This can result in more precise outcomes and enforce the scalability of PM. All in all, it can lead to a robust approach for strengthening business processes in real-world situations where emergent behaviors are on the agenda.

The traditional *modus operandi* exploits three different stages: process discovery, conformance checking, and enhancement. Process

Table 1: Some statistics within the log file

Unique Activity identifier:	7
Total Number Activities:	49520
Max lenght:	40
Mean lenght:	34.70
Min lenght:	32

discovery creates a comprehensive model based on the behavior observed in the log file. Conformance checking compares the process with the recorded behavior, and finally enhancement adds data from the event log to the process model.
We have taken inspiration from different papers while developing this project. "Business process mining: An industrial application" by W.M.P. van der Aalst ([5]) aims at translating PM techniques from the world of simulation to the real one. Specifically in the paper he has described a case study in which process mining was applied to a provincial office of the Dutch National Public Works Department, responsible for the construction and maintenance of roads and of water infrastructure. The study has used a variety of techniques in order to analyze how invoices were sent by subcontractors and suppliers.
Some years later, in 2016, this same author has published another paper, "Distributed Process Discovery and Conformance Checking" ([3]). More specifically this paper deals with the phases of process discovery and conformance checking. It is also pointed out that one of the key motivations for using process mining is the availability of event data. However, as soon as event logs become larger, performance can constitute an issue. To address this, process mining problems can be distributed over a network of computers, such as multi-core systems, grids, and clouds, and tackled in various ways, including an horizontal or a vertical partitioning of the event log. These techniques are discussed in the context of both procedural models, such as Petri-nets, and declarative process models.
The last paper of note is "An agent-based process mining architecture for emergent behavior analysis" by Rob Bemthuis, Martijn Koot, Martijn Mes, Faiza Bukhsh, Maria-Eugenia Iacob, and Nirvana Meratnia [1]. This paper has inspired this report and moreover it has worked with the same data set.
Regarding the deep learning part, we have found a good starting point in "Using Convolutional Neural Networks for Predictive Process Analytics" ([4]), which suggests to exploit convolutional neural networks to predict activities within a business process. This approach involves the conversion of temporal data of the event logs into spatial patterns. Images are at a later stage used to train the neural network. Such a predictive model is a powerful tool to support participants in completing business processes, by acting proactively in anticipation.

3 APPROACH

So, *How can we really optimize the studied process*? In this section we will try to suggest some ways to do it. First statistics about log files in the dataset are provided.

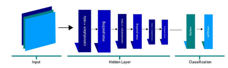

Figure 1: Layout CNN

3.1 Traditional Process Mining

The traditional approach to process mining requires *ProM*, an open-source tool with a wide range of features and functionalities, free and quite customizable. For the aim of this project we have chosen to use version 6.12 because some of the needed plugins were not otherwise present in the lite version.
The general framework for discovering bottlenecks and figuring out how to improve the efficiency of the process, consists of two principal steps. Let's delve into them. *Process discovery* is a crucial phase for the understanding of the study-case scenario and the identification of patterns as well as of relationships between events. It is meant to analyze the data and extract a visual representation of the sequence of activities. Among the different plugins suggested for this scope, *Inductive Visual Miner*, *IVM*, seemed to be the most suitable thanks to its intuitiveness, its ability to provide a clear visual representation through flowcharts, and its robustness to noise and outliers. The process is represented as a directed graph, where each node is an activity and edges are flows between activities. Such a visualization makes it easy to identify immediately bottlenecks and areas for improvement. Furthermore *IVM* is based on an interesting combination of heuristics and machine learning which allows to solve the problem of loops. The proper outcome of the discovery phase ([2]) will be discussed more in details in the experiments section.
The second phase is the analysis of the process itself and two different plugins have been used for this scope, *Replay a Log on Petri net for performance/conformance*, *RLPN*, and *Conformance Checking for DPN*, *CCDPN*. Regarding *RLPN*, it features a large variety of information which can be extracted from the data. Here in particular we are interested in the knowledge you can gain from timestamps and the respective waiting time in the process.
Lastly, *CCDPN* aims at determining the conformance of a certain event log to the model. If an event log does not fit in it correctly, then this may indicate an error within the process and could lead to an uptick in performance if fixed. In order to work properly, it requires the petri-net of an event log considered "good" and taken as an example. It also provides some useful statistics.

3.2 A new interesting collaboration - Process Mining and Neural Networks

A very interesting collaboration to explore is the one between deep learning and process mining. Training a neural network on historical event logs in order to predict the next action in a business process could be very useful for its own optimization. Bottlenecks would be detected and consequently eliminated.
In the specific case of this project, this was attempted through a convolutional neural network. Such a neural network is meant to exploit the spatial correlations of 2-D images which were obtained after the conversion of the temporal information into the spatial one.
To make the dataset acceptable by the neural network, a pre-processing phase was required: log files were imported and converted into pandas dataframe. From this dataframe the columns listing the activities (mapped into integer numbers) as well as the timestamps were the ones useful for the prediction and so kept and used as dimension of the images.
Generate prefix trace takes in the dataframe and the number of case IDs as parameter and splits the dataframe into training and test sets.
So *Generate image*, as the name suggests, is the function which creates the images. For a correct representation it requires information about the maximum length of the trace, and on the total number of activities in the dataset. It iterates over the length of the activities, gets the starting time of the trace, calculates its duration, and sets this value in the image matrix. The dimensions are given by the actions performed (y) and the temporal extension of those same actions (x), respectively. The images must have all the same size, which is why the number of rows is set equal to the lenght of the longest action. Each pixel in the (x, y) position is a 2-dimensional vector providing information of the activity and the performance channel. The activity channel shows how many times a certain activity has occurred in the prefix trace up to the considered timestamp. The performance channel instead displays the duration.
Finally, the labels for the training and test dataset are generated by using the *get label* method. The target y is equal to the particular activity which should follow the corresponding input sequence X. Labels are converted in one-hot-encoding notation.
The neural network is now ready to be fed. It has the following structure (see figure 1): a first convolutional layer receives the image as input and applies to it its filters as well as the *ReLu* activation function. A pooling layer is subsequently used to ensure temporal invariance and to reduce the size. In this case we are dealing with max pooling. Convolution and max pooling layer alternate and decrease the number of features processed yet also the resolution of the image. The final layer is fully connected and extracts the outcome of the prediction. As long as we are dealing with probabilities, the activation function applied here is the softmax. The resulting output is essentially a matrix of probability values among which the network chooses the highest one representative of the activity sought.
The training phase requires the classical backpropagation algorithm, with Nadam as optimizer. The regularization techniques of early stopping, batch normalization and L2 have been implemented with the purpose of preventing the phenomenon of overfitting.
The predicted labels are then translated into the original events using the *inverse transform* method.
Performances are evaluated through the *classification report* by comparing the predicted labels to the expected ones.
Furthermore the code shows the activities classified as bottlenecks. In this specific case the network has detected inefficiencies in the early phase of the process, namely activities labelled with IDs 1 and 5, that are exactly the ones corresponding to the transport phase, already detected as optimizable by the *ProM* analysis.
Bottlenecks are defined as those actions for which the ratio between the specific duration and the mean overall duration is greater than 1.5.

4 EXPERIMENTS

Here a quantitative analysis of the experiments is carried out. But let's first analyze better the dataset.

4.1 Dataset

Each event log file lists the activities, the *trace*, of the process as well as the amount of time needed to carry them out.
Events are identified by a code, a date, the type of the commodity (console, helicopter, or robot), the specific action (arrival, draining, drilling, painting, sawing, transport, welding), and the status of the completion of the action itself. The agents taking part in the experiment are three, namely machine agents, responsible for organizing the flow of operations performed by the machinery, Automated Guided Vehicles Agents, AGVs, used to transport the products, and the products themselves. The AGVs travel along a single track and need to avoid collisions.
The table below reports the distribution of actions within event log 613, one of the best from an efficiency point of view.

Identify activity	Activity Distribution
2	24063
3	5979
4	5967
5	3910
7	3893
1	2854
6	2854

4.2 General experimental setup

An event starts with the arrival of a product. Then the product goes through three to four processing stages before being drained and removed from the process. Already at this early stage, it is evident how a big amount of time is spent on the transportation phase. Secondly, depending on the type of product, loops are a serious an issue in this phase.
Each experiment is characterized by three variables, X, Y, and Z, where X is the number of a particular vehicle $(4, 5, 6)$, Y is the direction of the vehicle itself, and Z specifies how the delivery is carried out. As a consequence there are three different scenarios with respectively three variations, resulting in a total of 27 combinations. The first scenario involves changing the number of vehicles, the second of the direction of the vehicles, and the third of the dispatch of AGVs.

4.3 Traditional Process Mining Results

As discussed in *Approach* section, here the scope is to delve deeper into the findings derived by the various plug-ins used, and draw conclusions from the results.

4.3.1 Discovery Phase. The outcomes of *IVM* can be visualized in figure 6. Let's delve into it. First of all it was necessary to load experiment 613 in *ProM* and select *Mine with Inductive Visual Miner* plugin. As a consequence, a very high-level overview of the process has been plotted, but to have a more in-depth view of the process it was essential to add as classifier "concept:instance". This results in the model of figure 6 which is much more clear.
However, from an approximate analysis of the process some suggestion regarding the possibilities of improvement can be made. In fact, all the products have gone through the following loop: transport $\rightarrowtail$ activity $\rightarrowtail$ transport $\rightarrowtail$ activity. This is of course very inefficient. Moreover, *IVM* plugin considers the timestamps and moves dots over the model so as to represent workflows. Dots show how much time is spent in transportation.
So, what the process owner could do, would be to try to reduce the amount of time spent on transport in the fabrication process. There are several ways to deal with this issue:

- Using an optimization algorithm to figure out what is the most convenient path to follow (for example in the field of Bio-Inspired Artificial Intelligence there is an algorithm called Ant Colony Optimization whose purpose is precisely to solve graphical combinatorial problems like this);
- By applying the principles of Lean Manufacturing, you can reduce waste and improve transportation efficiency. Lean Manufacturing is an approach focused on reducing waste and increasing efficiency. When these principles are applied to transportation in manufacturing, it is possible to reduce waste and improve transportation efficiency;
- Using "Just-In-Time" (JIT) delivery, that is, ensuring that materials are delivered exactly when they are required, can reduce the time spent in transportation. JIT delivery aims to reduce waste and increase efficiency by delivering only the materials that are needed at the precise time they are required. This reduces the time that is spent in transportation because there is no accumulation or storage of unused materials;
- By combining multiple deliveries into one trip, the overall time spent on transportation can be reduced;
- By automating transportation processes, such as materials handling and loading, transportation efficiency can be improved. Automating these processes means using automated technologies and systems to perform certain tasks, instead of manual work. This increases the efficiency of transportation, as tasks are performed faster and with greater accuracy;
- By incorporating technology, such as real-time monitoring and optimization software, it is possible to monitor and improve transportation efficiency.

4.3.2 Analysis Phase. As mentioned earlier, during the analysis phase two plugins were used. First of all *Replay a Log on Petri-net for performance/conformance* has been tested. It requires an event log and a Petri-net to run. In particular, in order to create figure 7, the event log of experiment 412 was used as well as the corresponding Petri-net (this latter generated through *Inductive Miner*).
Thanks to this we are able to detect eventual bottlenecks. Besides the numerous issues affecting transportation that have already been commented, two other activities spent an out of proportion amount of time in carrying out their task, namely welding and painting. Therefore the project manager should look deeply into these two phases and try to understand how to increase the efficiency. Some suggested strategies are:

- Process standardization: Standardizing the welding and painting processes can help reduce variability and improve efficiency;
- Workflow optimization: Analyzing and optimizing the workflow of welding and painting can help reduce the time spent on these activities;
- Equipment optimization: Upgrading or optimizing the equipment used in welding and painting can improve the efficiency of these processes;
- Employee training: Providing proper training to employees on welding and painting techniques can improve the efficiency and quality of these processes;
- Automation: Automating some parts of the welding and painting processes can improve efficiency and reduce the time spent on these activities.

Time between transition class analysis has helped in these two phases. It is essentially a tool used to measure and analyze the average time between process transitions and how that time is distributed. Its purpose is to identify bottlenecks, inefficiencies, and opportunities for improvement in the process. This analysis provides information on the duration between different stages of the process and can highlight areas where tasks take too much time or where there are long waiting times. By examining this data, process owners can make informed decisions on how to improve their processes, such as by reducing task completion time or introducing automation.
Lastly, *Conformance Checking for DPN* compares a modeled process with an actual process execution. By providing a Petri net model and an event log, the tool can check the degree of conformance between the two and calculate a fitness score. Therefore, in order to work, it needs the petri-net model and the event log of an experiment deemed excellent. Within the provided dataset, experiment labelled with 613 and 621 are the ones recommended.
So, we have tried to optimize experiment 413 using the petri-net of experiment 613 as reference model. The average fitness gained was equal to 0.95 meaning that the 95% of the discovered process steps match the actual steps, a really good result.
We have tried this technique also on other dataset and the results can be found in the table below:

Experiment #	*Average fitness*
412	0.95
413	0.95
433	0.95
511	0.95
522	0.95
631	0.96

Table 2: Accuracy's scores

accuracy	0.55
macro average	0.40
weighted average	0.44

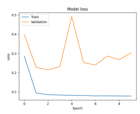

Figure 2: Valid and train loss

They seem very consistent and high proving that conformance is probably not a big issue.

4.4 A new interesting collaboration - Process Mining and Neural Networks Results

In table 2 we can see the accuracy gained after having trained the neural network for 100 epochs. Moreover, in figure 2 the trends of training and validation loss may be compared. It is of course possible via the appropriate python function to switch back from the image to the log file, then feed it to *ProM* for further analysis..

5 DISCUSSION

Some considerations about the two alternative approaches are provided here.

5.1 Traditional Process Mining

The first thing we have done was to try to document ourselves about the various plugins that exist in *ProM*. There are so many of them, and it would have been interesting to try others for sure, but we felt that the ones we have selected were the most suitable for the unique value proposition of this project.

5.2 Deep Learning/Neural Networks

Regarding the deep learning part, we had initially tried to use an LSTM, since usually such a neural network deals very well with temporal patterns and thus with log files. However, after having read the paper [4], we have noticed that the approach described there, (the conversion of temporal patterns into spatial patterns) was very interesting and so we have tried to implement it.
The values in the table 2 represent accuracy, that is, the percentage of correct predictions within the test set, the macro average, i.e. the average of the metric calculated without considering the number of instances for each class and the weighted average, which takes into account the proportion of instances for each class.
Clearly, our accuracies are far from those reported in the paper, which are around 0.8. This is because we did not have an appropriate dataset on which to train the model. In fact, as a training set we simply used the same log file we were trying to optimize, and at the same time it was too small to allow the neural network to fully understand correlations between events. The trends of train loss and valid loss can also be observed in figure 2.
However, with appropriate training dataset, such an approach would prove to be useful and effective.
In fact, in the real world, sometimes business processes have very complicated patterns, known as *spaghetti-like* models. So, having a pre-compiled neural network able to analyze them, could really save time and efforts to managers.

6 CONCLUSIONS

In summary, *ProM* and neural networks have their own strengths and limitations. *ProM* is a well-established and reliable tool that excels in process mining tasks, whereas neural networks are advanced machine learning methods that excel in prediction and optimization. By utilizing both methods together, it may lead to even better optimization outcomes. For example, at first the problem could be tackled by the neural network and then, the resulting predicted sequence could be treated as input for *ProM* and analyzed also there.
To sum up, why should a manger take into account these factors when dealing with a business process?

- Enhancing process effectiveness: by using data from process-aware systems, leaders can detect delays and inefficiencies in the workflow. This can result in enhanced process performance and increased output;
- Making more informed choices: Process mining can provide leaders with valuable insights into process performance, which can be employed to make better decisions. For instance, leaders can use process mining to pinpoint the underlying cause of process issues and take steps to rectify them;
- Improving customer satisfaction: By enhancing process efficiency, leaders can decrease the time needed to complete a process, which can lead to higher customer satisfaction. Additionally, process mining can be used to identify customer needs and adapt processes to meet those needs;
- Ensuring compliance: Process mining can be used to verify if the observed behavior of a process adheres to the process model, and can pinpoint any deviations, which can be employed to guarantee compliance with regulations and internal policies;
- Reducing costs: Enhancing process efficiency can result in cost savings by reducing waste and rework, and increasing productivity.

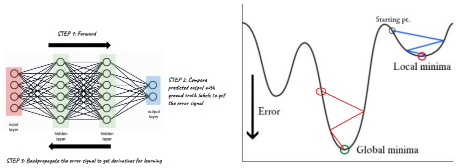

Figure 3: backpropagation

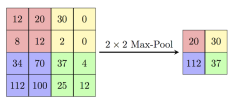

Figure 4: max pooling

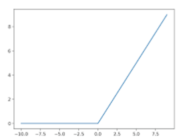

Figure 5: ReLu activation function

Table 3: CNN structure

Layer (type)	Output Shape
convolutional	(None, 40, 7, 32)
batch normalization	(None, 40, 7, 32)
activation	(None, 40, 7, 32)
max pooling2d	(None, 20, 4, 32)
convolutional	(None, 20, 4, 64)
batch normalization	(None, 20, 4, 64)
activation	(None, 20, 4, 64)
max pooling	(None, 10, 2, 64)
flatten (Flatten)	(None, 1280)
Dense output	(None, 6)
Total parameters	41,190
Trainable parameters	40,998

A APPENDIX

In this appendix section we will provide a description for some more technical concepts used both in *ProM* analysis and in Deep Learning. Also the python code related to the deep learning will be provided.

A.1 CNN layout

Observing the figure (see table 3) it is possible to dig deeper into the internal layout of the neural network used. It consists of two convolutional layers, two max pooling layers and one fully connected layer.
In convolution layers the hidden units are connected to local patches within the feature maps of the previous layer through kernels. The units in a local patch are convolved by the weight matrix, and then passed through the non linearity activation function, in this case a ReLu (see figure 5). Output feature maps are computed by the convolution between all the previous layer's feature maps and a bank of filters. A bias is added.
The pooling layer aims at merging semantically similar features that are detected by the convolutional layer. Essentially these layers can be considered as sub-sampling layers which reduce the dimension of features as well as make the representation invariant to small shifts and distortions of the input. In fully connected layers each neuron is connected to all the neurons of the previous layer.
Finally the output layer is a softmax classifier and it has the purpose of predicting the classification of the input sample. The label of the class is a one-hot encoding vector.

A.2 Regularization techniques

In order to prevent the problem of overfitting three regularization techniques have been implemented: L2-regularization, batch normalization and early stopping.
First of all, L2-regularization, or weight decay, was applied to convolutional layers. It aims at limiting the model capacity by adding the squared magnitude of the coefficient as an additional term to the loss function.
Secondly, the batch normalization is a method to reparameterize a deep network. In fact, as learning progresses, the distribution of layer inputs changes due to parameter updates. This phenomenon is known as internal covariate shift and can result in most inputs being in a nonlinear regime and slow down learning. Batch normalization, by calculating the mean and the standard deviation of all the activation of a layer, can solve this.
Lastly, early stopping means starting with small weights and stopping the learning before it overfits. As stopping criterion a monitor on the loss in validation set has been set.

A.3 Backpropagation

Backpropagation is a method for computing gradients, where the gradient is the vector of partial derivatives with reference to all the coordinates of the weights. Each partial derivative measures how fast the loss changes in one direction. Gradient descent aims at finding the model parameters that correspond to the best fit between predicted and actual outputs (see figure 3).

A.4 Nesterov-accelerated Adaptive Moment Estimation, Nadam

As optimizer Nadam was chosen. It is essentially a variation of the Adam algorithm and includes Nesterov momentum which leads to an improvement in the performance of the optimization process. The loss function is the function that needs to be minimized in the process of fitting a model, and it represents a selected measure of the discrepancy between the observed data and the data predicted by the fitted function. Here it detects how closely the predicted probability of each class matches the actual outcome, either 0 or 1. The score from the log loss function increases as the predicted probability becomes different from the expected ones. If they are very different, then the score will be high. On the other hand, if the predicted probability is close to the actual outcome, the score will be low.

A.5 Python Deep Learning Code

Here below are reported all the screens of the python code implemented:

```
array([[9.99941230e-01, 4.72640641e-05, 5.66892686e-06, 1.38860443e-07,
        6.75992773e-09, 5.65192295e-06],
       [9.99998331e-01, 5.78436129e-07, 5.61146692e-07, 2.34330972e-08,
        2.24324448e-10, 4.72977348e-07],
       [9.99993563e-01, 5.30837542e-06, 5.53532686e-07, 2.09205879e-08,
        2.61624583e-10, 5.12397150e-07],
       ...,
       [4.36671631e-04, 0.00000000e+00, 1.36869057e-18, 2.94294911e-23,
        9.99563277e-01, 2.73246624e-31],
       [2.13579288e-09, 0.00000000e+00, 4.02033336e-23, 3.04550904e-27,
        1.00000000e+00, 7.50570479e-37],
       [1.30603835e-11, 0.00000000e+00, 2.06351271e-25, 2.63104480e-29,
        1.00000000e+00, 0.00000000e+00]], dtype=float32)
```

```python
import numpy as np
import tensorflow as tf
from sklearn import preprocessing
from tensorflow.keras.layers import Conv2D, Activation
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import BatchNormalization
from ImagePPMiner import ImagePPMiner
import matplotlib.pyplot as plt
seed = 123
np.random.seed(seed)
tf.random.set_seed(seed)
import argparse
import warnings
warnings.filterwarnings("ignore")
```

```python
if __name__ == '__main__':

    dataset = '613'
    n_layer = 2
    pm = ImagePPMiner(dataset)

    # Create an instance of the ImagePPMiner class and pass in the dataset file name as a parameter
    log = pm.import_log()

    # Import and process the log file to generate a dataframe
    max_trace, n_caseid, n_activity = pm.dataset_summary(log=log)

    # Generate summary statistics of the log data and assign them to variables
    train_act, train_temp, test_act, test_temp = pm.generate_prefix_trace(log=log, n_caseid=n_caseid)

    # Split the log data into training and test datasets for activities and timestamps
    X_train = pm.generate_image(act_val=train_act, time_val=train_temp, max_trace=max_trace, n_activity=n_activity

    # Generate image representation of the training data
    X_test = pm.generate_image(act_val=test_act, time_val=test_temp, max_trace=max_trace, n_activity=n_activity)

    # Generate image representation of the test data
    l_train = pm.get_label(train_act)

    # Get the labels for the training data
    l_test = pm.get_label(test_act)

    # Get the labels for the test data
    le = preprocessing.LabelEncoder()
```

```python
    # Create an instance of the LabelEncoder class
    l_train = le.fit_transform(l_train)

    # Encode the labels for the training data
    l_test = le.transform(l_test)

    # Encode the labels for the test data
    num_classes = le.classes_.size

    # Get the number of classes in the dataset
    X_train = np.asarray(X_train)

    # Convert X_train to a numpy array
    l_train = np.asarray(l_train)

    # Convert l_train to a numpy array
    X_test = np.asarray(X_test)

    # Convert X_test to a numpy array
    l_test = np.asarray(l_test)

    # Convert l_test to a numpy array
    train_Y_one_hot = tf.keras.utils.to_categorical(l_train, num_classes)

    # One-hot encode the training labels
    test_Y_one_hot = tf.keras.utils.to_categorical(l_test, num_classes)

    # One-hot encode the test labels
    test_Y_one_hot = tf.keras.utils.to_categorical(l_test, num_classes)
```

```python
    # Calculate the average duration of each activity
    activity_durations = {}

    # Iterate over each activity
    for i in range(n_activity):

        # Get the mean duration of the current activity
        activity_durations[i] = np.mean(X_train[:, i])

        # Calculate the average duration of all activities
        mean_duration = np.mean(list(activity_durations.values()))

    # determine which activities are bottlenecks
    bottlenecks = []

    # Iterate over each activity
    for i in range(n_activity):

        # Calculate the ratio of the current activity's duration to the mean duration
        ratio = activity_durations[i] / mean_duration

        # If the ratio is greater than 1.5, consider the current activity to be a bottleneck
        # you can adjust the value of 1.5 to suit your needs
        if ratio > 1.5:
            bottlenecks.append(i)
```

```python
    for bottleneck in bottlenecks:

        # Get the index of the bottleneck activity
        index = bottleneck

        # Iterate over the columns in the dataframe
        for col in log.columns:

            # If the column name contains "concept:name" and the value is 1, print the activity name
            if "concept:name_" in col and log.iloc[index][col] == 1:
                print(col.split("_")[-1])


    # Print the bottleneck activities
    print("Bottleneck activities:", bottlenecks)
```

```python
    # Neural network architecture
    model = Sequential()
    reg = 0.0001
    input_shape = (max_trace, n_activity, 2)
    int(n_layer) == 2
    model.add(Conv2D(32, (2, 2), input_shape=input_shape, padding='same', kernel_initializer='glorot_uniform',
                     kernel_regularizer=tf.keras.regularizers.l2(reg)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

    model.add(Conv2D(64, (4, 4), padding='same', kernel_regularizer=tf.keras.regularizers.l2(reg), ))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

    model.add(Flatten())
    model.add(Dense(num_classes, activation='softmax', name='act_output'))

    print(model.summary())

    opt = Nadam(lr=0.0002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004) #clipvalue=3)
    model.compile(loss={'act_output': 'categorical_crossentropy'}, optimizer=opt, metrics=['accuracy'])
    early_stopping = EarlyStopping(monitor='val_loss:', patience=9)
    history = model.fit(X_train, {'act_output': train_Y_one_hot}, validation_split=0.2, verbose=2, callbacks=[earl

    y_pred_test = model.predict(X_test)
    max_y_pred_test = np.argmax(y_pred_test, axis=1)
    max_y_test = np.argmax(test_Y_one_hot, axis=1)
    print(classification_report(max_y_test, max_y_pred_test, digits=3))
```

```python
    le = preprocessing.LabelEncoder()
    l_train = le.fit_transform(l_train)
    l_test = le.transform(l_test)
    num_classes = le.classes_.size

    # Get predictions for the test data
    predictions = model.predict(X_test)

    # Plot the training loss and validation loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()
```

```python
from pm4py.objects.conversion.log import converter as log_converter
from pm4py.objects.log.importer.xes import importer as xes_importer
import numpy as np

class ImagePPMiner:
    def __init__(self, eventlog):
        """
        Initialize the ImagePPMiner class with the eventlog file name.
        Parameters:
        eventlog (str): name of the eventlog file.
        """
        self._eventlog = eventlog

    def import_log(self):
        """
        This method imports the XES log using the xes importer and converts the log to a pandas dataframe.
        Then it maps the names of activities to unique integers, select the columns "case:concept:name", "concept:
        """
        # Import the XES log using the xes importer
        log = xes_importer.apply('ImagePPMiner-master/dataset/'+self._eventlog+'.xes')
        # Convert the log to a pandas dataframe
        dataframe = log_converter.apply(log, variant=log_converter.Variants.TO_DATA_FRAME)

        # Get the unique names of activities in the log
        unique = dataframe['concept:name'].unique()
        # Create a dictionary mapping the names of activities to unique integers
        dictOfWords = { i : unique[i] for i in range(0, len(unique) ) }
        dictOfWords = {v: k for k, v in dictOfWords.items()}
        for k in dictOfWords:
            dictOfWords[k] += 1
        # Replace the original names of activities with the unique integers
```

```python
        # Replace the original names of activities with the unique integers
        dataframe['concept:name'] = [dictOfWords[item] for item in dataframe['concept:name']]
        # Select the columns "case:concept:name", "concept:name", "time:timestamp" from the dataframe
        dataframe = dataframe[["case:concept:name", "concept:name", "time:timestamp"]]
        # Return the modified dataframe
        return dataframe

    def generate_prefix_trace(self, log, n_caseid):

        # grouping the log dataframe by case:concept:name and taking the concept:name column
        # and applying the list function to it, to convert the values into a list
        act = log.groupby('case:concept:name', sort=False).agg({'concept:name': lambda x: list(x)})

        # grouping the log dataframe by case:concept:name and taking the time:timestamp column
        # and applying the list function to it, to convert the values into a list
        temp = log.groupby('case:concept:name', sort=False).agg({'time:timestamp': lambda x: list(x)})

        # calculating the size of the train set, using n_caseid as 2/3 of the total dataframe size
        size = int((n_caseid / 3) * 2)

        # splitting the act dataframe into train and test sets
        train_act = act[:size]

        # splitting the temp dataframe into train and test sets
        train_temp = temp[:size]

        # splitting the act dataframe into train and test sets
        test_act = act[size:]

        # splitting the temp dataframe into train and test sets
        test_temp = temp[size:]
```

```python
        # returning the train_act, train_temp, test_act, test_temp variables
        return train_act, train_temp, test_act, test_temp

    @staticmethod
    def generate_image(act_val, time_val, max_trace, n_activity):

        # Initiate loop variables and image matrix
        i = 0
        matrix_zero = [max_trace, n_activity, 2]
        image = np.zeros(matrix_zero)
        list_image = []

        # Iterate over the length of time_val
        while i < len(time_val):
            j = 0
            list_act = []
            list_temp = []

            # Create a list of integers from 1 to n_activity
            a = list(range(1, n_activity + 1))
            dict_act = dict.fromkeys(a, 0)
            dict_time = dict.fromkeys(a, 0)

            # Iterate over the length of activity value
            while j < (len(act_val.iat[i, 0]) - 1):

                # Get start time of the trace
                start_trace = time_val.iat[i, 0][0]

                # Increase the value of the activity in the dictionary
                dict_act[act_val.iat[i, 0][0 + j]] += 1
```

```python

                # Increase the value of the activity in the dictionary
                dict_act[act_val.iat[i, 0][0 + j]] += 1

                # Calculate duration of the activity
                duration = time_val.iat[i, 0][0 + j] - start_trace

                # calculate days
                days = (duration.total_seconds())/86400
                dict_time[act_val.iat[i, 0][0 + j]] = days
                l_act = list(dict_act.values())
                l_time = list(dict_time.values())
                list_act.append(l_act)
                list_temp.append(l_time)
                j = j + 1
                cont = 0
                lenk = len(list_act) - 1

                # Create image
                while cont <= lenk:
                    z = 0
                    while z < n_activity:
                        image[(max_trace - 1) - cont][z] = [list_act[lenk - cont][z], list_temp[lenk - cont][z]]
                        z = z + 1
                    cont = cont + 1
                if cont > 1:
                    list_image.append(image)
                    image = np.zeros(matrix_zero)
            i = i + 1
        return list_image
```

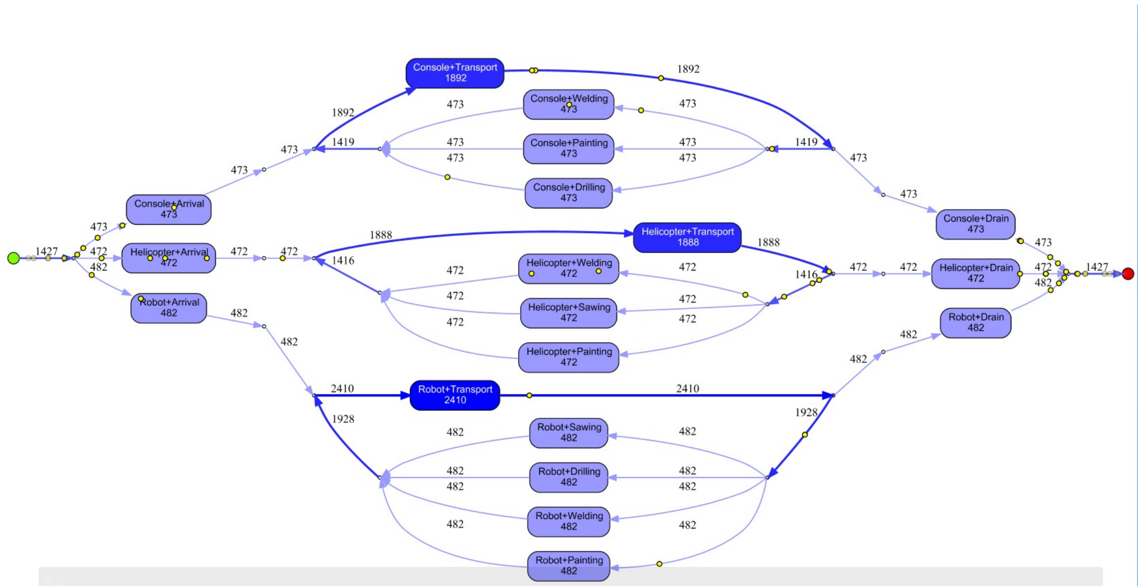

Figure 6: Inductive Visual Miner on experiment 613

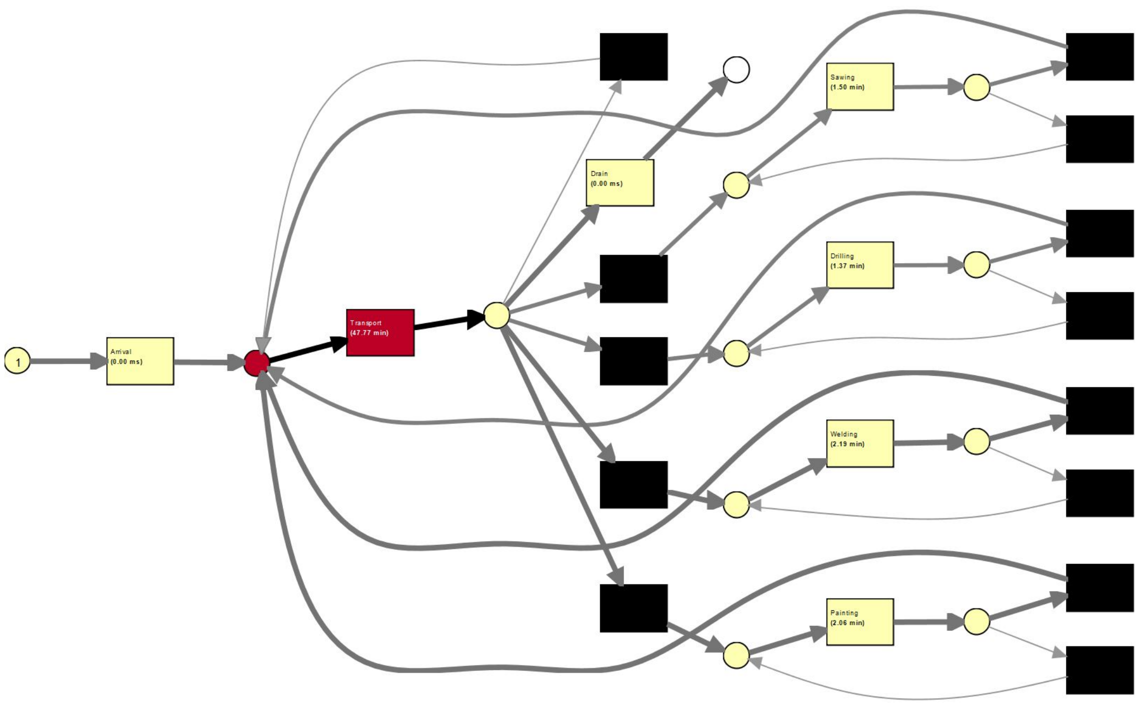

Figure 7: RLPN on experiment 412

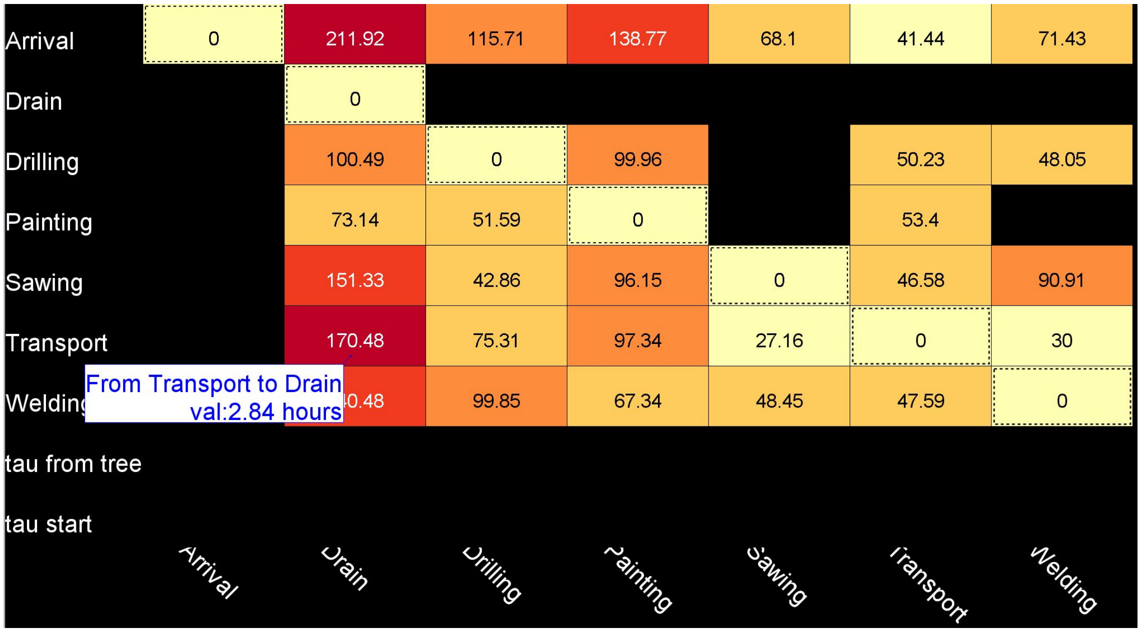

Figure 8: RLPN - Time between transition class analysis visualization - on experiment 412

REFERENCES

[1] Rob H. Bemthuis, Martijn Koot, Martijn R. K. Mes, Faiza A. Bukhsh, Maria-Eugenia Iacob, and Nirvana Meratnia. 2019. An agent-based process mining architecture for Emergent Behavior Analysis. *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)* (Oct 2019). https://doi.org/10.1109/edocw.2019.00022

[2] Sander J.J. Leemans. 2017. Inductive visual miner - leemans.ch. https://www.leemans.ch/publications/ivm.pdf

[3] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. 2016. Scalable process discovery and conformance checking. (Jul 2016). https://doi.org/10.1007/s10270-016-0545-x

[4] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, and Donato Malerba. 2019. Using Convolutional Neural Networks for Predictive Process Analytics. (2019). https://doi.org/10.1109/ICPM.2019.00028

[5] W. M. P. van der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. van Dongen, A. K. Alves de Medeiros, M. Song, and H. M. W. Verbeek. 2006. Business process mining: An industrial application. (2006). https://doi.org/10.1016/j.is.2006.05.003