33RD CONGRESS
OF THE INTERNATIONAL COUNCIL
OF THE AERONAUTICAL SCIENCES
STOCKHOLM, SWEDEN, 4-9 SEPTEMBER, 2022

ICAS 2022 SWEDEN

REAL-TIME IDENTIFICATION OF USER ACTIONS IN COMPUTER-AIDED MANUFACTURING USING MACHINE LEARNING AND COMPUTER VISION

Bernd Peukert[1], Jakob Vesterberg[1] & Jesper Birberg[2]

[1]Kungliga Tekniska Högskolan
[2]Saab Aeronautics

Abstract

Computer-Aided Manufacturing (CAM) is of high importance when machining complex geometries. Due to its complexity, the CAM process significantly contributes to the preparation time of new parts, and the final part quality depends on the individual programmer's experience. This paper introduces the groundwork for an expert agent to improve the knowledge transfer of experienced to new programmers and the programming flow of the CAM process in general. The envisioned expert agent runs alongside the CAM software and observes and catalogues user interactions with the CAM programming environment. The goal is to derive in-situ suggestions to comply with the established CAM workflows of experienced users. This initial work relates to monitoring and identifying user actions in real-time using computer vision and machine learning. Within this paper two models are presented, i.e., CNN and YOLOv5. Furthermore, a prefilter in the form of active window detection is used to improve the predictive capabilities of the CNN model. The results show that a simple CNN in combination with proper prefiltering is capable of logging and cataloging the user actions for subsequent querying of relevant information from an external database.

Keywords: CAM, Expert Agent, Computer Vision, Machine Learning

1. Introduction

Nowadays, an airplane consists of several thousand digitally designed parts and requires complex machining instructions. The digital processes of designing new parts and preparing them for manufacturing are Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM). Especially in the aerospace industry, the CAM process consumes a large part of the realization time. This is due to the high demands on aerospace parts with their low margin of errors and tight tolerances. Industry experts report training periods of years until the CAM programmers reach the required level of expertise and skills to work both efficiently and reliably within the established workflows. The complexity involved results in the CAM process being a bottleneck during large projects such as designing a new aircraft.

The CAM process translates digital design data to manufacturing instructions understood by Computer-numerically Controlled (CNC) machine tools on the shop floor. The resulting quality of a manufactured part, however, depends on several individual factors chosen by the CAM programmer: the selected tools, the machining operation, the machining parameters, the machining strategy, the machine tool capabilities, and the fixture. Besides, also the skill of the machine tool operator significantly affects the manufacturing outcome. Unfortunately, there is almost no reverse information flow from the shop floor to the CAM programming department in the current industry besides reporting obvious failures after the quality assurance. Often operators manually adjust the machining parameters directly on the shop-floor without notifying the CAM programmers, further preventing learning and improvements.

1.1 Scope

The research in this paper aims at understanding the CAM programmers' intentions by recording a live feed of the computer screen, which is further processed by models of different architectures which perform image classification and recognition, see Figure 1.

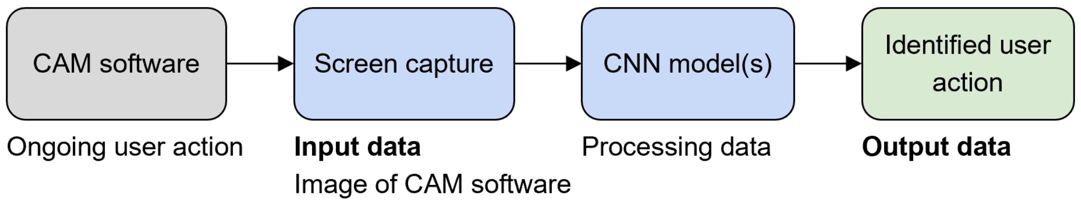


Figure 1 – Process flow from start to finish. An ongoing user action is captures as an image in the first blue square and processed in the second. The gray square is the true user action within the CAM software. The blue squares are parts of the expert agent, and the green square represents the identified user action.

Within this paper, two possible solutions are explored, i.e., Convolutional Neural Network (CNN) models for image recognition and You Only Look Once (YOLOv5) which utilizes object detection. The first requirement on both models and their architectures is to achieve an inference time that allows for real-time estimations of user actions. The second requirement is to achieve sufficiently high accuracy on the output of the estimates, whereas both requirements are necessary to allow for reliable usage in industrial applications.

This project's scope was limited to two commercially available CAM solutions, namely CATIA v5 by the company Dassault Systèmes, Vélizy-Villacoublay, France, and Mastercam by CNC Software LLC, Tolland, Connecticut, United States of America. Figure 2 displays an example screenshot of the the CATIA v5 GUI as an example of the CAM programmer's experience. The screenshot is exemplary for CAM programs, which share similar approaches when connecting CAD design features to machining operations via pop-up operation controls and respective widgets.

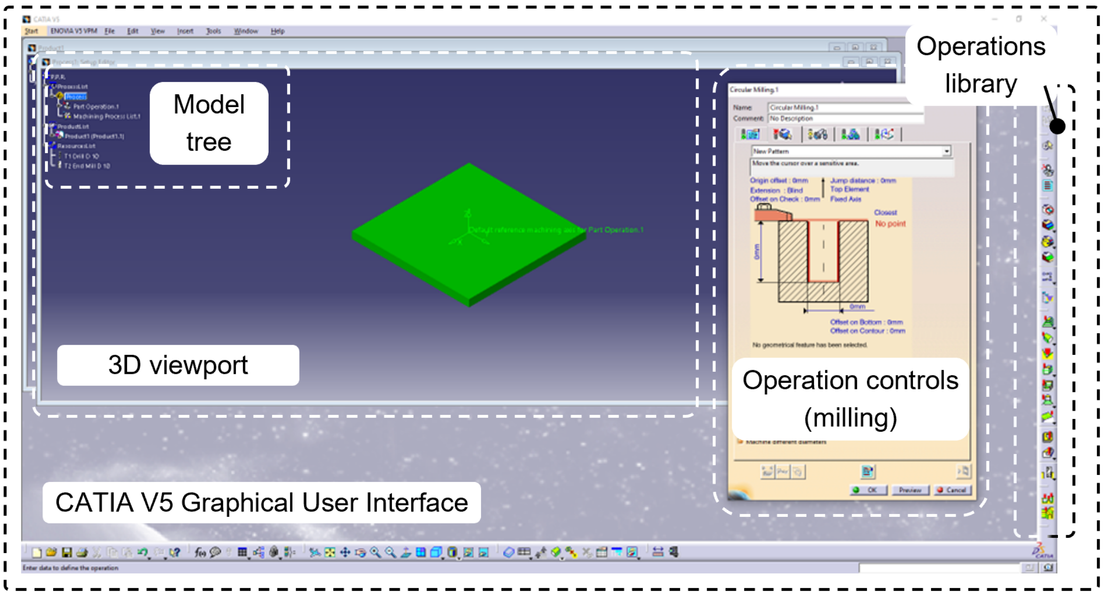

Model tree
3D viewport
CATIA V5 Graphical User Interface
Operations library
Operation controls (milling)

Figure 2 – Example image of the CATIA v5 program interface. The model tree shows the contents of the CAM program. The 3D viewport displays the CAD part. By clicking on the icons in the operations library, the programmer brings up the operation controls, in this case 'Circular Milling'.

1.2 Problem statement

There are several tabs and sub-menus in the active window for an operation, and all of these sub-menus need to be correctly identified to classify the current action and derive suggestions to the CAM programmer. In addition, toolbars and other elements of the program change as the user works with different machining operations. These toolbars are seen at the right border of Figure 2. In contrast to the object detection problem of finding and identifying the active window or current toolbar, looking at all of the elements at once, or a subset of the elements, is an image recognition problem.

Speaking generally, and not explicitly relating to CAM programming, using ML and Artificial Intelligence (AI) in guiding employees is a promising approach. A fully developed expert agent could provide new employees with support and knowledge which require advanced employees' months or years to acquire. Alleviating the need of support from senior employees to younger employees during high load projects will be instrumental in increasing efficiency and keeping costs low.

The described Machine Learning (ML) approach was favored over application specific Application Programming Interfaces (APIs) due to the possibility to incorporate a new CAM program into a working ML model quickly. A switch of the software would hence only require gathering a new data set for that new program instead of adapting the code to a possibly vastly different API with varying capabilities. Hence, in the case of an API-based communication to the CAM backend, the solution would be only tailor-fitted to that specific CAM solution and almost the entirety of the user prediction would have to be rewritten.

2. Background

At the core of Industrie 4.0 is the digitalization of every aspect of manufacturing. Computer vision is predicted to be an elementary part of the fourth industrial revolution [1]. Hereby, expert systems, i.e. software agents, play an important role in the decision making process. They are capable of intelligent problem solving and use declarative knowledge [2]. Zhao et al. [3] applied cooperative agents for the process planning. Later, Feng et al. [4] researched the improvement of the process modeling and knowledge share between different manufacturing applications. Zakarian applied neural networks in an expert system to support CAD [5]. Another research track is the monitoring of user actions to model user behavior and predict recommendations based on previously performed tasks [6]. A prerequisite for this is the accurate capturing of the user interaction, .e.g, by understanding the software via computer vision. This becomes feasible as modern hardware and computer vision models are capable of real-time processing [7].

The detection of pop-up windows and widgets in graphical user interfaces (GUIs) was approached by several researchers. Jaganeshwari and Djodilatchoumy [8] used OpenCV, a free and open source computer vision library, and PyAutoGUI to monitor the activities of the mouse cursor and classify the widgets with a KNN classifier in near real-time. Goyal et al. [9] performed Hierarchical Layout Analysis on screenshots to classify images, icons, and text in mobile applications with high accuracy. Cheng et al. [10] utilize a You Only Look Once (YOLO) model to identify bounding boxes and recognize significant parts of GUI layouts. Radzikowski et al. [11] benchmark several machine learning algorithms for widget detection using a pre-filter in the form of Canny edge detection [12] and post-process the detected objects with an Optical Character Recognition (OCR) algorithm. Chen et al. [13] gives a comprehensive study of similar approaches and approaches without edge detection.

2.1 Convolutional Neural Networks (CNN) and Machine Learning (ML)

Object detection and recognition is a common application of Convolutional Neural Networks (CNN) [14]. CNNs are a deep learning model inspired by the animal visual cortex. They are best suited for processing data such as images, which are stored in a grid-like pattern [15]. CNNs consist of three types of layers: convolution, pooling and fully connected layers. The main benefit of a CNN is that it automatically identifies relevant features without human supervision [16]. Three key benefits of a CNN according to Goodfellow et al. [17] are equivalent representations, sparse interactions, and parameter sharing.

This results in an small number of parameters, which simplifies the training process. As a result, it speeds up the network and makes real-time usage possible. The output of each layer is fed into the next layer, which allows for extracted features to become hierarchically and progressively more complex as one travels through the network [15]. Figure 3 shows an example of a CNN architecture. The input of each layer has three dimensions: height, width and depth, denoted as $m \cdot m \cdot r$ [16]. The height is the same as the width, as square matrices are used for mathematical operations. The depth in an RGB image is equal to three, representing the three channels of red, green and blue. A grayscale input image can thus be described by a square matrix, where each pixel is described by a decimal value in the matrix, which ranges from zero to one. The value of one would be a fully white pixel, whereas zero would correspond to a completely dark pixel.

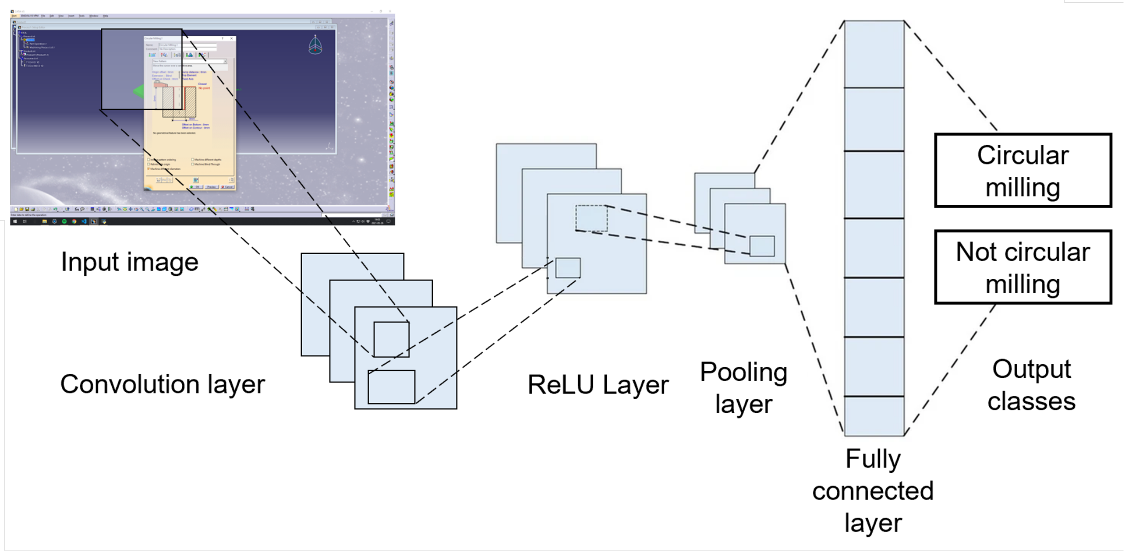


Figure 3 – An example of a CNN architecture adapted from Alzubaidi et al. [16].

The convolutional layer performs convolution operations on the input. The kernel, or filter, is a grid of discrete numbers where each value is called a kernel weight. These weights are adjusted each training epoch, thereby allowing the kernel to learn to extract significant features. The hyperparameters include the filter size F and stride S [4]. Figure 4 shows the calculations within a convolutional layer with an input picture averaged around zero and normalized to a maximum of three as an example.

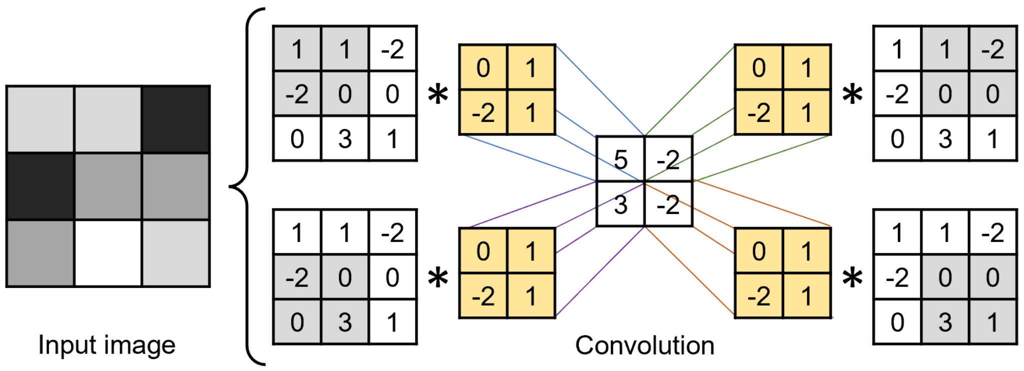


Figure 4 – Calculations performed in each convolutional layer. The input matrices are the outer matrices outlined in gray. The kernel, or filter, is depicted in yellow. The feature map is the matrix in the center. In this example, the filter size is $F = 2x2$, and the stride $S = 1$.

The filter slides over the input, moving the number of pixels equal to S. The dot product between the input image and kernel is determined and its value is then stored in a single cell of the output, called a feature map or activation map. In a CNN model, Several convolutional operations with different filters are performed on the input. The resulting feature maps are then put together as a final output of

the convolutional layer. The pooling layer is a down sampling operation, reducing the dimensionality and is typically placed after the convolutional layer. It shrinks large-size feature maps into smaller feature maps, reducing the dimensionality while attempting to maintain dominant information [2]. One of the most common types of pooling is max pooling, which selects the maximum value of the current window $(2x2)$. Some information may be lost during the pooling operation, which potentially reduce the overall CNN as a result. The benefits of the pooling operation, however, outweigh the detriments, which is why it is still utilized. The benefits are a smaller set of features, which shorten the computational time of the entire network, and the reduced risk of overfitting.
An important factor, which occurs after each layer, is the activation function. It determines whether to fire a neuron or not, depending on the output of the previous layer. There are different types of activation functions, whereas one of the most common for CNNs is the Rectified Linear Unit (ReLU) in the form of $f(u) = \max(0,x)$.
In the ReLU, the input is converted into positive numbers with the benefit of a lower computational load, as compared with other activation functions, e.g., the Sigmoid function.
The fully connected layer is located at the end of a CNN architecture. It consists of an input layer, multiple hidden layers and an output layer. Every node in one layer is connected to every node in the next and is utilized as the CNN classifier [16].
One of the goals of this work was good prediction accuracy, which in turn signifies a generalization capability of the model [5]. The models are trained on a sample dataset, and proper training must ensure that the the model does not adapt to the noise and the training data itself. The term for such an occurrence is overfitting, see Figure 5 for a visualization. Overfitted models perform poorly on new or unseen data, which is the input during practical use.

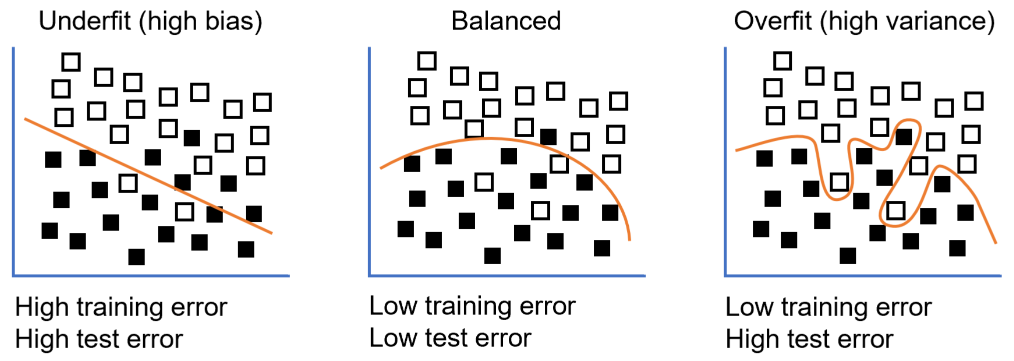


Figure 5 – Example of underfit, overfit, and balanced models.

Various concepts are used to help regularization, thereby avoiding overfitting [18]. Two techniques are utilized in this paper, i.e., dropout layers [19] and data augmentation [20]. Dropout is the action of randomly dropping certain neurons during each training epoch, which forces the model to learn different independent features. Data augmentation increases the amount of training data by creating duplicates of input images with, e.g., blurring, slight translation or rotation or by removing colors.
Loss functions, i.e., minimize the error (the difference between the actual and predicted output), are at the core of all supervised learning algorithms. The network parameters should update for all training epochs, while the network searches for the locally optimized answer in order to minimize the error [2]. The learning algorithm utilized in this paper is called Adaptive Moment Estimation (Adam). It represents one of the latest trends in deep learning optimization, which is a combination of RMSprop and Stochastic Gradient Descent with momentum [6]. It is less computationally demanding and more memory efficient when compared to similar methods.
There are several different CNN architectures and model architecture selection is vital for improving the prediction performance. Generally, an increased depth enhances generalization of the model but also increases the risk of overfitting. The first architecture of notoriety is AlexNet, which has had considerable significance for recent CNN generations, since it employed various techniques which reduces the risk of overfitting. Visual Geometry Group (VGG) proposed the idea of utilizing a number

of smaller filters ($3x3$), by experimentally showing that the parallel assignment of small filters could produce the same result as large-size filters ($7x7$ and $5x5$) [16]. Smaller filters also decrease the number of parameters, which reduces the computational complexity. The Residual Network (ResNet) was the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015. The objective of the architecture is to design an ultra-deep network without the vanishing gradient issue, weights approach zero as one travels through the network which in turn results in mathematical errors (division by zero is not an eligible operation). Table 1 gives a short overview of the discussed models.

Model	Main finding	Depth	Error rate	Input size	Year
AlexNet	Utilizes dropout and ReLU	8	16.4	$227 \times 227 \times 3$	2012
VGG	Increased depth, small filter size	16, 19	7.3	$224 \times 224 \times 3$	2014
ResNet	Robust against overfitting	152	3.6	$224 \times 224 \times 3$	2016

Table 1 – Short table of different network architectures adapted from [16].

This paper utilizes the idea of these networks when attempting to construct a network suitable for the task of real-time classification of CAM operations.

2.2 You Only Look Once object detection

One issue with using CNNs for this type of problem is that the output is a classification of the image. The benefits of locating the viewport, tool panel and other menus in CAM software would allow for more options when designing additional features for the expert agent. Because of this reason, object detection, i.e., is the task of locating objects in an image, is also explored as a solution.
State-of-the-art methods utilize Region-CNN (R-CNN), Fast R-CNN, Faster R-CNN to achieve this task. The problem with these models is the inference time, which is too slow for real-time detection. To solve this problem, You Only Look Once (YOLO) introduces a faster way of object detection suitable for real-time detection [21].

3. Method

3.1 Model selection

The model type determines the format of the data. Within this paper, three models were selected:

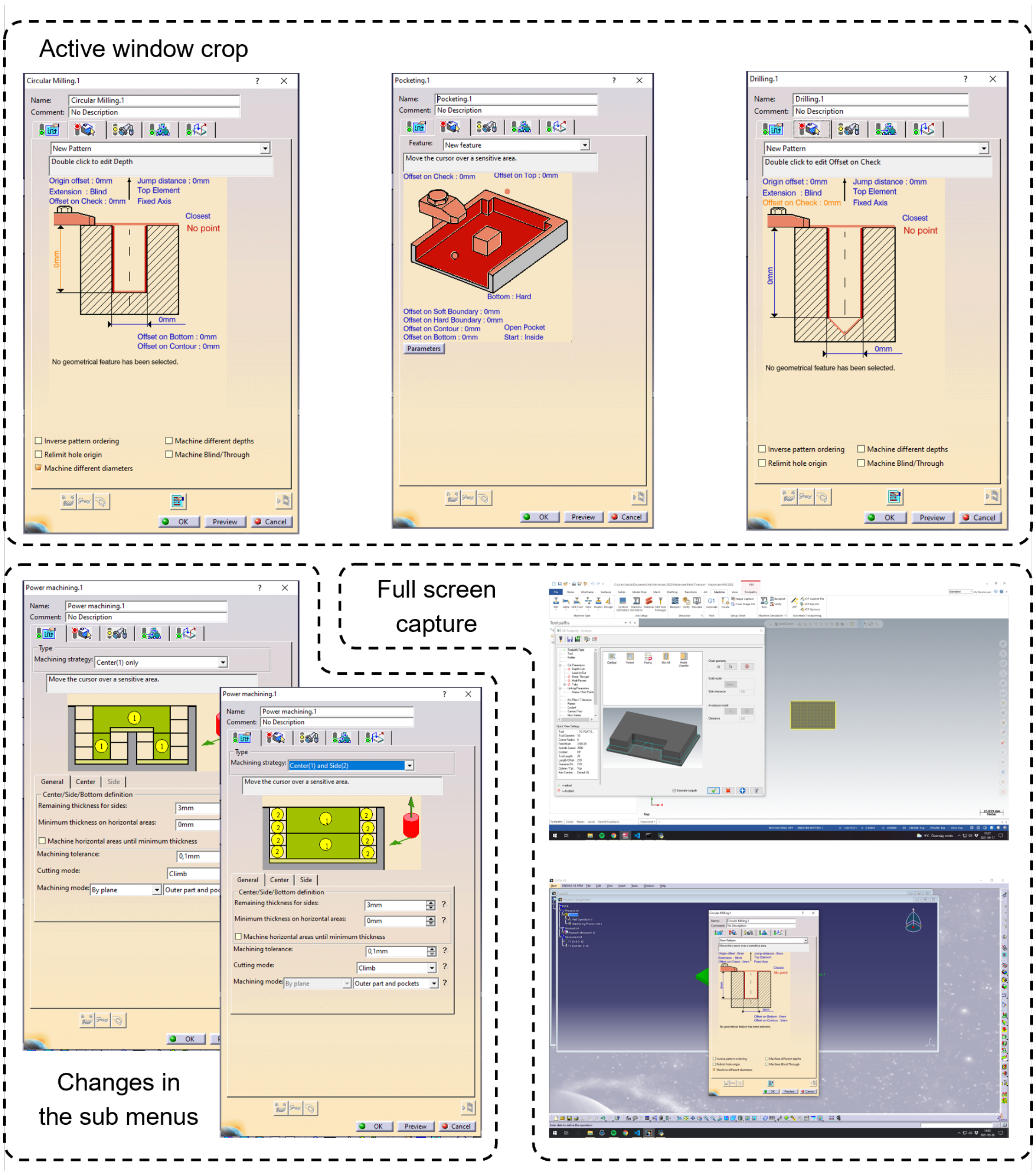
- CNN classifying a full screen capture
- CNN classifying a section of the screen, i.e., the active window
- YOLOv5 object detection

The underlying assumption for the full screen capture CNN is that some information would be lost if only the active window was captured. This information would be located in the menus of the application, which usually change as the user works in the CAM software. The potential advantages of the section model based on the active window is that it is highly likely that most of the relevant information on the user action is located here. Preprocessing the data in this manner is likely to eliminate irrelevant data. A successful implementation of YOLOv5 would allow for a wider array of uses at a later stage, e.g. helping geometry macros, geometry construction, or file handling. The CNN models were built in Python using Tensorflow, whereas the YOLOv5 model utilized was the implementation created by Ultralytics [22] in PyTorch.

3.2 Labeling, data collecting and evaluation metric

The data set was manually created by capturing screenshots from CATIA v5 and Mastercam and then labeling the images by hand. A supporting Python program was made in order to capture data from either the entire screen or the active window. This code is later reused to create the live feed of screen data for predictions using the trained models. In the case of YOLOv5, images were captured by the same Python program and uploaded to https://roboflow.com to create bounding boxes and labels for objects in the images.

Care was taken to create balanced data sets, i.e., data sets in which there is an equal amount of labeled samples belonging to each class. The validation set in Tensorflow is the hold-out set, and therefore equivalent to the testing set. Rather than randomly sampling and splitting one large data set into a training and validation set, dedicated training and validation data sets were created to maintain balance between the classes of the set during training, but also in order to ensure correct classification due to the nature of the data sets.
The validation set was crafted in such a way as to provide realistic scenarios for unseen data. The sizes of the data sets also pose a set of unique problems. In most cases, a data set contains images ranging in the 100'000s, whereas due to practical reasons the three data sets utilized for this paper contain images in the 100s. Given the variety of classes, this leads to a comparatively low number of samples per label. Figure 6 displays some of the training and validation images taken in preparation of the training process. Shown are the input images of the active window crop, full-screen captures, and changes in the sub menus. Table 2 gives an overview on the manually created data sets.



Circular Milling.1
Name: Circular Milling.1
Comment: No Description
New Pattern
Double click to edit Depth
Origin offset : 0mm
Extension : Blind
Offset on Check : 0mm
Jump distance : 0mm
Top Element
Fixed Axis
Closest
No point
Offset on Bottom : 0mm
Offset on Contour : 0mm
No geometrical feature has been selected.
Inverse pattern ordering
Relimit hole origin
Machine different diameters
Machine different depths
Machine Blind/Through

Pocketing.1
Name: Pocketing.1
Comment: No Description
Feature: New feature
Move the cursor over a sensitive area.
Offset on Check : 0mm
Offset on Top : 0mm
Bottom : Hard
Offset on Soft Boundary : 0mm
Offset on Hard Boundary : 0mm
Offset on Contour : 0mm
Offset on Bottom : 0mm
Open Pocket
Start : Inside
Parameters

Drilling.1
Name: Drilling.1
Comment: No Description
New Pattern
Double click to edit Offset on Check
Origin offset : 0mm
Extension : Blind
Offset on Check : 0mm
Jump distance : 0mm
Top Element
Fixed Axis
Closest
No point
No geometrical feature has been selected.
Inverse pattern ordering
Relimit hole origin
Machine different depths
Machine Blind/Through

Power machining.1
Name: Power machining.1
Comment: No Description
Type
Machining strategy: Center(1) only
Move the cursor over a sensitive area.
General | Center | Side
Center/Side/Bottom definition
Remaining thickness for sides: 3mm
Minimum thickness on horizontal areas: 0mm
Machine horizontal areas until minimum thickness
Machining tolerance: 0,1mm
Cutting mode: Climb
Machining mode: By plane | Outer part and pockets

Power machining.1
Name: Power machining.1
Comment: No Description
Type
Machining strategy: Center(1) and Side(2)
Move the cursor over a sensitive area.
General | Center | Side
Center/Side/Bottom definition
Remaining thickness for sides: 3mm
Minimum thickness on horizontal areas: 0mm
Machine horizontal areas until minimum thickness
Machining tolerance: 0,1mm
Cutting mode: Climb
Machining mode: By plane | Outer part and pockets

Figure 6 – Examples of the images in the data sets depicting the full screen capture, cropped images to the active window, and displaying occurring changes due to the selection of different sub menus in the operation controls panels.

Data set	Utilized for	Number of images	Preprocessing	Number of labels
#1	CNN	168	none, full screen capture	20
#2	CNN	114	active window crop	12
#3	YOLOv5	109	Data augmentation	40

Table 2 – Summary of the three generated data sets

It is noted that there is very little to no variation between images depicting a certain user action on the same tab in the active window. For example, capturing a new image of the drilling operation controls panel would look the same, so it would simply have the same effect as duplicating the image in the data set. It is therefore of importance to have an even distribution between the number of images for different labels. It was therefore ensured that all labels in the training data set had an even distribution in the number of images.

The validation data set contains images of previously unseen data. Sub menus were changed, which in turn change the appearance of the active window. Training data for all the different sub menus and options was not provided, as the goal was to create a model which could generalize without providing data for every possible scenario. The features learned by keeping a simple training data set and a validation set with specific examples ensure proper generalization. The validation data set was not balanced in the same way as the training set.

Since the data set was ensured to be balanced, accuracy is an appropriate evaluation metric for the models [23]. Other than accuracy, the loss function and error rate were also analyzed to judge the models' performance.

3.3 Model development, training, and refinement

Due to architecture complexity and depth in contrast to the limited data set, the models would quickly overfit and produce poor results. Therefore, a simple version of the VGG model was implemented, which showed increased accuracy. The filter size was fixed to $F = 3x3$, as it had been shown that $F = 3x3$ filters of increasing depth can be equivalent to filters of larger size. Depth was increased if validation accuracy increased. This was done until the model showed signs of overfitting.

The training process was set up in such a way as to save two versions of the model during run time. The epoch with the best validation accuracy was saved for predictions, while the model of the last epoch was saved for later iterative training. Early stopping was implemented at a later stage, but due to above mentioned constraints models effectively applied early stopping by saving the first instance of its highest validation accuracy. The number of epochs was set to 120, but in the case of need of further training a model could be loaded and trained for an additional 60 epochs at a time. Batch size was set to 128, which basically allowed the entire data set to be fit into one batch. The chosen optimizer was Adam, with a decay learning rate equal to 0.9 and a starting learning rate of 0.01. Since the data set had an even distribution of samples between labels, accuracy was deemed an appropriate evaluation metric. The image size was scaled down to 512x512 pixels, since studies have shown a general trend of increased accuracy at larger image sizes. ReLU was chosen as the activation function for the convolutional layers. Other than network depth and complexity, the dimensionality of the output space in the first convolutional layer was deemed important for increasing the accuracy of the model. Increasing the dimensionality of the output space of this layer increased accuracy of the model.

YOLOv5 was trained in Google Colab using GPU resources, due to the computational intensity of training process. YOLOv5 could be trained from scratch or be implemented with transfer learning, i.e., using models which were previously trained for example on the Common Objects in Context (COCO) data set. The image size for YOLOv5 was tried at both 512x512 and 1280x1280 pixels. YOLOv5s, YOLOv5m and YOLOv5l were trained in order to ascertain differences in accuracy for each model on the problem. A test run of training for 150 epochs was applied in each case. The training time was approximately 4 hours for each test.

4. Results and discussion

4.1 CNN-based prediction of user actions

By running the model and observing the output of the predictions while performing operations in CAM softwares it was determined that the full screen capture models were overly sensitive to translational invariances of the control panels, e.g., While repositioning the active window, the VGG-based model could estimate the action to be circular milling instead of drilling. This was dependent on whether the positioning for the drilling control panel was the same as one of the training samples for circular milling.

Since the data set for the section model contains 12 different labels, a model picking a label at random would have a $1/12$ chance of being correct. Any accuracy above 8.33 % would therefore indicate results better than random guessing. Rather than increasing the data set with the goal of having enough images to bake the translational information into the data set, the feature was eliminated by creating code which captured the part of the screen which contained the active window, i.e., operation controls after pressing the respective button, and used that as input data. This greatly increased the accuracy on test data.

Both AlexNet and ResNet never reached sufficient accuracy on the test data. Due to the small size of the data set, a complex architecture like ResNet quickly overtrained. This made it unsuitable for the task of real-time classification of user actions. In comparison, AlexNet performed slightly better. Reviewing the results of AlexNet and ResNet the optimal architecture would exhibit just enough training depth to learn more features and does not overfit on training data. Therefore, the chosen architecture was a simplified version of the VGG architecture. Figure 8 shows the training and testing plots over the various training epochs.

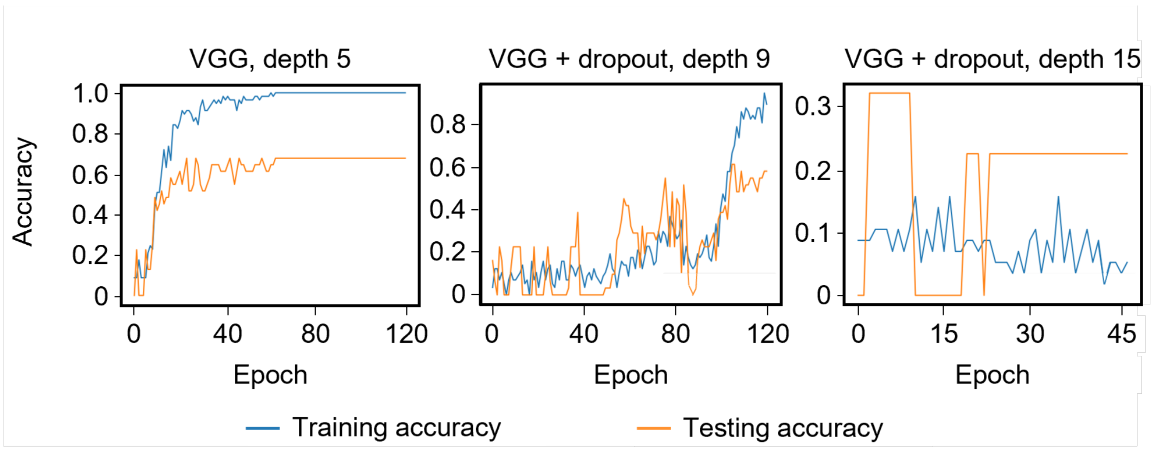


Figure 7 – Accuracy of the training process and the testing of the VGG CNN models.

The network architecture for all VGG-based models consist of several instances of three repeating layers. The first layer is a convolutional layer, with smaller $F = 3 \times 3$ filters and a stride of $S = 1$, the second layer is a pooling layer with a kernel size of $F = 2 \times 2$ and stride $S = 2$. The last layer is a dropout layer. The number of filters of the convolutional layers increase as one travels down the network. The architecture was then extended with more instances of the three-layer-structure if the testing accuracy increased. The resulting network architecture is displayed as a TensorBoard in Figure 8.

Ensemble learning, through average voting, did not increase accuracy when combining three of the classifiers. This result could be due to the setup of the training data set. Randomly sampling the data set on training could create differentiated classifiers which produce better results once in ensemble. It's difficult to predict, however, if such an ensemble would outperform a single classifier trained on a balanced data set.

Dropout layers on both the final fully-connected layer and the max-pooling layers increased accuracy, because they could be utilized in conjunction with a deeper network. The simple VGG wothout dropout layers and a depth of 5 could not reach a validation accuracy above 68 % on the test data.

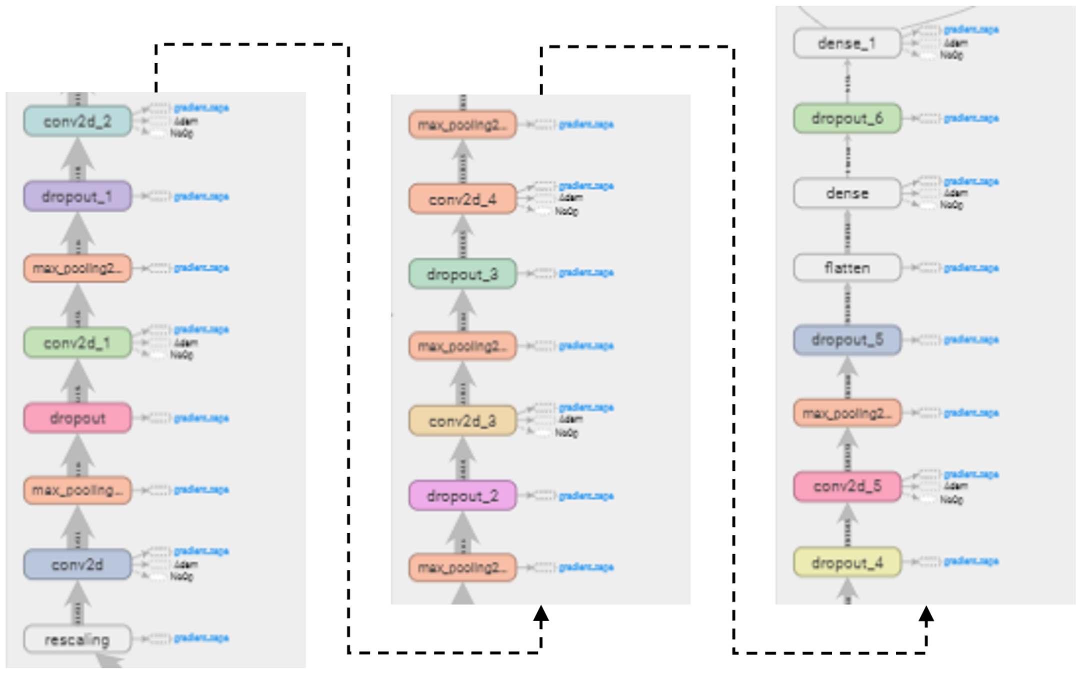


Figure 8 – Network architecture of "VGG-based" visualized by TensorBoard.

This is most likely due to the depth of the network limiting the learning of features at some point. By adding dropout layers at above mentioned positions, the depth of the network could be increased without incurring overfitting. This resulted in a model with the best validation accuracy. On the other hand, adding layers up to a depth of 15 increased the complexity of the model, and performance decreased which lead to higher inference times around 150 ms. Table 3 gives the summary of the performed analyses. Figure 9 shows some results of the real-time prediction of user actions using the CNN-based model architecture within CATIA v5.

Architecture	Input	Accuracy		Depth
		Training	Test	
VGG	screen	15.3	16	19
AlexNet	screen	63.2	16.2	8
ResNet	screen	100	3.05	50
VGG	screen	99.2	69.5	13
VGG	section	100	67.7	5
VGG	section	5.2	22.2	15
VGG Dropout	section	97.3	74.3	9
VGG Dropout	section	98.3	90.3	13
VGG Dropout + Ensemble	section	98.3	90.3	13

Table 3 – Results of the CNN training process

4.2 YOLOv5-based detection of user actions

YoloV5 never achieved good enough results. In fact, the model did not make any predictions for object detection in the tested images. According to the YoloV5 documentation: "Most of the time good results can be obtained with no changes to the models or training settings, provided your data set is sufficiently large and well labeled." [8]. Since the data set is too small, YoloV5 may not be a suitable model for this type of problem. The data set was slightly increased by the usage of data augmentation, i.e., creating images with blur, varying degrees of grayscale. This proved insufficient

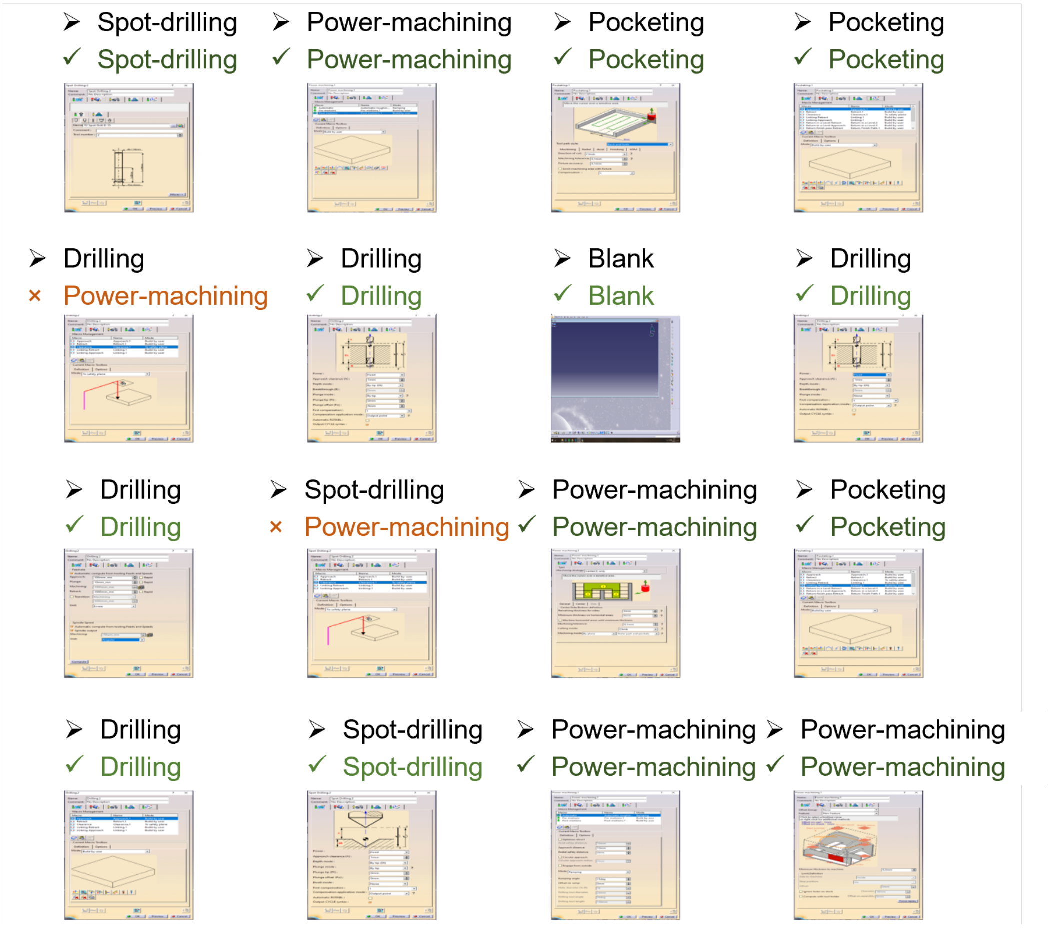


Figure 9 – Real-time predictions of user actions in CATIA v5 with the VGG-based architecture.

at solving the issue, however. It is noted that the data augmentation was not optimal in this case as many of the performed operations did not preserve the manual labels post-augmentation. Figure 10 displays some of the labeled objects and the accuracy versus recall plot.

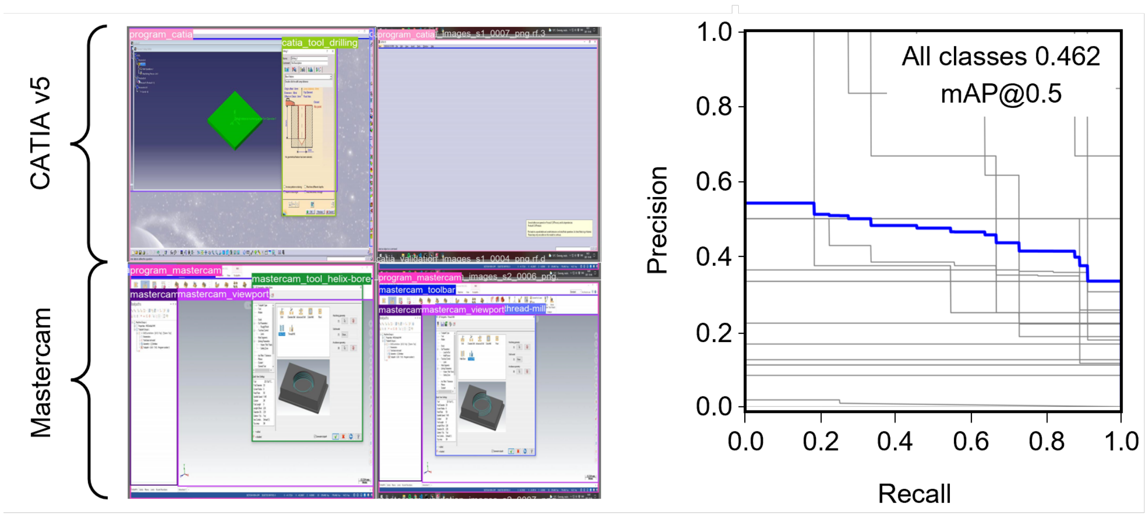


Figure 10 – Input data for the YOLOv5 model. Objects, such as the operation controls, view-port, and taskbar were manually labeled in CATIA v5 and Mastercam. The model fails to recall the manually labeled objects.

5. Conclusion and outlook

A simple CNN with active window detection proved sufficient for the detection of intended user actions withing the researched CAM software. It was found that model complexity is the greatest factor on accuracy while working with limited data sets of this type. There is a balance between model depth and the implementation of dropout layers, where additional depth and accuracy can be reached through the use of dropout layers in the models. Additional max-pooling dropout layers proved beneficial in this case. Data augmentation and ensemble learning using average voting proved less beneficial in increasing accuracy for this type of problem and data set.
Future work will cover the extraction of additional information from CAM software. The input parameters on the operations could be read from external databases so that an expert system can generate informed decisions regarding tool selection in real-time. As for solving the problem of attempting to increase accuracy in small data set with this type of problem, more advanced ensemble techniques could prove beneficial. If a classifier which utilized the history of previous user actions were trained on predicting the next action, this result could be included in such an ensemble. There has also been research indicating that an ensemble consisting of classifiers with different image sizes as inputs could improve accuracy as well.

6. Contact Author Email Address

bpeuke@kth.se

7. Acknowledgement

This work was funded by the Vinnova project 'IntDemo-Flygplan' under the grant 2020-00187. The paper summarizes the results of a Master Thesis by Jakob Vesterberg together with KTH Royal Institute of Technology, SAAB Aeronautics, and Brogren Industries.

8. Copyright Statement

References

[1] Melvyn L. Smith, Lyndon N. Smith, and Mark F. Hansen. The quiet revolution in machine vision - a state-of-the-art survey paper, including historical review, perspectives, and future directions. *Computers in Industry*, 130:103472, 2021.

[2] Takashi Matsuyama. Expert Systems for Image Processing, Analysis, and Recognition: Declarative Knowledge Representation for Computer Vision. *Advances in Electronics and Electron Physics*, 86(C):81–171, 1993.

[3] F. L. Zhao, S. K. Tso, and Paul S.Y. Wu. Cooperative agent modelling approach for process planning. *Computers in Industry*, 41(1):83–97, 2000.

[4] Shaw C. Feng, Keith A. Stouffer, and Kevin K. Jurrens. Manufacturing planning and predictive process model integration using software agents. *Advanced Engineering Informatics*, 19(2):135–142, 2005.

[5] Vaagn L. Zakarian and Mark J. Kaiser. An embedded hybrid neural network and expert system in a computer-aided design system. *Expert Systems with Applications*, 16(2):233–243, 1999.

[6] B. Stephen and L. Petropoulakis. An ambient software monitoring system for unsupervised user modelling. *Expert Systems with Applications*, 28(3):557–567, 2005.

[7] Vijaykumar Janga, Sravan Kumar V., and Vinay Kumar Enugala. Advanced machine learning-based implementation patterns for computer vision and real-time multimedia applications. *Materials Today: Proceedings*, (xxxx), 2020.

[8] K. Jaganeshwari and S. Djodilatchoumy. A Novel approach of GUI Mapping with image based widget detection and classification. *Proceedings of 2021 2nd International Conference on Intelligent Engineering and Management, ICIEM 2021*, pages 342–346, 2021.

[9] Manoj Goyal, Rachit S. Munjal, Sukumar Moharana, Deepak Garg, Debi Prasanna Mohanty, and Siva Prasad Thota. ScreenSeg: On-Device Screenshot Layout Analysis. *Proceedings of the International Joint Conference on Neural Networks*, 2021-July, 2021.

[10] Jing Cheng, Junxian Li, and Jingyi Chen. Research on Recognition Method of Interface Elements Based on Machine Learning. *Proceedings - 2021 International Conference on Intelligent Computing, Automation and Applications, ICAA 2021*, pages 233–236, 2021.

[11] Kacper Radzikowski, Karol Chęciński, Mateusz Forc, Łukasz Lepak, Michał Jabłoński, Wiktor Kuśmirek, Bartłomiej Twardowski, Paweł Wawrzyński, and Robert M. Nowak. Widget detection on screenshots using computer vision and machine learning algorithms. In *Proceedings SPIE 11176, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments*, page 28, 2019.

[12] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[13] Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. Object detection for graphical user interface: Old fashioned or deep learning or a combination? *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1202–1214, 2020.

[14] Aseem Patil and Milind Rane. Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition. *Smart Innovation, Systems and Technologies*, 195(December):21–30, 2021.

[15] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, aug 2018.

[16] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. *Review of deep learning: concepts, CNN architectures, challenges, applications, future directions*, volume 8. Springer International Publishing, 2021.

[17] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. 2016.

[18] Nitish Srivastava Salakhutdinov, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929—-1958, 2014.

[19] Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural Networks*, 71:1–10, 2015.

[20] Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 2019.

[21] Raian Rahman, Zadid Bin Azad, and Md Bakhtiar Hasan. Densely-Populated Traffic Detection Using YOLOv5 and Non-maximum Suppression Ensembling. *Lecture Notes on Data Engineering and Communications Technologies*, 95(September):567–578, 2022.

[22] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu , changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements, October 2020.

[23] Hossin M and Sulaiman M.N. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining Knowledge Management Process*, 5(2):01–11, 2015.