ClaVis: An Interactive Visual Comparison System for Classifiers

Frank Heyen
VISUS, University of Stuttgart
Stuttgart, BW, Germany
frank.heyen@visus.uni-stuttgart.de

Tanja Munz
VISUS, University of Stuttgart
Stuttgart, BW, Germany
tanja.munz@visus.uni-stuttgart.de

Michael Neumann
IMS, University of Stuttgart
Stuttgart, BW, Germany
michael.neumann@ims.uni-stuttgart.de

Daniel Ortega
IMS, University of Stuttgart
Stuttgart, BW, Germany
daniel.ortega@ims.uni-stuttgart.de

Ngoc Thang Vu
IMS, University of Stuttgart
Stuttgart, BW, Germany
thangvu@ims.uni-stuttgart.de

Daniel Weiskopf
VISUS, University of Stuttgart
Stuttgart, BW, Germany
daniel.weiskopf@visus.uni-stuttgart.de

Michael Sedlmair
VISUS, University of Stuttgart
Stuttgart, BW, Germany
michael.sedlmair@visus.uni-stuttgart.de

ABSTRACT

We propose ClaVis, a visual analytics system for comparative analysis of classification models. ClaVis allows users to visually compare the performance and behavior of tens to hundreds of classifiers trained with different hyperparameter configurations. Our approach is plugin-based and classifier-agnostic and allows users to add their own datasets and classifier implementations. It provides multiple visualizations, including a multivariate ranking, a similarity map, a scatterplot that reveals correlations between parameters and scores, and a training history chart. We demonstrate the effectivity of our approach in multiple case studies for training classification models in the domain of natural language processing.

CCS CONCEPTS

• **Human-centered computing** → **Visual analytics**; Information visualization; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Visualization, visual analytics, classifier comparison, machine learning.

ACM Reference Format:
Frank Heyen, Tanja Munz, Michael Neumann, Daniel Ortega, Ngoc Thang Vu, Daniel Weiskopf, and Michael Sedlmair. 2020. ClaVis: An Interactive Visual Comparison System for Classifiers. In *International Conference on Advanced Visual Interfaces (AVI '20), September 28-October 2, 2020, Salerno, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3399715.3399814

AVI '20, September 28-October 2, 2020, Salerno, Italy

ACM ISBN 978-1-4503-7535-1/20/09…$15.00
https://doi.org/10.1145/3399715.3399814

1 INTRODUCTION

Machine learning has gained much attention over the last few years. It reaches out into almost all areas of science, technology, and society and has become the state of the art in research areas such as computer vision and natural language processing.

Building and understanding machine learning and especially deep learning models has remained a major challenge though. One promising approach leverages interactive visualization that allows machine learning developers to better understand the different facets in the development process [11, 29].

In this work, we focus on the challenge of comparing different classification models and hyperparameterizations thereof, although our approach could be extended to other domains, such as regression or sequence-to-sequence transformation. We use the word hyperparameter to encompass every parameter that influences the structure or training of a classifier, including data preprocessing options, neural network architecture, and learning rate.

Classification is a task with many important applications, such as autonomous driving [13, 25] and medical diagnosis [43]. However, finding the best classifier strategy, architecture [18, 21], and parameters for a given problem is a nontrivial task. Developers have to consider how to preprocess the dataset and which features to extract from it and then carefully choose the kind of classifier and appropriate values for its hyperparameters. They often do this through trial-and-error, by manually or automatically training multiple models with different parameter configurations and then comparing scores such as accuracy or precision and recall. In many cases a good prediction is not enough, and the classifier has to also fulfill other requirements or constraints such as low time and resource costs for training and prediction.

The analysis of all those scores is tedious and limits the developer's insight. For example, it might be hard to quickly grasp trends

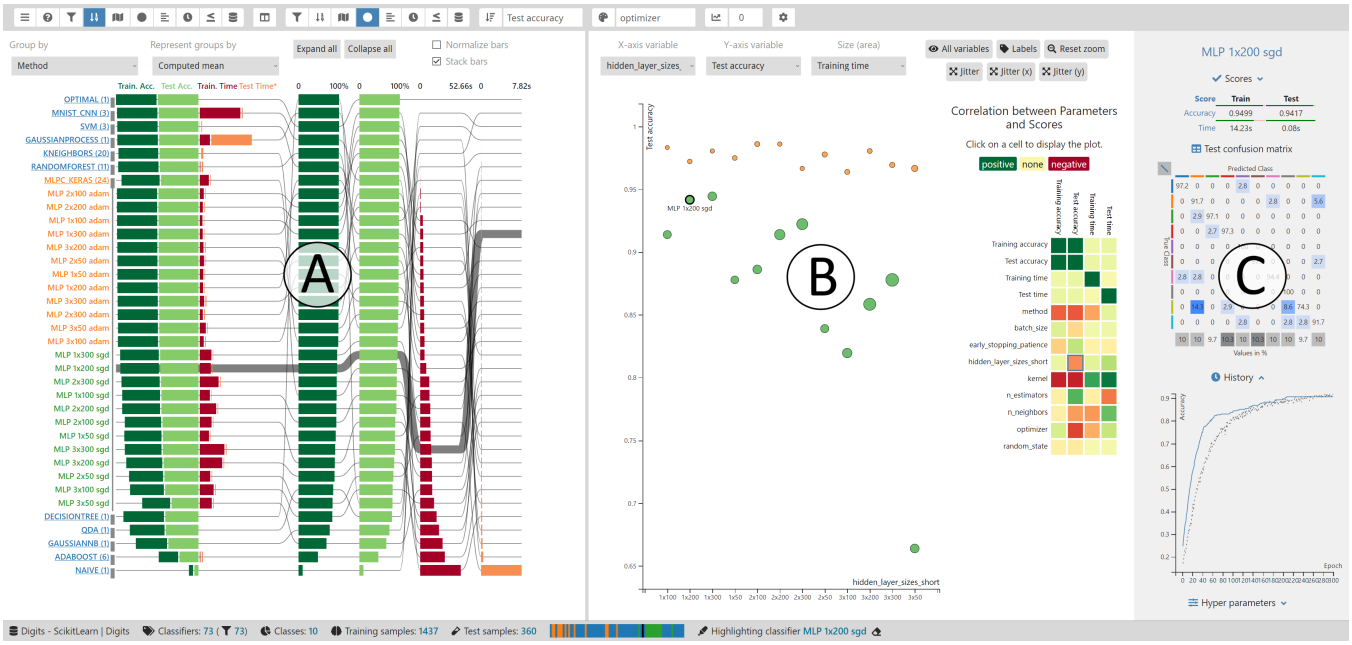

Figure 1: Screenshot of ClaVis with a ranking of classifiers (A) and a scatterplot showing characteristic quantities (B). A sidebar (C) displays details of the currently highlighted classifier. In this example, we are interested in how size and optimizer choices affect the performance of Multilayer Perceptrons (MLPs). With coloring by optimizer, the ranking reveals that the Adam optimizer (orange) outperforms SGD (green) in accuracy and training time. In the scatterplot, we choose *layer sizes* and *test accuracy* as X and Y axes and encode *training time* as area. We see that MLPs trained using Adam are higher up (better) and smaller (trained faster).

or trade-off relations, where one classifier is better in one score but worse in another, by only looking at numbers. There is also no insight into how different hyperparameter values contribute to the result, or why certain values do not produce a well-performing classifier. Related work mostly focuses on visualizing the inner workings of classifiers, to better understand them, and does not help developers make an informed parameter choice by providing a visual way to inspect the influence of different parameterizations.

To mitigate these problems, we propose to leverage visual parameter space analysis [34] for classifier comparison. We first sample different classifier parameterizations and train a model for each and then provide developers with interactive visualizations to give them an overview of these models (usually in the range of tens to hundreds of models). This allows them to better understand similarities, groupings, and patterns such as correlation between parameter values and performance scores. Our visual parameter space analysis approach is model-agnostic and thereby able to support arbitrary classification algorithms instead of being limited to, for example, convolutional neural networks or decision trees.

In this work, we make the following contributions: We compile a set of high-level tasks that are common in classifier design and propose design considerations to address these tasks. Based on our considerations, we implement an interactive visualization system called ClaVis that instantiates the idea of visual parameter space analysis for classifier comparison. ClaVis provides support for both the sampling and analysis process. It was developed by a team of three visualization experts and three machine learning experts in an iterative design process inspired by design study methodology [35] and the nested model framework [26, 28].

We evaluate ClaVis with multiple case studies using real datasets and classifiers from the domain of natural language processing. The results show that our approach effectively supports important tasks such as finding good classifiers, detecting problems in their prediction or training behavior, and analyzing the influence of hyperparameters on the prediction behavior.

In our supplementary material[1], we provide the full source code of ClaVis and a video that shows how users interact with it.

2 RELATED WORK

We review related work on classifier analysis, parameter analysis, and user-guided model selection.

2.1 Classifier Analysis

Most existing approaches for classifier analysis focus on single classifiers or support the comparison of only a few at once.

TensorBoard, a part of *TensorFlow* [1], displays a node-link diagram of a model's structure [42]. It also allows users to analyze training logs of classifiers by displaying metric line charts for all epochs to reveal already during the training if and how well the

[1] https://github.com/fheyen/ClaVis/

classifier learns. We implemented an aggregated version of history line charts to facilitate analyzing and comparing the training behavior of multiple classifiers or groups of them.

Some previous work supports the creation or analysis of single classifiers. With *EnsembleMatrix* [38], users can interactively combine classifiers via ensemble learning and see their confusion matrices alongside the matrix of the result. *RuleMatrix* [27] helps analysts better understand the behavior of a classifier by treating it as a black box and approximating its prediction behavior with a rule list, which is easier to explain and visualize. *ModelTracker* [4] shows the predictions of a binary classifier as colored squares that are positioned by probability to reveal the distribution of probabilities and errors. An approach to *inter-active learning* [16] includes the visualization of weak classifiers to help understand and improve combined classifier performance. While these approaches allow the user to create and analyze single classifiers, they are not suited for the comparison of multiple classifiers.

Certain approaches only allow comparing a few classifiers or do not work for a high number of classes. *Squares* [31] supports the analysis and comparison of arbitrary multi-class classifiers. It shows the distribution of a classifier's predicted probabilities in one binned chart per class, indicating the confidence of predictions. The visualization can display further details by drawing small squares for all data samples and coloring them by their correct class. *Classilist* [19] shows the distribution of predicted probabilities for each class separately and all classes combined. It can compare multiple classifiers by showing juxtaposed visualizations for each of them, but does not scale to more than a few classifiers. Alsallakh et al. [3] use a confusion wheel to visualize the entries of one binary confusion matrix per class. Each class has its own sector where it shows the entries of that matrix as stacked bars for different probability bins. All sectors are connected by curves that encode the class confusion in their width. The visualization may also be used to compare two classifiers by showing what they have in common and where one is better than the other, but cannot compare many classifiers at once. Our work goes beyond these existing approaches by offering a model-agnostic analysis framework for comparing up to several hundreds of classifiers and their parameterizations.

2.2 Parameter Space Analysis

The main idea of visual parameter space analysis is sampling different inputs, computing the respective outputs, and using visual analytics to understand the relation between both [34]. This approach is generic and fits well to analyzing machine learning models.

On the *TensorFlow Playground* website [37], users can build and train small neural networks. It visualizes the decision boundaries and weights for all neurons and is useful for learning and teaching about the general concept of neural networks and the importance of hyperparameters and feature selection. However, it does not scale to larger and more complex types of networks and does not support strategies other than neural networks, such as Support-vector Machines (SVMs) or tree-based classifiers.

To compare classifier behavior, Japkowicz et al. [2, 17] perform a projection of classification results via dimensionality reduction (DR). They do this by flattening confusion matrices or other scores into vectors that are then projected to a two-dimensional subspace. We extend this approach to the predicted class probabilities that contain more information on the classifier's behavior. Their work also inspired us to include the optimal classifier as a reference that shows the location of an ideal prediction in the projection. This allows us to visually compare classifiers to each other and the optimum in a way that is more informative than a simple ranking.

VisCoDeR [11] shows the different behaviors of DR algorithms and how they distort the original distances between data points. Juxtaposed scatterplots display the result of each algorithm for comparison. In a meta visualization, VisCoDeR compares the influence of DR algorithm and parameter choices by projecting the dimensionality-reduced data of all algorithms from a space of projections down to a two-dimensional image. This is similar to our similarity map visualization, where we project the scores or predictions of multiple classifiers in order to show similarities.

2.3 User-Guided Model Selection

Automated Machine Learning (AutoML) systems [5, 7, 18, 20, 39], simplify the search for appropriate hyperparameter values via automatic sampling of the parameter space. This process can be made more efficient by incorporating user knowledge about a dataset or problem. Gil et al. [14] propose a human-guided machine learning approach, where users interactively collaborate with an AutoML system, and compile user tasks and requirements for this approach. Users distrust AutoML and tend to use as many resources for hyperparameter search as they can, making the process less efficient. *ATMSeer* [41] addresses this by allowing users to compare models on three levels of detail and modify the search space in real time according to their findings. In the ranking visualization, it is not possible to compare different properties. The focus is on comparing different models, and a compact overview for one specific classifier is not available.

In addition to pipeline creation and model selection, Santos et al. [33] also support the user in augmenting a dataset by searching for matching data online and incorporating it. While they include a ranking that can be sorted according to different metrics, the values are only displayed as numbers. Our ranking visualization provides a visual comparison between values. Their approach for comparison is also limited to comparing one model to all others and does not allow users to compare different sets of models that are, for example, grouped by classification algorithm.

Cashman et al. [10] propose an exploratory model analysis workflow that guides the user from the exploration of the dataset and possible problem types through model generation to model exploration and selection. They focus on supporting the user in producing good models rather than gaining insights. Only a simple ranking and confusion matrices are available for classifier comparison, no further comparative visualizations.

In contrast to our work, the above model selection approaches do not allow expert users to implement their own classifiers, but instead use AutoML libraries that are often limited to certain algorithms. Developers cannot simply incorporate their existing classifiers' code in order to compare them or use them as a baseline. None of these approaches provides insight into the training history. They support multiple machine learning tasks and are not specifically designed for classifier comparison, so some of them do not

include typical visualizations such as confusion matrices. As our approach currently focuses on classification, we provide confusion matrices for single and multiple classifiers. Our filtering is more flexible, making it easier to select interesting subsets of models for the analysis, for example by filtering on scores and hyperparameters. In addition to scores that measure the quality of predictions, we support an analysis of the required resources that are reflected in the time that classifiers need for training and prediction.

3 PROBLEM CHARACTERIZATION

Developing well-performing classifiers is a nontrivial task, since there are many possible hyperparameter choices and the influence of parameter values or combinations thereof is unclear. During multiple meetings and a pre-study with machine learning students, our team of visualization and machine learning experts found these high-level tasks to be important for classifier design and analysis:

T-optimal. Finding good classifiers by choosing appropriate hyperparameters
T-sensitivity. Understanding parameter influences on scores
T-similarity. Comparing classifiers' prediction behaviors to each other and to an optimum
T-training. Understanding how hyperparameters influence the training behavior
T-confusion. Detecting class-specific errors and difficulties
T-separability. Analyzing the dataset for class overlaps and outliers

To support every type of classifier and since models can be huge, we focus on analyzing and visualizing the following data:

D-parameters. The chosen hyperparameter values, including data preprocessing options and the classifier algorithm and architecture
D-predictions. The resulting predictions for training and test set, i.e. the predicted class labels and class probabilities for all samples
D-scores. The classifiers' scores such as accuracy, training time, and confusion matrix for both training and test set
D-history. The training history for classifiers that are trained in iterations or epochs, such as neural networks
D-dataset. The data samples with their features and class labels

4 DESIGN

In this section, we will first provide an overview of the workflow and then explain how we address each of the tasks from Section 3.

4.1 General Overview

The high-level steps of the user workflow in classifier development are (1) choosing hyperparameters and training multiple classifiers, (2) analyzing the resulting data such as scores and predictions, (3) if the result is not satisfactory, repeat from (1) and make use of the newly gained insight to improve or extend the search space. Our approach reflects those steps and consists of two parts: a training framework and a visual analytics frontend.

The detailed workflow consists of the following steps: (1) The developers implement plugins for their datasets and classifiers. (2) They then create a batch job that contains all hyperparameter configurations they want to train. (3) Our training system loads and preprocesses the data and samples models from the hyperparameter space. (4) After all models have been trained, the developers analyze the resulting data in our interactive visual analytics frontend.

Our plugin system is based on Python to allow developers to reuse existing code and to have access to common machine learning libraries. This system brings a lot of freedom: Different network structures for example can be either represented in separate plugins or controlled via hyperparameters, depending on the developer's choice. Together with our implementation, we also provide a set of various example datasets and classifiers, that allow you to quickly try out our system or to get a baseline on new datasets.

ClaVis' frontend consists of multiple coordinated views that allow for different perspectives on the classification models, including a ranking that helps to find the best classifier in regard to different scores, a scatterplot that shows correlations between hyperparameters and scores, a similarity map where classifiers with similar predictions are clustered together, and a history line chart that helps to spot problems such as overfitting.

We support general interaction techniques like filtering, sorting, grouping, zooming, and panning. For example, classifiers can be filtered by manual selection, by hyperparameter values, or by selecting the top n classifiers after sorting by an attribute. Users can change color schemes to compensate for color blindness or bright environments. The supplementary material for this paper contains a video that shows these features in more detail.

We now explain how ClaVis supports the different tasks listed in Section 3 through its visual encodings and interactions. To keep things simple, we use small example datasets for illustration. Use cases with large-scale real data will be covered in the next section.

4.2 T-optimal: Finding Well-Performing Classifiers

A common strategy for building a classifier is training multiple models and then selecting the one with the best accuracy, F_1 score, or similar. Other factors, such as the time required for training and prediction, which has an impact on resource needs and response time, might also be relevant and force the developer to make compromises. It is therefore helpful to visualize multiple scores at once, while ordering classifiers by one score at a time, to show *how much* better or worse one is compared to the others. Rankings are an intuitive way to represent items ordered by some attribute, and bar charts are commonly used to compare them. LineUp [22] combines those visualizations into a multivariate ranking.

Our ranking view (Figure 1A) is inspired by LineUp and shows which classifiers are good in which regards, by allowing the user to sort or group them by hyperparameter values or scores. Single score rankings allow users to find good classifiers regarding that score; a line connects each classifier's bars in all rankings.

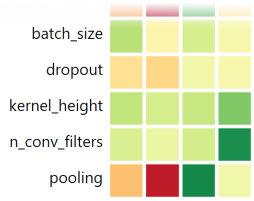

Figure 2: The correlation matrix (cut, see also Figure 1) provides an overview of hyperparameter and score relationships by encoding correlation as red (negative), green (positive), and yellow (no correlation). Each row represents a hyperparameter and the columns correspond to training accuracy, test accuracy, training time, and test time (left to right). In this case, we see that the chosen pooling method has a significant impact on the test accuracy (red square).

4.3 T-sensitivity: Analyzing the Influence of Hyperparameters on Scores

Hyperparameters influence the training of a model in complex ways, as multiple parameters have to be tuned together and work well only when correctly combined. Classifier developers need to understand the influence and stability of parameters in order to know what parameter causes a problem and to be able to choose appropriate values. The correlation between two variables is commonly analyzed in a scatterplot.

We provide a scatterplot view that encodes classifiers as points. The axes correspond to hyperparameters or scores; the latter can also be encoded by the area or color of the points (Figure 1B). This reveals interesting patterns, such as a positive correlation between a parameter and a score until a certain optimum and then a plateau or negative correlation when the parameter's value increases further. When using two scores as axes, the scatterplot can indicate overfitting by revealing classifiers with high training scores but low test scores. Next to the scatterplot, we display a correlation matrix to provide an overview of all possible combinations of hyperparameters and scores (Figures 1 and 2).

4.4 T-similarity: Analyzing the Prediction Behavior

Performance scores, such as accuracy, can be misleading in several ways. One classifier might be better than others in accuracy even when its predicted class probabilities are not very confident, since only the most probable class is predicted each time. This might lead to the developer choosing a classifier with low confidence, that could perform poorly in production. Classifiers that are more similar to the correct predictions should be considered to be better than those that are not. Closeness and clusters of similar items are usually visualized by distance matrices or scatterplots.

We implemented and evaluated a color-coded distance matrix in a pre-study, but did not find it effective for the analysis. Instead, ClaVis provides a similarity map that displays classifiers as dots in a scatterplot. Users can see how similar classifiers are in both their predicted class labels and the raw class probabilities

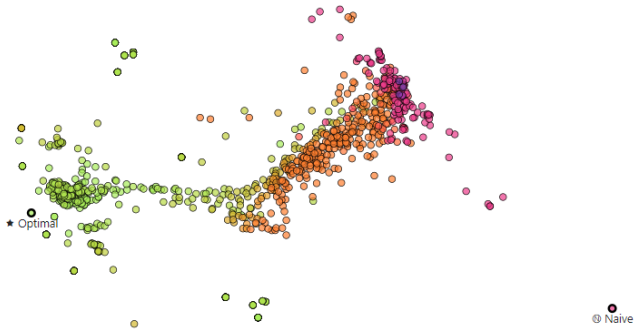

Figure 3: Our similarity map shows that classifiers with similar predictions have a similar performance. Green indicates high, magenta low test accuracy. Classifiers are near others with similar accuracy. Good classifiers are also closer to the correct predictions (*Optimal*) and farther away from the majority class baseline (*Naive*). The classifiers shown here and in Figure 4 are trained on the IRIS dataset [12].

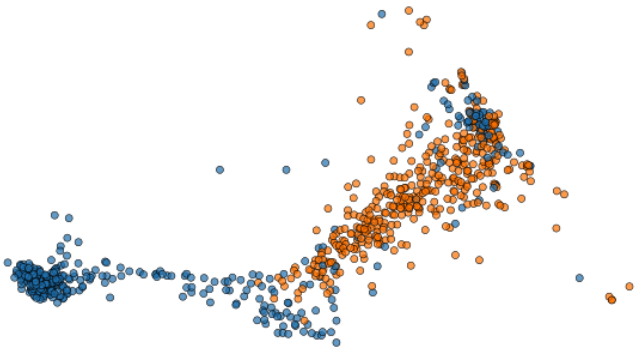

Figure 4: From the classifiers shown in Figure 3, we keep the MLPs and color them by their optimizer (blue: Adam, orange: Stochastic Gradient Descent (SGD)). Classifiers with the same optimizer are generally closer to each other than to those with different ones. This is the only parameter that clearly clusters them in this experiment, indicating a strong influence of optimizers on predictions.

by visually analyzing how close they are to each other. We compute the scatterplot's layout by concatenating the predictions into vectors, as Japkowicz et al. did [2, 17], and projecting them to a two-dimensional image via dimensionality reduction techniques such as PCA [30], MDS [40], t-SNE [23], or UMAP [24], depending on the user's choice. Figures 3 and 4 show that similar classifiers are indeed closer to each other and well-performing ones are generally closer to the optimal classifier, which serves as a reference by always returning the correct predictions.

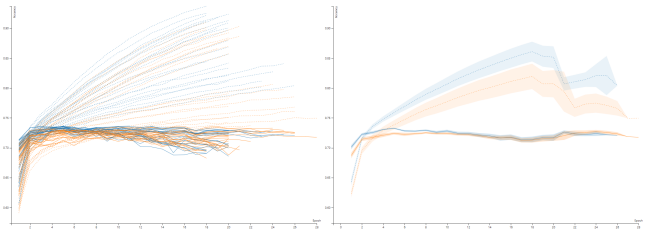

Figure 5: To facilitate the comparison between groups of classifiers in the history view, we provide the option to only show the mean and confidence interval of each group's histories (right) instead of all histories at once (left).

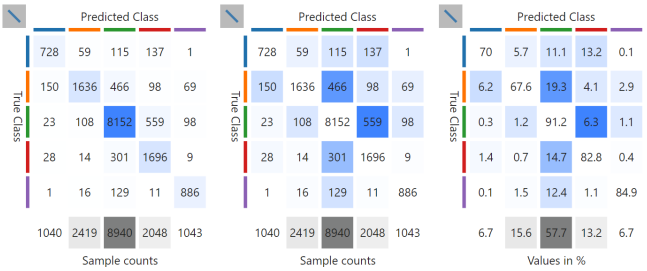

Figure 6: The confusion matrix shows how a classifier labeled the samples of each class. Numbers below the matrix show the size of each class. We support different modes: On the left, all values are colored and the correct predictions (diagonal) dominate the view. In the middle, only errors are colored to make them easier to distinguish. On the right, all numbers are shown in percent of the class size for matrix cells or dataset size for class sizes.

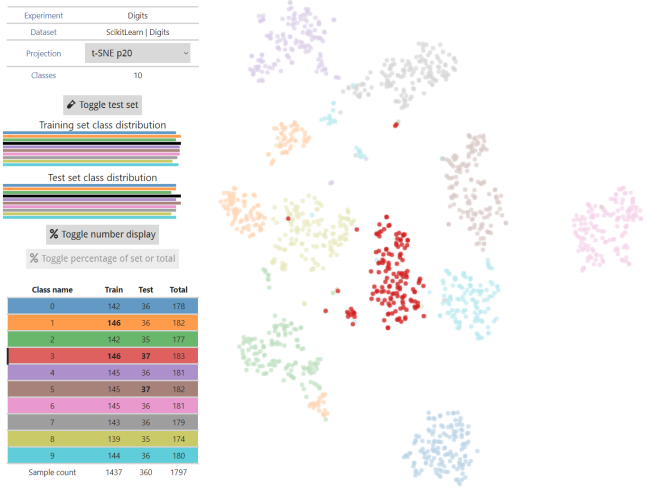

Figure 7: The dataset view reveals outliers and class overlap in a subset of the MNIST dataset [6] using t-SNE [23]. This figure shows how highlighting supports this task.

4.5 T-training: Analyzing the Training Behavior

Some classifier types, such as neural networks, are trained iteratively in several epochs. After each epoch, scores such as loss and accuracy are computed on the training set and a separate validation set. These scores are helpful to see how fast and how steady the performance of a classifier increases, if it does increase at all, or if it plateaus or drops after some number of epochs. When developers only consider the final score, they might miss problems such as under- or overfitting, which they could possibly avoid by adjusting hyperparameters or using early stopping instead of abandoning a certain parameterization entirely. Temporal data is usually displayed in line charts, as it is done in TensorBoard [1] which allows for the live-analysis of model training progress.

We provide a similar interface, so users that are familiar with TensorBoard can use it without the need to learn a new system. Since our approach needs to display many histories at once, we add coloring by hyperparameters and scores and an aggregated representation where groups of classifiers are shown as mean and confidence interval (Figure 5).

This view was helpful during the development process, when we tested our implementation and a neural network performed worse than expected. Its history showed that one model with this network had low scores at the beginning but then quickly arrived at a high accuracy. Apparently, others were stopped in their training before they could do the same. Increasing the early stopping patience indeed solved this problem.

4.6 T-confusion: Detecting Class-Specific Errors

In most datasets, some classes are easier to separate than others. Confusion matrices are commonly used to show which classes are confused with which and how often.

We display confusion matrices for each classifier and also provide an average confusion matrix for a selection of multiple classifiers. Values are color-coded and shown relative to the class size or as absolute sample counts (Figure 6).

4.7 T-separability: Analyzing the Dataset

Some of the problems that confusion matrices indicate might be caused by properties of the dataset. One example are different numbers of samples per class, which can lead to biased classifiers. Classes with a lot of overlap are hard to separate and outliers are often hard to classify.

To reveal such patterns, we include a dimensionality reduction scatterplot of the data, with samples colored by their class label. Additionally, a bar chart visualizes the class sizes and a table shows relative and absolute sample counts. Interactive highlighting of all samples from a class helps find possibly problematic samples in the scatterplot (Figure 7).

5 CASE STUDIES

In this section, we demonstrate how ClaVis can effectively support classifier developers in their work. We do this in multiple case

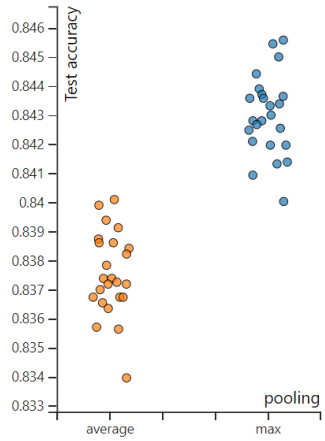

Figure 8: After choosing the axes and applying jitter to the pooling dimension, the scatterplot view clearly shows that max pooling leads to a better accuracy than average pooling in most of the classifiers.

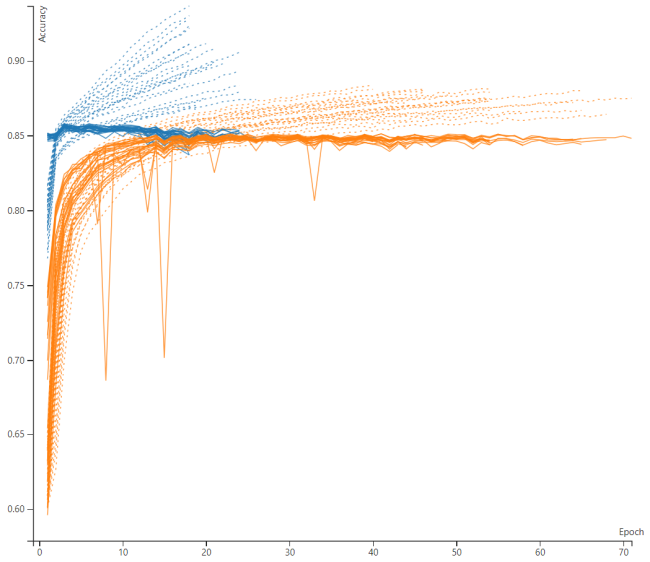

Figure 9: Classifiers show different training behaviors depending on pooling method. Here, all classifiers' training histories are shown together. Dashed lines represent training and solid lines validation accuracy. Average pooling (orange) sometimes leads to a drop in validation accuracy for single epochs. Classifiers trained with max pooling (blue) learn faster and more stable.

studies with different real world, natural language processing (NLP) datasets from the machine learning experts in our team.

First, we analyze models trained for dialog act classification on the datasets *Meeting Recording Dialog Act Corpus* (MRDA) [36] and *Switchboard Dialog Act Corpus* (SWDA) [15]. To avoid focusing on only one kind of NLP problem, we also study the speech corpora *IEMOCAP* [8] and *CREMA-D* [9] for emotion recognition.

5.1 MRDA and SWDA

We trained convolutional neural networks (CNNs) with different options for pooling, dropout, number of filters, and kernel size. Although the datasets have different structures (5 classes in MRDA and 42 in SWDA), the results were similar. We therefore only show images for the MRDA dataset.

The most salient difference we saw was between max pooling and average pooling. In the scatterplot view, the correlation matrix showed that there was probably an impact of the pooling method on scores (Figure 2). When clicking on the corresponding square to select *pooling method* and *test accuracy* as axes, we observed that max pooling was clearly better in most cases (Figure 8). We then swapped accuracy for training time and found that CNNs with max pooling were also training faster.

In the similarity map, we saw two clearly separated clusters for classifiers that used average pooling and max pooling, respectively. This means that the pooling method has a strong influence on the predicted probabilities.

Furthermore, the history view for the MRDA dataset showed a strange training behavior for classifiers with average pooling, where the validation accuracy sometimes decreased substantially for a single epoch and then improved again (Figure 9). This was not the case for max pooling, which was easy to spot by coloring the lines by pooling method. Stopping the training at one of these epochs would lead to a significantly worse test accuracy than stopping one

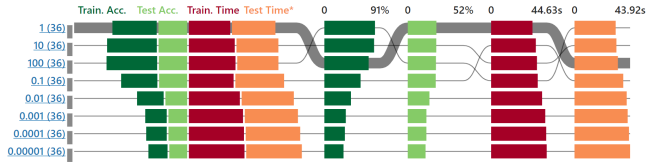

Figure 10: The ranking view can group classifiers by hyperparameter values and then show mean or median values or the best classifier for each group. In this case it shows the mean accuracies and times for different C values in SVMs. We can see that a value of 1 leads to good results in both accuracy and time.

epoch later. Without the history view, we would not have known about this difference in stability between pooling methods.

After deciding against using average pooling, we filtered out the respective classifiers and looked for further differences. Although these were not as pronounced, the ranking, scatterplot, and history views all revealed, for example, that in most cases a dropout ratio of 0.5 performed better than a ratio of 0.7. With these new insights, we could now refine our search space and train more models using the hyperparameters that worked well and then analyze them for further insights.

5.2 IEMOCAP

For the IEMOCAP dataset, we trained MLPs and SVMs and evaluated them via five-fold cross validation. We first looked at the ranking and grouped the results by fold number, to see if the classifiers have similar mean test accuracies on all folds. Since this was the case, we assumed that the folds where chosen fairly.

Next, we wanted to know which value works best for the C hyperparameter of SVMs. We therefore grouped them by this parameter in the ranking and saw that on average a value of 1 works fine for both accuracy and training time (Figure 10).

For the MLPs, we wanted to know if the cross validation folds have an influence on the training behavior. When grouping by fold, we found that the classifiers that were tested on fold 1 (i.e. trained on all other folds) were trained for almost twice as many epochs than those that were tested on fold 3.

5.3 CREMA-D

For the CREMA-D dataset, we looked at the dataset view to see which classes are easier to separate than others. To do this, we highlighted each class and looked for overlaps. The classes *angry* and *sad* seemed to be easier to separate than the rest. This confirmed prior experience and was also visible in the confusion matrices. Furthermore, we saw that the data looked rather scattered instead of clustered by classes, regardless of which dimensionality reduction we tried (PCA, UMAP, and t-SNE). This fact inspired one of the domain experts in our team to reflect on his feature selection, thinking about ways to make the features more informative in order to improve class separability.

6 CONCLUSION

We empirically compile a set of high-level tasks that are common in classifier design, complementing previous goal characterizations for visualization-assisted machine learning [32]. To address these tasks, we present a system named ClaVis, that helps users easily train tens to hundreds of different classifiers, visually compare them, find appropriate hyperparameters, and better understand hyperparameters and their effects. Our case studies demonstrate that ClaVis is an effective support for various tasks related to classifier engineering and analysis. As with all work, our approach comes with a specific focus and several limitations, which we discuss in the following.

Contrary to some related work [10, 14, 33, 41], we focus on a flexible solution for classifier developers that are tech-savvy and want to implement their own code. This focus comes at the cost of guidance and usability. We include plugins for many classifier algorithms and various datasets to allow for a quick start and to provide baselines. Own plugins can be shared with colleagues or students for reproducible results. While it may take some time to learn all features of ClaVis, we are confident that it will cover most use cases and that it can be extended for even more.

At the same time, we are optimistic that – with some additional usability engineering – the approach can also be useful for other target audiences. For instance, we conducted an initial study with students, in which we wanted to understand the approach's value for learning and education (details in our supplementary material). We presented ClaVis in a machine learning class to 16 students and collected qualitative feedback regarding their subjective judgments on utility, usability, and understandability. They generally agreed that an approach like ClaVis is helpful for studying and better understanding the machine learning models they were taught in class, as it gives them an easy way to experiment and play around with different datasets and hyperparameters. The students also pointed out how we could improve ClaVis in the future, for example with more guidance on how to use our visualizations effectively, an extension to analyze behavior on multi-machine clusters, and adding visualizations for domain-specific data such as word embeddings. We leave a broader, quantitative usability study for future work.

There are limitations in terms of the perceptual scalability of our visualizations and the computational effort required to produce them. Multiple factors may impact the performance of our system, namely the number of classifiers and the size of the predictions, which is proportional to the number of classes and samples in the dataset. We tested our visualizations with more than a thousand classifiers shown at once. While it was still usable performance-wise, some views such as the ranking cannot present that amount of information and still be readable. We address this limitation by providing ways to sort, filter, and group classifiers, which allow the user to focus on interesting subsets and reduce the amount of visual clutter. Some views, for example the similarity map and the scatterplot, can still reveal coarser patterns while showing the complete data. The size of the predictions impacts the time and memory required to produce the projections for the similarity map. To avoid long waiting times, we compute the projections in the background while the user can already examine all other views. With a larger number of classes, confusion matrices become hard to read. We still found them to show useful patterns with hundreds of classes or more. In our dataset view, we use colors to encode the class of each sample. Since humans are unable to easily distinguish more than a few colors, we found it hard to detect patterns such as class overlaps, although highlighting a class helps.

Currently, our implementation is limited to classification. Views such as the ranking, scatterplot, similarity map, and history can be easily adapted to other problems, while parts such as the confusion matrices can be replaced by problem-specific visualizations. Thereby, ClaVis could be extended to other machine learning domains such as regression or sequence-to-sequence transformation. In principle, our approach could support any comparison problem for which scores can be produced. Adding an interactive selection of multiple groups of classifiers would allow users to analyze them in juxtaposed views or see visually encoded differences. We could further extend our confusion matrices, for example to display the samples that are misclassified or classified differently by two classifiers on demand.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 251654672 – TRR 161 (A08) and under Germany's Excellence Strategy – EXC-2075 – 390740016.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, et al. 2016. TensorFlow: A System for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 265–283.

[2] Rocio Alaiz-Rodríguez, Nathalie Japkowicz, and Peter Tischer. 2008. Visualizing classifier performance on different domains. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Vol. 2. IEEE, 3–10. https://doi.org/10.1109/ICTAI.2008.21

[3] Bilal Alsallakh, Allan Hanbury, Helwig Hauser, Silvia Miksch, and Andreas Rauber. 2014. Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 20, 12 (2014), 1703–1712.

[4] Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, 337–346.

[5] James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference (ScyPy)*. 13–20.

[6] Léon Bottou, Corinna Cortes, John S Denker, et al. 1994. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR), Vol. 3 - Conference C: Signal Processing*. IEEE, 77–82. https://doi.org/10.1109/ICPR.1994.576879

[7] Brent Komer, James Bergstra, and Chris Eliasmith. 2014. Hyperopt-Sklearn: Automatic hyperparameter configuration for Scikit-Learn. In *Proceedings of the 13th Python in Science Conference (ScyPy)*, Stéfan van der Walt and James Bergstra (Eds.). 32–37. https://doi.org/10.25080/Majora-14bd3278-006

[8] Carlos Busso, Murtaza Bulut, Chi-Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N Chang, Sungbok Lee, and Shrikanth S Narayanan. 2008. IEMOCAP: Interactive emotional dyadic motion capture database. *Language Resources and Evaluation* 42, 4 (2008), 335–359.

[9] Houwei Cao, David G Cooper, Michael K Keutmann, Ruben C Gur, Ani Nenkova, and Ragini Verma. 2014. CREMA-D: Crowd-sourced emotional multimodal actors dataset. *IEEE Transactions on Affective Computing (TAC)* 5, 4 (2014), 377–390.

[10] Dylan Cashman, Shah Rukh Humayoun, Florian Heimerl, et al. 2019. A User-based Visual Analytics Workflow for Exploratory Model Analysis. *Computer Graphics Forum (CGF)* 38, 3 (2019), 185–199. https://doi.org/10.1111/cgf.13681

[11] Rene Cutura, Stefan Holzer, Michaël Aupetit, and Michael Sedlmair. 2018. VisCoDeR: A tool for visually comparing dimensionality reduction algorithms. In *Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[12] Ronald A Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7, 2 (1936), 179–188.

[13] Hongbo Gao, Bo Cheng, et al. 2018. Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics (TII)* 14, 9 (2018), 4224–4231.

[14] Yolanda Gil, James Honaker, Shikhar Gupta, et al. 2019. Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces (IUI) (IUI '19)*. ACM, 614–624. https://doi.org/10.1145/3301275.3302324

[15] John J Godfrey, Edward C Holliman, and Jane McDaniel. 1992. SWITCHBOARD: Telephone speech corpus for research and development. In *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1. IEEE, 517–520.

[16] Benjamin Höferlin, Rudolf Netzel, Markus Höferlin, Daniel Weiskopf, and Gunther Heidemann. 2012. Inter-active learning of ad-hoc classifiers for video visual analytics. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*. 23–32. https://doi.org/10.1109/VAST.2012.6400492

[17] Nathalie Japkowicz, Pritika Sanghi, and Peter Tischer. 2008. A projection-based framework for classifier performance evaluation. In *Machine Learning and Knowledge Discovery in Databases (KDD)*, Walter Daelemans, Bart Goethals, and Katharina Morik (Eds.). Springer, 548–563.

[18] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (KDD '19)*. ACM, 1946–1956. https://doi.org/10.1145/3292500.3330648

[19] Medha Katehara, Emma Beauxis-Aussalet, and Bilal Alsallakh. 2017. Prediction scores as a window into classifier behavior. In *NIPS 2017 Symposium on Interpretable Machine Learning*. NIPS. https://arxiv.org/pdf/1711.06795.pdf

[20] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *The Journal of Machine Learning Research (JMLR)* 18, 1 (2017), 826–830.

[21] David Laredo, Yulin Qin, Oliver Schütze, and Jian-Qiao Sun. 2019. Automatic model selection for neural networks. *arXiv preprint arXiv:1905.06010* (2019). https://arxiv.org/pdf/1905.06010.pdf

[22] Alexander Lex, Samuel Gratzl, Nils Gehlenborg, Hanspeter Pfister, and Marc Streit. 2013. LineUp: Visual analysis of multi-attribute rankings. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19, 12 (2013), 2277–2286. https://doi.org/10.1109/TVCG.2013.173

[23] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research (JMLR)* 9, Nov (2008), 2579–2605.

[24] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. 2018. UMAP: Uniform manifold approximation and projection. *The Journal of Open Source Software (JOSS)* 3, 29 (2018), 861.

[25] Gledson Melotti, Cristiano Premebida, Nuno MM da S Goncalves, Urbano JC Nunes, and Diego R Faria. 2018. Multimodal CNN pedestrian classification: A study on combining LIDAR and camera data. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3138–3143.

[26] Miriah Meyer, Michael Sedlmair, and Tamara Munzner. 2012. The four-level nested model revisited: Blocks and guidelines. In *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors-Novel Evaluation Methods for Visualization*. ACM, 11.

[27] Yao Ming, Huamin Qu, and Enrico Bertini. 2018. RuleMatrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 25, 1 (2018), 342–352.

[28] Tamara Munzner. 2009. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 15, 6 (2009), 921–928.

[29] Thomas Mühlbacher and Harald Piringer. 2013. A partition-based framework for building and validating regression models. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19, 12 (2013), 1962–1971. https://doi.org/10.1109/TVCG.2013.125

[30] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.

[31] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. 2016. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 23, 1 (2016), 61–70.

[32] Dominik Sacha, Matthias Kraus, Daniel A Keim, and Min Chen. 2019. VIS4ML: An ontology for visual analytics assisted machine learning. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 25, 1 (2019), 385–395.

[33] Aécio Santos, Sonia Castelo, Cristian Felix, et al. 2019. Visus: An interactive system for automatic machine learning model building and curation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA) (HILDA'19)*. ACM, Article 6. https://doi.org/10.1145/3328519.3329134

[34] Michael Sedlmair, Christoph Heinzl, Stefan Bruckner, Harald Piringer, and Torsten Möller. 2014. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 20, 12 (2014), 2161–2170.

[35] Michael Sedlmair, Mariah Meyer, and Tamara Munzner. 2012. Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 18, 12 (2012), 2431–2440. https://doi.org/10.1109/TVCG.2012.213

[36] Elizabeth Shriberg, Raj Dhillon, Sonali Bhagat, Jeremy Ang, and Hannah Carvey. 2004. The ICSI meeting recorder dialog act (MRDA) corpus. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue at HLT-NAACL 2004*. 97–100.

[37] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. 2017. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788* (2017). https://arxiv.org/pdf/1708.03788.pdf

[38] Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S Tan. 2009. EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1283–1292.

[39] Chris Thornton, Frank Hutter, et al. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 847–855. https://doi.org/10.1145/2487575.2487629

[40] Warren S Torgerson. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17, 4 (1952), 401–419.

[41] Qianwen Wang, Yao Ming, Zhihua Jin, Qiaomu Shen, Dongyu Liu, Micah J. Smith, Kalyan Veeramachaneni, and Huamin Qu. 2019. ATMSeer: Increasing transparency and controllability in automated machine learning. In *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI) (CHI '19)*. ACM, Article 681. https://doi.org/10.1145/3290605.3300911

[42] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. 2017. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 24, 1 (2017), 1–12.

[43] Cheng Xue, Qi Dou, Xueying Shi, Hao Chen, and Pheng-Ann Heng. 2019. Robust Learning at Noisy Labeled Medical Images: Applied to Skin Lesion Classification. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 1280–1283. https://doi.org/10.1109/ISBI.2019.8759203