Parallel Embeddings: a Visualization Technique for Contrasting Learned Representations

Dustin L. Arendt
dustin.arendt@pnnl.gov
Pacific Northwest National Laboratory
Richland, WA

Nasheen Nur
nnur@uncc.edu
The University of North Carolina at Charlotte
Charlotte, NC

Zhuanyi Huang
zhuanyi.huang@pnnl.gov
Pacific Northwest National Laboratory
Richland, WA

Gabriel Fair
gfair@uncc.edu
The University of North Carolina at Charlotte
Charlotte, NC

Wenwen Dou
wdou1@uncc.edu
The University of North Carolina at Charlotte
Charlotte, NC

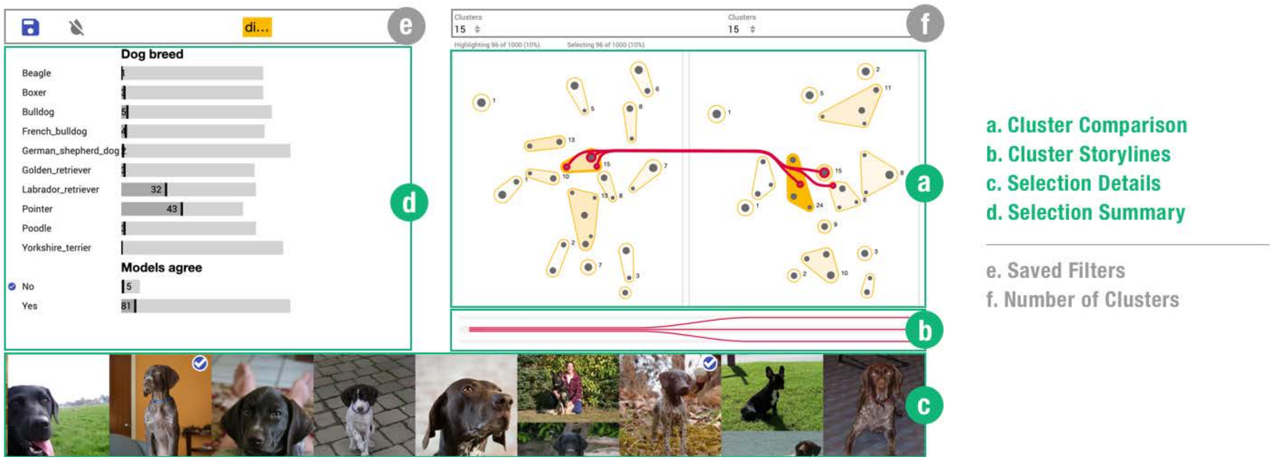


Figure 1: *Parallel Embeddings* has 4 primary views: (a) the "cluster comparison" highlights differences in embeddings of learned representations, (b) the "cluster storylines" overviews the current selection, (c) the "selection details" shows the raw data for the selection, e.g. images, and (d) the "selection summary" shows the distribution of metadata pertaining to the current selection. The user can also (e) save and load filters and (f) change the number of clusters in the cluster comparison view.

ABSTRACT

We introduce "*Parallel Embeddings*", a new technique that generalizes the classical Parallel Coordinates visualization technique to sequences of learned representations. This visualization technique is designed for concept-oriented "model comparison" tasks, allowing data scientists to understand qualitative differences in how models interpret input data. We compare user performance with our tool against *TensorBoard Embedding Projector* for understanding model accuracy and qualitative model differences. With our tool, users were more accurate and learned strategies for the tasks more quickly. Furthermore, users' analytical process in the comparison condition was positively influenced by using our tool beforehand.

CCS CONCEPTS

• **Human-centered computing** → **Visualization techniques**; *Interactive systems and tools*; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Dimension reduction visualization, machine learning explanations, model comparison, image classification, user studies

ACM Reference Format:
Dustin L. Arendt, Nasheen Nur, Zhuanyi Huang, Gabriel Fair, and Wenwen Dou. 2020. Parallel Embeddings: a Visualization Technique for Contrasting Learned Representations. In *25th International Conference on Intelligent User Interfaces (IUI '20), March 17–20, 2020, Cagliari, Italy.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3377325.3377514

1 INTRODUCTION

Training a good machine learning model often involves a difficult final step of choosing one model from many candidates, sometimes referred to as "model selection." Model selection can be performed naively by choosing the model that minimizes an arbitrary performance metric, e.g. F-score [38], and is partially automated via "hyper-parameter search" [5]. "AutoML" techniques, e.g. Autosklearn [9], fully automate the model and hyper-parameter search.

However, there may be other considerations besides performance for selecting a good model [2], e.g. robustness, explainability/interpret-ability, bias/fairness, data quality/annotator reliability, and higher consequences of certain errors. To support these other objectives, several other model selection & comparison tools have been developed [1, 4, 18, 44]. Even more tools have been recently developed for understanding models in general [14], which can also be used to compare and select models.

In this paper we present *Parallel Embeddings*, which is a visualization technique that is designed to be used by data scientists for concept-oriented model comparison, i.e. going beyond F-score. Inspired by the parallel coordinates visualization technique [13], which shows how a collection of samples vary across a sequence of features, our technique, *Parallel Embeddings*, shows how a collection of samples vary across a sequence feature matrices. Applied to machine learning models, our visualization allows data scientists to understand differences in models' learned representations.

2 RELATED WORK

2.1 Model Comparison

Many visualization tools perform model comparison and seek to answer: (1) what is the best model? (2) how are two models similar or different? We identify two overall patterns in the literature for model comparison visualizations. Some visualizations are "performance-oriented" and focus on explaining *how well* different models performed their task. Other tools are concept oriented and focus on explaining *what* different models learned to perform their task. Finally, a variant of the model comparison problem is to understand how models change *over time* throughout the training process. One model at different times is arguably two different models. Below we discuss several model comparison tools in detail within this conceptual framework.

2.1.1 Performance-oriented Visualizations. Performance-oriented visualizations support model comparison by allowing the user to understand the differences in model performance, e.g. F-score. While not the main contribution of the work, McMahan et al. [28] created a visualization tool to compare three model variants against a control model. Their tool shows a heatmap of model variant by evaluation criteria relative to the control model.

MLCube [18] and ActiVis [17] are exploratory visualization tools which allow data scientists to understand how model performance correlates with subsets of instances, where subsets are user-created by grouping instances based on their features and attributes, i.e. instance metadata not available to the model. MLCube directly supports A/B model comparison by using color to encode whether model A or B is performing better for a particular subset.

Manifold [47] is a visualization tool oriented around a grid of performance scatter plots. The tool compares models in a pairwise fashion and visually separates true/false positives/negatives into separate scatter quadrants.

2.1.2 Concept-oriented Visualizations. Concept-oriented visualizations convey semantic differences between models, i.e. how they interpret data similarly or differently. Alexander et al. [1] presented a tool for comparing topic models that used spatial encoding and color to accentuate topic model disagreement on particular documents. Notably this comparison is done at the document level, i.e. is instance-based. The tool is designed to help a user understand topics, topic similarity, and topic change over time.

Yu et al [44] presented a technique for visualizing and comparing convolutional neural networks (CNNs) for image classification. The tool visualizes learned representations of a CNN at different layers (using t-SNE [25] for dimension reduction). Rather than encoding instances as points, they are represented as their corresponding images, and all empty space is interpolated with its nearest neighbor. The end result is a 2-D collage of images that conveys some semantic information about how the model interprets different input images. Comparison between two layers is accomplished by simply juxtaposing the visualization of the two layers.

Although not specifically for model comparison, a variant of this technique is used in a tool by Liu et al [23] which performs hierarchical clustering on the model's learned representations. Liu et al used a rectangle packing algorithm to allocate space for images. The end result is also a 2-D image collage that conveys the models semantic organization of input images.

2.1.3 Time-oriented Visualizations. Time oriented visualizations focus on how one or more models are changing over time, either in terms of model performance or the learned representations. For example, the NetworkDissection tool [4] allows users to understand how the number of training epochs affects what is learned by the model. The tool summarizes layers and units based on what objects they detect. The tool is also used to understand how initialization, dropout, batch normalization, and size affect layer function.

CNN Comparator [46] compares a convolutional neural network (CNN) after different number of epochs, e.g. 10 and 100. It shows the distribution of parameter differences at a given layer and allows the user to compare the classification probabilities of a single instance across two different models. The tool also shows relevant cropped image patterns based on activation patterns and compares different layers at different numbers of epochs.

DeepTracker is designed to let a data scientist explore the full training output of a deep network [22]. The tool relates the layers in an abstract architecture diagram to their activations across a set of instances. Notably, the tool also groups models with similar evolving trends during training into clusters, allowing data scientists to identify similar types of models; these models are visualized as sequences examplar input images.

For reinforcement learning, understanding how a model's policy evolves over time is crucial. Wang et al [40] developed a performance-oriented tool, DQNViz, which related the successfulness of the model over time to its action sequences. ReLVis [34] lets a user understand the relationships between state sequences (observations) and model performance (reward) as a single model changes over time, or across multiple models.

2.2 Explanation via Dimension Reduction

Dimension reduction visualization is a recurring theme in the literature we surveyed related to model comparison and explanation [14]. Typically dimension reduction visualization represents instances as points, and encodes model prediction or ground truth using color. Doing so allows data scientists to begin discerning what concepts the model has learned by a particular layer. For example, dimension reduction visualization of learned representations is central to ActiVis [17] in addition to visualizing individual neuron activation. Dimension reduction visualization made it clear that subsequent layers in the model were separating classes more effectively.

EmbeddingProjector [36] is a general purpose exploratory dimension reduction tool that incorporates multiple popular embedding techniques and allows users to explore embeddings (one at a time) across different network architectures, layers, datasets. The tool represents instances as points or as glyphs, e.g. raw images or text, which can have arbitrary metadata encoded with color and size. Dimension reduction visualization of neuron activation is also central to the model comparison tool by Rauber et al [30]. This tool uses t-SNE in a novel way to show how activation patterns have changed across training epochs and relies on edge bundling to reduce clutter.

For reinforcement learning model comparison, Saldanha et al. [34] and Jaderberg et al [15] used dimension reduction to help reveal patterns in agent-state sequences. Zhavy et al [45] used t-SNE to understand the model's representation of its current state.

3 DESIGN

Several of the authors work with domain experts, data scientists, and machine learning experts on a daily basis across a variety of science and national security projects involving applied machine learning and artificial intelligence. From this past working experience we identified *who* could benefit, *what* was needed, and *how* this could be accomplished. We consolidated our design insight into three use cases that can each be addressed by our visualization approach for model comparison. We outline this approach and its implementation as a visualization tool and conclude this section with a discussion of alternative designs considered.

3.1 Target Users

We designed *Parallel Embeddings* with two types of users in mind. *Parallel Embeddings* is primarily designed for machine learning practitioners and experts, which we refer to for brevity as "data scientists". However, less research effort has been spent on developing tools to help "decision makers", who we characterize as non-data scientists who make decisions about the automation of workflows with machine learning models. While most current explanation tools are targeted towards data scientists who understand machine learning and deep learning [14], tools that target non-expert users, e.g. decision makers, have the potential for broader impact. Furthermore, a hypothetical tool that is intuitive and effective for decision makers, should also be helpful for data scientists. We employ an example-based explanation design approach that should be suitable for communication with both data scientists and decision makers.

3.2 Needs, Goals, and Use Cases

Our interactions with data scientists over the past several years revealed two major needs relating to model comparison.

- N1 Data scientists need ways to reduce the time and computational cost of training machine learning models. Model training is both time and resource intensive, i.e. data scientists and GPU's are shared across many projects.
- N2 Decision makers need to be convinced that, when a new model replaces an old one, the new model is both qualitatively *and* quantitatively better. In other words, a model with a higher F-score may actually contain hidden biases or new types of errors, e.g. due to over-fitting.

We identified the following ways each of the above needs could be met using a visualization tool, which we define as the various "goals" of our tool.

- G1 Determine what concepts a model has learned or forgotten between arbitrary training epochs. G1 supports N1.
- G2 Understand the function of layers in the network to determine if they are needed for the particular machine learning task and data at hand. G2 supports N1.
- G3 Determine how two models' conceptual representation of example data differ, and how this relates to their correctness, i.e. did the "better" model learn a helpful new concept or just over-fit the training data? G3 supports N2.

We identified three use cases, i.e ways the high-level goals might be accomplished with a visualization tool.

- UC1 A data scientist uses visualization to understand the purpose of a specific layer in a deep network.
  1. The user wants to know if the layer has learned a useful concept
  2. Layers that do not perform a useful task might be reduced or deleted.
- UC2 A data scientist uses visualization to understand how the model has improved over time, e.g. as a result of training.
  1. The user wants to know if important concepts are learned early in training or later.
  2. If important concepts are learned earlier, the training time can be reduced to save costs.
- UC3 A data scientist or decision maker uses visualization to understand the differences between a proposed model and one already in production, e.g.
  1. The user wants to compare a more accurate model with a simpler or more explainable one.
  2. The user wants to know what effect a de-biasing algorithm has on model behavior.

Many techniques from the literature we surveyed relied on 2-D embeddings of model activations given input data to understand the purpose of particular layers in the network. This technique can be leveraged for model comparison if a user can understand the similarities and differences between 2-D embeddings of the same input data for different models. Embeddings of data tend to create clusters that correspond to semantically meaningful concepts upon inspecting their membership. Thus, model comparison can be accomplished by (1) identifying interesting clusters based on their contents and (2) understanding how and why these clusters split and merge between models being compared.

3.3 *Parallel Embeddings* Design & Overview

With these needs, goals, and use cases in mind, we designed a new visualization technique, *Parallel Embeddings*. We followed the parallel coordinates metaphor, relying on juxtaposition and integration (linking the same entity across views) to dictate the overall design. Embeddings of learned representations are arranged horizontally as square *frames* and each embedding is allocated the same screen space. For scalability purposes and to simplify the visualization, we apply clustering to each embedding. Clustering has been used for parallel coordinates to reduce clutter and make broader trends more easily discernible [16, 26].

Parallel Embeddings supports comparisons across multiple embeddings of the same example data by visually encoding what sets of samples remain nearby to each other throughout the entire sequence. We refer to these groups as *cohorts*. Embeddings are arranged horizontally, and sets of samples are connected with lines in a manner inspired by the classical Parallel Coordinates visualisation technique, hence the name *Parallel Embeddings*. Thus, a cohort is the set of examples that belong to the same sequence of clusters across frames. To indicate multiple cohorts belong to the same cluster within a frame, we enclose those cohorts with a polygon defined by the convex hull of the cohorts' centroids. Convex hulls have been used in the past to indicate cluster membership [8] and group membership in graph visualization [39].

Below we describe the four primary components (a-d) and two supporting components (e, f) in *Parallel Embeddings* (see also Fig. 1). **(a) Cluster Comparison:** Primarily, users understand differences between models with *Parallel Embeddings* by understanding differences between clusterings of their respective embeddings. Users reveal these cluster differences by first hovering over a cluster or cohort to get an idea of where the target cohorts appear in other frames. Then she can select the cohorts she is interested in by clicking individual cohorts or a cluster (which selects all cohorts it contains). Once selected, the link between cohorts across frames is visualized with a curved path that is routed efficiently around unrelated clusters it does not belong to. **(b) Cluster Storylines:** The storyline view is visible when at least one cohort is selected in (a). This more clearly conveys the cohort's cluster membership by encoding cluster membership on the y-axis using the technique described by Arendt and Pirrung [3]. Clusters are aligned across frames based on the degree of overlap, i.e. if two clusters overlap by more than 50%, they are considered the same cluster. **(c) Selection Details:** When one or more cohorts are selected, their details, i.e. raw data, are rendered in a grid. This view gives the user more context about the underlying data to assign semantic meaning to clusters and cohorts. **(d) Selection Summary:** The selection summary shows a histogram of instance features for a separate metadata table provided to the tool. This is intended to convey information like model predictions, ground truth, etc. Hovering on a cluster/cohort in the cluster comparison view, shows a second, inner bar corresponding to the distribution of that metadata for just the highlighted cluster/cohort. Selecting bars from this histogram configures a simple filter that reveals what clusters in the cluster comparison view best match the selection. This feature allows users to focus on the subset of data that is interesting to them if they know this ahead of time. Instances visible in the selection details view that match the current filter are indicated with a check mark.

Other supporting functionality includes: **(e) Saved filters:** *Parallel Embeddings* also allows users to save the current filter configuration and assign it a color. Clusters in the cluster comparison view are colored according to this. **(f) Number of clusters:** The user can also interactively change the number of clusters independently for any frame in the Cluster Comparison view.

3.4 Design Alternatives

We considered several design alternatives, discussed below. Initial designs and prototypes assumed each sample would be represented as a point, similar to many existing dimension reduction visualizations used in a machine learning context [14, 15, 17, 30, 34, 36, 37, 45]. Similar to parallel coordinates, points representing the same sample in adjacent frames were connected with a line segment. However, this created a large amount of clutter due to edge crossings and made the visualization unreadable (see Fig. 2a). Edge bundling [20] could be used to reduce this clutter (see Fig. 2b), and has been demonstrated for model comparison within superimposed dimension reduction plots [30].

Clustering can be used to increase the abstraction of the visualization by aggregating points into closely related groups. Liu et al [23] performed hierarchical clustering on the representation matrix to help explain an image classifier. For comparison of a classifier as it trains, different layers within a classifier, or multiple different classifiers, hierarchical clustering can be applied to each representation matrix to produce a "nested tracking graph". There have several different techniques developed recently to visualization such data structures [19, 24, 42, 43].

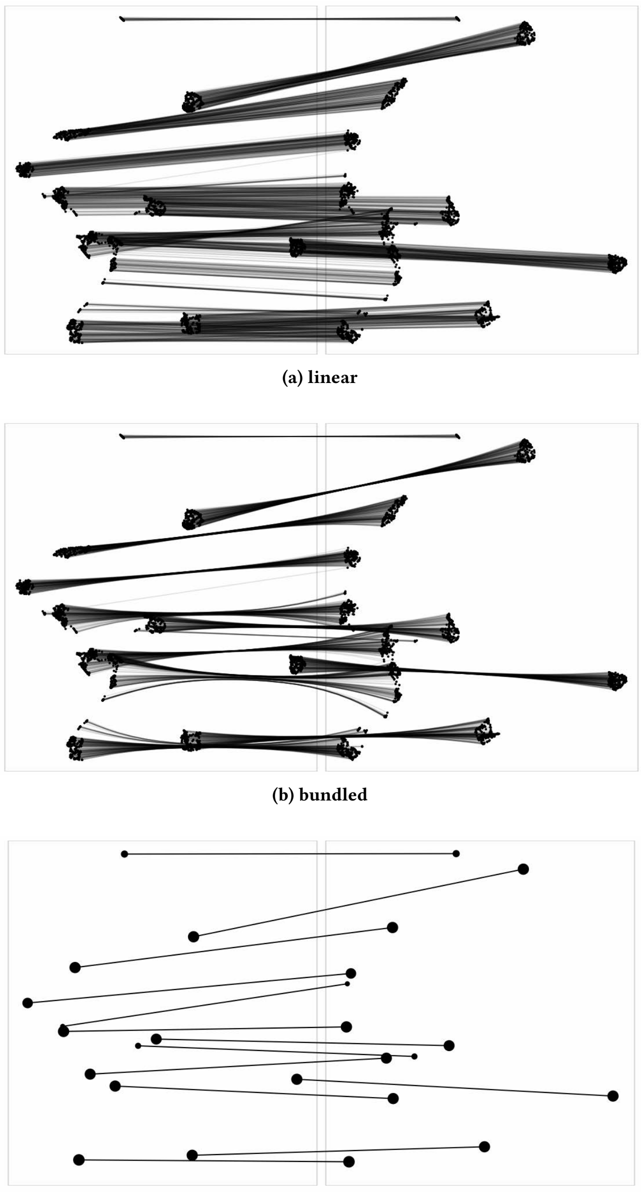
Storyline visualization can be used to visualize instances changing cluster membership over time [3, 31]. Storyline visualizations become cluttered for relatively few lines, which can be mitigated by further abstraction to show just the most prevalent trends (of changing cluster membership) using Alluvial diagrams [33].

4 SYSTEM IMPLEMENTATION

Parallel Embeddings takes as input a length k sequence of sample by feature matrices $S = (X_1, X_2, ..., X_k)$ and produces a visualization that emphasizes which samples remain nearby each other throughout the sequence S. Each matrix is an $n \times m(t)$ matrix where n is the number of samples and $m(t)$ is the number of features for frame t, which may be different for each frame. Row i for each X_t corresponds to the same sample.

Recall from Section 3.3 that the visualization corresponding to each embedded matrix is a *frame* and sets of instances with similar trajectories across frames are *cohorts*. Given the above definitions, *Parallel Embeddings* are computed by the following steps:

(1) Project each X_t into 2-D X'_t.
(2) Cluster (hierarchically) each X'_t to produce a parent pointer array P_t.
(3) Realize each P_i as a partition of $n(t)$ clusters based user input.
(4) Group instances into cohorts based on unique cluster membership across frames.


(a) linear

(b) bundled

(c) aggregated

Figure 2: *Parallel Embeddings* Design alternatives. (a) naively connects points in adjacent frames with straight lines. (b) bundles lines based on cohorts. (c) aggregates cohorts into single lines and encodes the size of the cohort with the radius of the endpoint.

(5) Compute the convex hull around the centroid of each cohort within each cluster in each frame.
(6) Render cohorts as paths connecting their centroids in each frame, with paths routed efficiently around clusters' convex hulls.

Steps 1 and 2 are completed prior to any user interaction. Steps 3 and after are repeated each time the user changes the number of clusters for any frame. Below we discuss the relevant technical details and justification for the above steps. In Section 4.3 we also discuss relevant software engineering choices.

4.1 Projection and Clustering

Wenskovitch et al [41] discuss the implications of combining clustering and dimension reduction techniques, stating that the choice of dimension reduction algorithm is important when using "dimension reduction preprocessing for clustering". Therefore, we selected a neighbor embedding technique, i.e. UMAP [27], as our dimension reduction algorithm. Neighbor embeddings [25, 27] optimize an objective that encourages neighbors in the higher dimensional space to be neighbors in the lower dimensional embedding, addressing some of the issues when clustering after dimension reduction discussed by Wenskovitch et al [41].

We used the `scikit-learn` implementation of hierarchical clustering[1] to cluster each X'_t. *Parallel Embeddings* converts the sequence of cluster merges found by hierarchical clustering into a parent pointer array. This data structure allows a partitional cluster to be created on the with linear time complexity in response to the user specifying a particular number of clusters.

4.2 Path Routing

While patterns like correlation have meaningful analogs between Cartesian and parallel coordinates, we expect no such analogs in parallel embeddings due to the arbitrariness of spatial positioning of samples produced by dimension reduction — only spatial proximity matters. Therefore, we modify the path routing of cohorts between frames to reduce visual clutter and make the lines easier to trace between frames. We defined the following path routing aesthetic criteria listed in order of importance:

A1 Paths should never reverse direction,
A2 Paths should avoid crossing clusters,
A3 Paths should wiggle as little as possible, and
A4 If a path wiggles, it should do so as far to the left as possible.

Paths representing cohorts are expected to be read from left to right, thus **A1** explicitly forbids pathways from ever traveling from right to left. The purpose of edge routing is to avoid cluster-path crossings, hence **A2**. However, we cannot explicitly forbid this, because the pathway must enter/cross each cluster it belongs to. Also, in some rare cases, e.g. due to grid discretization, there may be no route available without crossing a cluster, so **A2** can be violated but is strongly discouraged. Straighter lines are easier to trace, so **A3** implies paths with fewer wiggles (changes in the y-coordinate) are preferred over more efficient paths with more wiggles. However, usually a pathway with fewer wiggles is more efficient. Finally, **A4** states if a wiggle is required for routing, it is preferred this happens as earlier in the path, i.e. as far left as possible.

While the state of the art edge routing algorithms can optimally route paths around convex hulls [7], it was too difficult to extend this framework to satisfy the above aesthetic criteria. Instead we adapted an older technique based on shortest path routing within a

[1]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

grid discretization of the rendering area [10]. Our aesthetic criteria are enforced through placement of the nodes, direction of edges, and weighting of edges.

To do so, we used the following procedure:

(1) Discretize the rendering area into a set of points forming a hexagonal lattice, with adjacent points x-coordinates being spaced $\Delta x = 25$ pixels apart.
(2) Remove lattice points that intersect clusters' convex hulls.
(3) Add lattice points corresponding to each cohort's centroid in each frame.
(4) Add edges from the Delaunay triangulation of the modified lattice points.
(5) Remove lattice edges which do not connect cohort centroids and are longer than Δx pixels
(6) Orient the direction of lattice edges such that the source of each edge has a smaller x-coordinate (is farther left) than the destination.
(7) Weight edges according to Equation 1
(8) The control points between two adjacent cohort centroids follow the shortest path in the graph defined above.

Aesthetic criterion **A1** is enforced by the orientation of lattice edges found in step 6. Aesthetic criteria **A2-4** are reflected in the definition of the following cost function over lattice edges:

$$cost(i, j) = |y(i) - y(j)| + w_1 \cdot x(i) + w_2 \cdot (c(i) + c(j)), \qquad (1)$$

where $w_1 = 10^{-4}$ and $w_2 = 10^4$ act as arbitrarily low and high weights relative to Δx. The function $c(i)$ indicates whether a lattice node i corresponds to a cohort's centroid or not, i.e. was added in step 3 above. Thus, $c(i)$ returns 1 if so, and 0 otherwise. Figure 3 illustrates the modified triangular lattice and weighting.

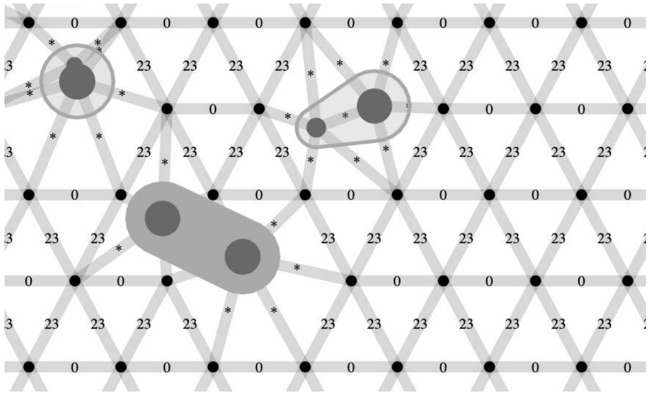


Figure 3: *Parallel Embeddings* path routing. Routing follows a triangular lattice except where modified by the presence of clusters. Arbitrarily high weights are denoted with a *.

4.3 Workflow Integration

As discussed above, *Parallel Embeddings* is designed and intended to be used by data scientists. Many data scientists choose as their development environment an interactive Python shell with a graphical web-based front end for visualization, e.g. Jupyter Notebooks[2]. While standalone desktop and web-based visualization prototypes tools are commonplace in this research community, we decided tight integration with Jupyter Notebooks would simultaneously simplify the implementation of the *Parallel Embeddings* tool as well as make it easier for users to install and use. Thus, data scientists can simply "pip install" the *Parallel Embeddings* python package, enable the widget for Jupyter, and use it immediately.

The *Parallel Embeddings* tool is split into two modules, (1) a python module which performs the preprocessing, i.e. dimension reduction, hierarchical clustering, JSON data formatting, and (2) a front-end React component that renders the visualization and handles all interaction. The front-end React component is packaged as a standalone JavaScript library that can be "npm installed" and included within a larger web-based application if desired. This organization allows *Parallel Embeddings* to be immediately used by our target end users while still providing flexibility to be included in other environments in the future.

The *Parallel Embeddings* tool is imported and instantiated like a typical python package/class. It accepts a sequence of NumPy[3] matrices as the input and an optional pandas[4] DataFrame defining metadata for each instance to be filtered on, e.g. ground truth label, predicted class, correct classification, etc.

5 EVALUATION

Parallel Embeddings is designed to help data scientists draw comparisons between different machine learning models allowing data scientists to make informed decisions in real-world scenarios. To assess *Parallel Embeddings* on these grounds, we conducted an in-laboratory within-subjects user study with data scientists from different disciplines. Our study is limited to machine learning experts and practitioners to ensure participants could complete the experiment tasks.

The experiment is designed to evaluate the following hypotheses:

- **H1:** *Parallel Embeddings* enables better performance for completing the experiment tasks measured by accuracy and time
- **H2:** The design of *Parallel Embeddings* led participants to form more efficient strategies during the analysis process.

To evaluate these hypotheses we used a mixed study design, with both between and within subjects variables. Our between subjects variable was *use case*, either understanding layers or understanding training over time. Our within subject variable was *tool*, either *Parallel Embeddings* or *TensorBoard Embedding Projector* (as a comparison condition). Section 5.2 further motivates this choice.

Each participant used both *TensorBoard Embedding Projector* and *Parallel Embeddings*, but only answered questions for one use case. Participants either used *Parallel Embeddings* first before *TensorBoard Embedding Projector* or vice versa. The combination of the 2 use cases and 2 possible tool orders created 4 unique treatment groups of participants. We balanced the number of participants and gender within each treatment. On average each session took one hour and thirteen minutes to complete.

[2]https://jupyter.org

[3]https://numpy.org

[4]https://pandas.pydata.org

5.1 Participants

We recruited 24 participants for the study; all participants were from the computer science and data science programs at the University of North Carolina at Charlotte. Among the 24 participants 5 were female and 16 were pursuing their MS degree and (the remaining were pursuing a doctoral degree). The age of the participants ranged from 22 to 31 years. We compensated participants with a $30 Amazon gift card after completion of the study.

5.1.1 Recruitment Criteria & Selection Process. Before starting the study the experimenter asked qualification questions to make sure the participants had a sufficient understanding of machine learning and deep learning algorithms (the models participants were asked to make sense of during the user study tasks were deep neural networks). The qualification questions focused on the basic concepts of supervised and unsupervised machine learning and the basic mechanism of a deep learning model, e.g. epochs and layers. The purpose of the qualification questions was to ensure the user studies were conducted with the target audience *Parallel Embeddings* is designed to benefit. In total, 33 people signed up for the user study but 6 were disqualified based on their answers to the qualification question and 3 did not show up for the study.

5.1.2 Experience & Expertise. Based on the response to one of the questions in the pre-questionnaire regarding participants' experience with neural networks, all participants self-identified as either deep learning researchers or practitioners. More specifically, 12 participants self-identified as deep learning experts who *implement and extend* deep learning models thus have a deeper understanding of how the deep learning algorithms work. The other 12 participants identified as deep learning practitioners who *apply* deep learning algorithms using existing libraries for their work and research.

In our pre-questionnaire, in addition to asking participants' experience with machine learning and deep learning, we also included questions on their familiarity with visualization and dimension reduction techniques. Twelve of the participants did not have any academic training in information visualization. All of them had at least "some" familiarity with information visualization and 10 of them identified themselves as above average to very familiar with it. Three participants did not have any familiarity with dimension reduction techniques; while some had a vague understanding of PCA, they were unfamiliar with UMAP and t-SNE. Fourteen of the participants self-identified as having sufficient to very good understanding of the applications and implementation of deep learning algorithms, whereas the rest of them claimed some to average knowledge in these areas.

5.2 Within Subjects Variable: Tool

To evaluate the efficacy of *Parallel Embeddings* for model comparison, we chose *TensorBoard Embedding Projector*[5] as our comparison condition. *TensorBoard Embedding Projector* is a freely available visualization tool for exploring learned representations via 2- or 3-dimensional projections. We adapted *TensorBoard Embedding Projector* for model comparison by opening the tool in two separate windows, with each window showing a different embedding representation. Each participant performed the same set of tasks with both tools. To control for learning effects and fatigue, we randomized the order of which tool participants experienced first.

We believe that *TensorBoard Embedding Projector* was a fair and valid comparison condition for several reasons. Of the research we surveyed, the most similar concept-oriented visualization [44] simply juxtaposes dimension reduction plots of the CNN model's layers—*Parallel Embeddings* builds on this technique. However, *TensorBoard Embedding Projector* is a feature rich tool for exploring high dimensional data; its polished user interface helps avoid experimental confounds due to usability issues that are more likely to be found in research prototypes. Though *TensorBoard Embedding Projector* was not designed to do so, the user can also similarly juxtapose by creating two views side by side. Thus our comparison condition directly evaluates the main contribution of *Parallel Embeddings*, i.e. integrating these side-by-side views, while attempting to avoid confounds due to usability issues.

5.3 Between Subjects Variable: Use Case

Our experiment involves two use cases — (1) Understanding differences in what a model learns over time (epochs) and (2) Understanding differences between layers for two different tools.

We asked 3 sets of questions when participants perform tasks with each tool. The questions are:

(1) **Overall Understanding:** How are the two epochs (layers) different in terms of instance-level memberships and how does the visualization portray the agreement or disagreement?
(2) **Accuracy Task (AT):** Can you identify the classification accuracy or inaccuracy by looking into the visualization?
  (a) Which are the top 2-3 breeds mostly misclassified and with which breeds are they being confused with?
  (b) Which are the top 2-3 breeds most often correctly classified?
(3) **Comparison Task (CT):** Can you summarize what concepts the model learned or unlearned in the later (right frame) compared to earlier (left frame)?
  (a) Which breeds' classification improved over layers or epochs?
  (b) Which breeds' classification unimproved over layers or epochs?

5.4 Experiment Materials

5.4.1 Data for the experiment tasks. We trained a classification model on the Stanford Dog Breed dataset[6]. This dataset is built with images and annotations from ImageNet[7] on 120 breeds of dogs across the world for fine-grained image categorization. We chose the images of 10 most represented breeds in the dataset.

5.4.2 Deep Learning Model. We built a Convolutional Neural Network to predict the breed of a given image of a dog. The model uses the pre-trained VGG-19 [35] and Resnet-50 [12] models as a fixed feature extractor, where the last convolutional output of both networks is fed as input to another, second-level model. Combining these two models achieves a small boost compared to using them separately. At the beginning of the second-level model, we added

[5]https://https://projector.tensorflow.org/

[6]http://vision.stanford.edu/aditya86/ImageNetDogs/
[7]http://www.image-net.org/

a global average pooling layer to make the network invariant to small translations or perturbations of an image. We then added multiple batch normalization layers which help to learn on a more stable distribution of inputs and to limit the co-variate shift by normalizing the activation of each layer. Several dropout layers are used in to help the network learn independent representations and prevent overfitting. Then we add a fully connected layer with a softmax with one node for each dog category to extract the last convolutional output for both networks.

5.4.3 Use-cases. For the **epoch usecase**, we projected the activations from the second layer to the output layer. The accuracy of the second epoch in the left frame of the visualization is 76.9% and the right frame (epoch = 20) is 80.3 %. Since we cannot project two activations in one single embedding view of *TensorBoard Embedding Projector*, we projected the two activations in two different *TensorBoard Embedding Projector* in two browsers (the left monitor was used for epoch=2 and the right monitor is used for the later epoch). This setup was to ensure both tools provide the same amount of information for the participants to complete the experiment tasks.

In the **layer usecase**, the left frame of the visualization presents the activation from an earlier layer (a dense layer with relu activation function) in the network to the output layer; the right frame presents activation from a later layer (batch normalization layer) to the output layer. The activations from the two different layers are collected in the same epoch (epoch=20). After training the model until 20^{th} epoch, the accuracy of the model was 80.3%.

5.4.4 Ground Truth. We computed the ground truth for the use cases on 1000 testing instances for the target variables of each subtask allowing us to quantitatively measure user performance. We identified misclassified breeds for both left and right epoch/layer, properly classified breeds, and the breeds which have an improved classification in the later frame or have a decline.

5.5 Data Collections and Measurements

At the beginning of each study, we collected the following demographic information: age, major, sex, familiarity with deep learning algorithms, information visualization, and dimension reduction. We then trained participants on the first tool explaining the features and functions, and provided a printout of the task questions. We also provided participants with a reference sheet for the features of the tool. The experimenter was available throughout the study to answer questions about the tools or the tasks.

During the experiment, the participants wrote down their answers directly on the paper with experiment questions. The experimenter also recorded participants' response time for the task and sub-tasks for each tool during each experiment session.

After completion of each task, we asked participants to answer a post questionnaire that employs a subset of the NASA TLX [11] questions in order to assess the task, system, and effectiveness of their performance. In the post-questionnaire, we also asked participants to report the usefulness of different features of the tool they just experienced on a Likert scale [21]. We collected data regarding the common strategies participants used to complete the task with the tool and asked for suggestions for improvements to the *Parallel Embeddings* tool. We repeated the same process for the second tool.

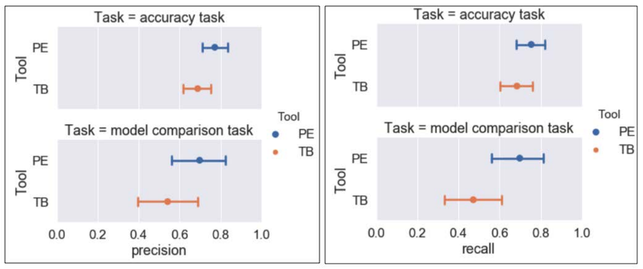


Figure 4: Precision vs. Recall for model comparison and accuracy tasks.

The post-questionnaire for the second tool included an extra set of questions asking participants for their preferences on the two tools for both model comparison and accuracy estimation tasks.

6 RESULTS

6.1 H1: Analyzing Difference in Participants' Performance

To compare the participants' performance between *Parallel Embeddings* and *TensorBoard Embedding Projector*, we calculated the precision and recall per participant for the accuracy estimation (AT) and model comparison (CT) tasks. To present our analysis results, we followed the advice from HCI literature on best practices to present effect size [6] and used a non null-hypothesis statistical testing approach focusing on sample means and bootstrapped 95% confidence intervals (C.I.s). Figure 4 shows bootstrapped confidence interval plots for both tools with model comparison and accuracy tasks separately. Overall, the results showed participants are more likely to achieve higher precision and recall with *Parallel Embeddings* in comparison to *TensorBoard Embedding Projector*, with the recall of the model comparison task being significantly different between the two tools.

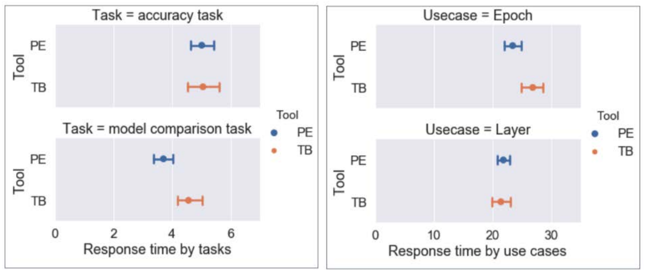


Figure 5: Response time by tasks and use cases: *Parallel Embeddings* Vs. *TensorBoard Embedding Projector*.

In addition to analyzing differences in accuracy, we also investigated differences in response time between the two tools. Overall, for the epoch use case (during which the users completed both accuracy and model comparison tasks), participants using *Parallel Embeddings* took significantly less time (Figure 5 right). The response time for the layer use case did not exhibit significant differences between the two tools. Across different tasks, the response time for the model comparison sub-task is significantly lower for *Parallel Embeddings* (Figure 5 left). Our analysis results showed *Parallel Embeddings* yielded better accuracy and response time in the model comparison task and the epoch use case.

Overall, our findings on accuracy and response time support **H1**. Moreover, the performance gain in accuracy with *Parallel Embeddings* did not come at the cost of longer response time.

6.2 H2: Analyzing Participants' Strategies

To identify user strategies, we adopt an open coding process followed by selective coding focusing on concepts related to strategies [29]. We observed the performance and analytical process for the first tool influenced the overall strategy, and the participants are likely to employ a similar process for the second tool they experienced. Overall, the participants spent more time to complete the same tasks with the first tool (avg *27 minutes 57 seconds*) compared to the second tool (avg *18 minutes 47 seconds*). We suspect the reduction in completion time with the second tool is due to the participants being familiar with the data and tasks after the first tool. To control for the impact of the possible learning effect on task completion time, we further analyzed the response time when using either *Parallel Embeddings* and *TensorBoard Embedding Projector* as the first tool. We observed patterns similar to those already presented in Figure 5; participants completed the tasks faster with *Parallel Embeddings*. We also observed *Parallel Embeddings* is more intuitive and easier to start with to understand what concepts are captured by the embeddings and how this relates to the metadata. This observation is supported by less training time and fewer clarification questions with *Parallel Embeddings*.

We observed participants' strategies not only depend on the first tool they used, but also vary by treatments (Epoch vs. Layer); Table 1 presents a summary of participants' strategies. We noticed that when the participants see *Parallel Embeddings* first, they are more likely to start by examining multiple clusters based on their accuracy while participants using *TensorBoard Embedding Projector* first are more likely to dive into one particular cluster at a time. We summarize the overall strategy derived from *Parallel Embeddings* as "overview first, individual clusters next", and the strategy derived from *TensorBoard Embedding Projector* as "one cluster at a time".

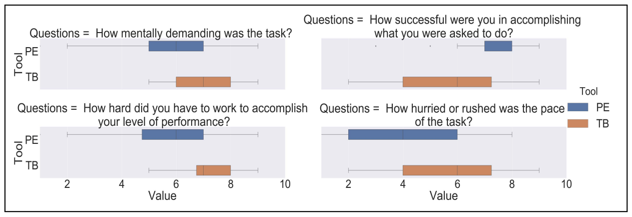


Figure 6: NASA TLX ratings for *TensorBoard Embedding Projector* and *Parallel Embeddings.*

For example, one participant who started with *Parallel Embeddings* inspected the clusters with the darkest color and verified the distribution of different breeds in those clusters. The participant then did a downstream analysis by increasing the cluster numbers to divide the clusters to smaller number of breeds as much as possible to do a breed-wise analysis. This participant then followed a similar process using *TensorBoard Embedding Projector* by looking into neighbors via neighborhood search within different clusters.

Table 1: Top Strategies for use cases and *Parallel Embeddings* (PE) vs. *TensorBoard Embedding Projector* (TF).

Groups	Top Strategies
First tool vs. Second tool: Epoch use-case	**First tool (PE)**: Clicking between the clusters, dog types, and filters to see how things would change. **Second tool (TF)**: Starting with the inspector panel to search for proper classification and misclassification for left and right layer/epoch's to make a judgement of the accuracies for different breeds. And then clicking on different instances frequently to see the closest neighbors and to find out how similar/dissimilar they are with the selected instance.
	First tool (TF): Counting the number of misclassified instances for each breed one by one for the both epochs/layers. **Second tool (PE)**: Looking into the clusters which exclusively clustered just one breed with darker color (higher classification accuracy), and exploring the clusters which did not classify that breed successfully and then observing the classification errors for each cluster
First tool vs. Second tool: Layer use-case	**First tool (PE)**: Checking the distribution of the metadata for a selection (either cluster or line) or for clusters highlighted with the color filters. **Second tool (TF)**: Using the distance of nearest neighbors feature in the inspector panel.
	First tool (TF): Looking specifically to the nearest neighbors of a misclassified dog to find out if its neighbor and their classification is correct or not. **Second tool (PE)**: Checking the cluster color and opacity to identify which breeds were classified correctly by taking additional help from the distribution of the metadata.

Another participant mentioned that he started looking into breed one at a time in the *TensorBoard Embedding Projector* to find out the accuracy by breed and then proceeded into analyzing different instances to find out their association with nearest breeds with cosine or euclidean similarities in the clusters. He applied the same strategy with *Parallel Embeddings* as the second tool. In *Parallel Embeddings*, he started with selecting each breed from the "Dog breed" bar charts group and then looked into the distribution for darkest clusters in left/right frames. We observe the participants' overall strategies are heavily influenced by the first tool they experience.

In addition to presenting the summary and individual strategies, we plotted responses from the post-questionnaire which employed the NASA TLX on the stress level and efficiency of the participants in Figure 6. The results showed that participants tend to report the tasks as less demanding and perceive their performance as more successful with *Parallel Embeddings*. The results also showed the participants thought they needed to work harder to complete the tasks with *TensorBoard Embedding Projector*. These findings support **H2**, that *Parallel Embeddings* facilitates discovery of more efficient strategies and makes the overall experience less demanding.

7 DISCUSSION

7.1 Interpretation of Results

Participants were faster and more accurate on the model comparison task with *Parallel Embeddings* compared to *TensorBoard Embedding Projector*. We can account for these differences as follows. The *Parallel Embeddings* tool was explicitly designed to contrast multiple models compared to *TensorBoard Embedding Projector* which is intended for exploring a single model in detail. For example, *Parallel Embeddings* abstracts clusters of examples into fewer points which can be more easily interpreted. The tool also shows exactly what changes between clusters. Using *TensorBoard Embedding Projector*, users would need to flip back and forth between views and perform manual lookups of examples to find the same example in both views. Furthermore, *TensorBoard Embedding Projector* contained extra features which served to increase the complexity of the interface and potentially distracted users. This could account for the increased learning time and time on task.

When analyzing the rating of each visual component collected through the post-questionnaire, participants reported all visual components and interactive features as equally helpful or very helpful for *Parallel Embeddings*. In comparison, almost all of the participants found the inspector panel was the most helpful feature in *TensorBoard Embedding Projector* with other views less helpful for the tasks. This is not surprising considering each of the features in *Parallel Embeddings* were designed to support more challenging model comparison. Using *TensorBoard Embedding Projector*, participants used a sub-set of the features, and had to rely the inspector panel for multiple uses, including discerning cluster meaning.

Interestingly, when *Parallel Embeddings* was experienced as the second tool in the user experiments, participants appreciated the storylines more than the participants who used *Parallel Embeddings* first. Some of them mentioned that since they struggled with model comparison task in *TensorBoard Embedding Projector*, and were unsure of how to start with finding out the difference. They thought *Parallel Embeddings* gives a good starting point with the storylines for identifying disagreements in two different frames quickly. This is an indication that further abstracting cluster differences provided a helpful overview visualization.

In addition, some participants were excited about the images in *Parallel Embeddings* and said the images for a selected cluster provide visual information on the dog breeds they are not familiar with. While the images were also available in *TensorBoard Embedding Projector*, they may have been too small in the tool to be usable. However, some participants thought the images in *Parallel Embeddings* were less helpful and wanted to know the breeds to explore the images in a cluster. This could be addressed by visually encoding metadata, e.g. dog breed, onto these example images so this information is visible in the context of the images, similar to how this information is available in *TensorBoard Embedding Projector*.

7.2 Limitations & Possible Improvements

The number of cohorts visible in the tool can increase rapidly as additional frames are added. In the worst case, when there are cohorts between every pair of c clusters across f frames, the number of cohorts is $O(f^c)$. However, this number is also bounded by the total number of samples, as samples can only belong to a single cohort, and cohorts must have at least one sample. In practice the number of cohorts depends on how different the adjacent model embeddings are — more similar models will lead to less cluttered visualizations when more frames are present.

While the routing of cohorts' paths around clusters helps reduce clutter, the routing technique causes paths to sometimes overlap. An improvement would be to bias path to also avoid overlapping an already routed pathway. Finally, while the use of cohorts abstracts the visualization and removes clutter, sometimes this can be misleading. We plan to incorporate a view which allows the user to see all of the samples in a cluster. This allows the user to better understand how well the clustering reflects the underlying distribution of samples in the embedding, letting the user more effectively choose the number of clusters.

Our evaluation of *Parallel Embeddings* has limitations as well. For experimental control, we chose *TensorBoard Embedding Projector* as a comparison condition to measure the primary contribution of our approach — integrating the side-by-side views of the learned representations. While the experiment produced evidence that this was a valuable contribution, our experiment did not address whether our tool was more useful than other performance- or concept-oriented model comparison tools. Future work that makes this direct comparison would be valuable. Other experimental designs that more directly assess users' mental models of the learned representations while using the tool would also be beneficial. This would be a significant contribution beyond the tasks used in our experiment.

8 CONCLUSION

Data scientists and decision makers need tools that allow them to compare models beyond coarse performance metrics like F-score. Existing tools like *TensorBoard Embedding Projector* target data scientists as users, allowing them to explore the functionality of a particular layer in a single network. However, these tools do not directly support concept-oriented model comparison. To address this, we designed *Parallel Embeddings* to contrast sequences of learned representations and applied it to two model comparison use cases. We evaluated our tool's ability to help data scientists understand how two models interpret the same data differently. We found users were more efficient and more accurate with *Parallel Embeddings* for these tasks, in part because our tool was designed with these more challenging tasks in mind. *Parallel Embeddings* might be used more generally for use cases outside machine learning, for example, as an alternative to animated scatter plots [32].

ACKNOWLEDGMENTS

The research described in this paper was conducted under the Laboratory Directed Research and Development Program at Pacific Northwest National Laboratory, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy.

REFERENCES

[1] Eric Alexander and Michael Gleicher. 2015. Task-driven comparison of topic models. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2015), 320–329.

[2] 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*. ACM, 337–346.

[3] Dustin Arendt and Meg Pirrung. 2017. The "y" of it Matters, Even for Storyline Visualization. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 81–91.

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. 2017. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6541–6549.

[5] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[6] Pierre Dragicevic. 2016. Fair statistical communication in HCI. In *Modern Statistical Methods for HCI*. Springer, 291–330.

[7] Tim Dwyer and Lev Nachmanson. 2009. Fast edge-routing for large graphs. In *International Symposium on Graph Drawing*. Springer, 147–158.

[8] Niklas Elmqvist and Jean-Daniel Fekete. 2009. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics* 16, 3 (2009), 439–454.

[9] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2962–2970.

[10] Kārlis Freivalds. 2001. Curved edge routing. In *International Symposium on Fundamentals of Computation Theory*. Springer, 126–137.

[11] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in Psychology*. Vol. 52. Elsevier, 139–183.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 770–778.

[13] Julian Heinrich and Daniel Weiskopf. 2013. State of the Art of Parallel Coordinates.. In *Eurographics (STARs)*. 95–116.

[14] Fred Matthew Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. 2018. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics* (2018).

[15] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. 2018. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281* (2018).

[16] Jimmy Johansson, Patric Ljung, Mikael Jern, and Matthew Cooper. 2005. Revealing structure within clustered parallel coordinates displays. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*. IEEE, 125–132.

[17] Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. 2017. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2017), 88–97.

[18] Minsuk Kahng, Dezhi Fang, and Duen Horng Polo Chau. 2016. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 1.

[19] Wiebke Köpp and Tino Weinkauf. 2018. Temporal treemaps: Static visualization of evolving trees. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 534–543.

[20] Antoine Lhuillier, Christophe Hurter, and Alexandru Telea. 2017. State of the art in edge and trail bundling techniques. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 619–645.

[21] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology* (1932).

[22] Dongyu Liu, Weiwei Cui, Kai Jin, Yuxiao Guo, and Huamin Qu. 2018. Deeptracker: Visualizing the training process of convolutional neural networks. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 1 (2018), 6.

[23] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. 2016. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 91–100.

[24] Jonas Lukasczyk, Gunther Weber, Ross Maciejewski, Christoph Garth, and Heike Leitte. 2017. Nested tracking graphs. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 12–22.

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

[26] Kevin T McDonnell and Klaus Mueller. 2008. Illustrative parallel coordinates. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1031–1038.

[27] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[28] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*. ACM, 1222–1230.

[29] Paul Mihas. 2019. Qualitative data analysis. In *Oxford Research Encyclopedia of Education*.

[30] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. 2016. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 101–110.

[31] Khairi Reda, Chayant Tantipathananandh, Andrew Johnson, Jason Leigh, and Tanya Berger-Wolf. 2011. Visualizing the evolution of community structures in dynamic social networks. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 1061–1070.

[32] Hans Rosling and Zhongxing Zhang. 2011. Health advocacy with Gapminder animated statistics. *Journal of epidemiology and global health* 1, 1 (2011), 11–14.

[33] Martin Rosvall and Carl T Bergstrom. 2010. Mapping change in large networks. *PloS one* 5, 1 (2010), e8694.

[34] Emily Saldanha, Brenda Praggastis, Todd Billow, and Dustin L. Arendt. 2019. ReLVis: Visual Analytics for Situational Awareness During Reinforcement Learning Experimentation. In *EuroVis 2019 - Short Papers*, Jimmy Johansson, Filip Sadlo, and G. Elisabeta Marai (Eds.). The Eurographics Association.

[35] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[36] Daniel Smilkov, Nikhil Thorat, Charles Nicholson, Emily Reif, Fernanda B Viégas, and Martin Wattenberg. 2016. Embedding projector: Interactive visualization and interpretation of embeddings. In *Proc. Neural Inf. Process. Syst. Workshop Interpretable ML Complex Syst.*

[37] Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M Rush. 2018. Seq-2-seq-vis: A visual debugging tool for sequence-to-sequence models. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 353–363.

[38] C. J. Van Rijsbergen. 1979. *Information Retrieval (2nd ed.)*. Butterworth-Heinemann.

[39] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. 2017. Visualizing group structures in graphs: A survey. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 201–225.

[40] Junpeng Wang, Liang Gou, Han-Wei Shen, and Hao Yang. 2018. Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 288–298.

[41] John Wenskovitch, Ian Crandell, Naren Ramakrishnan, Leanna House, and Chris North. 2017. Towards a systematic combination of dimension reduction and clustering in visual analytics. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2017), 131–141.

[42] Wathsala Widanagamaachchi, Cameron Christensen, Valerio Pascucci, and Peer-Timo Bremer. 2012. Interactive exploration of large-scale time-varying data using dynamic tracking graphs. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 9–17.

[43] Moritz Wittenhagen, Christian Cherek, and Jan Borchers. 2016. Chronicler: Interactive exploration of source code history. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 3522–3532.

[44] Wei Yu, Kuiyuan Yang, Yalong Bai, Hongxun Yao, and Yong Rui. 2014. Visualizing and comparing convolutional neural networks. *arXiv preprint arXiv:1412.6631* (2014).

[45] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. 2016. Graying the black box: Understanding DQNs. (2016), 1899–1908.

[46] Haipeng Zeng, Hammad Haleem, Xavier Plantaz, Nan Cao, and Huamin Qu. 2017. CNNComparator: Comparative analytics of convolutional neural networks. *arXiv preprint arXiv:1710.05285* (2017).

[47] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. 2018. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 364–373.

A SUPPLEMENTAL MATERIALS

A.1 Additional Demonstration

Figure A.1 demonstrates how the two adjacent layers of a simple convolutional neural network changes towards correct classification. We developed a simple convolutional network for classifying ten classes of the MNIST handwritten digits dataset [8] dataset. We visualized the learned representation (neural activation) of the two adjacent dense layers of the network with *Parallel Embeddings*. As expected, we found the learned representations differ slightly between the adjacent layers — though the apparent number of clusters remains the same, their memberships differ.

For example, in the earlier layer (See Figure A.1.a), the digit "two" clustered with mostly the digits "seven", "nine" and "one". Eventually in the later layer (See Figure A.1.b), the other digits are being separated from the digit "two" and mostly being clustered with themselves. The highlighted storyline represents that the cluster membership is not changing in the later layer for the digit "two" .

A.2 Additional Implementation Details

Figure A.2 shows an example modified triangular lattice for the *Parallel Embeddings* path routing algorithm. The routing follows a triangular lattice except where modified by the presence of clusters. Dislocations in the lattice are due to the removal of regular lattice points and the addition of cohort centroids.

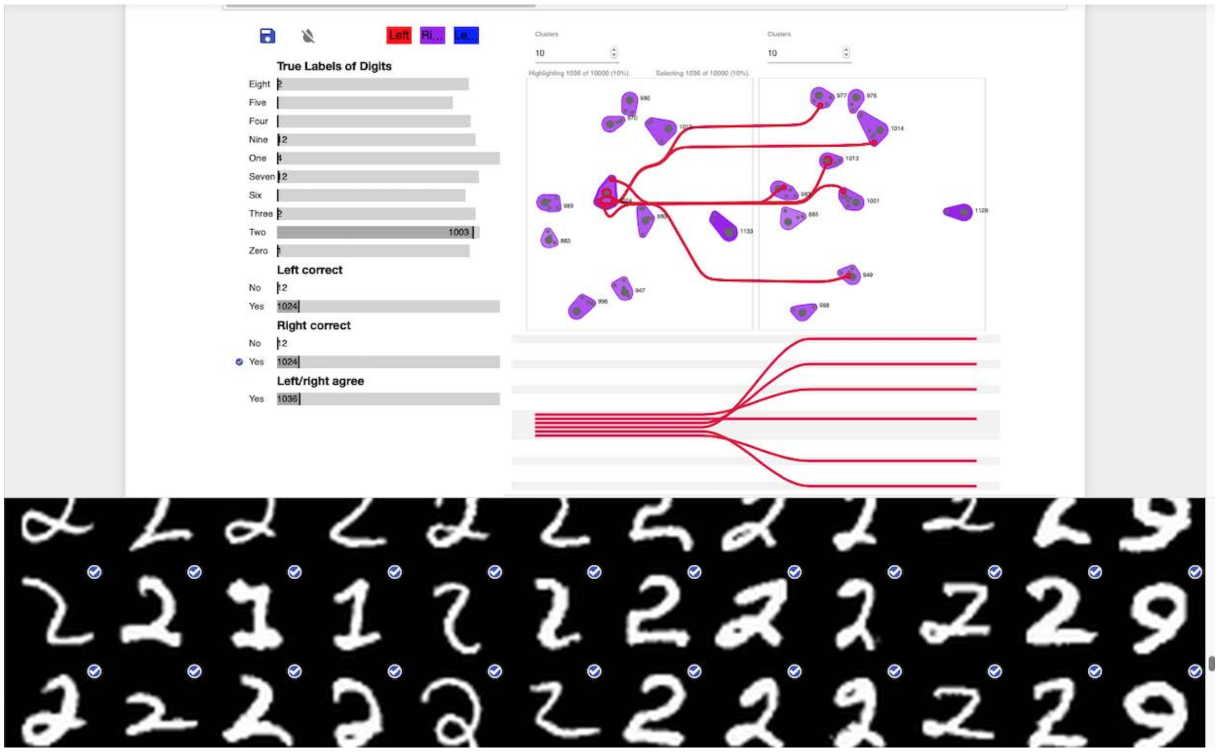
Figure A.3 shows the integration of the *Parallel Embeddings* tool with a Jupyter Notebook. It also highlights the minimal code needed to pass learned representations to the tool. For scalability reasons, *Parallel Embeddings* is most effective for comparing two learned representations. However, the tool is capable of arbitrary n−way comparison. In Figure A.4 shows *Parallel Embeddings* with a four-frame comparison for a simple dataset.

A.3 Additional Study Details

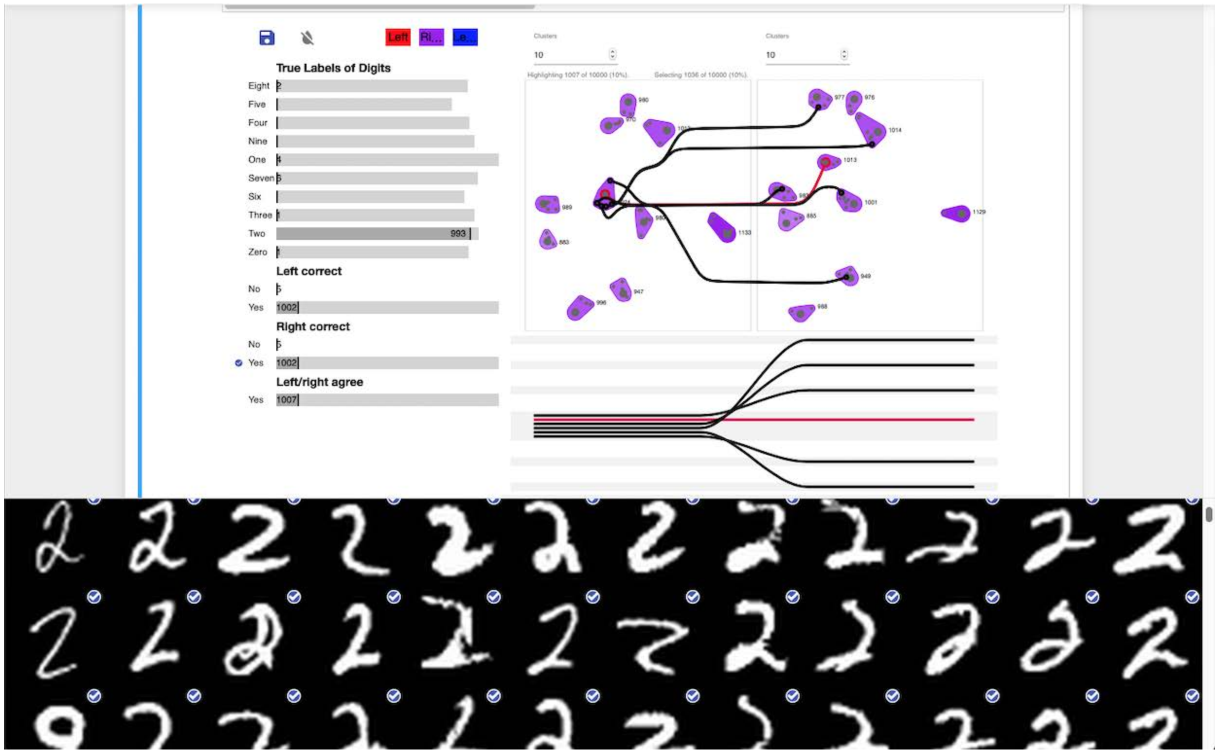
Figure A.5 shows what the dataset from the user study looked like in the *TensorBoard Embedding Projector* comparison condition. We used a dual monitor setup to let participants interact with the two embeddings. The user can choose the data columns to color from the “color by” menu and to label the instances for hovering from the “label by” menu in the *Data Panel*. For example, if the user wants to label by “Dog breed”, while hovering it will show the name of the breeds.

In the *Projection Panel*, the user can select a number of “Neighbors” to do the projection and then hit the “Run” button. For this user study, we only used “UMAP” and 3D projection. In the *Inspector Panel* on the right side, the user can search for particular metadata and see list of nearest neighbors selected on the projection panel. The user can also inspect nearest-neighbor subsets. Clicking on a point causes the right panel to list the nearest neighbors, along with distances to the current point.

[8]MNIST Dataset: http://yann.lecun.com/exdb/mnist/

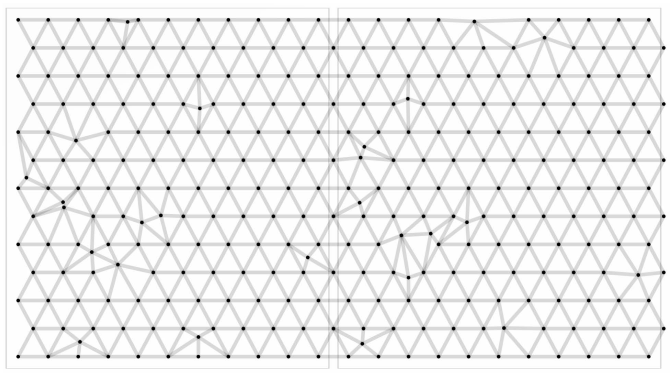


(a) One cluster in the left layer containing several classes of digits splits into many clusters in the right layer.

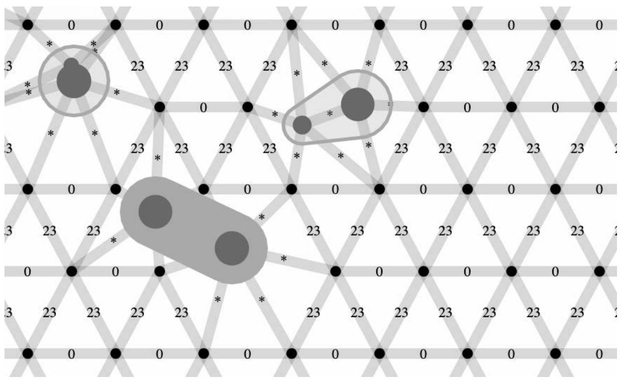


(b) Inspection of a particular cohort from the splitting cluster shows it to be more coherent than the original left cluster.

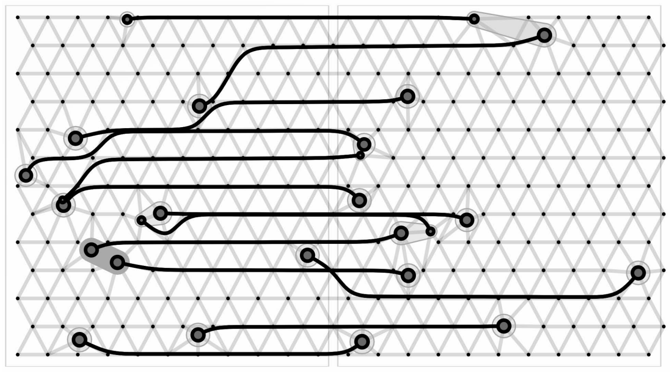
Figure A.1: MNIST dataset: exploring layers of a simple convolutional network (a) highlights transition of cluster membership from earlier to later layers of the network. (b) highlights a cluster with a highest membership of digit "two" and the corresponding metadata, demonstrating that the second dense layer is further separating instances by their class.



(a) path routing graph

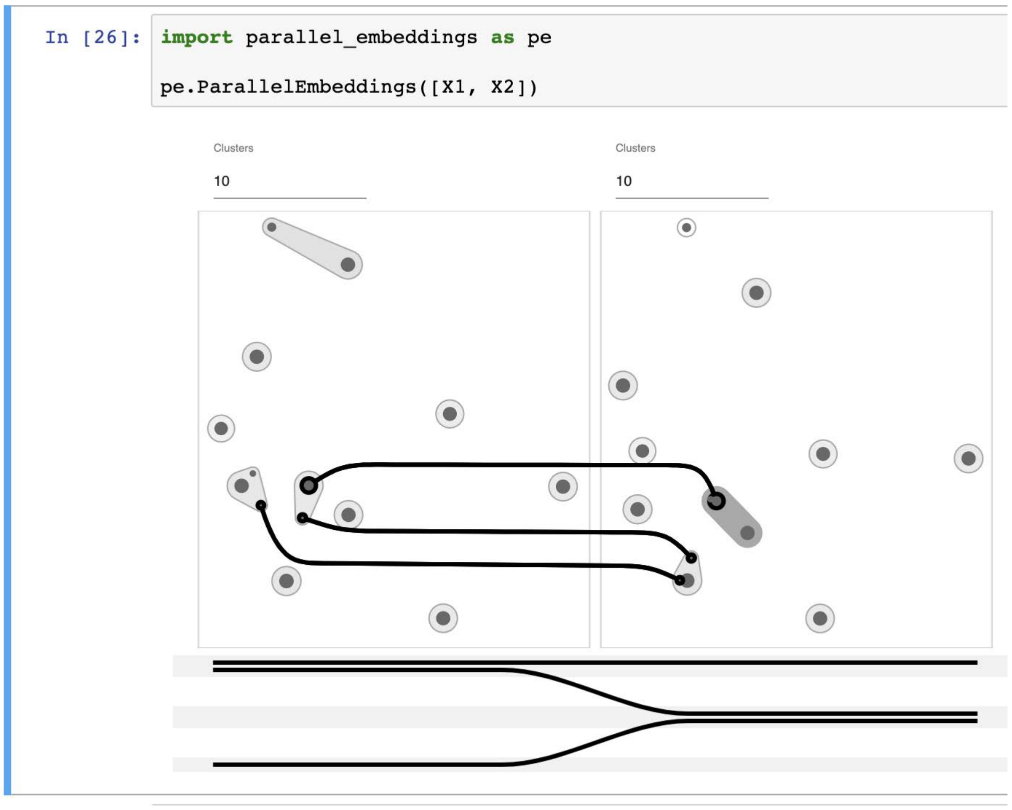


(b) weighted lattice



(c) actual paths with routing graph

Figure A.2: Path routing. (a) shows the control point graph, a modified triangular lattice. (b) shows the lattice edge weights (arbitrarily high weights are denoted with a *) (c) shows the cohort paths routed between clusters in the context of the control point graph.



```
In [26]: import parallel_embeddings as pe

         pe.ParallelEmbeddings([X1, X2])
```

Figure A.3: ***Parallel Embeddings*** **integration with Jupyter Notebooks. The tool is simple to instantiate and renders and interactive visualization below the code block.**

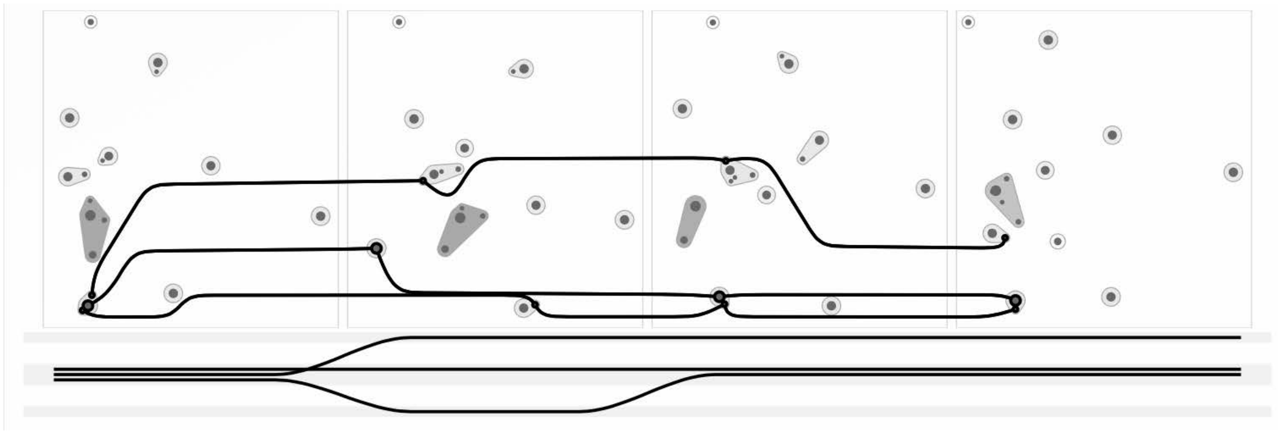
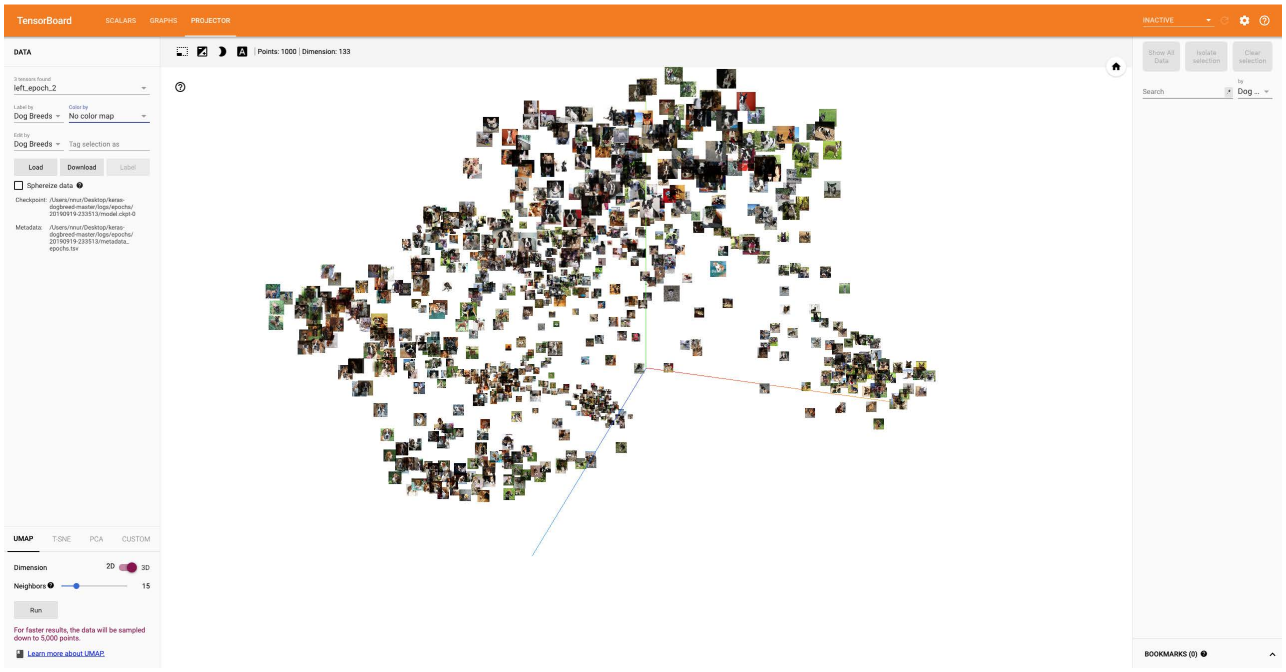


Figure A.4: A demonstration of ***Parallel Embeddings*** **with more than two frames.**



3 tensors found: left_epoch_2
Label by: Dog Breeds
Color by: No color map
Edit by: Dog Breeds
Tag selection as
Sphereize data
Checkpoint: /Users/nnur/Desktop/keras-dogbreed-master/logs/epochs/20190919-233513/model.ckpt-0
Metadata: /Users/nnur/Desktop/keras-dogbreed-master/logs/epochs/20190919-233513/metadata_epochs.tsv
Points: 1000 | Dimension: 133
Dimension 2D 3D
Neighbors 15
For faster results, the data will be sampled down to 5,000 points.

Figure A.5: Example of the *TensorBoard Embedding Projector* comparison condition used in the study.