# Interactive Weight Visualizaiton Tool for Feature Weight Analysis

Lyric Li and Anni Zheng

**Abstract**—Traditional machine learning models excel at predicting outcomes but struggle to explain the mechanism behind their predictions, often because the models are either too complex, act as black boxes, or are too stochastic. A tool that enhances the interpretability of machine learning models is crucial to aid in explaining and building credibility. This project designed a toolkit that provides a visual portal for people in various fields to analyze their models. It focuses on interpreting the information carried by weight matrices in TensorFlow binary classification models, regardless of their complexity. The project also included a case study where a Video Vision Transformer model (ViViT) was used to classify videos as depicting either a car crash or a normal scenario. The toolkit offers insights into the significance and scale of importance of each feature used in classification, along with the model's accuracy and loss through TensorFlow's callback mechanism. The toolkit features an interactive user interface within the Jupyter Notebook environment and provides three kinds of visualizations, drawn using Matplotlib, to offer dynamic, real-time, and transparent evaluations of the model's classification process.

**Index Terms**—Computer Vision, Data Visualization, TensorFlow Visualization, Feature Importance Analysis

## 1 Introduction

Inspecting the behavior of machine learning models presents a significant challenge due to the many layers and complex mathematical operations processed during training. This challenge is particularly heightened for recent popular models such as Video Vision Transformers, where layers are applied to input data before the training stage. This layering of embeddings, each with a specific role in feature definition, extraction, and transformation, increases data dimensions for the machine to assess but also makes it more obscure for humans to understand, which would hinder our thought process in determining whether a model is logical enough to be published. Fortunately, dataset information remains accessible to users, as does model training information, such as weight values. Both are valuable in explaining how the model's decision-making process is created: weights outline the process, while data justify and improve it. In any case, deciphering the logic behind a machine learning model is important to boost transparency in how the model functions, aiding in demonstrating its potential to convince others of its capabilities.

While numerous visualization tools are available to assess the capability of these complex models, high accuracy does not always equate to logical decision-making. Explaining model results is crucial, particularly when they impact human safety, such as in road safety. More specifically, accurately distinguishing between crash and normal driving scenarios in autonomous driving and driver assistance systems is a challenge with safety implications and models need to be logical enough to capture what makes a scene dangerous. Since the car driving scenario is often in video format, Video Vision Transformer could be used as one of the models for capturing dangerous interactions. However, as mentioned above, the transformer model is complicated. To resolve partial complexity of this kind of model, a visualization tool was developed in this project to improve the interpretability of any TensorFlow binary classification model so researchers can better understand the logic behind it, allowing users to visualize and interpret model behavior across different datasets and problem domains.

The primary contributions of our project include:

- Universal Application: Our visualization toolkit is compatible with any TensorFlow binary classification model, allowing it to serve a wide user base across different fields. This universal design underlines the toolkit's versatility and broad applicability.
- Interactive Visualization Interface: Utilizing TensorFlow's callback mechanism, Matplotlib, and Jupyter Widgets, the toolkit offers a dynamic interface that provides real-time insights into the model's decision-making process during training. This interactivity is pivotal for users who need to understand and possibly intervene in how models evolve and learn from their data.
- Detailed Feature Analysis: The toolkit includes sophisticated tools for deep analysis of feature significance and model metrics over training epochs. Users can explore how specific features influence model predictions, assess feature importance, and detect biases, facilitating a deeper understanding of the model's internal mechanics.
- Enhanced Model Insights: Through visualizations such as t-Distributed Stochastic Neighbor Embedding (t-SNE) graphs and feature tendency charts, the package aids in evaluating the effectiveness of the model in separating classes. These tools are crucial for users to refine models, enhance accuracy, and ensure robustness.

Our visualization tool helps translate complex data patterns into actionable insights, enabling users to make more informed decisions about their models. By offering ways to visualize, analyze, and interpret model behaviors, our toolkit improves transparency and fosters trust in machine learning applications, providing a valuable contribution to the field of data science and machine learning.

## 2 Related Work

### 2.1 ViViT: A Video Vision Transformer

The Video Vision Transformer (ViViT) model, introduced by Arnab et al. (2021), represents a significant advancement in video classification through the application of transformer architectures, which were initially popularized in the field of natural language processing by Vaswani et al. (2017).

Transformers were first adapted for image recognition tasks as demonstrated by Dosovitskiy et al. (2020) in their groundbreaking work with Vision Transformers (ViT). ViT marked a departure from conventional convolutional neural networks (CNNs) by applying a pure transformer approach to sequences of image patches. Building on this foundation, ViViT extends the transformer model to video classification by introducing mechanisms to handle the spatio-temporal nature of video data effectively.

ViViT innovates on ViT by addressing the unique challenges posed by video data, which includes not just spatial but also temporal dimensions. The paper discusses various strategies for tokenizing video input, such as uniform frame sampling and tubelet embedding, which are critical for capturing dynamic information over time. These methods allow the model to maintain high levels of accuracy while managing the increased computational demands of video data.

Our project builds upon the framework established by ViViT, adapting its transformer-based approach to develop a specialized visualization tool for TensorFlow models. This tool aims to make the complex

dynamics captured by models like ViViT accessible and interpretable to users, facilitating deeper insights into model behaviors and enhancing the transparency of automated decision-making processes in applications such as autonomous driving.

By situating our work within the context of these developments, we highlight our contribution to the broader field of data visualization and model interpretability.

reference(to do): https://ar5iv.labs.arxiv.org/html/2103.15691

### 2.2 Efficient Feature Significance and Importance

The article by Kay G. and Enguerrand H. underscores the critical need for quantifying feature importance and develops various methodologies to assess the significance of features within complex predictive models. Their work contributes to a deeper understanding of how different features influence model outcomes, an essential aspect of both model optimization and interpretation.

- Statistical Feature Significance: The article from ar5iv on first-order significance testing for feature importance introduces a rigorous statistical framework to evaluate the significance of individual features in a model. This method is particularly adept at highlighting that many features in high-dimensional datasets do not contribute equally or significantly to the predictive accuracy. By employing a controlled testing environment that adjusts for false discovery rates, this approach ensures that only features with a statistically significant impact on model outcomes are considered important.
- Permutation Importance: Complementarily, permutation feature importance, as described by Fisher, Rudin, and Dominici (2019), provides a model-agnostic metric that assesses the increase in prediction error when the values of each feature are shuffled. This method, widely used due to its simplicity and effectiveness, directly measures the impact of scrambling a feature on the accuracy of the model, highlighting features that are crucial for making accurate predictions.
- Shapley Values: Similarly, the use of Shapley values to assign importance to features in a model highlights the uneven impact of features. By calculating the marginal contribution of each feature across all possible combinations, this method effectively demonstrates that some features play critical roles in prediction, whereas others may have negligible effects.
- Regularization Techniques: Techniques like Lasso regression also emphasize the disparity in feature importance by penalizing the coefficients of less important features, often driving them to zero. This method not only helps in reducing model complexity but also clearly delineates which features are crucial for the model's predictions, highlighting the unequal distribution of feature significance.

The work reviewed in the cited article reinforces the concept that understanding which features significantly affect the predictions is crucial for creating efficient and interpretable models. It addresses the challenge of feature selection in high-dimensional data, where it becomes even more critical to identify and focus on features that genuinely matter, thus avoiding the noise and redundancy that can lead to overfitting and obscure model interpretation.

### 2.3 SHAP Values on ImageNet

An important aspect of model interpretability in machine learning, especially in the context of deep learning, is understanding which inputs most influence model predictions. An illustrative example of this is the application of SHAP (SHapley Additive exPlanations) values to interpret the predictions of a pre-trained ResNet50 model. Originally developed by Lundberg and Lee (2017), SHAP values provide a robust theoretical framework derived from cooperative game theory to explain the output of any machine learning model.

ResNet50, a widely-used convolutional neural network that has been trained on the ImageNet dataset to classify images into one of 1,000 categories, serves as an excellent model for such analysis due to its complex architecture and broad application in image recognition tasks. The process typically involves:

- Loading the ResNet50 model pre-trained on ImageNet.
- Selecting a subset of images for explanation.
- Applying a masker to blur parts of the image not under evaluation to isolate the effects of specific regions.
- Utilizing SHAP's Partition Explainer to compute SHAP values, which quantitatively measure the impact of different image regions on the model's output.

Visualization of these SHAP values can be achieved using tools like shap.image_plot, which creates plots displaying how different segments of the image contribute to the model's predictions for each classification label. This visualization aids in the practical demonstration of integrating SHAP values into deep learning workflows, enhancing transparency and interpretability, and thereby facilitating easier diagnosis and understanding of model behaviors.

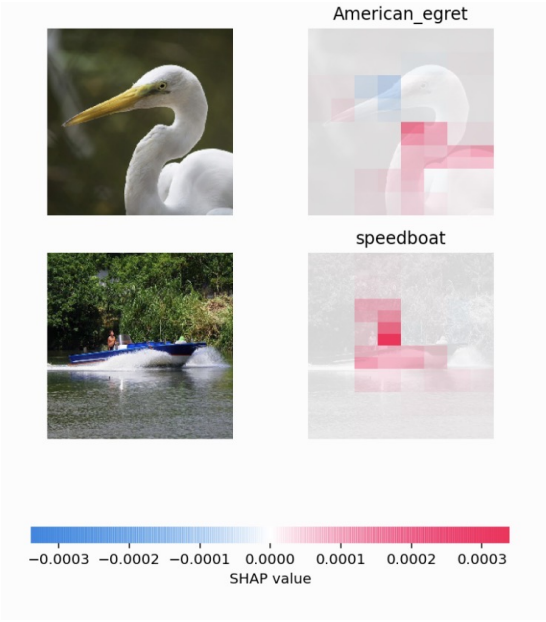



Fig. 1: SHAP Values on ImageNet

### 2.4 Visual Analytics for Interpreting Predictive ML Models

The field of machine learning model interpretability has increasingly emphasized the need for visualization tools that can demystify the decisions made by complex models. The article by [Author(s) Name(s)] (2016) on arXiv provides a comprehensive review of visualization techniques aimed at enhancing the interpretability of machine learning models. This paper serves as a crucial reference in understanding the evolution and application of various interpretative techniques in the field.

The article discusses several key approaches to visualizing the behavior of machine learning models, particularly focusing on black-box models where the relationship between input and output is not readily apparent. These visualization techniques are designed to aid in the exploration of how models process input data and make predictions, which is essential for validating model accuracy and fairness.

Model Interpretation through Visualization of Input/Output Behavior: One of the primary methods outlined involves the direct visualization of the input-output relationships. This approach is valuable as it does not depend on the internal mechanics of the model, thus offering a flexible and generic way to assess any model. Such visual analytics tools help reveal how changes in input features influence the predictions, providing insights into the model's dependency on specific features.

Prospector – A Case Study: The paper introduces 'Prospector', a visual analytics system that exemplifies the practical application of these concepts. Prospector allows users to interactively explore the influence of individual features on model predictions. This system utilizes partial dependence plots to illustrate how prediction scores change with feature values, enabling a deeper understanding of feature importance and model behavior.

Impact of Visualization on Model Transparency: The article emphasizes that visualization tools not only aid in interpreting model predictions but also play a crucial role in identifying biases and errors in models. By enabling the interactive exploration of how models respond to changes in input data, these tools help ensure models operate as intended and adhere to ethical guidelines.

This review underscores the importance of developing robust visualization tools that can cater to the growing complexity of machine learning models. Our project builds upon this foundation by offering a toolkit that enhances the interpretability of TensorFlow binary classification models, addressing the specific needs highlighted in the field of autonomous driving and beyond.

By situating our work within the context of these developments, we aim to contribute significantly to the broader field of data visualization and model interpretability, ensuring our tools meet the evolving demands of machine learning applications.

## 3 Background

This project incorporates a case study for testing its visualization tool. The dataset this case study used was the Car Crash Dataset (CCD)provided by a Multimedia conference in 2020 in Association for Computing Machinery (ACM). It is a collection specifically designed for traffic accident anticipation. The datasets comprises of 4,500 videos, each 5 seconds long, categorized into two distinct classes: 3,000 videos depict normal driving circumstances, while 1,500 showcase crash situations. This diverse dataset provides a comprehensive foundation for training and evaluating our Vision Transformer (ViT) model used in case study in differentiating between "Crash" and "Normal" driving events.


Fig. 2: One of the original image-frames from a video in the Dataset

The primary design goals of our visualization tool are centered around enhancing the interpretability and insightfulness of machine learning models, specifically those applied to the complex task of video-based driving scenario classification. Here are the key insights we aim to provide through our visualization designs:

- Highlighting Critical Features: Our tool aims to identify and visually highlight the features within the data that are most influential in determining the model's classification decision. This is also crucial for understanding what aspects of the driving scenario the model focuses on when distinguishing between 'crash' and 'normal' classifications.
- Bias and Influence Analysis: We plan to highlight potential biases in model predictions and the disproportionate influence of certain abstract features. Identifying these biases is essential for developing fairer and more robust models.
- Interaction with Model Predictions: By allowing users to interact with the model's predictions — for example, by adjusting thresholds for classification or selecting different segments of the video for detailed analysis — our visualizations aim to provide a hands-on experience that deepens users' understanding of the model's functioning and reliability.

Through these design goals, our visualization tool seeks to bridge the gap between complex machine learning outputs and practical, actionable insights that can aid in the safe implementation of autonomous driving technologies. This approach not only enhances the transparency of model's decision-making processes but also empowers users to make informed adjustments to improve model performance and reliability.

## 4 Methods

### 4.1 Overview

The Python tool created can be broken down to three parts – the input part, the interface part, and the output part, which the output parts can be interactively controlled via the interface. Both the interface and the output part are meant to be directly seen by users to express interface's interactivity and dynamic changes made inside of the model. The tool would take the model and the model's training information as inputs; then, it would train, test, and validate the model automatically; and finally, it would output an interface for users to analyze the features within the model.

The results displayed are quantitative. The tool is designed to serve as a tool to help users understand the significance of features – their positive or negative tendencies of features in classifying an input to particular class, and how these tendencies change through epochs, and whether any biases exist within the model, meaning whether features influence classification towards one class more than others. Given that models can be quite complex and contain numerous features, interaction techniques like scroll bars are employed to filter information, allowing users to focus on specific phases of the model. The interface is built entirely for TensorFlow and the Python environment using TensorFlow's callback mechanism, Matplotlib, and Jupyter Widgets, so users can easily access it while training.

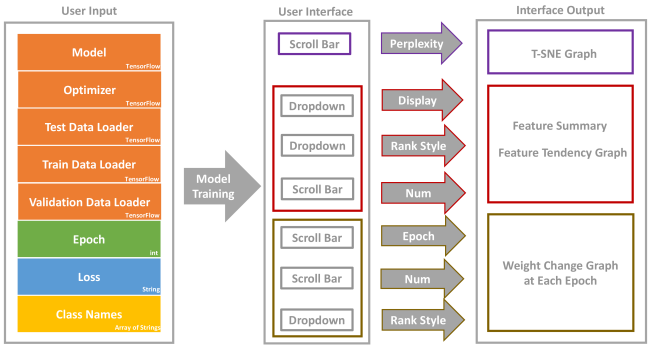



Fig. 3: Interactive Weight Visualization Tool Work Flow

### 4.2 Design Flow

There is only one function, 'run', that can be used directly by users. In this function, users input the TensorFlow model, optimizer, test and train data loaders, the number of epochs for training, and the name of the loss function. Users may optionally input the validation data loader and the class names. Note that the number of class names is limited to two. If users choose not to input the class names, the tool will automatically name them "Class 0" and "Class 1." The model is compiled with the input optimizer, loss function, and both sparse categorical accuracy and sparse top-5 categorical accuracy metrics. The sparse categorical accuracy measures how often the model's top prediction matches the true class label, while the sparse top-5 categorical accuracy measures how often the true class label is found within the model's top five predictions. The first metric provides insights into the model's performance, while the second metric shows how close the model is to

achieving the correct predictions. Together, they determine how well the model performs.

During the training stage, a weight history callback object, provided by TensorFlow Keras, collects the weight of each feature at every epoch, which will later be used to create a weight change visualization that will be introduced subsequently. After a model is fully trained, the weights of the model's final layer, which determine the final result, will be retrieved to create a feature tendency visualization, and the model's accuracy and loss at each stage will be useful in another component of the weight change visualization. Finally, the trained model is tested on the provided test data loader, and the classification results are used to create a T-SNE graph so users can verify if the classifier is making distinctions (i.e., classifying) within the input data.

### 4.2.1 T-SNE Graph

The first visualization a user will see is the t-Distributed Stochastic Neighbor Embedding (t-SNE) graph. t-SNE is a dimensionality reduction technique used for data analysis. It maps high-dimensional data into a lower-dimensional space, in this case, 2-dimensional (2-D), and maintains the data points' local relationships. The model is tested using the test data loader, which then outputs the testing results to the 'TSNE' function provided by the scikit-learn manifold library. The purpose of this t-SNE graph is to demonstrate how separate or close the prediction points output by the binary classification model are. Logically, the more separate the clusters of the two classes, the better the model is at distinguishing between them.

The 'TSNE' function would project the binary class predictions onto the 2-D graph, with Class 0 prediction points colored sky-blue and Class 1 prediction points colored pink. The distinctive colors allow users to visually distinguish between the two types of prediction points, so the more mixed the sky-blue and pink points are, the less capable the model is of distinguishing between the two classes. The tool recognizes that the t-SNE graph is highly impacted by the perplexity value and that it only preserves the local structure of data points. Therefore, a scrollbar controlling the perplexity is provided to users for a more comprehensive analysis of the t-SNE graph. The hidden function in this tool, responsible for controlling the t-SNE graph, will use the value provided by users via the scrollbar to produce the corresponding graph. Note that only the perplexity value can be changed, while the iteration number (300) and the learning rate (200) remain fixed for simplicity.

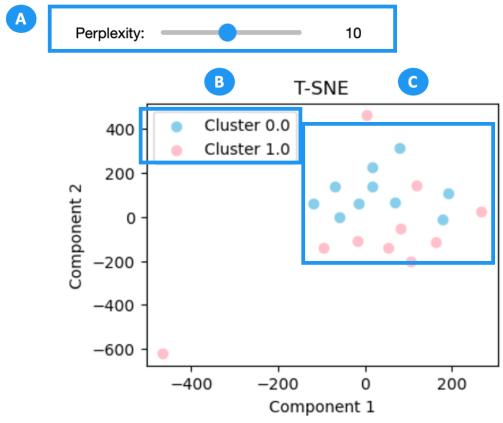



Fig. 4: A T-SNE graph display controlled by a scrollbar. (A) A scrollbar that controls the perplexity of the current T-SNE graph. (B) A legend box indicating which data point belongs to which class. (C) The T-SNE graph indicating the classification result. The more prominent the data points are forming two groups of clusters, the better the classifier is to distinguish between input of two classes.

### 4.2.2 Tendency Graph

The second visualization is the feature tendency graph. This tool defines two types of tendencies. First, a feature's positive or negative contribution to either class is determined by its weight. Second, a feature's greater contribution to one of the classes is determined by comparing the absolute value of its positive or negative contribution to the two classes. Insights drawn from these two tendencies include: firstly, the feature's significance — if a feature positively impacts both classes, then it might not be significant in determining the classification result; second, the scale of the feature's significance—whether a feature contributes positively or negatively, largely or mildly, to the classification of the two classes; third, the correlation of feature significance — whether a feature's positive influence on one class indicates a negative influence on the other.

Users can select one of two ways to map the tendencies via a dropdown menu. The tool will capture the display option and, for each feature, draw a bar graph with the x-axis representing the weights and the y-axis representing the class names or bar placements. The bars will extend to the left to show a negative influence or to the right to show a positive influence. Class 0's bar will be sky-blue, and Class 1's bar will be pink to match the color choice in the t-SNE graph and to visually distinguish the differences. The number of features a user wants to inspect can be adjusted via a scrollbar. Additionally, a "rank style" option, organized by a dropdown menu, allows users to rank the features in ascending or descending order based on their contribution to Class 0 or Class 1 classification. This feature helps users gain further insights into feature significance.

Lastly, there will be a text summary at the top of the user interface that informs users of the total number of features, the number of features that contribute positively or negatively to the two classes, and the maximum and minimum influences on the two classes. From these, one may inspect if a model is potentially "biased." For instance, if the model is contributing more positively towards classifying an video as it is a display of a car crash, and at the same time either less negatively contributing to this classification or more negatively towards classifying an input to another class, then there might be a tendency present in this model of classifying an input to only one class (i.e., it is more likely to classify a video as a car crash video either maybe because the model has went through more training steps on car crash videos or that the model is fundamentally bad).

The interactivity of this interface was achieved by Jupyter Widgets and the creation and styling of the bar graphs were achieved by Matplotlib.

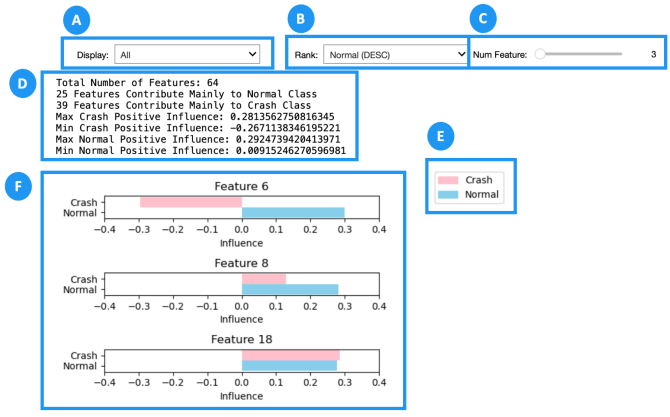



Fig. 5: The second component of this interface is a feature tendency graph. (A) A dropdown menu for selecting the kinds of display. (B) A dropdown menu for selecting by what ways users want to rank the features. (C) A scrollbar of the displayed number of features. (D) A summary of the weights produced by the last layer of the current model. (E) A legend box indicating which colored bar belongs to which class. (F) The feature tendency graph.

#### 4.2.3 Feature Weight Change Graph

The third visualization is the feature weight change graph. The graph has two components: a line graph showing model accuracy and loss from the first epoch to the current epoch as determined by the user, and a heatmap of weights during that epoch for a selected number of features as set by the user. The x-axis of the heatmap represents the class name, while the y-axis shows the corresponding feature for each weight. The colormap used is "coolwarm" from Matplotlib. The larger the weight and greater the feature's influence, the darker the color. The more positive the feature's influence, the bluer the color; the more negative, the redder it will be. Features that are not significant will either have two columns of the same color, indicating no influence in distinguishing between classes, or two columns of very light colors, indicating a small influence. A diverging colormap was chosen to visually distinguish between positive and negative influences and to more clearly show their scale.

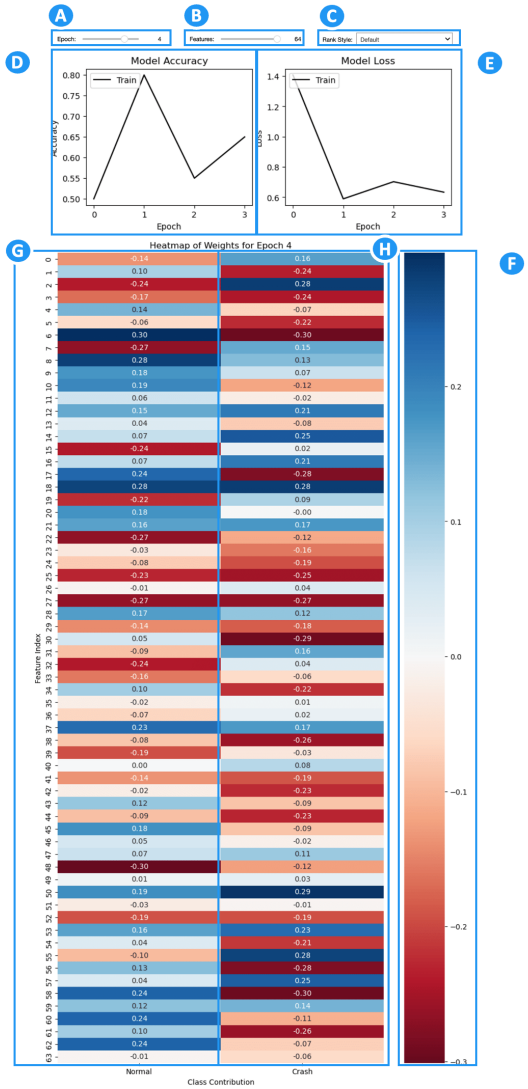



Fig. 6: The third component of the interface would be a weight change graph. (A) A scrollbar that controls the epoch where the visualizations would get produced. (B) A scrollbar of the number of displayed features. (C) A dropdown menu for selecting the kinds of display. (D) The model accuracy line plot. (E) The model loss line plot. (F) A colormap showing how warm or cool colors reflect features' weight values. (G) Feature's weight values that either positively or negatively contribute to Class 0's classification. (H) Feature's weight values that contribute to Class 1's classification.

In the user interface, users can see three interactive components: a scrollbar to control the number of epochs they want to see the feature weights for, along with the model accuracy and loss for that epoch; a scrollbar to specify how many features they want to examine; and a dropdown menu offering four ranking styles for organizing those features. The ranking styles are the same as those described in the feature tendency graph above. The interactions are provided not only to filter out irrelevant information but also to provide a dynamic way to view how feature weights relate to model performance during training. Jupyter Widgets are used to enable interactivity, and Matplotlib is used for both the line plot and heatmap.

## 5 Evaluation

### 5.1 A Case Study with Video Vision Transformer

#### 5.1.1 Data Preprocessing

The input data for this project is in video format and must be converted to a processable format before being fed into the model. The videos, which consist of sequences of multiple image frames, are converted into collections of feature maps derived from all the frames. Specifically, each video is divided into frames using an OpenCV tool, and each image frame is initially processed with only pixel information. Since all videos are 5 seconds long, 50 frames are extracted from each one. Once the videos are converted to frames, TensorFlow's decode and resize methods are used to convert the RGB pixel information to grayscale, resize each image to 128 × 128 pixels, and normalize the pixel values. After processing the input data, a 60:40 train-test split is performed based on the video names, and 25% of the training videos are selected as validation videos.

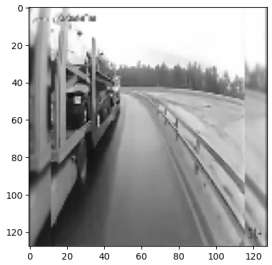


Fig. 7: This is a 128x128 sample input in grayscale. It is a only the first frame of a crash video. The rationale behind using grayscale image is to hopefully let the model detect the motion of the car crash instead of the pixel change of it and to release the computational burden carried by the RGB image data.

#### 5.1.2 Preparing Data to Train Model

After preparing and grouping the input data, test, train, and validation data loaders are created using the TensorFlow Dataset class. The batch size is set to eight for this project. Once the datasets or data loaders are created, embedding and position encoder functions are defined and derived from the keras.layers library to process the video data. The logic is that since each frame of the video can be seen as a sentence, with some frames being more closely related (e.g., sequences of crash events), a positional encoder is needed so the data can be aware of the positions of its own and corresponding frames. This allows the model to comprehend the order of frames and patches. Before the positional encoder, there is a tubelet embedding that converts frames into embeddings more easily processed by the positional encoder and makes the input compatible with transformer models in general. Both the tubelet embedding and positional encoder serve as early layers in the Video Vision Transformer used in this project.

### 5.2 Building Video Vision Classifier

After that, a video vision classifier is defined with the input shape of each video data being (50, 128, 128, 1). The 50 indicates the number of frames per 5-second video, while each frame is sized at 128 × 128 pixels, and 1 signifies that all frames are in grayscale. In this Video

Vision Transformer model, the number of attention heads is set to 6, and the number of embedding layers is 64. Thus, each video is divided into 64 features, which the model uses for classification. As previously mentioned, the primary goal is to identify which videos depict a car crash scene and which do not. Therefore, two classes—"Normal" and "Crash"—are defined for classification.

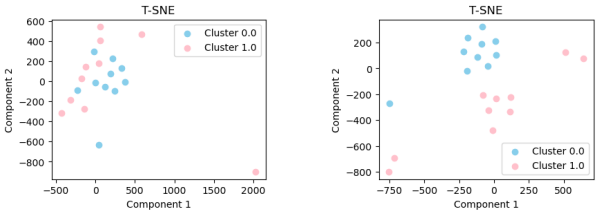
The model consists of 60 layers in total. Two of them are the tubelet embedding and positional encoder layers, which transform the input data into a suitable format. Next, there are six iterations of a normalization layer, a multi-head attention layer to capture contextual information within the 50 video frames, a layer that combines the original transformed patches of video data with the attention output for the skip-connection step to prevent overfitting and help the model better understand both the original and transformed input features, and finally two dense layers with Gaussian Error Linear Units (GELUs) activation for classification, followed by another skip connection. After these iterations, the output layers are normalized again, and average pooling is performed on each feature map to extract only the essential information for classification. Lastly, a dense layer with a SoftMax activation function is used for the final classification step.

### 5.2.1 Model Training and Visualization

In the code panel, an $int_weight_vis$ tool, which stands for "interactive weight visualization" and is the name of the tool developed in this project, is imported. The run function is then called and is ready to train and output a user interface for the analysis step. In this case study, the parameters of the run function are a defined classifier model, test, train, and validation data loaders, a sparse categorical cross-entropy loss function, five epochs, and the two class names, "Normal" and "Crash."

### 5.2.2 Read the Visualization

Based on the t-SNE graph (Fig 8), if one sets the perplexity to 11, sparse sky-blue and pink dots appear across the graph, with neither forming a clear cluster. However, if the perplexity is set to 7, the clusters of pink and blue are more defined. Unfortunately, out of all the perplexity values allowed for the t-SNE graph, a perplexity of 7 is the only setting where one can clearly see the clusters. Therefore, one may deduce at first that the model is not classifying the videos well. In fact, the testing accuracy in this case is only 68%, which is not high.




(a) Valid and Correct Classification (b) Invalid yet Correct Classification

Fig. 8: T-SNE Graph of "Crash" and "Normal" classification result from the test loader and with perplexity set to 11 and 7. One may see clearly that at perplexity 11 (the left image), the sky-blue and the pink points are mixed in one cluster. While when one sets the perplexity to 7 (the right image), the two clusters appear though there are also points at either edge of the graph. Unfortunately, the right graph is the only graph with two defined clusters out of 20 t-SNE plots that could be graphed.

Inspecting the feature tendency graph (Fig 9), if one sets the rank style to "Crash (DESC)" — meaning the features are ranked by their weight or influence from the most positive to the most negative—one would see that, surprisingly, the first two features (Feature 50 and Feature 18), although they have the most positive influence in classifying a video to the "Crash" class, show no significance during the classification process because they also positively classify a video to the "Normal" class.

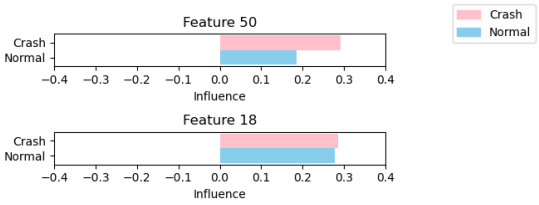



Fig. 9: Two insignificant features in this case study, because they are classifying an input to both classes, so no actual classification or distinction is made.

Same phenomenon (Fig 10) happens when doing "Normal (DESC)," though one may see feature 6 and feature 58 being significant.

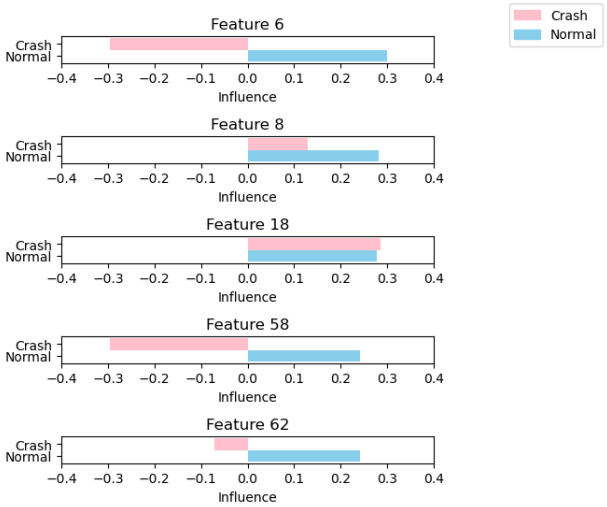



Fig. 10: Display of top 5 features that contribute most positively to the classification to the “Normal” Class. Feature 8 and 18 are clearly not significant because they are not classifying a video to either of the classes - they consider a video as it belongs to both classes.

If one sets the rank style back to "default" and inspect only five of them, then one may see that the first four features tend to contribute to the classification to the "Crash" class more while feature 4 contributes to the classification to "Normal" class more.


Fig. 11: This is a display of the first 5 features and which class are they either positively or negatively contributing more to. Feature 0 is having a weak contribution, positively or negatively, to classify an input to Class 1 or Crash class.

Looking at the third visualization, one can quickly deduce, by dragging the epoch scrollbar, that there is a sudden drop in accuracy at epoch 3 along with a slight increase in loss. One can then proceed down to find the weight heatmap associated with that drop and, for instance, select the top 5 features that contribute most positively to the "Crash" class. Features 50, 18, and 14 show no significance in the third epoch in classifying a video to either class, while Feature 2 performs well by contributing positively to one class and negatively to the other. Feature 2 shows the most significance in helping to classify a video since it, at the same time, negatively contribute to the classification to "Normal" class and positively contribute to the classificaiton to "Crash" class, which at least shows that it is making a determinant classificaiton. However, since we are inspecting the potential cause of a low point in accuracy, we may suspect if this determinant distinction is indeed accurate. To validate feature 2's weights, one may create another TensorFlow callback object that either reverses the two weights or mitigates the effects of the two weights, and re-run the model training step with that callback object included to see if the classification accuracy cahnges.

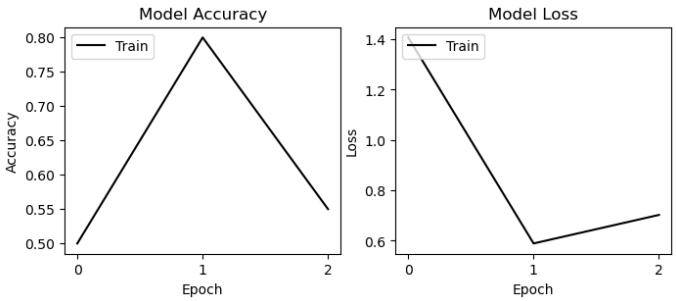



Fig. 12: The accuracy and loss graph in this case study. One may see a sudden drop of accuracy at epoch 3.

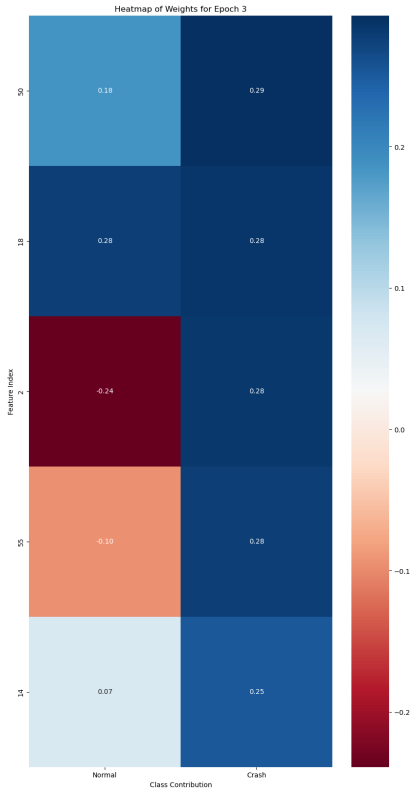



Fig. 13: Heatmap of weights at epoch 3. Weights are ranked in descending order of their contribution to video's classification to the "Crash" class.

As one continues to adjust scrollbar with a specific ranking style set, such as the "Crash (DESC)" rank style in this case, an animation can be generated to show any features are increasing in their contribution to classifying a video positively as a "Crash." This can help users identify which features are gaining importance in the classification process. Unfortunately, in our model, only a few weights change (Fig 13 and 14), indicating that the model may not be fully differentiating between the two types of videos, as the weights are similar to their initial randomized values.

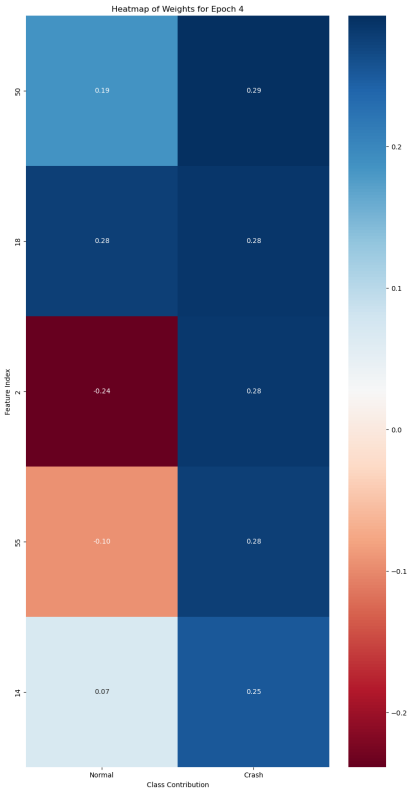



Fig. 14: Heatmap of weights at epoch 4

This limited change could indicate the need for additional training data, as only 100 out of the available 4,500 videos were used for both the training and testing processes. It's also possible that the current structure of the classifier could be improved. Two dense layers might not be enough in capturing the differences between the classes effectively. Alternatively, expanding the dimensionality of the data during the dense layer stage or increasing the number of embedding layers could provide a more nuanced representation of the data.

### 5.3 Evaluation Set-Up

The evaluation is fully qualitative. The user interface was only shown to two Data Science students from other classes. Each was asked to describe the good and bad aspects of the interface, if any, with a brief explanation. For simplicity, they did not test their own models using this tool; instead, they used the interface developed during the case study portion and were given verbal instructions on some operations to explore before using the interface directly. Thus, the evaluation is mainly based on the adaptability and interactivity of the interface and does not provide insights on whether the interface successfully convey the message it aims for.

### 5.4 User Reviews

Most users understood the parameters of the run function because they correspond to those in TensorFlow's compile function. However, they still suggested shortening the number of parameters required or providing guided instructions for input, as having six required parameters

was somewhat burdensome. Another piece of feedback received was confusion about the tendency display. Primarily, the x-axis of both displays was confusing at first glance because its meaning changes between the two. In the first display, the left and right directions of the bar represent positive and negative influence, while in the second display, they represent which class the feature predominantly contributes to. One user's advice was to create two separate interfaces for the two displays instead of merging them into one, or we should at least give a visual or text guidance at the side of the interface showing what each graph means. They both advised to remove the number scale when the displays were switched to "tendency" mode.

All users were satisfied with the weight change interface and the filter and rank functionalities provided by the tool because they allow them to focus on analyzing a selected amount without getting distracted. However, it was unfortunate that the interface could only be used for binary classification models, which are not very common in the market.

## 6 Conclusion

### 6.1 Interface Development

A Python tool that deploys an interactive visualization tool for analyzing a model's weight features was developed to help users gain insights into how each feature contributes to classification results, its significance, scale of significance, and any potential bias in the model if more features classify an input into one class. The tool can be used for any general TensorFlow binary classifier model, from a simple CNN model to a complex Video Vision Transformer model. One may easily download images from the interface provided by this tool to aid in explaining one's model performance. Unfortunately, in our case, we can utilize the fact that the weights were not changing much and that many insignificant features are present as noise features, and a t-SNE graph of relatively indistinguishable clusters, to highlight how poorly our model performed. But the tool matches its aim in making a complex model less black-box.

### 6.2 Potential Improvements on the current Video Vision Transformer

There are several improvements we can make to our case study's model based on our understanding and the insights provided by our visualization tools. We can allocate more time for the training step by feeding the model with RGB data through additional embedding layers, such as 128 or 256 layers instead of 64. We can also increase the learning rate so that the feature weights can change more quickly. Additionally, we could incorporate a callback object mapped to the weight matrix to disable features that aren't significant for decision-making or use it to apply a function that doubles the effect of certain weights. We could also switch the weights of prominent features to see how this affects the model's fine-tuning. Of course, we'd remain aware of overfitting and would try to address it as we are fixing the model.

### 6.3 Limitations and Future Works

As mentioned earlier, this tool is limited to binary classifier models and does not explicitly convey the implications behind features. This is evident in many figures of the case study where features are labeled with numbers rather than specific names. When we initially tested the tool on a complex model involving various embeddings that transform data, it was difficult to associate a weight with a particular feature or set of features. However, we believe this issue comes from our naive understanding of transformers. Previous work on decoding Natural Language Processing (NLP) transformers has shown how individual words contribute to the model's comprehension of their meaning. This requires quantitative data from the multi-head attention matrix, which in transformers quantifies the contextual relationships between input tokens. In the case of Video Vision Transformers, each input token would represent a video frame, and the attention mechanism enables one frame to focus on another. Together, frames that most strongly attend to each other may form a sequence, such as a car crash.

If we can decode the multi-head attention data and overlay it onto the input video frames, we could theoretically create maps visualizing how each frame attends to others. This mapping is crucial because it reveals the logic behind the video vision transformer. A logical model should produce attention maps indicating direct, consecutive sequences of actions rather than scattered attention across distant frames. This visualization could also serve educational purposes for those interested in learning about video vision transformers. At one point, we managed to extract multi-head attention data but couldn't decode it due to time constraints. In the future, we plan to incorporate an interface for visualizing the multi-head attention matrix in our tool.

Furthermore, we have not fully evaluated the insights users can gain from our interface and need to ensure our visualizations are clear. Therefore, we plan to write documentation and incorporate directions within the interface to guide users if operations are too complex or burdensome. We would also like to include a video guide or animation to help users take full advantage of the tool. That being said, we also plan to implement this interface to a web page using JavaScript instead of limiting it to only Jupyter Notebook environment.

Finally, we would fix some logical bugs, such as removing the x-axis numbers on the feature tendency graph when it is in tendency mode. The tendency mode displays a bar graph that extends left or right to indicate whether weights classify an input toward one class more than another, regardless of positive or negative contributions. Depending on the information we want to convey in the future, we may remove the x-axis numbers entirely to focus on comparisons. We might need to provide a logical method to rank the excess intensity since comparisons depend on how much one weight exceeds another.

In the future, we aim to enhance the interpretability of our interface and specifically investigate decoding embeddings and multi-head attention layers to overlay this information on video frames, mapping attention between them. Our goal is to refine the interface specifically for analyzing specifically video vision transformers.

## 7 Reference


1. Arnab, A., Dehghani, M., Heigold, G., Sun, C.,Lučić, M., & Schmid, C. (2021). *ViViT: A Video Vision Transformer.* Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 6836-6846. Retrieved from https://ar5iv.labs.arxiv.org/html/2103.15691\
2. CoderzColumn. (n.d.). *An In-depth Guide to Interactive Widgets in Jupyter Notebook.* Retrieved from https://coderzcolumn.com/tutorials/python/ipywidgets-an-in-depth-guide-to-interactive-widgets-in-jupyter-notebook
3. Kay, G., & Enguerrand, H. (2019). *Computationally Efficient Feature Significance and Importance for Machine Learning Models* Retrieved from https://ar5iv.labs.arxiv.org/html/1905.09849
4. Krause, J., Perer, A., & Ng, K. (2016). *Interpreting the Predictive Power of Machine Learning Models.* arXiv:1606.05685. Retrieved from https://arxiv.org/abs/1606.05685
5. SHAP. (n.d.). *Multioutput Regression SHAP*. Retrieved from https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/model_agnostic/Multioutput%20Regression%20SHAP.html