visualkeras: A Python Package for Visualizing Keras and Tensorflow Models

Paul Gavrikov [1,2] and **Santosh Patapati** [3]

1 IMLA, Offenburg University **2** University of Mannheim **3** Cyrion

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

visualkeras is a Python package designed to facilitate the visualization of Keras and TensorFlow models. It provides an intuitive developer interface for generating visual representations of model architectures, making it easier for researchers and developers to understand and communicate their designs. The package supports layered volumetric views in 2D / 3D space and directed node-edge graph-based layouts. When provided with a functional or sequential Keras model, visualkeras can generate a highly customizable visualization through various parameters such as color, spacing, dynamic sizing modes, legends, dimensionality, textual annotations, orientation, and more.

Statement of Need

The visualization of Artificial Intelligence (AI) and Machine Learning (ML) models plays a crucial role for understanding and communicating their architecture. The effectiveness of such visualizations plays a key role in the scientific process. Although detailed descriptions of model architectures and mathematics are often provided in research papers, architectural diagrams are essential for conveying complex structures and relationships in a more accessible manner.

The Keras package provides a high-level API for building and training deep learning models. Keras and its underlying framework, Tensorflow, have been widely adopted in the AI and ML community (cite). However, the built-in visualization tools in Keras are primitive and do not provide the flexibility needed for proper architectural representation. Images generated using Keras's built-in visualization tools require significant effort for readers to understand and are simply not suitable for scientific publication or communication purposes. visualkeras addresses this gap by providing a comprehensive set of tools for visualizing Keras models in a way that is both informative and visually appealing.

Key Features

visualkeras offers a range of visualization features and customization options. The framework is split into two main components.

Layered View

This component is designed to render both sequential and functional models using a pseudo-3D stacked box layout in a single continuous view. Each box visually represents a layer, with its width, height, and depth corresponding to the layer's spatial and channel dimensions under one of five sizing modes (`accurate`, `balanced`, `capped`, `logarithmic`, `relative`).

Rendering options can be toggled between three-dimensional (volumetric) and two-dimensional (flat) modes via the `draw_volume` parameter. Funnel-style connectors can be displayed between boxes using `draw_funnel`, and `shade_step` controls the deviation in lightness to improve depth perception. Logical spacing can be introduced through special “dummy” layers (`SpacingDummyLayer`) which are incorporated into the model object itself. Users may add custom annotations to each box via a `text_callable` function, which can be further customized with vertical offset adjustments provided by `text_vspacing`. A flexible `color_map` parameter allows users to color boxes based on layer type or user-defined attributes.

Layout control is further refined by adjusting `spacing` (inter-layer gaps), `padding` (margins at the beginning and end), and orientation settings. One-dimensional layers can be oriented using `one_dim_orientation`, and individual layers can be constrained to 2D rendering via `index_2D`. An entire model can be rendered flat by disabling volumetric rendering. Support for better visualizing decoder-like architectures is available through the `draw_reversed` option.

A configurable legend can be added, with options for adjusting text spacing (`legend_text_spacing_offset` font properties (`font`, `font_color`), and whether to show dimensions at each layer (`show_dimension`). Finally, users can control scaling across the x-y plane and z-axis using `scale_xy` and `scale_z`. These dimensions can be explicitly capped or floored using `max_xy`, `max_z`, `min_xy`, and `min_z` parameters.

The final visualization is produced as a Pillow `Image` object, which can be displayed in Jupyter notebooks or saved to disk.

Graph View

This component generates a left-to-right node-edge visualization of any `Keras` or `tf.keras` model by treating each layer (or individual neuron) as a node and drawing directed connectors to represent data flow. Given a `Model` instance, the function computes a hierarchy of layers based on their graph depth, places nodes evenly spaced in horizontal layers, and centers them vertically within the image canvas. Each node is drawn as a fixed-size circle or box (specified by `node_size`), and may represent the entire layer or each neuron it contains, depending on the `show_neurons` parameter.

Connectors between nodes are rendered as lines whose color and thickness can be controlled through the `connector_fill` and `connector_width` arguments. Layout parameters such as `layer_spacing`, `node_spacing`, `padding`, and `background_fill` allow users to adjust the overall compactness, margins, and canvas appearance. For models with a large number of neurons in a layer, the `ellipsize_after` parameter can be used to replace excess nodes with an ellipsis symbol to prevent overcrowding. The `inout_as_tensor` option determines whether each tensor input or output is showns as a single tensor (rectangular shape) or expanded into multiple units.

Node coloring is fully customizable via the `color_map` parameter which maps layer classes to fill and outline colors. This allows for visually distinguishing different layer types.

Like the Layered View, the Graph View produces a Pillow `Image` object that can be displayed in Jupyter notebooks or saved to disk.

Usage Examples

In this section, we provide examples of how to use visualkeras to visualize Keras models in different ways. The examples shown in Figure 1, Figure 2, Figure 3, and Figure 4 demonstrate the flexibility and customization options available in the package. The generated graphical visualizations can be displayed in Jupyter notebooks or saved as image files for use in publications or presentations.

Layered View

Basic Usage

```python
import visualkeras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout

# Define a simple sequential model
simple_sequential_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Basic usage of visualkeras
basic_layered_img = visualkeras.layered_view(simple_sequential_model, draw_funnel=False)

# Display the image
basic_layered_img.show()
```

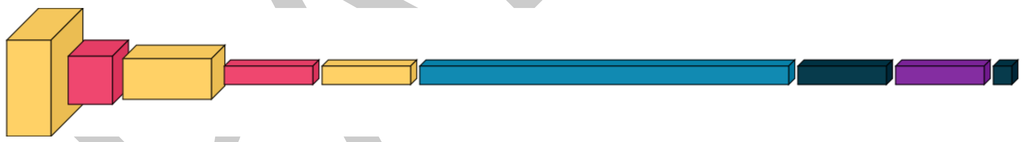

Figure 1: An example of layered style visualization on a simple sequential model with little styling

Advanced Usage

```python
import visualkeras
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from PIL import ImageFont
from collections import defaultdict

# Define custom font for the model visualization
custom_font = ImageFont.truetype("arial.ttf", 24)

# Define custom color map
color_map = defaultdict(dict)
color_map[Conv2D]['fill'] = 'orange'
color_map[MaxPooling2D]['fill'] = 'red'
color_map[Dense]['fill'] = 'teal'

# Define a larger sequential model with more complexity
complex_sequential_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Advanced usage of visualkeras with custom styling
advanced_layered_img = visualkeras.layered_view(
    complex_sequential_model,
    legend=True,                    # Show legend
    font=custom_font,               # Custom font for legend
    color_map=color_map,            # Custom colors
    draw_volume=True,               # 3D volumetric rendering
    draw_funnel=True,               # Show funnel connectors
    spacing=50,                     # Increase spacing between layers
    padding=30,                     # Add padding around the visualization
    scale_xy=2,                     # Scale x-y dimensions
    scale_z=1,                      # Scale z dimension
    max_z=400,                      # Cap maximum z dimension
    font_color='black',             # Legend font color
    one_dim_orientation='y',        # Orientation for 1D layers
    sizing_mode='accurate',         # Use balanced sizing for layers
    type_ignore=[Flatten, Dropout], # Ignore Flatten and Dropout layers
)

# Display the image
advanced_layered_img.show()
```

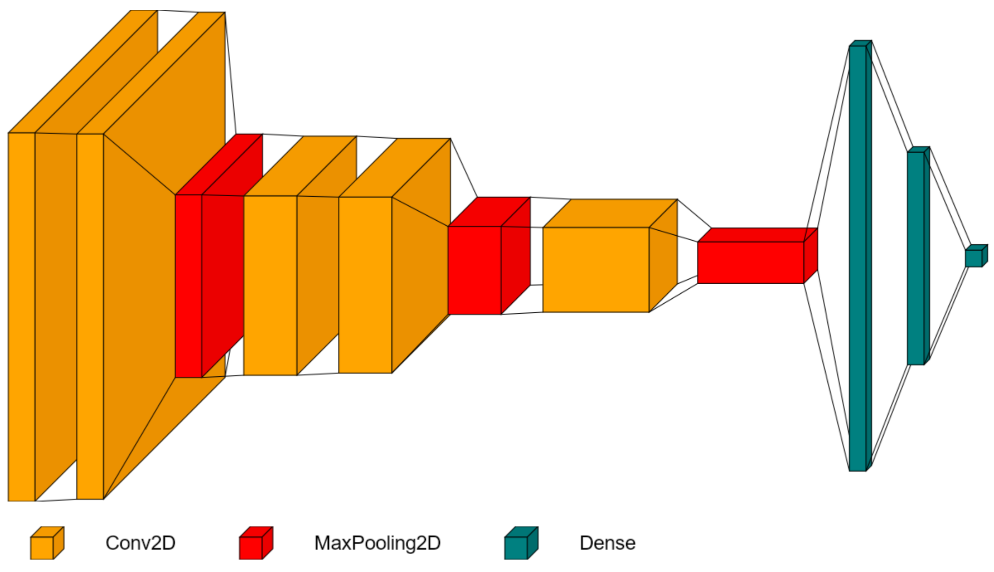

Figure 2: An example of a more complex model's Layered View with custom styling

85 Graph View

86 Basic Usage

```
import visualkeras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout

# Define a simple sequential model
simple_sequential_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Basic usage of visualkeras to create a graph view
basic_graph_img = visualkeras.graph_view(simple_sequential_model)

# Display the image
basic_graph_img.show()
```

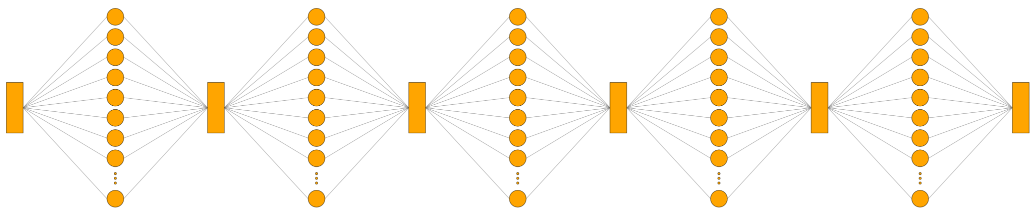
Figure 3: An example of Graph View visualization on a simple sequential model with little styling

Advanced Usage

```python
import visualkeras
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from collections import defaultdict

# Define a larger sequential model with more complexity
complex_sequential_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Define custom color map for the graph view
graph_color_map = defaultdict(dict)
graph_color_map[Conv2D]['fill'] = 'lightblue'
graph_color_map[Conv2D]['outline'] = 'darkblue'
graph_color_map[MaxPooling2D]['fill'] = 'lightcoral'
graph_color_map[MaxPooling2D]['outline'] = 'darkred'
graph_color_map[Flatten]['fill'] = 'lightgreen'
graph_color_map[Flatten]['outline'] = 'darkgreen'
graph_color_map[Dense]['fill'] = 'lightyellow'
graph_color_map[Dense]['outline'] = 'darkorange'
graph_color_map[Dropout]['fill'] = 'lightpink'
graph_color_map[Dropout]['outline'] = 'purple'

# Create advanced graph view with customizations
```

```
advanced_graph_img = visualkeras.graph_view(
    complex_sequential_model,
    color_map=graph_color_map,      # Custom color scheme
    node_size=60,                   # Larger nodes
    connector_fill='gray',          # Gray connectors
    connector_width=2,              # Thicker connectors
    layer_spacing=180,              # More spacing between layers
    node_spacing=40,                # Spacing between nodes in same layer
    padding=40,                     # Padding around the diagram
    background_fill='white',        # White background
    ellipsize_after=8              # Ellipsize layers with >8 neurons
)

# Display the image
advanced_graph_img.show()
```

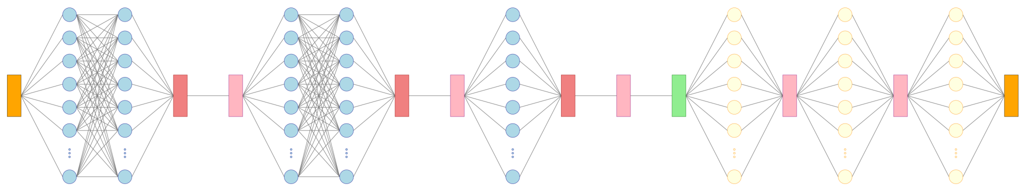

Figure 4: An example of a more complex model's Graph View with custom styling

Acknowledgements

We thanks and acknowledge the contributions of the open-source community which has supported the development of visualkeras over the years through their feature requests, bug reports, issue discussions, and pull requests. These contributions have allowed us to continously improve the package and adapt it to the needs of the community. We are grateful for the collaborative spirit that has made visualkeras a valuable tool for visualizing Keras and TensorFlow models.

References