On Stylistic Differences Between 3D Meshes

Callum McMahon

August 2020

This report is submitted as part requirement for the MSc Degree in Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

Contents

Abstract

Gram matrices are used extensively to represent style for image data, however these techniques have thus far not been used on three-dimensional data. In this report, we demonstrate the efficacy and flexibility of gram matrices over existing techniques in the context of identifying regions of style similarity between 3D shapes. The meshCNN model framework is used to extract features per edge of mesh data, which form the basis of style representations. Saliency maps are generated through backpropagating differences in style representations between shapes, and are visualised via mesh colouring. Performance of style classification using our gram based style representations is compared to benchmarks, and style analysis results performed on images has been replicated using our methods for 3D shapes. An analysis is made quantifying the sensitivity of presented methods to pre-processing techniques. A code used for this report is publicly available on github at github.com/CallumMcMahon/MeshCNN

Chapter 1

Introduction

1.1 Problem Outline

Artistic style has been studied extensively over the last few years. The allure being that style is often thought of as a vague property of a medium, whose definition is very subjective. Methods have been devised to capture this style, with most methods being framed for use on two-dimensional images. Success has come from re-purposing coarse to fine-grained feature extractors that were tuned specifically to maximise content information extraction from images. Three-dimensional data poses many problems when trying to recreate this success. The first being that data no-longer has an obvious "best" format, where the choice of representation for the underlying structure becomes a design decision. Triangular meshes, point clouds, voxels and signed distance functions are examples of some of the more common formats. Once a representation is chosen, the "base" features often need to be hand-crafted from which richer features will be learned. This design choice is not required for images, where the red, green and blue channels per pixel can be used directly as base features. Given a data representation, a compatible model then has to be chosen. The unique strengths and weaknesses of each representation-model pair for different tasks makes it difficult to predict which will be most useful for a new task. This is in contrast to the image domain where convolutional operators unanimously make up the feature extraction component of model.

Once the machinery of image-based style representations have been successfully translated to the three-dimensional domain, the question remains of whether these methods will retain their success. Specifically, will the stationary feature space formulated to work on texture representations from a

structured grid still be effective when used on shape representations from unstructured data. And will the properties of style representations still hold true, such as semantic meaning of distances between style representations.

1.2 Aims and Goals

Given a target 3D object to be analysed, as well as a reference object with a desired style, the aim is to identify locations on the target object where style resembles or differs from the reference. The method should ideally be flexible enough to compare styles of abstract shapes such as sculptures, as well as more mechanical parts such as those used to make cars and phones, not relying on prior knowledge of shape structure or characteristics of the style.

1.3 Project Approach

The project will leverage recent advances in model architectures for three-dimensional data. The MeshCNN framework will be used to generate surface representations per edge, operating on manifold triangular mesh data. Like traditional image-based models, deeper layers in the network capture higher level features of the data in a hierarchical manner, using modified convolutional and pooling operators applied to edges and their neighbourhood of edges. Therefore edges and their corresponding features can be treated in much the same way as pixels in images.

From there, gram-based style analysis methods will be used to generate stationary style representations for data, independent of shape content. These can be formed from activations of each layer of the network, capturing style at different scales. These style representations can then be compared for any two 3D shapes, regardless of whether the style or object class has been observed during training of the model. Back-propagating dissimilarity between these style representations enables visualisations of precise locations on the analysed shape which contributed to the disagreement.

Being a relatively new method to process three-dimensional data, the sensitivity of the method to different factors is not documented. This includes the mesh resolution, i.e. the number of edges used to describe the same shape, the varying density of edges throughout the mesh, the difference in resolution and domain between the training and test data, different training objectives such as content or style classification, as well as the level of pooling applied at test time. These factors

could then in turn affect the quality of the style representations generated from the model. An analysis is performed to compare some of these different effects.

The dataset used contains 3D objects of many different categories, each of with having a few different styles. Mesh pre-processing methods are used to transform the dataset into a format accepted by the model. Different modifiers are applied during this pre-processing stage to standardise input data, to aid feature comparison between meshes.

1.4 Outline of Report

The report starts by getting the reader up to speed with background material that the methods used in this report build upon. This includes a summary of commonly used representations for three-dimensional data, with discussion of their ability to express artistic style. This is followed by a review of the breakthrough paper by [11] describing the gram-based style representation which led to the recent successes in neural style transfer. Moving to three-dimensional data, a summary is made of recent research which attempt to identify and categorise styles in 3D models, a well-studied task which closely relates to the problem proposed in the report.

Methods presented make use of deep-learning models which process three-dimensional data from various representations, therefore a summary of recent promising model designs is given for each of the representations. This is followed by an investigation of relevant datasets, comparing size, quality, and ability to compare styles.

Next are the methods used in the report. Many different pre-processing pipelines are compared. This is followed by implementation details of the model, detailing modifications needed to support the chosen dataset. An explanation is given of the different evaluation methods used in the experiments, followed by results.

A conclusion is then made, which includes a discussion of the strengths and successes of the presented methods as well as possible use cases in user facing applications. A critical analysis is made of the method's shortcomings, as well as possible extensions which may help resolve these issues and extend capabilities further.

Chapter 2

Background

2.1 Data Types

Three-dimensional data comes in many formats, where thus far no single representation appears to have dominated the focus of the research [55]. This is because there are many positive and negative properties to each representation which vary in importance depending on the application. An overview of a selection of standard representations is given, with an evaluation of each representation's ability to retain and express style elements.

2.1.1 Meshes

This format is most commonly found online, being the de facto choice for creators designing 3D shapes for use in animation, video games and simulations to name a few examples. 3D modelling software [36] often use meshes to represent shapes as they are an intuitive format for users to interact with and manipulate.

Triangular meshes, also known as tri-meshes, are given as an ordered list of vertices followed by a list of faces. Each vertex is specified by three coordinates to represent their position in three-dimensional space. Each face is then specified by the indices of the three vertices which define it, where the ordering of the given vertices informs on the direction of the face normal. The convention is to specify vertices in a counter-clockwise

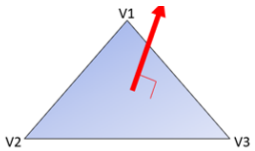


Figure 2.1: Face normal direction

order to indicate that the face normal is coming towards the viewer and
away from the mesh, as is shown in figure 2.1. Meshes focus solely on defining the 2D surfaces which make up 3D shapes [3].

Meshes can also be constructed from polygons with more than three sides, where the number of vertices specifies per face is no longer restricted to three, although these can be reduced to triangular meshes.

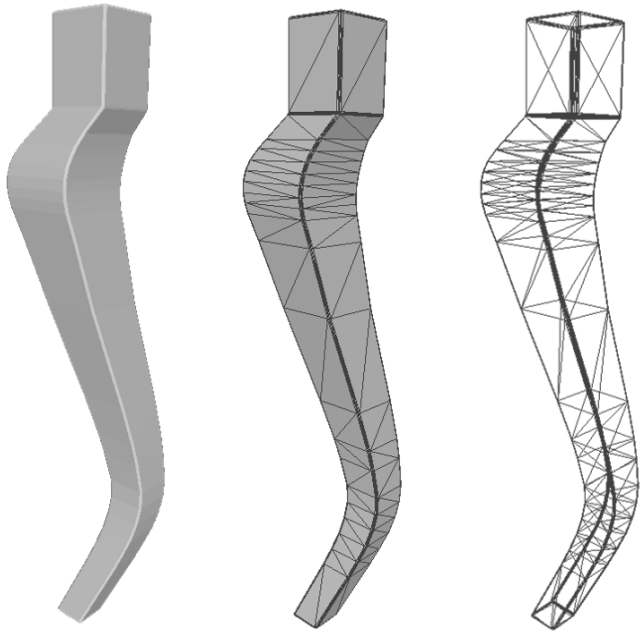


Figure 2.2: Left: 3D model, Center: Surface represented as connecting triangles, Right: Wireframe view of mesh representation data. Smooth leg data from [19] used in methods section.

The quality of mesh data can vary dramatically. A well-behaved shape for our purposes is known as manifold. [14] defines this property as follows.

> "various constraints on a singly connected mesh with the set $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ that enables manifoldness.
>
> - Each edge $e \in \mathcal{E}$ is common to exactly 2 faces in $\mathcal{F}$
> - Each vertex $v \in \mathcal{V}$ is shared by exactly one group of connected faces

- Adjacent faces F_i, F_j have normals oriented in same direction"

Meshes may also be constructed as the union of a set of many manifold components.

On the other end of the quality spectrum are polygon soups. These are simply a collection of faces with no organised structure or guarantee of forming a coherent surface. Faces may appear to be adjacent but not share common vertices meaning that there is no systematic way of knowing how the overall surface is made up. Badly behaving meshes, such as those which are non-manifold or containing self-intersections, can be the result of inexperienced designers or taking shortcuts during the modelling process where the resulting aesthetic is unaffected.

Properties of meshes are that they can have many different "remeshings" where the same surface is represented (or optionally closely approximated) using a different tessellation of faces, as well as using vastly different numbers of vertices and faces. A mesh's vertex count can be thought of as analogous to the resolution of image data [15], a property which is standardised through a host of pre-processing techniques. Similar techniques can be formulated for meshes such as rescaling through mesh simplification and division, cropping, and rotating.

Triangular meshes are made up of flat, planar faces. Therefore, all mesh surface locations not incident to vertices or edges are locally linear and do not capture the shape's curvature. Curves need to be approximated by many small faces which greatly increases the memory requirements and resolution of a mesh. This has an impact on the ability of meshes to represent style. Curvature is an important component of a shape's style, so needing to approximate this feature with many flat surfaces may not fully capture the desired style accurately. A mesh format known as a precise mesh overcomes the problem of surface curvature by modelling surfaces as Non-Uniform Rational B-Spline patches [40], however no deep-learning models can currently process this format of data so further discussion is not given in this report.

The faces making up a mesh do not have any constrains on size, meaning that large flat regions of a shape may be represented by a single face, whereas small curved shapes such as a handle may need hundreds or possibly thousands of faces to adequately represent the desired shape. Therefore the density of faces throughout shapes can vary dramatically, which has the advantage of not wasting storage space on simple regions of the image. A consequence of this feature is that given a surface patch of fixed size, the amount of information conveyed per face (or edge/vertex) varies greatly. This is in contrast to images where a fixed size region will always contain the same quantity of pixels evenly spaced out which sample the represented content.

Due to the widespread use of mesh formats for shape creation, shapes originally created in this format retain all stylistic intent from the original artist. Style details are not lost through transformation into a different representation. Mesh pre-processing techniques could however have similar side-effects on loss of style detail.

2.1.2 Point Clouds

Point clouds naturally arise from sensor data such as Lidar sensors. They measure the distance of projected points from a sensor to build up a collection of points in 3D space which all lie on the surface of a shape. Alternatively, point cloud data can be generated from meshes by uniformly sampling along the surface faces. [13]

Advantages of this representation are that points are generally evenly spread out along surfaces of shapes. Sampled points are not required to be concentrated around curved features of shapes, unlike with meshes. There is full control over the resolution of data based on how many points are samples, standardising the resolution of data for a downstream model. When sampling points from a mesh, point clouds are very robust to low-quality data and largely manages to capture the look that an original design aims to convey while smoothing over many of the problems that can occur in meshes.

Disadvantages are that finer details are lost when sampling from a shape, potentially smoothing over an intricate texture or pattern pertaining to the style of a shape. The connectivity of points is also not part of the representation, so surfaces of shapes are not explicitly mapped in the data. Methods to reconstruct a surface from point clouds often use nearest neighbours which can lead to undesirable effects such as interpreting thin 3D volumes as 2D surfaces. This is less of a concern for analysing point clouds sampled from meshes as face to point correspondences are known, however these limitations become more problematic for generative tasks. When converting data from mesh to point cloud formats, storage requirements often increase substantially, due to many samples being taken per mesh face. This affects the processing time and memory requirements for training and inference of a model.

2.1.3 Voxels

Voxelization is the process of representing shapes as a collection of cubes, or voxels. They are evenly spaced out in a grid pattern, and can be naturally thought of as the extension of pixels

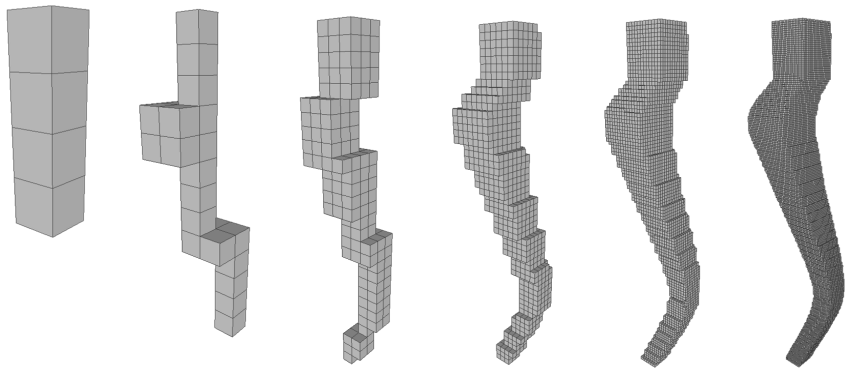


Figure 2.3: Voxel representation of Smooth leg 6 at different octree depths from 3 to 8. Smooth leg data from [19] used in methods section.

to a third dimension. Instead of representing colour values such as red green and blue, each voxel typically stores binary occupancy information relating to whether the corresponding area of space is contained within the surface of an object. An octree format for voxel data can reduce the redundant information required to store vast regions representing same binary value, resulting in most of the storage being used on the boundary locations along surfaces of shapes. Despite this optimisation, the addition of an extra dimension results in memory requirements growing exponentially quicker than an equivalent image of the same height and width. This limits shape discretisation to 256^3 or 512^3 voxels at most to be processed on modern GPUs [42, 49].

A signed distance function based format also exists where voxel values represent their proximity to shape surfaces, where negative values indicate interior voxels and positive values represent exterior voxels [6].

The advantages of this representation are that existing machinery designed for two-dimensional convolutions extend naturally to three dimensions. However, the restrictions in resolution severely limit the level of detail that can be captured from shapes. While meshes are limited to using flat surfaces to describe shapes, voxel representations have the further restriction that these flat surfaces are aligned to the coordinate axes. This further degrades the approximation of curved surfaces, especially in conjunction with limited voxel resolution. Details smaller than the chosen resolution for voxels are lost, making it infeasible to capture surface texture [54].

2.1.4 Implicit Neural Representations

Implicit Neural representations (INR) are a novel method of framing data. In the case of two-dimensional images, data can be thought of as sampling from a view of a ground-truth scene, discretising the continuous view of changing colours and curved edges into "buckets" represented as pixels. Alternatively, the data can be kept in a continuous space represented by a function. Resolution is then no longer a consideration when pre-processing data, as the limiting factor now stems from the expressive power of the function.

Data is represented by a function conditional on a vector associated with a shape, as well as a three-dimensional position to sample the function at. The underlying model is expected to harness the repeating characteristics between shapes to compress many different objects into a single model-vector pair of limited size. Reconstructing the original shape involved either solving for the zero-set of the function or generating a mesh from evenly sampled points using an algorithm such as Marching Cubes [31]. Vector representations for new unseen shapes within the same domain as training data may be obtained by optimisation of a random vector, using sample points as well as ground truth distances from shape surfaces. Model parameters stay fixed during this process, only modifying the vector which the model becomes conditioned on to recreate the shape.

Advantages of INRs are that there are no discretisation errors [38], meaning that complex styles can be preserved for analysis. Querying a given point makes use of the entire model, therefore meaning that representations are not restricted to local views of a shape, such as through a limited receptive field.

Disadvantages are that although shapes can theoretically be represented arbitrarily accurately, practical performance is in large part determined by the flexibility of the model used, such as the activation function chosen [48].

2.1.5 Summary

Many different representation exist for three-dimensional data, each presenting their own limitations, assumptions and expressive abilities. Whereas meshes and point clouds explicitly represent shapes in terms of their surface, voxels and implicit neural representations focus on the volume of shapes, where the surface is a property that arises from the boundary between regions inside and outside of a shape.

The fidelity of representations to the underlying data varies between formats. Implicit neural representations, followed by meshes, capture details sufficiently well to retain smaller stylistic elements, whereas point clouds and voxels either smooth over details or do not have the required resolution to represent the intended style of shapes. Therefore point cloud and voxel data structures for representing style will not be investigated further in this report.

2.2 Style Analysis

Style can have many interpretations, therefore having a well-defined framework is crucial for discussing the problem as well as measures of success. Most of the literature focuses on image data, therefore a review is first made of establishes methods in that domain, before being extended to three-dimensional data. Much of the work on image style has involved transferring styles from a source to a target image. Methods capable of performing this task adequately are considered to have captured a good representation of style, making them very relevant for the task of style identification.

2.2.1 Two-dimensional Data

Two-dimensional image style research makes reference to the separation of content and style from representations of data [11]. In images, content is often thought of as the subject of an image. This is also the semantic meaning that models are optimised for during image classification training. A desirable property for content representations is the lack of detailed knowledge about pixel and low-level features of images. Experiments by [11] show that attempts to reconstruct an image based purely on model activations at different levels of convolutional operators all adequately reconstruct the content of the scene, but later layers lose much of the texture information which is smoothed out. This property of the later layers meet the requirements for content representations, and hence are used. Two images with similar high-level features are thought to contain similar content for this reason.

Note that the higher-level content layers have traditionally been optimised to represent a single class from 1000 possible classes in the popular ImageNet data set [44], although in practice these layers tend to provide more granular information. The high-level convolutional layers have been shown to preserve spacial information of region that represents the primary object in the scene, as

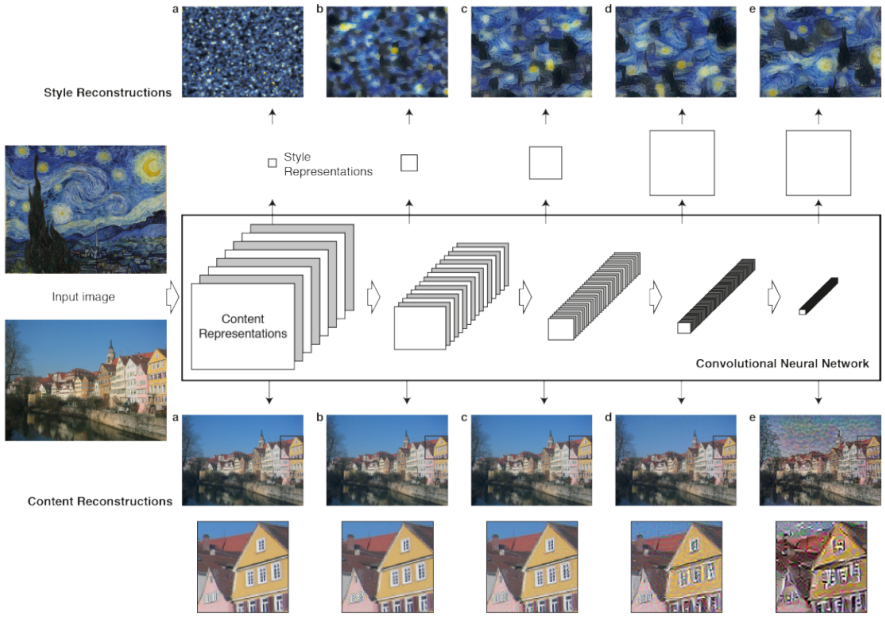


Figure 2.4: Figure 1. Reconstruction of the original image from activations at different layers shows that pixel-level details are lost at higher layers. Reproduced from [11]

well as identifying background objects [45]. This is especially useful when performing style transfer as the spacial locations of objects and background classes are considered to be part of the content. This choice can be viewed as an assumption about the definition of content, whereas a definition which wishes to be invariant to spatial locations of content might use a fully-connected layer later from a model after spacial structures have been removed.

Style is then defined as all possible choices for texture and feature representations such that the content remains unchanged (or similar within a tolerance). The breakthrough paper by [11] describes a method for obtaining a "stationary, multi-scale representation of the input image, which captures its texture information but not the global arrangement". This is achieved by summarising feature statistics using a gram matrix, which measures the correlation between different channels of feature maps. Spacial information is removed by flattening pixel structure from a grid to a singular dimension list. Correlations between feature channels are then computed producing a covariance-like matrix, knows as a gram matrix G [11].

$$G(F^l(I_s)') = \frac{1}{C \cdot N}[F^l(I_s)'][F^l(I_s)']^T \tag{2.1}$$

where N is the number of pixels and C is the dimension size of the feature vectors, I_s is the image that style is extracted from, and F^l are the feature activations from layer l of a model.

Similar to the experiments used to identify content representations, images can be constructed to match style representations, giving insight into which details are captured by the representation. The texture of different scales are recovered with finer details from lower levels and global style at higher levels, "a result that can be explained by the increasing receptive field sizes and feature complexity" [11].

The styles captured by these techniques are all texture-based, which is expected as the base features input into the model are colour pixel values, and convolutional kernels are known to activate to the presence of textures that emerge from the colour data.

2.2.2 Three-dimensional Data

Many areas of research are attempting to recreate the success of 2D neural style transfer in the 3D domain. Many of these methods involve projecting 3D shapes onto a 2D plane before apply standard 2D convolutional neural network machinery, in hopes of harnessing their powerful information extraction capabilities.

[30] uses renderings of shapes from multiple views which are each passed through a CNN to generate a representation of the object. Views from positions around a sphere centred on each shape are selected based on maximising entropy, in hopes of capturing the greatest style information. Five views with the greatest entropy are used by concatenating their outputs from a CNN into a single representation. A network is trained on triplet samples which contain both a positive and negative style similarity to a reference sample. Metric learning is used to overcome the subjective nature of style similarity, as no ground truth style annotations are needed, instead relying on relative closeness of styles. The problem of limited data and over-fitting is dealt with by using vast amounts of labelled image data available online for various styles, which are used in conjunction with the rendered views of the 3D objects. The ability to utilise much larger quantities of image data is a benefit of projecting three-dimensional data down to two dimensions.

[57] provide a much more comprehensive implementation of a multi-view approach, providing the ability to back-project identified features onto a shape for style localisation, as well as incorporating many views into a single representation of shape using partially shared latent factor learning. Because all views will contribute to an overall representation, the quality of specific views is less

important, allowing for 12 standardised view positions at 30° intervals around each shape. Lighting and other viewing artefacts present in standard renders are avoided, instead relying on feature lines based on the geometry of shapes. Image patches representing isolated stylistic elements are identified using k-means clustering, producing "mid-level patches" of a fixed size. These patches are convolved over new views to produce feature maps indicating regions of an image where a particular style patch may be present. The process of fusing patch information from the multiple views also predicts a most likely style.

Other methods use some form of neural renderer. [25] creates a differentiable pipeline where meshes are rendered to a 2D image, and losses can be backpropagated to alter mesh surface texture and shape. Images of chosen styles can be transferred to the texture of a 3D object while preserving style characteristics from the rendered view of the object. The silhouette outline of a mesh is also shown to be optimisable, although projection into a lower-dimensional plane limits the possible information extraction of more complex shapes with concave regions. [27] and [28] model three-dimensional fluids using vector fields and particles which evolve over time using physics based simulation techniques. At each timestep, the state of the simulation is rendered onto a 2D plane where NST techniques can be directly applied and backpropagated to the representation for the fluid, rendering the given view of the fluid with a desired style from a provided source image. These projective techniques are particularly useful when the final results are only observed from the two-dimensional view used during optimisation, while other reasons require that they be represented in 3D, such as for simulations or interactions with a scene.

It should be noted that these methods use activations from 2D images and therefore treat style in the same regard as the original NST methods, relying heavily on textures for their representation.

An alternative approach taken is to create features which directly describe 3D shapes in three-dimensional space without the loss of information associated with rasterizing shapes onto a plane. Particularly for style analysis, these methods have focused on representations which describe the surfaces of objects. This ensures that each point on the surface of a shape can be taken into account by the chosen model.

The most prominently used descriptors are all handcrafted solutions, which aim to describe surface patches of a shape. For example, [19] described these features as "curvature of the points, saliency of point sets, ambient occlusion, average geodesic distance, point height and upright angle, shape diameter function (SDF), and PCA-based features".

[7] maps these features only a two-dimensional image minimising distortion in the mapping, before treating these maps as image channels to a standard convolutional neural network. This approach differs from methods that directly project a rendered view onto an image, as three-dimensional information is extracted first.

[19] instead clusters the handcrafted features into a few centroids which represent the most prominent features on each shape. A linear SVM classifies these features into a small distinct selection of styles present in their dataset. Multiple styles can use the same features, therefore style signatures are created from selecting the 20 most discriminative style elements using the minimal-redundancy-maximal-relevance criterion. Defining elements are selected by removing the most discriminative elements one at a time until the remaining elements have a style classification performance of less than 0.9. These style-defining elements can also be visualised on 3D shapes.

Other methods have tried to take a more direct approach, training models to distinguish between different object styles directly. So far, these methods have, to the best of our knowledge, all involved predicting style types directly using features, handcrafted or otherwise. This is reminiscent of early image style work that built classifiers to predict styles directly from the outputs of CNNs [24]. As mentioned by [11], "transformation into a stationary feature space such as our style representation might achieve even better performance in style classification".

2.2.3 Saliency

A saliency map is a method of visualising the relative importance of components from data in making a decision [1]. In our case, this can be the regions of a 3D shape which contribute to classification of a certain style, or regions where style from a shape matches or clashes with that of another. While some methods will provide saliency values for points across the entirety of analysed data, others only identify highly critical regions for decision making.

[19] attempts to cluster handcrafted feature embeddings and classify style based on the frequency of features from each cluster. Clusters which differentiate a chosen style most accurately are deemed to be most representative of the style. Therefore, patches belonging to these clusters are used to visualise regions which best represent the style. A scalar field over shapes can therefore be created, where each sampled point receives a normalised vote per style-defining element based on feature similarity, defined by a margin threshold from a trained SVM. Each sampled point represents a shape region of fixed radius, therefore providing a scalar value for each point on shapes. This scalar

field is however reliant on accurate style-defining elements being identified for a given style before analysis.

[32] also generates saliency maps, although this method results in a coarse indication of style agreement. Shapes are first segmented into large semantically meaningful patches, where a mapping is made between matching elements between shapes. This restricts analysis to shapes which contain similar content. A style distance metric is computed by aggregating many sampled points on a patch and comparing this representation to those on the corresponding patch of another shape. Therefore, only a single saliency score is given per segmented element.

2.2.4 Style Definition

In an attempt to recreate the success of the breakthrough paper by [11] in the three-dimensional domain, the original assumptions, definitions and desired properties for content and style must be revisited.

Content is chosen to be defined as the underlying structure that encapsulated the semantic meaning of a shape. Functional components of an object often help convey the content, such as legs to a chair and surfaces to table. Without them, the content of shapes would be ambiguous. These aspects are considered to be less driven by artistic choice but rather by necessity to convey what the shape is trying to represent. Content should be invariant to scaling, translations and rotations.

The style of a shape is then defined as every artistic choice made when representing the underlying content. This notion should be very closely linked to the traits of a shape's surface. The style of a 3D shape can be separated into multiple levels of abstraction. Low-level stylistic features are very localised and are less related to the overarching content being represented, such as surface texture, engravings, and smoothness. High-level stylistic features are much more global and span large sections of the model. They should represent artistic decisions about how the basic components of an object's content are to be presented, such as the use of cylinders or cuboids to represent legs of a chair, and the use of a flat or concave surface when designing a plate. Different assumptions can be made about which operations style should be invariant to. [56] treats anisotropic part scaling (scaling each spatial dimension by a different factor) as a style-defining feature. A consequence of this assumption is that data augmentation should only scale 3D models equally along each coordinate axis. Rotations along the vertical z-axis are generally considered to preserve style, as human perception is used to observing objects from different directions, while considering the

style unchanged. A more nuanced decision is that of whether rotations along the horizontal x and y coordinate axes should be considered to have affected the style. Handcrafted features used by methods such as [19] assume that data is in an upright position, therefore style is to be considered dependent on x and y-axis rotation in such a method.

Low-level features are expected to be extracted after only a small amount of aggregating of surface information, whereas high-level features are expected to require much more aggregation for global style trends to emerge. The creation of these features and aggregation from a local to a global scale should be learned by a model, similar to the role of a convolutional neural network for image data. Therefore, model designs used to process 3D data in various formats will be compared.

2.3 Three-dimensional Shape Models

Various models will be compared, focusing analysis on each model's ability to generate meaningful features for use in gram matrix based style representations. Desirable model traits for our purposes include the ability to apply the trained model to object classes unseen during training, increasing generalisation capabilities. Additionally, models which are fully end-to-end differentiable would allow the ability to locate precisely which regions of an object contributed to stylistic differences, as well as potentially optimising the 3D shapes towards a chosen style in a similar fashion to neural style transfer methods for images.

2.3.1 Implicit Neural Representations

Being a relatively new representation for three-dimensional data, a more limited selection of models are available which make use of implicit neural representations. They also result in a very non-standard approach to generating feature activations. Conventional deep learning models receive data provided by the user in an easily understood format leaving the model to aggregate complex relationships between simple features leading to answering a specialised abstract task such as classification which cannot be easily quantified in algorithmic terms. Implicit Neural Representations work in much the opposite fashion, taking as input a complex representation of data which is not interpretable by humans, and extracts easily understood occupancy information about specified points. The standard notion of model features becoming increasingly complex throughout successive layers is less apparent.

Earlier work by [38] lays out a framework for this class of methods. The model defined contains Rectified Linear Unit activation functions, as these are the most commonly used activation functions for tasks such as image-based convolutional models. Shapes are successfully recreated from generated latent codes as well as a trained model. Interpolation between latent codes is shown to produce meaningful shapes with traits from both shapes associated with the given codes.

Later works by [47] and [34] find that periodicity induced into the model leads to much more accurate representations of data. [47] implements this through sinusoidal activation functions, and justifies this by higher-order derivatives of the activation function being non-zero, reflecting naturally occurring phenomena better. [34] provides additional features to the standard x, y, and z sample point positions when training their model, by passing these values through sine and cosine functions of various frequencies. They show that higher frequency data such as fine-grain detail in shapes are better preserved by the inclusion of these augmented input features.

As far as can be found, no analysis has been made of the semantic meaning of model activations, particularly such as those performed in [11] where the original data is attempted to be reconstructed from model activations at different layers. This would provide insight into which aspects of a shape are represented by activations. It is hypothesised that earlier layers might break three-dimensional space up into sections, where if a query point lies on one side of a hyperplane represented by an activation function, then it can be concluded that the point must be outside of the shape. Further layers may break regions up further, performing many more inclusive/exclusive tests, before aggregating the results. This might work similarly to an octree representation, where lower levels of detail split a shape up into coarse occupancy regions, and later levels refine the boundary through further division of the space, although without the restriction of axis-aligned surface boundaries. If this were indeed the case, layer activations would provide varying levels of granularity to shape details, as well as style. However, this investigation has not yet been made and the quality of shape features produced by implicit neural representations are unknown.

2.3.2 Meshes

There are many different approaches to processing mesh data. Many earlier works have attempted to describe mesh surfaces through the use of handcrafted features. These include Signature of Histogram of Orientation(SHOT) descriptors [50], curvature, saliency, ambient occlusions, average geodesic distance amongst others. They do not try to capture features of the object at different

scales, and can therefore be more sensitive to unpredictable behaviour in the features as well as the quality of the mesh.

A few models have been developed to provide a mechanism for aggregating features of meshes, sharing some key elements. All methods choose a base component from mesh representations to use as locations for sampling input features. This could be at each vertex, edge or face. An aggregating operator is defined, which often involves the local neighbourhood of features around each location. This operator needs to be order invariant, as there is no systematic way of spatially relating neighbouring locations due to the unstructured nature of meshes. A pooling operator is also often defined, to reduce the complexity of mesh data as it is processed.

[12] uses handcrafted features per face to generate 600-dimensional embeddings. Faces are processed independently by the model to produce a compressed embedding which non-linearly combines the 600 dimensions into a lower-dimensional state. Information sharing between faces only occurs in the last stage of the model where graph-based techniques are used to enforce consistency between predictions of nearby faces.

[9] developed a model called MeshNet. Three simple features are defined per face, namely the coordinate of the centre point, "corner" vectors from the centre point to face vertices, as well as the unit normal vector. Only the centre point contains absolute spacial information, whereas the other two features are translation invariant. Corner vectors are processed using a three-position rotating convolutional kernel, taking as input two ordered vectors at a time, removing order semantics. The regularity of triangular faces is exploited, as each face is guaranteed to be connected to three other faces. Face normals can therefore be compared to their three adjacent faces, using polar coordinates to simplify the representation. Once the per-face features have been processed, they are aggregated with features from the local neighbourhood of adjacent faces through max pooling, removing the dependence on the semantic order that faces are processed in.

[16] developed MeshCNN, operating on mesh edges. Instead of exploiting the regularity of face neighbourhoods, the method instead notes that each edge connects two faces, each of which is defined by two additional edges, forming a 1-ring of 4 neighbouring edges for each edge in the mesh. The input features per edge are a 5-dimensional vector representing the angle between two connected faces, the corner angle of each face at the vertex not incident to the given edge (referred to as the inner angle), as well as the ratio between the edge length and the perpendicular height of the incident faces. The features taken from both incident faces are sorted to remove ordering

ambiguities. Aggregation of features between edges is performed by first applying a set of symmetric functions to the ambiguous pairs of edges in the 1-ring, before convolving the resulting values. A pooling operator is also defined by collapsing edges, and averaging the features of the three edges associated with each of the two incident faces. The choice of which edges to pool is determined by the magnitude of the edge features. This is expected to encourage the collapse of sparse regions which do not aid in the task of classification, while also reducing the model's dependence on meshing techniques. Repeated layers of convolutions and pooling are applied to mesh data, before global average pooling is used to aggregate the features of all remaining edges removing the remaining spatial relations between edges. This has a similar effect to a flattening operation in convolutional neural networks which also removes the spacial grid structure of images. Finally, fully-connected layers are used to predict global properties about the data such as shape classification. Additional unpooling operators are defined for U-Net based architectures [43], however for our purposes of quality feature extraction, the classification based model architecture is most relevant. A similar choice of U-Net architectures exist for image-based models, yet style research has so far only focused on classification-based feature extractors.

MeshCNN is framed using much the same core components of image-based models, redefining convolutions and pooling. This similar framework is therefore hypothesised to generate data features with similar properties to image-based methods, such as that receptive fields increase through the layers of the model, as well as later layers capturing a smoother representation of content which is less aware of surface-level details. Surface curvature is deemed to be an important feature of shape style, therefore emphasis on base features incorporating edge angle information seems promising for capturing the style of shape surfaces.

2.4 Dataset

Many 3D datasets exist, with the most commonly available format being of mesh data. The subject of these datasets ranges from very niche categories such as human poses [2], to large datasets spanning as much as 270 object classes [4]. The intended use of each dataset varies from segmenting shapes into semantic parts, content classification, pose estimation, as well as correspondences between shapes. For our purposes, a dataset which enables rich features to be extracted by our chosen model is required. The quality of data needs to be sufficient to not hinder the performance

of the chosen model, where high-quality mesh surfaces are critical.

2.4.1 Large-scale Classification Datasets

The most commonly used classification benchmark dataset is Princeton's ModelNet [54]. Available in both 10 and 40 class variants, this dataset enables easy comparison of vastly different models utilising a range of different processing techniques. As of the September 2020, 63 different models are available for comparison on a maintained leaderboard. Data quality is generally acceptable, although shapes are not required to be manifold. Various modifications have been made to the dataset, such as ensuring that shapes are orientation aligned, as well as attempting to transform data into watertight manifolds [20], although some artefacts may be introduced by this process.

2.4.2 Style Datasets

Recent papers analysing style have attempted to generate sets of shapes all belonging to distinct styles [19, 57]. Methods are then evaluated on their abilities to classify shapes into these various styles. No standardised dataset has been exclusively used across multiple research papers, where instead new papers have provided additional data to the existing dataset. The number of styles varies, from 19 in work by [19] to 33 in work by [57]. User studies are also performed comparing specific stylistic results between papers, in order to overcome the subjective nature of interpreting style. However, these user studies cannot be incorporated into new works, instead requiring new studies to be carried out for easy comparison.

Another method for overcoming the subjective nature of style analysis is through the use of triplet examples, where the styles of two positive example shapes are similar, but both are different to the style of a third negative example shape [32]. Datasets are formulated in this way to avoid providing hard labels for classifying style, instead relying on a distance metric to inform on the various styles present. These forms of datasets require that the chosen model supports metric learning.

Chapter 3

Methods

3.1 Choice of Model

The decision of which data representation and model to use was largely subjective, based on the perceived likelihood of success. As was mentioned previously, voxels and point clouds do not appear to capture adequate details of a shape's style to be able to generate a satisfactory style representation. This eliminated them from consideration for the project.

Although implicit neural representations are an exciting new avenue of research, the techniques developed have not yet been scaled up to large data sets, only training on the reconstruction of single shapes or relatively small datasets. The model needs to be able to adequately reconstruct shapes used in a style comparison if generated features are expected to have much meaning. The quality of the reconstruction informs to what extent the original data is "seen" by the model, therefore if no conditional embedding vector can closely represent the provided shapes then the model features could only be expected to represent structure from the closest reconstruction of the shape. A model would also need to be trained from scratch to produce any sort of preliminary results, as training is specific to the dataset used.

This leaves mesh representations. MeshCNN appeared to show the most promise, with the paper's experiments showing that the model is able to capture enough semantic surface information to segment shapes, as well as using a similar convolving mechanism to the those using in the original style transfer paper by [11]

3.2 Dataset

The dataset chosen for analysis was by [19], available at vcc.szu.edu.cn/research/2017/style/. Having relevant results from [19] and [57] is useful for comparing the quality of the produces results. The data is also well-behaved enough for use with meshCNN, comprising mostly of manifold mesh data.

3.2.1 Overview of Data

The data consists of 5 classes of objects, namely building, car, drinkware, furniture and leg. The furniture category contained many sub-categories referred to as bed, cabinet, chair, stool and table which are extracted and treated as separate categories, making a total of 9 classes of objects. Each class contained objects of a few different styles such as Ming, Japanese and European which are consistent across multiple classes, resulting in styles which transcend the content they are applied to. Since the final model will be evaluated on its ability to separate styles, the underlying content associated with each style must be comparable to ensure that differences identified are indeed due to stylistic dissimilarities. Shape classes whose content deviates too much between styles were considered for removal from the analysis.

The online source providing the data updated mid-way through the project, applying some post-processing effects to the meshes. Whereas in the earlier iteration, large flat regions of mesh surfaces were spanned by fewer larger faces, the updated data has an even distribution of faces throughout each mesh. This consistent distribution of faces is believed to be a desirable pre-processing step when working with meshCNN and is tested in the results section, however the resolution needs to be decreased to fit on available hardware, known as mesh simplification.

The data set originated from two sources therefore the quality and meshing techniques differ between classes. The buildings were obtained from work by [32] and are of much lower quality than the rest of the models for the purposes of our analysis. Many surfaces are non-manifold, where seemingly connected faces are modelled as two boundaries. Some structures in the shapes were modelled as two-dimensional planes without volume, contrary to the real-world equivalent, portraying the intended style only from certain angles and not others. The floors of buildings were also not modelled, meaning that the meshes were not enclosed and water-tight. The various building styles also have vastly different layouts, which impact the isolation of style from content.

The car class was another outlier, as data was much more complex than in the remaining classes. It contains a variety of different automotives, each with many intricate parts. The different "styles" present in the class would be better described as differing content, as per the definition in 2.2.2. The many detailed sections of each mesh also posed challenges for mesh simplification, as algorithms struggle to preserve meaningful shapes in topologically complex regions.

Attempts at pre-processing the data into a format accepted by meshCNN are covered in the following sections. These involved running shell scripts to pass mesh data to various external programs in an automated fashion to systematically modify each mesh. This restricts modification instruction to general rules which can be applied equally to all meshes, ideally fixing low-quality meshes while preserving details in the remainder.

3.2.2 ManifoldPlus

A recent tool developed by [20] produces watertight manifold meshes from low-quality data. Many previous tools which attempted to perform this task were not robust to some conditions, or produced noticeable side-effects. Any side-effects of pre-processing steps would be visible to the model and may affect results, being treated as valid signal. This is especially true when using gram-based style representations, as statistics aim to describe the distribution of feature vectors. Artefacts would greatly increase the variance of feature responses, decreasing the underlying style signal present in the data. ManifoldPlus was applied to the dataset in an attempt to improve the quality of the building and car classes. Results showed that the tool had trouble for some surface boundaries, producing sharp unnatural edges where none are visible in the original mesh. The tool also struggled with the lack of floors in the building meshes. This is expected as the original data was clearly not aiming for watertight meshes, and the optimal solution of placing a flat plane under each building is not an expected behaviour of the method. Instead, ManifoldPlus wraps the surface under the building, continuing the surface on the inside each building effectively doubling each surface with back-to-back faces. This indeed ensures that shapes are manifold, although the inside surfaces of buildings are not an intended aspect of the buildings that the original shape creator intended to be visible. From the model's point of view, all surfaces with a given curvature would now also contain a second copy with the inverse curvature, for example concave to convex. For these reasons, the building class was deemed unusable with MeshCNN.

3.2.3 Blender Simplification

The first attempt at simplifying up the data was using the open-source tool Blender [36]. The meshCNN code repository contains a helper script to simplify meshes, providing all interfacing code between python and Blender software. However, it does not provide much visual feedback as to what each modifier changes without simply running the script and loading the results into mesh rendering software. The parameters that can be changed seem quite limited, and further examination of the simplification process within Blender shows that not many options are available. This is because Blender's functionality is tailored to the needs of mainstream users such as 3D designers and not for research interests. After a few iterations of tweaking the output resolution of meshes, specified by a target number of faces, a desirable trade-off is reached for an initial mesh, retaining detail while greatly reducing the number of faces. However, upon applying this script to the rest of the dataset, it soon becomes apparent that these settings are quite sensitive to individual meshes and breaks many meshes beyond recognition. This is believed to be in part due to an absolute target resolution, bringing all meshes down to a fixed size such as 2000 faces. The script was implemented to reduce all meshes to a fixed size to make the training of meshCNN more efficient, reducing the need to apply padding to each batch to account for varying mesh sizes. However, for a dataset with such a large variety in mesh size and complexity, this process is infeasible. Shapes in the dataset span multiple orders of magnitude, where drinkware can be as much as 100 times smaller than cars or buildings. If a similar feature were present on two such objects, the first might preserve all detail from the original data whereas the second much larger mesh would be forced to greatly simplify the feature in order to retain enough faces to represent the rest of the mesh. A ratio simplification can therefore be used to overcome some of these challenges, however the limited flexibility of the simplification algorithm ultimately led to searching for alternative methods.

3.2.4 MeshLab Simplification

Many stand-alone algorithms exist to perform various actions to meshes. They are therefore not standardised in their choice of input mesh type, options, and each requires the user to compile the code to test out the performance of the algorithm. This is a time-consuming process when trying out multiple different algorithms. MeshLab is a research tool which streamlines this process by standardising many algorithms into a single software package. The tool comes with a mesh

renderer to visualise the results of each algorithm applied, allowing for immediate feedback as to the different parameters of each algorithm, which were found to be thoroughly documented. Each task has multiple algorithms to choose from, each taking a different approach, allowing for quick testing of various algorithms.

After some trial and error, a suitable workflow was devised to simplify mesh data. Processing starts with a remeshing to generate a more even distribution of edges across the mesh, using isotropic explicit remeshing. Because we only want to modify the density of edges and not affect the underlying shape, no deviation from the surface of the original shape is permitted. Refine, collapse, edge-swap, smooth and reproject operations are allowed within the remeshing parameters to aid the algorithm to converge. A simplification algorithm is then applied, known as quadratic edge collapse decimation, to reduce the computational burden of later analysis. A reduction to 10% of the original size was chosen, which seemed appropriate given the resolution of the original data. Post-simplification cleaning was disabled as this was found to generate much more non-manifold elements which need to be removed for the mesh to be used by the meshCNN model. A final process of splitting vertices connected to two or more non-adjacent faces was performed, removing artefacts created by the previous algorithms. There is also the possibility of removing non-manifold faces as another process in MeshLab, however an implementation already exists within the meshCNN model, aiding compatibility of data that has not been pre-processed with external tools before use.

MeshLab can generate a script of all algorithms used, which can be applied from the command-line. A bash script is written to apply the script sequentially to each mesh file in the dataset. When training meshCNN on the data, it is found that a few meshes were still too large to process using an efficient implementation of the model, therefore the script automatically removes meshes with more than a threshold number of edges, set to 20000.

An example modification of mesh data is shown in figure 3.1.

3.3 Modifications to the Model

Much time was spend getting a working copy of meshCNN operational. A few modifications then needed to be made to the model, due to a few reasons. The first being that a few sections of the model were not robust to features present in our dataset, which were not seen in the pristine manifold data used in the original paper. The second reason was that the original implementation

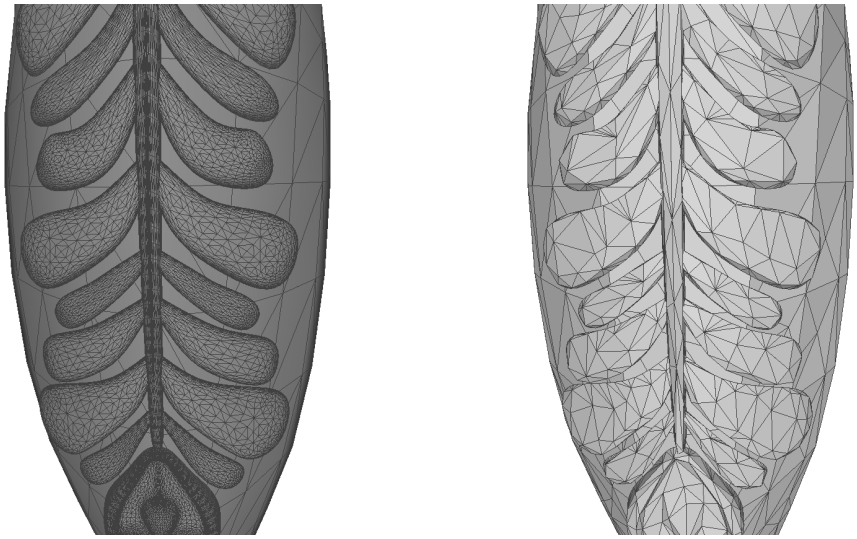


Figure 3.1: Before (left) and after (right) pre-processing steps have been applied to data by [19].

did not use many standard conventions used when building deep learning models. Refactoring, documentation and transitioning much of the boilerplate code to a dedicated library helped to clean up the code base and ease the development of the analysis performed in the report.

3.3.1 Implementations

At the time of writing the report, meshCNN has three popular implementations available online. These are from the original meshCNN paper [16], the point2mesh paper [17] and the Kaolin library [22].

3D Deep Learning Frameworks

Many 3D and geometric deep learning libraries exist. They aid in formatting the input data as well as provide easy to use components from which models can be implemented. This can greatly speed up development time as much of the same code is needed across many model implementations to read in data files and build up a representation. These libraries require a few years to establish themselves, as the right level of abstraction is not easy to predict in the early stages of an emerging field. Multiple libraries competing to perform the same task mean that new models are often built from scratch before being reimplemented in various libraries. In the PyTorch framework [39], the

main libraries are Pytorch3D [41], Kaolin [22], Pytorch Geometric [10] and Deep Graph Library [53]. Of these, Kaolin is the only library to have integrated meshCNN into their model zoo, a collection of ready-to-use models. It appeared that Kaolin's implementation was the most recent with an advertised 110x speedup over the original implementation for some operations in the model. MeshCNN's author also praises this implementation making this the first implementation explored in the project.

Kaolin Implementation

Much time was spent on the Kaolin implementation. The first issue was one of dependencies. Being a large library offering a lot of functionality, many other libraries are relied upon, with specific versions needed to work with Kaolin. Many attempts were made to set up the proper environment to use the library, from installing all the requirements manually, followed by trying the provided Conda file which supposedly installs the necessary packages without further instruction, to trying a provided Docker file to build a custom virtualised contained with a dedicated operating system in an attempt to emulate the intended system requirements identically. None of these solutions worked, and eventually the manual process of systematically testing different versions of each required package was performed until a successful installation was made. The problem was caused by some of the specified libraries not including a version number, meaning that they defaulted to the latest version which was not compatible, combined with dependencies also not including fixed version requirements meaning python package managers such as Conda and Pip could not automatically find a valid combination of dependency versioning.

Once the environment was established, more problems were encountered with the implementation. The example given in the repository raises many errors upon running, and after fixing these errors the training times were very slow, much slower than those reported in the original paper. This implementation ultimately ended up running three times slower than the first implementation written by the paper authors. After an issue was raised with the Kaolin library authors, they confirmed that this was a known issue that was not documented, and as of this report have not fixed the issues or accepted pull requests of fixes by other users. This lead to looking elsewhere for a working implementation.

Original Implementation

The original code repository accompanying the meshCNN paper was tried next. This has the advantage of being the canonical implementation, meaning the results should best reflect those found in the meshCNN paper [16]. The code-base was however quite inconvenient to work with, having no comments explaining the purpose of large sections of code. There was also large quantities of boilerplate code which heavily obfuscated the control flow of the program. Because this implementation of meshCNN seemed most promising, attempts were made to clean up the existing code to enable faster prototyping of methods presented in this report. Documentation was made of all existing code to facilitate modifications, of which a few were made. These are detailed in the following sections.

3.3.2 Processing of Large Meshes

Typical meshes can have anywhere from hundreds to hundreds of thousands of faces. This poses a problem for deep learning models, as hardware memory is limited. Therefore the memory footprint of a model should be carefully considered, to not unnecessarily restrict the possible size of input data.

The original meshCNN implementation pools edges sequentially. New embeddings for each edge after pooling are a result of averaging surrounding edges. Specifically, when a face is pooled, the resulting edge contains the average embedding of the three edges which define the face. This resulting pooled edge may then be further used in another pooling operation where more embedding averaging occurs. Therefore the original implementation optimises this process using what they refer to as bookkeeping. A large n by n matrix is created to store which edges from the unpooled mesh need to be averaged. Each row represents an edge, and each column represents the edges whose representation will be averaged and stored in the edge of the corresponding row. This greatly speeds up computation as actual embeddings need not be indexed and averaged after each pooled face, and can instead be computed in a single vectorised matrix multiplication operation. The problem with using this method is that the memory requirements grow at a rate of $O(n^2)$ where n is the number of edges. This quickly becomes unwieldy for large meshes, so solutions need to be devised to further optimise this process. It is by far the most memory-intensive process in the model.

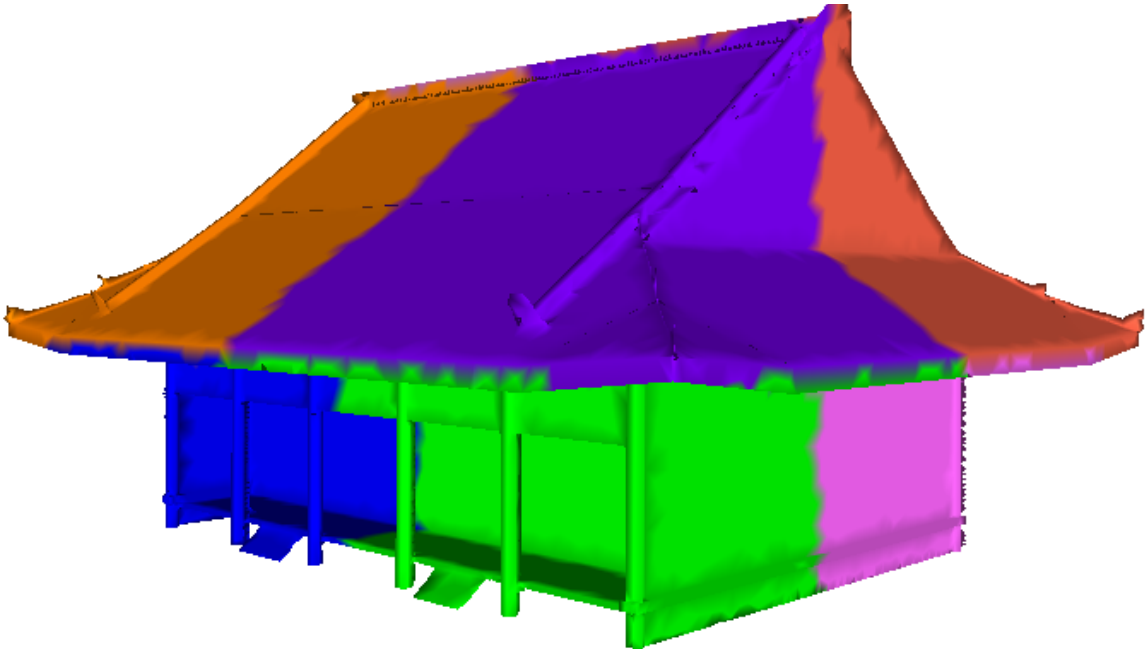


Figure 3.2: Building shape split into 8 submeshes, one for each quadrant. Building data by [19]

Part-mesh Splitting and Recombining

The follow-up paper from meshCNN, called point2mesh [17], uses the same underlying model. However due to a need for processing large high-quality meshes, methods were devised to deal with the overwhelming memory requirements.

A highly engineered approach to a solution was implemented by [17] where meshes are subdivided into multiple parts, referred to as sub-meshes and shown in figure 3.2. These parts are then processed by the model separately, before finally being recombined afterwards. A problem with this approach is that it causes unnatural boundaries in the processed sub-meshes which affect the computed embedding. This is because the receptive field of a boundary edge, the surrounding edges which affect the computed embedding, are not all present in a sub-mesh. These effects are reduced by allowing an overlap between each sub-mesh, before averaging embeddings belonging to edges present in multiple sub-meshes.

This solution avoids the root cause of the problem, which is that memory requirements grow unnecessarily large as the resolution of meshes increases.

Sparse Matrix Implementation

A different approach was taken for this project, as generalisation capabilities and ease of use were an important consideration for applying the developed methods to as many different types of unseen data as possible.

It is observed that when building the matrix of edges to pool, most entries remain unaltered at zero. The main diagonal of the matrix always contains non-zero values, as edges always use their own embedding as part of the averaging process. Depending on how many pooling operations are performed, the number of non-zero elements to a row could increase to three or sometimes more than ten. However, in comparison to the total number of edges in a mesh, the number of non-zero elements is orders of magnitude smaller. This bookkeeping matrix can therefore be considered a sparse matrix, a data type which can be efficiently stored in memory, only requiring storage of non-zero values as well as their corresponding coordinates. Since the number of non-zero elements per row can be confidently bounded by a small value under all but the most pathological cases, the memory requirements can be considered $O(n)$ on average for bookkeeping using sparse matrices.

Unfortunately, not all functionality of dense matrices is currently present in the PyTorch [39] implementation of sparse matrices. This requires understanding the underlying data structure of sparse matrices. The implementation used by PyTorch is in COO format, short for coordinate format. For a matrix with n non-zero entries, the underlying data consists of a values vector of length n, as well as an index matrix of dimensions $2 \times n$. The initial bookkeeping matrix can therefore be formed from a value vector of ones of length n, as well as an index matrix of the form $[0, 1, \ldots, n-1]$ for both coordinate rows. To add a row of the matrix to another for pooling, the relevant entries are extracted using built-in row indexing functionality, but are then modified to change their row index to the desired row before being added back to the matrix.

The main difficulty encountered was with applying a Boolean mask to index only relevant edges that were not removed during pooling. This is because the underlying matrix data does not appear in an ordered grid of memory to be indexed into. The only solution found online was to loop through a set of relevant indices and extract elements where the index is present in that set. This is found to be the main bottleneck of this sparse approach, slowing computation times by between 8x and 10x. However despite these disadvantages, the solution can handle a much larger number of edges, and subsequently much larger mesh data. This method is much more straightforward to utilise

than the previous solution proposed by [17], and scales better for larger meshes. Experimentally, meshes with as many as 8 times more edges could be processed on available hardware.

3.3.3 Pooling Operator Stability

Although the original implementation [16] is found to work without error for the dataset used in the analysis from the original paper, external datasets cause problems to arise. These have been raised by other users of meshCNN in the repository's issue tracker, and a solution is proposed.

Connected Components

The original implementation assumes that the input mesh is comprised of a single manifold component. In graph theory, this is referred to as a connected graph [51], where every node has a path to all other nodes. The model works somewhat with meshes comprised of multiple components, however the model assumes that all meshes can continue to be pooled indefinitely. This is mostly accurate for single manifold component meshes as pooling can continue to occur until only a single face remains, which would not occur under normal use. Pooling is generally not used to collapse all spacial information and indeed in the meshCNN paper, pooling occurs only until 279 edges remain [16]. These assumptions break down for meshes made from multiple components.

Un-poolable Models

Pooling occurs unevenly across different sections of an input mesh, depending on the magnitude of an edge's feature vector. Therefore it often occurs that some components of a mesh are pooled much more than others, due to either the component being small, or having consistently low magnitude features. It is then possible that some components are pooled down to a single face while other components of the mesh can be pooled further, as shown in figure 3.3. The model initially breaks when trying to pool single face components, as the 1-ring neighbourhood of each edge consists of the two other edges repeated twice, which should not occur under normal circumstances. This edge case is skipped, preventing further pooling. However some meshes seen in the dataset contain such a large number of connected components that they are all pooled down to single faces where they cannot be pooled further, leading to the model being unable to compress the mesh to the desired number of edges for the specified pooling operator. This problem is overcome by identifying when this situation arises and halting the pooling operation, allowing the model to continue to process

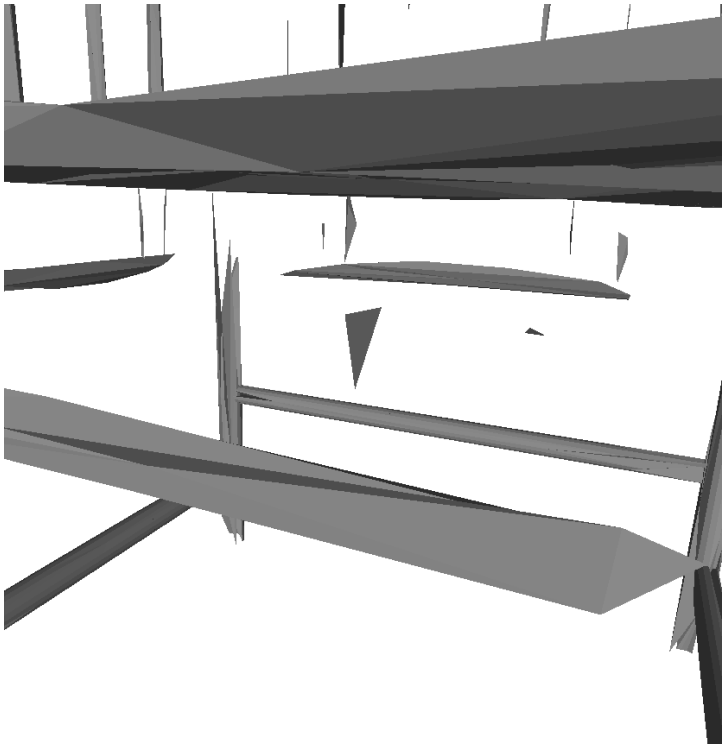


Figure 3.3: Component pooled down to a single face, where other components can be pooled further. Chair data by [19].

the data. Appropriate padding is applied to other meshes in a batch as all meshes in the batch can no longer be guaranteed to be of the size specified by the pooling. This prevents all pooling-related errors caused by different datasets.

3.3.4 Visualisations

The original implementation of meshCNN [16] includes a custom rendered to visualise edge colourings for the purpose of mesh segmentation. This renderer is quite restrictive in its features, such as not colouring faces with the average colour of surrounding edges, as well as limited panning and camera controls. Being able to harness the powerful features of a stand-alone mesh rendering software was deemed an important feature, especially to ease the integration of the developed methods into a professional workflow. This is achieved through the ability to add colour information to mesh obj files to be interpreted by external programs. The process of generating visualisations for pooled

meshes throughout the forward pass of the network was also simplified and extended for use with edge colourings.

Edge Colouring

Standard mesh formats do not support edge colouring information, however the obj file format used by the style dataset supports vertex colouring. A solution to representing edges colours in the output file was devised, by the addition of more nodes and faces to mesh files. Specifically, for each edge of a mesh, a new vertex is added, positioned at the midpoint of the edge and equidistant from the vertices which define the edge. These new vertices are used to store the desired colour values for each edge. Original vertices of the mesh are coloured with the average colour of incident edges, a necessary step to generating a valid mesh file as each vertex requires a colouring for vertex colourings to be used.

A new face is also added per edge, whose vertices are those associated with the edge, as well as the new vertex defined at the edge midpoint. This is to ensure that new vertices are connected to the mesh and contribute to the visualised mesh colouring.

Visualisation of Pooling Retrospectively

An important feature for model analysis is detailed visualisations of each stage of the process. For meshCNN, the remaining information which model activations do not provide is the dynamics of edge pooling throughout the forward pass of the network.

The original implementation of meshCNN [16] contains a method of creating a mesh file from the pooled state of a mesh, however it does so during the forward pass of the network. This is quite limiting as information needed to colour the edges of the pooled state of the mesh may only arise after a full pass of the network has been completed, for example after a loss is computed and gradients backpropagated through the network. Therefore the method for storing the intermediary state of data through the model is modified to allow for recreation of pooled mesh states at any later time after the forward pass occurs. This is achieved by saving the internal state of vertex positions instead of relying on the current state at the time of model inference. A vertex mask is also computed relative to the original mesh vertex ordering, instead of relative to the state of the mesh before each pooling stage occurs. These features can be disabled during the training of a model, to remove the time and memory requirements of saving these extra states, and can be

re-enabled for data visualisation and analysis. Examples are shown in figure 4.16 from the results section.

3.3.5 Additional Modifications

A few additional modifications have been made to the code-base, which are not substantial enough to warrant their own section, but greatly aid usability of the code. The meshCNN repository contains many contrived methods of formatting and implementing Pytorch features. These have been standardised to enable faster prototyping, as well as enhance features through offloading much of the verbose code to dedicated libraries.

Data transformations and augmentations allow trained models to generalise better to noisy test data. It is important that they are clearly laid out as they make assumptions about the data, as well as how the model views the data. Augmentations already implemented into meshCNN include rescaling input edge features, adding noise to vertex positions, as well as flipping the orientation of edges. All of these steps have been modularised and are accessible from the PyTorch dataset class, meaning that all augmentations are visible in a single location, and further augmentations may be easily added.

Previously, mesh target classes were determined by the file structure of the data, where all shapes of each class are located in distinct folders. This process has been modified to also accept the first section of the file name (before the first underscore) as a style class, which can later be used as the target for training a model.

Input edge features to the model are computed as a function of the position of vertices associated with the two faces incident on each edge. These computations were originally implemented in numpy [35], a numerical computational library which supports efficient operations on n-dimensional array data. These have all been ported to equivalent PyTorch operations, which enable to the ability to backpropagate model gradients not only to each edge, but also to the vertex position which generated the initial edge features. This fully-differentiable process allows for vertex positions to be adjusted through backpropagation, optimising the underlying mesh data based on model targets such as predicting a certain style or class. A related method was implemented in the follow-up paper to meshCNN, point2mesh [17]. However their methods differ in that they attempt to predict vertex displacement directly as the output of the model. Their methods attempt to generate a mesh which accurately models the shape of point cloud data, through optimisation of an initial

convex hull. A perfect fit can be made for their purposes, where every point in the point cloud data lies perfectly on the optimised mesh, although an imperfect solution is sought which smooths out noise in the point cloud data as well as harness self-similarity in repeating patterns of the mesh to guide the solution. Commentary in the paper suggests that optimising vertex positions directly for their purposes leads to overfitting. This is believed to be due to the nature of their model objective, which can achieve a perfect fit without regularisation. In contrast, the task of optimising the vertices of a mesh towards an abstract goal such as a similar style representation of another mesh is hypothesised to not be as affected by local optima.

The code-base also contained a contrived method of training the model. A class is used to prepare the data, perform the forwards and backwards passes through the model, save and load models, as well as evaluate performance. This system is replaced with the pytorch-lightning library [8] which maintains all current functionality while being thoroughly tested to ensure the boilerplate code works as intended. Many features come with using the library which are particularly relevant to training a model with the dataset used. Mixed precision training is automatically handled, which allows the use of 16-bit floating-point integers where appropriate to store data as well as model weights. This is found to be crucial for large meshes as it greatly decreases memory requirements allowing batch sizes of 8 instead of 4 on the limited hardware used for the project, an NVidia RTX 2080. Gradient accumulation is also taken care of, allowing an effective batch size to be chosen regardless of the batch size used. Model gradients are accumulated across multiple batches until the desired number of data examples have been seen. This allows for effective batch sizes of 32 to be used by accumulating 4 batches of gradients at a time, reducing the effects of class imbalances as well as the high variance of mesh resolutions by allowing the model to observe a larger variety of meshes between parameter updates.

3.4 Style Analysis

3.4.1 Gram Matrix Analysis

A gram-based style representation is generated in much the same manner as in image style analysis by [11]. Mesh data is passed through a trained model, in our case meshCNN, and model activations are extracted at each level of the network after a convolution has occurred. Because the trained models contain 4 convolutional operators at different data resolutions between pooling op-

erations, 4 different activation tensors are produced for analysis. From these, style representations are calculated.

The original formulation of gram matrices for style representations presented in equation 2.1 treats each pixel in an image equally, flattening the two-dimensional grid of pixels into a single dimension effectively removing any spacial information about the relation between feature embeddings. A similar formulation can be made for features from the meshCNN model [16], by not retaining any spacial relation between edges. This is convenient as the data is already presented in this format, with a separate accompanying data structure used to store the spacial relation between edges left unused during analysis. The activation matrix A_l for each layer l in of dimension $N_l \times C_l$ where N_l represents the number of edges at layer l and C_l represents the lengths of edge embeddings at layer l, referred to as the number of channels. For a given layer activation, the Gram matrix G_l is therefore computed as $\frac{1}{C_l \cdot N_l} A_l^T A_l$, where $G \in \mathbb{R}^{C_l \times C_l}$. This normalises for the varying number of edges in each layer, as well as the size of gram matrix, enabling the distance between gram matrices to have comparable magnitudes across each layer as C_l varies.

We can analyse these gram matrices directly, assessing whether similarly stylised meshes produce gram matrices of closer proximity to one another. In practice, this involves generating a gram matrix for every mesh in the data set. As mentioned in section 3.2.1, low-quality and overly complex shape data was removed from the dataset, leaving 561 different meshes spanning 9 styles. Gram matrices are flattened, resulting in vectors of lengths $64^2, 128^2, 256^2, 256^2$ from activations of the four layers of the model. Experiments are then performed on these vectors. Dimensionality reduction techniques are used to visualise these high-dimensional representations on a two-dimensional plane, with results from t-SNE and UMAP compared. Linear classifiers are then fit to the vector representations, indicating the linear separability of the various styles present in the data. Due to the high-dimensional nature of the representations, regularisation is applied in an attempt to not overfit the large number of parameters.

3.4.2 Style Dissimilarity Visualisations

Gram matrix style representation encapsulates global statistics about correlations between feature embedding channels. We have chosen to compare gram matrices element-wise using a mean squared error loss, where each element contributes to the loss with equal weighting. This aggregated loss value provides a coarse overview of similarity between styles of two meshes, however automatic

differentiation allows the loss to be back-propagated all the way through the model to the mesh data, revealing the key factors which contributed to dissimilarity (as well as similarity) to be traced back to individual edges of a mesh.

Calculated gradients are presented as a five-dimensional vector per edge, corresponding to each of the five base features used as input to the model. Techniques used to transform three-dimensional gradient vectors for images corresponding to red, green and blue channels into a single dimension saliency value per pixel can also be applied to our five-dimensional vectors. Many variations exist [37], such as taking the maximum value of each gradient vector [46]. The sum of the absolute gradient value was chosen, as the sign of the gradient is arbitrary in our case due to a gradient of zero representing perfect style agreement. To render these values on a mesh, normalisation needs to be applied to obtain values between zero and one. Subtracting the minimum value and dividing by the range ensures that regions of the saliency map express the entire range of possible values. However when extreme values are present, smaller values are compressed towards zero due to dividing by a large range, rendering small changes in values as imperceivable. This transformation also removed the ability to make relative comparisons between the magnitude of disagreement between various styles. The proposed solution to this problem was to first compute the sum of absolute gradients for all meshes, then obtaining a threshold value as a percentile of these values, such as at 80%. Now, using zero as the minimum value and the computed percentile as the maximum, saliency maps may be visualised without extreme values drowning out smaller values, as well as being able to make relative statements about perceived magnitudes between saliency maps.

Gram based saliency maps presented in this report do not have as many restrictions as previous methods discussed in section 2.2.3. The quality of style representations is only dependent on the quality of extracted features produced by the chosen model. Gram matrices present a detailed view of correlations between feature channels, therefore unidentified styles only occur as a result of extracted features not presenting a distinctive enough signature. Other benefits of our method are that saliency maps are naturally produced at every point on shapes, and specific styles to not have to have been observed previously to perform analysis.

To perform repeated comparisons of new data to a corpus of reference shapes all belonging to a specific style, the loss function needs to use gram matrices from each reference shape. To save on space and compute time, an average style representation may be generated by averaging the gram matrices of each reference shape. This method will be shown by generating an average

style representation per style in the dataset. Test shapes will be compared to each of the styles, highlighting the various regions of disagreement particular to each style.

3.4.3 Pre-processing Importance Analysis

The receptive field of meshCNN is described in section 2.3.2. Feature embeddings for mesh data are therefore hypothesised to be highly dependent on the specific meshing of data, affecting the size of local regions seen by each edge. The effects of different meshing techniques are studied, comparing embeddings generated from meshes in their original format from the dataset, as well as data that has been pre-processed using techniques presented in section 3.2.4.

3.4.4 Style Clustering

Extensive research has been produced on gram based style analysis for image data [29, 21, 23] as is mentioned in section 2.2.1. Many properties have been observed on the nature of style throughout an image, as well as the various behaviours of gram matrices. It is investigated whether these findings transcend purely image-based style analysis and can be observed in the 3D domain, specifically in the generated representations from our work. This would suggest that gram based style representations behave in much the same way across domains, and are not simply a special phenomenon only observed with texture-based features or data with a regular grid structure.

In particular, the methods from [58] will be applied to mesh data. In their paper, they perform dimensionality reduction of feature activations, and observes that distinctive clusters appear, identifying multiple distinct styles within images. Gram matrices can then be formed from individual clusters of features, to capture a more specialised style representation for the various styles within an image, instead of requiring that the gram statistics capture the style of the whole image. This clustering is performed on feature activations from our data, with the separate clusters visualised on the underlying mesh through edge colouring methods developed in section 3.3.4. The expectation is that distinctive style traits would be segmented, revealing the individual components which compose together to form an overall style.

Chapter 4

Experiments and Results

4.1 Gram Matrix Analysis

Gram matrices were generated for every viable mesh in the dataset, leading to high dimensional style representations. To visualise the relations between these representations, the dimensionality reduction tools t-SNE [52] and UMAP [33] have been used. t-SNE and UMAP have as parameters perplexity and number of neighbours respectively, which indicate to which extent the algorithm should focus on local vs global patterns in embedding space. Low values of these parameters lead to the data being separated into many local clusters, whereas larger values allow the algorithm to focus on trends throughout the entire dataset, more meaningfully placing local clusters in relation to each other. Various values of these parameters are explored for both t-SNE and UMAP to better visualise the emergent structure in figures 4.1, 4.2, 4.3, and 4.4. It should be noted that these tools do not preserve meaningful distances between points in the new lower-dimensional subspace, and are purely for human analysis of the general structure present in the representations.

These plots reveal many global properties of the dataset, such as class imbalances. Many more instances of the Ming style shown in pink can be seen throughout the embedding space, compared to the relatively few instances of Chinese style. Although distinct clusters have not emerged to cleanly separate the various styles, many regions of increased density of certain styles are seen throughout the plots. In 4.1, embeddings from layer 1 shown all Chinese mesh embeddings isolated from other points, shown in dark green. In figure 4.2 layer 1, most Japanese style embeddings are shown on the right-hand side of the figure. Layer 3 from the same figure shows a high concentration of Children styled points in the lower right-hand side in orange. Clearly these preliminary results show that there is some meaningful style signal captured by the representations.

Both [19] and [57] produce model accuracies for classification of mesh styles, therefore comparable results are also produced from our methods. Due to the incompatibility of building and car classes for use with the chosen meshCNN model, they are omitted from the comparison.

The provided results from [19, 57] classify styles within each category of data, for example distinguishing between Straight, Smooth and Cabriole styles from the furniture leg dataset without attempting to distinguish these styles from drinkware or generic furniture styles. There are three groups of mesh content from the data which models are trained with. 10-fold cross-validation is used, however due to the limited size of the dataset, there can be a little as 9 data points per test fold. Support vector machine (SVM) classifiers have proven to be effective for classifying styles from image-based gram matrices [5], therefore an SVM is also used for our model. A variety of input features were experimented with, such as only using the gram matrix from a single layer, concatenating multiple layers of representations, as well as performing dimensionality reduction on these features. Results were found to be moderately improved using layer 2 activations, and using PCA dimensionality reduction to 20 dimensions, and are provided in figure 4.5.

A model was also trained to classify all 9 styles simultaneously. 10-fold cross-validation training is performed on representations from each layer separately. Results are compared to a baseline accuracy of predicting only the most frequent class, as seen in figure 4.6.

4.2 Style Dissimilarity Visualisation

A few examples are given comparing the style of a reference mesh with a target mesh to be analysed. Saliency maps are generated from the loss at each layer, as well as from a loss which utilises all four

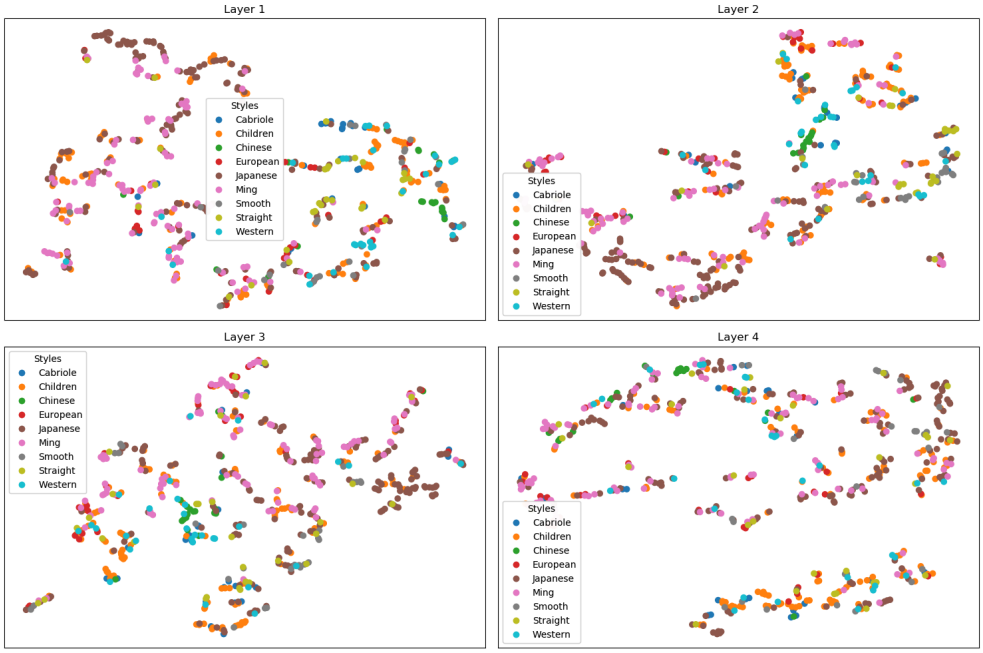


Figure 4.1: t-SNE dimensionality reduction with perplexity set to 5

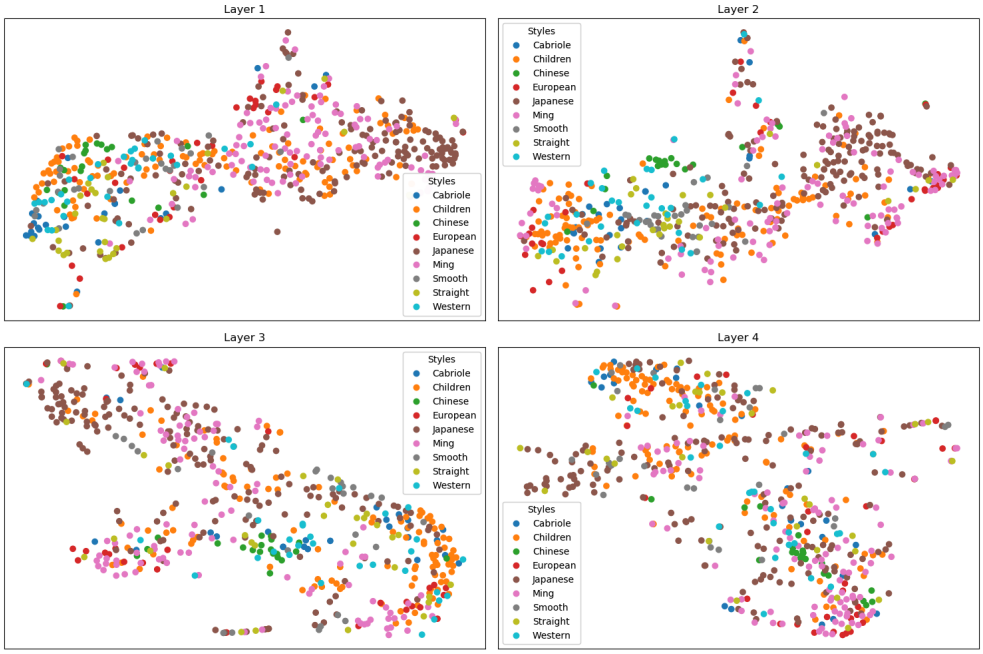


Figure 4.2: t-SNE dimensionality reduction with perplexity set to 50

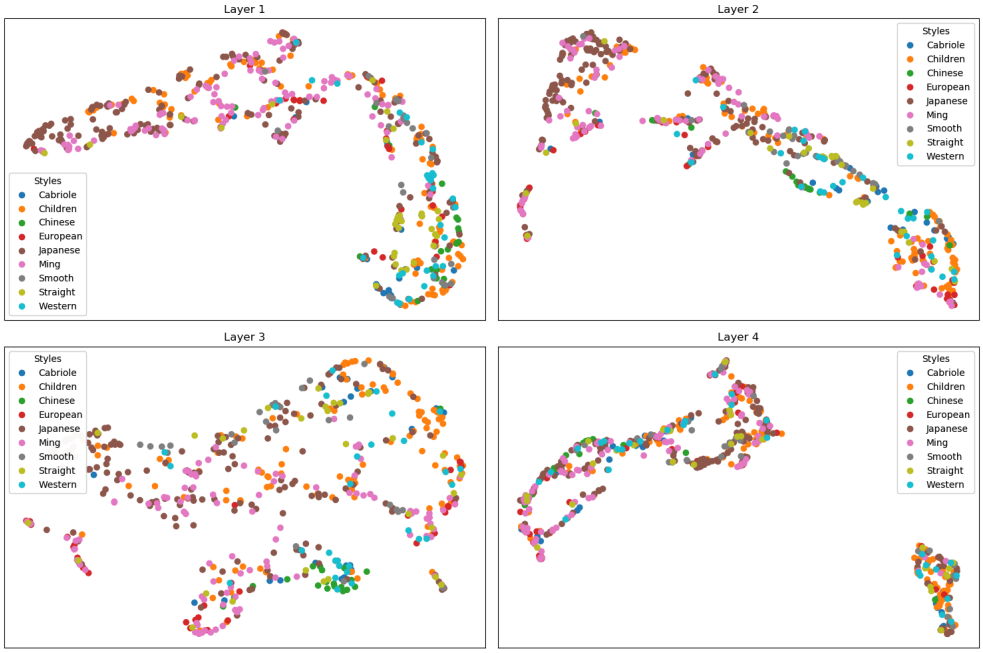


Figure 4.3: UMAP dimensionality reduction with number of neighbours set to 15

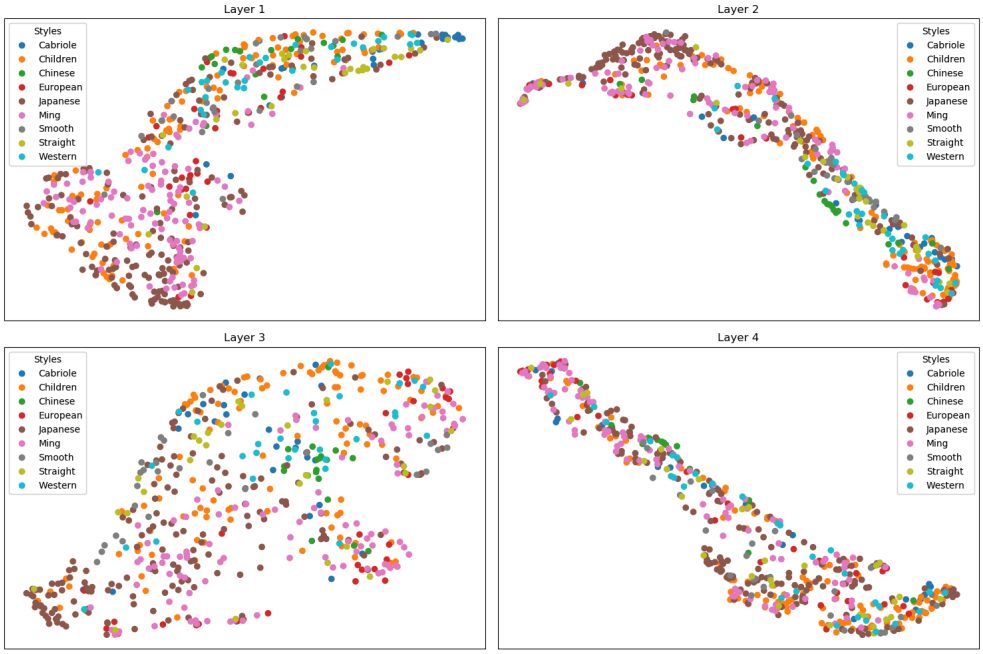


Figure 4.4: UMAP dimensionality reduction with number of neighbours set to 200

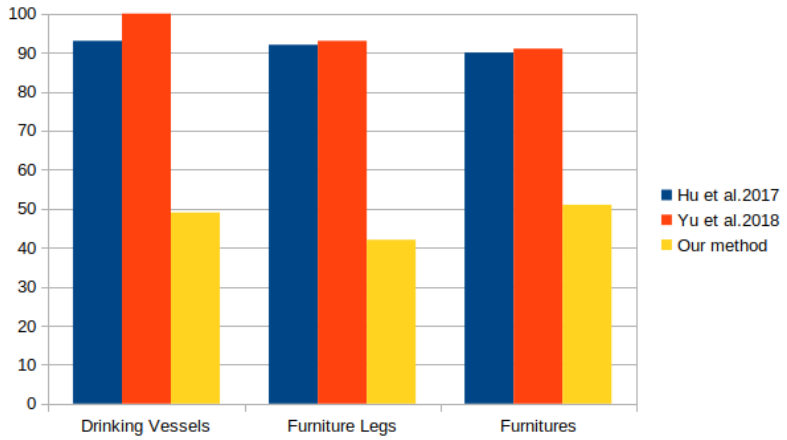


Figure 4.5: Comparison of style classification to previous works.

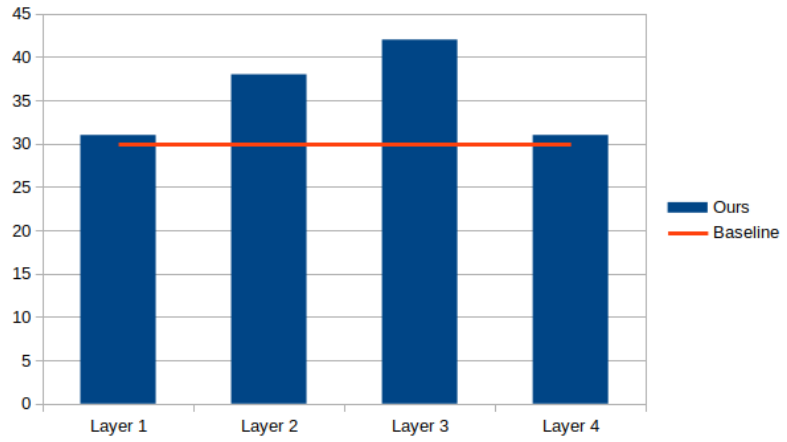
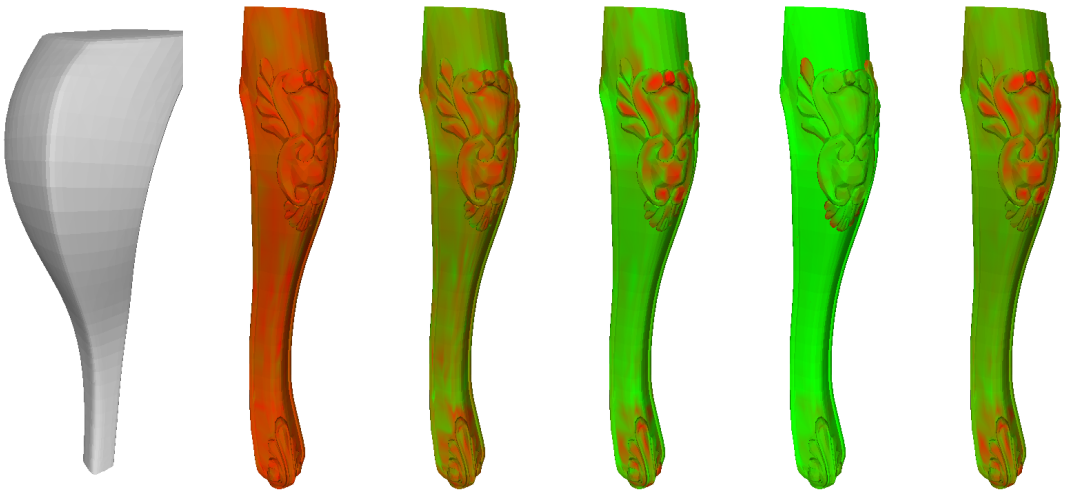


Figure 4.6: Classification performance for all style concurrently with baseline.



(a) (b) (c) (d) (e) (f)

Figure 4.7: Comparing style similarity to reference mesh Smooth_4 (a). Saliency map of loss from: (b) Layer 1, (c) layer 2, (d) layer 3, (e) layer 4, (f) all layers (1 to 4)

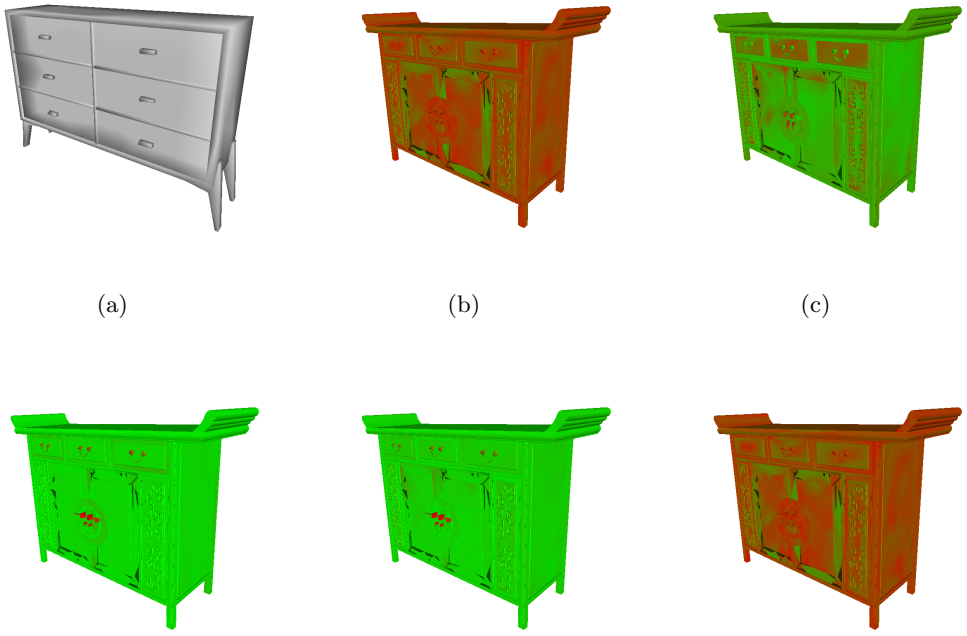
layers. These are shown in figures 4.7 and 4.8.

In figure 4.7, the floral texture visible along the front and bottom of the mesh can clearly be seen to distinguish Cabriole style from a Smooth style. These regions are highlighted by some of the saliency maps, with particularly good results in saliency maps generated from layers 2 and 3, as well as the saliency map taking into account all layers.

In figure 4.8, the reference mesh is a simple Japanese style cabinet composed of many flat surfaces, without any smooth curved or narrow tubing. The intricate geometric motif visible on sides of the Ming cabinet are highlighted in the saliency map from layer 2, as well as the intricate handles being identified at layers 3 and 4.

The consistent good quality saliency maps generated from layer 2 lead to choosing that layer for use in a large scale style dissimilarity analysis across all styles. For each style, an average gram matrix style representation is computed. Sample meshes from each style are compared to the average gram representation per style, as shown in figure 4.9.

It can be inferred that European, Japanese and Ming styles are the most distinctive, from the large dissimilarity visible when comparing these styles to those of all other styles of mesh. For



(a) (b) (c)

(d) (e) (f)

Figure 4.8: Comparing style similarity to cabinet reference mesh japanese_8 (a). Saliency map of loss from: (b) Layer 1, (c) layer 2, (d) layer 3, (e) layer 4, (f) all layers (1 to 4)

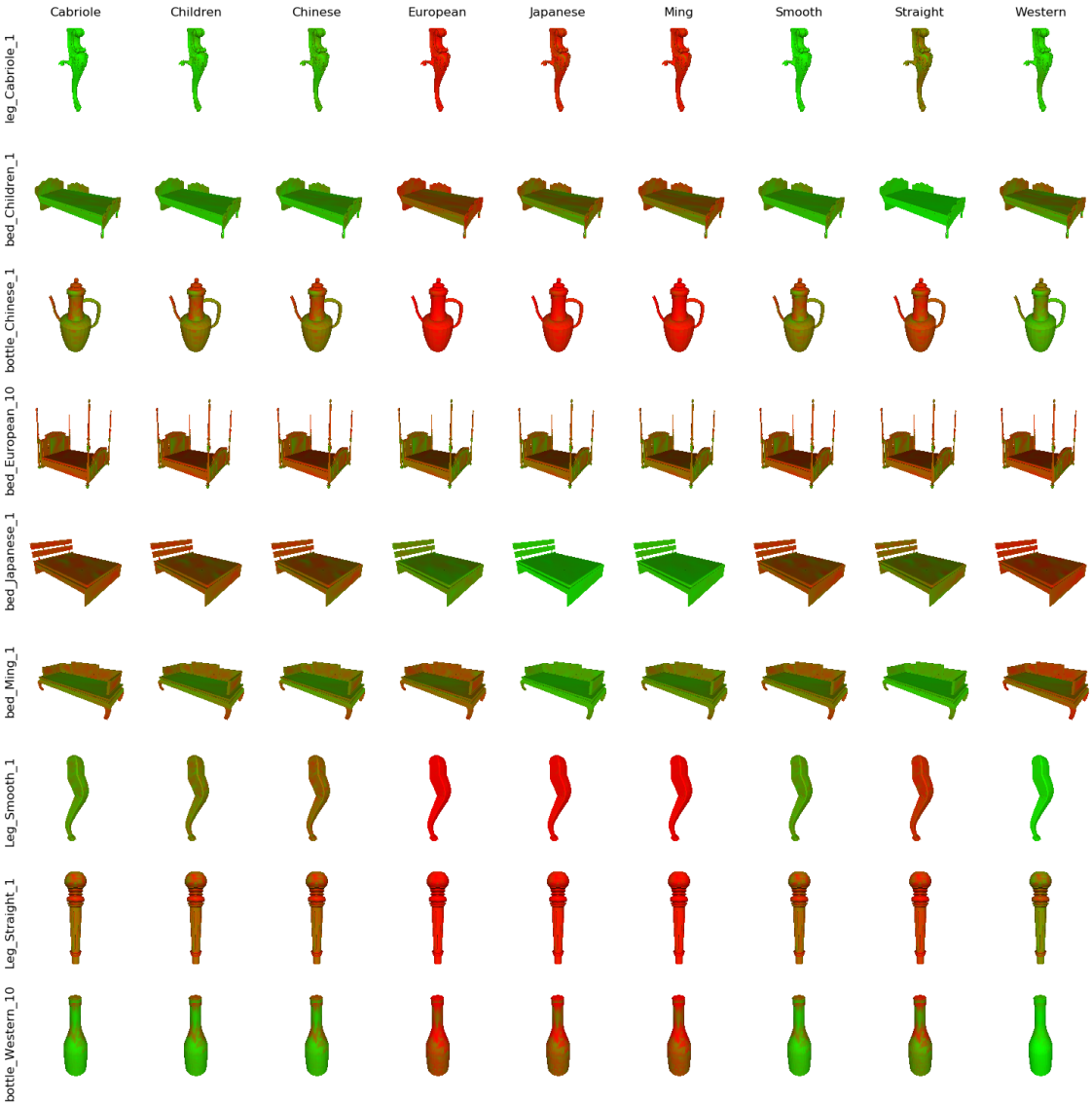


Figure 4.9: Each row represents a sampled mesh from each of the 9 styles, and each column represents the average representation per style used to compute saliency maps.

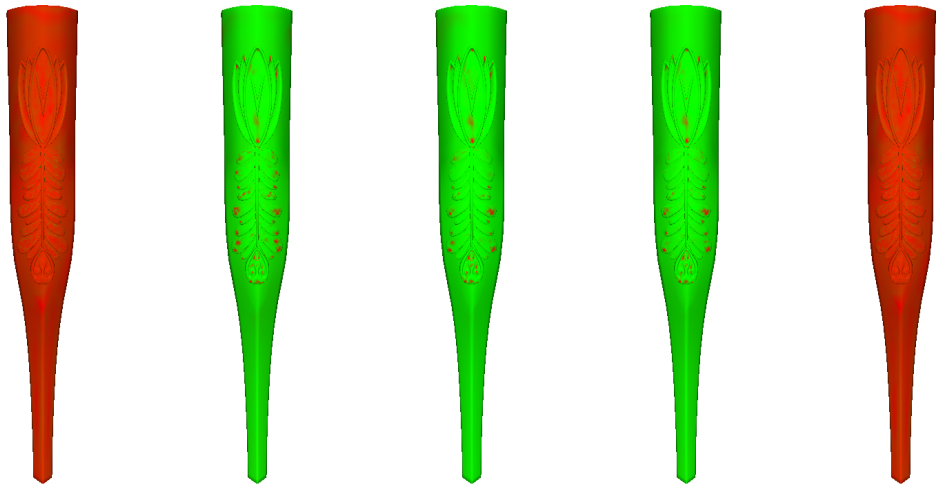
Japanese styles, it is thought that this is due to all examples being composed of mostly flat surfaces with sharp right-angle bends without the use of any curves. Therefore wherever curved surfaces occur, large quantities of dissimilarity are shown. The Ming bed is the only other shape without large amounts of dissimilarity to the Japanese style, presumably due to the abundance of flat surfaces and right-angles across the surface of the bed mesh used for comparison.

Observing the Chinese bottle mesh, there is disagreement with European, Japanese and Ming style as is to be expected. There is also disagreement with the straight furniture leg style which represents mostly straight flat surfaces. The remaining styles mostly suggest stylistic agree with the smooth curved body of the bottle, however the more extreme curves present on the lid provide a much larger disagreement. Interestingly, the only style which indicated stylistic similarity with the bottle lid is Western, which also contains bottles with lids. The suggests that the gram matrix style representation is flexible enough to "identify" the traits of semantically meaningful shape parts such as lids.

4.3 Pre-processing Importance Analysis

A comparison is made of the effects of data pre-processing on the quality of saliency maps produces. Style representations are generated for furniture leg mesh Smooth_6, as well as two copies of Cabriole_10. A close view of both versions can be seen in figure 3.1, where the first shows the data as it is provided from [19], and the second version is obtained after applying pre-processing steps described in section 3.2. Due to the reference style being smooth without details, similar results are expected to those seen in 4.7, where all additional textures and motifs and identified as a dissimilar style. Indeed in the pre-processed data, these features are correctly identified, as seen in figure 4.11. However, using the original meshing shown in figure 4.10, only a few edged along the region expected to be dissimilar are highlighted. This is thought to be because edges which show the largest difference in style are along the boundaries of connected components as well as in areas of high curvature, where edge features deviate most from the style of the smooth mesh. The higher density of edges means that those representing the more extreme features dominate the saliency map.

The base features of meshCNN [16] do not encode shape scale, meaning that properties such as the rate of curvature can only be generated from averaging the angle between mesh faces in a



(a) (b) (c) (d) (e)

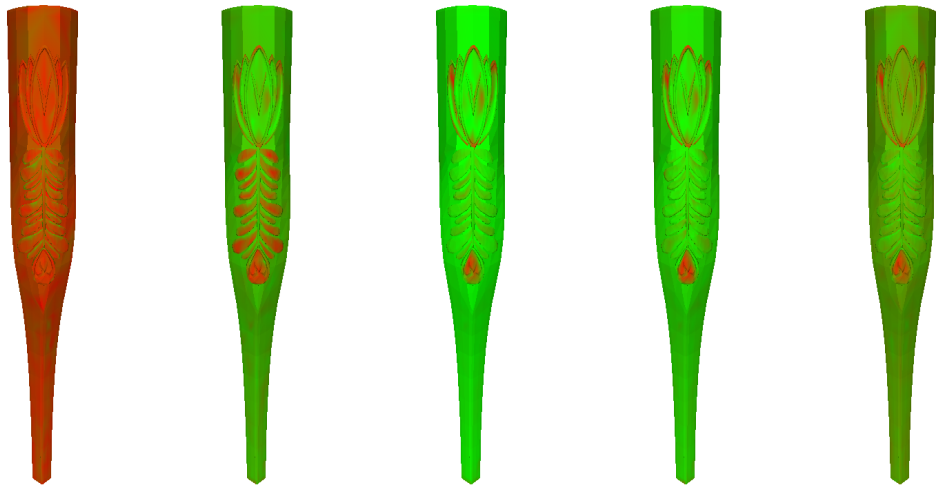
Figure 4.10: Original mesh data for leg cabriole_10. Saliency map of loss from: (a) Layer 1, (b) layer 2, (c) layer 3, (d) layer 4, (e) all layers (1 to 4)

local region. This leads to extracted features being highly dependent on the number of edges in a given region. Pooling does not appear to relax this dependency, as saliency maps generated from later layers in the model after pooling has occurred still struggle with highly localised highlighted regions.

4.4 Style clustering

Similar to methods by [58], t-SNE visualisations have been made from feature activations, shown in figures 4.12, 4.13, and 4.14. Distinctive structure emerges in these plots which are then clustered using k-means clustering. These clusters are then visualised on the mesh data. As is found in results by [58], these clusters appear to segment semantically different styles. In figure 4.12, the curved and ball-shaped sections are segmented from the straight sections of the mesh. In 4.13 and 4.14, the distinctive features from the Cabriole style are all segmented into the blue regions with only a slight erroneous overlap near the segmentation boundaries.

Attempts were made at clustering activations in layers after pooling had occurred, however much



(a) (b) (c) (d) (e)

Figure 4.11: Remeshed data for leg cabriole_10. Saliency map of loss from (left to right): Layer 1, layer 2, layer 3, layer 4, all layers (1 to 4)

of the semantic meaning of the shapes is lost at that stage. The style details which we aim to capture with clustering are mostly no longer visible after pooling. Style information may still be present in the representations of the pooled edges, however the pooled mesh used to visualise these features would no longer show the intended styles. Embedding clusters, as well as mesh colourings, can be seen in figures 4.15 and 4.16, although clusters in pooled layers quickly lose semantic meaning. Due to there not being a one-to-one correspondence between edges of the unpooled and pooled mesh, attempts were not made to visualise clusters from features of later layers on the original mesh.

4.5 Summary

Analysis was first performed of the high-dimensional gram matrices representing style. Dimensionality reduction techniques showed that various styles appeared somewhat in clusters, suggesting that gram representations of style could adequately describe the subtle differences between different style classes. Attempts at classifying these styles directly lead to inferior performance to current state-of-the-art results, presumably due to the very high dimensionality of the generated

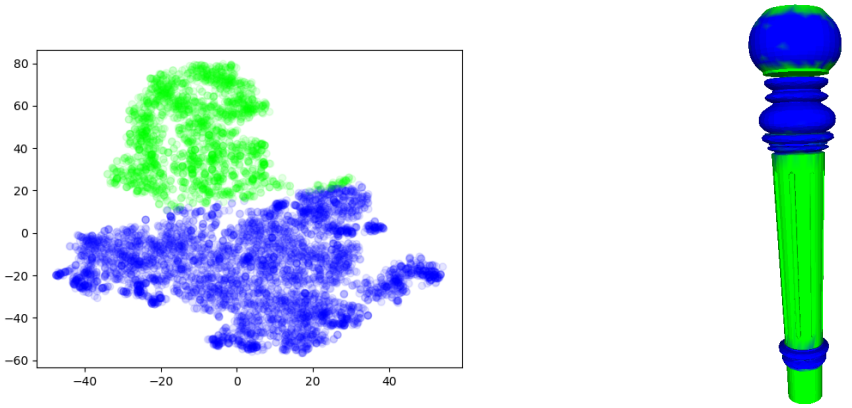


Figure 4.12: Straight 1 from the Leg class. 2 feature clusters visualised.

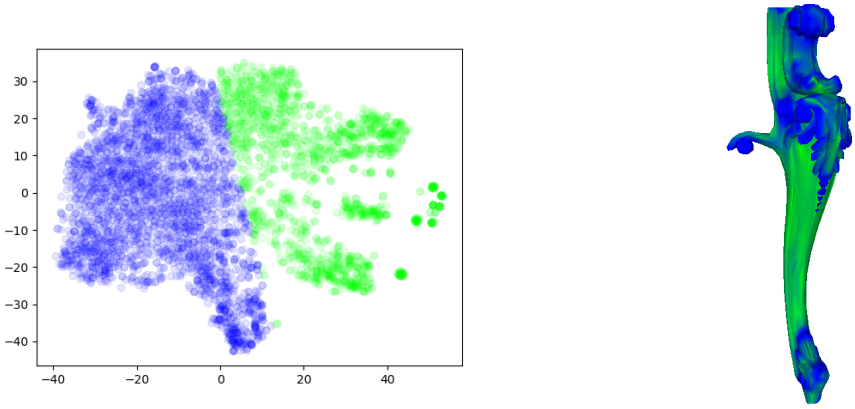


Figure 4.13: Cabriole 1 from the Leg class. 2 feature clusters visualised.

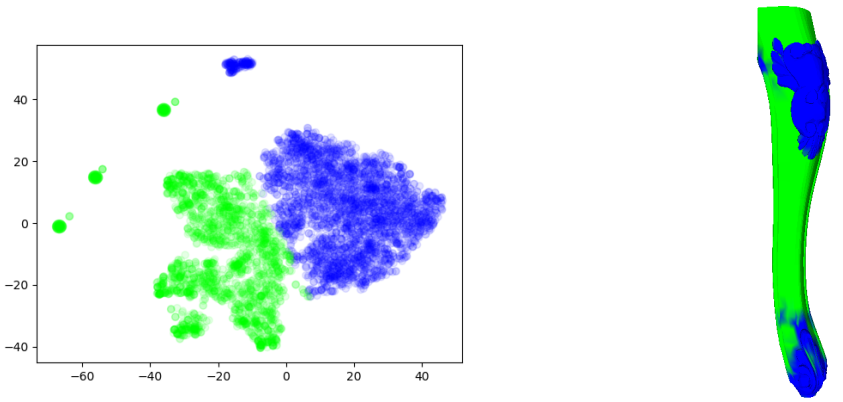


Figure 4.14: Cabriole 5 from the Leg class. 2 feature clusters visualised.

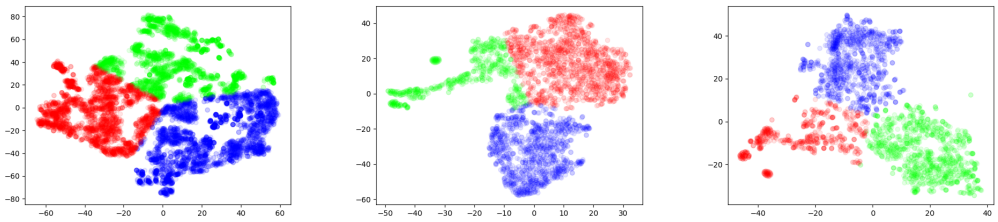


Figure 4.15: Clustered feature embeddings for layers 1, 3 and 4 of a Ming Chair.

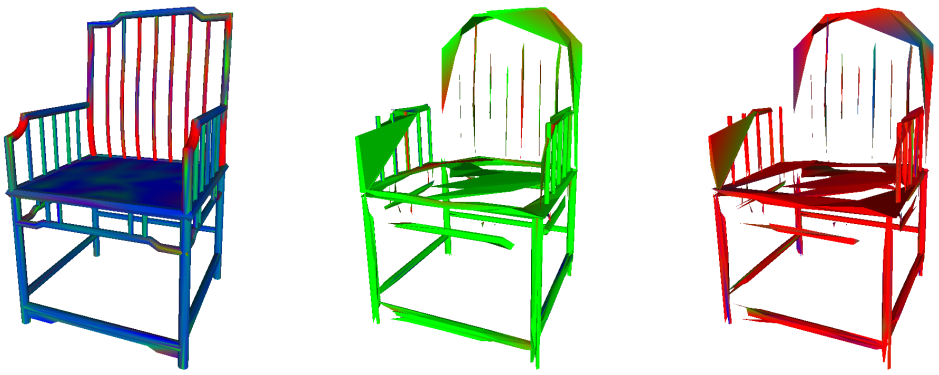


Figure 4.16: Visualised clusters on pooled meshes for layers 1, 3 and 4 of a Ming Chair.

style representations leading quickly to over-fitting on the relatively small dataset.

However the benefits of the presented methods become more apparent when observing the quality and flexibility of generated saliency maps to represent stylistic similarity and differences between shapes. Distinctive styles were clearly identified by saliency maps generated at different layers of model activations, with particular success at layer 2 activations. Average representations were generated for each style in the dataset, which were then used to create saliency maps for various sample shapes. Similarities and differences between the saliency maps generated by each style suggested which styles had more overlapping traits, such as Japanese and Ming.

The effects of pre-processing mesh data on the quality of generated style representations were then analysed. The unprocessed data was found to only highlight very isolated edges which exhibited the most extreme traits of the different style, whereas pre-processed data successfully highlighted entire regions of differing styles. This is thought to be due to fewer edges allowing each edge to capture more traits of the differing styles.

Finally, results from image-based style analysis were replicated on our representations for 3D data. This suggested that generated features are of comparable quality, as well as confirming that certain phenomena found in image-based styles can be generalised to 3D shape styles. Specifically, clustering performed on feature activations reveal distinct styles within a single shape, allowing for segmentation of these styles.

Chapter 5

Discussion and Conclusions

5.1 Discussion

Gram-based style classification models performed well below previous methods by [19, 57]. It is believed that these methods perform so poorly due to having orders of magnitude more model parameters than training examples. Previous methods focus on generating a low number of high-quality features, an approach much better suited for small datasets. After concatenation of multiple layers of gram matrices, model inputs have between 10^3 and 10^5 parameters, which is believed to simply be too many for a model to extract a meaningful signal from. Attempts to reduce the effective degrees of freedom of the model through regularisation, dimensionality reduction, and selective use of layer activation did not resolve these issues.

Generated saliency maps provide useful visual insight into the regions which have a similar style to reference meshes. However many different settings need to be altered to view all results. For example, the threshold used to convert model gradients into scalar values may be varied to provide sensitivity to different ranges of values. The layers of activations used to generate saliency maps are another parameter to chose. If used in a user-facing application, these options would need to be easily accessible to make full use of the many views of information generated.

Results from feature clustering show a promising method of segmenting different styles within shapes. This feature could have value outside of style analysis. It is though that users of 3D modelling software might make use of such a tool which highlights certain regions of shapes based their properties. For example, to smooth out regions which appear to contain a Cabriole style,

an overly complex texture which a 3D printer or game engine would find particularly difficult to reproduce, or simply bulk modifying repeated regions of a shape which are hard to describe programmatically. This could be accomplished in a semi-supervised manner, where a few sample points are identified by a user, which are then used by a clustering algorithm such as mixture of gaussians to locate all regions of similar higher-order features which cannot simply be located using hand-crafted features such as curvature.

5.2 Critical Analysis

Currently, gram matrices produced from many of the layers produce irrelevant information. This is easy to identify when the expected regions of style dissimilarity are known in advance, allowing us to divert attention to the layers which produced the clearest output. However, when style elements are more subtle, the task of discarding less useful layer activations may be less trivial, especially when different layers produce the desired output depending on the complexity of the style. A systematic way of evaluating the quality of saliency maps is not established, as well as testing the methods on more subtle styles. The potential pitfalls of presented methods are therefore not fully understood.

Segmentation through clustering shows a lot of promise, however the shapes analysed in the dataset all have distinctive changes in style. These simple examples may not generalise as well to more complex shapes. Also, clustered regions are often not clean, producing noisy results along boundaries. This could be improved using graph-based post-processing such as graph cuts.

Analysis of the effects of pre-processing mesh data could have been conducted more thoroughly. Pre-training a model on the used dataset could have enabled activations to become less dependent on the density of edges within each mesh. Results show that lack of pre-processing can certainly lead to problems, however insufficient attempts at remedying this problem were attempted.

The pre-processing decisions made were determined by intuition as to which procedures would generate meshes believed to work well with meshCNN. Instead, this process could have been guided more by actual results produced by the model.

5.3 Extensions and Future Work

The use of different underlying models could be compared to meshCNN. The most directly related model being MeshNet [9], where no pooling step is used, however face normals are used as a base feature of the model which could aid in disentangling similar styles which are dependent on the direction of faces. A recent model by [18] provides similar features, and could also be investigated.

Future work could also focus on reducing the dependence of results on specific remeshing techniques. This could either be through training on larger datasets, incorporating scale-independent components into the model, potentially through an attention mechanism, or using a different model type altogether such as implicit neural representations.

Metric learning could also be used to train meshCNN, where positive and negative examples could be synthesised through data augmentations. This would be similar to work by [26], although with the aim of extracting high quality features for style classification instead of content classification.

A form of neural style transfer could also be attempted based on the generated gram matrices. Results already exist from [18], however they require training a generative adversarial network to function. Using the simplicity of gram matrices similarly to the original neural style transfer paper [11] could offer better generalisation capabilities. It is hypothesised that models which focus solely on surfaces of shapes may not adequately represent content for style transfer to take place. [17] uses a chamfer distance loss to retain content through limiting surface distance from an original shape, which in combination with optimising gram matrix style representations may lead to successful style transfer.

5.4 Conclusions

This research aimed to develop methods for identifying regions across 3D shapes where style differs from reference shapes of a given style. The introduction framed the problem, identifying challenges associated with 3D shapes and generation of feature representations. A background review of common representations for 3D data is given where implicit neural representations and triangular meshes were identified as the most promising for retaining style information. Fundamental aspects of image style research, which separate data into style and content, are translated to three-dimensional

shapes. Style is redefined from a texture-based trait to one of surface shape and topology, and content is simply defined as the object portrayed by the shape. Various models which process either implicit neural representation or mesh formats are studied for their abilities to capture the defined notion of style. MeshCNN was chosen due to its focus on edges, which are thought to allow a strong understanding of surface topology, as well as a pooling operator which could reduce the dependence on specific triangulations of the data. Modifications are made to allow for a more diverse range of input data, as well as to facilitate analysis.

Gram matrices are generated for style representations at each of the four layers of activations of the model, with empirical evidence suggesting that activations from the second layers are most informative of style. Direct comparisons are made of style representations from pairs of meshes, with a loss defined as the mean squared difference between style representations. Backpropagating this loss generates gradients per edge, which can be transformed into detailed saliency maps of style similarity. Saliency maps are shown experimentally to highlight regions expected to contain different styles, successfully solving the initial problem posed in this report. Further extensions included averaging style representations of entire classes of styles, to analyse the similarity of a given mesh to a style defined by a set of reference shapes. Feature activations were also analysed directly, presenting an effective tool for segmenting various styles present within a shape.

Bibliography

[1] Ahmed Alqaraawi, Martin Schuessler, Philipp Weiß, Enrico Costanza, and Nadia Berthouze. Evaluating saliency map explanations for convolutional neural networks. In *International Conference on Intelligent User Interfaces, Proceedings IUI*, pages 275–285, 2020.

[2] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[3] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Levy. *Polygon Mesh Processing*. 2010.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. dec 2015.

[5] Wei Ta Chu and Yi Ling Wu. Deep correlation features for image style classification. In *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, pages 402–406, 2016.

[6] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3D-encoder-predictor CNNs and shape synthesis. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 6545–6554, 2017.

[7] Danielle Ezuz, Justin Solomon, Vladimir G Kim, and Mirela Ben-Chen. GWCNN: A Metric Alignment Layer for Deep Shape Analysis. *Computer Graphics Forum*, 36(5):49–57, 2017.

[8] WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[9] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. MeshNet: Mesh Neural Network for 3D Shape Representation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:8279–8286, 2019.

[10] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[11] Leon Gatys, Alexander Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *Journal of Vision*, 16(12):326, aug 2016.

[12] Kan Guo, Dongqing Zou, and Xiaowu Chen. 3D mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics*, 35(1), 2015.

[13] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep Learning for 3D Point Clouds: A Survey. 2020.

[14] Kunal Gupta and Manmohan Chandraker. Neural Mesh Flow: 3D Manifold Mesh Generation via Diffeomorphic Flows. Technical report.

[15] Rana Hanocka. Meshcnn wiki - data processing, Sep 2020.

[16] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN: A network with an edge. *ACM Transactions on Graphics*, 38(4), sep 2019.

[17] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2Mesh: A Self-Prior for Deformable Meshes. may 2020.

[18] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. Deep Geometric Texture Synthesis. *ACM Trans. Graph. 39, 4, Article*, 108:11, jun 2020.

[19] Ruizhen Hu, Wenchao Li, Oliver Van Kaick, Hui Huang, Melinos Averkiou, Daniel Cohen-Or, and Hao Zhang. Co-locating style-defining elements on 3D shapes. *ACM Transactions on Graphics*, 36(3), 2017.

[20] Jingwei Huang, Yichao Zhou, and Leonidas Guibas. ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups. may 2020.

[21] Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 1510–1519. Institute of Electrical and Electronics Engineers Inc., dec 2017.

[22] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research. nov 2019.

[23] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural Style Transfer: A Review. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, may 2017.

[24] Sergey Karayev, Matthew Trentacoste, Helen Han, Aseem Agarwala, Trevor Darrell, Aaron Hertzmann, and Holger Winnemoeller. Recognizing image style. In *BMVC 2014 - Proceedings of the British Machine Vision Conference 2014*, 2014.

[25] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D Mesh Renderer. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3907–3916. IEEE Computer Society, dec 2018.

[26] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. apr 2020.

[27] Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. Transport-based neural style transfer for smoke simulations. *ACM Transactions on Graphics*, 38(6), may 2019.

[28] Byungsoo Kim, Vinicius C Azevedo, Markus Gross, and Barbara Solenthaler. Lagrangian Neural Style Transfer for Fluids. *ACM Trans. Graph*, 39(4):10, 2020.

[29] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming Hsuan Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 386–396. Neural information processing systems foundation, may 2017.

[30] Isaak Lim, Anne Gehre, and Leif Kobbelt. Identifying style of 3D shapes using deep metric learning. *Eurographics Symposium on Geometry Processing*, 35(5):207–215, 2016.

[31] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*, pages 163–169, New York, New York, USA, aug 1987. Association for Computing Machinery, Inc.

[32] Zhaoliang Lun, Evangelos Kalogerakis, and Alla Sheffer. Elements of style: Learning perceptual shape style similarity. In *ACM Transactions on Graphics*, volume 34, 2015.

[33] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. feb 2018.

[34] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. 2020.

[35] Travis Oliphant and Jarrod k. Millma. A guide to NumPy, 2006.

[36] Blender Online Community (Organization/Institution). Blender - a 3D modelling and rendering package, 2020.

[37] Utku Ozbulak. Pytorch cnn visualizations. `https://github.com/utkuozbulak/pytorch-cnn-visualizations`, 2019.

[38] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, pages 165–174, jan 2019.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, dec 2019.

[40] Les Piegl. On NURBS: A Survey. *IEEE Computer Graphics and Applications*, 1991.

[41] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[42] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning deep 3D representations at high resolutions. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 6620–6629, 2017.

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9351, pages 234–241. Springer Verlag, may 2015.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015.

[45] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, 128(2):336–359, 2020.

[46] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. Technical report.

[47] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions. jun 2020.

[48] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. jun 2020.

[49] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 2107–2115. Institute of Electrical and Electronics Engineers Inc., dec 2017.

[50] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6313 LNCS, pages 356–369, 2010.

[51] W. T. TUTTE. *Connectivity in Graphs*. University of Toronto Press, 1966.

[52] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2625, 2008.

[53] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. sep 2019.

[54] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1912–1920, 2015.

[55] Yun Peng Xiao, Yu Kun Lai, Fang Lue Zhang, Chunpeng Li, and Lin Gao. A survey on deep geometry learning: From a representation perspective, feb 2020.

[56] Kai Xu, Honghua Li, Hao Zhang, Kai Xu, Honghua Li, Yueshan Xiong, Zhi Quan Cheng, and Daniel Cohen-Or. Style-Content Separation by Anisotropic Part Scales. *ACM Transactions on Graphics*, 29(6):1–10, 2010.

[57] Fenggen Yu, Yan Zhang, Kai Xu, Ali Mahdavi-Amiri, and Hao Zhang. Semi-supervised co-analysis of 3D shape styles from projected lines. *ACM Transactions on Graphics*, 37(2), apr 2018.

[58] Yulun Zhang, Chen Fang, Yilin Wang, Zhaowen Wang, Zhe Lin, Yun Fu, and Jimei Yang. Multimodal style transfer via graph cuts. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, pages 5942–5950, 2019.