Raw Material Selection for Object Construction

Jason Perlow
607348

A dissertation, supervised by
Dr. Benjamin Rosman, Dr. Bradley Hayes and Pravesh Ranchod,

submitted to the
Faculty of Science, University of the Witwatersrand,
in fulfilment of the requirements for the degree of Master of Science.

July, 2018

Declaration

I, Jason Perlow, hereby declare the contents of this research dissertation to be my own work unless otherwise explicitly stated. This dissertation is submitted for the degree of Master of Science (Dissertation) at the University of the Witwatersrand, Johannesburg. This work has not been submitted to any other university, nor for any other degree.

Signature

Date

Declaration

Similar work has appeared in a paper titled *Raw Material Selection for Object Construction* presented at the 2017 PRASA-RobMech conference [Perlow et al., 2017].

Abstract

Agents, such as assembly robots, are typically incapable of building objects without a predefined goal or predefined set of materials. Extending the construction capabilities of agents to objects and materials that an agent has not seen before would therefore greatly improve the scope of objects that agents can construct. An important step in building novel objects is the ability to recognise combinations of raw materials which are likely to be useful. As a step toward automating this step, we aim to exploit the intuition that the visual characteristics of candidate raw materials provide useful cues to their potential combinations. Toward this end, we present a Siamese neural network based model that is able to recognise unseen raw materials present in objects given a list of candidate material images.

We demonstrate the utility and efficacy of our model within two domains. The first is a single material selection domain that uses the ShapeNet 3D model dataset where we attempt to recover the materials present in a model. The second is a multiple material domain using the adventure game Minecraft where we predict the combinations of materials that will result in a target tool. We empirically demonstrate that by recognising the visual similarities between objects and materials our model is able to learn from a subset of object material pairs and generalise to unseen objects, materials and texture packs. We perform such tests by showing that our model outperforms chance and baseline methods.

Acknowledgements

I would like to extended my gratitude to my supervisors Benji, Brad and Pravesh for their countless discussions with me regarding this work and related ideas. Without their consistent advice and encouragement throughout the course of my degree this work would not have been possible.

I thank my anonymous conference reviewers for their commentary, criticism and recommendations that have greatly refined the framing for the paper upon which this work is based.

I would also like to thank my fellow students at Wits for the opportunity of having a test audience to bounce ideas off as well as providing guidance to fulfil the requirements of the dissertation submission process.

Contents

List of Figures

Chapter 1

Introduction

When an agent, such as an assembly robot, is required to build objects it typically can only make use of a pre-programmed set of materials to construct a pre-programmed set of objects. Humans, however, when presented with an object and a set of possible materials, can intuitively estimate which materials are more likely or less likely to exist within an object even if they have not seen the object or its constituent materials before. As a long term goal we aim to close this gap by creating an agent capable of building objects that contain materials that an agent has not seen before. Such an ability is desirable since the scope of possible objects that an agent could construct would not be constrained to what a programmer has explicitly instructed it to do.

As a small step in this direction, we focus our attention on the problem of selecting raw materials from a set of candidate materials, which are necessary to construct potentially novel objects. We are especially concerned with the case where materials are unseen which, due to the many possible materials within novel objects, would typically be the case. To perform recognition of materials that an agent had not seen before the agent would need to perform material recognition that does not consist of predefined classes. Instead it might need to transfer knowledge from familiar raw materials and object classes to novel raw material and object classes.

Toward this end, we propose a model that performs recognition by making use of the general visual relationship between a given material and an object it needs to construct. The model learns the general visual relation by being given a subset of objects and materials which allows the model to compare unseen objects and materials without additional training.

For example, consider the case shown in Figure 1.1 where previous observations consist of recognising that the iron ingot on the top left can be used to create the iron shears on the top right. By analogy, the agent can then predict that the materials on the bottom left are necessary to create the iron pickaxe on the bottom right. In such a case, the agent has not seen the target tool nor one of the materials. By learning to recognise the similarities between the first tool and its material, it can use this similarity knowledge to select appropriate materials for the unseen tools and materials.

1.1 Visual Similarity

We propose a model inspired by the recognition of visual similarities between required materials and a desired object that an agent needs to construct. By solely using features based on their visual appearance, this allows an agent to construct unseen objects more quickly than trying all possible materials available. We assume that an agent would receive visual information in the form of images. Our method therefore

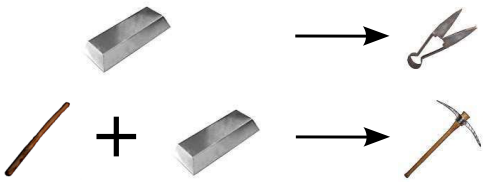

Figure 1.1: (Top) Source domain - Construction of iron shears. (Bottom) Target domain - Constructing an iron pickaxe from raw materials.

allows an agent to recognise required unseen raw material images and link them to corresponding novel object images. For example in Figure 1.1, based on the visual similarities between the material and tool in the top row, our agent should infer visual similarities between the materials and tool in the bottom row.

1.2 Domains and Contributions

We demonstrate the efficacy of using visual similarities to perform material selection in two domains. The first domain, presented in Chapter 5, is that of material selection performed on 3D furniture models from the ShapeNet dataset [Chang et al., 2015]. The ShapeNet domain aims to show that our method could possibly, in the future, form part of a real-world model for an agent capable of object construction.

As shown in Figure 1.2, in the ShapeNet domain we aim to determine the presence of raw materials in a given 3D object model by matching a list of 3D model textures with rendered 3D models that contain exactly one of those textures. This domain therefore assumes that textures correspond to raw materials such as wood and fabric (shown in Figure 1.2). Here we test the ability of our model to generalise to unseen materials and similar objects once trained on a constrained subset of material object pairs.

Figure 1.2: The ShapeNet material selection domain. Possible materials for building a bench.

The second domain, presented in Chapter 6, is that of tool construction in the sandbox adventure game and reinforcement learning domain Minecraft [Aluru et al., 2015]. In Minecraft, raw materials can be mined from the environment and combined or crafted into food, tools and other objects that can be used in the game. Building on the intuition of the ShapeNet domain, we exploit the same visual similarities between objects and their constituent materials in the Minecraft domain. This domain aims to tentatively show that our model can select multiple materials, also known as a recipe, to construct objects such as shown in Figure 1.3. In particular, we test the ability of our model to find the combination of visual features from materials that best matches the visual features from a tool that we wish to construct.

As an example of determining the recipe of the tool by finding the most plausible raw material combination, consider possible recipes in Figure 1.3. Even though both the tool and material are unseen, from inspection of Figure 1.3, a human without any Minecraft experience or even knowledge of the object names can see that the third combination is most likely. It is precisely this intuition derived from visual

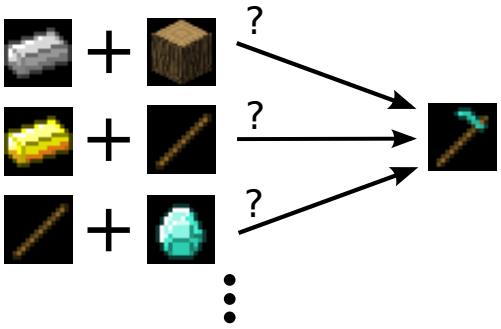

Figure 1.3: Possible inventory item combinations for crafting a diamond hoe. Note the visual similarity between the third row that is a stick and a diamond (the correct pair) and the tool. This intuition inspires our model.

features (such as shape and colour) that we successfully model in this work.

To further motivate the adaptivity of our method to visual variation we also consider generalisation to tool images of unseen *visual style*. Since the chosen domain of Minecraft is restricted in the visual variety of available items, we mitigate the limitations of the default dataset, the Vanilla texture pack, by selecting a variety of additional texture packs. For example, we use the LB Photorealism texture pack (see Figures 1.1 and 6.1), which has images with a more realistic appearance compared to the Vanilla texture pack.

Our primary contribution in this work is a Siamese neural network-based model that predicts a construction/crafting success score given component raw material images. In the ShapeNet domain, the material image is $\boldsymbol{M}$ from candidate materials $\mathcal{M}$. More formally, for a particular target or goal object image $\boldsymbol{G} \in \mathcal{G}$, the model assigns a score to a series of mappings, which when ranked, approximate the construction procedure that maps a candidate material to a goal item:

$$\mathcal{M} \rightarrow \mathcal{G} \tag{1.2.1}$$

To find the correct crafting policy π given G, we invert the construction model to yield the policy

$$\pi : \mathcal{G} \rightarrow \mathcal{M}. \tag{1.2.2}$$

However, in the Minecraft domain we wish to find a single pair of materials and thus consider $\mathcal{M}^2$ instead of $\mathcal{M}$ in Equations (1.2.1) and (1.2.2).

We show that our model successfully extracts visual features from the appropriate raw materials that best match the visual features of the object to be constructed, allowing for more rapid object production than competing methods.

To the best of our knowledge, this is the first such model using visual similarity transfer to accelerate vision-based object construction from component parts, and in doing so we provide a benchmark for future work to compare against within Minecraft and ShapeNet domains.

1.3 Outline

The remainder of this document is structured as follows. In Chapter 2 we present the preliminary technical theory necessary to understand the methods used in this document. In particular, we present the theory of manifold learning and neural networks. In Chapter 3 we present a survey of prior work and

compare it with the presented research. In Chapter 4 we discuss methods used in the presented research, namely visual similarity. In Chapter 5 we present the technical details of our single material selection domain that is ShapeNet by describing the dataset and model results. In Chapter 6 we present the technical details of our multiple material selection domain that is Minecraft by describing the dataset and model results. Finally, in Chapter 7 we summarise the document and provide concluding remarks and possible directions for future work.

Chapter 2

Background

In this chapter we provide a brief summary of the necessary theoretical background for the methods presented in Chapter 4 and as well as results presented in Chapters 5 and 6. Since our method uses neural networks, we motivate their use in Section 2.1 by the use of manifold learning in the extraction of visual features from the visual data in the studied domains. We further justify our method by its success in related modern computer vision work that makes use of neural networks as discussed in Chapter 3.

We discuss the definition, terminology, typical architecture of neural networks in Section 2.2 and training of such models in Section 2.3. In these sections we mainly focus on methods used in this work while briefly mentioning popular related algorithms. We provide a brief overview of the theory of the components used in our model, namely a simple neural network the multilayer perceptron (MLP) and related algorithm the convolutional neural network (CNN) in Sections 2.2.1 and 2.2.2 respectively.

Another architectural influence for our model, discussed in Section 2.2.3, are Siamese neural networks. They are a class of neural network that includes multiple identical subnetworks in its architecture, particularly well suited for finding similarity. This is followed by Section 2.3 where we discuss gradient descent, a means of training neural networks.

2.1 Manifold Learning

The application of machine learning to computer vision problems, such as in the presented work, typically involve the use of parametrised functions to approximate an operation on visual data. However, images typically consist of thousands to millions of pixels which each correspond to variables or dimensions. This presents a challenge for classical general function approximations such as fitting a multivariate polynomial to data. In particular, the number of parameters in such functions grow exponentially with the dimensionality of the data they aim to approximate [Bishop, 2006]. In any practical dataset, the exponential number of parameters will greatly overwhelm the number of dataset samples. The number of dataset samples will therefore be insufficient for determining accurate parameters. This problem is known as *the curse of dimensionality*.

Fortunately, a practical image dataset of fixed size images occupies a structured subspace of all conceivable fixed size images. Examples of structure include small distortions of images such as rotation, translation and shearing that do not significantly change the meaning of the data. The discrepancy between the relatively small number of ways in which a practical image can vary without changing its meaning and the number of dimensions in an image leaves the possibility for dimensionality reduction to find latent variables, also known as an embedding [Hinton et al., 1997].

In an example of extracting latent variables, called manifold learning, data points are assumed to be distributed around a smooth subspace with far fewer dimensions than the original dataset. In the case of computer vision the manifold is embedded in the space of all theoretically possible images. This latent variable, in part, corresponds to small distortions that result in images that occupy only a small subspace of all conceivable images. The purpose of finding the latent variable is that we can then flatten out or unfold it into a low-dimensional surface. This process projects the manifold onto a low-dimensional Euclidean space, thus reducing dimensionality of data as to avoid the curse of dimensionality.

If we want to model a linear latent variable we can use principal component analysis (PCA) for dimensionality reduction where the manifold is a hyperplane with Gaussian noise embedded in higher dimensional Euclidean space. However, if our data is distributed around a non-linear surface, as is the case in computer vision, we may need a more complex approach such as a neural network. Neural networks consist of non-linear components or layers that can extract low-dimensional latent variables from a non-linear high-dimensional space. Neural networks are therefore a solution to the curse of dimensionality in computer vision [Bishop, 2006]. In particular, before the final layer, neural networks typically reduce a high-dimensional vector of features to an embedding. By the definition of manifolds, this embedding is low-dimensional and smooth or Euclidean, allowing traditional statistical methods such as linear regression and correlation to perform decision making.

In this work we aim to find a joint embedding of materials and objects that best allows us to match them together. Hopefully, the materials and objects approximately lie on a manifold so we can avoid the curse of dimensionality that would otherwise result from the high dimensional image data from the problem at hand. Since neural networks can adapt to non-linear manifolds they are applicable to image processing problems that tend to exhibit such manifolds. For example, changes in the grain of a wood texture would result in a non-linear manifold due to pixels being switched with each other. In such a situation, linear techniques such as PCA would not accurately extract the best features and would therefore be poorly suited for the problem at hand.

Our use of non-linear manifold learning therefore provides a motivation for the class of machine learning algorithms we make use of in the presented work, namely neural networks. In particular, our use of convolutional neural networks (CNNs), which is adapted to deal with image processing problems, is applicable to the computer vision problem in the presented work. Manifold learning is also amenable to our approach since we attempt to correlate image embeddings from a joint embedding. This is because embeddings are Euclidean are therefore amenable to standard linear algebra methods such as correlation, which we use in our work.

2.2 Neural Networks

2.2.1 Multilayer Perceptrons

An example of a simple neural network is the multilayer perceptron (MLP). Each layer is a transformation between the input vector $\boldsymbol{x}$ and output vector $\boldsymbol{y}$ as

$$y_i = \sigma(b_i + \sum_j W_{i,j} x_j), \tag{2.2.1}$$

where σ is a non-linear function. y_i, b_i, $W_{i,j}$ and x_j are scalars which correspond to output values, biases, weights and input values respectively. The variables i and j index the values associated with the output elements and input elements respectively. Each layer is called a *densely connected* layer, *dense*

layer or a *perceptron* [Rosenblatt, 1958]. A dense layer is shown in Figure 2.1 where the dot product is taken between features and weight vectors to produce output features.

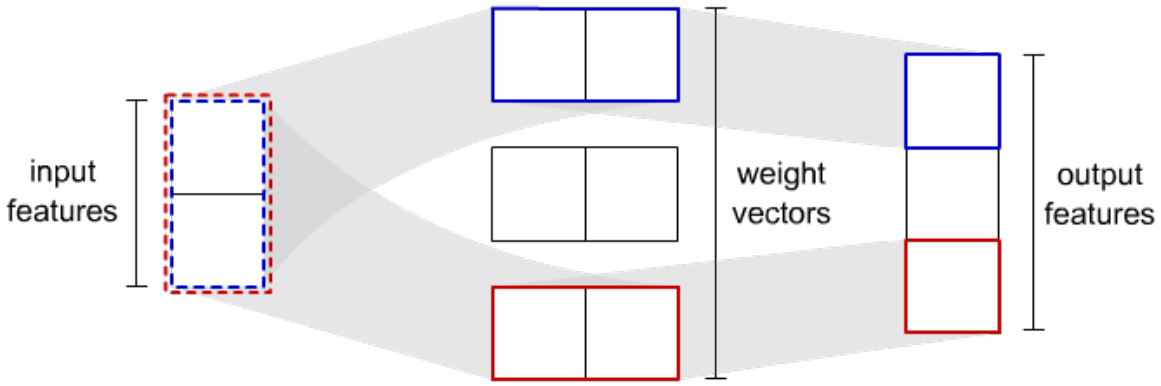

Figure 2.1: Simplified dense layer example without biases. This example has two input values, three weight vectors and three output features. Example computations of output values are given in blue and red.

In Equation 2.2.1 the bias and summation terms together form a matrix multiplication with a constant added to it, $\mathbf{b} + \boldsymbol{A}\mathbf{x}$, also known as an affine transformation. The addition of the non-linearity σ allows for the creation of non-linear functions compared to a layer without σ, which would just be an affine function. Typical non-linearities include $tanh(\cdot)$ and more recently the rectified linear unit $ReLU(\cdot)$, defined as

$$ReLU(x_i) = \begin{cases} 0 & x_i \leq 0 \\ x_i & x_i > 0 \end{cases}. \tag{2.2.2}$$

An MLP is structured such that when layers are composed features or embeddings from each layer, the outputs of each later, can be fed into the next layer to yield a chain of feature extractors. The $ReLU(\cdot)$ non-linearity has the advantage over $tanh(\cdot)$ of being resistant to the *vanishing gradient problem* where, during training, later layers coordinate poorly with earlier layers. This is because $tanh(\cdot)$ and similar non-linearities exhibit the phenomenon of *saturation* where they threshold inputs with high or low values [Goodfellow et al., 2016]. This is contrast to $ReLU(\cdot)$ which replicates any input above zero while causing lower information loss as data is passed through layers. $ReLU(\cdot)$ also has the advantage of requiring very low computation by being, in practice, no more than a conditional branching statement.

The type and way in which layers are connected is called the *architecture* of the network. *Hyperparameters* on the other hand refer to parameters that govern the behaviour of architecture components, such as the number of weights in each layer as well as parameters for training the model. The hyperparameters are found by trying minor variations of standard configurations on training data and choosing the one that performs best.

2.2.2 Convolutional Neural Networks

In this work we deal with a vision problem. We therefore extract visual features by using CNNs, a variant of a neural network particularly adapted to work with images [Fukushima and Miyake, 1982]. CNNs have recently become popular in many computer vision tasks by vastly outperforming traditional techniques such as SIFT and other techniques that rely on hard-coded low-level features [Krizhevsky et al., 2012; LeCun et al., 2015]. Traditional techniques are typically followed by a general classifier

such as an SVM to learn high-level features. CNNs have an advantage over such approaches by being able to learn both low-level and high-level features, thus being adaptable to specific computer vision problems.

Typical CNNs have multiple convolutional layers for feature extraction followed by densely connected layers for classification or regression [LeCun et al., 2010]. In a manner analogous to an MLP layer, the convolutional layers transform $\boldsymbol{X}$, an image that has multiple channels or 2D arrays, into another image $\boldsymbol{Y}$ as

$$Y_i = \sigma(b_i + \sum_j W_{i,j} * X_j), \tag{2.2.3}$$

where Y_i, W_{ij} and X_j are now two-dimensional arrays. b_i is a scalar bias and the variables i and j index the input and output features respectively. $*$ is the convolution operator, which in two dimensions is defined as

$$(W * X)_{i,j} = \sum_m \sum_n W_{m,n} X_{m+i,n+j}, \tag{2.2.4}$$

where i and j are associated with the pixel coordinates of an output image and m and n are associated with the pixel coordinates of arrays that get adjusted during training called *filters* or *kernels*.

As shown in Figure 2.2 a convolution layer maps input features or channels to output features or channels. The filters are convolved across an image to produce output pixels by performing pointwise multiplications and summing across corresponding patches of input images. Each filter has a *stride*, which determines how many pixels should be skipped in each dimension before the filter is applied. Output images from convolutional layers are learned features, which in the presented work are interpreted as visual features that best characterise the visual similarities between materials and objects.

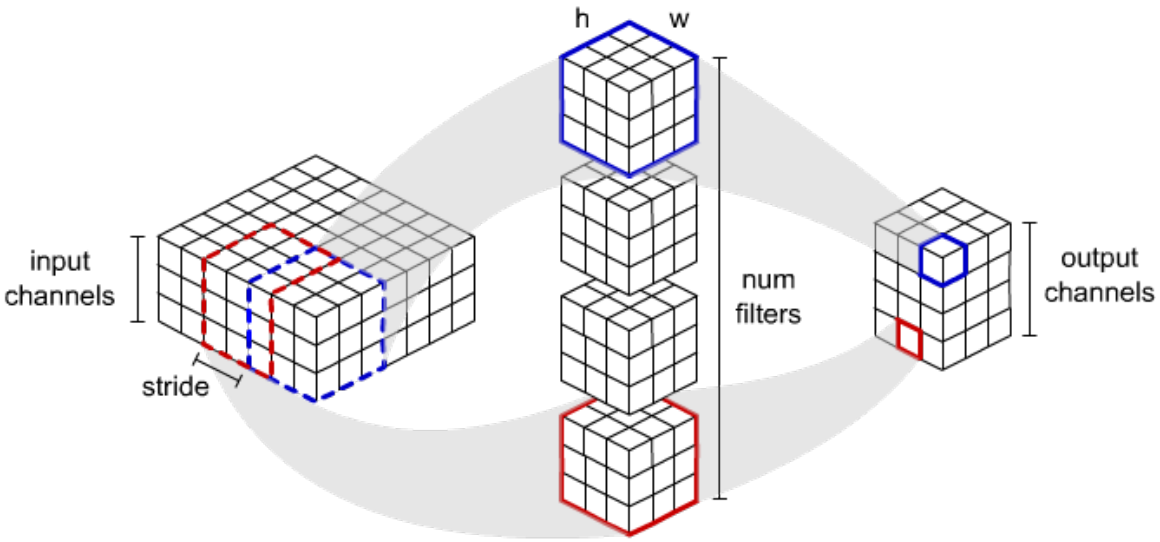

Figure 2.2: Simplified convolutional layer example without biases. This example has three 7×7 input channels, four filters, stride = 2, filter height $h = 3$, filter width $w = 3$ and four 3×3 output features. Example computations of output pixels are given in blue and red.

Conceptually, convolutions manipulate small regions of an image by recognising local patterns occurring in an image. Convolutional layers therefore assume that similar features such as edges or textures exist across an image [Zeiler and Fergus, 2014]. This is in contrast to MLPs which have independent weights for all incoming features and thus do not exploit the local statistical properties of images [Murphy, 2012].

In CNNs, the placement of non-linearities after convolutional layers has the effect of creating complex non-linear filter behaviour [Wiatowski and Bölcskei, 2018]. Non-linear CNN layers emphasise frequencies in the Fourier domain that are the distinctive features that characterise a given image [Saxe et al., 2011]. In modern CNNS, the $ReLU(\cdot)$ non-linearity is typically used. As with MLPs the $ReLU(\cdot)$ non-linearity remedies the vanishing gradient problem and is less computationally expensive than other non-linearities.

Pooling splits images into a grid of patches and reduces each of these patches to a single pixel. Patches are either reduced to a single pixel by taking the maximum pixel value or the average of all pixel values in the patch, yielding what are called *max pooling* and *average pooling* respectively. Pooling has the effect of downsampling which reduces the dimensionality of images. It also makes the network immune to small translation and scale perturbations of input images. Typically the pooling stride is equal to the dimensions of the pooling patch. This makes the image dimensions smaller by a factor equal to the dimensions of the pooling patch.

In a similar manner to MLPs, convolutional layers are composed to form a chain of feature extractors. The output of the final convolutional layer is flattened into a single vector that is fed into MLP layers, yielding embeddings that are amenable to standard linear algebra and statistical operations. This is desirable since the properties of standard linear algebra and statistical operations are well understood, thereby reducing a challenging computer vision problem into something more familiar.

2.2.3 Siamese Networks

In the field of supervised machine learning, labels are assigned to data points before training. However, in some cases, labels may be sparse or non-existent. In a field of machine learning called *metric* or *similarity learning* data points are matched together, meaning that data points, in effect, act as labels. In particular, by just being given an example, the model can make decisions on other data points even if it has not seen the other data before. An example of similarity learning with a single example is called *one-shot learning* and is discussed further in Chapter 3. An example of a machine learning model that can perform one-shot learning is a Siamese network, which we use in this work.

Siamese networks have multiple neural network branches with shared weights that produce vectors that are fed into a similarity metric [Bromleyh et al., 1993]. A common similarity metric used in Siamese networks is the cosine similarity $cos(\cdot, \cdot)$, which is closely related to the angle between two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. In particular, it is expressed as a normalised dot product $\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|}$. Since $\mathbf{v}_1$ and $\mathbf{v}_2$ are feature vectors, the angle measures the similarity of features contained in the vectors. In the situation where all features in one vector are a positively scaled factor of the other vector they are perfectly co-directed, that is similar, yielding a cosine similarity of +1. This is in contrast to if they were a negatively scaled factor of each other, that is opposites of each other, in which case they would be negatively co-directed, yielding -1.

When the neural network branches are CNNs, the whole model can be stated as

$$cos(CNN_w(\boldsymbol{X_1}), CNN_w(\boldsymbol{X_2})), \tag{2.2.5}$$

where $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$ are images and w are weights which are shared between the two CNN branches. Since the outputs of the CNN branches are feature vectors and the cosine similarity measures similarity between vectors, the cosine similarity therefore measures the extent to which features are shared between input images. In particular, a Siamese network extracts the same features from two images and compares them to determine the extent to which they share visual features. This can be done since the branches act as feature extractors that allow standard statistical and linear algebra techniques to be used, which is

in this case the cosine similarity. An example of a Siamese network operating on images of a material and a target object is shown in Figure 2.3.

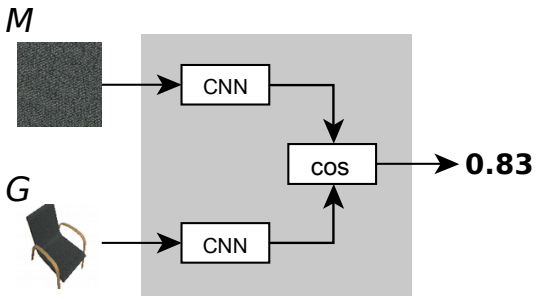

Figure 2.3: Siamese network.

2.3 Training

2.3.1 Loss Functions

To find a network that accurately embeds a given vector, weights need to be adjusted such that an error or $loss$ is minimised. Conceptually, a loss function computes the difference between the output y of the network and the ground truth $\bar{y}$. One simple example of a loss function is the Euclidean distance between the ground truth and the output, closely related to the Mean-Squared Error (MSE).

Since a Siamese network outputs a scalar similarity, training reduces to a regression problem. We update weights such that a loss function is minimised between the ground truth $\bar{y}$ and output y of the network. For Siamese networks a hinge loss given by

$$loss(y, \bar{y}) = \begin{cases} y, & \bar{y} = 1 \\ max(0, m - y), & \bar{y} = -1 \end{cases} \tag{2.3.1}$$

is typically used, where m is a threshold called the margin.

Unlike the MSE loss, the hinge loss function prevents dissimilar inputs from being pushed too far apart. This behaviour can be understood by using a physical analogy, where data points are connected by pairs of springs such that similar pairs are pushed together while dissimilar pairs are pushed apart when beyond a threshold that is the margin [Hadsell et al., 2006]. This behaviour is illustrated in Figures 2.4 and 2.5.

Building on the concept of a hinge loss is a related loss function called the cosine embedding loss. This loss is defined in the Torch machine learning library [Collobert et al., 2002] as

$$loss(\boldsymbol{y}_1, \boldsymbol{y}_2, \bar{y}) = \begin{cases} 1 - cos(\boldsymbol{y}_1, \boldsymbol{y}_2), & \bar{y} = 1 \\ max(0, cos(\boldsymbol{y}_1, \boldsymbol{y}_2) - m), & \bar{y} = -1. \end{cases} \tag{2.3.2}$$

This loss function is designed to keep the cosine function, which ranges between -1 and +1, a positive function. The cosine embedding loss is therefore a hinge loss applied to the cosine similarity function. Like the hinge loss in Equation 2.3.1 this loss prevents dissimilar inputs from being pushed apart, making it amenable to similarity problems, which we attempt to solve in the presented work.

Each training iteration makes use of a subset of training data called a batch. For each batch the loss function is summed across its samples, which could be the whole training set (full-batch learning)

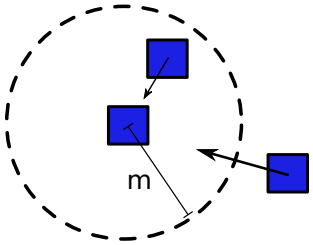

Figure 2.4: Attraction (indicated by arrows) between pairs of the same class. In this case pairs are attracted together regardless if they are beyond the margin or not.

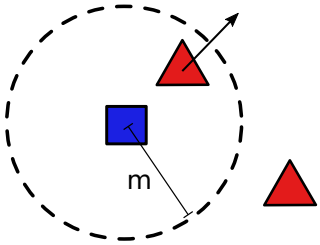

Figure 2.5: Repulsion (indicated by an arrow) between pairs of a different class. In this case the pairs are only repelled if they are within the margin.

or a subset (mini-batch learning). Work in modern neural network literature tends to use the terms batches and mini-batches interchangeably [Hinton et al., 2016]. Similarly, we use the term batch when we technically mean mini-batch. Since batches average the loss across many samples they improve network generalisation compared to using single samples [LeCun et al., 2012] while also improving memory usage compared to using the whole dataset. Batch learning also makes training amenable to parallelisation since the contribution of each batch element to the loss is not dependent on other batch elements.

2.3.2 Optimisers

Once we have defined a loss function we need to minimise it. This is usually done by iteratively updating weights by choosing from a family of optimisers that make use of an idea called *gradient descent*. This process is physically analogous to a ball rolling down a hill, where the altitude of the ball corresponds to the loss of the model and the longitude and latitude correspond to the current weight values and the direction in which the ball rolls is the direction of steepest descent.

To minimise the loss, weights are adjusted in small steps such that the loss is maximally minimised within a local region of the loss surface. Mathematically, a common feature of gradient descent optimisers is that they use the gradient, that is the derivative, of a loss function with respect to the weights of a model and tend to increment weights in the direction of steepest descent.

The derivative of the loss function is computed using an algorithm called *backpropagation* which is derived from the chain rule of calculus. Considering that neural networks have many layers the term backpropagation refers to the fact that errors are propagated back from the output layer of the network to the input layer during computation of the gradient [Dreyfus, 1973; Rumelhart et al., 1988].

One of the simplest gradient descent optimisers is Stochastic Gradient Descent (SGD) which updates weights in proportion to the average gradient of a given batch as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{n} \sum_{i} \alpha \nabla_{w} loss(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i), \tag{2.3.3}$$

where α is the step size or learning rate, i is the batch index and n is the batch size. The gradient is first computed on each element of a batch and then averaged together to perform a weight update.

SGD can get stuck on local optima or saddle points and exhibit oscillation [Goodfellow et al., 2016].

To remedy these problems, momentum, stated as

$$\begin{aligned} \boldsymbol{v} &\leftarrow \beta \boldsymbol{v} - \tfrac{1}{n} \textstyle\sum_i \alpha \nabla_w loss(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i) \\ \boldsymbol{w} &\leftarrow \boldsymbol{w} + \boldsymbol{v} \end{aligned} \tag{2.3.4}$$

is used. To continue the analogy of a ball rolling down a hill, $\boldsymbol{v}$ refers to the velocity of the ball if it had a mass of unity. Momentum corresponds to the ball continuing to roll in a direction based on its recently encountered terrain. More precisely $\boldsymbol{v}$ is a weighted running average, scaled by β, of previous gradients that gives the model short-term memory of previously computed gradients. Momentum therefore makes the current weight update similar to previous updates.

A further problem is that weights do not need to change at the same rate determined by a shared learning rate. To solve this optimisers with adaptive learning rates have been introduced which have an independent learning rate for each weight [Goodfellow et al., 2016]. An early example is AdaGrad which is short for Adaptive Gradients [Duchi et al., 2011]. A modern example of an adaptive optimiser is Adam [Kingma and Ba, 2014]. Adam adjusts the learning rate according to an exponentially weighted average of the gradients and squared gradients. When added these become the first and second moments, that are the mean and variance respectively. The name Adam refers to these adaptive moments.

Pseudocode for Adam is shown in Algorithm 1, where t is the training iteration and $\boldsymbol{g}$ is the gradient. The moments $\boldsymbol{m}$ and $\boldsymbol{v}$ are initialised at zero and tend to be biased towards zero during training. Terms $\hat{m}$ and $\hat{v}$ are intended to correct these biases. Based on experiments that use Adam for standard image and natural language machine learning problems, Kingma and Ba [2014] recommend $\beta_1 = 0.9$, $\beta_2 = 0.999$ as moment decay rates and $\epsilon = 10^{-8}$ for numerical stability. We make use of Adam in our work since it is a widely available optimiser that achieves state of the art results. We mostly make use of default parameters while only tuning the learning rate parameter. Future work should try using other optimisers and other optimiser parameters to see how they compare to the results obtained in Chapters 5 and 6.

Algorithm 1 Adam pseudocode [Kingma and Ba, 2014]. Operations on vectors are element-wise.

Require: $\boldsymbol{w}, \alpha, \beta_1, \beta_2, \epsilon$
  $\boldsymbol{m} \leftarrow \boldsymbol{0}$
  $\boldsymbol{v} \leftarrow \boldsymbol{0}$
  $t \leftarrow 0$
  while not converged **do**
    $t \leftarrow t + 1$
    $\boldsymbol{g} \leftarrow \frac{1}{n} \sum_i \nabla_w loss(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i)$
    $\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1)\boldsymbol{g}$
    $\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1 - \beta_2)\boldsymbol{g}^2$
    $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{m}/(1 - \beta_1^t)$
    $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v}/(1 - \beta_2^t)$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}} + \epsilon)$
  end while

2.3.3 Weight Initialisation

Before optimisation can take place, we need to initialise weights. Preferably, we want weights to have different values so gradient descent won't update equally contributing weights to the same value. Weights are typically randomly initialised according to a heuristic distribution such as uniformly sampling from the range $U[\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$, where n is the number of features being fed into that layer [LeCun

et al., 2012]. This is done to keep the input of a sigmoidal non-linearity between +1 and -1 to prevent saturation. As mentioned in Section 2.2.1, saturation has the effect of restricting the flow of information between layers. This is especially important during learning so backpropagated error signals are not attenuated across layers. More recent techniques applicable to $ReLU(\cdot)$ non-linearities sample either from a normal distribution or from $U[\frac{-1}{\sqrt{n+m}}, \frac{1}{\sqrt{n+m}}]$ where m is the number of output features [Glorot and Bengio, 2010]. Together with these schemes, biases are typically initialised to zero.

2.4 Summary

Manifold learning algorithms, in particular CNNs can be used in computer vision tasks while remaining robust against the high dimensionality of image data. A typical CNN consists of multiple convolutional layers followed by an MLP. For purposes of modelling visual similarity, CNN branches with shared weights can be fed into a similarity metric to create a Siamese Network which we use in our proposed model stated in Chapter 4. A loss function is defined to measure the extent to which a specific set of weights in the model capture the mapping between images and their respective ground truth labels. To optimise this mapping, in a process called training, an objective function called a loss function is defined. The loss function is typically minimised in a process called gradient descent by iteratively adjusting weights in the direction of maximum local improvement.

Chapter 3

Related Work

The following sections discuss related work to our proposed model presented in Chapter 4 and results in Chapters 5 and 6. In particular, we discuss work that uses one-shot learning in Section 3.1, material recognition in Section 3.2, affordances in Section 3.3 and visual analogies in Section 3.4. In each of these sections we summarise the key points of related work and then point out the similarities and differences to our presented work.

3.1 One-shot and Few-shot Learning

Computer vision models are typically required to make decisions based on fixed classes and therefore need many explicit pairs of images and labels. This presents a problem when we want a model to recognise classes it has not seen before while also having little data to learn the new class. In computer vision, one common approach is similarity learning which finds a common embedding between images to match similar images without using labels.

This typically works by the model being first trained to extract features from a set of images. It is then presented with a single example of a new class that is different to the original set of images and the model is thereafter required to match the new class to unseen images. The example image is therefore analogous to a class label, which means the model can quickly classify unseen labels compared to models that are trained to recognise fixed classes which cannot. When a model recognises classes based on one example it is called one-shot learning [Fei-Fei et al., 2006]. Related to this is few-shot learning which is roughly defined as a model that recognises classes after being exposed to less than half a dozen examples [Triantafillou et al., 2017]. One-shot learning techniques create a joint embedding that are the common features between inputs that allow them to generalise to unseen classes, assuming the embedding remains constant [Li et al., 2015; Han et al., 2015].

The problem of having a small dataset to generalise to unseen classes and its proposed solution in form of one-shot learning bears a strong resemblance to the problem and proposed solution within the presented work. Like one-shot learning we also aim to recognise materials without labels when we are required to match an object image and a potentially unseen material image. We also approach this problem in a similar way by using similarities between images to find the material that present in an object even if we do not know what the names of a given object or material. For this reason our model does not output classes, as would typically be used to computer vision problems.

After training, when our model performs material selection it does so when presented with a single example object. This process is closely related to one-shot learning since the appearance of example

objects effectively become classes to be recognised among available materials. This means that our model does not need additional training for unseen materials, which is useful in the task of material selection.

Modern examples of one-shot learning typically make use of Siamese networks and related models to measure similarity [Lake et al., 2011]. One-shot learning using Siamese networks has been used to match characters in the Omniglot dataset which consists of 1600 handwritten characters from 50 different languages [Koch et al., 2015]. Matching Networks, proposed by Vinyals et al. [2016], have a similar architecture to Siamese networks but also have a Long Short-Term Memory (LSTM) neural network based attention component. This component allows the network to attend to multiple example class images during training such that it can be trained end-to-end to perform few-shot learning problems. The use of a Siamese network to solve one-shot learning problems are therefore an inspiration for the approach taken in the proposed model.

Related to the cosine embedding loss used in the presented model is the triplet loss where each batch element has three components: an anchor as a reference point, a positive example which is similar to the anchor and a negative example which is different to the anchor [Schroff et al., 2015; Hoffer and Ailon, 2015]. This is in contrast to the cosine embedding loss in the presented work which only makes use of an anchor with a single example which could be either positive or negative. Application of the triplet loss instead of the cosine loss used in the presented is left as future work.

3.2 Material Recognition

Material recognition is a field within computer vision that attempts to classify a material in a given image into categories such as wood, ceramic, etc. Early approaches by Varma and Zisserman [2005] have focused on classifying textures within grey-scale images by matching the output of fixed filters such as edge detectors with learned textons. Other early work by Sharan et al. [2009] ran tests on humans to classify 1000 images of material patches split into nine material categories. A computational implementation of this behaviour was then given by Liu et al. [2010] using a similar dataset. They did this by applying a variety of hard-coded features such as colour, SIFT and Gabor filters to perform classification. These filters are functionally analogous to the filter banks that exist within the CNN model in the presented work. They also aim to exploit the local statistical properties of images. CNNs, however, have multiple layers of filters with non-linearities and can therefore model complex non-linear filter behaviour [Wiatowski and Bölcskei, 2018].

Instead of using hard-coded features, Bell et al. [2015] learn features by using a CNN based model with conditional random fields to perform material recognition as well as segmentation. They recognise a total of 23 material categories in scenes. This is similar to the presented work since we also learn features instead of using hard-coded features. However, it also differs since we apply our method to full isolated objects and not full scenes or patches within a scene. We also deal with 3D data from multiple angles whereas Bell et al. [2015] only look at scenes from single angles.

Material recognition is related to material selection in the presented work since we also need to recognise the presence of materials in an object to perform material selection. However, prior work in material recognition has mainly focused on recognition with fixed material classes. This is in contrast to the presented work which recognises materials by being shown an example and is therefore not restricted to fixed material classes. Unlike existing material recognition research we therefore demonstrate the ability to transfer knowledge to the task of recognising unseen materials. Our work is therefore an extension of previous work on material recognition.

3.3 Affordances

Humans use visual cues encoded in their environment, known as affordances, to enable them to interact with unfamiliar objects. This is possible since unfamiliar objects contain easily recognisable visual cues, such as the shape of an unfamiliar handle giving the cue that one can grasp the object. Such visual cues are called affordances because they afford recognisable actions, such as affording the action of grasping in the case of a handle [Gibson, 1986]. Affordances can be used to make decisions under dynamic conditions since a particular affordance will transfer across domains.

Since affordances are a powerful behavioural tool they been extended beyond an analysis of human behaviour to the behaviour of automated agents [Horton et al., 2012]. An example affordances applied to agents is Myers et al. [2015] who attempt to find the components of household objects such as spatulas and hammers that correspond to affordances which can be used for tasks such as cutting and pounding without having seen the particular examples of these objects before. As shown in Figure 3.1, Nguyen et al. [2016] extend this work by using CNNs operating on RGB-Depth (RGB-D) images to achieve greater accuracy in recognising the areas of objects that are useful. A similar approach has also been applied to detecting affordances in scenes to determine if surfaces are walkable, reachable, etc. [Roy and Todorovic, 2016]. The proposed agent in the presented work also uses visual cues for the purpose

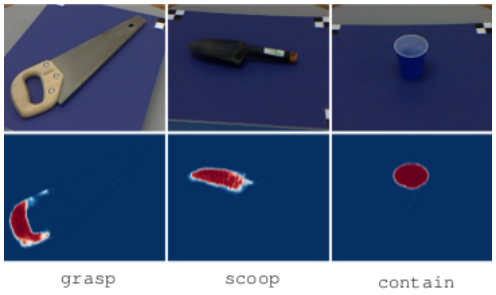

Figure 3.1: (Top row) Objects from test set. (Bottom row) Detected affordances. [Nguyen et al., 2016].

of planning. In terms of affordance literature, our work can be roughly interpreted as extracting visual cues from objects that afford the action of material selection. Like affordances, visual similarities allow adaptation to unseen data, namely materials. The affordance literature has therefore inspired our use of visual features to accelerate decision making in unfamiliar conditions.

Similar to the presented work, affordances have been used in Minecraft to prune the search space of possible actions [Barth-Maron et al., 2014]. In particular, Barth-Maron et al. [2014] use object affordances such as a door as being a passage to prune a list of possible paths leading to a desired object. This work has inspired our experimental setup where we use affordance-like features that are visual similarities to prune a list of possible recipes. Ontologies associated with objects have also been used gain an increased degree of resourcefulness in unseen environments [Balint and Allbeck, 2013]. For example, knowing to use a shoe for hitting a nail instead of a hammer can be determined by retrieving the properties of a shoe from an ontology or hierarchical database describing a shoe and related objects. The presented work therefore draws inspiration from Barth-Maron et al. [2014] and Balint and Allbeck [2013] by its use of affordances to accelerate policy searching in unfamiliar situations without additional training.

Our proposed framework also has similarities with affordance based tool and grasping research. An example is using hard-coded affordance features to cluster the grasping types of 3D tool models [Mar et al., 2015]. This allows for the detection of grasping configurations of unseen tools. Grasping research relates to the proposed method since we also use visual similarities to solve problems using tools. Other

examples of an agent using affordances include a robotic arm learning to interact with novel objects using a simulated robotic arm [Sinapov and Stoytchev, 2008] or a real-world robot arm [Ridge et al., 2010]. The existence of real-world affordances being used in robotics domains influence our proposal that visual cues are a adequate means for performing material selection in a future real-world domain.

3.4 Visual Analogies

Visual analogies are inferred mappings between pairs of source images, which are then applied to novel target image pairs [Stafford, 2001]. An example is Sadeghi et al. [2015] who use a Siamese network to predict visual relations of objects in the following form: given a source mapping $A \rightarrow B$, find x in the relationship $C \rightarrow x$. An example shown in Figure 3.2 is that given a source mapping between pictures of brown bears and white bears, find an image that completes the target mapping between a brown dog and a list of candidate animal images. This problem is related to the presented construction problem in that we want to find mappings between materials and objects given prior analogous mappings. Their approach of using a Siamese network with CNN branches to rank a list of candidate images is a key influence of the model in the presented work. Their work demonstrates that Siamese networks are an appropriate means of ranking images according to visual similarity and therefore are an appropriate approach for solving the presented problem.

Figure 3.2: Visual analogy example from Sadeghi et al. [2015] posing a colour analogy problem.

In a planning setting, Fitzgerald et al. [2014] use visual analogies to transfer analogous skills to unseen objects. They use SIFT features to find transformations or skills that connect before and after overhead images of objects that have undergone an operation such as pouring or opening. They aim to transfer skills from a source domain containing one set of objects to a target domain with a different set of objects. In particular, when presented with an unseen object in the target domain, they aim to find the most similar skill in the source domain.

Inspiration for image addition in our Minecraft model and baseline comes from Reed et al. [2015], who performed image additions using an additive CNN with a decoder to predict the visual appearance of video game character sprites in unseen orientations. Their additive CNN aims to find an embedding such that features can be added and subtracted in a meaningful way. They refer to this process as manifold traversal since when embedded images are added together they are assumed to result in an embedding that is close to an embedding of a target image within a manifold of images.

For example, one feature may correspond with the angle of rotation of the game sprite. When the difference in angle between two game sprites are added to a third sprite, the total angle will be represented as a single feature that can be used to create the same feature extracted from a new image. This is similar to the presented work in that we also aim to create features that are amenable to addition. Work by Reed et al. [2015] has therefore inspired our use of image addition in our Minecraft multiple material model.

In the Minecraft domain our method extends the idea of drawing analogies between pairs of images, that are materials and objects, to multiple images and a single image, that are materials and a tool. Our Minecraft model can therefore be interpreted to use combinations of visual analogies to perform the task of material selection.

3.5 Summary

Since our proposed model is capable of ranking the similarity between materials and an object, it is related to one-shot learning. Our work is also related to material recognition since we recognise the materials present in objects. Material recognition is however restricted to fixed classes compared to our model which is not. The visual similarities used in the presented work relate to affordances since they are visual cues used in a planning or classification setting. Further similarities exist between the proposed model and visual analogies since we aim to infer the visual mapping between objects and materials based on analogous prior mappings.

Chapter 4

Visual Similarity

In this chapter we present a method for modelling the visual similarity between objects and materials, thus allowing us to determine the likely constituent materials of a given object. In particular, we introduce two models using the techniques described in Chapter 2. First, in Section 4.1 we present a baseline model which is intended to be a conceptual starting point for our full model. Then in Section 4.2 we introduce our full model used in Chapters 5 and 6. In both these sections we first present a single material selection model and then a multiple material selection model. Finally, we discuss justifications for the presented model in Section 4.3.

4.1 Pixel Correlation

In this work materials and objects are represented using images. The simplest way of measuring similarity between data points is with distance or correlation functions. However, distance and correlation functions typically operate on vectors. We therefore reshape images into vectors so they can be correlated. For single material objects, our baseline correlates the relative pixels between a material image $\boldsymbol{M}$ and a goal image $\boldsymbol{G}$ as

$$Corr(\boldsymbol{M}, \boldsymbol{G}) = cos(\boldsymbol{M}, \boldsymbol{G}). \tag{4.1.1}$$

where our correlation function is the cosine similarity. The cosine similarity ranges from -1 to 1, where 1 is perfect correlation, 0 is no correlation and -1 is inverse correlation.

For an object with two materials we propose a multiple material baseline that adds two materials together before correlation. To be clear, each pixel is added to the corresponding pixel of the other image while not reshaping images and concatenating them into one vector. Addition is preferred over concatenation because with addition the sum image vector and goal image vector have the same length. As mentioned in Section 3.4, addition has the effect of adding similar extracted features together. Due to the addition operation, the two material baseline model is commutative. This models the fact that order does not matter to select materials for crafting. In our formulation, material pairs can include two of the same image to account for the situation where a recipe only has a single ingredient.

Examples of images being added together and being compared with a goal object are shown in Figure 4.1. Without changing ranking order in the examples given, we normalise images between -1 and +1 for visualisation purposes instead of between 0 and 1 as for experiments in the results sections. Note that the image sum in the first row appear to bear a strong resemblance to the goal, while the image sum in the second row does not. The cosine similarity between the sum and goal images capture this appearance since the cosine similarity in the first row is greater than the cosine similarity in the second row.

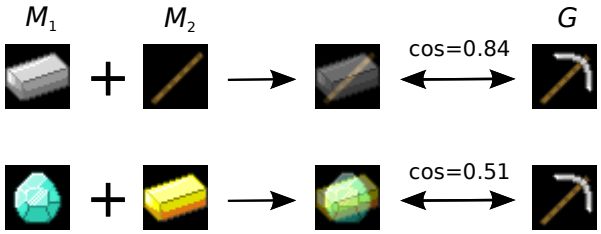

Figure 4.1: Examples of high scoring and low scoring Minecraft baseline material selections.

Here we measure the cosine similarity between the *sum* of the two material images and the goal object we wish to construct, $\boldsymbol{G}$. This is stated formally as

$$Baseline(\boldsymbol{M}_1, \boldsymbol{M}_2, \boldsymbol{G}) = cos(\boldsymbol{M}_1 + \boldsymbol{M}_2, \boldsymbol{G}), \tag{4.1.2}$$

where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are the two material images. Additions are pixel-wise and are not clamped.

To find the best images to use to construct a particular goal, we execute the model on all possible policies. For single materials this is just all possible materials. For multiple materials this is all the unique pairs of available materials $\mathcal{M}$ which are all combinations plus all materials paired with themselves. To find the materials that have the highest similarity with the object we want to construct, correlations with $\boldsymbol{G}$ are ranked and construction attempts are made until a correct set of materials is found. The aim of the model is therefore to reduce the number of construction attempts needed to produce an object. This is in contrast to our model outputting confidences in material classes where our model would be limited to seen materials as is the case with previous work stated in Chapter 3. Stated formally, the optimal policy predicted by the model for single materials is therefore

$$\pi_1(\boldsymbol{G}) = \arg\max_i Corr(\boldsymbol{M}_i, \boldsymbol{G}), \tag{4.1.3}$$

and for two materials is

$$\pi_2(\boldsymbol{G}) = \arg\max_{i,j} Baseline(\boldsymbol{M}_i, \boldsymbol{M}_j, \boldsymbol{G}), \tag{4.1.4}$$

where $i \leq j$.

The multiple material method is stated fully in pseudocode in Algorithm 2. The algorithm first creates a list of material combination scores. It then sorts the list by score. Finally, it goes down the list of candidate materials until it manages to find a successful combination of materials $\bar{M}_1$ and $\bar{M}_2$. In this case the model is the baseline model but the same processes applies for the full model.

Algorithm 2 Method pseudocode for multiple materials. For single material pseudocode replace M_1, M_2 with M as well as ignoring the contribution of combinations to candidate materials.

Require: $\mathcal{M}, G, \bar{M}_1, \bar{M}_2$
  candidate_materials ← combinations($\mathcal{M}$, 2)
  candidate_materials.append(zip($\mathcal{M}$, $\mathcal{M}$))
  for $M_1, M_2 \in$ candidate_materials **do**
    score ← model(M_1, M_1, G)
    scores_materials.append(score, M_1, M_2)
  end for
  scores_materials.sort_by(score)

for score, $M_1, M_2 \in$ scores_materials **do**
  if $M_1, M_2 = \bar{M}_1, \bar{M}_2$ **then**
    break
  end if
end for
return M_1, M_2

4.2 Siamese Neural Networks

We now build on the intuition of our correlation model and multiple material baseline by extracting visual similarities from input images before correlation takes place. In particular, we place CNN feature extractors with shared weights before correlation, as shown in Figure 4.2. This model takes the form of a Siamese network[1]. As stated in Sections 3.2 and 3.1, our model deviates from prior work regarding material recognition by measuring image similarity instead of outputting a class predictions. This is because we want our model to generalise to unseen materials without further training.

For multiple materials, we similarly place CNN modules with shared weights at all inputs of the model. This structure, called CraftNet, is similar to a Siamese network but, as shown in the architecture diagram in Figure 4.3, the outputs of the material CNN modules are added before similarity with the goal is computed.

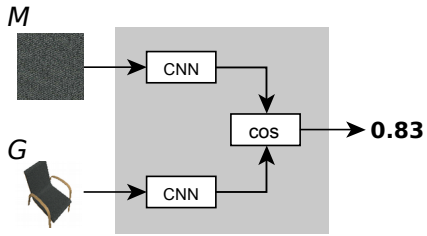

Figure 4.2: Single material model architecture (in grey) with example input and output.

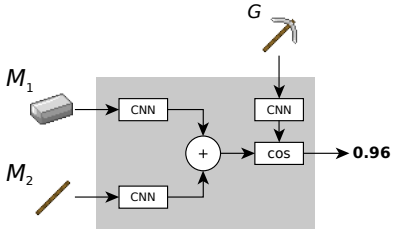

Figure 4.3: Double material model architecture (in grey) with example input and output.

We use CNN layers in an attempt to recover latent variables such that they can be more easily added and correlated as vectors. Each latent variable is interpreted as a visual feature, which may include texture, colour, etc. The sum operation therefore takes the combination of visual features from materials and allows them to be correlated with the visual features of the goal.

To train models with CNN branches, we use the cosine embedding loss mentioned in Chapter 2. In the case of the single material model we want the model to output construction correlations. Model weights are therefore optimised such that the model outputs 1 when a material is predicted to exist in an object and -1 when it does not. Similarly, when training our multiple material model, the sum of the embedded ingredient vectors is expected to share similarities with the embedded goal item and is also therefore regressed to 1 when presented with a correct recipe. The sum of embedded ingredients is also expected to be dissimilar to the embedded tool and is therefore regressed to -1 when presented with an incorrect recipe.

During testing, as with our baseline models, all possible unique materials and pairs are fed into the model to produce construction correlations. Construction attempts are also then made by going down a list of ranked construction correlations. This enables the ranking procedure in Equations (4.1.3) and (4.1.4) to also accelerate finding the correct materials necessary to construct an object.

[1]We implement our models using PyTorch (`pytorch.org`) and Lua Torch (`torch.ch`).

4.3 Discussion

The use of a similarity metric in both models means that one set of materials can map to many goal objects. This is especially useful for the task of selecting materials in Minecraft where a single set of materials can craft many different tools. This is because different tools are crafted depending on the placement of ingredients on a 3×3 grid called the crafting table. For our work, we do not address such geometric considerations as we aim to only find which material are necessary to craft a tool.

Both the single material and multiple material baselines and models use the cosine similarity. This particular measure of similarity was mainly chosen for its prior use by Sadeghi et al. [2015] on a similar image matching problem discussed in Section 3.4. Another reason we choose the cosine function, especially when compared to other similarity functions like the Euclidean distance, is that this metric is invariant to the magnitude of input vectors. This is important in the case where a tool needs two instances of the same ingredient. Here, the cosine between the sum of two identical images $\boldsymbol{M}_1 + \boldsymbol{M}_1$ and another image $\boldsymbol{M}_2$ is the same as the cosine between a single image $\boldsymbol{M}_1$ and the other image $\boldsymbol{M}_2$. Since we represent single ingredient recipes as $\boldsymbol{M}_1 + \boldsymbol{M}_1$ where $\boldsymbol{M}_1$ is the ingredient, this cosine property models the intuition that crafting both one or two ingredients should have the same result. More sophisticated distance functions are left as further research.

Empirical tests have shown that a Siamese neural network, which has two branches, outperforms a CNN with a single branch that takes in concatenated images when dealing with visual analogy problems [Sadeghi et al., 2015]. Since our model is inspired by Siamese neural networks and deals with visual analogy problems, we draw inspiration from these results. Furthermore, by having separate branches for each input the model does not prematurely mix data from separate images until the addition and cosine operations are performed. This is in contrast to a monolithic CNN model which would add images from different objects together in early convolutional layers (as per Equation (2.2.3)).

In our baseline models we exclude convolutional layers. By comparing the performance of the baselines with the full models that do have convolutional layers, we can determine if convolutional layers do indeed extract visual features that aid the task of material selection.

4.4 Summary

In this chapter we introduced the models used in this work together with baseline models that are both inspired by Siamese networks that determine the visual similarity between a given object and a material. Both approaches require raw materials and desired object images as inputs, and both output the correlation of input-output pairs to indicate a successful or unsuccessful construction. For the baseline we propose a simple correlation based approach and for our full model we propose the extraction of visual features using CNN modules.

Chapter 5

Single Material Selection

In this chapter we present the application of our model specified in Chapter 4 to the task of selecting single materials to build an object. In particular, we apply our model to the ShapeNet dataset which is described in Section 5.1. Furthermore, in Section 5.2, we discuss the architecture of our model followed by results demonstrating the ability of our model to perform single material selection in this domain. This is then followed by a discussion of results in Section 5.3.

5.1 ShapeNet Dataset

ShapeNet is a database of 3D models intended to be used for computer vision and robotics research [Chang et al., 2015]. ShapeNet objects are split into object categories. From ShapeNetCore, a manually verified subset of ShapeNet, we select categories of objects that an assembly agent could plausibly construct from raw materials. In particular, we select household objects such as benches, cabinets and beds, while excluding categories such as aeroplanes, laptops and wine bottles. We also balance the data such that there are no more than 500 models per object category. Samples of objects that we focus on in this work are shown in Figure 5.1. Note that the objects exhibit a variety of surface textures and therefore materials, which are the subject of our study.

Figure 5.1: Examples of ShapeNet 3D models from a variety of categories that we focus on in this work.

Each model is essentially a group of polygons stuck together in 3D space. The way in which these polygons are stuck together is called a mesh, which determines the geometry of the object. The appearance of each polygon, and thus the whole object, is determined by either solid colours stating how each polygon should be coloured in the mesh or a group of images pasted on to the mesh called textures.

For our model, which operates on images to process 3D data, we render objects from all eight cardinal

views using the provided ShapeNet viewer[1]. An example rendering produced by the viewer is shown in Figure 5.2. We standardise textures and rendered object images to 128×128 images and scale pixel values between 0 and 1.

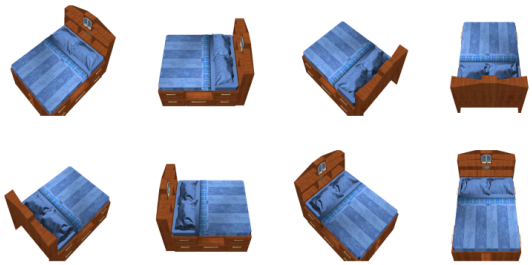

Figure 5.2: An example of all eight renderings of an object produced by the ShapeNet viewer.

Since we assume that textures largely correspond to materials, such as wood or fabric, they are critical to our study. Textures are simply images linked in the mesh data and can therefore be obtained readily. However, when the surface appearance of an object is determined by solid colours specified in the mesh data of an object, the colours do not correspond to raw materials that we focus on in this work. An example of an object with half a texture and half mesh is shown in Figure 5.3. In such a case, we assume that the solid white colour of the table, which partially covers the object, is not useful to our study. We therefore focus exclusively on objects that are largely covered by textures while excluding objects that are largely covered by solid colours specified in mesh data. Achieving such a separation is difficult since renderings do not show, by default, which parts of the image are from textures and which are from the mesh colour data.

Fortunately, since textures are just images, we can swap a texture for another texture and see what effect it has on a rendering of an object from a particular viewpoint. In particular, as shown in Figure 5.4, we swap all textures for first blue textures then red textures and compare renderings. Notice that in the first row there is a minimal change in appearance between renderings due to the appearance of an object being largely determined by mesh colour data. This is in contrast to the second row where there is a significant change in appearance between renderings due to textures almost completely covering the object.

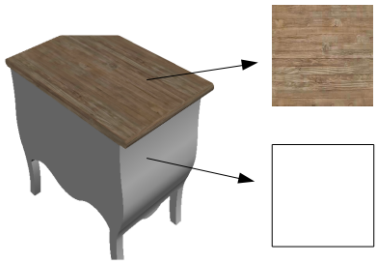

Figure 5.3: A table with both a texture and mesh data determined surface. The arrows show a wood region from a texture and solid white region from mesh data.

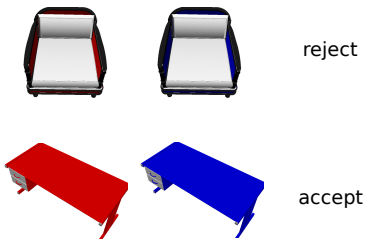

Figure 5.4: Change of textures in models to measure the impact of textures on visual appearance. (Top) Minimal change - Reject. (Bottom) Significant change - Accept.

To measure the difference between images we take Euclidean distances between images that have

[1] http://github.com/ShapeNet/shapenet-viewer

been reshaped into 1D arrays and then divide by the number of foreground pixels. We then plot distances of objects within a category as a histogram and look for a dip as a threshold that separates the set of objects that we use in our study from other objects. As shown in Figure 5.5 when distances are plotted for the chairs object category, they form a bimodal histogram that indicates the two groups of 3D models. We keep all images beyond a threshold defined at the first dip in order to keep images from the second

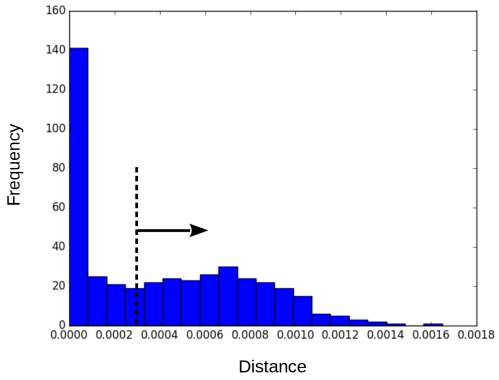

Figure 5.5: Histogram of Euclidean distances between chair renderings with modified textures. Cutoff point shown as the portion to the right of the dotted line.

group, which are images that are largely covered by textures. The use of a threshold based on the first dip is justified with a loose analogy to binarisation in image processing. Here instead of separating the background from the foreground we aim to separate the object largely covered by textures from objects largely covered by coloured mesh data. This filtering process yields a total of 1008 unique models.

5.2 ShapeNet Model

Recall that we place an emphasis on the ability of our approach to handle unseen materials. We achieve this by using the method specified in Algorithm 2 by determining the visual similarity between two images, a material and an object by assigning a score to each candidate material. We then rank the materials according to score and go down the materials list until we find the correct material. Since our ShapeNet model performs selection of a single material, we propose that visual similarity should be performed using a vanilla Siamese network as proposed in Chapter 4. Each branch of the model is an AlexNet style CNN [Krizhevsky et al., 2012] with two convolutional layers followed by one dense layer. Each branch has a total of $\sim$ 250k parameters. The full architecture of each branch is shown in Table 5.1. For convolutional layers, we double the number of features produced by each layer, as is typical for AlexNet style models. Recall that the output of these branches are connected with a similarity function as shown in Figure 5.6. We do not put a ReLU after the last layer since if all elements were positive the cosine cannot evaluate to negative values.

We predict materials within a known object category and expect the model to generalise to unseen materials and objects within that category. In particular, we hold back 20% of all materials in an object category and test on the remaining 80%. To do this we take all objects with unique textures within a category to split the objects into train and test sets. Once we have split the unique materials into train and test we place all objects with those materials into the dataset that contains the same material. The textures from the original 3D model files and are paired with renderings of those objects to form the train and test sets that we use in experiments. We cap the number of unique materials in category to 160 since

Table 5.1: Full ShapeNet model architecture for each branch. ReLU non-linearities are used after all conv layers.

Layer	Dimensions
Input	$128 \times 128 \times 3$
1. conv	$3 \times 3 \times 16$
2. maxpool	2×2
3. conv	$3 \times 3 \times 32$
4. maxpool	2×2
5. dense	$(32 \times 30 \times 30) \times 8$
Output	8

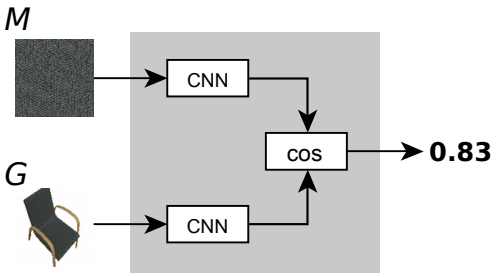

Figure 5.6: Siamese network.

this is the minimum number of materials in a category. This allows us to combine results from different categories (as is the case later in Figure 5.9). This means that there are 128 materials in the training set and 32 materials in the test set.

A simplified example of this setup using the chair category is where we may have chair A with a dark wood texture, chair B with a light wood texture, chair C with a leather texture and chair D with a fabric texture. Say we split the objects as A and B for training and C and D for testing. In such a case we would train our model to match the light wood texture to a rendering of chair B and the dark wood texture to a rendering of chair A. For testing we would then require the model to match the fabric texture with a rendering of chair D and leather with a rendering of chair C. The intention of such a setup is to test the ability of our model to generalise to unseen materials and to generalise to somewhat different objects within a particular category of objects. For our simplified example we would have tested the ability of our model to generalise to chairs C and D while also generalising to unseen materials that are leather and fabric.

A key goal of the proposed ShapeNet model is to accelerate the task of recommending materials when the model is exposed to unseen materials when trained on objects and materials from the same object category. Performance should therefore be measured in a way that decreases when our model makes more failed attempts and thus fails to accelerate material recommendation. We measure this ability using the average number of failed attempts to select materials needed to build an object when going down a list of ranked materials. This average is then normalised by dividing it by the maximum number of possible attempts that is the number of unique materials in an object class test set, which is 32. Finally, we minus this from one to give us $1 - \frac{failed}{possible}$. We refer to this measure as the mean recall of the model since our problem is a ranking problem [Sadeghi et al., 2015]. Note though that our model only has a single correct answer and is therefore not strictly speaking a recall but is nevertheless analogous to it. A perfect model would have no failed attempts yielding a recall of 1, while a completely imperfect model would have failed attempts equal to the number of possible attempts yielding a recall of 0.

We use benchmarks to justify the complexity of our model and to determine the significance of the results obtained by our model. In addition to the correlation benchmark stated in Chapter 4, we also compare our proposed model against three additional benchmarks. First, we uniformly sample from available remaining materials, which we call *chance*. Since our model attempts to bring a material near the top of the list of ranked materials, we expect chance to place the correct material in the middle of the material list. Chance is then a uniform distribution between 0 and 1 with expectation equal to a half.

Second, we compare our proposed model to another model that has an identical architecture but with random weights. This tests if our trained model does indeed learn features instead of just using features resulting from the untrained architecture of the network. This is a concern since the untrained architecture of CNN models alone has been shown to contribute to the creation of meaningful features in vision tasks [Saxe et al., 2011]. Without this comparison, we do not rule out the possibility of our model may appear to perform well while only using features resulting from the architecture.

Third, we compare our model against colour histogram based features. Here we take a histogram of the three colour channels and concatenate the channel histograms together to form a feature vector. The idea of using a colour histogram is intended to test if our model learns additional features beyond simple colour combinations that may seem to characterise the visual appearance of objects and materials. For example, the idea of comparing the colour of the material and object in Figure 5.6 is a plausible yet simple way of determining visual similarity. This method is therefore suited as a benchmark that our model should outperform to justify the model complexity.

For an example executions of the trained model on an unseen table and chair refer to Figures 5.7 and 5.8. In the first figure we have an example where the model performed well with correct material found with a single failed attempt. In the second figure we have an example where the model performed poorly with 31 failed attempts. Note that the high ranking materials in Figure 5.7 all look similar to the target object while low ranking materials look dissimilar. This confirms that our model achieves our goal of using visual similarity to rank materials.

Figure 5.7: Example with high recall ranking. (Top) An object – A table. (Bottom) Material selection attempts. Correct attempt marked with a tick.

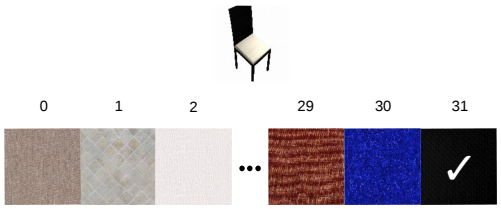

Figure 5.8: Example with low recall ranking. (Top) An object – A chair. (Bottom) Material selection attempts. Correct attempt marked with a tick.

Results are averaged across object categories to give overall performance. Mean recall results for experiments are shown in Table 5.2. A plot of these results are also shown in Figure 5.9. In particular, Figure 5.9 shows the proportion of unseen test objects correctly constructed at a threshold of k the number of failed attempts, yielding Recall at Top-k [Sadeghi et al., 2015]. For this figure, results for chance are averaged over 100 trials and have error bars of one standard deviation. The purpose of the Top-k plot is to see the whole distribution of failed attempts instead of just the measures of central tendency that are the average recalls shown in Table 5.2. The Top-k gives us insight into what portion of objects our model managed to find materials in few failed attempts and what portion of objects our model managed to find correct materials in many failed attempts. Interpretation of results is given in Section 5.3.

Table 5.2: Mean recall of the trained ShapeNet model compared to our correlation baseline, colour histogram baseline, randomly initialised weights and chance. Standard deviation errors are shown for chance experiments. Best result in bold.

Unseen Data	Pixel Correlation	Colour Histogram	Random weights	Model	Chance
Materials	0.522	0.525	0.559±0.256	**0.751**	0.497±0.332

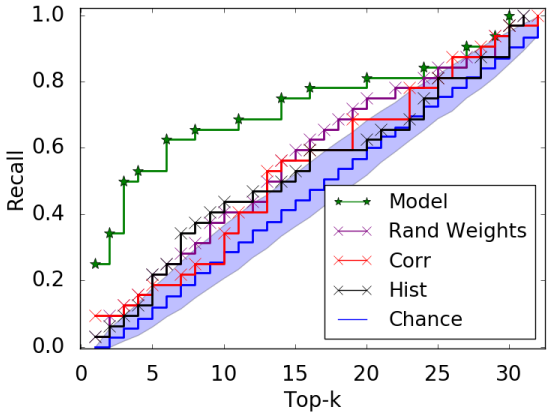

Figure 5.9: A comparison of the number of failed attempts k needed to select materials for test objects between chance, the proposed model, colour histograms and the correlation baseline.

5.3 Discussion

In the majority of cases the trained model performed better than chance, our baseline model, the colour histogram baseline and the full model with randomised weights. This demonstrates that our ShapeNet model does indeed learn features that are useful in accelerating the task of material selection beyond baseline methods. In particular, the fact that our trained model outperformed our baseline with no convolutional layers shows that the conventional layer learns useful features. Based on the fact that our model also performed better than the colour histogram benchmark, it therefore managed to learn features that were more complex than simple colour combinations. Furthermore, the fact that our proposed model outperforms the same model with random weights shows that it learns features that are not as a result of architectural anomalies within our model.

However, as shown in Figure 5.9, at a threshold of around 25 failed attempts the model line comes close to the benchmarks. This means that in terms of the attempts to construct some objects, our model performed comparably to the baselines. One possible reason is that our filtering method does not exclude 3D models which have an appearance largely determined by internal mesh data. An example of this is shown in Figure 5.8 where the appearance of the seat is determined by mesh data while the back rest is not. Here the top ranking materials are similar in appearance to the seat but not the back rest, showing that the model matches the mesh colour and not the texture.

Neural networks are difficult to interpret due to their use of multiple layers that together correspond to complex operations. We use CNNs that are a class of neural network. This means that results from our model are difficult to interpret. Attempts to interpret CNNs in an computer vision context have been made by Zeiler and Fergus [2014]. They create a two dimensional plot of how different parts of an image influence the classification probabilities made by a CNN by blocking out different parts of an image fed into the model. We perform a similar experiment by plotting the similarity score from the output of our

model as we vary the corresponding position of a small grey square (30×30 pixels) as it slides across the object input. We normalise these maps between 0 and 1 to emphasise variation. The maps are shown in Figures 5.10 and 5.11. By blocking out regions of the object image we aim to determine which parts are

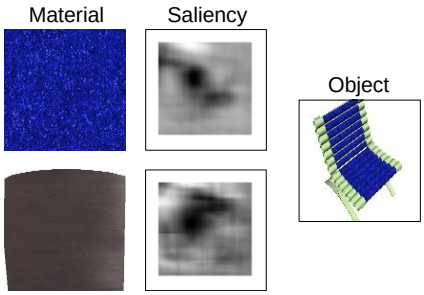

Figure 5.10: A plot of the regional influence of a chair with two textures as inputs. This example had a perfect recall of 1.

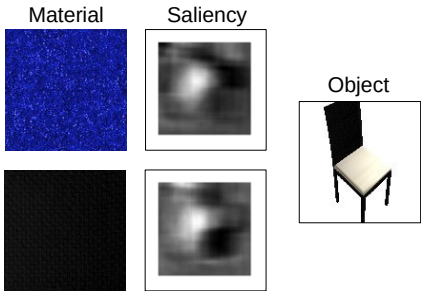

Figure 5.11: A plot of the regional influence of a chair with two textures as inputs. This example had a poor recall of 0.

needed for the model for it to output high correlation scores. When blocked out, an area that positively contributes should therefore be dark, while an area that contributes negatively should be white and a neutral area should be grey.

To create the plots we expose a trained model to objects from the training set. We also expose the model to two different textures from the training set, one that exists in the object and one that does not. Note how in Figure 5.10 when the model is exposed to the correct texture, dark areas on the map overlap with areas where the texture exists on the object. This is because those are the regions that contribute most to high correlation. However, when exposed to the incorrect texture, the dark regions have less of an overlap with the totality of the chair. In this high recall case the dark regions tend to match the expected behaviour of an ideal classifier. In Figure 5.11, the correct texture triggers the seat region where the texture does not exist, while dark regions for the incorrect texture are essentially random. This shows that, as stated previously in Section 5.2, when the model fails it uses information from parts of the image that do not have a texture. In this low recall case the dark regions tend to match the expected behaviour of a poor classifier.

5.4 Summary

In this chapter we demonstrated the effectiveness of our model described in Chapter 4 at performing a single material selection task using the ShapeNet dataset. We showed this using experiments that demonstrate our model outperforming a variety of baseline methods. We also provided some intuition about the behaviour of our model by presenting examples of our model ranking material. We also gain some intuition about the regions that our model uses to make decisions by noting the effect of blocked out parts of images fed into our model.

Chapter 6

Multiple Material Selection

In this chapter we present the proposed model in Chapter 4 applied to the task of selecting multiple materials for object construction. This model builds on the model presented in Chapter 5. This Chapter is structured as follows. In Section 6.1 we describe our use of sprites from Minecraft. In Section 6.2 we discuss our Minecraft model followed by results and analysis. This is then followed by a discussion of results in Section 6.3.

6.1 Minecraft Data

In Minecraft, when a player collects an object from the environment the object is represented in the player's inventory using small images called sprites. A collection of sprites is called a texture pack. The default texture pack is called the Vanilla texture pack. The Vanilla texture can be replaced with other textures, such as those shown in Figure 6.1, providing an alternative visual style to objects in the player's inventory.

Figure 6.1: Examples from the texture packs used as training data in experiments. In experiments 4, 5 and 7, two of the texture packs were unseen.

In Minecraft, players can combine inventory items in a process called crafting. We focus our attention only on selecting necessary ingredients and ignore the position and quantity of materials on the in-game crafting grid or table. Out of the many inventory items in Minecraft, we focus on recipes that craft a subset of all items. Namely, we focus on tools made from wood, stone, iron, gold and diamond[1]. The ingredients that can be used to make these tools include refined materials and their respective ores. For example, we include iron ingots as well as iron ore as part of our study. The dataset then has a total of 26 tools and 11 ingredients. All the sprites from a single texture pack used in the presented work are shown in Figure 6.2. Minecraft recipes consist of the combination of materials to create an object.

[1]Tools are defined in http://minecraft.gamepedia.com/Tools.

Figure 6.2: All sprites from the Faithful texture pack that are used in the presented work.

An example of a Minecraft recipe using materials M_1 and M_2 to form goal tool G is shown in Figure 6.3. Materials on the right are combined with to produce an object on the right. Recipe ground truths were collected from the Minecraft Wiki[2]. The collected recipes consist of mappings between one or two inventory items and a single tool.

Figure 6.3: Crafting an iron pickaxe from a stick and an iron ingot in the Vanilla texture pack.

Tools and ingredients in vanilla Minecraft are represented by 32×32 pixel sprites. For the purpose of our experiments we resized sprites from other texture packs to 32×32 pixel images. Pixel values were scaled between 0 and 1, but were not adjusted for brightness or contrast. Sprite samples from different texture packs are shown in Figure 6.1. Transparency layers for sprites in the Faithful, Arestians's Dawn and LB Photorealism texture packs were used to set the background to black. This makes these texture packs have the same background as the Vanilla texture pack.

6.2 Minecraft Model

Since our Minecraft model performs selection of multiple materials it needs to determine the similarity between the combination of two images and another image before ranking can take place. As stated in Chapter 4, we propose that this should be done using a modified Siamese network shown in Figure 6.4. Each branch of our Minecraft model is an AlexNet style [Krizhevsky et al., 2012] two layer convolutional and one dense layer based model with $\sim$ 200k parameters with architecture shown in Table 6.1.

We evaluate our model on a total of seven experiments, using different sets of raw materials, goal tools, and texture packs. The experiments are designed to use increasingly sparse data. In particular, experiments 1–3 each exclude individual categories of tools, materials and texture packs (representing different styles) from training data respectively. In particular, in experiments 1–3, we exclude tools,

[2]http://minecraft.gamepedia.com/. See Appendix A for a list of recipes.

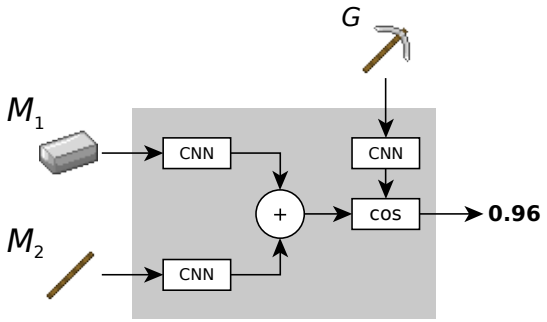

Figure 6.4: Minecraft model architecture (in grey) with example input and output.

Table 6.1: Full Minecraft architecture for each branch. ReLU non-linearities are used after all conv layers.

Layer	Dimensions
Input	$32 \times 32 \times 3$
1. conv	$3 \times 3 \times 4$
2. maxpool	2×2
3. conv	$2 \times 2 \times 16$
4. maxpool	2×2
5. dense	$(2 \times 2 \times 16) \times 256$
Output	256

materials and texture packs respectively. Training data is further reduced in experiments 4–6, where we exclude combinations of two of these unseen categories. In particular, in experiments 4–6, we exclude tools and texture packs, materials and texture packs and materials and texture packs. Finally, in experiment 7 we exclude data from all categories that are materials, texture packs and tools. The schedule for which materials in which experiment is stated in Table 6.2.

Table 6.2: Tool material types used as training data in the first 6 experiments. Experiment 7 uses the same tools as experiment 6, but is only trained on two texture packs, similar to experiments 4 and 5.

	Wood	Stone	Iron	Gold	Diamond
Axe	1–6	1–6	1–6	1,3	1,3
Hoe	1–6	1–6	1–6	1,3	1,3
Sword	1–6	1–6	1–6	1,3	1,3
Pickaxe	2–4	2–4	2–4	3,6	3,6
Shovel	2–4	2–4	2–4	3,6	3,6
Shears	none	none	1–6	none	none

In addition to decreasing training data, we also change the orientation of material test images by flipping them along their vertical axis. The experiments are therefore constructed to test the robustness of our model against increasing degrees of unseen data and visual variation. In all experiments baseline methods are benchmarked against chance, which uniformly samples from available remaining material combinations.

For example in experiment 6, we train the model on sprites from all four texture packs, which are shears together with wood, iron and stone hoes, axes and swords. We then test our Minecraft model on the remaining tools and materials from all texture packs, which are gold and diamond shovels and

pickaxes. This experiment therefore tests the ability of our model to generalise to unseen materials and tools.

In addition to tool and material constraints, in experiments 3, 5 and 7, we perform experiments similar to the other four experiments but train the model on two texture packs (Vanilla 1.9 Minecraft texture pack and Arestian's Dawn) and test it on the remaining two (Faithful and LB Photorealism). For example, in experiment 3, we train on all tools and materials from these two texture packs and test the model on the remaining two texture packs. Sample sprites from all four texture packs are shown in Figure 6.1. When a model makes a prediction input images are drawn from the same texture pack as the target tool so visual similarities can be detected.

Since crafting can be successful or unsuccessful, we use both positive and negative training pairs. For example, a gold ingot and stick together with a golden sword are a positive example, whereas a stick and iron ore together with a golden sword are a negative example. In experiments where we exclude materials, the materials and the ingredients used to make those materials are also excluded from negative training pairs. Many more item combinations cannot be crafted than can be crafted since there is only a single correct answer among $\binom{11}{2} + 11 = 66$ unique pairs. This means there are far more negative training pairs than positive training pairs. To remedy this imbalance, in each training round we randomly select a different subset of 10 items from the oversampled class. This is so the model is both exposed to the whole undersampled class while not being overwhelmingly exposed to data from the oversampled class.

A key goal of the proposed model is to generalise material recommendations to novel tools where the model has not seen materials, tool types or graphical styles. To quantify generality we measure the average number of failed attempts taken to craft all unseen tools. This average is then normalised by dividing it by 65, the maximum possible number of failed attempts. The possible number of failed attempts comes from the possible combinations of materials plus materials in isolation minus one. When we subtract from one we get average recall as in Chapter 5 as $1 - \frac{failed}{possible}$. The results of the 7 experiments are summarised in Table 6.3.

Table 6.3: Mean recall of our Minecraft model compared to the multiple material baseline and chance. Experiment results for flipped sprites are in parenthesis. Standard deviation errors are shown for chance experiments. Best results are in bold.

Unseen Data	Baseline	Minecraft model	Chance
1. Tools	0.674 (0.395)	**0.826 (0.806)**	0.481±0.296 (0.542±0.315)
2. Materials	0.612 (0.301)	**0.777 (0.717)**	0.572±0.289 (0.507±0.272)
3. Texture packs	0.742 (0.531)	**0.855 (0.848)**	0.529±0.260 (0.432±0.272)
4. Tools & texture packs	0.691 (0.506)	**0.855 (0.852)**	0.430±0.332 (0.565±0.315)
5. Materials & texture packs	0.775 (0.598)	**0.851 (0.832)**	0.490±0.292 (0.581±0.294)
6. Materials & tools	0.655 (0.441)	**0.812 (0.817)**	0.555±0.268 (0.568±0.308)
7. Materials, tools, & texture packs	0.737 (0.557)	**0.860 (0.880)**	0.501±0.285 (0.484±0.287)

As in Chapter 5, Figure 6.5 shows the proportion of unseen test tools correctly crafted at a threshold of the number of crafting attempts, Recall at Top-k [Sadeghi et al., 2015]. For this figure, results for the same tool from different texture packs are averaged together while the chance model is averaged over 100 trials and has error bars of one standard deviation.

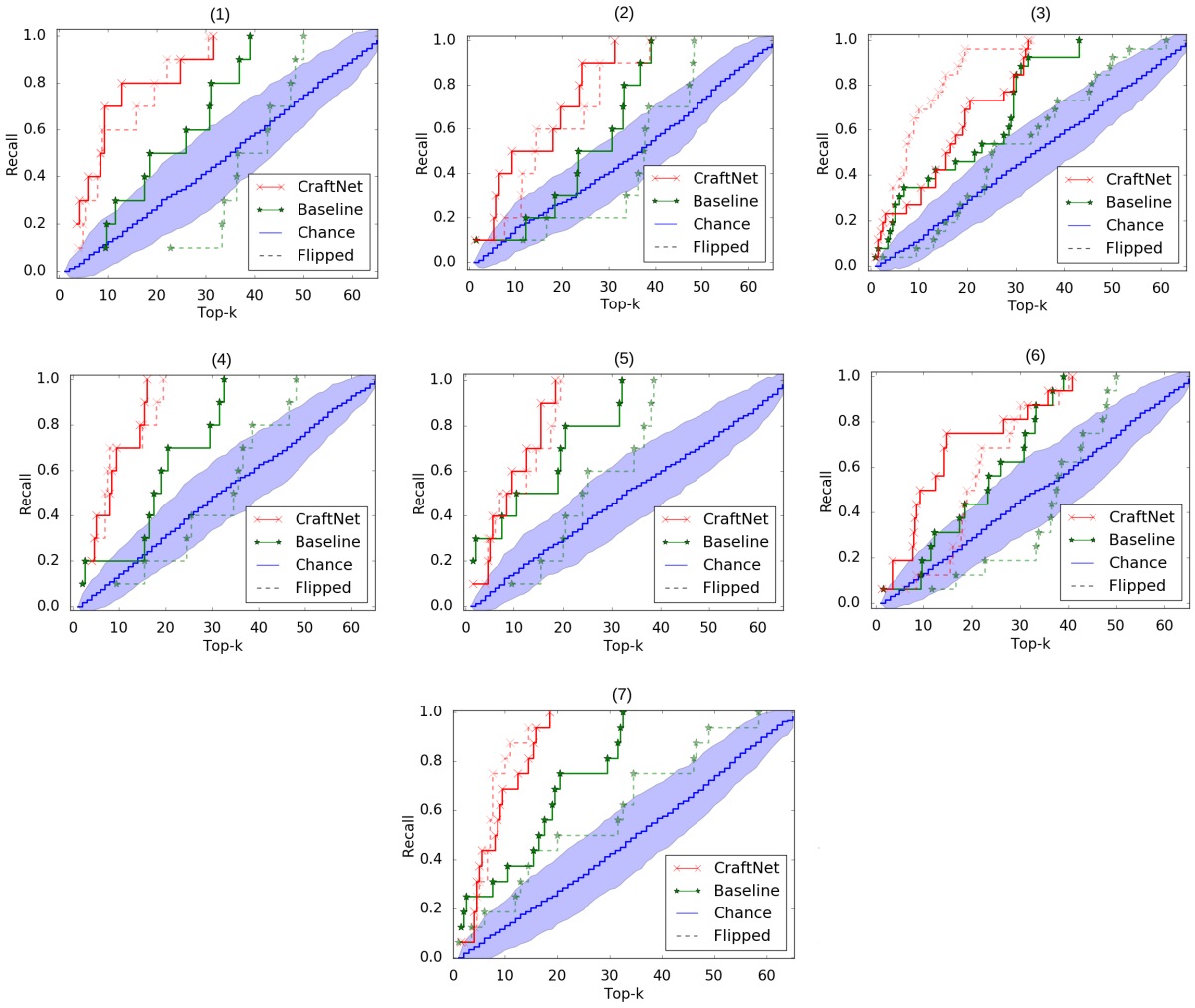

Figure 6.5: A comparison of the number of failed crafting attempts k needed to craft test tools between chance, the proposed model and the baseline in experiments 1–7. "Flipped" refers to the more difficult case of some of the input sprites being flipped along the vertical axis.

6.3 Discussion

In all experiments, our Minecraft model performed on average better than the Minecraft baseline and chance. For most experiments our model performed consistently better than baselines with the CraftNet line rarely going far past 30 failed attempts. This means that all tools took less than 30 failed attempts to craft. However, as shown in Figure 6.5(6), in the particular case of experiment 6, the CraftNet line crosses the baseline. In this case our Minecraft model therefore performed comparably to the baseline for some tools. That is, for tools such as a diamond shovel the baseline and the proposed model are comparable. An example of experiment 6 operating on golden and diamond shovels from the Faithful texture pack is given in Figures 6.6 and 6.7 respectively. In these cases the model tended to correctly rank sticks highly but also tended to rank diamonds low.

Even when some of the input sprites are flipped along their vertical axis, our Minecraft model does not decrease in performance and still performs better than chance despite having not being trained on flipped sprites (with the exception of shovels in the LB Photorealism texture pack). This is in sharp contrast to baseline which under the same conditions is comparable with chance when attempting to craft any item.

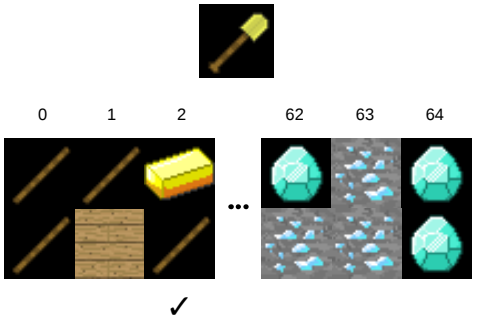

Figure 6.6: Example with a high recall. (Top) Tool to be crafted – A golden shovel. (Bottom) Candidate inventory item attempts. Correct attempt marked with a tick.

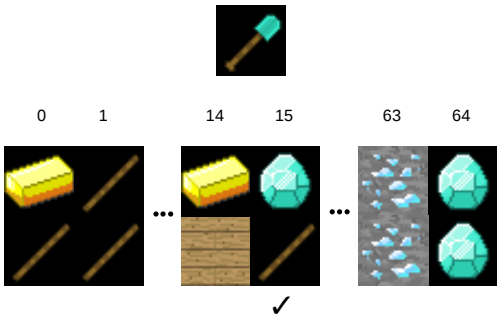

Figure 6.7: Example with a low recall. (Top) Tool to be crafted – A diamond shovel. (Bottom) Candidate inventory item attempts. Correct attempt marked with a tick.

A possible reason for this is that, as shown in Figure 6.8, flipped sprites often do not overlap with the goal item. Since the baseline measures similarity via a simple correlation, successful material selection would only be achieved when similar pixels in the object and material overlap. This result demonstrates the fragility of the baseline when compared to the proposed model. The results therefore indicate that the proposed Minecraft model is robust against simple perturbations that an agent would be expected to handle. For this reason, the addition of convolutional layers in our model are justified.

Figure 6.8: An example of poorly overlapping flipped ingredient sprites resulting in poor baseline performance. Sprites are from the Vanilla texture pack.

To better interpret the behaviour of our model we create plots as performed in Section 5.3 [Zeiler and Fergus, 2014]. Recall that we do this by plotting the similarity score from the output of our model as we vary the corresponding position of a small grey square. In this case it is a 7×7 pixel square that slides across the tool input. We normalise plots between 0 and 1 to emphasise variation. By blocking out regions of the object image we aim to determine which regions increase or decrease the correlation score outputted by our model. We expose our model to two of the same material to test the extent to which each material influences correlations in the model.

We attempt to use these plots to analyse the poor results in experiment 6. When the model is exposed to the correct material, light areas on the map indicate which parts should be blocked out to increase correlation. For example, in Figure 6.9, when exposed to a stick, the area of the map corresponding to the top of the shovel exhibits high correlation. This shows that when the top of the shovel is blocked out, the handle correlates well with the stick. However, when exposed to a gold ingot and a diamond, the map does not present any meaningful correlations. This anomalous behaviour may explain the poor recall on this tool.

We now apply the same analysis to the well performing case of experiment 1. In particular, we use the iron pickaxe from the Faithful texture pack in Figure 6.10. When we cover the handle or nearby pixels, the iron sprite map has higher correlation over the totality of that region compared to the diamond map. This corresponds to the fact that when the handle of the tool is de-emphasised, higher correlation is achieved. In the stick sprite plot there is high correlation in the head region of the tool. This similarly

shows that when the head of the tool is de-emphasised, the stick region leads to a higher correlation score. These results tentatively indicate that our model can use sensible regions of tool images to match materials and not rely on statistical anomalies from other parts of the tool image.

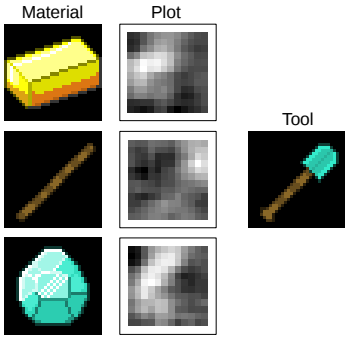

Figure 6.9: A plot of the regional influence of a diamond shovel with different material inputs. This example had a low recall.

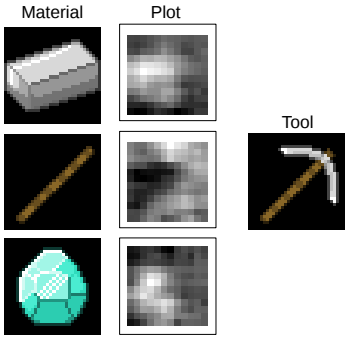

Figure 6.10: A plot of the regional influence of an iron pickaxe with different material inputs. This example had a high recall.

6.4 Summary

In this chapter we have shown that our proposed model in Chapter 4 is able to select the correct materials to craft Minecraft tools. It on average performs better than chance and a baseline under multiple constraints such as limiting training data and changing the orientation of test data. We provided some intuition about what causes our model to succeed and fail by using plots that show which regions of input images contribute to higher correlation scores.

Chapter 7

Conclusions and Future Work

We have shown that the use of visual similarity can accelerate the task of selecting the materials necessary to build an object. In particular, we have shown that by being trained on the materials present in a subset of objects, analogous construction tasks for unseen materials can be performed. Our models proposed in Chapter 4 outperform chance as well as an image correlation baseline in a single material domain and multiple material domains in Chapters 5 and 6 respectively. These domains consist of varying object types and visual appearance. The superior performance of the model beyond the baselines therefore shows that the addition of CNN-based feature extraction layers in the presented models are necessary. Furthermore, we provide evidence from maps tracking regional influence that the convolutional modules tend to localise relevant parts of an image and therefore extract useful features from object images.

Since the ShapeNet models and the LB Photorealism texture pack have realistic images, they hint at the possibility of a real-world object construction agent. However, even in these datasets, objects are still not completely realistic. They sometimes have intentionally exaggerated colour (e.g., bright blue for diamond tools and materials). Also, even in realistic sprites and rendered 3D models, there is no background clutter, which would exist in real domains. Future work can therefore focus on models that use attention as a way of ignoring visual information in the background of an agent's camera.

We have demonstrated the applicability of Siamese inspired architectures with similar branch architectures to domains with different visual characteristics such as colour, orientation and varying degrees of realism. This implies that our model architecture may generalise to domains with similar characteristics. In particular, we have demonstrated that our methods are applicable to the construction of a variety of adventure game tools and household furniture items. This approach may therefore also be used in similar domains where materials exist on the surface of the object to be constructed so that visual similarity methods are applicable.

Since both proposed models use only visual data as inputs, they are incapable of selecting materials that need non-visual information such as knowledge about the density of materials or other chemical properties. An example is iron ore in the Minecraft domain, which is an ingredient of an iron ingot. In this case, the ingredient does not share much visual similarity with the object to be constructed. In such a situation, recipes cannot be directly inferred and additional knowledge about chemical processes is necessary. However, in the domains analysed, our intuition that visual similarity can be used to perform object construction is confirmed, provided the materials can be seen in objects.

We have tentatively shown that our model can handle multiple materials in Chapter 6. However, the combinatorial nature of the construction search algorithm means that the model needs to operate on

many inputs, $O(n^2)$, before ranking can take place. To remedy this we could use a two-level ranking process inspired by the Learning to Rank literature [Broder et al., 2003]. Namely, a coarse filter would first rank groups of materials, and then the proposed method would be used to refine the results within top ranking groups.

The proposed method is largely based on simple two layer CNN modules. We therefore interpret results as a lower bound for performance since the architecture of our model can be replaced with more sophisticated alternatives. Firstly, when dealing with 3D data, we can make use of Multiview-CNNs (MV-CNNs) to better model data from multiple angles simultaneously [Su et al., 2015]. Secondly, we could also make use of Matching Networks that have the advantage of being able to take in multiple images for our multiple material selection domain [Vinyals et al., 2016]. In particular, Matching Networks use an LSTM instead of the addition operator proposed in CraftNet to combine information from multiple images. Lastly, the architecture of each module may extract better features if more advanced CNN modules such as VGG-16 [Simonyan and Zisserman, 2014] and ResNet [He et al., 2016] are used.

While our model may need additional features to work in a real-world domain, the idea of using visual similarity to perform material selection has been shown to have promise. In particular, we have tentatively shown that our models can deal with realistic domains and can handle multiple materials. Thus, despite the limitations of our methods we have satisfied our aim making a small first step toward our long-term goal of creating an agent capable of raw material selection for the task of object construction.

References

K. Aluru, S. Tellex, J. Oberlin, and J. MacGlashan. Minecraft as an experimental world for AI in robotics. In *AAAI Fall Symposium*, 2015.

J. T. Balint and J. M. Allbeck. Macgyver virtual agents: Using ontologies and hierarchies for resourceful virtual human decision-making. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1153–1154, 2013.

G. Barth-Maron, D. Abel, J. MacGlashan, and S. Tellex. Affordances as transferable knowledge for planning agents. In *AAAI Fall Symposium*, 2014.

S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2015.

C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 426–434, 2003.

J. Bromleyh, I. Guyon, Y. LeCun, E. Sackinger, and R. Shah. Signature verification using a "Siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7 (04):669–688, 1993.

A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3D model repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

R. Collobert, S. Bengio, and J. Mariéthoz. Torch: A modular machine learning software library. Technical report, Idiap, 2002.

S. Dreyfus. The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control*, 18(4):383–385, 1973.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimisation. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

T. Fitzgerald, K. Mcgreggor, B. Akgun, A. K. Goel, and A. L. Thomaz. A visual analogy approach to source case retreival in robot learning from observation. In *Workshops at the AAAI Conference on Artificial Intelligence*, 2014.

K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

J. J. Gibson. *The Ecological Approach to Visual Perception*. Psychology Press, 1986.

X. Glorot and Y. Bengio. Understanding the difficulty of trainig deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, volume 2, pages 1735–1742, 2006.

X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3279–3286, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

G. Hinton, N. Srivastava, and K. Swersky. Overview of mini-batch gradient descent, 2016.

G. E. Hinton, P. Dayan, and M. Revow. Modeling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997.

E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92, 2015.

T. E. Horton, A. Chakraborty, and R. S. Amant. Affordances for robots: A brief survey. *AVANT. Pismo Awangardy Filozoficzno-Naukowej*, 2:70–84, 2012.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *International Conference on Machine Learning Deep Learning Workshop*, volume 2, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Cognitive Science Society*, volume 33, 2011.

Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256, 2010.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 2015.

Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Y. Li, H. Su, C. R. Qi, N. Fish, D. Cohen-Or, and L. J. Guibas. Joint embeddings of shapes and images via CNN image purification. *ACM Trans. Graph.*, 34(6):234–1, 2015.

C. Liu, L. Sharan, E. H. Adelson, and R. Rosenholtz. Exploring features in a Beyesian framework for material recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 239–246, 2010.

T. Mar, V. Tikhanoff, G. Metta, and L. Natale. Multi-model approach based on 3D functional features for tool affordance learning in robotics. *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 482–489, 2015.

K. P. Murphy. *Machine learning: A probabilistic perspective*. MIT press, 2012.

A. Myers, C. L. Teo, C. Fermüller, and Y. Aloimonos. Affordance detection of tool parts from geometric features. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 1374–1381, 2015.

A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis. Detecting object affordances with convolutional neural networks. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 2765–2770, 2016.

J. Perlow, B. Rosman, B. Hayes, and P. Ranchod. Raw material selection for object construction. In *PRASA-Robmech International Conference*, 2017.

S. Reed, Y. Zhang, Y. Zhang, and H. Lee. Deep visual analogy-making. In *Advances in Neural Information Processing Systems*, pages 1252–1260, 2015.

B. Ridge, D. Skočaj, and A. Leonardis. Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 5047–5054, 2010.

F. Rosenblatt. The perceptron: A probabalistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

A. Roy and S. Todorovic. A multi-scale CNN for affordance segmentation in rgb images. In *European Conference on Computer Vision*, pages 186–201, 2016.

D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

F. Sadeghi, C. L. Zitnick, and A. Farhadi. Visalogy: Answering visual analogy questions. In *Advances in Neural Information Processing Systems*, pages 1882–1890. Curran Associates, Inc., 2015.

A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. In *International Confernce on Machine Learning*, pages 1089–1096, 2011.

F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

L. Sharan, R. Rosenholtz, and E. Adelson. Material perception: What can you see at a brief glance? *Journal of Vision*, 9(8):784–784, 2009.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

J. Sinapov and A. Stoytchev. Detecting the functional similarities between tools using a hierarchical representation of outcomes. In *IEEE International Conference on Development and Learning*, pages 91–96, 2008.

B. M. Stafford. *Visual analogy: Consciousness as the art of connecting*. MIT Press, 2001.

H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. ICCV*, 2015.

E. Triantafillou, R. Zemel, and R. Urtasun. Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems 30*, pages 2252–2262. Curran Associates, Inc., 2017.

M. Varma and A. Zisserman. A statistical approach to texture classification from single images. *International Journal of Computer Vision*, 62(1):61–81, 2005.

O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

T. Wiatowski and H. Bölcskei. A mathememattical theory of deep convolutional neural networks for feature extraction. In *IEEE Transactions on Information Theory*, volume 64, pages 1845–1866, 2018.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833, 2014.

Appendix A

Minecraft Recipes

Table A.1: Iron items.

Item	Ingredients	
Iron Axe	Stick	Iron Ingot
Iron Sword	Stick	Iron Ingot
Iron Pickaxe	Stick	Iron Ingot
Iron Hoe	Stick	Iron Ingot
Iron Shovel	Stick	Iron Ingot
Iron Shears	Iron Ingot	
Iron Ingot	Iron Ore	

Table A.2: Diamond items.

Item	Ingredients	
Diamond Axe	Stick	Diamond
Diamond Sword	Stick	Diamond
Diamond Pickaxe	Stick	Diamond
Diamond Hoe	Stick	Diamond
Diamond Shovel	Stick	Diamond
Diamond	Diamond Ore	

Table A.3: Wood items.

Item	Ingredients	
Wood Axe	Stick	Wood Planks
Wood Sword	Stick	Wood Planks
Wood Pickaxe	Stick	Wood Planks
Wood Hoe	Stick	Wood Planks
Wood Shovel	Stick	Wood Planks
Wood Planks	Wood	

Table A.4: Stone items.

Item	Ingredients	
Stone Axe	Stick	Cobblestone
Stone Sword	Stick	Cobblestone
Stone Pickaxe	Stick	Cobblestone
Stone Hoe	Stick	Cobblestone
Stone Shovel	Stick	Cobblestone
Cobblestone	Stone	

Table A.5: Gold items.

Item	Ingredients	
Gold Axe	Stick	Gold Ingot
Gold Sword	Stick	Gold Ingot
Gold Pickaxe	Stick	Gold Ingot
Gold Hoe	Stick	Gold Ingot
Gold Shovel	Stick	Gold Ingot
Gold Ingot	Gold Ore	