CoNNOR: Convolutional Neural Network for Outsole Recognition

by

Miranda Tilton

A Creative Component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Statistics

Program of Study Committee:
Susan Vanderplas, Co-major Professor
Danica Ommen, Co-major Professor
Kris De Brabanter

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ABSTRACT

A common goal of forensic shoeprint analysis is to identify shoe models or designs that are similar to that of a given print, such as a print found at a crime scene. Quantifying similarity between shoe outsole patterns is difficult because it requires both a set of well-defined features and an accurate method to classify outsoles according to those features. A set of geometric features was developed based on common geometric shapes, such as circles and quadrilaterals. A new classifier was then trained for the pretrained convolutional neural network base of VGG16 to create a model, named CoNNOR (Convolutional Neural Network for Outsole Recognition), to automatically classify portions of outsole images into the new geometric scheme of outsole class characteristics. During the analysis of CoNNOR's performance, new diagnostic plots were developed which provide a better method to assess classification errors of multi-class, multi-label models. In general, CoNNOR performs well on images with unambiguous shapes and moderate color contrast; additional improvements may be realized by preprocessing the images to improve contrast as well as by integrating spatial relationships between geometric features. CoNNOR represents a significant improvement to the current manual classification of footwear tread patterns, facilitating new modes of data collection and automatic processing that will expand the data available for assessment of footwear class characteristic frequency in the population.

CHAPTER 1. INTRODUCTION

1.1 Motivation

In forensic science, shoeprints and outsole characteristics fall into the category of pattern evidence. When a shoeprint or impression is found at a crime scene, the investigator may ask a series of questions. Initially, it may be important to determine the make and model of the shoe, which may help detectives locate comparison shoes from suspects. Later in the investigation, the forensic examiner may consider individualizing characteristics found in the print; that is, small defects that make it possible to tie a specific shoe to the print left at the scene. In cases where such individualizing characteristics are not considered (estimated at 95% of cases in the United States according to some experts[1]), it is useful to be able to assess the *random match probability,* that is, the probability that the specific model of shoe which made the print would be found in a random member of the population's possession. This question is much more difficult than identifying the make and model of the shoe, because it requires that the forensic examiner have access to a database containing information about the frequency of shoes in the local population, where the local population itself may be difficult to define. Any tractable solution to the problem of assessing the random match probability of a shoeprint based only on class characteristics (Bodziak, 2000) (make, model, and other characteristics determined during the manufacturing process) requires a way to assemble this database: an automated solution to efficiently classify many types of shoes within a common system. This project is designed to address the computational and statistical process of assembling features which can be used to assess similarity between two or more images, with the goal of producing software which can be integrated into an system to collect and automatically identify features in images of shoe soles.

[1] Leslie Hammer, presentation to CSAFE on March 5, 2018

1.2 Outsole Class Characteristics

According to Gross et al. (2013), the four generally accepted conclusions that can be made from a footwear examination are elimination, inconclusive, class association, and identification. Typically, the ultimate goal of an examination is identification: matching a shoeprint to an individual shoe, ideally owned by the suspect. This is difficult because identification to a specific individual's shoe requires the matching of randomly acquired characteristics, which occur due to wear and damage, and that information is frequently unavailable due to the quality of the print or impression recovered from the crime scene. Thus, examiners spend most of their time considering class associations to identify one or more shoe models and sizes which are consistent with the recovered print.

Class characteristics are defined as the set of features which allow an object to be placed into a group with other physically similar objects. In the context of footwear, the term refers to the design and physical dimension of the shoe, particularly with regard to the shoe's outsole. While class characteristics are not sufficient for identification, they in many cases enable the exclusion of footwear (Bodziak, 2000).

When categorizing a shoe, it is common to use features like brand, size, and general type (e.g., boot, tennis shoe, dress shoe). Unfortunately, these features prove quite difficult to use as characteristics for identification or exclusion. Size, for example, is far from straightforward, as size standards vary significantly across different manufacturers and scales in different countries, and different shoe styles with same size inside may have different size outsoles, making direct measurement or estimation of foot size difficult (Bodziak, 2000). Identifying the model of shoe is also not trivial because manufacturers are constantly developing new models, discontinuing existing models, or reviving discontinued models. There are also look-alikes for many common models that are difficult to distinguish from the models they emulate. Thus, when defining features to describe any possible shoe outsole that may be found at a crime scene, it is important that any set of descriptors be general enough to describe a large variety of shoes and specific enough to differentiate between shoes that may have similar qualities. One such set of descriptors are geometric shapes, which have been found to be useful in differentiating between different shoes (Gross et al., 2013). Match-

ing tread patterns by comparing the spatial distribution of geometric shapes "is of considerable evidential value" (Hancock et al., 2012) for class characteristic comparisons.

1.3 Computational Image Analysis and Convolutional Neural Networks

Shoeprint evidence from crime scenes is most commonly collected in the form of a photograph, so any useful method to automatically identify outsole characteristics must take the form of an image analysis task. There are a number of methods that may be employed to identify shapes and features in an image. The Hough transform is a feature extraction technique carried out in parameter space that was classically used to identify straight lines but has been extended to identifying circles and ellipses, as well as other shapes (Ballard, 1981). There are a number of other low-level feature extraction methods aimed at detecting specific shapes, such as edges, corners, blobs, or ridges (Jain et al., 1995, Ch 15). While these methods are useful in identifying these specific features at a low level, they are very computationally intensive and only identify features on a very small scale; as a result, they cannot reliably identify large geometric shapes like those that may be found in an outsole image. These classical methods are also extremely sensitive to lighting or color changes. As one or more models would be required for each geometric shape, classical methods would require use of additional machine learning models such as random forests to aggregate low-level features into functional geometries found on outsoles.

Convolutional neural networks (CNNs) are widely recognized as superior for novel image classification. CNNs are a form of artificial neural network which make use of the image convolution operator used by many low-level feature extraction methods[2]. As CNNs have evolved, their architecture has become more complex, but the fundamental reliance on the image convolution operator sets CNNs apart from other artificial neural networks (Gu et al., 2018). CNNs have deep architectures that can be trained to identify complex patterns, but they are structurally similar to the human visual architecture and output binary or probabilistic predictions for given labels that are

[2]The Google Trends interest graph for CNNs and computer vision shows a massive increase in convolutional neural networks between mid-2014 and 2018 https://trends.google.com/trends/explore?date=2010-01-01%202019-02-18&q=convolutional%20neural%20network,computer%20vision

readily interpretable. As CNNs make use of labeled training data, the predictions generated are for features which are similar to those identified by humans, resulting in models with greater face validity. Once a CNN is trained, it is relatively fast and easy to apply the model to new images and obtain classifications.

1.3.1 General Approach

Visual classification (i.e., assigning a label to an object based on visual input) is a complex task that humans do very well (Mallot et al., 2000). Sight is our dominant sense and a significant part of our brain is dedicated to vision, which means that the structures used to impart meaning on a visual scene have been optimized through millions of years of evolution. As convolutional neural networks are organized to mimic the process of object recognition in the human visual cortex, it will be useful to briefly describe that process.

Human Vision The visual perception process begins with the transfer of information from the visual world to the brain via rods and cones in the retina. Chemical signals travel along the optic nerve from the retina into the brain, where the signals are processed by a series of biological modules which aggregate information across multiple cells and provide meaning and order on otherwise chaotic chemical and electrical signaling. As information is aggregated, spatial relationships between objects in the physical world are maintained in the brain. Specialized feature detector cells respond preferentially to specific stimuli (e.g. cells which respond to lines oriented horizontally, vertically, or at specific angles), and these feature detectors are aggregated to identify more complex stimuli (Goldstein and Brockmole, 2016, Ch. 4). In addition to the successive compilation of increasingly complex features, there are also specific modules for particularly important tasks, such as facial recognition; information from these regions is also integrated into the overall hierarchy of recognized objects from the visual input.

The problem of general object recognition is quite difficult—a three-dimensional object has infinitely many projections into two-dimensional space, and in real scenes objects are often at least partially obscured. In addition, a two-dimensional image can map back to many different three-

dimensional objects, because of the ambiguity introduced by the projection onto a flat surface. Several psychological theories exist as to how object recognition occurs within the brain (gestalt heuristics, recognition-by-components, and inferential contexts all have experimental support), but in general the process seems to require both spatial integration and learned associations (Goldstein and Brockmole, 2016, Ch. 5).

Differentiating between two objects is quite easy when features are distinct; however, there are many cases when differentiating features are rather subtle. For example, as shown in Figure 1.1, an orange caterpillar and a carrot may be of similar color, shape, and size, but one is more fuzzy than the other; remarkably, the distinction between the two categories is very strong even with all of the features that are shared. Thus, our brains have learned that when faced with a small, cylindrical orange object, texture is a critically important feature when assigning a label to that object (which keeps us from accidentally ingesting caterpillars).

Figure 1.1 A fuzzy caterpillar and a bunch of carrots have many similar visual features, but our brains easily distinguish between them.

Computer Vision Using Neural Networks While our brains are adept at parsing images and classifying the objects within them, the task has proved much more difficult for computers, as evidenced by Figure 1.2. Human visual processing is so complex in part because of the successive aggregation of increasingly complex features; only within the last 10 years have we been able to adequately mimic this process with computer modeling.

Convolutional neural networks are a widely implemented method for automated image recognition; their structure typically emulates the successive aggregation of low-level features into higher-

Figure 1.2 Computer vision was thought to be easy in 1966 when a researcher at MIT believed that teaching a computer to separate picture regions into objects and background regions could be completed as a summer project (Papert, 1966). The task proved much more difficult than expected, and has only become tractable with convolutional neural network based approaches. Image source: https://xkcd.com/1425/.

level features that is seen in the human visual cortex. CNNs perform comparably to humans on certain image recognition tasks (Geirhos et al., 2017). The ImageNet Large Scale Visual Recognition Competition (ILSVRC) is a widely followed contest to produce the best algorithm for image classification; since 2014, it has been dominated by convolutional neural networks (Russakovsky et al., 2015). Various models are tested on about 1.2 million images spanning 1000 categories, which are part of the ImageNet image dataset (Deng et al., 2009). These categories range from natural and man-made objects (e.g., daisy, chainsaw) to living creatures (e.g., ring-tailed lemur, sea lion, and dingo). There are also many categories which require subtle distinctions, such as golden retrievers and labrador retrievers, as shown in Figure 1.3.

Figure 1.3 The features used to distinguish between two similar categories may be subtle, like the features that would differentiate a golden retriever from a labrador retriever (Images from Deng et al., 2009).

1.3.2 Building Blocks of a Convolutional Neural Network

CNNs are made up of several distinct types of layers which transition from input image to output class probabilities; the remainder of this section discusses several important layer types which are used in most CNNs.

Image Convolution and Convolutional Layers Convolutional neural networks make use of convolutional layers, which use image convolution as a primary operation. Defined mathematically, image convolution is an function performed on an image x using a smaller-dimension matrix β.

Let x be an image represented as a numerical matrix, indexed by i, j, and β be a filter of dimension $(2a+1) \times (2b+1)$. The convolution of image x and filter β is

$$(\beta * x)(i,j) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \beta(s,t) x(i-s, j-t) \tag{1.1}$$

Convolutional neural networks are named to highlight their use of image convolution operations to extract information from an image. As shown in Figure 1.4, a single convolutional filter is a small array of real valued weights that represents some feature (shown in green). When a filter is applied to a portion of the image (shown in blue), a single value is returned that is associated with the presence of the feature for a given subsection of the input image. When applied over an entire image, the resulting matrix of values maps the strength of the feature across the entire image, as shown in Figure 1.5.

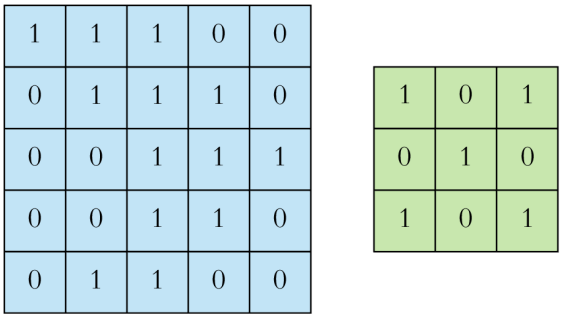

Figure 1.4 An image (blue) and a convolutional filter (green). Image from Prabhu (2018).

Once the entire image has been convolved with the filter, the resulting feature map is transformed using a nonlinear activation function like those found in Figure 1.7. A convolutional layer of a CNN takes a large number of these filters and passes them over the image to return one activation layer per filter.

Convolutional layers typically do not decrease the dimensions of the matrix by a significant amount, as the filter β is typically small: 3×3 or 5×5. Many models pad the input image in order to prevent any reduction in dimension. In these neural networks, dimension reduction is generally performed by pooling layers, which identify the strongest local features in each filter layer of the convolutional output.

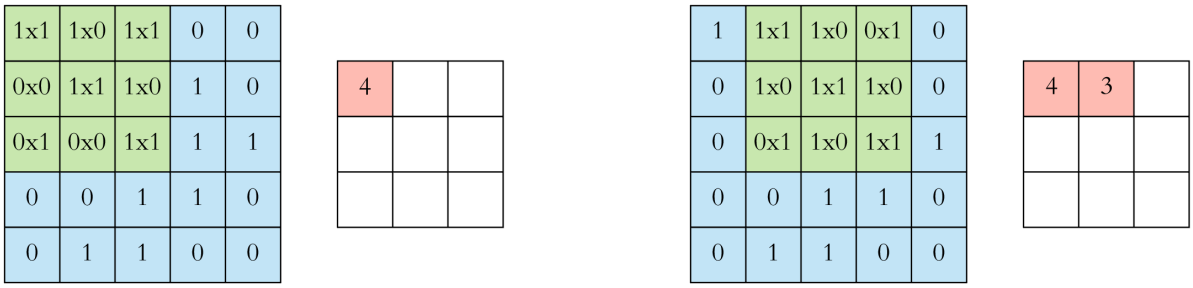

Figure 1.5 The convolution operation consists of the smaller filter (green) applied to each region of the larger image (blue); each application results in a single value which is stored in the feature map. Image from Prabhu (2018).

Pooling Layers Pooling layers are added to reduce the size, and therefore computational load, of feature maps through structured down-sampling. Most commonly, pooling layers apply the maximum function (max pooling) over adjacent regions of a feature map (using a sliding window). This encodes the maximum strength of a feature in a region of an image, while reducing redundant or unnecessary information about less prominent activations.

In most cases, the stride, or offset between subsequent pieces, is the same as the window size (non-overlapping pooling); for this simplified case, the pooling layer values can be calculated using the following relationship, for pooling function f, window size s, layer $\ell - 1$, and output layer ℓ:

$$x_{ij}^{\ell} = f\left(x_{(i-1)s+1 \le y \le is, (j-1)s+1 \le z \le js}^{\ell-1}\right) \tag{1.2}$$

As a general matrix notation for pooling is difficult to specify intuitively, we will define a pooling operator, p, which applies Equation 1.2 element-wise.

$$p\left(x, f, w, s\right) := \text{matrix-wise pooling on matrix } x \text{ with function } f, \tag{1.3}$$
$$\text{with window } w, \text{ and stride } s$$

For example, taking 2x2 pieces of a feature map and keeping only the largest of the four values reduces the size of the feature map by a factor of 4, as shown in Figure 1.6. Krizhevsky et al. (2012) suggests pooling layers reduce overfitting on certain datasets and increase translation invariance. Using too large of a pooling window can be destructive and limits the total number of convolutional layers which can be combined in a single network (CS2). Pooling layers do disrupt the

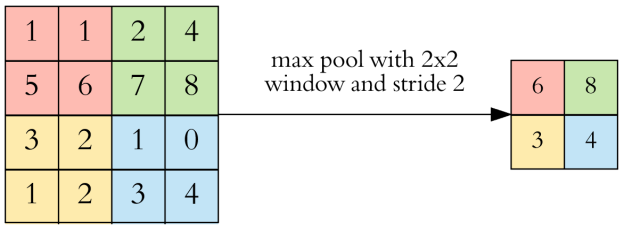

Figure 1.6 Max pooling layer. Image from Prabhu (2018).

model optimization process; as an alternative, some fully convolutional networks eliminate pooling layers and use convolutional layers with increased stride and window size to reduce layer dimensions (Springenberg et al., 2014).

Once the dimension of the image has been reduced and features have been identified using a combination of convolutional and pooling layers, local features must be integrated into a more unified whole. This integration is performed using densely connected layers.

Activation Functions The convolution operation discussed above relies on nonlinear activation functions, which operate on each feature map value. Different activation functions are used to produce different network effects—in the convolutional layers, the desire is often to minimize the computational complexity, so very simple activation functions are used, such as the Rectified Linear Unit (ReLU), Exponential Linear Unit (ELU), or a differentiable analog, SoftPlus. The output layer must map features onto binary predictions or probabilities, so the sigmoid activation function is commonly used for this purpose. Figure 1.7 shows several common nonlinear activation functions.

Activation functions are also used in densely connected layers, which are discussed in the next section. Frequently, the same activation function is used throughout a CNN, though it is (mathematically) possible to use a different activation function for each layer.

Densely Connected Layers Densely connected layers are typically the final layers in a CNN, making up what is known as the model head. These layers form the meaningful connection between the features of an image (detected by convolutional and max pooling layers) and the corresponding

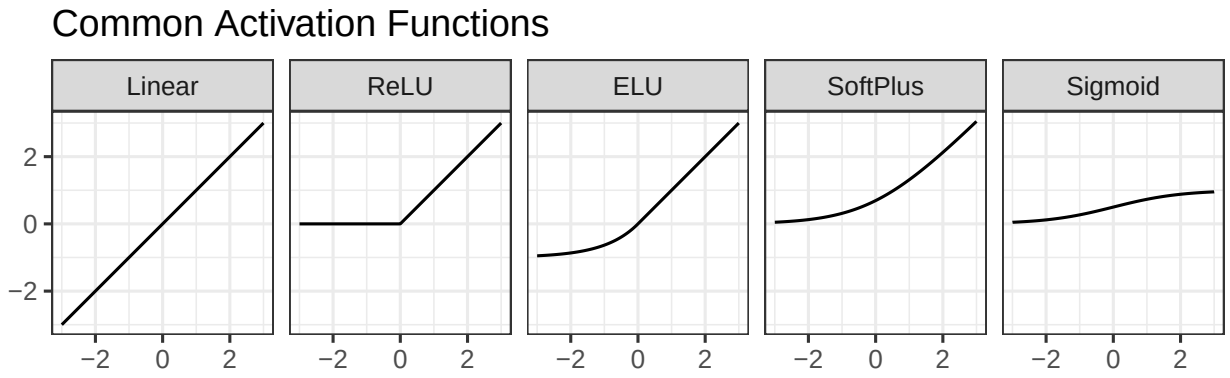

Figure 1.7 Common nonlinear activation functions used in neural networks.

labels associated with the image. Just as humans learn which combinations of features should be associated with a given label, densely connected layers use real-valued weights to represent these associations. For example, an item which is orange, small, and fuzzy is commonly associated with the word "caterpillar". As seen in Figure 1.8, fuzzy is not a feature typically associated with the word carrot, so there is little connection between the feature "fuzzy" and the label "carrot".

Similarly, in densely connected or fully connected layers, each final feature produced by the convolutional and pooling layers is connected to each possible label through weights (hence the name "densely connected") learned during the training process. Weights are optimized via back-propagation (discussed in Section 1.3.3) in order to minimize errors (measured by a cost function) and improve classification accuracy.

When training fully connected layers there is a danger that co-dependence will develop between nodes, which lessens the power of each individual node and usually leads to over-fitting. To prevent this, fully connected layers are often trained by utilizing a pruning mechanism known as drop-out, where at any given stage each node is kept in the model with probability p or temporarily disabled with probability $1 - p$ and training is performed on the reduced model. Densely connected layers with and without dropout nodes are shown in Figure 1.9.

Output Classification Layer In order to transform the neural network node values into output class probabilities, the final layer of the model uses an activation function selected to conform

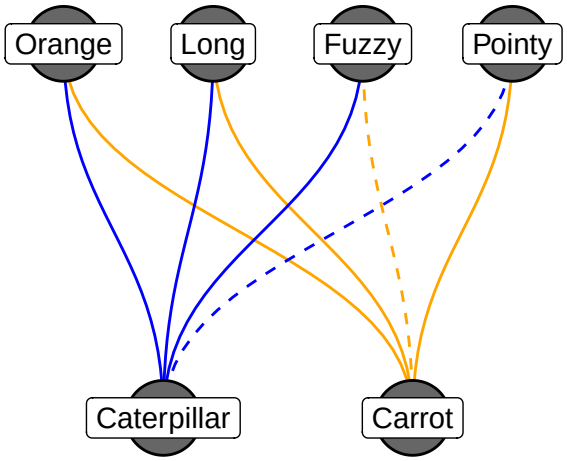

Figure 1.8 A representation of how different features are connected to labels for classification. Note that the feature "fuzzy" is connected to the notion of caterpillar but not to carrot, and "pointy" is only related to carrots.

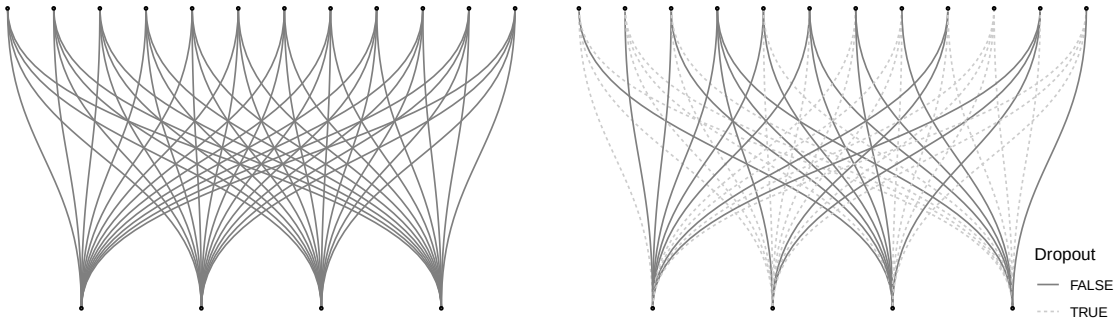

Figure 1.9 (Left) A densely-connected layer with 12 input nodes and 4 output nodes. (Right) A densely-connected layer with 12 input nodes, 4 output nodes, and a 50% dropout rate.

to the problem specifications. For instance, if the goal is to perform binary classification, the sigmoid activation function shown in Figure 1.7 will map any real valued number to a value between 0 and 1 (that is, to a class probability). In a multinomial classification problem, an extension of the sigmoid activation function, the *softmax* activation function, is used: For inputs y_i, $S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$. The softmax activation function produces class probabilities which sum to 1. In classification problems where there are n classes, but each object can have between 0 and n assigned labels (i.e., a multi-class, multi-label problem), the sigmoid activation function can be used for each class label separately, producing a n-dimensional vector of probabilities between 0 and 1.

1.3.3 Forward and Backward Propagation

Forward Propagation In order to evaluate an input image, it is necessary to move from the image representation through each of the layers in the network, with a final result of a set of n output class probabilities.

We first define some notation. Let $x^{(0)}$ be an input image, represented as a numerical matrix with two dimensions of length and height, and additionally a third dimension representing the color channels, if the input image is in color. Let x^ℓ be the layers in the network, that is, x^1 is the first (convolutional) layer, x^2 is the second, and so on. Convolutional layers are assembled from a set of filters, β_k^ℓ, where there are a set of p^ℓ $m \times m$ filters convolved with $x^{\ell-1}$ to create layer x^ℓ. Each convolutional layer also has a bias matrix γ^ℓ which is used in the calculation of all filters in the ℓ-th layer. We additionally define $\sigma^\ell(\cdot)$ as the nonlinear activation function used in layer ℓ (some common nonlinear activation functions are shown in Figure 1.7). Finally, we define W^ℓ to be a weight matrix used in fully connected layer ℓ; W's dimensions are chosen such that the output dimension is equal to the prespecified number of output classes.

During forward propagation, the calculation of the ℓth layer uses the $\ell-1$th layer in an iterative process:

$$\begin{aligned} x_k^{(\ell)} &= \sigma^\ell\left(\beta_k^\ell * x^{(\ell-1)} + \gamma^\ell\right) \text{ for convolution} \\ x_k^{(\ell)} &= p\left(x^{(\ell-1)}, \max, s, s\right) \text{ for max pooling layers} \\ x_k^{(\ell)} &= \sigma^\ell\left(W x^{(\ell-1)}\right) \text{ for densely connected layers} \end{aligned} \tag{1.4}$$

Loss and Cost Functions In a multi-label classification task, the standard loss function used is binary cross-entropy loss. For a single input image $x^{(0)}$ with true labels $\mathbf{y} \in \{0,1\}^C$ and prediction $\hat{\mathbf{p}} \in [0,1]^C$, where C is the number of output classes,

$$L(\mathbf{y}, \hat{\mathbf{p}}) = \sum_{i=1}^{C} -\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] \tag{1.5}$$

The cost function is defined as the average of the loss function over all training images.

Backpropagation Backpropagation, short for "the backward propagation of errors", is the name of the optimization algorithm that is used to train deep neural networks. In the process, the error of each output is calculated and then distributed backwards through the network's layers; weights are updated in order to reduce the errors for the next iteration of the algorithm. In essence, the goal of backpropagation is to use gradient descent to adjust the parameters of the network to achieve a local minimum of the cost function, where the gradient is computed through repeated application of the chain rule.

For convolutional layers, backpropagation works using the recurrence relationship in Equation 1.6.

$$\begin{aligned} \left(\frac{\partial L}{\partial \beta_k^\ell}\right) &= \underbrace{\frac{\partial L}{\partial\left(\beta_k^\ell * x^{\ell-1}\right)}}_{\text{gradient}} x^{\ell-1} \\ \frac{\partial L}{\partial\left(\beta_k^\ell * x^{\ell-1}\right)} &= \frac{\partial L}{\partial x^\ell}\left[\sigma'\left(\beta_k^\ell * x^{\ell-1}\right)\right] \end{aligned} \tag{1.6}$$

There is no backpropagation through pooling layers because there are no weights to optimize, so these layers are just pass-through layers. Backpropagation through fully connected layers takes place similar to Equation 1.6, shown in Equation 1.7.

$$\left(\frac{\partial L}{\partial W^{\ell}}\right) = \underbrace{\frac{\partial L}{\partial \left(W^{\ell} x^{\ell-1}\right)}}_{\text{gradient}} x^{\ell-1} \tag{1.7}$$

$$\frac{\partial L}{\partial \left(W^{\ell} x^{\ell-1}\right)} = \frac{\partial L}{\partial x^{\ell}} \left[\sigma'\left(W^{\ell} x^{\ell-1}\right)\right]$$

1.3.4 Transfer Learning

Convolutional layers and max pooling layers in a CNN are analogous to the human visual perception process, and densely connected layers behave like the human brain. In short, the approach to classifying an image is to detect the features in the image, like our eyes do, and then assign labels to combinations of those features, like our brains do. This analogy is also appropriate because it reflects the difficulty of the task: it takes many years and a significant amount of effort for humans to learn how to distinguish a large variety of features and also to connect those features to labels that are often complex, hierarchical, and subtle. Similarly, training a CNN is no small task. Even relatively simple CNNs can have many millions of trainable parameters in the convolutional and densely connected layers. Optimizing all of these weights requires an incredible amount of computational power. In addition, the features learned by a neural network trained on one dataset often generalize to different data sets: particularly in the initial layers, the feature maps of a trained neural network typically activate based on color, low-level textures, and other features which are broadly generalizable (Yosinski et al., 2014). Although later layers detect more complex and specific features, there are usually a larger number of these filters, so the huge number of possible combinations of many specific filters creates a model base that is generalizable to new image classification tasks. Examples of the filters found at several different layers of a trained network are shown in Subsection 2.2.2; it is clear that the layers shown in Figure 2.9 are more complex than Figure 2.5, but that combinations of filters from Figure 2.9 are able to detect a wide variety of features that the original model may not have been explicitly trained to recognize.

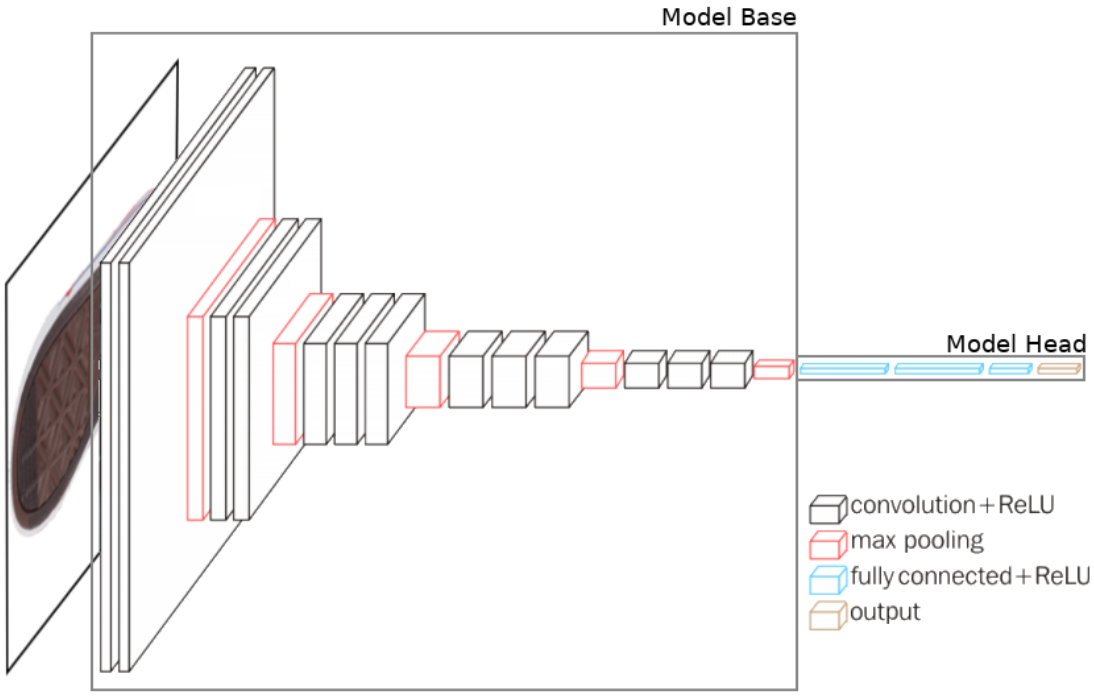

Figure 1.10 This diagram is for a pre-trained convolutional neural network. The model base consists of 5 convolutional blocks; the model head consists of several fully connected layers capped with an activation layer which transforms the aggregate visual input to output class probabilities. During transfer learning, the model base weights are fixed to the values derived from the initial input material used to train the original model; only the weights in the model head are retrained to accommodate the input training data.

Transfer learning is the process of using layers from a CNN (or other classification model) trained on a general image recognition task when fitting a model intended for a more specific purpose. The layers which are deemed to identify broadly generalizable patterns are used with their pre-trained weights; new layers are added to customize the model to the specific task at hand, and typically, only these new weights are updated when the model is fit. In many cases, the entire model base is used, and fitted with a new model head; the modularity of CNNs makes this process relatively simple, and allows researchers to leverage pre-trained networks when working with different sets of image data. Transfer learning allows CNNs to be applied to smaller datasets of several thousand images and also reduces the amount of computational time required to fit the model.

Transfer learning leverages the modularity of neural networks—the pretrained base of the model can be separated from the full model and a new model head can be trained to connect that base to the output classes, as shown in Figure 1.10.

1.4 Machine Learning Model Evaluation

Classification tasks are considered single-class when there is one decision between two mutually exclusive classes, and multi-class when there are more than two classes that an object may belong to. While the true classification is a binary decision, it is common for models to predict these classifications using probability, thus reporting a value between 0 and 1 reflecting how certainly the item can be attributed to a given class. In multi-class problems, the true classification and output probabilities are represented as vectors, with length corresponding to the number of classes. When the classes are mutually exclusive, the output probabilities sum to 1.

Multi-label classification is a special case of multi-class classification problems where categories are not mutually exclusive, that is, an item may fall into a combination of categories simultaneously. In this case, the model output is a vector of probabilities which are each between 0 and 1, and output probabilities do not sum to 1.

Evaluating model accuracy in classification problems requires addressing both the labels that are assigned and the labels that are not. Ideally, an image is labeled perfectly, and a true positive occurs when the model assigns a label which matches that of the image. A false positive in this scenario occurs when the model assigns a label which does not match that of the image. A true negative occurs if the model does not assign a label which does not occur in the image, and a false negative occurs when the model does not assign a label which does occur in the image. Table 1.1 illustrates these terms for a simple binary classification problem.

Recall, or *sensitivity*, which is the true positive rate, is the number of all true positives divided by the number of positive cases in the data. *Specificity*, which is the true negative rate, is the number of all true negatives divided by the number of negative cases in the data. *Precision* is the sum of all true positives divided by the number of positive predictions made by the model.

		Model—Assigned Label	
		A	Not A
True Label	A	True Positive	False Negative
	Not A	False Positive	True Negative

Table 1.1 Model errors for a two-class binary decision problem. Correct decisions are shown along the diagonal; incorrect decisions are in the off-diagonal cells.

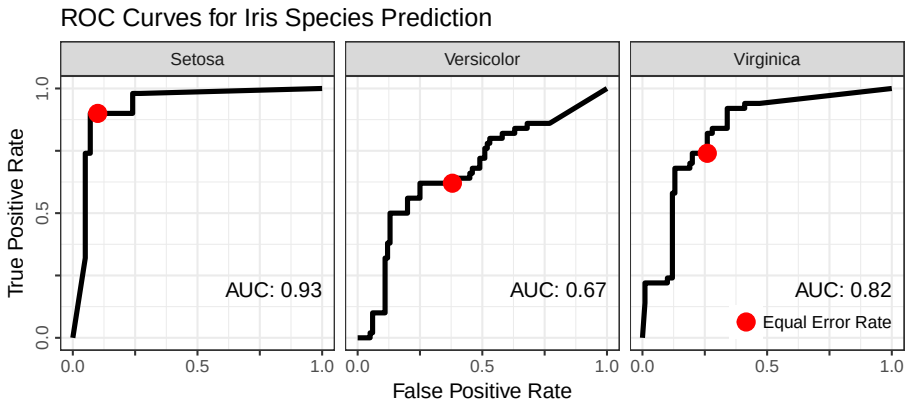

Figure 1.11 ROC curves, showing the performance of a random forest to predict iris species from sepal length from Fisher's Iris Data. Note that prediction is very accurate for species Setosa, but is not much better than random chance for species Versicolor.

		Model - Assigned Label		
		A	B	C
True Label	A	True Positive (A)	False Positive (B) False Negative (A)	False Positive (C) False Negative (A)
	B	False Positive (A) False Negative (B)	True Positive (B)	False Positive (C) False Negative (B)
	C	False Positive (A) False Negative (C)	False Positive (B) False Negative (C)	True Positive (C)

Table 1.2 Model errors for a three-class binary decision problem. Correct decisions are shown along the diagonal; incorrect decisions are in the off-diagonal cells. Note that for each misclassification, two incorrect decisions are made—the omission of the correct label and the addition of an incorrect label.

ROC curves plot the false positive rate (1 - specificity) against the true positive rate. Ideally, a model will have a high true positive rate and a low false positive rate, so a perfect ROC curve would hug the top-left corner of the graph; a straight line between corners would indicate that the prediction is no better than random chance. The Area Under the Curve (AUC) quantifies the shape of the ROC curve, with an AUC of 1 corresponding to perfect prediction and 0.5 indicating random chance. Examples of ROC curves are shown in Figure 1.11 for a random forest model to predict species for Fisher's Iris Data from sepal length. Typically, ROC curves are used for single-label, or binary, classification; Figure 1.11 follows this convention by showing a curve for each species being predicted. ROC curves have been extended to multi-class problems, but in most multi-class extension methods (Hand and Till, 2001), it is not easy to see how the model performs for each class; this issue is exacerbated when classes are unbalanced.

A confusion matrix is a cross-tabulation between the observed and the predicted classes of the data. A confusion matrix can help identify which classes are being correctly predicted and which classes are commonly mixed-up with others. Confusion matrices are typically used with binary classification problems; an aggregate version of Table 1.1 would be a confusion matrix, where totals for each decision would be shown in the matrix cells. Confusion matrices do not extend easily to multi-class problems because any single miscategorization produces both a false negative decision and a false positive decision; it is not clear which should be shown in each cell, as shown in Table 1.2.

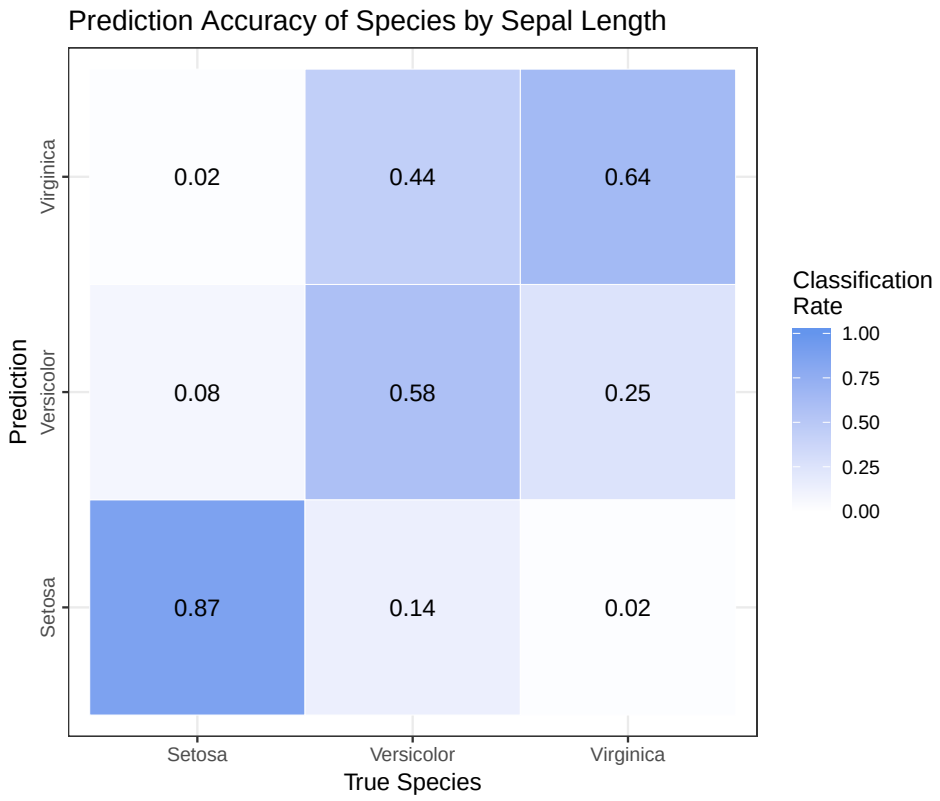

Figure 1.12 Confusion matrix, showing the performance of a random forest to predict iris species from sepal length from Fisher's Iris Data. Again, it is clear that prediction is very accurate for species Setosa, but the model is not always able to distinguish between the species Versicolor and Virginica.

However, in some situations, it is possible to reduce the classification problem to a series of binary classifications; in these situations, a modification of the confusion matrix is possible to better visualize model performance, as described in Section 3.2.

CHAPTER 2. DATA AND METHODS

2.1 Data

2.1.1 Geometric Class Characteristics

Class characteristics, as defined in Section 1.2, are characteristics which can be used to exclude shoes from a match at a crime scene, but cannot be used for individualized matching because they are shared by many shoes. A sufficiently well-defined set of features can separate shoes into make and model categories (Gross et al., 2013). Gross et al. (2013) define geometric features such as circle/oval, crepe, herringbone, hexagon, parallel lines, logo/lettering/numbering, perimeter lugs, star, and other. Working from these categories, a set of categories were assembled that is more suited to recognition by convolutional neural networks, as some of the definitions used in Gross et al. (2013) require spatial context which is not preserved during labeling (for example, lugs are required to be on the perimeter of the shoe). Table 2.1 shows three examples of each class.

Bowtie Bowtie shapes are roughly quadrilateral, with two opposite concave faces. The remaining two faces can be convex or straight, and the concave faces may have straight portions, so long as there is a concave region. Using this definition, shapes such as butterflies are included as bowties.

Chevron Chevron shapes include repeating parallel lines as well as individual "v" shapes. They may be angular but can also be curved.

Circle Circles include ellipses and ovals; they must be round.

Line Lines are repeated and parallel; a more general definition of a line would be difficult to differentiate from many other patterns. Lines can be mildly curved.

Polygon Polygons are defined in this standard to have more than 4 sides. They include pentagons, hexagons, and octagons.

Quadrilateral Quadrilaterals (quads) have four sides. They may have rounded or square corners.

Star Stars are any shape with alternating concave and convex regions, or lines which emanate from a central point. "X" and "+" shapes are also classified as stars.

Text Text is any shape which would be identified as text by a reasonable human. In most cases, the text on the outsole images used is made up of Latin alphabet characters; the model will likely not recognize text in other scripts (but could be trained if non-Latin text images could be obtained).

Triangle Triangles are any three-sided figure. Like quadrilaterals, they can have rounded corners. In some cases, it is difficult to distinguish between a trapezoidal shape and a triangle when rounded corners are involved.

Other Other features which were marked include logos, various textures (including crepe, stippling, etc.), and smooth regions with no discernible features. These regions are grouped and provide additional information—that none of the previous nine categories are present.

Defining categories this way does not remove all ambiguities. The best example lies in considering text. The letter "v" can easily be considered a chevron, and the letter "o" is clearly a circle. However, text is also an important category to encompass the variety of ways text appears on footwear outsoles, and it is not necessarily helpful (or possible) to try to categorize every shape in text into another category. Many of the ambiguities that arise can be solved by applying multiple labels to an image, but some shapes also do not fit into any categories. Applying comprehensive and consistent labels to difficult or ambiguous shapes is the most difficult part of this process. Figure 2.1 shows one shoe that has a tread pattern which does not easily fit into the predefined categories. Generally, the majority of this shoe is labeled "other".

Table 2.1 A set of geometric elements used to classify tread patterns. Categories modified from Gross et al. (2013).

Figure 2.1 A shoe which has elements which are not easily classified; the lines in the design do not meet the specified definition of lines (which must be roughly parallel). The rectangle in the center of the shoe can be correctly classified.

2.1.2 Data Collection

Thousands of outsole images were scraped online shoe retail sites. These images were then uploaded for use in a tool called LabelMe (Russell et al., 2008), a labeling/annotating interface which allows users to easily select and label regions of an image. To date, 4371 shoes have been labeled, yielding 27135 multi-label images.

After annotation using the LabelMe software package, images are processed by an R script that identifies the minimum bounding rectangle of the region, crops the image to that region, and scales the cropped area to a 256 x 256 pixel image suitable for analysis by the convolutional neural network. During this process, aspect ratio is not preserved, though efforts are made to label regions which are relatively square to minimize the effect of this distortion.

2.1.3 Data Characteristics

Figure 2.2 shows the relative frequency of each class in the labeled data. Quadrilaterals are the most frequent shape, followed by line and text; the least frequent shapes are bowtie and star. The large class imbalance must be accounted for in both model training and evaluation. Since backpropogation considers each training image equally when updating model weights, training with unbalanced data incentivizes the cost function to prioritize accuracy of the most common classes at the expense of the rarer classes. To mitigate this effect, the weight of each class was set during training with respect to its frequency in the data to ensure that training optimized performance across all classes evenly. Relevant model diagnostics (e.g., ROC curves) were also adjusted to account for class imbalance, most notably by breaking out measures for each class rather than relying on summary metrics that could obfuscate large errors in predicting small classes.

2.1.4 Augmentation

Labeled images are scarce relative to the amount of data necessary to train a neural network. One solution to this scarcity is to artificially enlarge the data set using a process called image augmentation (Krizhevsky et al., 2012). Augmentation is the transformation of original input data

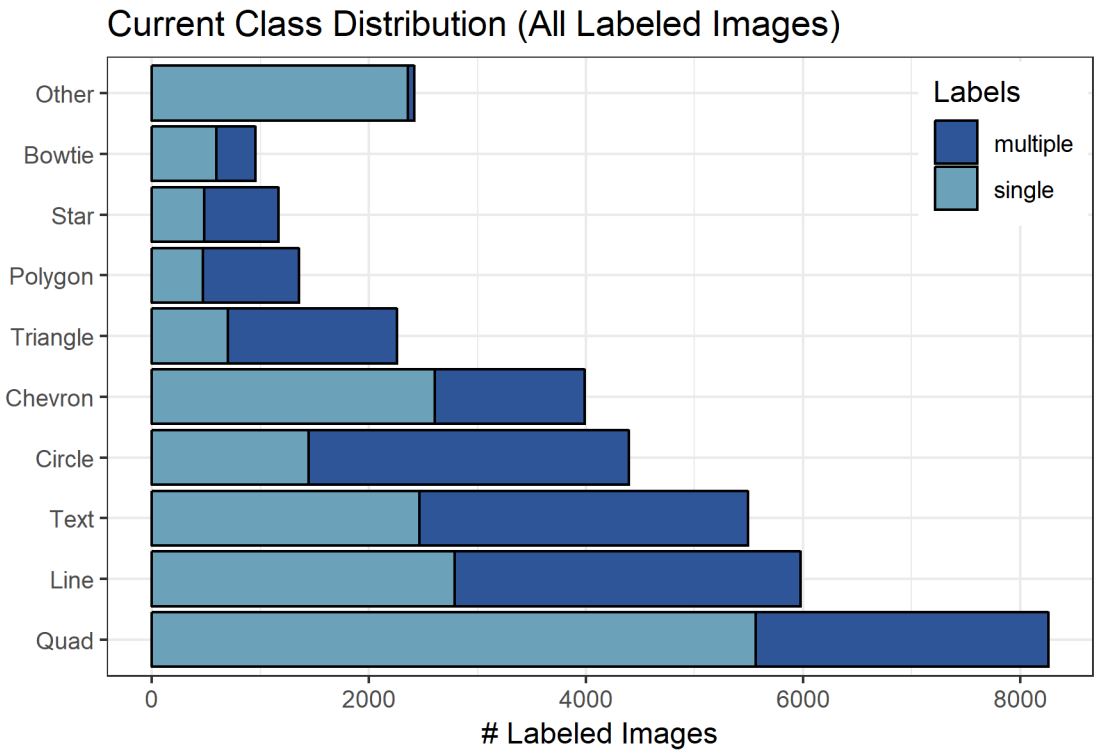

Figure 2.2 Distribution of classes in all labeled images. Quadrilaterals, lines, circles, text, and chevrons are relatively common; stars, polygons, and bowties are relatively rare.

Figure 2.3 Four sets of original (left) and augmented (right) labeled images.

using image operations such as cropping, zoom, skew, rotation, and color balance modification in order to distort or alter the image while maintaining the essential features corresponding to the label. This process reduces the potential for overfitting the model to the specific set of image data used during the training process, and also increases the amount of data available for training. During the model fitting process, images are augmented once using a subset of the augmentation operations discussed above; examples of pre- and post-augmentation images are shown in Figure 2.3.

2.2 VGG16

2.2.1 Architecture

Developed by Oxford's Visual Graphics Group, VGG16 is a CNN with 16 "functional" (i.e., convolutional and densely connected) layers and 5 "structural" max pooling layers, as shown in Figure 2.4. In contrast to other popular networks, like ResNet, VGG has a relatively simple structure that provides easier training and interpretability with very little sacrificed accuracy (the structure is shown in Figure 2.4). The simplicity of this structure provides the ability to peer into the inner workings of the network for diagnostic purposes, providing a distinct advantage over more complicated network structures with slightly higher accuracy ratings.

VGG16 and a similar network, VGG19, won the 2014 ILSVR challenge; they have since been surpassed in performance by networks with more complicated structures, such as GoogLeNet/Inception (Szegedy et al., 2015) and ResNet (He et al., 2015). VGG-style networks are still commonly used as building blocks for other types of convolutional networks; their streamlined structure allows for easy extension to other tasks, such as texture detection and style transfer (Gatys et al., 2016).

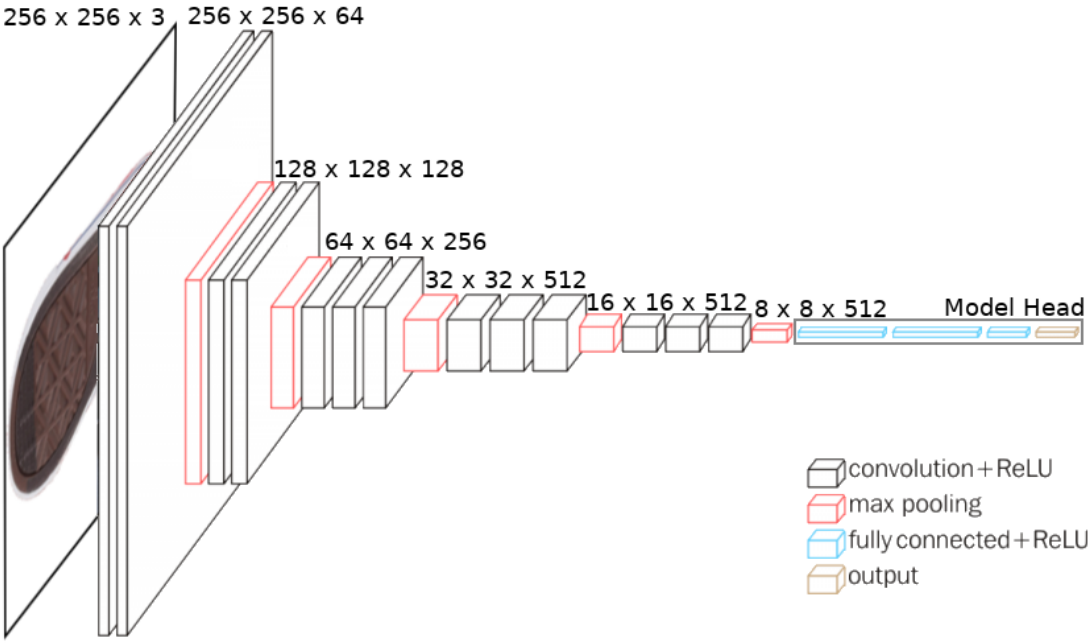

Figure 2.4 The VGG16 model structure, for input images of 256 x 256 pixels. There are 5 convolutional blocks in the VGG16 model base. Each block is followed by a max pooling layer, which reduces the size of the input to the next block by a factor of 4. The output of the final pooled block is then connected to output classes using several fully connected layers.

2.2.2 Convolutional Filters

Much like the specialized feature detector cells contained within the human visual system for detecting specific angles, colors, and shapes, convolutional filters are designed to detect a wide array of features. The early convolutional layers of VGG16 contain 64 filters that primarily detect colors and edge patterns, as shown by the activation maps in Figure 2.5 and Figure 2.6. Later convolutional layers detect increasingly complex features; Figure 2.7 shows a selection of activation maps from convolutional layers in block 3 of the model, and Figure 2.8 shows a selection from block 4 of the model. Note that some of the filters appear to detect more than one "feature"; some filters are multi-purpose. The final convolutional layers of VGG16, in contrast, contain 512 filters that represent much more complex features, like animal fur patterns or distinct bird heads.

In order to visualize the layers in a convolutional neural network, backpropagation can be used to generate an image which maximally activates a particular filter. This process generates so-called "activation maps", which provide a visual reference for the features identified by a particular layer. The activation maps shown in this section are generated using the `KerasVis` R package, which makes functions from the `keras-vis` python library (Kotikalapudi and contributors, 2017) available in R.

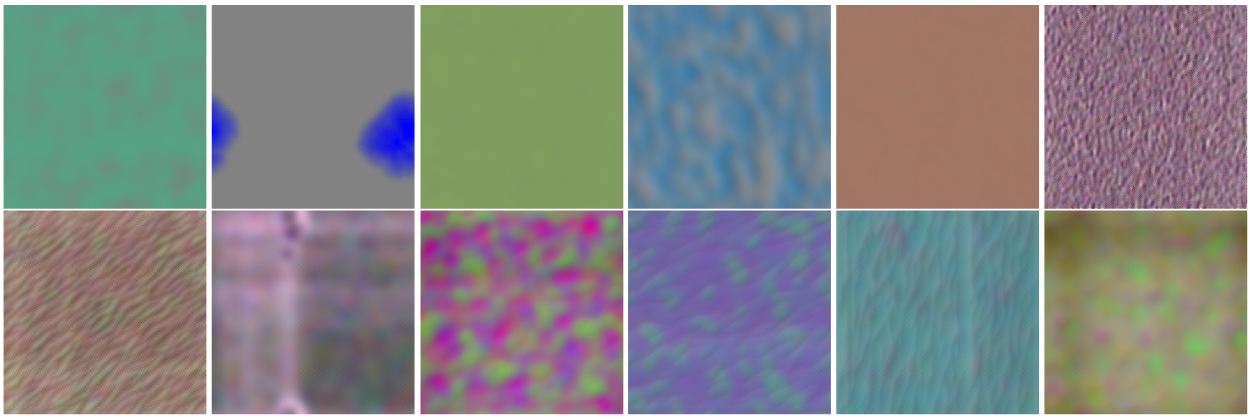

Figure 2.5 A selection of filters from block 1 of VGG16. Block 1 contains two convolutional layers, each with 64 filters.

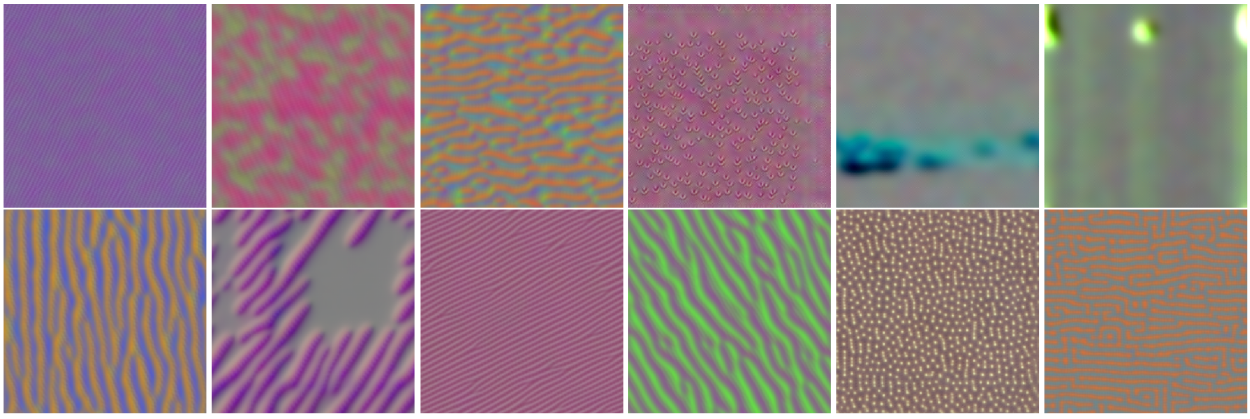

Figure 2.6 A selection of filters from block 2 of VGG16. Block 2 contains two convolutional layers, each with 128 filters.

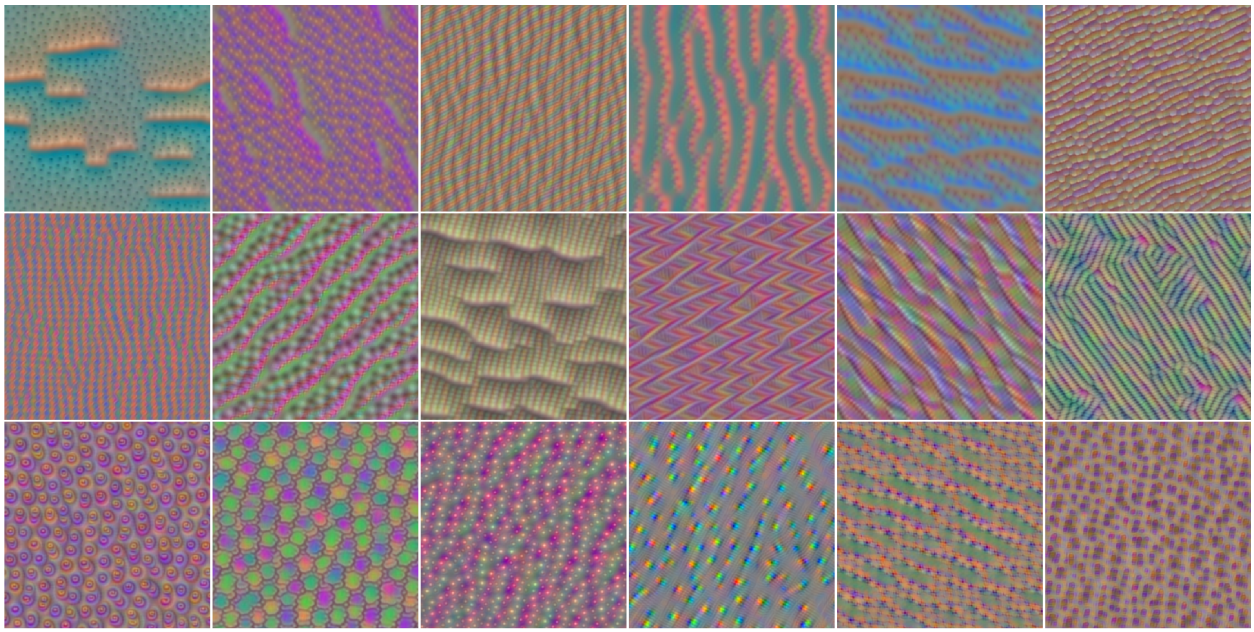

Figure 2.7 A selection of filters from block 3 of VGG16. Block 3 contains three convolutional layers, each with 256 filters.

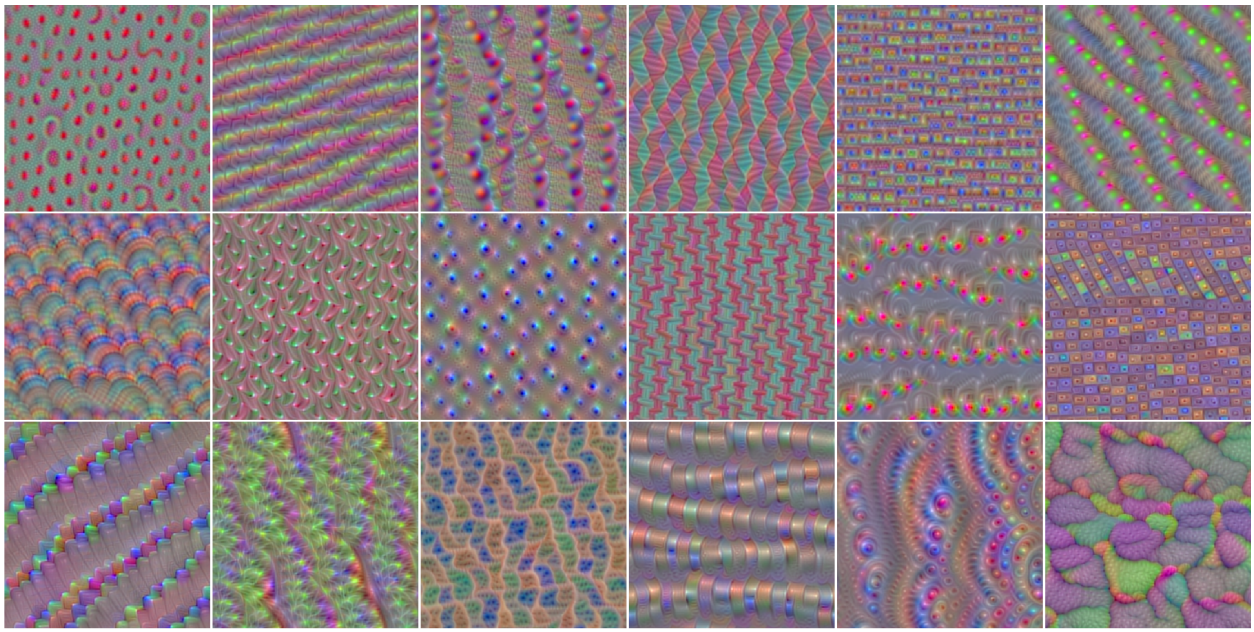

Figure 2.8 A selection of filters from block 4 of VGG16. Block 4 contains three convolutional layers, each with 512 filters.

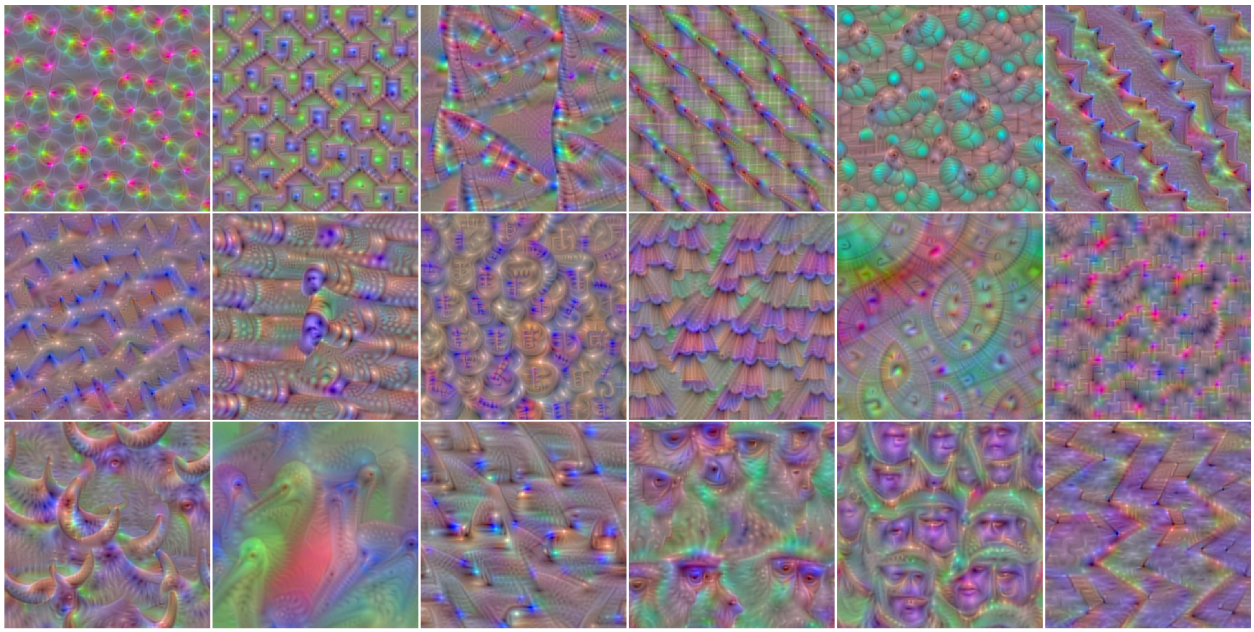

Figure 2.9 A selection of filters from block 5 of VGG16. Block 5 contains three convolutional layers, each with 512 filters.

2.2.3 Model Training

Computation Model training was conducted using the `keras` package in R (Allaire and Chollet, 2018), which provides an interface to the Python keras API. The keras API uses a TensorFlow (Abadi et al., 2015) back end. The model was trained using CPUs; the amount of memory required in order to train the model using the GPU was prohibitive.

Initial training utilized the output from the VGG16 convolutional base as input to a new model consisting of a densely connected layer with 50% dropout, followed by an activation layer with a sigmoid activation function (see Figure 1.7) and 9 output classes corresponding to the 9 geometric features that have been identified. Once the separate model head was fit, the convolutional base of VGG16 and the newly trained model head were combined into a single, unified model object for prediction.

Code is provided in Appendix A. Using a 48-thread CPU with 128 GB of RAM, the time required to process the LabelMe annotations, generate 256 $\times$ 256 pixel labeled images, augment

these images, and train the model is just under 3 hours; the model fitting process itself takes less than one hour.

Model Training Parameters The 27135 images were split such that 60% were used for training. Since the categories do not exist in equal proportion in the labeled data, the training data were weighted by proportion during the training process to prevent the loss function from being overwhelmed by more frequent categories. Of the remaining 40% of data, half were used for validation, to monitor the training process, and the remaining data were for testing the performance of the fitted model. Code to calculate the class weights is provided as part of Appendix A, but a static example of the weights and image counts is shown in Table 2.2.

Class	Count	Class Proportion
bowtie	1102	0.2422
chevron	4372	0.0610
circle	4810	0.0555
line	6490	0.0411
polygon	1364	0.1956
quad	8518	0.0313
star	1294	0.2062
text	6144	0.0434
triangle	2160	0.1235
Total	36254	1.0000

Table 2.2 Class counts and proportions. Note that multiple-label images are counted separately for each label. Proportions are passed to keras as class weights to ensure that lower-probability classes are learned with equal attention to the available data.

During the model training process, there are a set number of "epochs", which consist of a forward propagation step and a backpropagation step, for the entire set of training data. At the end of the forward propagation step, predicted probabilities are computed and the loss function is applied to the predictions; the average loss over all training images is then the cost function that is used in the backward propagation portion of the epoch. Finally, the validation data is used to assess the model's performance at the end of the backward propagation step; the validation data is used to assess the model, but is not used in any updating of weights. The separate validation set is used to

assess the model's performance and identify any tendency toward overfitting. Chapter 3 discusses the fitting process for this specific model and contains an evaluation of the model's performance.

CHAPTER 3. RESULTS

3.1 Model Training

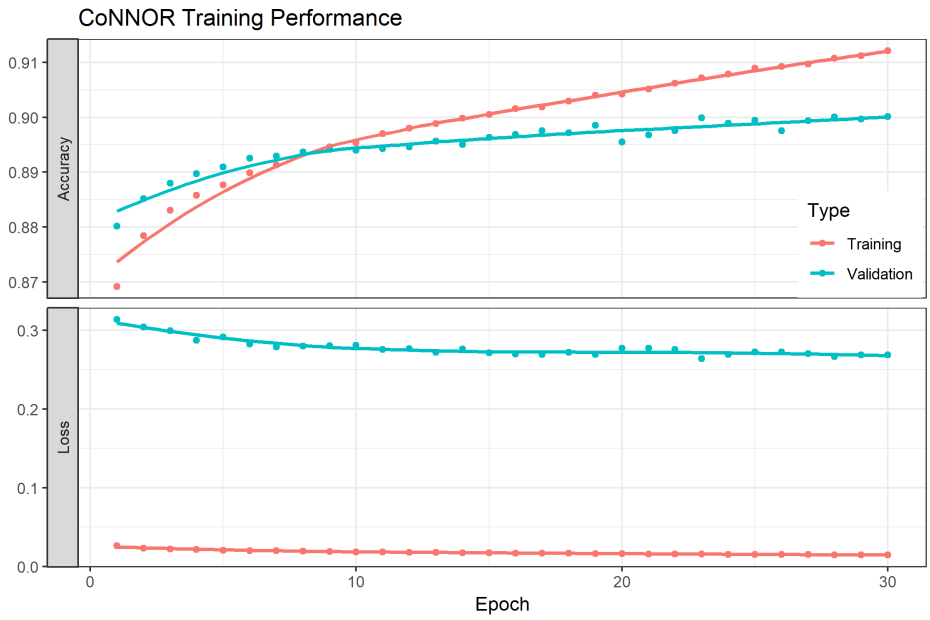

Figure 3.1 Training and validation accuracy and loss for each epoch of the fitting process. Training and validation accuracy reach 89.5% around epoch 9. After that point, validation loss remains the same and training loss decreases slightly, while validation accuracy increases more slowly than training accuracy.

Figure 3.1 shows the training and validation accuracy and loss at each epoch of the fitting process. Overfitting, or fitting a model which performs too well on the training data relative to the validation data, is seen when the validation loss starts to increase after reaching a global minimum. Alternately, underfitting occurs when the validation accuracy is still increasing when model optimization is terminated. Neither of these outcomes appears in Figure 3.1, indicating that the model optimization process was halted at an appropriate epoch.

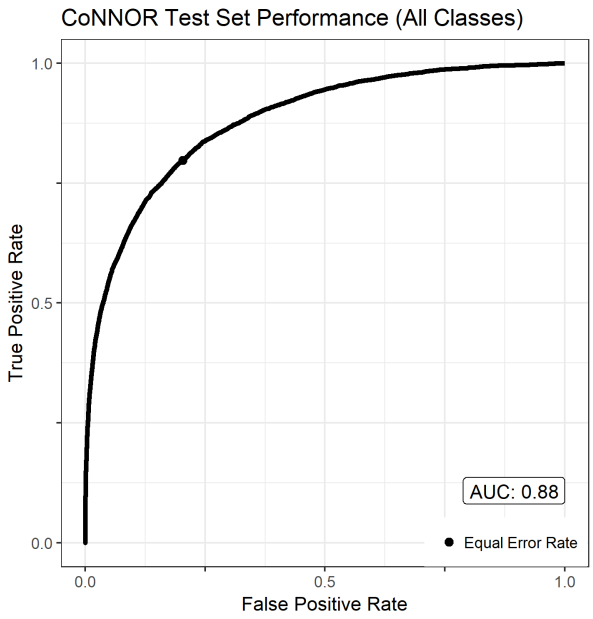

Figure 3.2 ROC curve showing overall model performance.

3.2 Model Accuracy

Figure 3.2 shows the ROC curve for the full model, and Figure 3.3 shows the curve for each class. The full model has an AUC of 0.88, and the AUC for individual classes ranges from 0.81 (for line) to 0.91 (for bowtie and text). While the class performances do vary slightly, each ROC curve is the same general shape and performs significantly better than random chance.

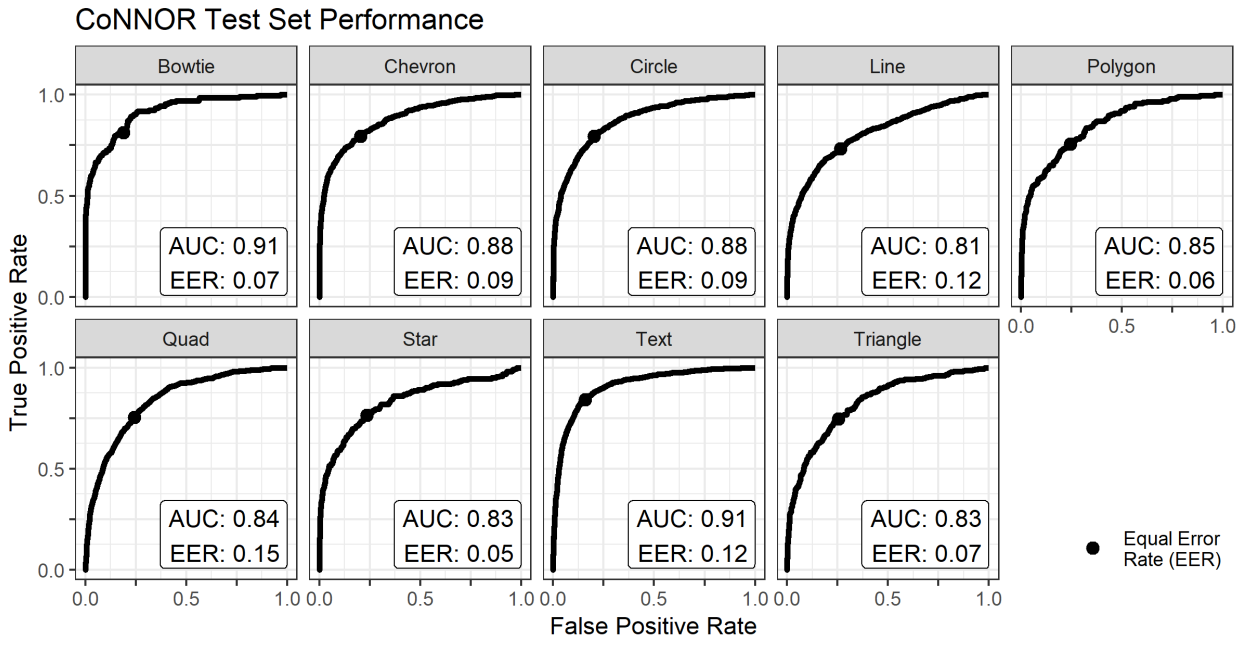

Figure 3.3 Class-by-class ROC curves. AUC is area under the curve, a measure of overall model performance. Equal error rates are marked, indicating the position at which there is equal probability of a false positive or false negative error.

As discussed in Section 1.4, a confusion matrix is a cross-tabulation of the labels assigned to an image and the labels predicted by the model. In a single-label problem, an image belongs to only one category and the model predicts only one label, so applying the wrong label is equivalent to failing to apply the correct label. Thus, every possible combination of true label and applied prediction is represented in the two-dimensional/traditional confusion matrix. Extending the confusion matrix to more complex problems requires additional considerations.

Although some efforts to generalize confusion matrices to multi-class problems do exist in the literature (Landgrebe and Duin, 2008), it appears that confusion matrices have not previously been applied to multi-label classification problems. Since a single image may belong to a combination of the n categories, it is no longer true that a false positive and a false negative are equivalent (see Table 1.2). The confusion matrix presented in Figure 3.4 is like a traditional confusion matrix in that all values along the diagonal represent the proportion of true positives captured within each category. The off-diagonal values, however, are adjusted from the traditional method to remove the effect of any true positives from the calculation of false positive proportions. For example, to calculate the proportion of images that contain triangles but are being falsely labeled as containing quadrilaterals, any image that truly contains both triangles and quadrilaterals is removed before calculating the proportion of false quadrilateral labels.

This modified confusion matrix preserves the ability to identify patterns in over- and under-predictions; for instance, the horizontal band in Figure 3.4 indicates that quadrilaterals are over-represented, as they are predicted more often than they should be for every true label. Similarly, polygons, stars, and triangles are often underpredicted relative to the images which actually contain these categories since they produce a large number of false positive predictions into other categories, as evidenced by the vertical bands.

Another interesting artifact in Figure 3.4 is that circles are often misclassified as text, and text is often mislabeled as containing circles. Not all of the images used in model training are accurately labeled: in some cases, the labels have not been updated after labeling guidelines have changed. The confusion between text and circles is a natural consequence of the overlap between geometric

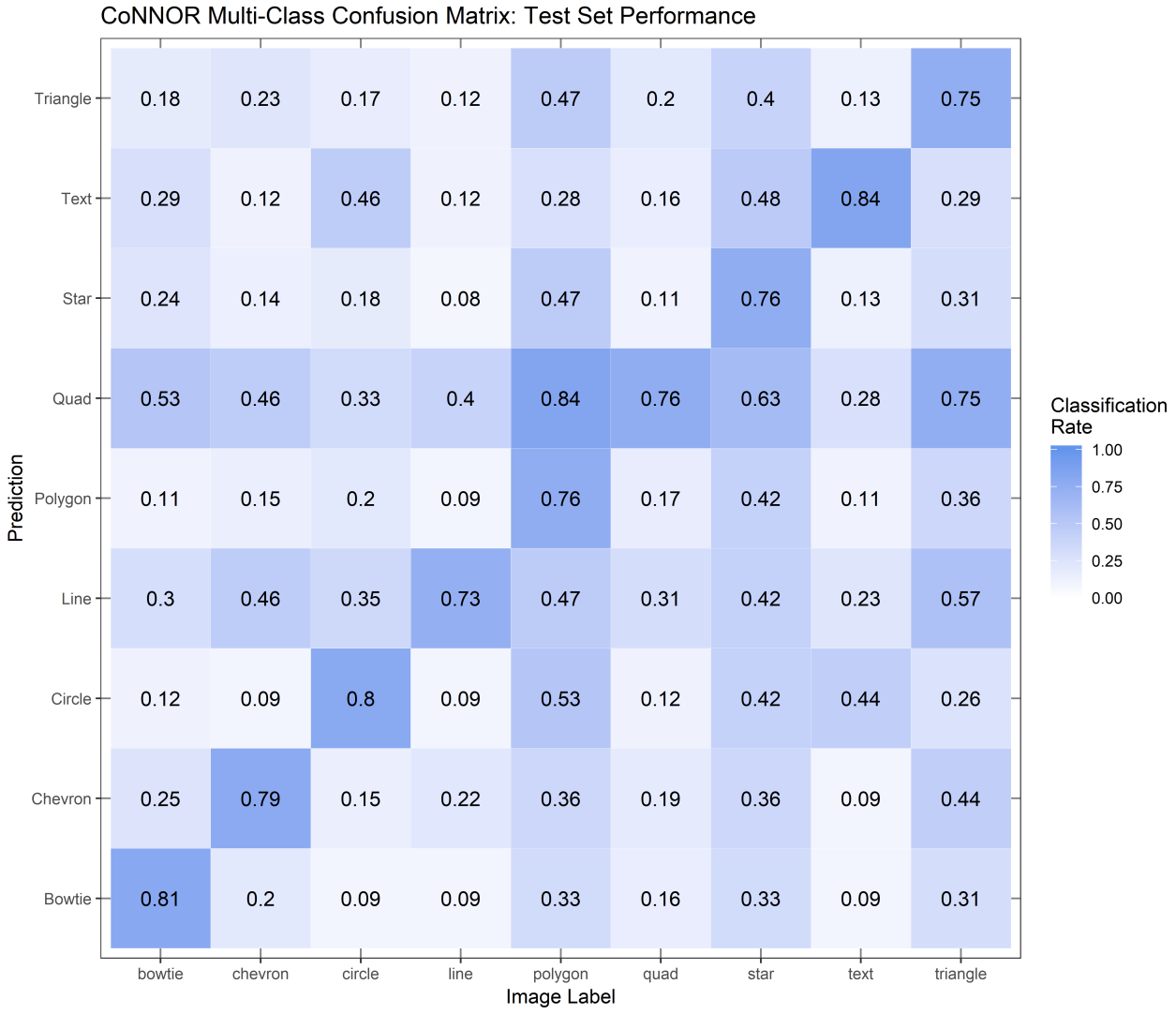

Figure 3.4 Confusion matrix, showing on the diagonal the correct classification rate and on the off-diagonal, classification errors. Note that in multi-label images, correct off-diagonal labels have been excluded from the calculation of false positives.

shapes and components of written text; it is not indicative of a lack of fit as much as a weakness in the classification system used to define the model.

Figure 3.4 shows the general weaknesses of the model, but does not necessarily suggest where to look as to *why* the model is not performing appropriately. In Sections 3.3 and 3.4, alternative visualizations are used to better understand the weaknesses of the model when applied to actual images used to train, test, and validate the model.

3.3 Model Consistency

In order to examine the model's consistency, and whether it can successfully detect features across a variety of images, several case studies are explored in this section that probe the model's strengths and weaknesses.

3.3.1 General shape recognition

CoNNOR generally performs well when predicting well-defined shapes. Figures 3.5, 3.6, and 3.7 show a number of test images containing chevrons, lines, text, and triangles. The chevrons in Figure 3.5 are usually detected when they are in a regularly repeating pattern, and also when the shape is distinctly pointed; there is one case shown where the chevron base is quite curved and model prediction is low across all classes. Another image shows chevrons that are made up of quadrilaterals; this case is strongly predicted as quadrilateral and only weakly as chevron. Instances like this occur in many contexts throughout the data and have played a significant role in modifying the labeling scheme to consider smaller shapes that are part of larger classes, such as labeling an "o" in text as a circle.

The images in Figure 3.6 are primarily all identified as containing text. The two exceptions are both images with low contrast; the darkest image is only identified as containing lines, and the lightest image is predicted with moderate probabilities into nearly every class. Contrast is also an issue in Figure 3.7, where a white image is correctly labeled as containing text but the triangle is not identified. The darkest images with triangles are correctly identified with moderate probabilities,

but the model also assigns similar probabilities to the class of quadrilateral, which is incorrect for these images. These images suggest that CoNNOR may have a general difficulty predicting images with low contrast, producing both false positive and false negative predictions.

The impact of contrast may be due to the artificiality of the shoe images. VGG16 was trained to recognize photographs of subjects that are more naturally-occurring and taken in real settings, such as in Figure 1.3. The outsole images used in this research, however, are much more artificial images, and any color variation in the image is part of the shoe design. Thus, the "color knowledge" of VGG16 is not perfectly generalizable to the domain of classifying outsole images.

3.3.2 Recognition across colors

In order to more fully understand CoNNOR's strengths and weaknesses, several shapes were identified which are consistently present in several different color patterns, either from the same shoe model, or from the same brand and very similar tread patterns. These cases explore model performance when the geometric patterns are controlled and color patterns are varied in real context—while colors could be varied artificially, additional artifacts may be introduced in the image manipulation process.

In the more homogeneous contrast situation shown in Figure 3.8, the color variation does not produce large variations in the output probabilities—the lightest image has only slightly lower probability than the darker images. On the other hand, Figures 3.9, 3.10, 3.11, and 3.12 show cases where CoNNOR is able to identify shapes when contrast is moderate, but has greater difficulty when contrast is very low. The effect of contrast adjustment on CoNNOR's predictions is further explored in Section 3.3.3.

3.3.3 Contrast Adjustment

In several of the model consistency case studies presented, color contrast was identified as a possible reason for poor feature identification. There are automatic ways to adjust the contrast of photos, enhancing CoNNOR's ability to notice details. Figure 3.13 shows the original predictions

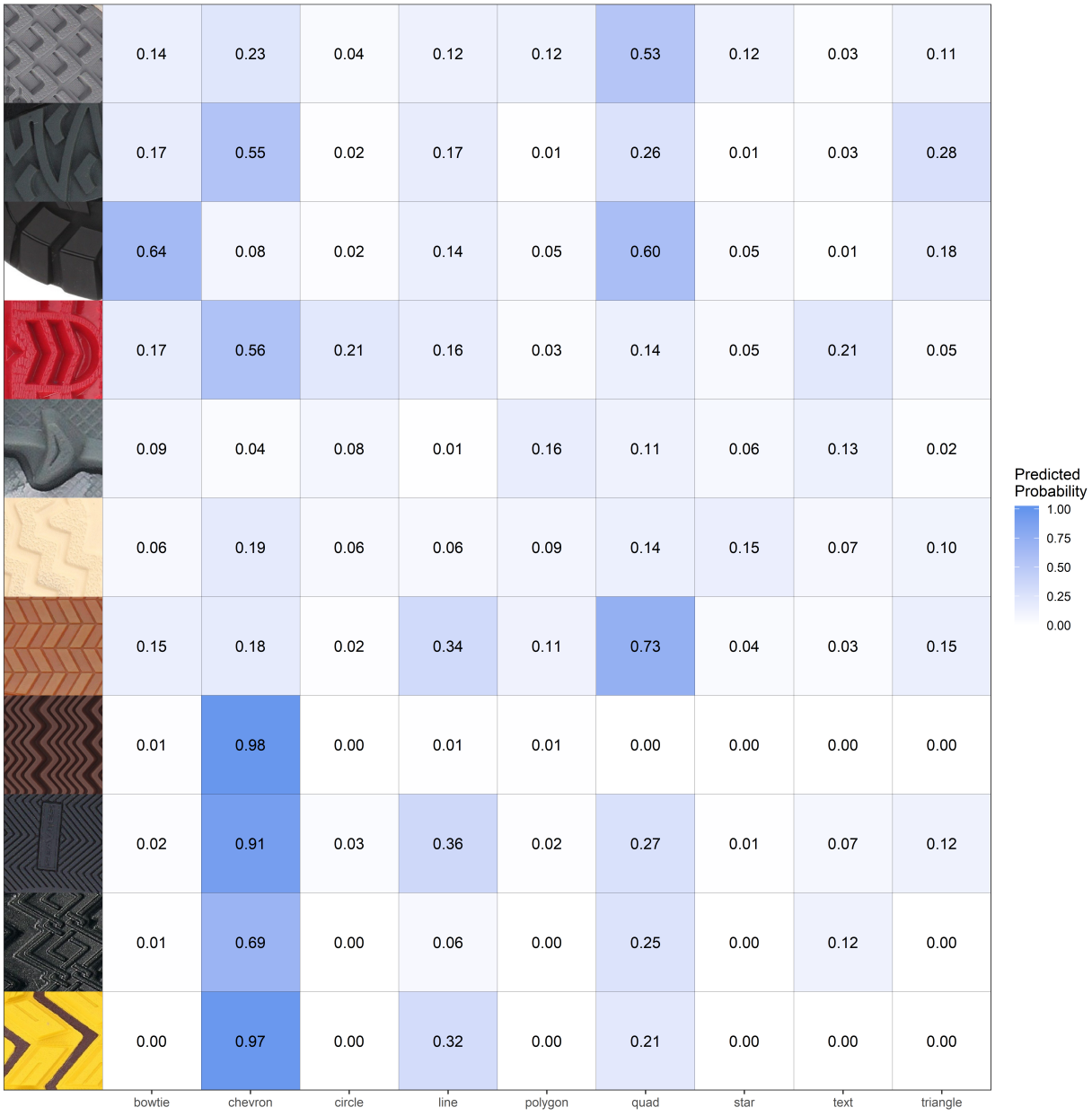

Figure 3.5 Most chevrons are correctly classified, though instances where two quadrilaterals make up a chevron are missed, and one low-contrast light sole has low probability across all 9 geometric figures, indicating that low color contrast may be an issue for CoNNOR.

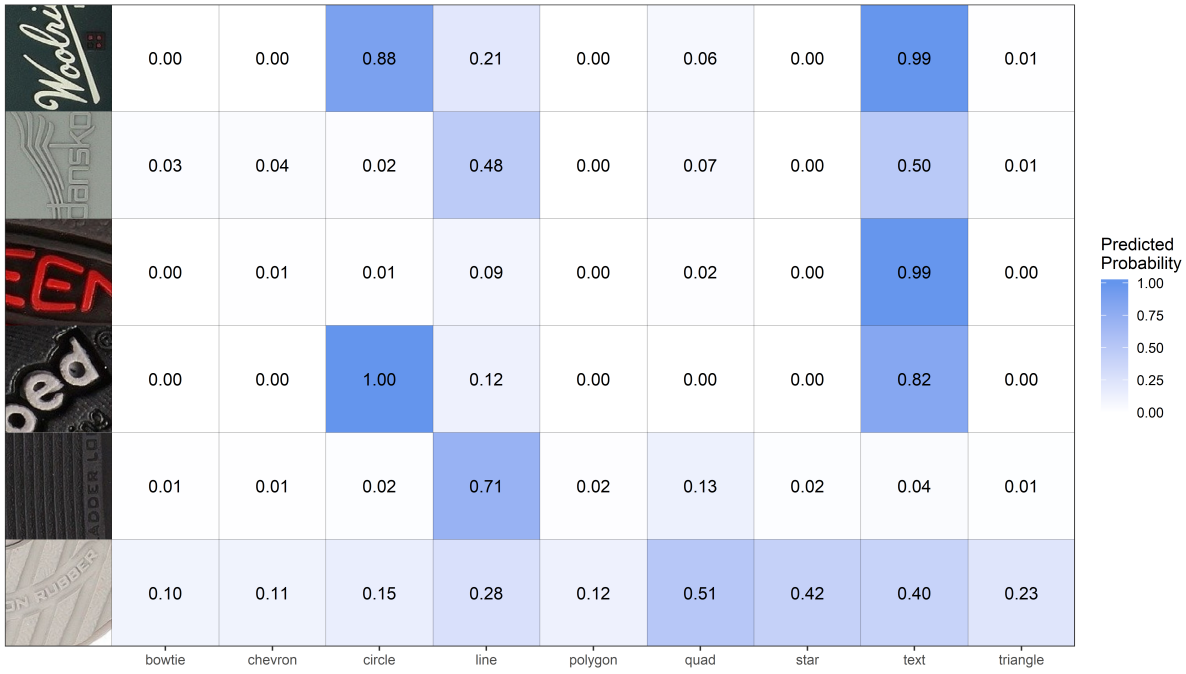

Figure 3.6 A selection of images containing text and/or lines. Most images are correctly identified as containing text, though again there is some indication that low–contrast images are poorly recognized. In addition, text which occurs towards the boundaries of the labeled region may not be successfully identified by the model.

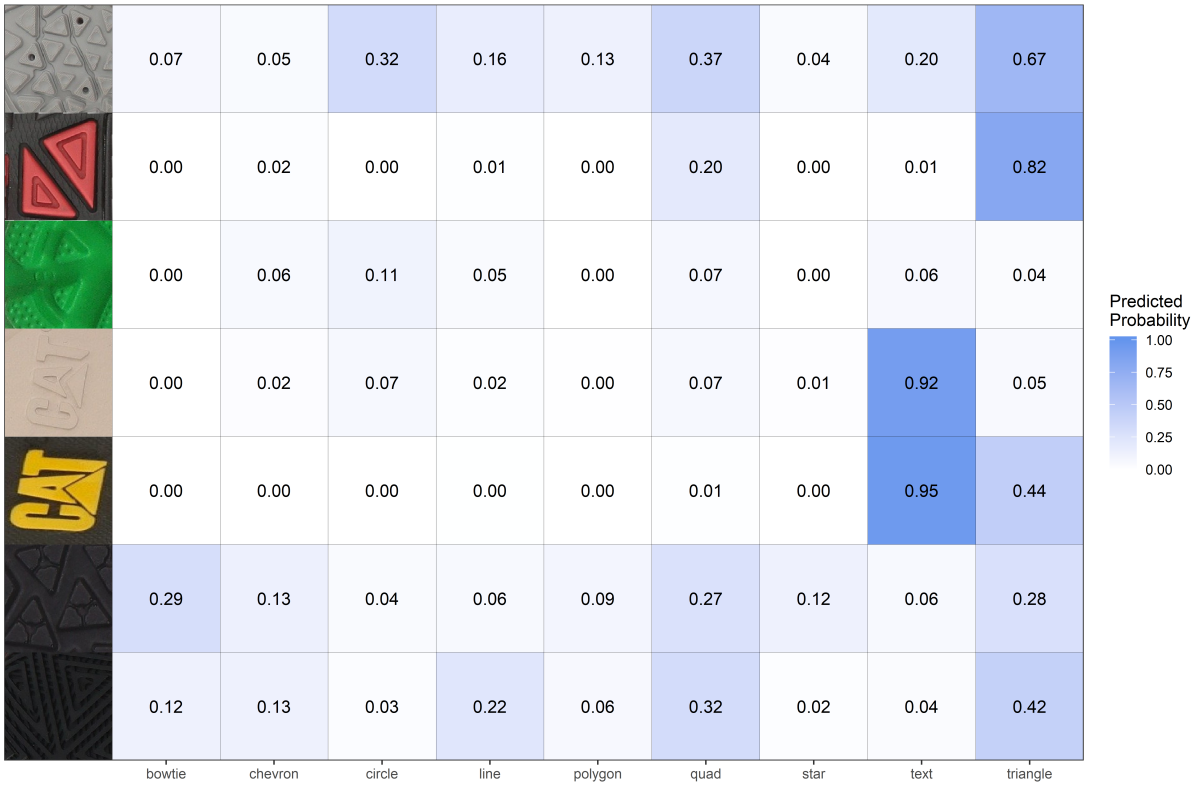

Figure 3.7 A selection of images with triangles, including text logos. Many of the triangles are correctly detected, but there are again problems with low-contrast images and with recognition of the triangles in the top image.

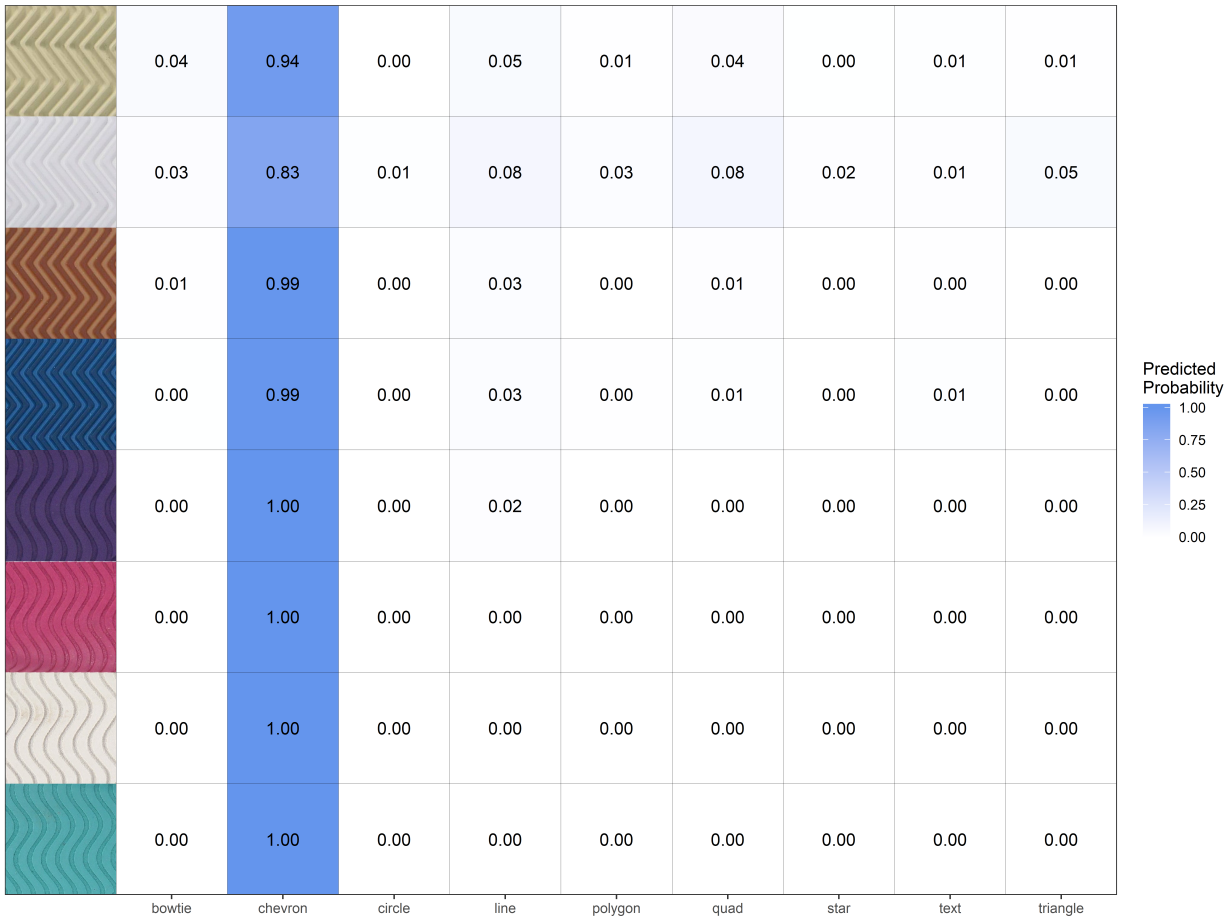

Figure 3.8 A selection of images containing a specific chevron pattern from Adidas shoes, in a variety of colors. In the more homogeneous contrast situations presented here, features are detected across color patterns.

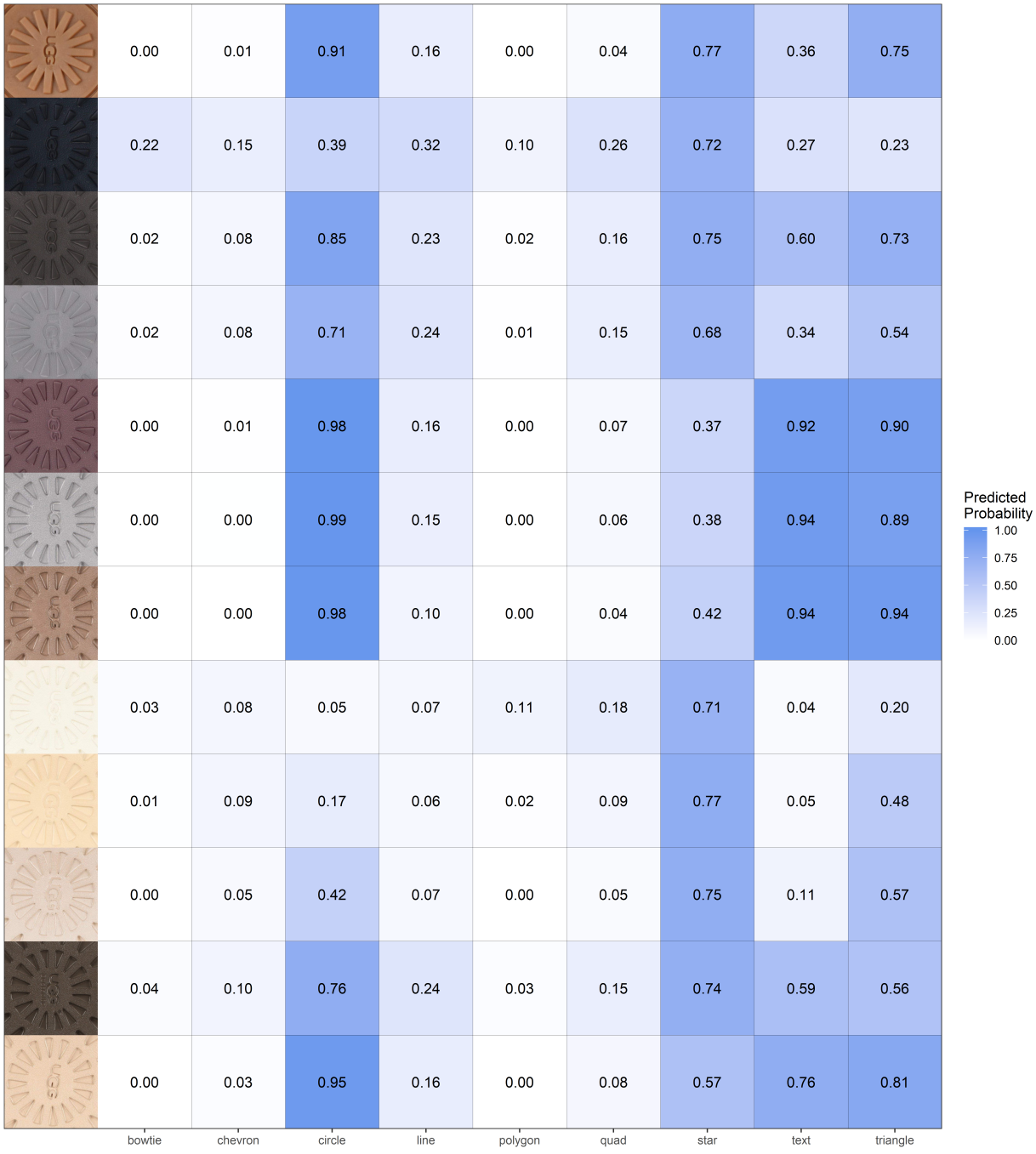

Figure 3.9 UGG logos as found on many different shoes, without a defined circle outline. The model predictions are relatively consistent, but detection of the circle and text elements vary based on image contrast and color.

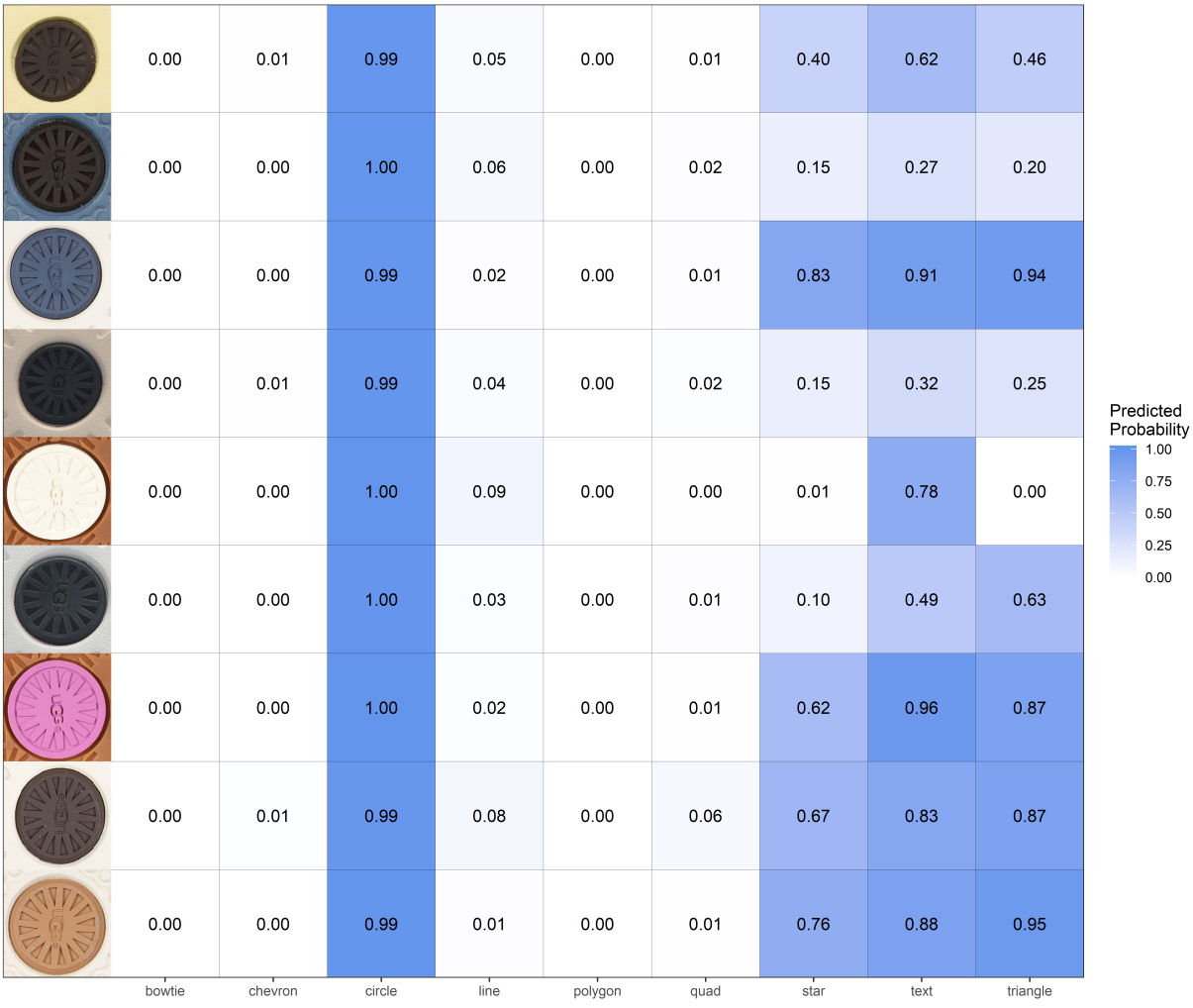

Figure 3.10 UGG logos as found on many different shoes, with a defined circle outline. Again, the model predictions are relatively consistent, but now detection of the triangle and star elements vary based on image contrast and color.

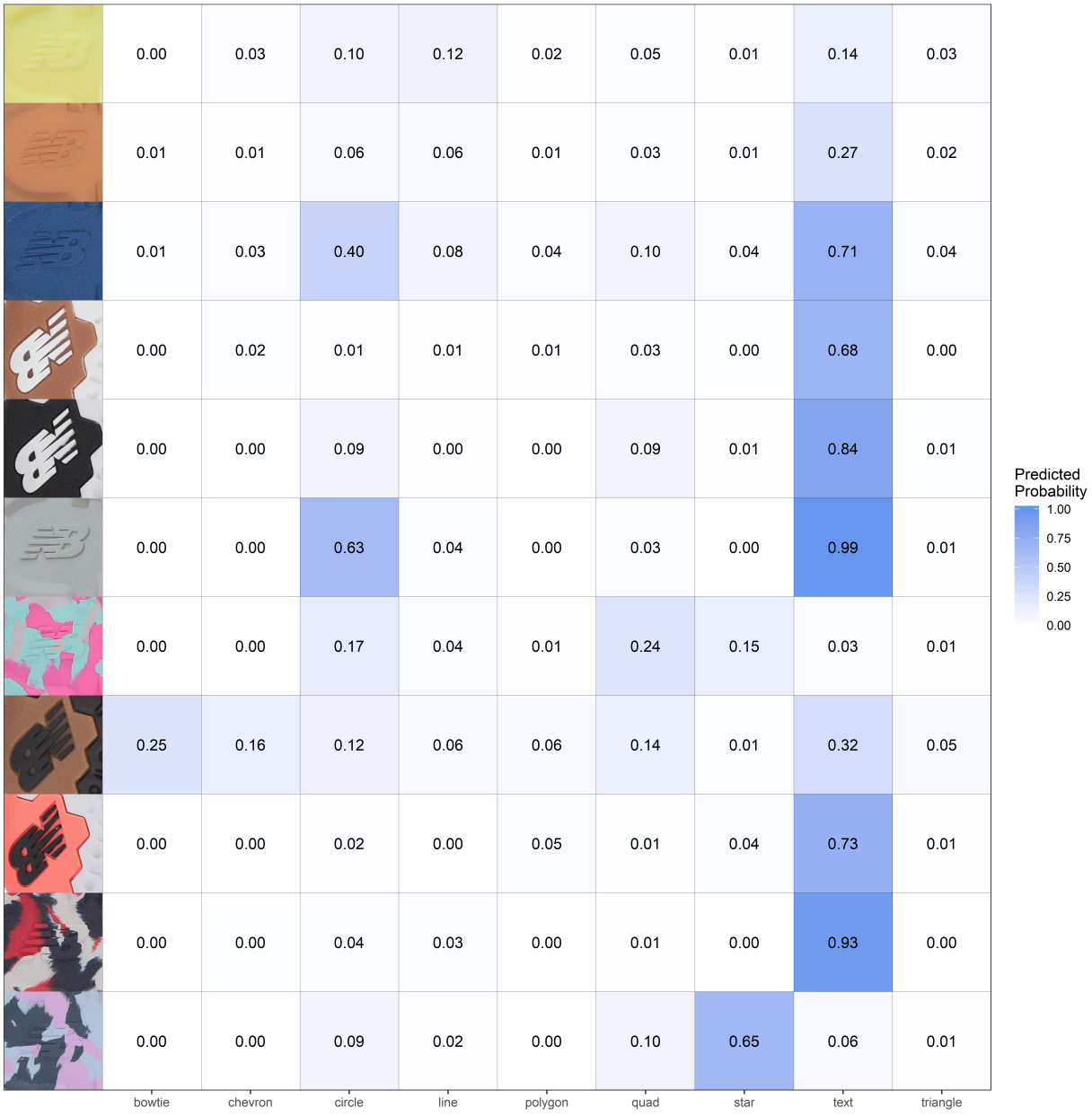

Figure 3.11 New Balance logos found on several different shoes; note that the model can sometimes detect the logo even with color/texture changes, but when the color pattern is of higher contrast than the logo, the model can no longer successfully recognize the logo.

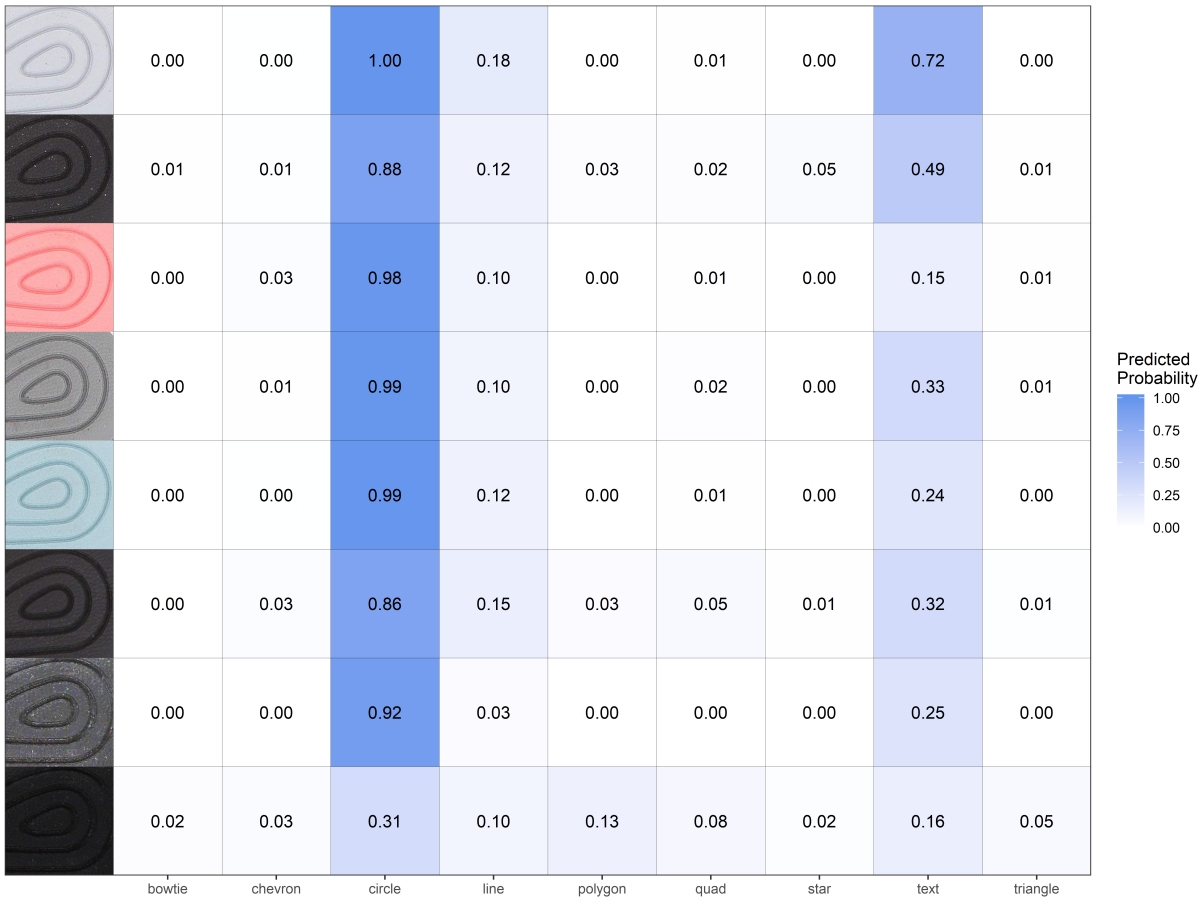

Figure 3.12 A selection of images from the same shoe model across different colors. The bottom figure, which is of a black sole and has very little contrast, is poorly recognized by the model. The image immediately above it is an automatically color-adjusted version of the same image, which is classified consistently with the other images.

of previous cases with low contrast paired with predictions of the images after contrast adjustment. Predictions of the adjusted images are not drastically different; the chevrons at the bottom take on a moderate false positive bowtie prediction, but the predictions of the UGG and the New Balance logos show significant improvement.

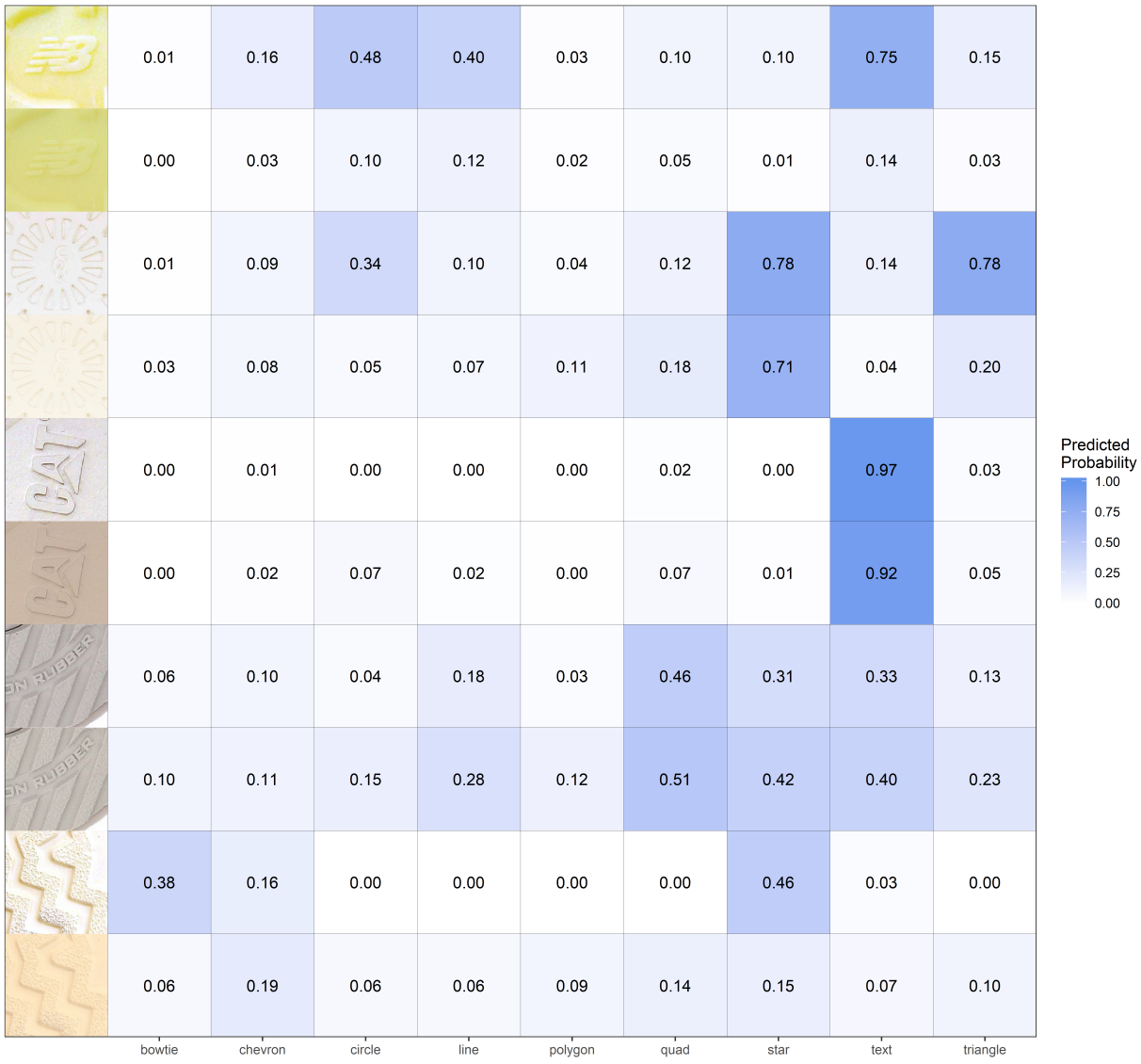

Figure 3.13 Images with low contrast from previous cases, with corresponding auto-adjusted color balance images. The color balanced images are on top of the corresponding low-contrast images. In most cases, auto color balancing increases the output class probabilities for relevant categories, though the chevron image at the bottom is strongly recognized as a bowtie instead of a chevron pattern.

3.4 Model Diagnostics: Heatmaps

A heatmap is a visual diagnostic tool that displays the portions of an image that are most significant to classification for a given class. Belonging to a category of techniques called class activation map (CAM) visualizations, heatmaps are computed by taking the output feature map of a convolutional layer (in this case, the final convolutional layer of VGG16) and weighting each channel of the feature map by the gradient of the class with respect to the channel (Chollet and Allaire, 2018). Heatmaps provide diagnostic information: using these graphics, we can identify which locations in an image contribute most to CoNNOR's predictions.

In the figures in this subsection, heatmaps are shown for a selection of three classes, which are either contained in the image or have high output probabilities, alongside the original image. Figure 3.14 shows a case where the chevron and circle predictions are quite good, but there is a line dividing the two sections that is falsely predicted to be a quadrilateral. Figure 3.15 shows an outsole with cork shapes; the model bases its high prediction of star on the concave corners where the corks meet. This example also shows the importance of including the category "other" during model training, as the model must be evaluated on images which do not contain any of the classes. The original image in Figure 3.16 contains repeated patterns of quadrilaterals, hexagons, and triangles; CoNNOR predicts all three classes correctly, and the heatmap indicates that the appropriate image features are being used for these classifications.

Figure 3.17 shows correct predictions for both quadrilateral and star, but a false positive label for triangle. This false label is not necessarily incorrect; the very small center of the star blurs the distinction between a single star and multiple triangles, and it might be reasonable to update the image label to include triangle. CoNNOR's confident prediction of a circle in Figure 3.18, however, is clearly incorrect. Still, the heatmap is encouraging because it provides evidence that CoNNOR's prediction of circles is closely linked to shapes with pronounced curves. Figure 3.19 shows a correct prediction of star which is identified by its pointed ends, but CoNNOR has difficulty identifying the bowties in the image.

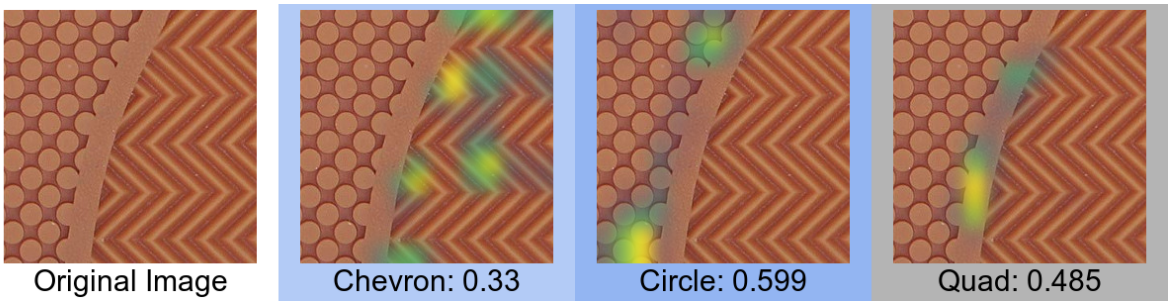

Figure 3.14 Chevrons are identified by sharp corners and parallel lines, but the thick ribbon between the two shape sections is falsely identified as a quadrilateral.

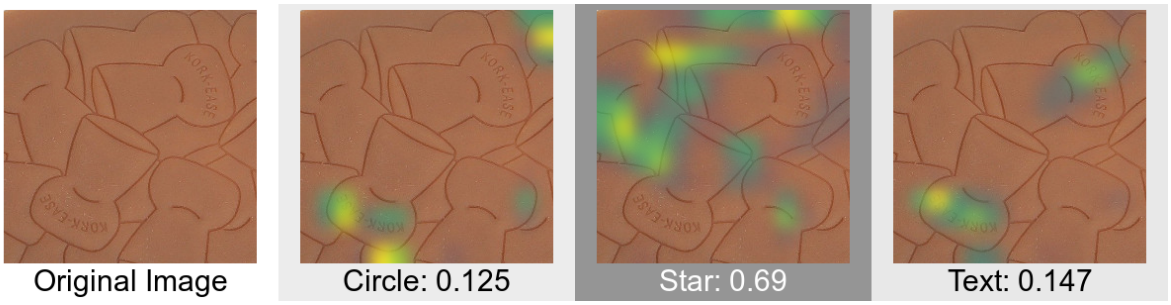

Figure 3.15 This outsole shows repeated outlines of corks, which do not fall into any of the nine classes; the concave corners where the corks overlap are predicted to be stars, and the round edges lead to predictions of circles.

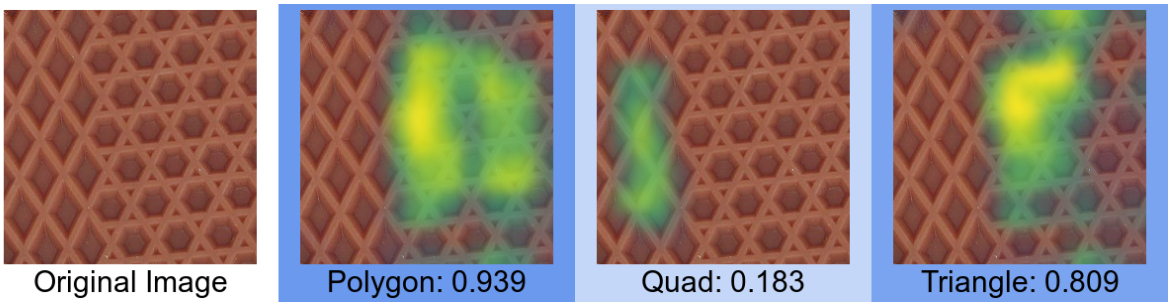

Figure 3.16 The quadrilaterals, hexagons, and triangles in this image are all correctly identified, and the heatmap indicates that CoNNOR is using appropriate features for these classifications.

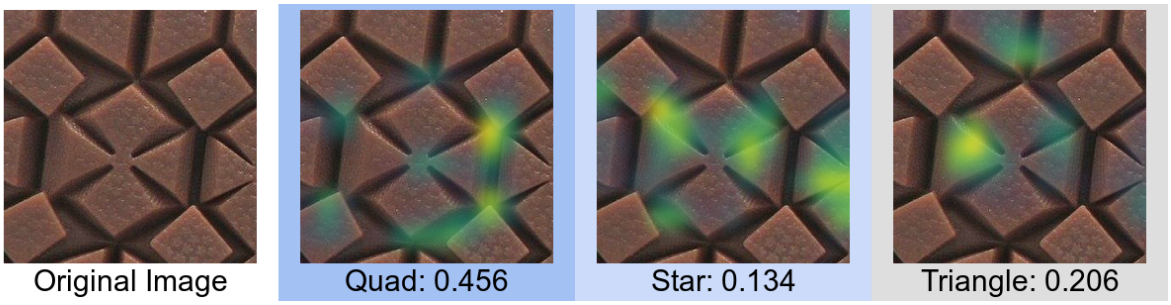

Figure 3.17 CoNNOR correctly identifies the quadrilaterals by looking at their corners, and the star is identified by the acute angles of its convex portions.

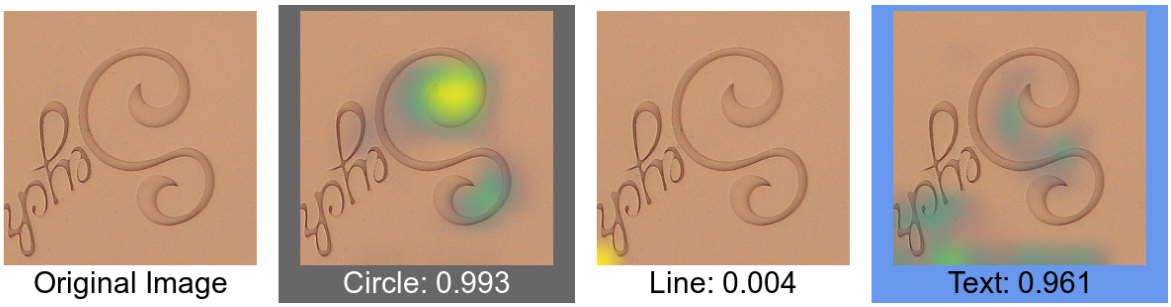

Figure 3.18 This heatmap illustrates that the high probability of a circle is a result of the highly curled tails of the 'S' in the text.

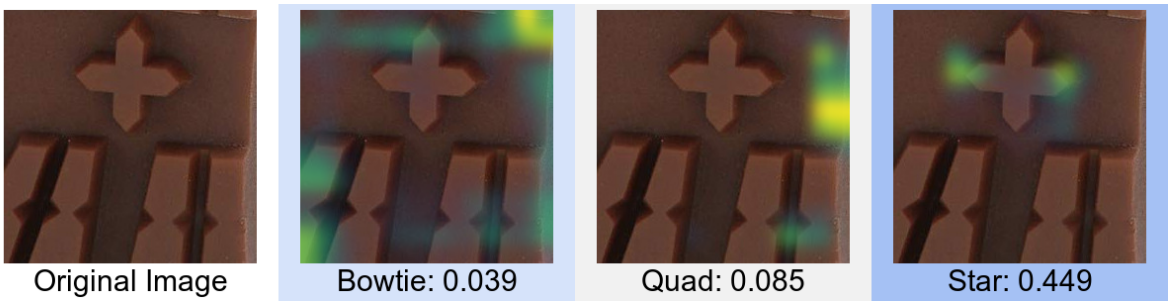

Figure 3.19 CoNNOR identifies the star by its pointed ends, but appears to have difficulty locating and identifying the bowties.

CHAPTER 4. CONCLUSION

4.1 Summary of Results

The goal of this research was to develop a method to automatically identify geometric class characteristics of shoe outsoles. Such a method takes the "impossible" problem of quantifying outsole features in a given population and lays the groundwork for a tractable solution. A set of geometric class characteristics was defined to both broadly classify a large variety of shoes and to narrow down similarity into a manageable number and type of features for further use in a shoeprint analysis. The final set of features used in this research achieves both goals quite well, and has potential for much finer-grained analysis when recognized features are combined with spatial information and relationships.

CoNNOR was developed using the convolutional base of VGG16 and a newly trained classifier/head to identify the defined set of geometric features. After training, CoNNOR performs well on the data provided. In general, the model is able to identify many well-defined geometric shapes in the images both consistently and accurately. The prediction accuracy of this model provides the ability to compute statistics for the frequency of given features in a well-defined population.

4.2 Future Work

Although CoNNOR performs well in its current state, there are still a number of ways to potentially improve prediction accuracy. For one, it is clear from Section 3.3 that image contrast still plays a large role in how well the model classifies the geometric shapes present. Thus, exploring methods of color correction, such as histogram normalization, may prove useful for eliminating the effect of contrast on predictions. Geometric features are also relatively simple with respect to the features that are being detected by the final convolutional block of VGG16, as in Figure 2.9; it is quite possible that prediction accuracy could improve by directly classifying the features that

are output by the fourth block, rather than using the full convolutional base. In addition, there is strong evidence that a CNN can be trained to differentiate texture from color (Gatys et al., 2015; Dumoulin et al., 2016; Gatys et al., 2016; Andrearczyk and Whelan, 2016, 2017). Thus, it could be useful to train CoNNOR such that the emphasis for classification is based less on color differences in the image and more on texture and other features.

Once CoNNOR's performance is optimized on the current data, which are 256 $\times$ 256 square pixel regions cropped from full outsoles, spatial information could be integrated to represent the information in a whole shoeprint or image. One way to accomplish this could be to divide an outsole image into many smaller regions and track the change in feature predictions across those regions. This would allow the features of an outsole to be "mapped" in such a way that location of a feature could provide more information than simply the presence or absence of a feature in determining similarity between given outsoles. Another alternative would be to transition to the use of a fully convolutional network (FCN), which has only convolutional layers; these networks can work with images of any size, providing a way to maintain spatial information through the classification process (Springenberg et al., 2014; Andrearczyk and Whelan, 2017).

BIBLIOGRAPHY

CS231n Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/convolutional-networks/#fc.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Allaire, J. and Chollet, F. (2018). *keras: R Interface to 'Keras'*. R package version 2.2.4.

Andrearczyk, V. and Whelan, P. F. (2016). Using Filter Banks in Convolutional Neural Networks for Texture Classification. *arXiv:1601.02919 [cs]*. http://arxiv.org/abs/1601.02919.

Andrearczyk, V. and Whelan, P. F. (2017). Texture segmentation with Fully Convolutional Networks. *arXiv:1703.05230 [cs]*. http://arxiv.org/abs/1703.05230.

Ballard, D. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

Bodziak, W. J. (2000). *Footwear Impression Evidence: Detection, Recovery, and Examination*. CRC Press, Boca Raton, Florida.

Chollet, F. and Allaire, J. (2018). *Deep Learning with R*. Manning Publications Company.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Dumoulin, V., Shlens, J., and Kudlur, M. (2016). A Learned Representation For Artistic Style. *arXiv:1610.07629 [cs]*. http://arxiv.org/abs/1610.07629.

Gatys, L., Ecker, A. S., and Bethge, M. (2015). Texture Synthesis Using Convolutional Neural Networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 262–270. Curran Associates, Inc.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, Las Vegas, NV, USA. IEEE.

Geirhos, R., Janssen, D. H. J., Schtt, H. H., Rauber, J., Bethge, M., and Wichmann, F. A. (2017). Comparing deep neural networks against humans: object recognition when the signal gets weaker. *arXiv:1706.06969 [cs, q-bio, stat]*.

Goldstein, E. B. and Brockmole, J. (2016). *Sensation and perception*. Cengage Learning.

Gross, S., Jeppesen, D., and Neumann, C. (2013). The variability and significance of class characteristics in footwear impressions. *Journal of Forensic Identification*, 63(3):332.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377.

Hancock, S., Morgan-Smith, R., and Buckleton, J. (2012). The interpretation of shoeprint comparison class correspondences. *Science and Justice*, 52(4):243–248.

Hand, D. J. and Till, R. J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*, 45(2):171–186.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*. http://arxiv.org/abs/1512.03385.

Jain, R., Kasturi, R., and Schunck, B. G. (1995). *Machine vision*. McGraw-Hill Science/Engineering/Math, 1 edition.

Kotikalapudi, R. and contributors (2017). keras-vis. https://github.com/raghakot/keras-vis.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Landgrebe, T. C. W. and Duin, R. P. W. (2008). Efficient Multiclass ROC Approximation by Decomposition via Confusion Matrix Perturbation Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):810–822.

Mallot, H., Allen, J., and Sejnowski, T. (2000). *Computational Vision: Information Processing in Perception and Visual Behaviour*. Bradford book.

Papert, S. (1966). The Summer Vision Project. MIT AI Memos.

Prabhu, R. (2018). Understanding of Convolutional Neural Network (CNN) Deep Learning. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*. http://arxiv.org/abs/1412.6806.

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA. IEEE.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc.

APPENDIX A. COMPUTER CODE

This appendix contains the code used to augment the training images and to train the model head of CoNNOR. Before presenting the code, it will helpful to describe the structure of the full project directory.

Directory structure All code is contained in the /models/shoe_nn directory. There are two main subdirectories—RProcessedImages and TrainedModels—that hold the labeled and processed images and the trained models, respectively. Each set of data and trained model are stored in directories that are named with the convention "YYYYMMDD-HHMMSS". This structure is shown explicitly below.

/ models/shoe_nn

- / RProcessedImages
  - / 20190307-130702
    - / test
    - / train
    - / validation
- / TrainedModels/
  - / 20190307-130702/

Setup

```
library(magrittr)
library(lubridate)
library(stringr)
library(jpeg)
library(keras)
use_backend("tensorflow")
# install_keras()
```

```
if (!exists("process_dir")) {
  process_dir <- list.files("/models/shoe_nn/RProcessedImages") %>%
    as_datetime() %>%
    max(na.rm = T) %>%
    gsub("[^0-9\\ ]", "", .) %>%
    gsub(" ", "-", .)
} else {
  process_dir <- file.path("/models/shoe_nn/RProcessedImages", process_dir)
}

if (!exists("aug_multiple")) {
  aug_multiple <- 3
}

if (!exists("epochs")) {
  epochs <- 30
}

dir_regex <- paste0("[[:punct:]]models[[:punct:]]shoe_nn",
                    "[[:punct:]]RProcessedImages[[:punct:]]{1,}")
process_dir <- gsub(dir_regex, "\\1", process_dir)
process_dir <- gsub("^[/\\\\]{1,}", "", process_dir)


work_dir <- "/models/shoe_nn/TrainedModels"
start_date <- Sys.time() %>% gsub(" ", "_", .)
model_dir <- file.path(work_dir, process_dir)
dir.create(model_dir)

name_file <- function(date, ext) {
  pretrained_base <- "vgg16"
  mod_type <- "onehotaug"
  nclass <- paste0(length(classes), "class")
```

```
  pixel_size <- "256"

  filename <- paste(date, pretrained_base, mod_type, nclass,
    pixel_size,
    sep = "_"
  )

  file.path(model_dir, filename) %>%
    paste0(., ext)
}


base_dir <- file.path("/models/shoe_nn/RProcessedImages", process_dir)
train_dir <- file.path(base_dir, "train")
train_aug_dir <- file.path(base_dir, "train")
validation_dir <- file.path(base_dir, "validation")
test_dir <- file.path(base_dir, "test")
```

Image Augmentation

```
n_train <- length(list.files(train_dir))
n_validation <- length(list.files(validation_dir))
n_test <- length(list.files(test_dir))

img_names <- list.files(train_dir) %>% str_remove(., "\\.jpg")
img_loc <- list.files(train_dir, full.names = T)

augment_img <- function(filename, times = 3) {
  # Determine number of times augmentation should happen
  rep_num <- str_extract(basename(filename), "^\\d") %>% as.numeric()

  if (!is.na(rep_num)) {
    file_dir <- dirname(filename)
    fix_file_name <- str_remove(basename(filename), "^\\d{1,}\\+")
    base_file_name <- file.path(file_dir, fix_file_name)
    file.rename(filename, base_file_name)
    newfilepath <- base_file_name
    newfilename <- basename(base_file_name) %>% str_remove("\\.jpg")
  } else {
    rep_num <- 1
    newfilepath <- filename
    newfilename <- basename(filename) %>% str_remove("\\.jpg")
  }
```

```
  img <- readJPEG(newfilepath)
  dim(img) <- c(1, dim(img))

  aug_generator <- image_data_generator(
    samplewise_std_normalization = T,
    rotation_range = 40,
    width_shift_range = 0.05,
    height_shift_range = 0.05,
    shear_range = 60,
    zoom_range = 0.1,
    channel_shift_range = .1,
    zca_whitening = T,
    vertical_flip = T,
    horizontal_flip = TRUE
  )

  images_iter <- flow_images_from_data(
    x = img, y = NULL,
    generator = aug_generator,
    batch_size = 1,
    save_to_dir = train_aug_dir,
    save_prefix = paste("aug", newfilename, sep = "_"),
    save_format = "jpg"
  )

  iter_num <- times
  while (rep_num > 0) {
    while (iter_num > 0) {
      reticulate::iter_next(images_iter)
      iter_num <- iter_num - 1
    }
    rep_num <- rep_num - 1
  }
}

for (i in list.files(train_dir, "*.jpg", full.names = T)) {
  augment_img(i, times = aug_multiple)
}
```

Convolutional Feature Extraction

This next section of code evaluates each image as processed by the convolutional base, and creates features from the last pooling layer. These features are used to train CoNNOR's model head.

```
conv_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(256, 256, 3)
)

extract_features2 <- function(directory, verbose = F) {

  files <- list.files(directory)
  sample_count <- length(files)

  features <- array(0, dim = c(sample_count, 8, 8, 512))
  labels <- array(0, dim = c(sample_count, length(classes)))

  cat(paste0("Sample count = ", sample_count, ". "))

  for (i in 1:sample_count) {

    if (verbose) {cat(paste0(i, ", "))}
    fname <- files[i]
    str <- substr(fname, 1, regexpr("-", fname) - 1)

    for (j in 1:length(classes)) {
      labels[i, j] <- grepl(classes[j], str)
    }

    img <- readJPEG(file.path(directory, files[i]))
    dim(img) <- c(1, 256, 256, 3)
    features[i, , , ] <- conv_base %>% predict(img)
  }

  list(
    features = features,
    labels = labels
  )
}
```

```
train <- extract_features2(train_dir, verbose = T)
validation <- extract_features2(validation_dir, verbose = T)
test <- extract_features2(test_dir, verbose = T)


reshape_features <- function(features) {
  array_reshape(features, dim = c(nrow(features), 8 * 8 * 512))
}

train$features <- reshape_features(train$features)
validation$features <- reshape_features(validation$features)
test$features <- reshape_features(test$features)
```

Training the Model Head

```
class_quantities <- colSums(train$labels)
class_proportions <- (1/class_quantities)/sum(1/class_quantities)
class_weights <- class_proportions %>%
  as.list() %>%
  set_names(as.character(1:length(class_proportions) - 1))

model <- keras_model_sequential() %>%
  layer_dense(
    units = 256, activation = "relu",
    input_shape = 8 * 8 * 512
  ) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = length(classes), activation = "sigmoid")

model %>% compile(
  optimizer = optimizer_rmsprop(lr = 2e-5),
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

history <- model %>% fit(
  train$features, train$labels,
  epochs = epochs,
  batch_size = 20,
  validation_data = list(validation$features, validation$labels),
  class_weight = class_weights
)
```

The output of the model training process is then saved in a variety of formats to facilitate different tasks—visualization of layers, predicting classes of new images, and more.

```
png(filename = name_file(start_date, ".png"), width = 1000, height = 1000,
    type = "cairo", pointsize = 16)
plot(history)
dev.off()

save_model_hdf5(model, name_file(start_date, ".h5"))
save_model_weights_hdf5(model, name_file(start_date, "-weights.h5"))

preds <- model %>% predict(test$features)
test_labs <- test$labels
colnames(preds) <- colnames(test_labs) <- classes

save(classes, preds, test_labs, file = name_file(start_date, ".Rdata"))
base::save.image(name_file(start_date, "fullimage.rdata"))

save(history, file = name_file(start_date, "-history.Rdata"))
```