This is the title of a thesis submitted to Iowa State University

Note that only the first letter of the first word and proper names are capitalized

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

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this Creative Component. The Graduate College will ensure this Creative Component is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ABSTRACT

This is the text of my abstract that is part of the thesis itself. The abstract describes the work in general and the heading and style match the rest of the document.

CHAPTER 1. INTRODUCTION

1.1 Motivation

In forensic science, shoe prints and outsole characteristics fall into the category of pattern evidence. When a shoe print 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 important to be able to assess the probability that the specific model of shoe which made the print would be found in the suspect's posessions. 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 statistical features which can be used to assess similarity beteween two or more images.

[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 identification, elimination, class association, and inconclusive. Typically, the ultimate goal of an examination is identification, which is matching a shoe print to an individual shoe, perhaps owned by a suspect of an the investigation. This is difficult because identification to a specific individual's shoe requires the matching of randomly acquired characterisics, such as damage, and that information is frequently unavailable due to the quality of the print or impression recovered from the crime scene. Most of the work being done with shoeprints involves class association, which means identifying 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 informally? categorizing a shoe, it is common (I want to say that it's the first thing that comes to mind, or that's what we do in conversation.) to use features like brand, size, and general type (e.g., boot, tennis shoe, dress shoe) No, it's not informal or conversational, it's actually the way examiners start out - can they classify it as an athletic shoe, dress shoe, etc.. When defining useful class characteristics for identification or exclusion, however, these features prove quite difficult to use. 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 easily characterized because manufactuers are constantly developing new models, discontinuing existing models, or reviving discontinued models, and there exist 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

is? (am I crazy that "be" sounds weird?) 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 able to differentiate between different shoes [need qualifier?] (Gross et al., 2013). Matching 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). In addition, there are a number of low-level feature extraction methods aimed at detecting specific shapes, such as edges, corners, blobs, or ridges(Ramesh Jain, 1995, Ch 15). While these methods are useful in identifying these specific features at a low level, they are very computationally intensive and features they produce/identify exist(?) on a very small scale, often only a few pixels wide, and they are thus not able to identify large geometric shapes like those that may be found in an outsole image. Furthermore, these methods are extremely sensitive to lighting or color changes, and thus require additional modeling (like random forests) to aggregate low-level features into the geometric shapes found on outsoles.

For novel image classification tasks, Convolutional Neural Networks (CNNs) have become standard. [2] (Gu et al., 2018).

[2]The Google Trends interest graph 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

These sentences belong somewhere in here, tbd where: Convolutional neural networks are a tool for supervised deep-learning that have become standard in recent years for automatic image classification. CNNs are a form of artificial neural network, which were inspired by biological processes in the brain (Gerven and Bohte, 2017).

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 readily interpretable by humans. As CNNs make use of labeled training data, the predictions generated are for features which are similar to those identified by humans, providing greater face validity to the model. 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 needs citation?. 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 god bless u 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 com-

pilation 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 sufrace. 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.

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.

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-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.

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 differentiating between a golden retriever and a labrador retriever. Too much? :) P.S. ImageNet is under maintenance at this moment, I'll snag a pic of a lab from http://imagenet.stanford.edu/synset?wnid=n02099712 when it's back up.

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.

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; in the remainder of this section, we discuss 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

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

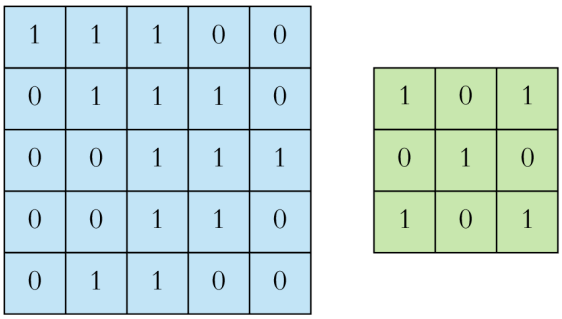


Figure 1.4 An image (blue) and a convolutional filter (green)

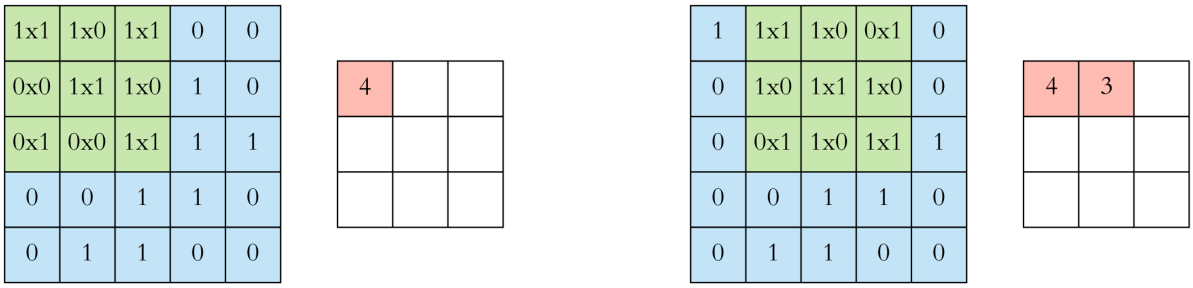


Figure 1.5 The convolution operation consists of the smaller filter (green) applied to each region of the larger image (blue); each application results in a single value which is stored in the feature map.

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

Don't we need to cite these images?

Convolutional layers typically do not hugely decrease the dimensions of the matrix; in many cases, the input image is padded in order to prevent any reduction in dimension. In many neural

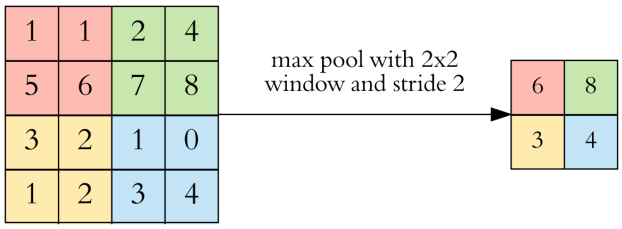


Figure 1.6 Max pooling layer illustration

network structures, dimension reduction is performed by pooling layers, which identify the strongest local features in each filter layer of the convolutional output.

Max Pooling Layers Max-pooling is a technique to reduce the size, and therefore computational load, of feature maps through structured down-sampling. Max-pooling layers apply a maximum function over adjacent regions of a feature map (like using a sliding window) to encode the important information of how strongly a feature was activated in a given region of the image while simultaneously reducing redundant or unneccessary information about smaller activations. 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. Max-pooling is also beneficial in that it allows CNN "vision" to be relatively translation invariant, because it emphasizes the relative position of a feature rather than its absolute position.

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 and max pooling operations discussed above rely on 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

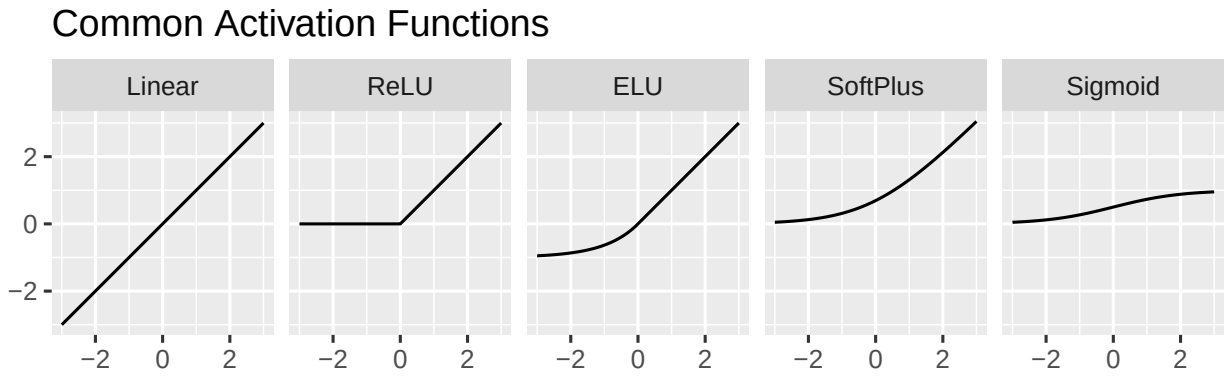


Figure 1.7 Activation functions commonly used in neural networks.

function is commonly used for this purpose. Figure 1.7 shows several common nonlinear activation functions.

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 labels associated with the image. These layers act like the human brain: just as we learned 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". 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 is connected to each label through weights (hence the name "densely connected") learned during the training process. Weights are optimized in order to minimize errors (measured by a loss function) and thus improve classification accuracy.

In some cases, fully connected layers may utilize a pruning mechanism known as a dropout rate, which forces a set percentage of the weights to be zero. This results in the maintenance of only the strongest connections between two layers of the model. I'm not sure this is true? My understanding

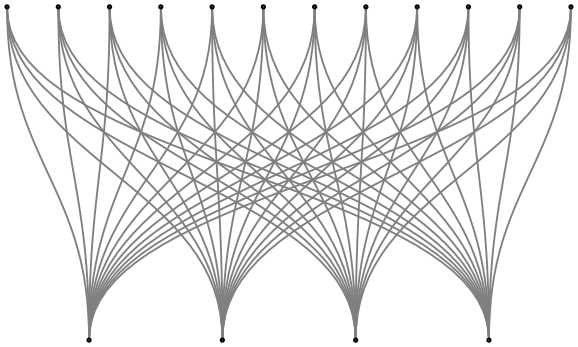


Figure 1.8 A densely-connected layer with 12 input nodes and 4 output nodes.

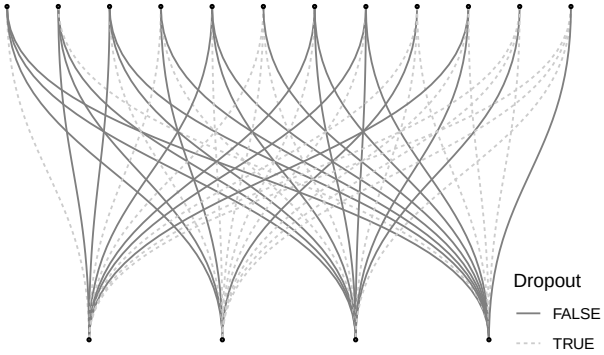


Figure 1.9 A densely-connected layer with 12 input nodes, 4 output nodes, and a 50% dropout rate.

is that it prevents any one node from deciding the prediction for a whole class – like making sure multiple people in an office can do a job in case one is sick.

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

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).

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

$$x_k^{(\ell)} = \sigma^\ell\left(\beta_k^\ell * x^{(\ell-1)} + \gamma^\ell\right) \text{ for convolution} \tag{1.1}$$

$$x_k^{(\ell)} = \sigma^\ell\left(\beta_k^\ell * x^{(\ell-1)} + \gamma^\ell\right) \tag{1.2}$$

$$\tag{1.3}$$

need to figure out how to represent max pool layers and fully connected layers here...

All the equations go in here.

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) and then assign labels to combinations of those features (brain). 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 convnets can have many millions of trainable parameters in the model base (the convolutional and pooling 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). *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. Transfer learning allows CNNs to be applied to smaller datasets of several thousand images, with additional gains in the amount of 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. One of the simpler pre-trained convolutional networks available, VGG16, has a structure which is ideal for our purposes - it consists of several blocks of convolutional layers, with a fully connected head, as shown in Figure 1.11. 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 outcompeted by networks with more complicated structures, such as GoogLeNet/Inception(Szegedy

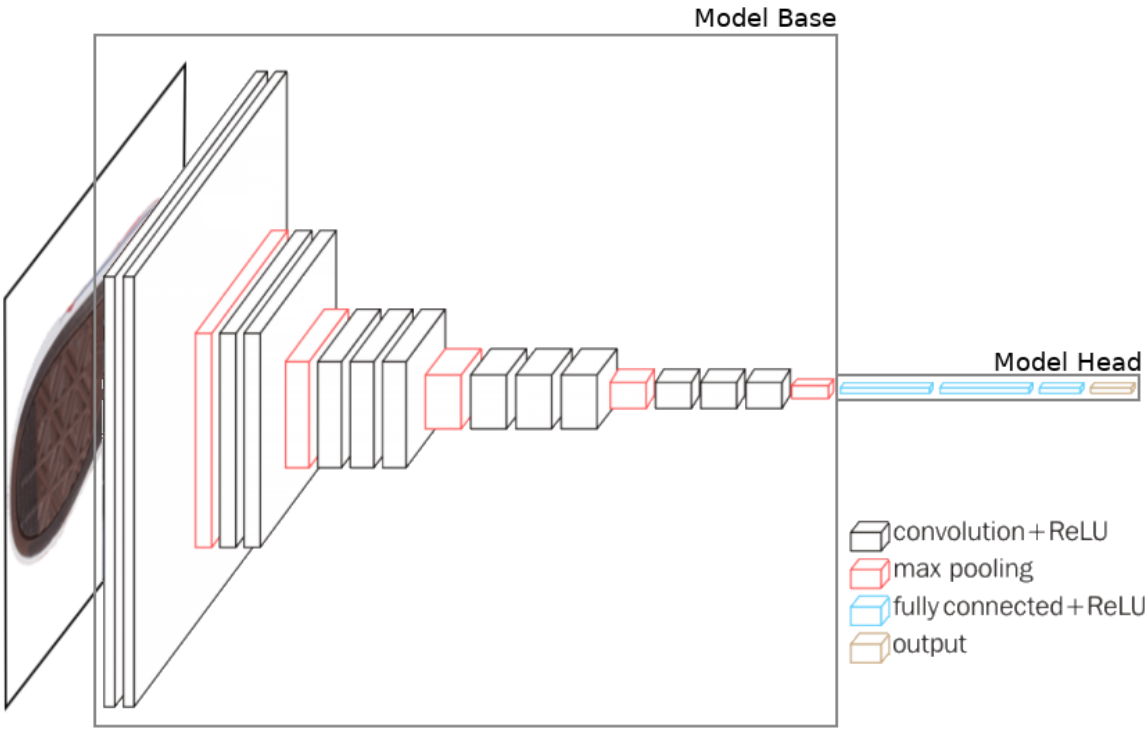


Figure 1.10 This diagram is for a pre-trained convolutional neural network. The model base consists of 5 convolutional blocks; the model head consists of several fully connected layers capped with an activation layer which transforms the aggregate visual input to output class probabilities. During transfer 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.

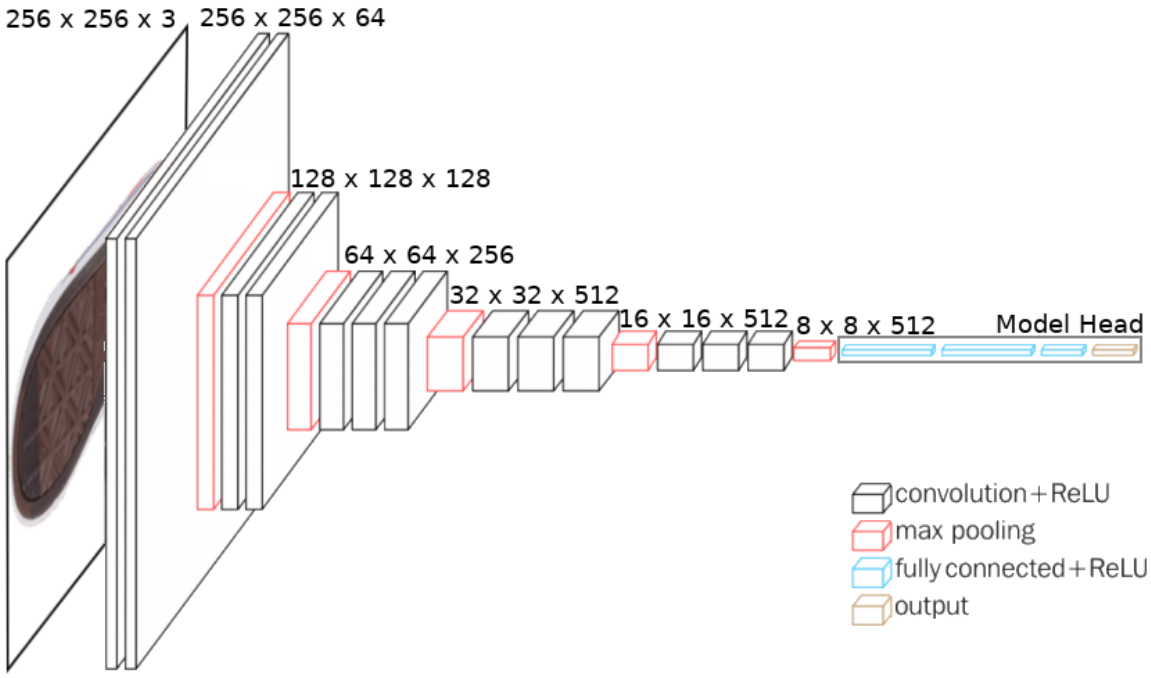


Figure 1.11 The VGG16 model structure, for input images of 256 x 256 pixels. Each convolutional block operates on a different image matrix size; max pooling layers are used to transition between convolutional blocks. There are 5 convolutional blocks in the VGG16 model, which are then connected to output classes using several fully connected layers.

		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.

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).

1.4 Machine Learning Model Evaluation

Classification tasks are considered single-class when there is a only one binary decision of whether an item belongs to a single class of interest, and multi-class when there are a larger number of classes that an object may belong to. Multi-label classification is the special case of multi-class classification where categories are not mutually exclusive , that is, an item may fall into a combination of categories simultaneously. 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.

Evaluating model accuracy on classification 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 sum of all true positives divided by the number of positive cases in the data. Specificity, which is the true negative rate, is the sum of

		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 ommission of the correct label and the addition of an incorrect label.

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.

ROC curves plot the false positive rate (1 - specificity) against the true positive rate. Ideally, we want 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 and 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. XXX - Add a sample ROC curve from ggplot2 to show as an example visualization

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.XXX - this table and last sentance may not be essential and could probably be cut, but serve as a nice tease for the original work done in this CC. XXX - Add a confusion matrix from ggplot2 to show as an example visualization

However, in some situations, it is possible to reduce the classification problem to a series of binary classifications; in these situations, which will be described in more detail in subsection 3.1.2, a modification of the confusion matrix is possible to better visualize model performance.

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, we assembled a set of categories which were 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 our images is made up of latin alphabet charcters; the model will likely not recognize text in other scripts at this point (but could be trained with text in other scripts if such training images could be obtained). Text frequently includes component shapes such as circles; where these shapes are clearly defined, they are labeled as well, as the model does not have the ability at this time to impose the necessary context on images to differentiate an “o” from a circle.

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.

Table 2.1 Geometric Elements. Categories modified from Gross et al. (2013).

2.1.2 Data Collection

Thousands of outsole images were web-scraped from Zappos.com, a large online shoe retailer. These images were then uploaded for use in a tool called LabelMe (cite), a labeling/annotating interface which allows users to easily select and label regions of an image. To date, about [2,200] shoes have been labeled, yielding about [24,000] multi-label images.

After annotation using the LabelMe software package, images are processed by an R script, which 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 images which are relatively square to minimize the effect of this distortion.

2.1.3 Data Characteristics

Describe Figure 2.1 in words. Talk about the fact that the class imbalance means we will have to weight the classes for the model, and that this also affects the utility of accuracy measures.

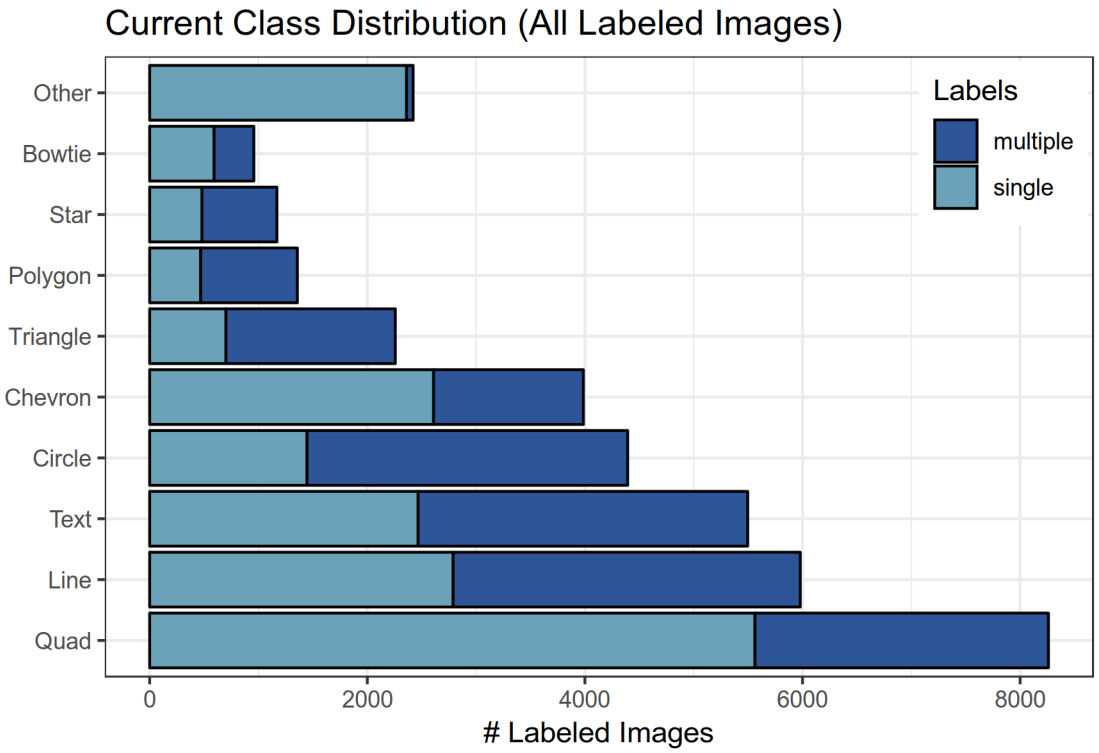


Figure 2.1 Distribution of classes in all labeled images. Quadrilaterals, lines, circles, text, and chevrons are relatively common; stars, polygons, and bowties are relatively rare.

Figure 2.2 Four sets of original (left) and augmented (right) labeled images.

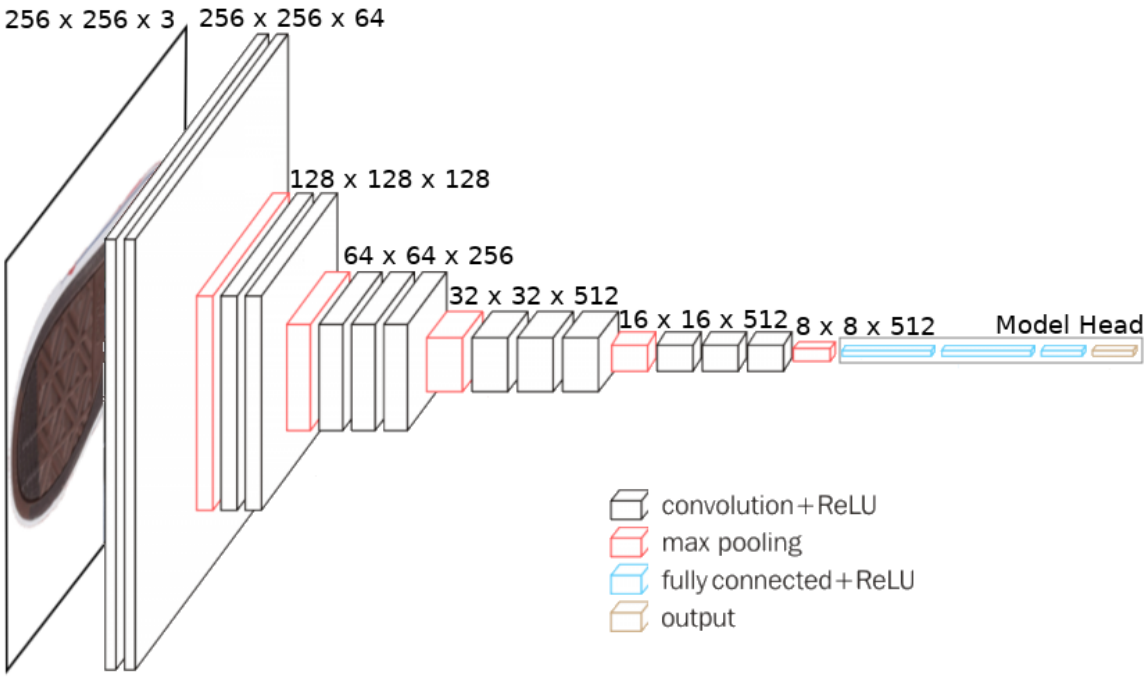
2.1.4 Augmentation

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 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 2.2.

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. 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 early convolutional layers of VGG16 contain 64 filters that primarily detect colors and edge patterns. Later convolutional layers of VGG16, in contrast, contain 512 filters that represent much more complex features, like animal fur patterns or distinct bird heads. The images 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.

Really want to include bird head images here. When will the KerasVis stuff be ready in R? :)

XXX VGG16 follows groups of 2 or 3 convolutional layers with a max-pooling layer, which summarizes the information in the feature maps and scales information down by a factor of 4.

2.2.2 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) backend. 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 we have identified. Once the separate model head was fit, we utilized keras' facilities for loading model weights to combine the convolutional base of VGG16 with the model head into a single, unified model object.

Code is provided in **??**. Using a 48-thread CPU with 128 GB of RAM, the time required to process the LabelMe annotations, generate 256×256 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, 20% 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 section A, but a static example of the weights and image counts is shown in Table 2.2.

class	count	class_proportion
bowtie	1102.00	0.24
chevron	4372.00	0.06
circle	4810.00	0.06
line	6490.00	0.04
polygon	1364.00	0.20
quad	8518.00	0.03
star	1294.00	0.21
text	6144.00	0.04
triangle	2160.00	0.12
total	36254.00	1.00

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.

CHAPTER 3. RESULTS

3.1 Model Evaluation

- Goal: process that helps model predict new data given repeated exposure to training data.

-Examples of prediction

3.1.1 Model Training

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. This point has not yet occurred, indicating that we have halted the model optimization process at an appropriate epoch.

- Model training image (Why over-fitting is dangerous and how we've addressed it with number of epochs. Image should show we stop before it occurs)
- Discuss how accuracy and loss change by epoch

3.1.2 Model Accuracy

-Ways to measure accuracy (TPR, FPR, ROC/AUC) This is in the introduction, you then use those measures and interpret them I think that's what I meant -Interesting case studies Let's get rid of this :) :) :)no, we're just moving the bullet point down from the previous section

3.1.3 Model Consistency

Look at how model predictions for the same feature of different color options for a shoe change. Should be a fun case study - does CoNNOR actually have the robustness we claim it should?

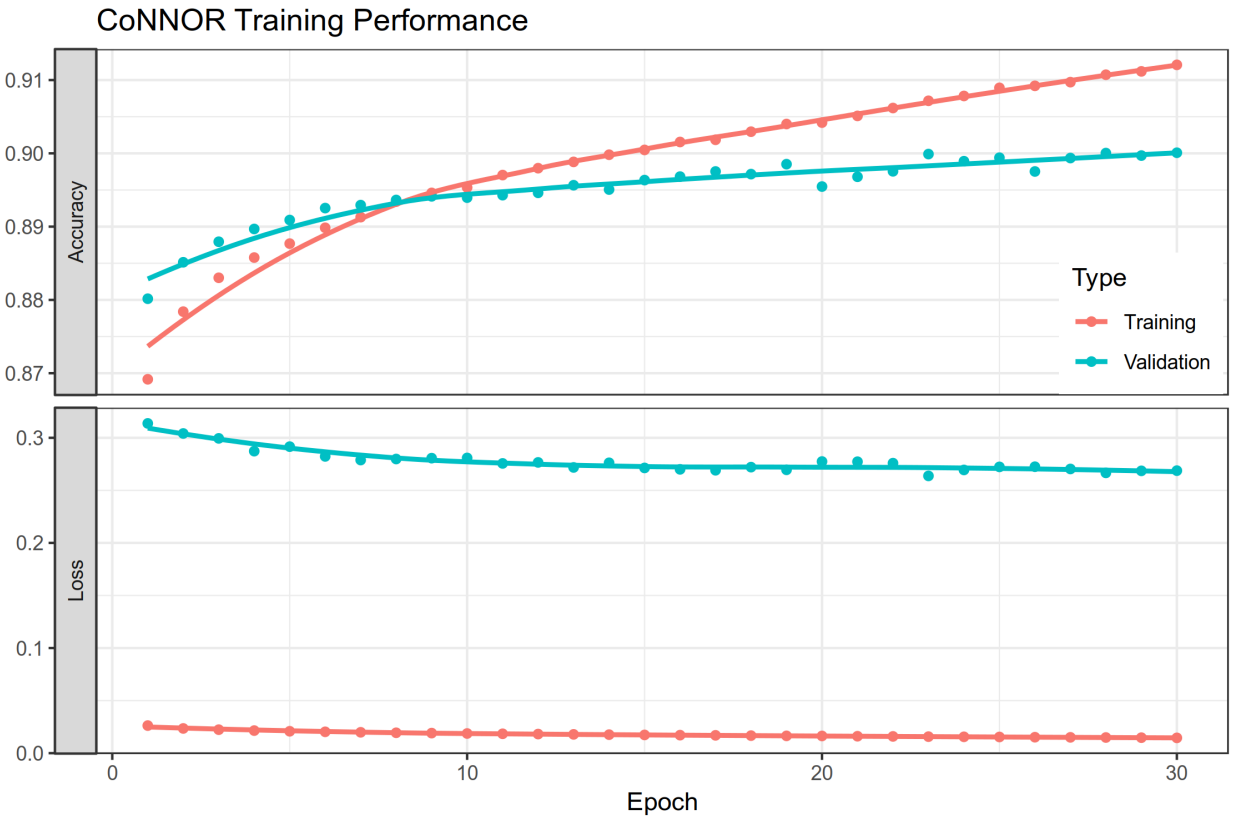


Figure 3.1 Training and validation accuracy and loss for each epoch of the fitting process. Training and validation accuracy reach 89.5% around epoch 9. After that point, validation loss remains the same and training loss decreases slightly, while validation accuracy increases more slowly than training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

3.1.4 Heatmaps - Model Diagnostics

Add the fun stuff in here!

CHAPTER 4. CONCLUSION

I don't heckin' know Write this part last... :) But, we can conclude that CoNNOR works well for identifying features within outsole images, that the geometric feature set is able to classify many common tread elements (and we can compute statistics for how many shoes have recognizable features...)

4.1 Future Work

Integrate features for whole shoes - add spatial information; explore color histogram normalization to increase discrimination power

4.2 Philosophical Conclusions

BIBLIOGRAPHY

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.

Ballard, D. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122. 05541.

Bodziak, W. J. (2000). *Footwear Impression Evidence: Detection, Recovery, and Examination*. CRC Press, Boca Raton, Florida. 00319.

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. 00757.

Geirhos, R., Janssen, D. H. J., Schütt, 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]*.

Gerven, M. v. and Bohte, S. (2017). Editorial: Artificial Neural Networks as Models of Neural Information Processing. *Frontiers in Computational Neuroscience*, 11.

Goldstein, E. B. and Brockmole, J. (2016). *Sensation and perception*. Cengage Learning. 03289.

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.

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

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. 36300.

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 100.

Ramesh Jain, Rangachar Kasturi, B. G. S. (1995). *Machine vision*. McGraw-Hill Science/Engineering/Math, 1 edition.

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.

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. 02191.

APPENDIX A. COMPUTER CODE

Setup

Describe /model directory structure

```
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
}

process_dir <- gsub("[[:punct:]]models[[:punct:]]shoe_nn[[:punct:]]RProcessedImages[[:punct:]]
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)
    base_file_name <- file.path(file_dir, str_remove(basename(filename), "^\\d{1,}\\+"))
    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)
}
```

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 a new 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))
# class_weights <- (sqrt(1/class_proportions) / sum(sqrt(1/class_proportions))) %>%
#   as.list() %>%
```

```
#   set_names(as.character(1:length(class_quantities) - 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"))
```

More stuff

Supplemental material.

APPENDIX B. STATISTICAL RESULTS

This is now the same as any other chapter except that all sectioning levels below the chapter level must begin with the *-form of a sectioning command.

Supplemental Statistics

More stuff.