<Hair-For-Your-Face> Neural Network

Version 1.0

Team Spectre

Kyaw Htet Paing Win
Frankie English
Matthew Berrios
Maddie Powers
Brandon Hoang
Steven Johnson

Section 1: Overview of the document

1.1 Stakeholder's benefit and application use

As the Kano model in the Business Requirement Document highlights, the must-have feature that our customers expect to see from us is the hairstyle recommendation. From what the team has learned, the geometry of the face shape plays a significant role when it comes to implementing this feature.

Hence, the objective of the technical team at this stage in our development of the software is to learn to identify and understand the appropriate approaches of machine learning to deploy to our problem and get exposure to machine learning through implementation.

1.2 Selecting appropriate approaches of machine learning

In researching the different approaches of machine learning to our classification problem at hand, we came across a number of candidates that might be useful. One such approach is the kernel method that essentially tries to solve the classification problem by trying to find the decision boundary with the help of classification algorithms known as support vector machines(SVM). However, this method turns out not to scale well to large datasets and the vision problem that we have.

Another approach is to use decision trees and random forests. We find that although this approach might not be the best for our image classification problem that we are trying to solve, it seems like we may be able to use it later in our development as we add more features to make hairstyle recommendation.

Finally, the approach that we find that is most appropriate for large-scale image classification is to utilize neural networks. As an example, the winner of the image-classification challenge ImageNet had achieved an accuracy of 96.4% in 2015 using a particular type of neural network called convolutional neural network, which we will explore and use to solve our problem.

Section 2: Overview of deep learning

2.1 What is the central task of machine learning and deep learning?

In contrast to classical programming, machine learning systems are presented with many examples or data points that are relevant to the task to find structure in these examples that ultimately allows the system to come up with rules for automating the task without being explicitly programmed. Figure 1.2 below illustrates this point.

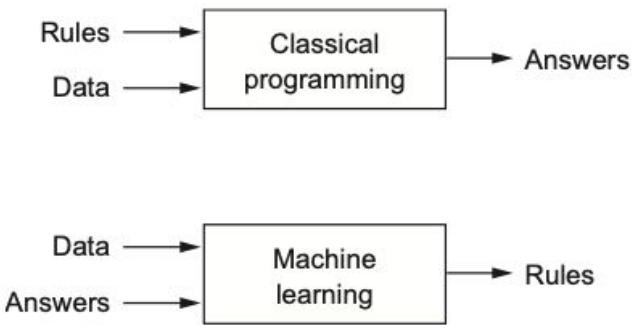


Figure 1.2 Machine learning: a new programming paradigm

Hence, the central task in machine learning and deep learning is to meaningfully transform the data - i.e finding useful representations (which refers to different ways of representing or encoding data) of the input data that is given - to representations that get us closer to the expected output.

2.2 What's needed to do deep learning?

In order to do machine learning, three things are needed:

1. Input data points
2. Examples of the expected output also known as label
3. A way of measuring how well the algorithm is doing. It's important to be able to measure how the algorithm's current output compares to the expected output. This feedback signal allows the algorithm to make adjustments in the way that it works. In this context, this adjustment step is referred to as learning.

2.3 How deep learning works?

Deep learning is a particular subfield of machine learning that puts emphasis on learning representations from data via successive layers of increasingly meaningful representations. These layered representations are learned via models known as neural networks, structured in literal layers stacked on top of each other. The figure 1.6 illustrates this multi-stage way that the network transforms the input digit image into representations that are increasingly informative about the final result (in this case to classify the handwritten digit)

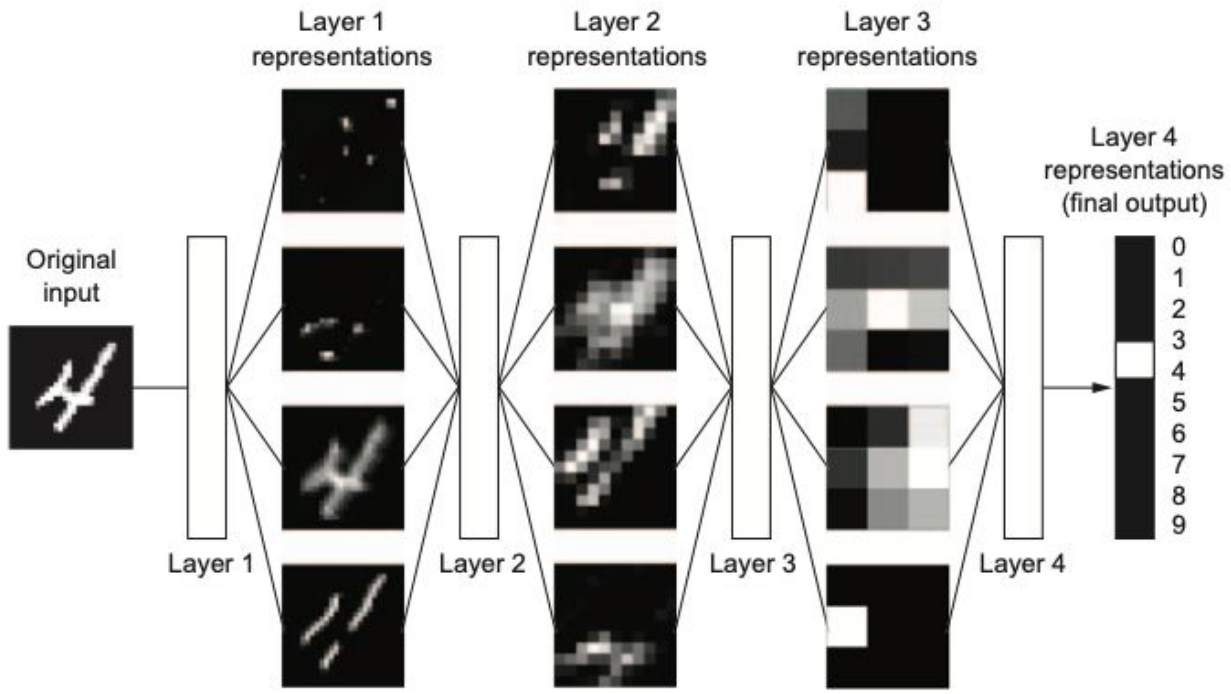


Figure 1.6 Deep representations learned by a digit-classification model

In short, the deep neural networks is mapping the inputs to targets through a deep sequence of simple data transformations, as performed by each layer in the network. How it learns to do these data transformations is through exposure to many examples.

2.4 The anatomy neural network:

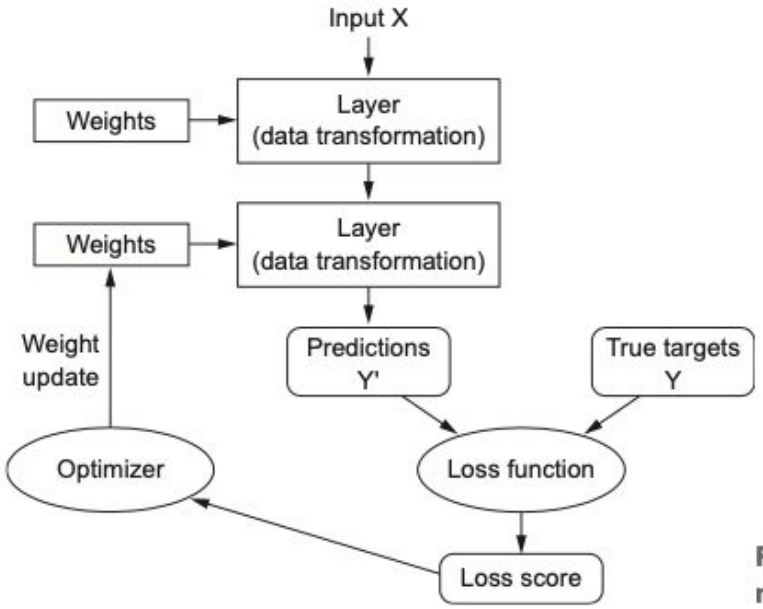


Figure 3.1 Relationship between the network, layers, loss function, and optimizer

As mentioned earlier, each layer performs transformation on the input data. What exactly the layer does to its input data is specified or parameterized by its weights. Then, the goal is to find a set of values for the weights of all layers in a network in order to correctly map the example inputs to their associated targets. Before it could be done, a mechanism is needed to measure how far the output of the network, that is Y', is from the targets, Y. Hence, the loss function (or objective function) takes the predictions of the network and the target to compute a loss score, which capture the network on how well it has done on this example. Then, this score can be used as feedback to adjust the value of the weights a little in a direction that will lower the loss score on this current example. Thus, with every example that the network processes, the weights are adjusted a little in the right direction and the loss score decreases as we will later see in a digital classification model that we will build.

Section 3: Implementation of the neural network

3.1 Goal of this section:

The goal of this section is to get exposure to building the neural network. Here we will build a neural network that can classify grayscale images of handwritten digits into their 10 categories or classes.

3.2 Deep Learning Library

The library that is used for the digital classification is the Keras library. It seems that it is sensible to use Keras for the following reasons:

1. It has a user-friendly API that is easy to prototype deep learning models; hence, appropriate for the beginner
2. Offers built-in support for convolutional neural networks
3. Keras can be used to build any deep-learning model due to its modularity.

3.3 Seven Steps of digital classification and discussion

(The program file that accompanies this tutorial will be included in the submission)

1. Load the MNIST dataset in Keras

The dataset that we use is the MNIST dataset that includes 60,000 training and 10,000 testing grayscale images. The MNIST dataset comes preloaded in Keras as a set of four Numpy arrays. The images encoded as Numpy arrays and labels are an array of digits, from 0 to 9, have a one-one correspondence.

After loading, we visualize the sample data by selecting a specific element of the dataset by tensor slicing. Moreover, we look at the important attribute of the tensor, its shape. Here, the dataset is presented in 3D tensor.

On a side note, we find that we will most likely be using 4D tensors when solving our own problem since an image typically needs to store height, width, and color.

2. Build the network

In this step, we come across a way to define the model in Keras using the Sequential class. The network consists of two Dense layers, which are densely or fully connected neural layers. The second layer is a 10-way softmax layer, that returns an array of 10 probability that the current digit image belongs to one of the 10 digits. The digit with the highest probabilities will be the one that the network will output.

In addition, we learn that each of these layers are applying a few tensor operations such as dot product, addition and relu operation to our input tensor. Furthermore, we saw how to specify the input shape and output shape of each layer that we build and that the input shape of our second layer is built dynamically when none is specified as in the example.

3. Preparing the network for training

In preparing the network for training, three of the following things are specified: loss function, optimizer, and metrics. It uses categorical_crossentropy as the loss function to generate a feedback signal that is used for adjusting the weight tensors. We also learnt that the minimizing of the loss function happens through a learning technique known as mini-batch stochastic gradient descent as defined in rmsprop optimizer. In this case, we use accuracy as our metric that tells us how many times the network correctly predicts the network out of the total number of inputs.

4.Preparing the image data

In this step, we come across another important tensor operation called reshaping. For this example, we configure the data by reshaping it into the shape that the network expects, where all values are in between 0 and 1. In this case, the array of shape (60000, 28,28) of type uint8 that was loaded with values between 0 to 255 will be scaled to a float32 array of shape (60000, 28*28) with values between 0 and 1.

5.Preparing the labels into category

Labels are encoded into categories.

6. Train the network on the training dataset

Train the network on the training dataset by calling the fit() function. We specify that the network should process data in a mini batch of 128 in 5 epochs, and in each iteration computing the gradients of the weights and updating the weights. As expected, we see that the loss of the network has dramatically decreased with each iteration; thus, the accuracy of the training dataset improves as more data are feeded into the network. The model reports an accuracy of 98.9%.

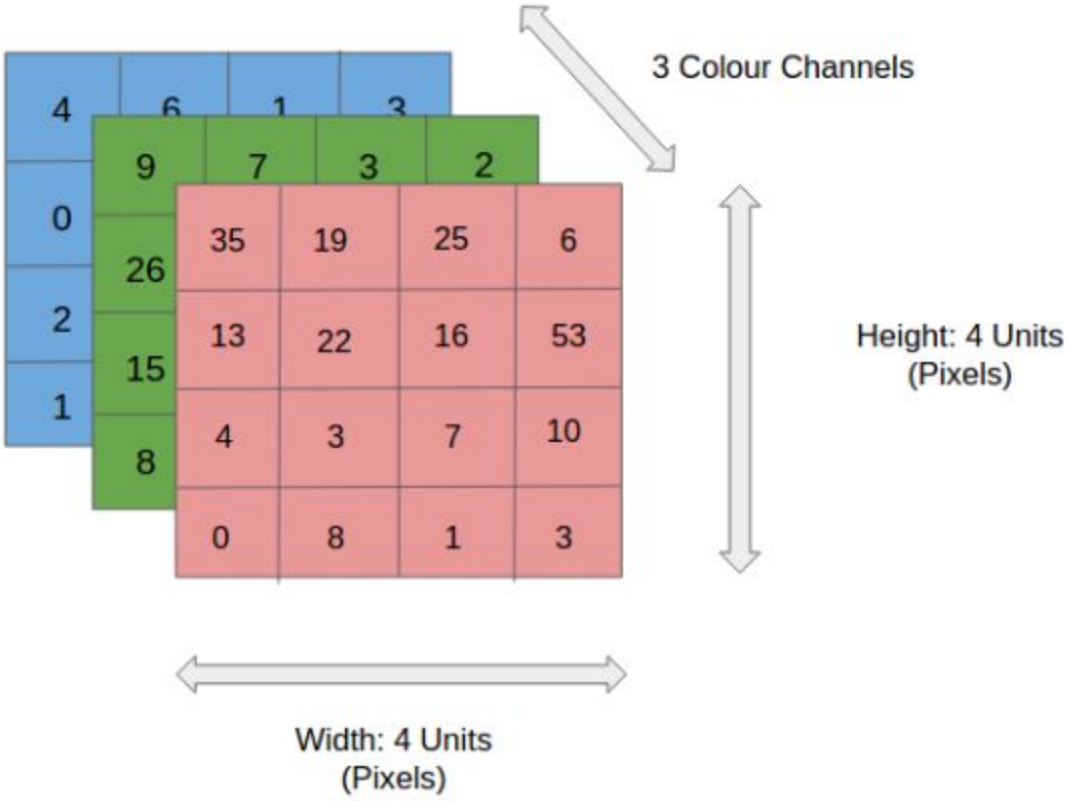
7. Evaluate the network with the test dataset

The network is fed with the test dataset that it has not encountered before, and the reported accuracy of this dataset is 97.8%, a little lower than that of the model, which is an example of overfitting.

In conclusion, this exercise not only gives us the workflow involved in developing neural networks with Keras but also compels us to learn about the fundamental concepts of neural networks such as tensor, tensor operation, gradient descent, and etc. that would allow us to take a more disciplined approach later on.

Section 4: Computer Vision

Computer vision is used in the example above to recognize numeric characters. In this instance the image is 28x28x1 these measurements denote the number of pixels that are defined within the input layer of our neural network. 28x28 pixels is the size of the image x1 is the number of channels being used, in the previous case only one channel was used to represent the black value of the pixel, in other words, how dark the pixel is. This can be easily expanded to work with colored images. The standard way to represent images is with RGB (red, green and blue) channels. These channels combine in a countably infinite number of ways to represent the colors we see on our computer monitors. For a color image the same size as the digits we would have an input size of 28x28x3 where each channel is split and input into the neural network.



Section 5:Convolutional Neural Network

After the image is input into the input layer, the image is processed by a number of hidden layers. Hidden layers are all connected to each other. They do matrix multiplication to locate edges and eventually make an assumption about the image based on the label that the image was given and after each iteration the hidden layers are adjusted to alter the matrices and provide a more accurate output. As with most machine learning, the larger the properly labeled data set, the better the neural network will be.

Section 7: Data Flow

The first step will be to first process the image.
The image must be the appropriate size in order to be accepted as input. This could involve scaling, descaling the image. We cannot skew the image in any way as it can and would affect the subjects face shape and lead to a inaccurately trained model.
The second step will be locating a face.
The measurements should only be taken if an image of a face is detected. The model cannot take measurements on a face if there is no face in the image to begin with. Then we will define the boundaries of the face where the independent local point estimators will be working.
The local point estimators will do their best to determine key points on the face that we can use to calculate one of the six face shapes that are most appropriate for the subject.
Once the algorithm has determined a face shape we are then able to present the user with an array of hairstyles they may then choose from.

Future iterations may include a decision tree that accepts: the subjects face shape, hair type, hair color and complexion, for the purpose of recommending hairstyles and colors.

Section 8: Project Libraries

Python 3.7 - Our language of choice with the most support for machine learning frameworks.
OpenCV - For its powerful image processing tools, such as cropping images and creating visualizations on top of the image to assist in training the model.
TensorFlow - As our primary machine learning framework with the build in Keras library.
Keras - Very high level view of our neural network, to enable ease of understandability by our team and our peers.
Matplotlib - For data visualization specifically to monitor our models performance.

References:

*** All the materials presented from Section 1 to 3, including the code and the diagrams, comes from this amazing book, Deep Learning with Python by François Chollet.

Yue Wu, Qiang Ji. Discriminative Deep Face Shape Model for Facial Point Detection. 9 Feb 2014.

Wisuwat Sunhem, Kitsuchart Pasupa. An Approach to Face Shape Classification for Hairstyle Recommendation. February 14-16, 2016.

Videos:
3Blue1Brown. But what is a Neural Network? Deep Learning, Chapter 1.
https://www.youtube.com/watch?v=aircAruvnKk .

Edureka!. OpenCV Python Tutorial | Creating Face Detection System And Monitor Detector Using OpenCV | Edureka. https://www.youtube.com/watch?v=-ZrDjwXZGxI .

Tensor Flow. Intro to Machine Learning Part 1-4.
https://www.youtube.com/watch?v=KNAWp2S3w94
https://www.youtube.com/watch?v=bemDFpNooA8
https://www.youtube.com/watch?v=x_VrgWTKkiM
https://www.youtube.com/watch?v=u2TjZzNuIy8