S.H.C.J

SneakerApp

Sneaker Recognition

Shahan Rahman • Hasib Islam • Jun Zheng • Carlos Vides

Senior Design Project CSC 59938

Professor George Wolberg

05/19/2020

Table of Contents

I. Introduction *3*

1. Abstract 3
2. Image Recognition 3
3. How Image Recognition Works 3
4. Image Data Pre-Processing Steps for Neural Networks 4
5. Building a Predictive Model for Images with Neural Networks 5
6. Convolutional Neural Networks (CNN) 5
7. Applications of Image Recognition 6
8. Summary of Image Recognition 7

II. Connors Shorten's Algorithm *8*

1. Shorten's Classifier using his data 8
2. Shorten's Classifier using our own data 11

III. Learning Algorithms *13*

1. Transfer Learning Algorithm 13
2. MobileNetV2 13
    1. Depthwise Separable Convolution 14
    2. Linear bottlenecks 16
    3. Inverted residuals 16
    4. Residual Network 17
3. Accuracy 18

IV. Final Algorithm *19*

1. Pivot 19
2. Libraries 19
3. Methods 20

4. **Data Used** 22

5. **Training Phase 1** 22

6. **Finding Learning Rate** 23

7. **Training Phase 2** 24

8. **Data Visualization** 26

9. **Prediction Code** 27

10. **Results** 28

11. **Website** 28

V. Reference 30

VI. Appendix 32

I. Introduction

1. Abstract

The overall goal of this project was to create a sneaker classification model that would predict the sneakers company. We accomplish this by implementing many technologies that allow us to train a 50-layer neural network model. The steps to this final project not only included many failed attempts with different technologies but also making constant modifications to our training dataset. As a byproduct of this work, we also created a frontend using GitHub pages to present our work including all of our course work and files.

Over the course of this project, we've changed directions three times which helped us achieve our goal. Not only did we meet our expectation for accuracy of our model but exceeded the algorithm's correctness. In this report we not only show our steps to the final product but our thought process and research into our problem.

2. Image Recognition

Image recognition can be defined as an ability of software to identify people, places, objects, actions, and writings. The current technology has incorporated Artificial Intelligence neural networks in achieving image recognition goals. It can be used in performing a wide range of operations, including autonomous robots, automated cars, labeling images, and accident avoidance systems. Image recognition requires the incorporation of Deep Machine Learning. However, the performance is considered to be high when using Convolutional Neural Net Processors. The Image Recognition algorithms work best through the use of 3D Models from varied angles; they are trained on using computer learning on pre-labeled pictures. Image Recognition technology has an array of applications in this digital world.

3. How Image Recognition Works

Like the human eye that perceives and interprets image signals through the cortex, the computer image recognition adopts this pattern. The computers recognize images in terms of raster or vector sets, which are then encoded into constructs to form physical features and objects. The vision systems installed in the computers are capable of analyzing the constructs. They begin by simplification of the image and extraction of important information; the data is then organized through feature classification and extraction. The Computer Vision Systems employ algorithms, and other classification approaches in deciding on the image portion or whole of it. An image classifier is a perfect example of Image Recognition Algorithms. It accepts an image input and predicts its contents. The output I produced in terms of classes; hence the algorithm requires training, learning and distinguishing the classes. For instance, to establish an algorithm worth identifying cats, one needs to train neural networks with numerous cat images. Different algorithms can be used in performing various complex activities.

4. Image Data Pre-Processing Steps for Neural Networks

The Image Recognition Algorithms used in neural networks highly rely on the dataset's quality. This includes the images used in the training and testing process of the model. Due to this, it is important to ensure essential considerations are made to guarantee the quality of image recognition results. Some of the essential elements to monitor are below.

- Ensuring Uniform Aspect Ratio – this is determined by the quality of the image. An image that is clear and capturing details assists in the extraction of more information. However, this kind of image requires high computing power and numerous Neural Network Nodes for processing. The neural networks assume a square input approach, which requires a check to confirm whether each image is in square size. The images can be cropped to attain a square ratio without interfering with the interest areas.

- Image Scaling – after ensuring that the images achieve a particular aspect ratio, it is essential to scale the images appropriately. The heights and widths of the images are determined; this may require down-scaling or up-scaling depending on the desired scale of the image.

- Standard Deviation and Mean of the Input – determining the mean using the pixels of each image can develop insights about the image structure. The perturbed images after the determination can be augmented in the avoidance of innate results.

- Image Normalization – the normalization ensures the similarity of data distribution for each data input. The normalization eases convergence during network training. The normalization process involves subtraction of mean from the pixels and dividing with the standard deviation. The pixel numbers need to be positive.

- Dimensionality Reduction – The Red, Green, and Blue channels can be collapsed into a gray-scale channel. Other dimensions can also be considered when the performance of the neural networks is perceived to be invariant or enhance the traceability of the training.

- Data Augmentation – this is a common pre-processing technique. The perturbed images are augmented to improve the results of image recognition. Rotations, scaling, and other transformations are applicable.

5. Building a Predictive Model for Images with Neural Networks

Upon preparation of the training images, there is a need to avail of a system worth processing them. The Artificial Neural Network can process the images. The algorithms in the network have the capability of classifying audio files, images, text, videos, and other formats. The first step involves the extraction of pixel features. After the conversion of the image into numerous features, they can be used in the training of the model. The model can be better trained with the availability of numerous images. The Neural Networks are made of interconnected nodes where each node picks an input data, most preferably an image pixel. The data then passes through simple computation, referred to as an Activation Function. The neurons contain the numerical weight, which affects results. After the results have been generated, they are fed to the next neural layer until a prediction of each pixel is determined. The figure below shows the layers in the process.

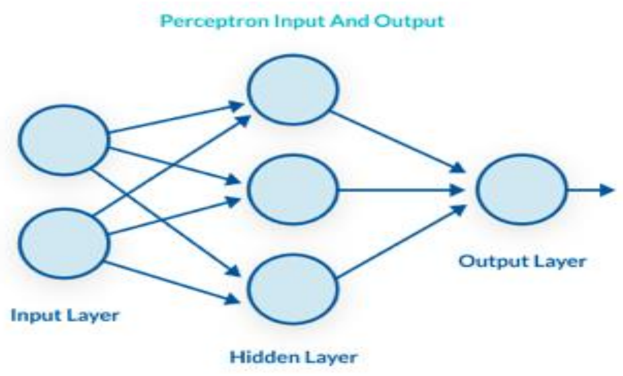


Figure 1: Layers in Building a Predictive Model for Images with Neural Networks

This process repeats until the available images are over. The network learns the weight of the neurons providing accurate predictions through backpropagation. After the model has been trained, it is then applied to the new image set that was not used in training. A validation test is carried out to determine the accuracy. After the accuracy of the model has been fully proved, it becomes applicable and useful in real-world image classification.

6. Convolutional Neural Networks (CNN)

The Convolutional Neural Network is different from the connected neural networks in how its neurons are not connected from one layer to another. CNN employs a three-dimensional approach allowing the neuron sets to analyze a particular region. The connections in CNN are measured in proximity, hence allowing computational training. The neuron groups focus on particular parts of the image. For instance, in an image of a dog, the neurons may focus on the head or other body parts like the foot. The algorithms use segmentation in the evaluation and analysis of the images. The output of this process

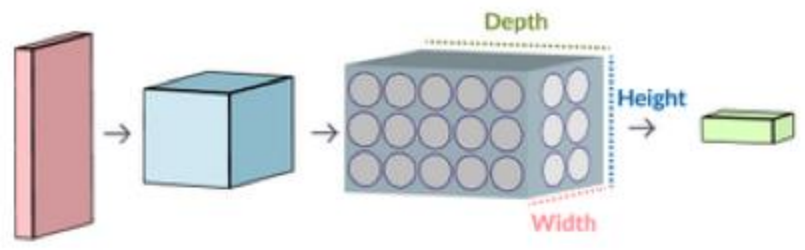


Figure 2: Three-Dimensional CNN structure

can be presented as probability vectors, which can predict for each image feature and its belongingness to the categories. Below is a presentation of the CNN structure.

CNN allows for the prediction of faces and objects with 95% accuracy. This rate of accuracy is higher than human capabilities. However, they are limited in their demand for power. The training process uses heavy machines and hence consumes a lot of power. CNN is liable to failure upon tilting or rotated. The images need to be positioned in the exact position as it was during training. This has led to the development of CAPSNet.

7. Applications of Image Recognition

Image Recognition (IR) technology can be implemented in numerous aspects. The areas of application include face recognition, surveillance, gesture recognition, visual geolocation, medical image analysis, image tagging, and driver assistance, among others. Image Recognition has advanced to video, photo, and faces and used by companies such as Google, Facebook, and YouTube, among others. Other uses of the Image technology include;

- **Automotive Industry** – The technology is being tested in the development of autonomous vehicles. They are installed as sensors to detect objects on the way and avoid accidents. Image Recognition is trained in identifying road objects, people, vehicles, pathways, and obeying road signs and traffic lights.

- **Game industry –** Image recognition has been applied in transposing a digital layer on the real-world images. The augmented reality has been used in addition to detailed layers to the existing ones.

- **Education –** It has been used in assisting students with learning difficulties. The application of Computer vision presents an opportunity for text-to-speech or image-to-speech, which helps students with vision challenges in studying.

- **Manufacturing industry** – Image recognition has been highly involved in the manufacturing cycle. It has been applied in reducing the manufacturing defects. For instance, keeping metadata images allows for efficiency in defect identification.

8. Summary of Image Recognition

Image recognition is an essential technology in this generation. The process can be achieved using Convolutional Neural Networks compared to connected neural networks. There is a set of processes that should be observed before the processing. It is essential to ensure the uniformity of the image aspect ratio. The images also need to be scaled to meet the desired levels. Augmentation is an important process in stabilizing the perturbed images. The neural networks are used in the classification of the pixels in an image until the formation of a predictive model. Despite CNN being a useful Neural Network framework, it requires high energy to be used. Image recognition has managed to earn a pool of applications, including manufacturing, game, and education, among others.

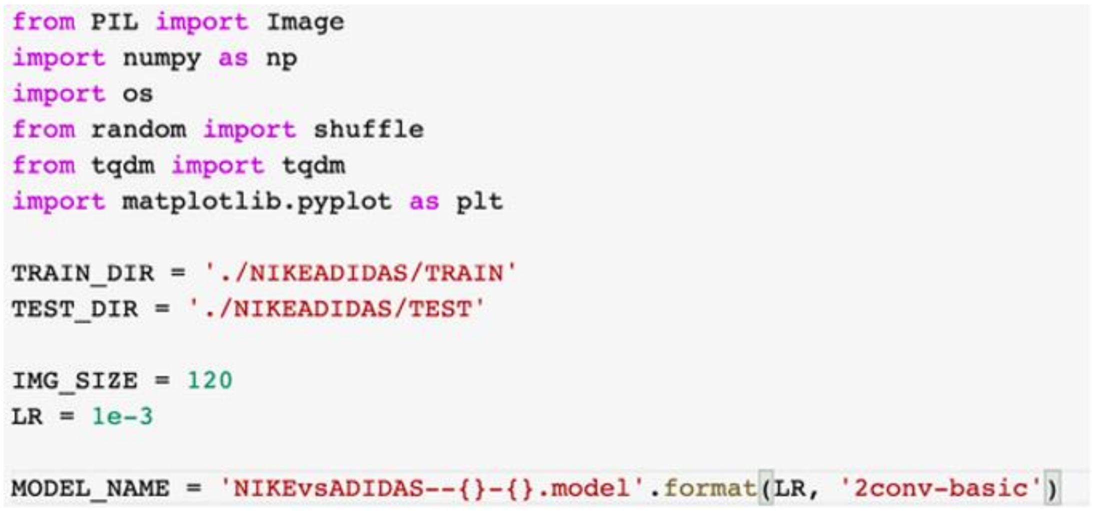
II. Connors Shorten's Algorithm

On the start of our project, we were tasked to look for competitors that were out in the market, and that resembled our project. One of those projects that we found were Connor Shorten's Convolutional Network Classifier.

1. Shorten's Classifier using his data

Shorten's classifier sought to distinguish and classify two types of brands shoes, these being Nike & Adidas basketball shoes. Shorten used 140 images in his dataset and was able to achieve a 91% accuracy classifying both types of shoes using a 4-layer convolutional network which is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. This would be helpful to us, as the objective of our project is similar to his work, trying to identify still images of sneakers and verifying the brand of the shoes plus its names (at the beginning phase of our project).

Therefore, our group decided to run Shorten's classifier on our computers. These are the dependencies needed to run the program on Google Colab, other than TensorFlow and tlearn:



```
from PIL import Image
import numpy as np
import os
from random import shuffle
from tqdm import tqdm
import matplotlib.pyplot as plt

TRAIN_DIR = './NIKEADIDAS/TRAIN'
TEST_DIR = './NIKEADIDAS/TEST'

IMG_SIZE = 120
LR = 1e-3

MODEL_NAME = 'NIKEvsADIDAS--{}-{}.model'.format(LR, '2conv-basic')
```

Figure 3 : Shorten's Dependencies

- PIL is used to pre-process and load images.
- Numpy is used to store our image data in arrays and feed them to our model.
- os is used to interface with our file system.
- random is used to shuffle the data so we don't have a dataset that consists of 20 straight Nike images, then 20 straight adidas images.
- tqdm is used to provide a loading bar as you load in data.

- matplotlib. pyplot is used to visualize our errors and images.

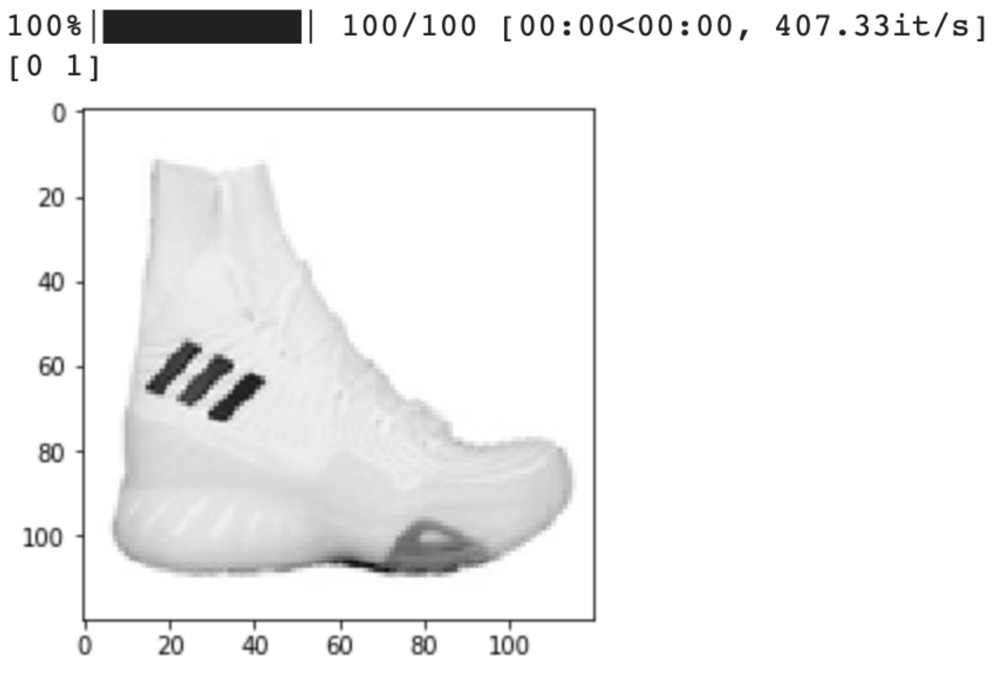
These dependencies will be used between modules to rely on one another. Also, variables are set with the relative path to where the train data and test data are located. The LR, which represents the learning rate is set to 1e-3 or .001, which is a hyper-parameter of how much control we are adjusting the weights of our network with respect to the loss gradient.

We will quickly go over Shorten's different pieces of codes, as to have a clear understanding of our program and at the end, use our own images and see how our results compare to his given that we have more extreme cases.

1. 2 different functions to load the training data and test data. With these functions we end up with an array that has a matrix and a vector. The formatting of these Tensors is one of the incredible features provided to us by the NumPy library.

2. A function to label the images

3. A function to create the training data, using matplotlib, the image is below having a 120x120 pixels size, and although it looks distorted, it is still distinguishable by the 3 dashes of Adidas brand



```
100%|██████████| 100/100 [00:00<00:00, 407.33it/s]
[0 1]
```

Figure 4: Image plot of training data

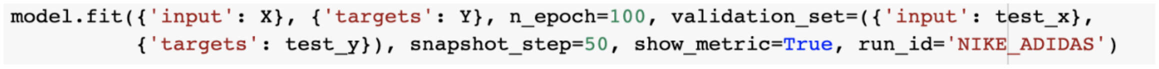
4. A function of the CNN (convolutional neural network) which uses as dependencies TensorFlow and $tlearn$

   a) TensorFlow is a machine learning framework created by Google used to design, build, and train deep learning models

b) $Tflearn$ is a library that wraps a lot of new TensorFlow API's in order to facilitate and speed-up experimentations

5. A function for data splitting, this is used to set aside a small percentage of data as a validation set as well as do parameter tuning before evaluation on our data set.

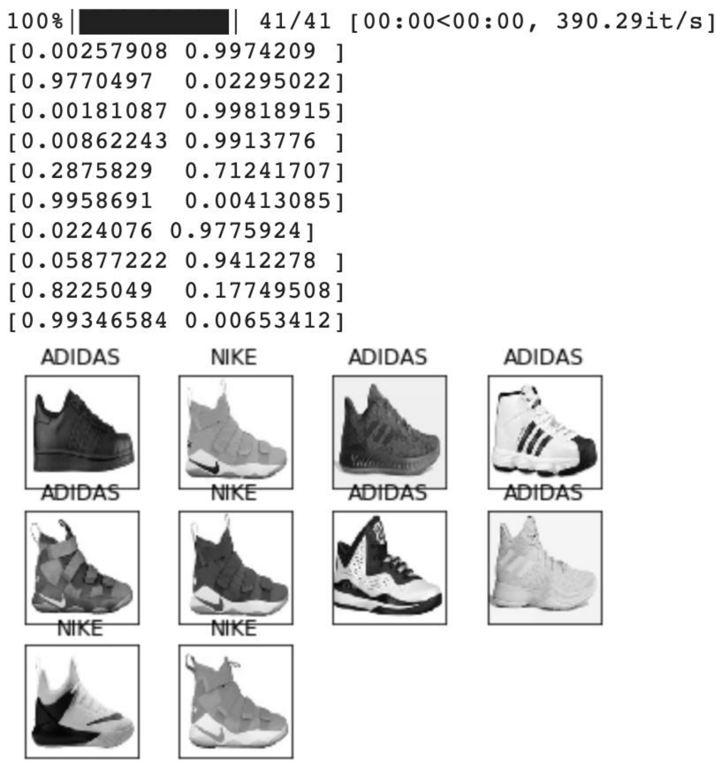
6. Afterwards we implement a function to fit the model and train the model using $fit()$ method, we make 100 full iterations over the sample of our training data, this is represented by the number of epochs.



```
model.fit({'input': X}, {'targets': Y}, n_epoch=100, validation_set=({'input': test_x},
          {'targets': test_y}), snapshot_step=50, show_metric=True, run_id='NIKE_ADIDAS')
```

Figure 5: Code used to fit the model

7. Finally, we are able to upload the test data to get a sense of how our model data is performing. In the results below only one of the choices is wrongly misclassified. There is a total of 10 shoes used for testing and for the first shoe there's a possibility of 99.7% being an Adidas shoe and only .2% being Nike.

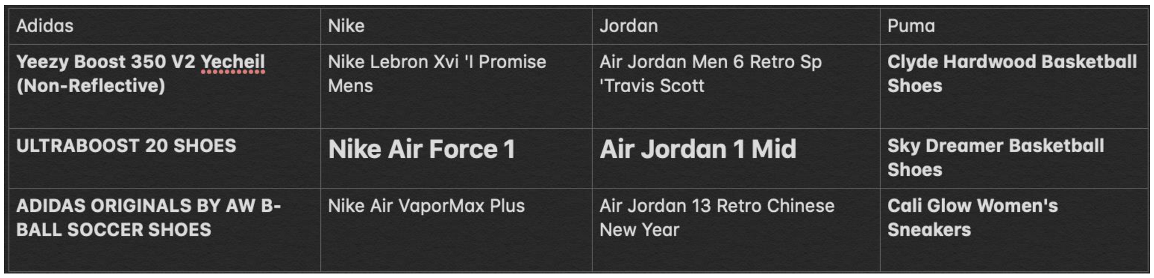


```
100%|██████████| 41/41 [00:00<00:00, 390.29it/s]
[0.00257908 0.9974209 ]
[0.9770497  0.02295022]
[0.00181087 0.99818915]
[0.00862243 0.9913776 ]
[0.2875829  0.71241707]
[0.9958691  0.00413085]
[0.0224076 0.9775924]
[0.05877222 0.9412278 ]
[0.8225049  0.17749508]
[0.99346584 0.00653412]
```

Figure 6 : Test data showing probability of each image (Shorten's Data)

2. Shorten’s Classifier using our own data

After finishing on implementing Shorten’s classifier and fully understanding how his program worked, we sough that it’d be useful to implement the same program using our own data.



Adidas	Nike	Jordan	Puma
Yeezy Boost 350 V2 Yecheil (Non-Reflective)	Nike Lebron Xvi 'I Promise Mens	Air Jordan Men 6 Retro Sp 'Travis Scott	**Clyde Hardwood Basketball Shoes**
ULTRABOOST 20 SHOES	**Nike Air Force 1**	**Air Jordan 1 Mid**	**Sky Dreamer Basketball Shoes**
ADIDAS ORIGINALS BY AW B-BALL SOCCER SHOES	Nike Air VaporMax Plus	Air Jordan 13 Retro Chinese New Year	**Cali Glow Women's Sneakers**

Figure 7: List of shoes classification (beginning phase)

Some problems that immediately emerged, where that Shorten’s work only allowed for 2 different brands and we wanted to capture many, or at most 4 different brands. We started on looking the data through different resources such as Google images. Then we came up with this brands, and names of shoes that we wanted to classify. For testing purpose with Shorten’s classifier we only used Nike and Adidas images.

After running Shorten’s classifier on our dataset we ran into difficulties as the results of our data showed a 49-50% accuracy. We believed this was caused because the images used by Connor were from stock images with clear backgrounds, and still images, and ours were not.

```
[0.50575125 0.49424872]
[0.50681233 0.4931877 ]
[0.5055083  0.49449167]
[0.50644284 0.4935571 ]
[0.50641346 0.4935866 ]
[0.5051054  0.49489465]
[0.50614613 0.4938538 ]
[0.5061658  0.49383417]
[0.5058904  0.49410963]
[0.505907   0.49409303]
```

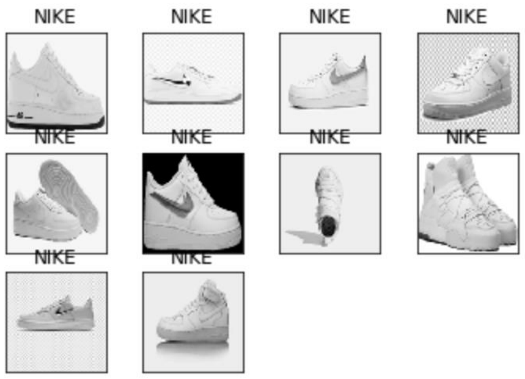


Figure 8: Test data showing probability of each image (Group’s Data)

We had a problem which were that our shoes had many “extreme-cases” this being we had a lot of shoes with different height sizes, this will make our pictures become very blurry and pixelated after the images were resized to 120x120 pixels. Also, some of our images were images with people who were wearing the shoes, this will make the learning process be tougher on our algorithm.

We had many extreme cases or outliers, which is an observation point that is distant from other observations. Extreme values can be present in both dependent & independent variables, in the case of supervised learning methods. Outliers increase the weights of misclassified points on every iteration and therefore might put high weights on these outliers as they tend to be often misclassified. Therefore, a solution for our data needed to be implemented, to overcome this issue we decided on make some changes on the data like getting clearer pictures with no background, that way we’ll still be able to use the same model.

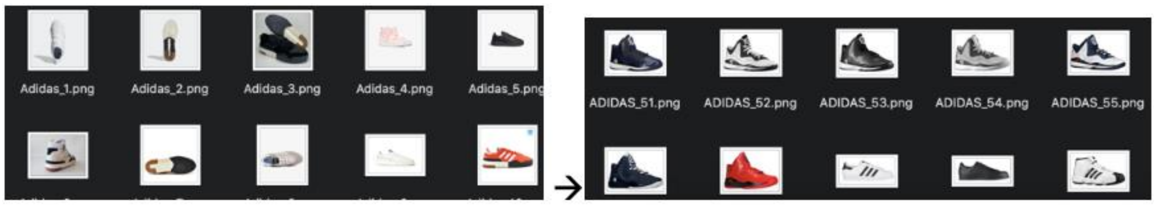
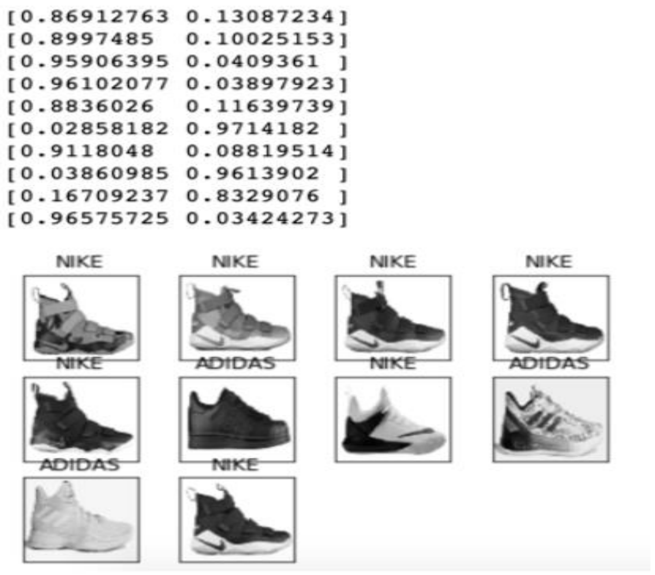


Figure 9: Training data figures

After cleaning our data, meaning we removed those extreme cases and added more still, same size, clearer background pictures, and were able to achieve a higher success rate.



```
[0.86912763 0.13087234]
[0.8997485  0.10025153]
[0.95906395 0.0409361 ]
[0.96102077 0.03897923]
[0.8836026  0.11639739]
[0.02858182 0.9714182 ]
[0.9118048  0.08819514]
[0.03860985 0.9613902 ]
[0.16709237 0.8329076 ]
[0.96575725 0.03424273]
```

Figure 10: Final test data (Group’s Data)

After the script ran we can with an accuracy of 86% similar to Shorten’s classifier, therefore we learned that our data had many outlier, thus we were satisfied with our results but we needed a new algorithm, of our own to classify our data, instead of using Shorten’s.

III. Learning Algorithms

1. Transfer Learning Algorithm

Transfer learning is one of the machine learning methods. It can reuse the model developed for one task in another different task and serve as the starting point for another task model. Consider that most of the data or tasks are related, we can reuse the learned model parameters, which is the “knowledge” learned by the model. We transfer that to the beginning of our own machine learning model; it can speed up and optimize. Our own model does not need to learn from zero. Basically, it’s used the knowledge we already know to learn new knowledge, as our humans did. For example, I already know how to write C++ code, I can learn Java easier and faster than other people based on the knowledge about C++.

Transfer learning is very suitable for our project. Because we do not have too much data to train our model. So, if we can find a model already trained for similar images, then we can use that model to help us train our own model. The most important work of transfer learning is to find the appropriate “knowledge” from the existing model transfer to our own model. For example, if we transfer the knowledge about English to learn Spanish is suitable, but if we transfer the knowledge about English to learn programing language such as C++, we will feel very painful

2. MobileNetV2

At the beginning of this project, we want to develop an iOS application for running our machine learning model. MobileNetV2 is one of the Keras Applications, which is a deep learning model that is made available alongside pre-trained weights on ImageNet. MobileNetV2 is an improved version of MobileNetV1. They both are light weight CNN networks applied on mobile applications proposed by Google. MobileNetV2 has better performance and a more accurate recognition rate.

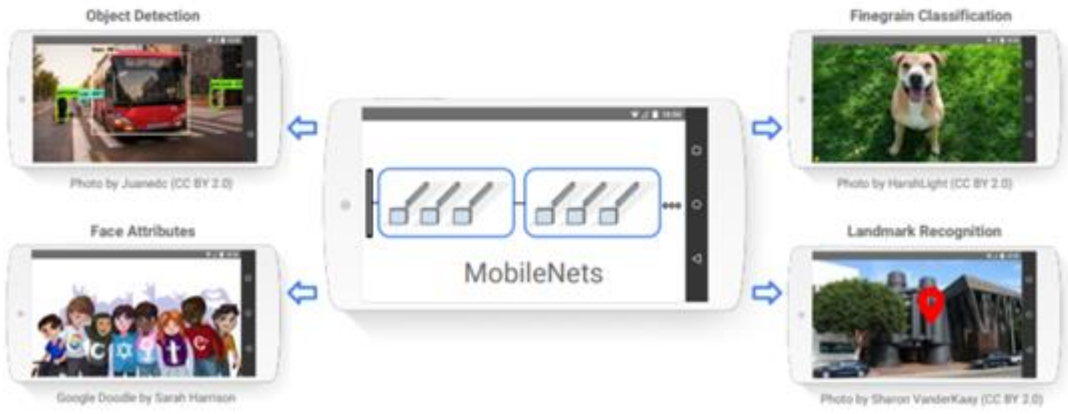

Object Detection
Photo by Juanedc (CC BY 2.0)
Face Attributes
Google Doodle by Sarah Harrison
MobileNets
Finegrain Classification
Photo by HarshLight (CC BY 2.0)
Landmark Recognition
Photo by Sharon VanderKaay (CC BY 2.0)

Figure 11: MobileNetV2

Before we talk about MobileNetV2, let us take a look at the MobileNetV1. MobileNetV1 is a small and efficient CNN model proposed by Google in 2017. Most CNN models pursue classification accuracy, the depth of the model is getting deeper and the model complexity is getting higher and higher. When we want to use the mobile device, a large and complex model is difficult to apply. We need a fast and light-weight CNN model for our iOS application. MobileNet is based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks.

1. Depthwise Separable Convolution

The base unit of the MobileNet is depthwise separable convolution. The depthwise separable convolution is combined by the depthwise convolution and pointwise convolution. As figure 12 shows, the standard convolution applies a different kernel with different image channels and does it all together, which will make the operation parameters larger. The depthwise separable convolution is separated as two convolutions, each convolution taking care of a different part of the image. The depthwise convolution uses the same kernel to all channels. Then the pointwise convolution applies a *1 x 1* kernel to the result of the depthwise convolution. This makes the operation parameters smaller and training faster and get the same result with the standard convolution.

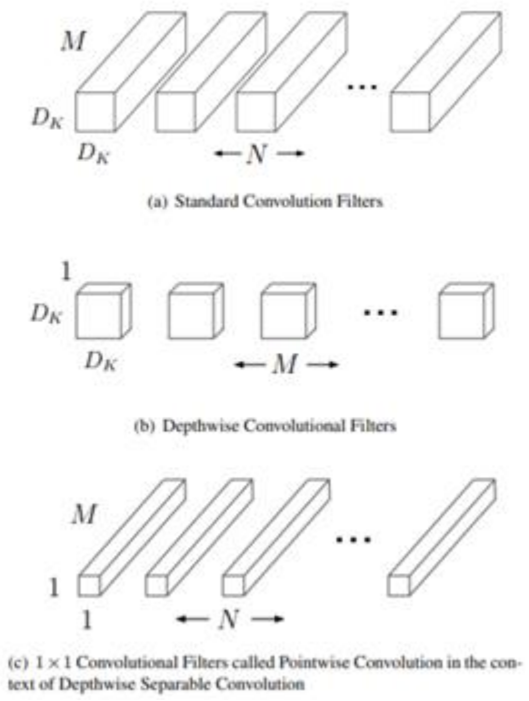

M
D_K
D_K
$\leftarrow N \rightarrow$
(a) Standard Convolution Filters
1
D_K
D_K
$\leftarrow M \rightarrow$
(b) Depthwise Convolutional Filters
M
1
1
$\leftarrow N \rightarrow$
(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 12: Depthwise Separable Convolution

For example, if our data is an image with three channels as a normal RGB image. We want to use a convolution layer to calculate four feature maps. The convolution layer must have 4 filters. In each filter, there has one $3 x 3$ kernel for each channel of the image, the total has 12 kernels. Therefore, the total convolution operation parameter is $12 x 9 = 108$. on the other side, the depthwise convolution only takes care of the "depth" part. So,

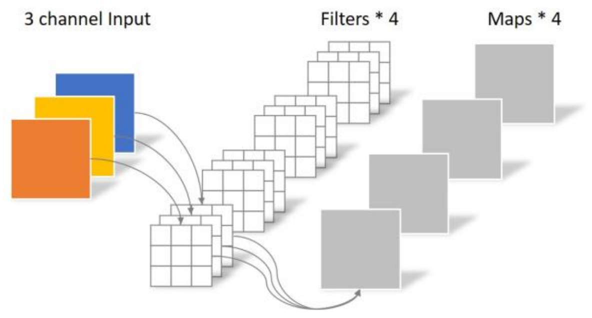


Figure 15: Standard convolution

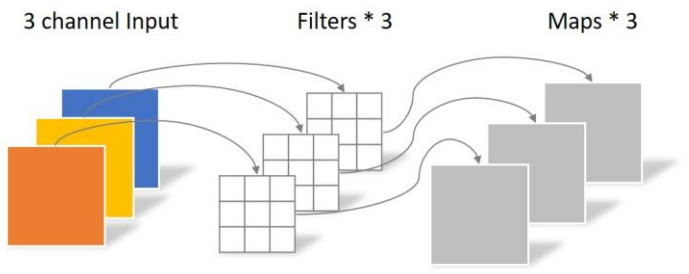


Figure 14: Depthwise convolution

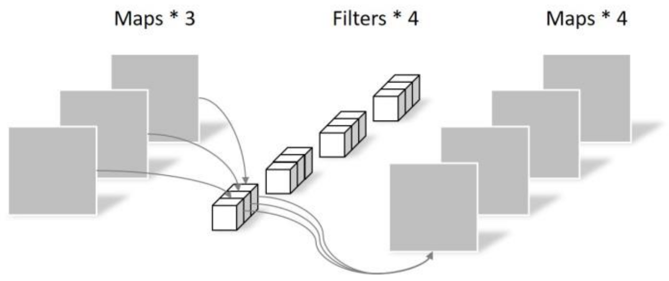


Figure 13: Pointwise convolution

there only has three $3 x 3$ kernels for each channel of the image. And it will output three feature maps as input to the pointwise convolution. The pointwise convolution takes the input, then apply four $1 x 1$ kernel. In the end, depthwise separable convolution will output four feature maps as the standard way we use the convolution layer, but the total convolution operation parameters are $3 x 9 + 3 x 4 x 1 = 39$.

2. Linear bottlenecks

Even though the MobileNet makes training faster and uses fewer computation resources, it has the same defect. The first one is we need to add a batch norm layer and a ReLU6 nonlinearity to narrow down the size of the image for next convolution and the

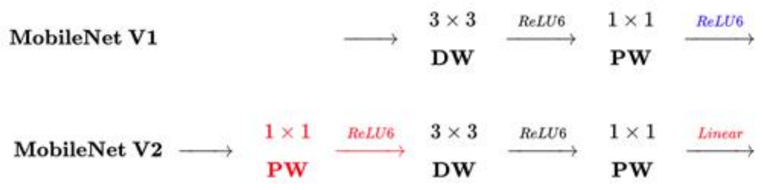


Figure 16: Linear bottlenecks

ReLU6 nonlinearity will make some features disappear. So, the MobileNetV2 add one *1 x 1* kernel pointwise convolution before the *3 x 3* depthwise convolution and change the last layer of the depthwise block as a linear instead of ReLU6 nonlinearity as the figure 16 showed.

3. Inverted residuals

The *1 x 1* kernel pointwise convolution has another feature that is to make the inverted residuals. It is the opposite way to impermeant residual block. The standard residual block is using a *1 x 1* convolution to make the number of channels smaller. Then add and regular convolution. After then, use another *1 x 1* convolution to make the number of channels back to the same as the beginning. The inverted residual block is using the *1 x 1* convolution to make the number of channels bigger which will make the depthwise convolution get more information. Then at the end-use the *1 x 1* convolution to make the number of channels smaller for change back to the same number of channels as the beginning.

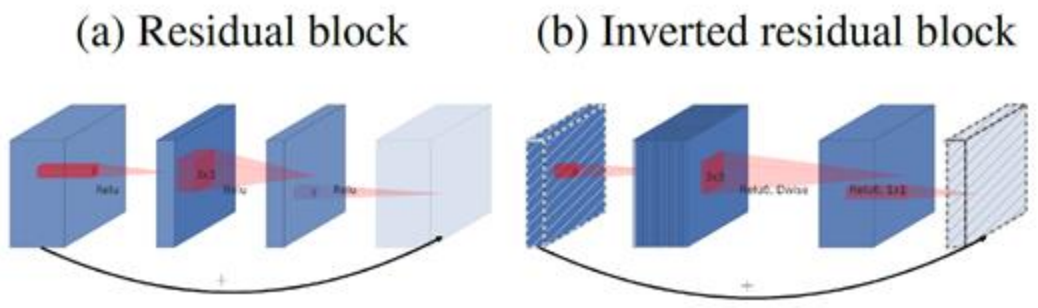


Figure 17: Inverted residual block

4. Residual Network

Since the MobileNetV2 uses the inverted residual block, which means it is one kind of residual net, it just has a little difference with the standard residual net. Why does the second version of MobileNet use a residual block? It is because the residual block will make the model increase the amount of layer without lost accuracy. In the section of Linear bottlenecks, we know that the ReLU nonlinearity will make some features disappear, and the experiment approves this theory. As the figure 18 is shown, if we just simply stack layers from 20 layers to 56 layers, both training error and test error goes larger. Which means the accuracy goes lower, that is the reason for using residual Net.

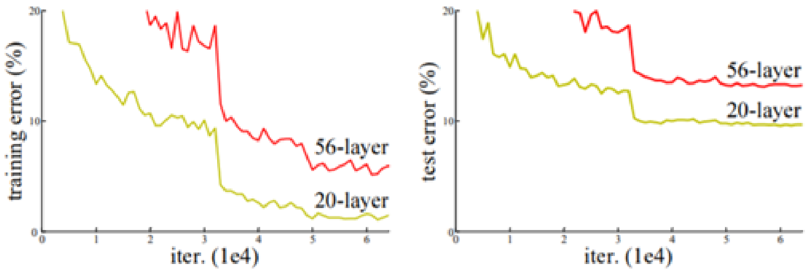


Figure 18: Training error for simple stack layer

In the standard residuals Net, there are two kinds of residuals blocks to deal with different situations. The first one is combined with two 3 x 3 convolution layer, the second one is put one 3 x 3 convolution layer into two 1 x 1 convolution layers as figure 20 showed. The difference between simply plain layer and the residual block is in the plain layer, each convolution layer takes the output from the last layer and calculates an output base on the input. That is the reason for something disappearing. The residuals block will save the output from the last residuals block and add to the end of the residuals block; therefore, the output of the residuals block is combined both the output of this residuals block and the last residuals block.

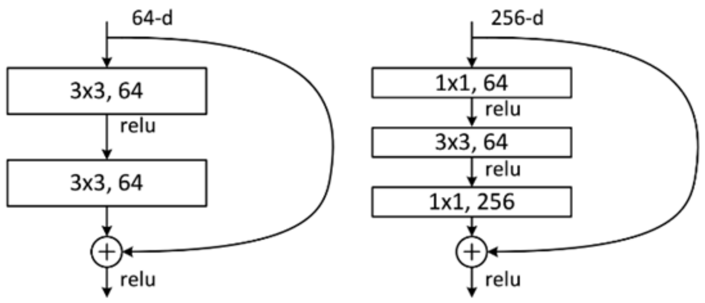


Figure 19: Residual network

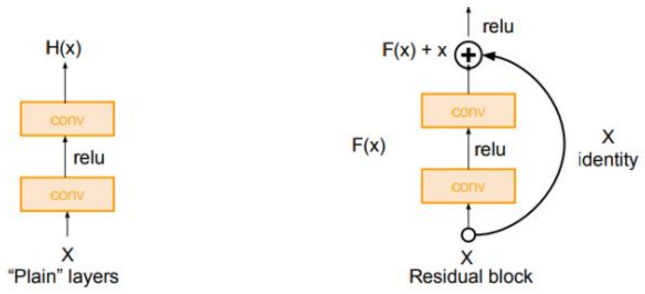
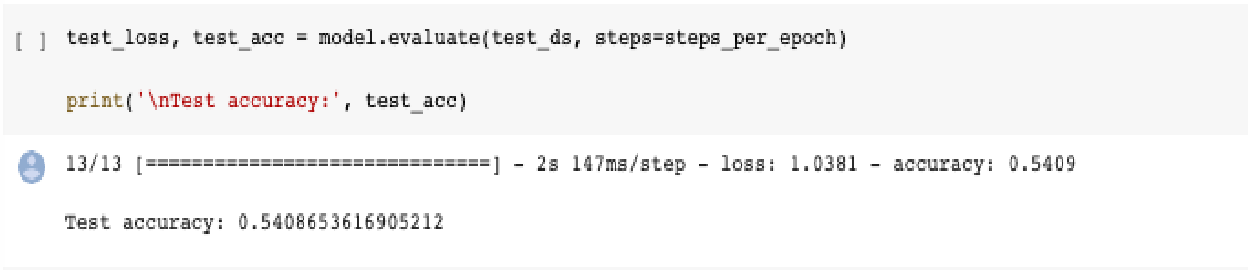


Figure 20: "Plain" layers vs. Residual block

3. Accuracy

Accuracy is our first priority. In this project, we want to get the accuracy rate as high as possible. The accuracy of Connor's classifier algorithm is 91%, so our first target is to use our own algorithm to reach the same accuracy as Connor. However, the accuracy of our algorithm is only around 54% as the figure 21 shown. We cleaned our data set, changed the algorithm, and added more data to our data set. After we did all we can do, the accuracy is still less than 80%. So, before the last meeting, we decided to lower our goal. We decided to use the last few weeks to do anything we can to reach 80%. We are lucky that before the final presentation we find a major bug in our program, and we fix it. Therefore, at the end of this project, we got 92% accuracy. That is not just to reach our goal, it is also to reach the accuracy of Connor's classifier algorithm.



```
[ ] test_loss, test_acc = model.evaluate(test_ds, steps=steps_per_epoch)

    print('\nTest accuracy:', test_acc)
```

```
13/13 [==============================] - 2s 147ms/step - loss: 1.0381 - accuracy: 0.5409

Test accuracy: 0.5408653616905212
```

Figure 21: Accuracy

IV. Final Algorithm

1. Pivot

During the last two meetings our group noticed the challenges with separating our data into individual sneakers. Since we started off with 4 companies and 3 shoes respecitly for each, our data was not only marginalized 12 times, but our model had less training data for each. After meeting with the team, we decided to not only make our model use sneaker companies but also take out the Jordan brand because the test images were inconsistent and had too many variations to have the model accurate. This pivot not only allowed us to be more consistent with our data but allowed us to use more of our extreme photos of certain company shoes since our data was being combined.

2. Libraries

The libraries used for this project are below with a brief description:

- ***TensorFlow***

  TensorFlow is a free and open-source platform for machine learning. Which is used for machine learning applications such as neural networks. It was developed by the Google Brain team for internal Google use. The initial version was released on November 9, 2015, the most recent stable version is 2.1.0 which is what I use in this project. This library provides functions for decode image, transfer image to data for machine learning, parking data to a dataset, define the neural network model, train model, and test the model.

- ***Keras***

  Keras is an open-source neural network library written in Python. In 2017, Google's TensorFlow team decided to support Karas in TensorFlow's core library. Now, it is TensorFlow's high-level API for building and training deep learning models. This is the main reason why we choose to use this library. It is designed to enable fast experimentation with deep neural networks, it focused on being user friendly, modular, and extensible. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models.

- ***Pathlib***

  Pathlib is a Python module for object-oriented filesystem paths. This module offers classes representing filesystem paths with semantics appropriate for different operating systems. Because we are not using the same operating system to develop this project, this module can help us deal with the problem of running another member's code in a different machine. The path classes are divided between pure paths and concrete paths. The pure paths are purely computational operations without input and output operations, it cannot access the filesystem. The concrete paths are inherited from pure paths, but it's also provided input and output operations, and this is the real path to access a filesystem. The pathlib will transfer pure paths, which we are given through code, to instantiate a concrete path for the platform the code is running on.

- ***Fastai***:

  The vision module of the *fastai* library contains all the necessary functions to define a dataset and train a model for computer vision tasks. It contains four different submodules to reach that goal: vision.image contains the basic definition of an Image object and all the functions that are used behind the scenes to apply transformations to such an object. *vision.transform* contains all the transforms we can use for data augmentation. vision.data contains the definition of ImageDataBunch as well as the utility function to easily build a DataBunch for Computer Vision problems. *vision.learner* lets you build and fine-tune models with a pretrained CNN backbone or train a randomly initialized model from scratch. This gives us the necessary functions to define a dataset and train a model.

- ***Matplotlib***

  Allow us to create, save and store our data charts in the notebook. This was especially handy when we created many visualization charts to show how the algorithm is performing and it saved on our google Colab script, so our group members were able to see the result the next day.

- ***Pandas***

  This was helpful when using tabular data, such as data stored in spreadsheets or databases. Since our data was in google drive, we treated it as a database and mounted our images in folders that Pandas allowed us to access and modify. Pandas helped us to explore, clean and process our data. In Pandas, the data table we used is called the dataframe.

- ***NumPy***

  NumPy is the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object, sophisticated functions, tools for integrating, useful linear algebra, Fourier transform, and random number capabilities. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. We used this library to handle the calculations and look for constant variables used in the application.

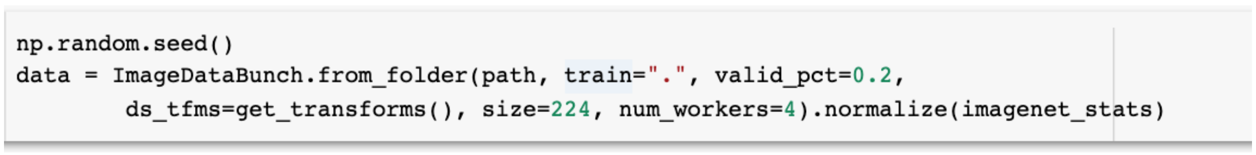
3. Methods

Before jumping into code, we need to explain the most important methods that the script uses. This will help the reader know what each line of the script is doing and give a deeper understanding of our machine learning algorithm. We narrowed the list down to eight methods to help the reader understand the code.

- $Numpy.random.seed$: This method creates a random generator so for each run the call will set the seed to a random number.
- $Cnn\_learner$: This module gets a model suitable for transfer learning. It was helpful to train the model on large datasets. The summary of this module is that it learns to recognize many features on all of this data, which benefits from this knowledge, especially since our dataset is small.
- $Cnn_learner.fit_one_cycl$e: We used to expose the neural net data and show it in fit_one cycle. This gives us an approximation for our validation loss and training loss.
- $Cnn\_learner.lr\_find$: The learner.lr_find lists out all the possible learning rates within the available space of memory. So, in our case our chart only displayed 13 epochs of learning rates since we used the free google Colab memory,
- $Cnn\_learner.recorder.plot(suggestion = True)$: This method displays us a graph where we can see the learning rate against the loss, then we place a red point on the gradient with the steepest drop. This was used to first guess the learning rate for our model.
- $ClassificationInterpretation.from\_learner(learn50)$: This method allowed us to train a 50-layer neural network from Resnet50. This did the majority of the backend work to create a neural network and connect all the nodes.
- $Plot\_confusion\_matrix$: The confusion matrix method allowed us to visualize the results for creating the cross validation set and give us an approximation for our accuracy.
- $\mathrm{imgK} = \mathrm{open\_image}('/\mathrm{content/drive/My\ Drive/sneakers/TEST/test2.png}')$ and $pred\_class, pred\_idx, outputs = learn50new.predict(imgK)$: This line of code just basically opens the test image and passes it through the learning algorithm to predict. We then get an output with the figure of the sneaker with the prediction on top in text.

4. Data Used

The data used for the final script was significantly increased. At the end of the project we provided more than 200 photos to help train our algorithm. This not only gave us the opportunity to train our model but gave us a bigger cross validation set. In total we worked with 500+ images of shoes. The test data is around 16% of the whole data set while the training data is the remaining 84%. To explain a little about the code below, the num workers method is the number of CPUs to use which are passed to the init method. Also the $get\_transforms$ method returns a tuple of two lists that transforms the training set and validation set. To train our model we used the following code shown below. This made the data to be broken to 20% validation set and 80% training the mode.

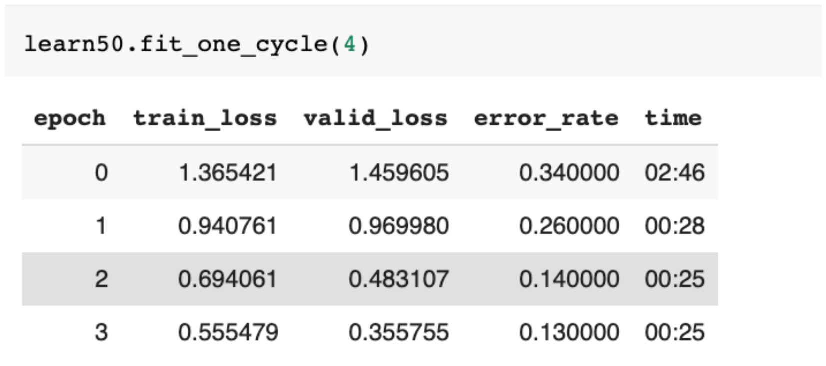


```
np.random.seed()
data = ImageDataBunch.from_folder(path, train=".", valid_pct=0.2,
        ds_tfms=get_transforms(), size=224, num_workers=4).normalize(imagenet_stats)
```

Figure 22: Preparing data to be used in algorithm

5. Training Phase 1

To train our data we wanted to see first what our initial results were for the training loss, validation loss and error rate. This would give us a starting point to see if our results were getting better when we included a learning rate constant. As you may see below, our training loss and validation loss were above 1 which is a terrible result because our objective was to get it as close as possible to zero. This would mean our error rate would



```
learn50.fit_one_cycle(4)
```

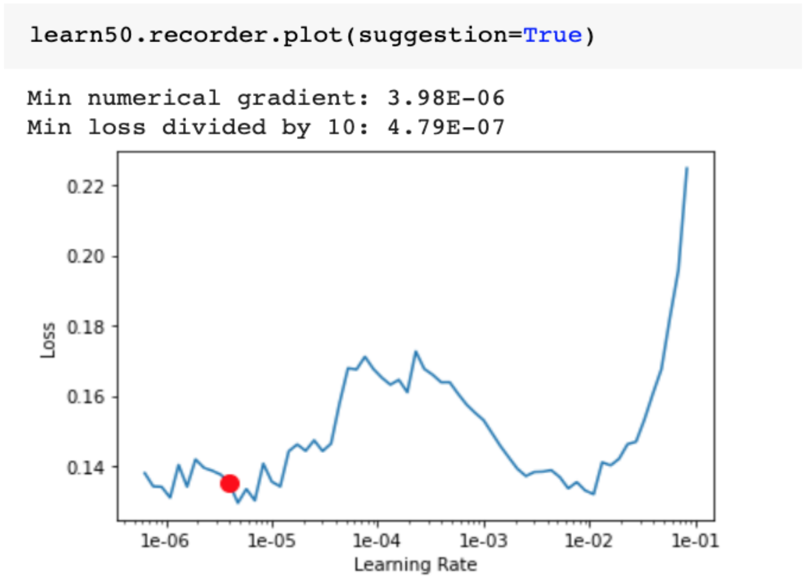
epoch	train_loss	valid_loss	error_rate	time
0	1.365421	1.459605	0.340000	02:46
1	0.940761	0.969980	0.260000	00:28
2	0.694061	0.483107	0.140000	00:25
3	0.555479	0.355755	0.130000	00:25

Figure 23: Using one cycle to see initial values for variables

decrease and our machine learning algorithm would be able to predict an image at a high accuracy.

6. Finding Learning Rate

Finding the learning was as easy to pass in the model and use the method $recorder.plot$. This allowed the model to produce a graph that shows the learning rate vs the loss. Then using the parameter suggestion = true, a red dot is placed on the gradient with the steepest drop. So, given the graph below our team decided to use the constant learning rate around the range of 1e-06 and 1e-05.



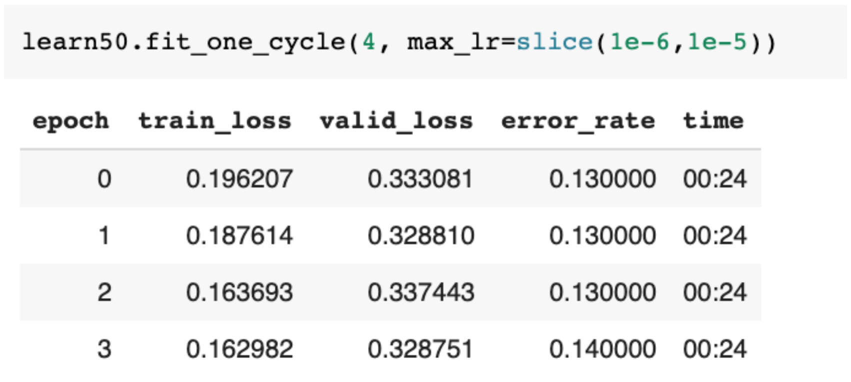
```
learn50.recorder.plot(suggestion=True)
```

```
Min numerical gradient: 3.98E-06
Min loss divided by 10: 4.79E-07
```

Figure 24: Learning rate graph

7. Training Phase 2

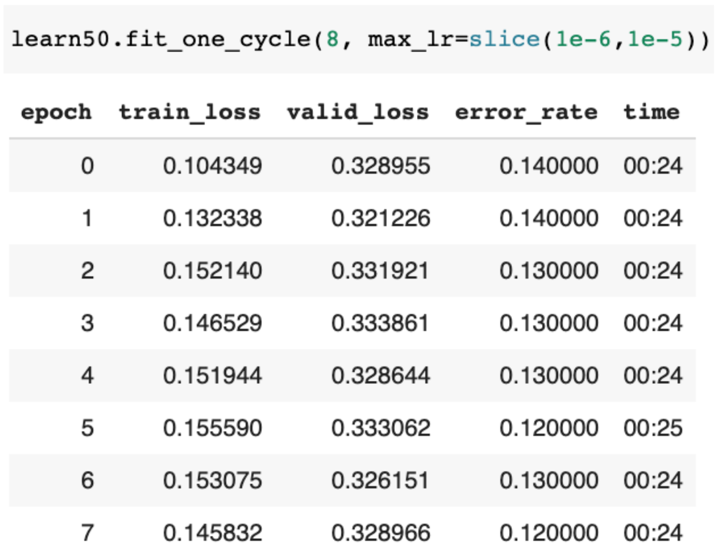
In the second training phase we use the *max_lr* parameter to limit where the learning rate could be positioned. This made our training loss and validation loss greatly decrease given four epochs. The idea the group had was that if we kept increasing our epochs our variables on the graph were decrease even more, moving towards a number closer to zero.



```
learn50.fit_one_cycle(4, max_lr=slice(1e-6,1e-5))
```

epoch	train_loss	valid_loss	error_rate	time
0	0.196207	0.333081	0.130000	00:24
1	0.187614	0.328810	0.130000	00:24
2	0.163693	0.337443	0.130000	00:24
3	0.162982	0.328751	0.140000	00:24

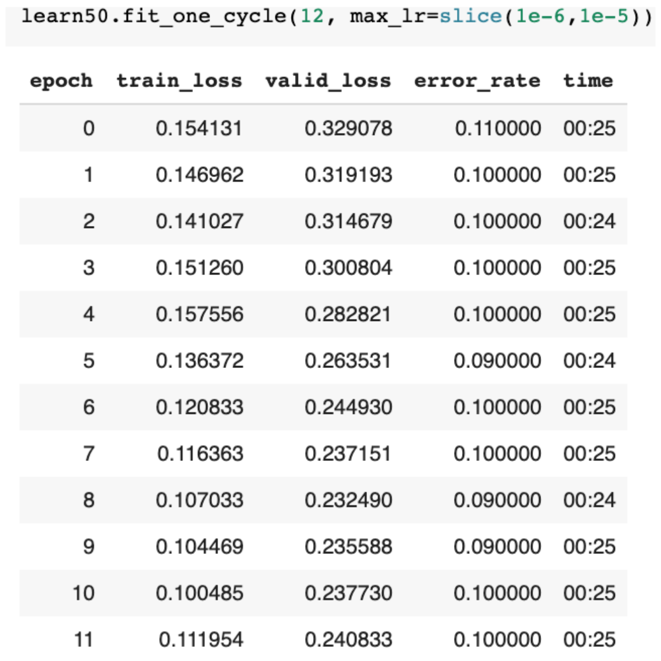
Figure 25: 4 epoch training with learning rate



```
learn50.fit_one_cycle(8, max_lr=slice(1e-6,1e-5))
```

epoch	train_loss	valid_loss	error_rate	time
0	0.104349	0.328955	0.140000	00:24
1	0.132338	0.321226	0.140000	00:24
2	0.152140	0.331921	0.130000	00:24
3	0.146529	0.333861	0.130000	00:24
4	0.151944	0.328644	0.130000	00:24
5	0.155590	0.333062	0.120000	00:25
6	0.153075	0.326151	0.130000	00:24
7	0.145832	0.328966	0.120000	00:24

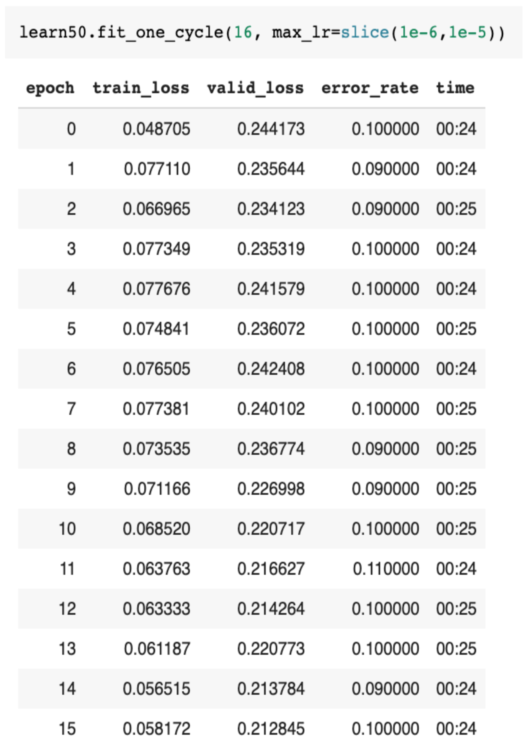
Figure 26: 8 epoch training with learning rate



```
learn50.fit_one_cycle(12, max_lr=slice(1e-6,1e-5))
```

epoch	train_loss	valid_loss	error_rate	time
0	0.154131	0.329078	0.110000	00:25
1	0.146962	0.319193	0.100000	00:25
2	0.141027	0.314679	0.100000	00:24
3	0.151260	0.300804	0.100000	00:25
4	0.157556	0.282821	0.100000	00:25
5	0.136372	0.263531	0.090000	00:24
6	0.120833	0.244930	0.100000	00:25
7	0.116363	0.237151	0.100000	00:25
8	0.107033	0.232490	0.090000	00:24
9	0.104469	0.235588	0.090000	00:25
10	0.100485	0.237730	0.100000	00:25
11	0.111954	0.240833	0.100000	00:25

Figure 27: 12 epoch training with learning rate



```
learn50.fit_one_cycle(16, max_lr=slice(1e-6,1e-5))
```

epoch	train_loss	valid_loss	error_rate	time
0	0.048705	0.244173	0.100000	00:24
1	0.077110	0.235644	0.090000	00:24
2	0.066965	0.234123	0.090000	00:25
3	0.077349	0.235319	0.100000	00:24
4	0.077676	0.241579	0.100000	00:24
5	0.074841	0.236072	0.100000	00:25
6	0.076505	0.242408	0.100000	00:24
7	0.077381	0.240102	0.100000	00:25
8	0.073535	0.236774	0.090000	00:25
9	0.071166	0.226998	0.090000	00:25
10	0.068520	0.220717	0.100000	00:25
11	0.063763	0.216627	0.110000	00:24
12	0.063333	0.214264	0.100000	00:25
13	0.061187	0.220773	0.100000	00:25
14	0.056515	0.213784	0.090000	00:24
15	0.058172	0.212845	0.100000	00:24

Figure 28: 16 epoch training with learning rate

The next three images of epoch training show an increased number of epochs and a better training loss value and validation loss.

8. Data Visualization

The next step was to visualize the prediction using the cross-validation set. This meant the system would predict what each shoe was and know the number of correct shoes. Below shows a confusion matrix which is a table that is often used to describe the performance of a classification mode on our set of data for which the true values are known. It allows the visualization of the performance of an algorithm. It can be seen that 28/30 adidas shoes were correctly identified, 35/40 Nike shoes were correct, and 27/30 Puma shoes were too correct.

```
interp1 = ClassificationInterpretation.from_learner(learn50)
```

```
interp1.plot_confusion_matrix()
```

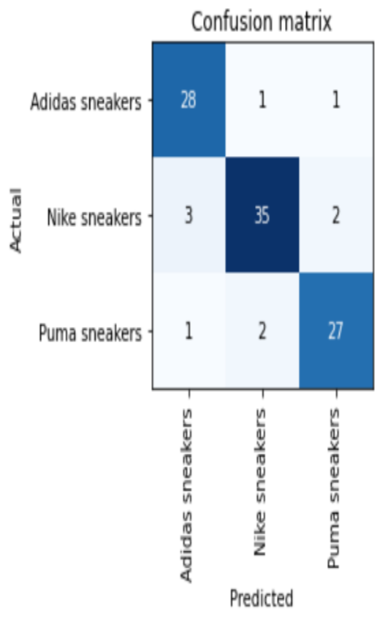
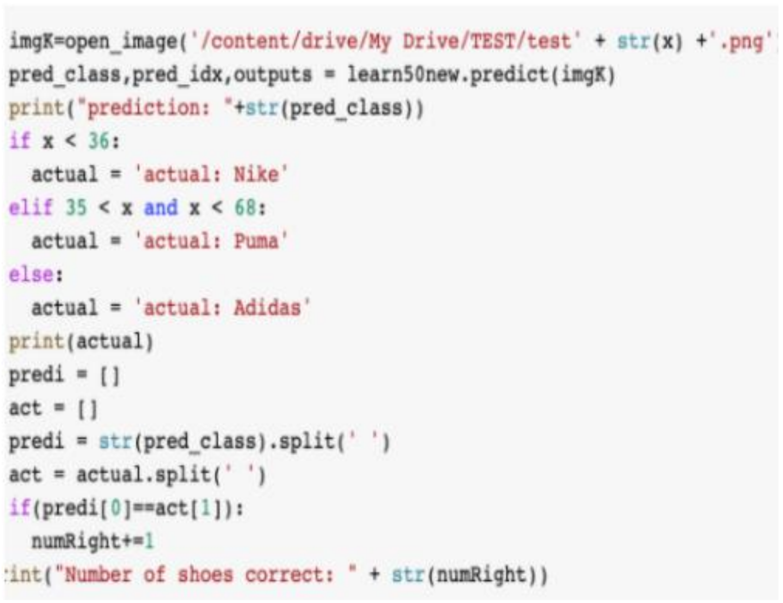


Figure 29: Confusion matrix of the algorithm

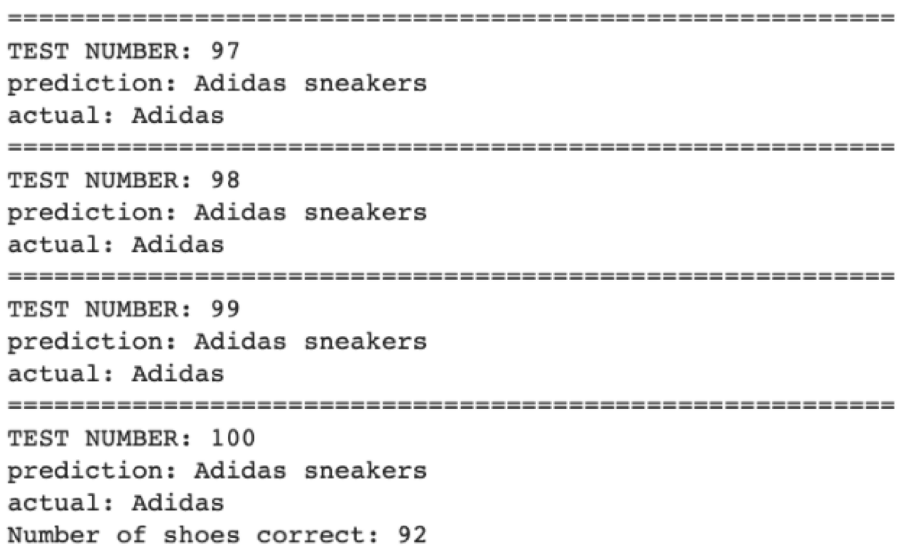
9. Prediction Code

Before jumping into the results, it is good to go over how our model determined if the predictions were right. This is done by using a counter then having the model spit our whatever guess it makes on the image. Then we made sure our test images were ordered in a certain way that made 1-35 test images into sure Nike shoes, 36-67 Puma shoes and 68-100 Adidas shoes. We then compared if the result given by the algorithm was the same as the range of shoes and incremented the counter. We then ended up with 92 shoes correct out of the 100 test images.



```
imgK=open_image('/content/drive/My Drive/TEST/test' + str(x) +'.png'
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction: "+str(pred_class))
if x < 36:
  actual = 'actual: Nike'
elif 35 < x and x < 68:
  actual = 'actual: Puma'
else:
  actual = 'actual: Adidas'
print(actual)
predi = []
act = []
predi = str(pred_class).split(' ')
act = actual.split(' ')
if(predi[0]==act[1]):
  numRight+=1
int("Number of shoes correct: " + str(numRight))
```

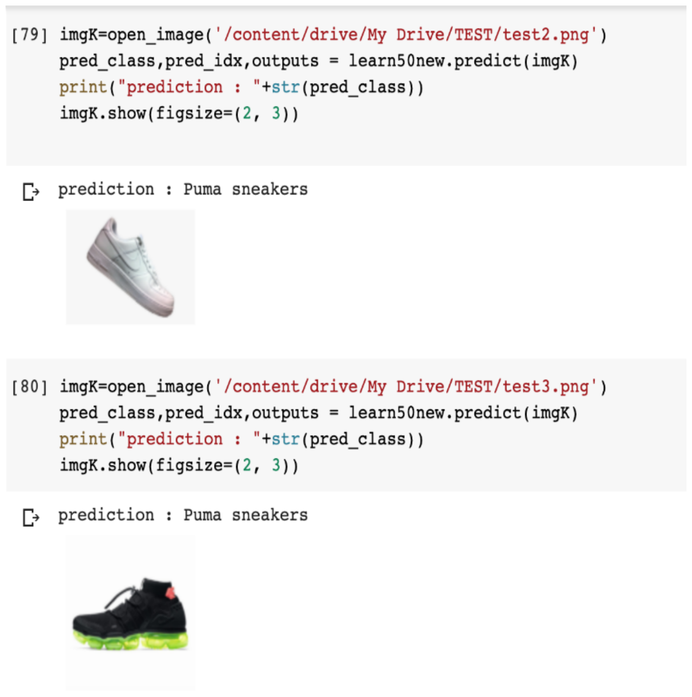
Figure 31: Prediction Code



```
==========================================================
TEST NUMBER: 97
prediction: Adidas sneakers
actual: Adidas
==========================================================
TEST NUMBER: 98
prediction: Adidas sneakers
actual: Adidas
==========================================================
TEST NUMBER: 99
prediction: Adidas sneakers
actual: Adidas
==========================================================
TEST NUMBER: 100
prediction: Adidas sneakers
actual: Adidas
Number of shoes correct: 92
```

Figure 30: Sample output for prediction

We also wanted to show how our data predicts with the given figure of the shoe. We were not able to show all the figures, but we managed to print out around 8 images with the result on top. The image below will only show two figures and the prediction, to see all the results please look the appendix for the final script.



```
[79] imgK=open_image('/content/drive/My Drive/TEST/test2.png')
     pred_class,pred_idx,outputs = learn50new.predict(imgK)
     print("prediction : "+str(pred_class))
     imgK.show(figsize=(2, 3))
```

```
prediction : Puma sneakers
```

```
[80] imgK=open_image('/content/drive/My Drive/TEST/test3.png')
     pred_class,pred_idx,outputs = learn50new.predict(imgK)
     print("prediction : "+str(pred_class))
     imgK.show(figsize=(2, 3))
```

```
prediction : Puma sneakers
```

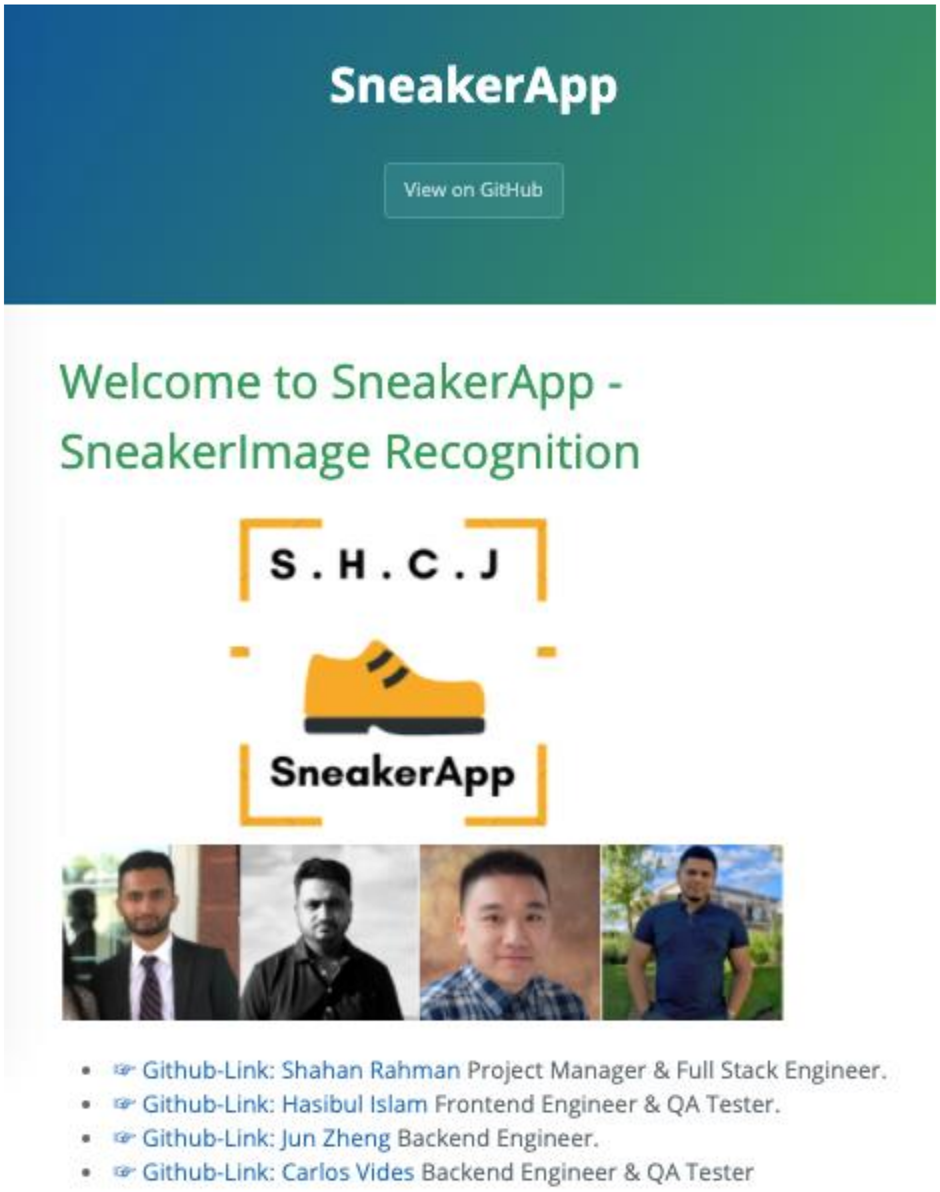
Figure 32: Predictions with corresponding shoe

10. Results

This model gave us a 92% accuracy of predicting the shoe company. We know this can be improved even more if Google Colab gave us unlimited Ram so we could run even more epoch training to our algorithm and add more to our datasets respectively. Since our research shows that published machine learning algorithms usually have thousands of epoch training to minimize error.

11. Website

To deploy our website, we created a GitHub page that would display all resources we created and worked with throughout the semester. This included all our data and presentations during the 2 week meetings with Professor Wolberg. We were able to share our script with the public and have a nice frontend to demo for our final presentation. The link to this webpage is: https://ccnyundergraduatecsdegree.github.io/SneakerApp/



Welcome to SneakerApp - SneakerImage Recognition

- ☞ Github-Link: Shahan Rahman Project Manager & Full Stack Engineer.
- ☞ Github-Link: Hasibul Islam Frontend Engineer & QA Tester.
- ☞ Github-Link: Jun Zheng Backend Engineer.
- ☞ Github-Link: Carlos Vides Backend Engineer & QA Tester

- GitHub is the Host Page: https://github.com/CcnyUndergraduateCsDegree/SneakerApp
- Website Page: https://ccnyundergraduatecsdegree.github.io/SneakerApp/
- Host page: README.md file to edit and add stuff to upload to the website.
- README.md : Team Member Name
- README.md : Final Report for Code using GoogleColab
- README.md : Each 2-weeks Presentation to follow guidelines with using a gif file.

V. Reference

Dataman. “What Is Image Recognition?” Medium, Towards Data Science, 29 Mar. 2020, towardsdatascience.com/module-6-image-recognition-for-insurance-claim-handling-part-i-a338d16c9de0.

“Getting Started¶.” Getting Started - Pandas 1.0.3 Documentation, pandas.pydata.org/docs/getting_started/index.html.

He, Kaiming et al. “Deep Residual Learning for Image Recognition.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016): 770-778.

Howard, Andrew G. et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” *ArXiv* abs/1704.04861 (2017): n. pag.

“Image Tutorial¶.” Image Tutorial - Matplotlib 3.2.1 Documentation, matplotlib.org/3.2.1/tutorials/introductory/images.html.

“Keras: TensorFlow Core.” *TensorFlow*, www.tensorflow.org/guide/keras.

“Keras.” *Wikipedia*, Wikimedia Foundation, 3 May 2020, en.wikipedia.org/wiki/Keras.

Khloe. “Image Recognition Applications: 7 Essential Future Uses.” Data Science Society, 16 Jan. 2020, www.datasciencesociety.net/image-recognition-applications-7-essential-future-uses/.

“Matplotlib.pyplot¶.” Matplotlib.pyplot - Matplotlib 3.2.1 Documentation, matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.html.

“Neural Networks for Image Recognition: Methods, Best Practices, Applications.” MissingLink.ai, missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/.

“NumPy¶.” NumPy, numpy.org/.

“Pathlib - Object-Oriented Filesystem Paths¶.” *Pathlib - Object-Oriented Filesystem Paths - Python 3.8.3 Documentation*, docs.python.org/3/library/pathlib.html#pure-paths.

RishabhRishabh 4, et al. “Purpose of "%Matplotlib Inline.’” Stack Overflow, 1 Dec. 1966, stackoverflow.com/questions/43027980/purpose-of-matplotlib-inline.

Sandler, Mark et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018): 4510-4520.

“TensorFlow.” Wikipedia, Wikimedia Foundation, 16 May 2020, en.wikipedia.org/wiki/TensorFlow.

"Vision.learner." Vision.learner | Fastai, docs.fast.ai/vision.learner.html.

"Vision." Vision | Fastai, docs.fast.ai/vision.html.

VI. Appendix

```
from google.colab import drive
drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.
from fastai.vision import *
# %matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib.image as mpimg

path=Path('/content/drive/My Drive/sneakers')

path.ls()

classes = ['adidas','nike','puma']

np.random.seed()
data = ImageDataBunch.from_folder(path, train=".", valid_pct=0.2,
        ds_tfms=get_transforms(), size=224, num_workers=4).normal
ize(imagenet_stats)

data.classes

data.show_batch(rows=4, figsize=(12,12))

data.classes, data.c, len(data.train_ds), len(data.valid_ds)

learn50 = cnn_learner(data, models.resnet50, metrics=error_rate)

learn50.fit_one_cycle(4)

learn50.lr_find()

learn50.recorder.plot(suggestion=True)
```

```
learn50.fit_one_cycle(4, max_lr=slice(1e-6,1e-5))

learn50.fit_one_cycle(8, max_lr=slice(1e-6,1e-5))

learn50.save('stage-50-1')

learn50.unfreeze()

learn50.fit_one_cycle(12, max_lr=slice(1e-6,1e-5))

learn50.unfreeze()

learn50.fit_one_cycle(16, max_lr=slice(1e-6,1e-5))

learn50.save('stage-50-1-231')

learn50.unfreeze()



interp1 = ClassificationInterpretation.from_learner(learn50)

interp1.plot_confusion_matrix()

losses1,idxs1 = interp1.top_losses()
len(data.valid_ds)==len(losses1)==len(idxs1)

interp1.plot_top_losses(9, figsize=(15,11))

learn50.export()

defaults.device = torch.device('cuda')

learn50new = load_learner(path)

numRight = 0

for x in range(1,101):
  print('=========================================================')
  print('TEST NUMBER: ' + str(x))

  imgK=open_image('/content/drive/My Drive/TEST/test' + str(x) +'.png')
  pred_class,pred_idx,outputs = learn50new.predict(imgK)
  print("prediction: "+str(pred_class))
  if x < 36:
    actual = 'actual: Nike'
  elif 35 < x and x < 68:
    actual = 'actual: Puma'
  else:
    actual = 'actual: Adidas'
  print(actual)
  predi = []
  act = []
  predi = str(pred_class).split(' ')
  act = actual.split(' ')
  if(predi[0]==act[1]):
    numRight+=1
print("Number of shoes correct: " + str(numRight))

imgK=open_image('/content/drive/My Drive/TEST/test2.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test3.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test5.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test7.png')
```

```
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test8.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test9.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test11.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test14.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test15.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

imgK=open_image('/content/drive/My Drive/TEST/test19.png')
pred_class,pred_idx,outputs = learn50new.predict(imgK)
print("prediction : "+str(pred_class))
imgK.show(figsize=(2, 3))

"""# Currently model has an accuracy of 92%."""
```