Machine Learning Engineer Nanodegree
Capstone Project

Mofei Liu
May 6th, 2019

I. Definition

Project Overview

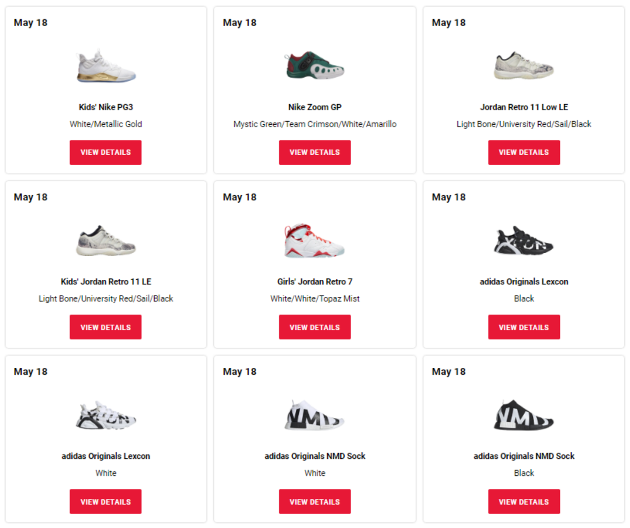


Fig.1 Footlocker Release Calendar

In recent years, sneaker culture and the industry itself has been grown fast. Last year, the U.S athletic footwear industry generated $17.5 billion, according to the NDP Group, and its resale industry is estimated to be worth $1 billion.

With more and more people start get into it collecting sneakers and using them as an outlet to express themselves, brands like Nike & Adidas start going at a very fast paced when it comes to sneaker release. As seen in Fig.1, Dozens of new colorway & new silhouettes being released weekly and hundreds new products every year, in 2018 along, Jordan brand has released 60 difference colorways of the classic Air Jordan 1. This made consumers are very overwhelmed by the new product they're seeing every day every week.

To solve this problem, I'm trying to create a web application that can correctly identify the product for the customer giving an image. There's no existing dataset for I can use so I'll try to build my own dataset from scraping images from the internet. This dataset will not be completely balanced due to the popularity/variations of each sneaker.

Problem Statement

The goal is to create a web application with the task involved in the following:

1. Scraping image dataset from the internet, and preprocess the training data
2. Train a classifier that can identify sneaker model
3. Make the classifier run on a web app

The final application is expected to be useful identify sneakers giving an user upload picture

Metrics

To evaluation the model vs benchmark model, I'll be using the logarithmic loss method[5], which works by penalizing the false classification. It should work well here since this is a multi-class classification problem.

Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below:

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

Where,

y_ij, indicates where sample i belongs to class j or not

p_ij, indicates the probability of sample i belonging to class j

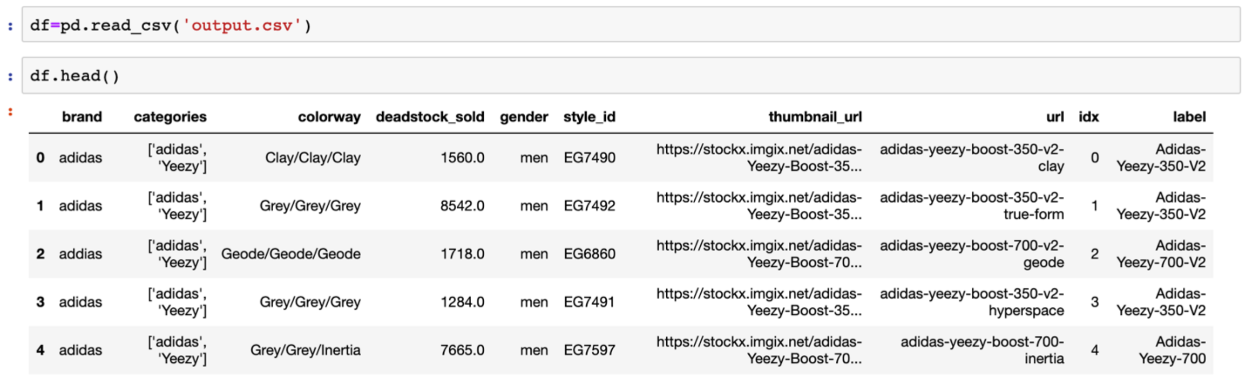
Log Loss has no upper bound and it exists on the range [0, ∞). Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.

II. Analysis

Data Exploration

Fig.2 images from the dataset

The image from the dataset has dimension of 2200 x 1468 pixels, with all 0-360 degree of the product, this add up a very good image argumentation to the dataset. In total we have 52593 images in 179 categories within this dataset, which is sufficient to train the classifier.



```
: df=pd.read_csv('output.csv')
```

```
: df.head()
```

	brand	categories	colorway	deadstock_sold	gender	style_id	thumbnail_url	url	idx	label
0	adidas	['adidas', 'Yeezy']	Clay/Clay/Clay	1560.0	men	EG7490	https://stockx.imgix.net/adidas-Yeezy-Boost-35...	adidas-yeezy-boost-350-v2-clay	0	Adidas-Yeezy-350-V2
1	adidas	['adidas', 'Yeezy']	Grey/Grey/Grey	8542.0	men	EG7492	https://stockx.imgix.net/adidas-Yeezy-Boost-35...	adidas-yeezy-boost-350-v2-true-form	1	Adidas-Yeezy-350-V2
2	addias	['adidas', 'Yeezy']	Geode/Geode/Geode	1718.0	men	EG6860	https://stockx.imgix.net/adidas-Yeezy-Boost-70...	adidas-yeezy-boost-700-v2-geode	2	Adidas-Yeezy-700-V2
3	adidas	['adidas', 'Yeezy']	Grey/Grey/Grey	1284.0	men	EG7491	https://stockx.imgix.net/adidas-Yeezy-Boost-35...	adidas-yeezy-boost-350-v2-hyperspace	3	Adidas-Yeezy-350-V2
4	adidas	['adidas', 'Yeezy']	Grey/Grey/Inertia	7665.0	men	EG7597	https://stockx.imgix.net/adidas-Yeezy-Boost-70...	adidas-yeezy-boost-700-inertia	4	Adidas-Yeezy-700

Fig.3 data annotation

The other annotation of the data is stored in a csv file, which have the following field:

- Brand
- Categories
- Colorway
- Deadstock_sold
- Gender
- Style_id
- Thumbnail_url
- url
- index
- label

In this classifier we're only using label as the output of the prediction.

Exploratory Visualization

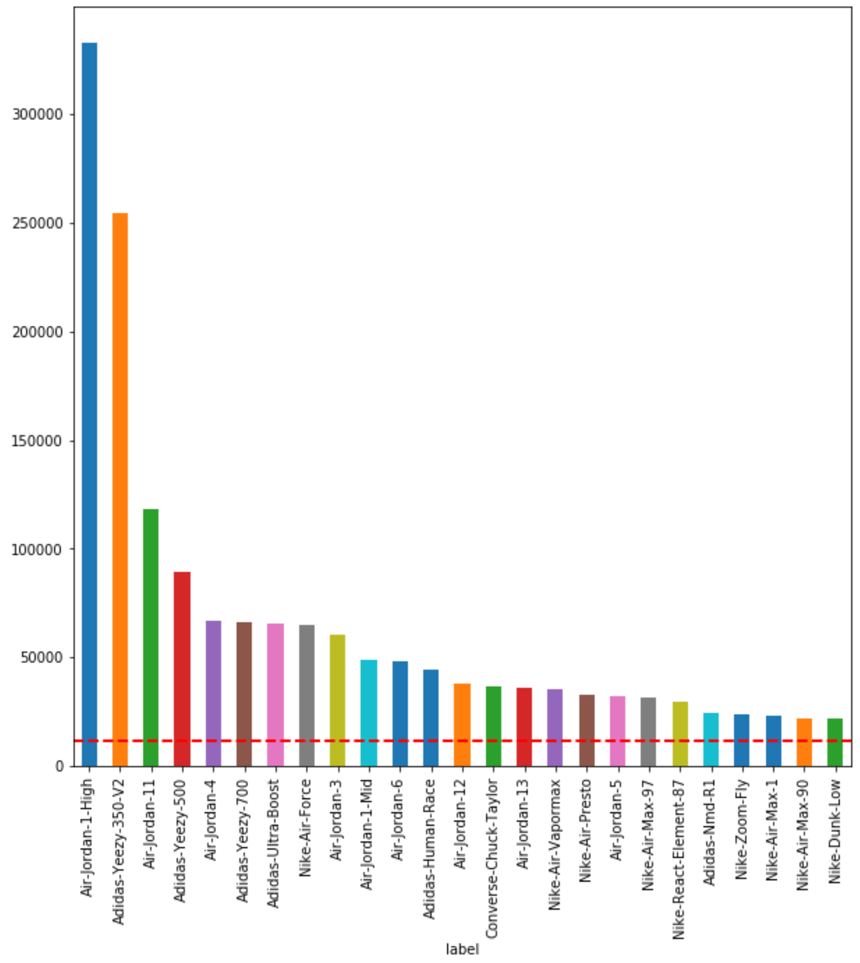


Fig.4 number of sold vs model

- Here’s the top 25 most popular sneakers sold on StockX by #of pairs sold.
- The top 3 models are Air Jordan 1 High, Adidas Yeezy 350 V2 and Air Jordan 11.
- On average there's 11385 pair have been sold on the platform for each style
- The distribution is right tailed.

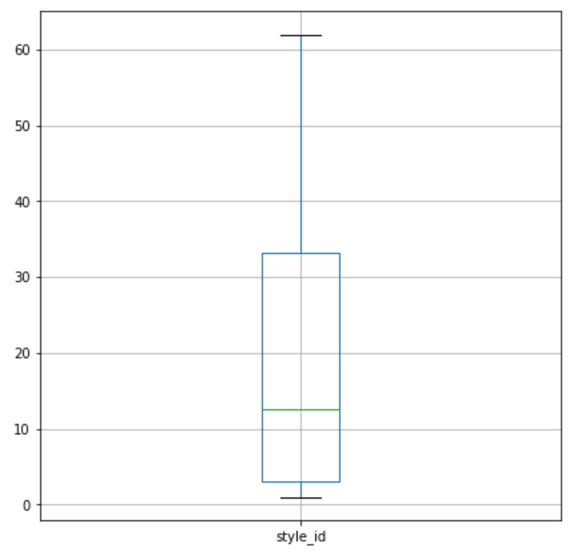


Fig.5 number of images per label

- Here's boxplot for how many data we have per each label, with max of 62 and min of 1 per label, median of 12.5
- I do think this data is not weighted properly, it could be affecting our result, I would either remove those labels with low data samples or gathering more data

Algorithms and Techniques

The classifier is Convolutional Neural Network, which is the most advanced algorithm for image processing task such as classification.
These following parameters can be tuned to optimize the classifier:

- ➢ Training parameters
  - Training Length (Number of Epochs)
  - Batch Size (# of images look at once during a single step)
  - Optimizer type
  - Learning rate
  - Momentum
  - Weight decay
- ➢ Neural network architecture
  - Number of layers
  - Layer types
  - Layer parameters

During the training, both training & validation sets are loaded into RAM then random batches are selected to be loaded into GPU memory, the training is done using the Mini-batch gradient descent algorithm.

Benchmark

1. Random Choice: A dummy classifier that random choice one of the labels.
2. A 4-layer CNN made from scratch is also used to measure the performance of transfer learning.

Benchmark model architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 16)      448       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 16)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 109, 109, 32)      4640      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 52, 52, 32)        9248      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 9216)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 179)               1649843   
=================================================================
Total params: 1,682,675
Trainable params: 1,682,675
Non-trainable params: 0
_________________________________________________________________
```

Fig.6 Benchmark CNN Architecture

- Conv2D: 4 Conv2D layer were used here with 16,32,32,64 filter sizes, relu activation function.
- MaxPooling2D: Add MaxPooling2D layer after each convolution layer.
- Flatten: Fully connected layer
- Dropout: One dropout layer with value of 0.3
- Dense: Fully connected layer with output corresponding number of labels.

III. Methodology

Data Preprocessing

Here’s the link of example data image:
https://stockx-360.imgix.net/Nike-Air-Max-1-97-Sean-Wotherspoon-NA/Images/Nike-Air-Max-1-97-Sean-Wotherspoon-NA/Lv2/img01.jpg?auto=format,compress&q=90&updated_at=1538080256&w=1300

From my observation, product link is constructed by the shoe name itself as the sub-link, and for the image link, in the folder it's in sequence for 0 to 360-degree image coded as img01.jpg to img35.jpg.

I created a simple python scraper using bs4 & urllib, downloading images into different folder named by the label, then separate entire folder with 80%/20% split for training & test data.

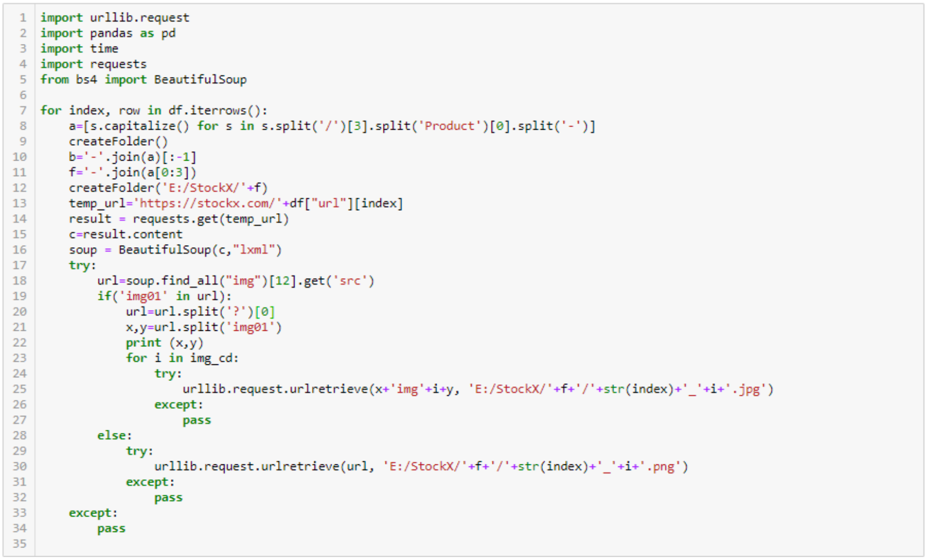
When using Keras CNN, it requires a 4D tensor as input, with shape of (nb_samples, row, cols, channels), here I’m using the build in function **keras.preprocessing.image.ImageDataGenerator** to resize the image to 224 x 224 pixel and then convert the image to a 4D tensor, so after the processing, our input will have a dimension of (1,224,224,3)

Implementation

Data Preprocessing

`(Data Processing.ipynb)`

Generate product url dataframe from giving keyword using stockxsdk library, save csv.
Create folders with label name, start image scraping process, from saved url csv file



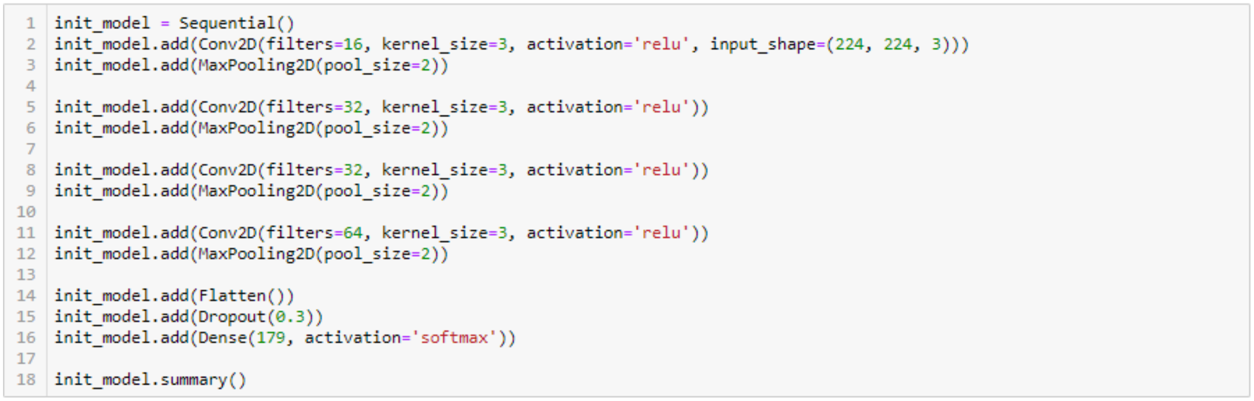
```python
import urllib.request
import pandas as pd
import time
import requests
from bs4 import BeautifulSoup

for index, row in df.iterrows():
    a=[s.capitalize() for s in s.split('/')[3].split('Product')[0].split('-')]
    createFolder()
    b='-'.join(a)[:-1]
    f='-'.join(a[0:3])
    createFolder('E:/StockX/'+f)
    temp_url='https://stockx.com/'+df["url"][index]
    result = requests.get(temp_url)
    c=result.content
    soup = BeautifulSoup(c,"lxml")
    try:
        url=soup.find_all("img")[12].get('src')
        if('img01' in url):
            url=url.split('?')[0]
            x,y=url.split('img01')
            print (x,y)
            for i in img_cd:
                try:
                    urllib.request.urlretrieve(x+'img'+i+y, 'E:/StockX/'+f+'/'+str(index)+'_'+i+'.jpg')
                except:
                    pass
        else:
            try:
                urllib.request.urlretrieve(url, 'E:/StockX/'+f+'/'+str(index)+'_'+i+'.png')
            except:
                pass
    except:
        pass

```

Clean up duplicate/wrong labels, total images scraped 52,593 within 179 folders (labels) Split train/test set at 80/20 with a helper function.

Model Training

(Model Training.ipynb)

- **Benchmark CNN**



```
init_model = Sequential()
init_model.add(Conv2D(filters=16, kernel_size=3, activation='relu', input_shape=(224, 224, 3)))
init_model.add(MaxPooling2D(pool_size=2))

init_model.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
init_model.add(MaxPooling2D(pool_size=2))

init_model.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
init_model.add(MaxPooling2D(pool_size=2))

init_model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
init_model.add(MaxPooling2D(pool_size=2))

init_model.add(Flatten())
init_model.add(Dropout(0.3))
init_model.add(Dense(179, activation='softmax'))

init_model.summary()
```

Input tensor shape (224,224,3) representing the image pixel and 3 channels, I used 4 convolutional layer and with maxpolling2d layer to slim down the dimensionality, 'Relu' activation function is used in each convolution layer and have increasing order of filter (16, 32, 32, 64) to achieve more complex pattern, kernel size is 3x3. A flatten layer is added after the last convolutional layer, then a dropout layer and finally a dense layer of output of 179 which matches number of labels. With this architecture there's 1,682,675 trainable parameters.

This benchmark CNN model is trained with pre-processed training data using validation split of 0.2, batch size of 256 and epoch of 50. As a result, this benchmark model achieved accuracy of 0.73 which is not bad for a very simple model.

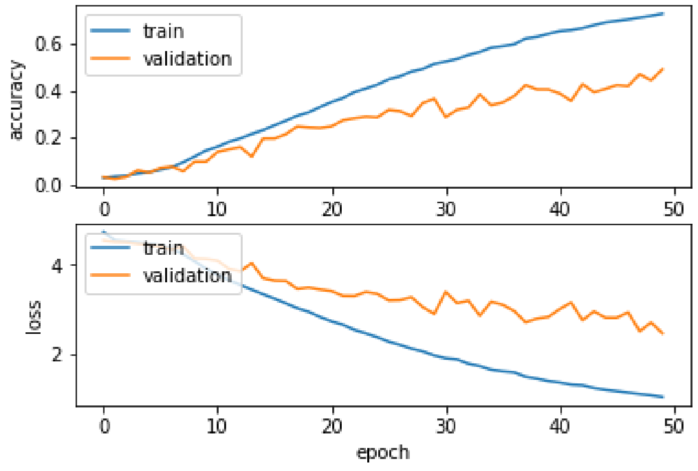


Fig.7 Benchmark CNN

- **VGG 16 Transfer Learning**

  First load the model without top layer

```
model = applications.VGG16(include_top=False, weights='imagenet')
```

Then run through the train set, save the bottleneck feature into a .npy file

```
bottleneck_features_train = model.predict_generator(
    train_generator, predict_size_train)
np.save(open('bottleneck_features_train.npy', 'wb'),
        bottleneck_features_train)
```

Next up we start adding the top model, first load the bottleneck feature saved before.

```
 # load the bottleneck features saved earlier
train_data = np.load(open('bottleneck_features_train.npy','rb'))
train_labels = train_generator_top.classes
train_labels = np_utils.to_categorical(train_labels, num_classes=num_classes)
```

Creating a new top layer have input of bottleneck feature shape, add dense layer and dropout, finally another dense layer with softmax activation function with output of numbers of label.

```
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='sgd',loss='categorical_crossentropy', metrics=['accuracy'])
```

This model reaches 0.83 accuracy & 0.82 log-loss after 100 epochs with 256 batch size.

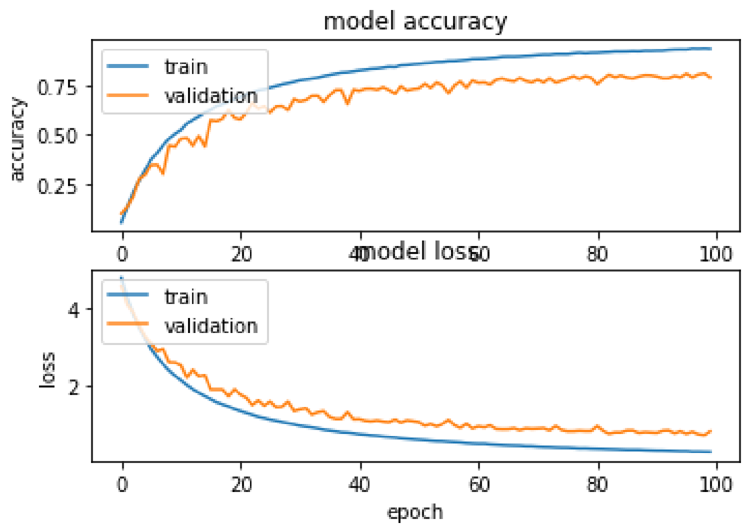


Fig.8 VGG-16 Transfer Learning

- **VGG 16 Fine Tuning**

  To further refine the model, we will use the top model weight to fine tuning.
  First same as last model, we load the VGG-16 imagenet model without top layer

```
model_ft = applications.VGG16(weights='imagenet', include_top=False,input_shape=(224, 224, 3))
```

  Then build the same top layer block, import the weight we already saved.

```
# build a classifier model to put on top of the convolutional model
top_model = Sequential()
top_model.add(Flatten(input_shape=model_ft.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(179, activation='softmax'))

# note that it is necessary to start with a fully-trained
# classifier, including the top classifier,
# in order to successfully do fine-tuning
top_model.load_weights(top_model_weights_path)
```

  Adding the top layer after the convolutional base, then freeze the first 15 convolutional layer in the VGG-16. This allow us to have more trainable parameters.

```
# add the model on top of the convolutional base
model_ft=Model(input=model_ft.input,output=top_model(model_ft.output))

# set the first 15 layers (up to the last conv block)
# to non-trainable (weights will not be updated)
for layer in model_ft.layers[:15]:
    layer.trainable = False
```

  Compile the model with an SGD optimizer with a slow learning rate

```
model_ft.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),
              metrics=['accuracy'])
```

  This fine-tuning model achieved 0.92 accuracy in 50 epochs.

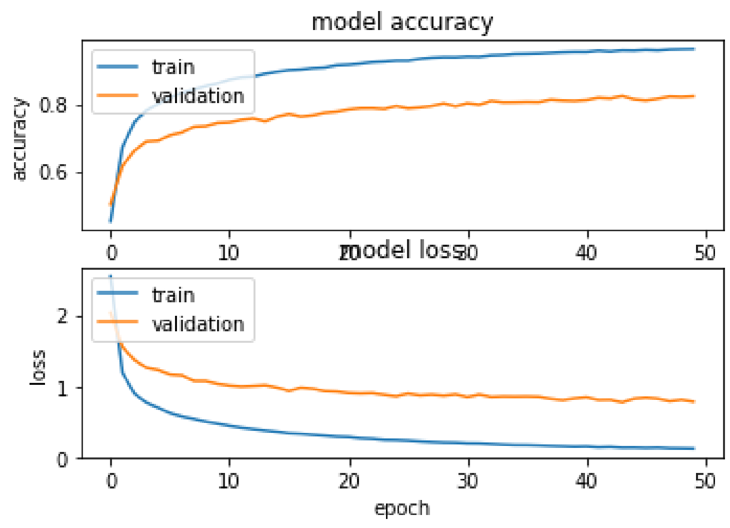


Fig.9 VGG-16 Fine Tuning

Refinement

After changing the batch size and learning rate, I was able to get a better accuracy on the fine-tuning model with 0.94 accuracy, which meet my expectations considering tuning gain vs time.

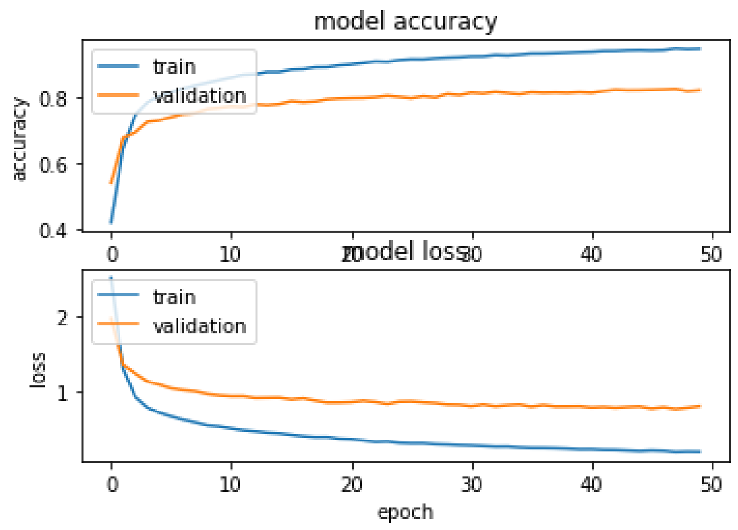


Fig.10 VGG-16 Fine Tuning Refinement

Since this model is trained on stock picture, which all have white background, pictures taken from user phone with varies background will have a lot of noise in the input so the model can't reach desired performance, to solve this issue, I added a simple cv grubcut function to remove the background of the input image for prediction.

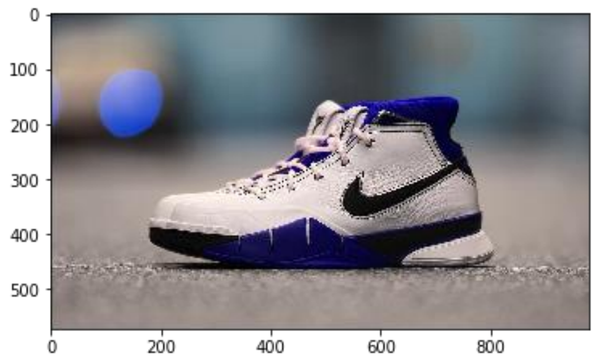


Fig.11-1 Original Image

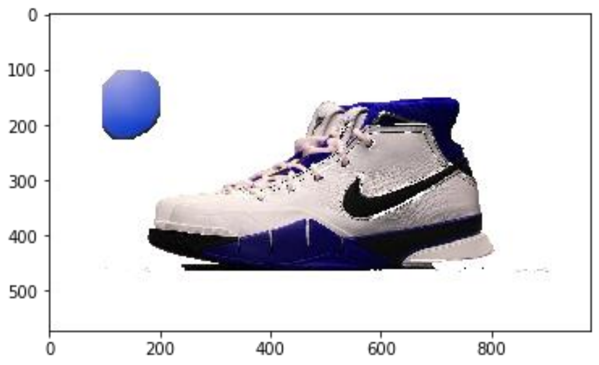


Fig.11-2 Original Image

IV. Result

Model Evaluation and Validation

The final fine-tuned model outperformed the benchmark models and all other models above. As seen from fig.9, both train/validation accuracy increased through each epoch and after 50 epoch the model achieved accuracy of 0.938 and log-loss of 0.254. Both parameters are with my expectations.

The test set data (unseen) have 10404 images, after making predictions it has an accuracy of 0.946 on the test set, which indicate this model still generalizes well to new data. Since its multi-classification model, a small perturbation in the train set should not affect the result significantly, also as seen in the training curve, very smooth no oscillating I would say this model is robust and the result from this model can be trusted.

Justification

Model	Accuracy	Log-loss
Benchmark CNN	**0.73**	**1.65**
VGG-16 Bottleneck Feature	**0.83**	**0.82**
VGG-16 Fine-tuning	**0.94**	**0.25**

The final model had a significant improvement from the benchmark model accuracy from 0.73 to 0.94 and log-loss from 1.65 to 0.25

V. Conclusion

Free-Form Visualization

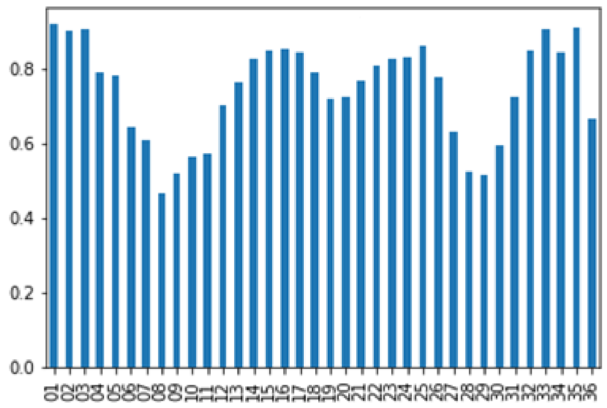


Fig.12 Accuracy vs Angle

From this bar chart we can see this model performance best on the side view (0-50 degree), have more error on front and back pictures (80-90 degree & 280-290 degree) which have fewer feature in the picture for model to predict also there are certain sneakers does have a same heel or very similar looks.

Reflection

The process of this project can be summarized as follows:

1. Propose the problem, looking for data
2. No public dataset available, build scraper
3. Pre-process the dataset
4. Build benchmark model for the classifier & initial training
5. Transfer learning training
6. Fine-tuning
7. Flask Web App
8. Adding features to improve result (removing background)

Through out the process, I feel like the fine-tuning part is most difficult in my experience, because it is computationally very expensive to train each time on GPUs when I try to tune the hyperparameters, this quite took me a few days, especially when the optimizer doesn’t have suitable momentum I can only see the accuracy/loss value ticking between epochs.

Still it is very satisfying to see the final model does perform exceptionally well than the benchmark model built from scratch.

Improvement

As mentioned above, this dataset I used still is not the image what we would see people taking from their phone, so there’s still background noise will affect the result. For improvement of the future work I would like to try to use other machine learning technique to help pre-process the image before the prediction, removing/replacing background with white so there’s minimal noise here.

Another improvement can be done on the data augmentation, as this dataset still unbalanced, few labels have way less images available than others, by doing this or gathering new images this dataset will be well-balanced. Training the model on a better-balanced dataset may give us a boost in the performance.