Mannequin Image Recognition in Retail Window Displays

The Problem: How can we reduce the need for manual data entry for an image-oriented retail database by establishing an image processing algorithm that determines the presence of mannequins within window display images? Ideally this automation should reduce two weeks of manual work to less than one day.

Context: WindowsWear is a company on the frontier of data collection in the fashion industry. Each month, the company captures over 1,000 images of window displays throughout fashion capitals around the world – London, Milan, Hong Kong, Paris, and New York. At the moment, the company manually examines each of the 1,000 images every month and determines the photo's main content and color scheme. The goal of the project is to implement a machine learning method that diminishes the need for manual data entry.

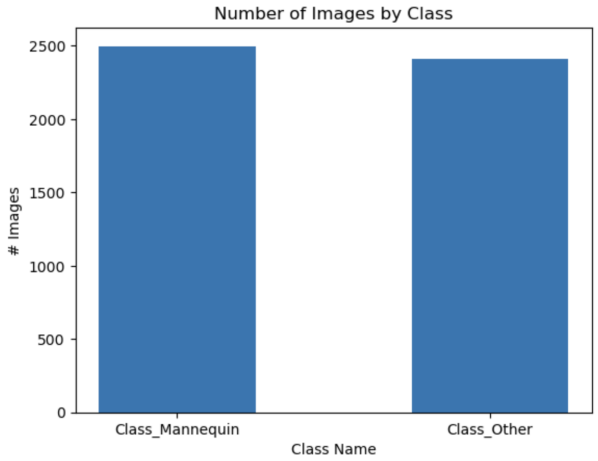
The Data: High resolution images were provided by Creative Director Raul Tovar via DropBox. For the sake of commercial interests, WindowsWear requested the images included in this published project to be condensed to a lower resolution.

Low resolution images used in this public analysis are included in a file located on my github: *https://github.com/chase-weber/Mannequin-Image-Detection/tree/main/data*

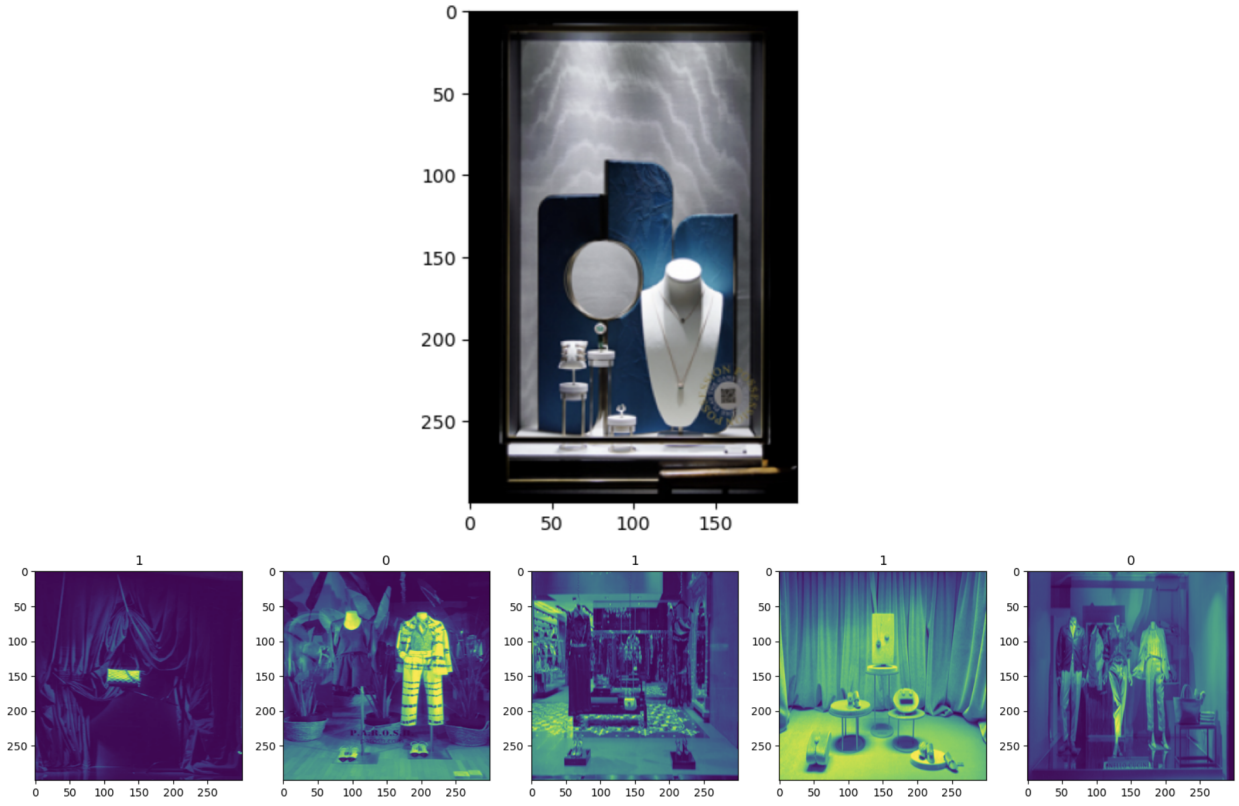
WindowsWear data includes over 5,000 colored photo images of retail fashion window displays around the globe.

Data Cleaning: I initially received the images in a dropbox folder. From there, I utilized Adobe Bridge to condense images to lower quality (as requested by the data provider), and manually sorted through each photo to determine whether or not it contains a mannequin. Photos were separated into two folders, one for mannequins, one for all others. The sorted data can be found on my github (link above).
I import the images into python using keras image_dataset_from_directory. I place a uniform shape on each photo, 300 by 300, enhance the contrast of the image, crop the image to contain approximately 75% of the original width and 85% of the original height, and convert to black and white as to avoid additional noise – color is not necessary to determine the presence (shape) of a mannequin in an image. I scale the data so all values are contained between 0 and 1, rather than 0 and 256. I also adjust the contrast of the image, by tuning this hyper-parameter .7 performed best.
All images containing a mannequin are labeled as 0 all images not containing a mannequin are labeled as 1.

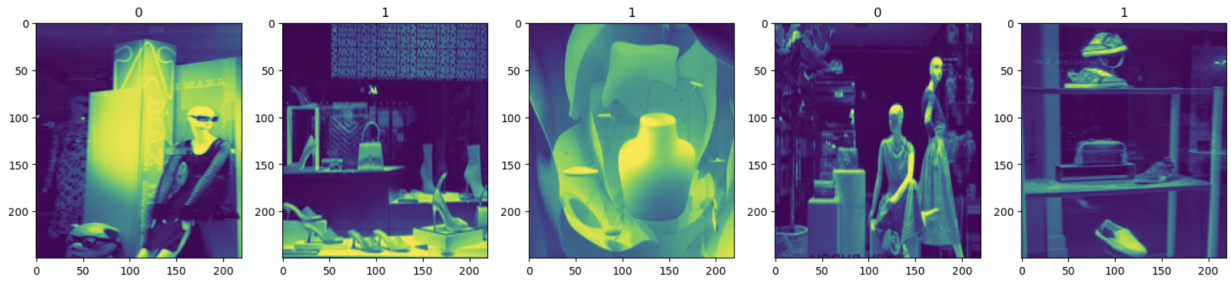
Exploratory Analysis: The first step in this exploratory data analysis is a check for imbalance data. Notice that of the 4,912 images imported, 2,499 contain mannequins and 2,413 do not; a fairly equal split:



We also do some initial checks of the photos to determine that they are importing correctly:

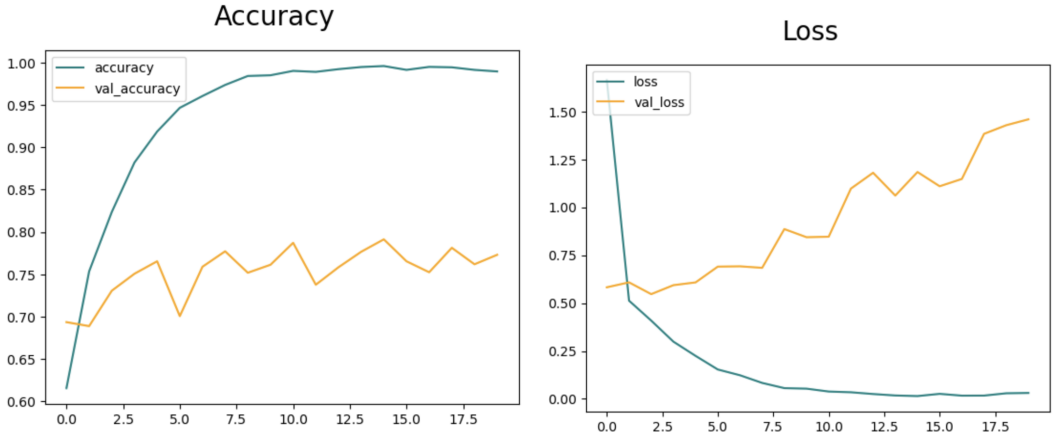


And that the cleaning is completed accurately:



Pre-processing: I split our data into a training set (70%), test set (20%), and validation set (10%).

Model Training:
The initial model runs a basic convolutional neural network with 3 hidden layers, maxpooling, “relu” activation, and “adam” optimizer.
We check the results - specifically comparing the training and validation accuracy and loss. We would expect to see the trends follow one another but we do not.



Test Scores: Precision : 0.79, Recall : 0.61, Accuracy : 0.75

As an enhancement, I build off of the previous model with regularization including .25 dropout layers.

I tune some parameters such as the contrast value (.7 is optimal), cropping size (250 x 220 is optimal), color v grayscale (grayscale is optimal), number of epochs (varies by model), and value of dropout (.25 generally is best).

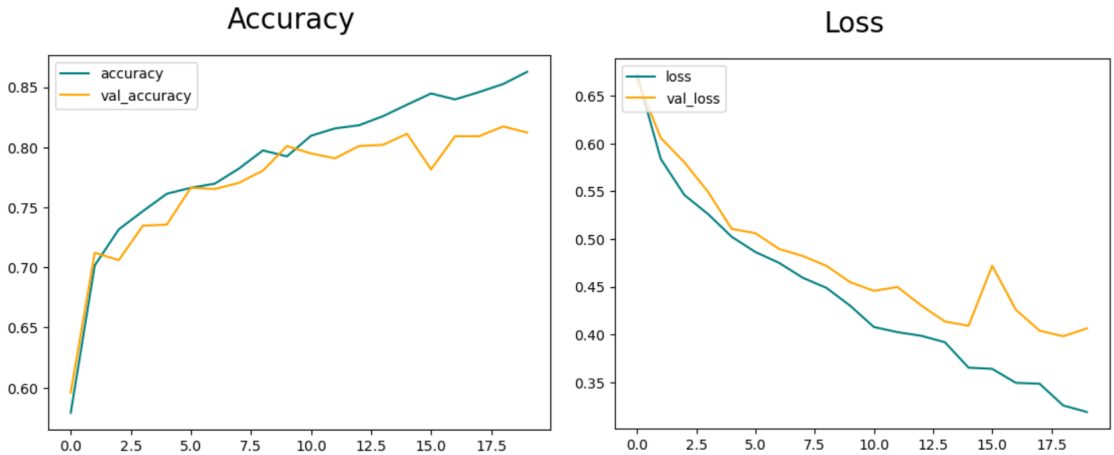
Through several iterations, I find several models that train well, but do not test well. I continue to tune until we see a balance between train and test scores.

After several iterations, the best performing model is selected - 5 hidden layers alternating between 16 and 32 nodes, relu activation, maxpooling, dropout of .25, and adam optimizer, and 20 epochs.

See the code below:

```
model = Sequential()
model.add(Conv2D(16, (2,2), 1, activation = 'relu', input_shape = (250,220,1)))
model.add(MaxPooling2D())
model.add(Dropout(.25))

model.add(Conv2D(32, (2,2), 1, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Dropout(.25))

model.add(Conv2D(16, (2,2), 1, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Dropout(.25))

model.add(Conv2D(32, (2,2), 1, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Dropout(.25))

model.add(Conv2D(16, (2,2), 1, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Dropout(.25))

model.add(Flatten())

model.add(Dense(256, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer = 'adam', loss = tf.losses.BinaryCrossentropy(), metrics = ['accuracy'])
```

Final Model Validation:



Test Scores: Precision : 0.80, Recall : 0.80, Accuracy : 0.82

Results:

Our final algorithm provides an accuracy of .82. WindowsWear will be able to accurately process and classify approximately 82% of their images in a fraction of the manual time.

Future Improvements:

In the future I hope to reconstruct this process with high-resolution images. I can also include images from other time-periods, cities, sources, etc…
I would like to analyze images for multi-label feature extraction and eventually expand the scope of the problem to extract information from images of store interiors, marketing campaigns, packaging, etc…