DeepFashion: **Dress Type Detection**

Prepared by
Ahjeong Yeom, Akhir Syabani, Kenji Laurens, Steven Wu

Deep Learning & Image Recognition, Spring 2022

Today’s Agenda

DeepFashion:
Dress Type Detection

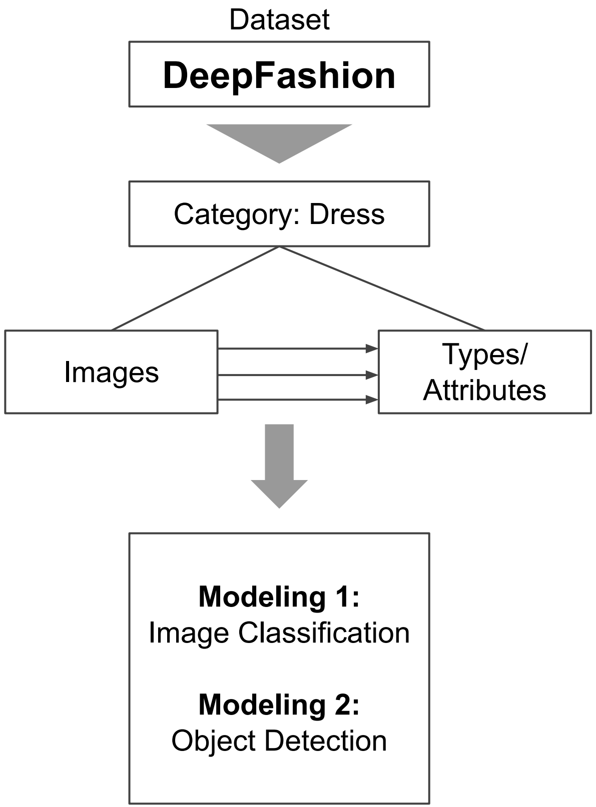
1. Abstract
2. EDA & Preprocessing
3. Model Building
   a. Modeling 1: Image Classification (ResNet, VGG, AlexNet, 2 Custom Models)
   b. Modeling 2: Object Detection (YOLO)
4. Model Deployment/ Management
5. Conclusion

Abstract

- **Problem:**
  - Fashion requires deeper understanding: it keeps evolving and producing new style variations.
  - Classifying different types of clothing is important as a base for more advanced works of deep learning in fashion.

- **Approach:**
  - Object detection model solves the problem of recognizing types of clothes.
  - From the the whole dataset of 200,000+ images on various clothing pieces, our team narrowed down the scope to female dress category and then focused on detecting three dress types (Mini, Midi, Maxi) due to hardware limitations.

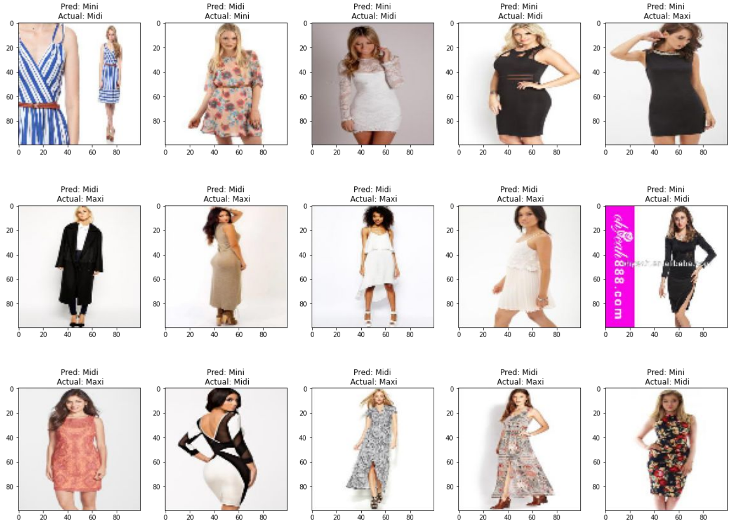
- **Results:**
  - >70% of accuracy detecting three types of dress.



EDA: Data Description

Dataset: **DeepFashion** Attribute Prediction Dataset ([Source](https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html))

- Multimedia Laboratory, The Chinese University of Hong Kong
- Images and attributes of 289,222 clothing articles of various types and categories
- Exactly 1000 attributes:
  - simple attributes (dress, trousers, shirt, blouse, etc)
  - complex attributes (plaid, a-line, collared, sheer, etc)
- Each clothing can and do possess multiple attributes
- Mostly frontal and focused on the clothes only
- Bounding boxes included, to further narrow down the image to each clothing only
- Background images are varied: empty or some environment
- Various resolutions, mostly around 300 x 300



Selected samples: female dress category
(attributes/types: mini, midi, maxi)

https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html

EDA: Data Description

Examples from Dataset

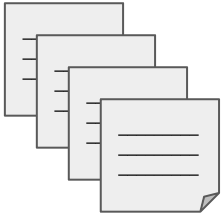
Different Dress Types:

Three dress types were trained given the restriction of the project scope.

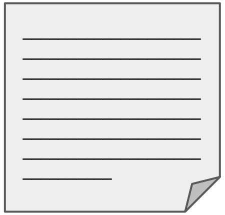
- **Mini dresses:** above the knee height around the thighs
- **Midi dresses:** crops at knee length
- **Maxi dresses:** longer than knee length, often to the ankles

EDA & Preprocessing:
Data Engineering & Feature Selection

Raw Data:
Multiple tables as .txt files + ~3GB img zip file



Merged



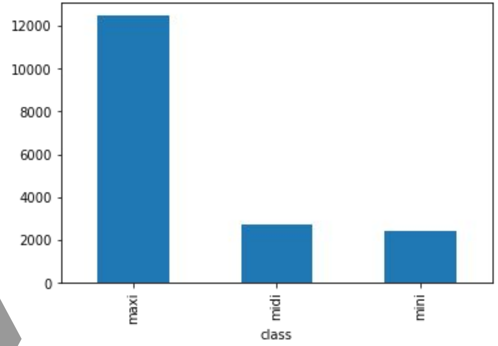
Selection:
- Image_name contains 'Dress'
- Attributes: 'Mini', 'Midi', 'Maxi

Cleaning:
Removed mislabeled images:
- No labels
- Multi-labeled

	image_name	x_1	x_2	y_1	y_2	mini	midi	maxi
198816	img/Open-Shoulder_Eyelash_Lace_Dress/img_00000...	21	152	64	254	1.0	NaN	NaN
198818	img/Open-Shoulder_Eyelash_Lace_Dress/img_00000...	34	141	42	202	1.0	NaN	NaN
198819	img/Open-Shoulder_Eyelash_Lace_Dress/img_00000...	60	152	39	202	1.0	NaN	NaN
198822	img/Open-Shoulder_Eyelash_Lace_Dress/img_00000...	1	207	53	300	1.0	NaN	NaN
198825	img/Open-Shoulder_Eyelash_Lace_Dress/img_00000...	1	207	59	300	1.0	NaN	NaN
...	...	...	...	...	...	...	...	...
284839	img/Chiffon-Paneled_Maxi_Dress/img_00000024.jpg	118	185	1	158	NaN	NaN	1.0
287902	img/Crocheted_Gauze_Maxi_Dress/img_00000020.jpg	57	151	15	281	NaN	NaN	1.0
215018	img/Dainty_A-Line_Dress/img_00000074.jpg	106	202	34	191	NaN	NaN	1.0
206860	img/Butterfly_Print_Maxi_Dress/img_00000066.jpg	106	201	53	300	NaN	NaN	1.0
201257	img/Watercolor_Ikat_M-Slit_Maxi_Dress/img_0000...	113	201	17	225	NaN	NaN	1.0

8185 rows × 8 columns

Sampling & Balancing Data:



	Clean Set	Sample
Maxi	12,468 (70%)	3,000 (36%)
Midi	2,756 (16%)	2,756 (34%)
Mini	2,429 (14%)	2,429 (30%)

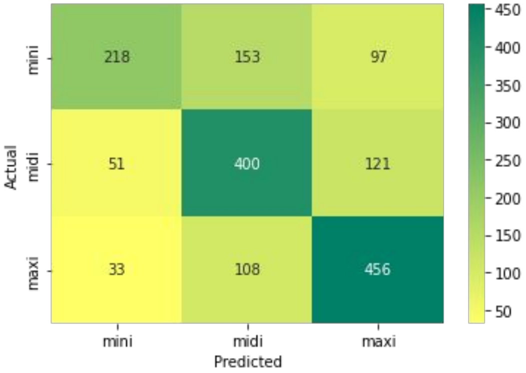
Pre-processing:
- Attributes are one-hot encoded (fill NA = 0)
- Two separate image arrays:
  - full image,
  - cropped to the bounding boxes
- All standardized and resized to (100, 100, 3) numpy array
- Other pre-processing to follow each model requirements

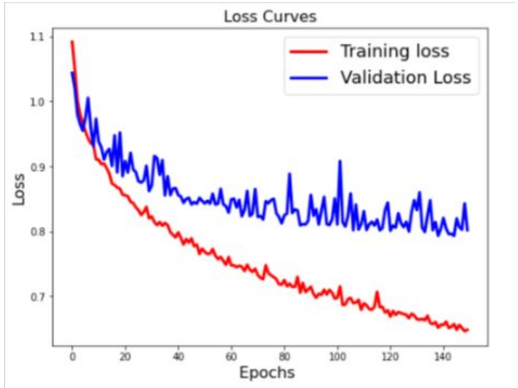
Modeling 1: Image Classification
ResNet50, VGG16, AlexNet, Custom Model 1 & 2

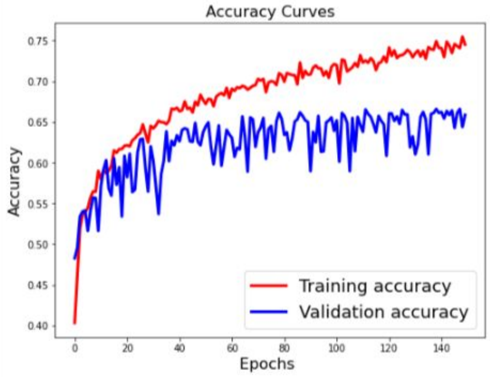
ResNet50	VGG16	AlexNet	Custom Model 1	Custom Model 2
Layers: • 48 convolutional • 1 max pooling • 1 avg pooling • Weight: imagenet • Activation: ReLU • Added dense layer with softmax • Early stopping Parameters: • 23 million	Layers: • 13 convolutional • 3 fully connected • 5 max pooling • Max pooling • Weight: imagenet • Activation: ReLU • Added dense layer with softmax • Early stopping Parameters: • 138 million	Layers: • 5 convolutional • 3 fully connected • 3 max pooling • Activation: ReLU • Added dense layer with softmax • Early stopping Parameters: • 62 million	Layers: • 4 convolutional • 3 fully connected • 2 max pooling • 4 dropouts • 1 batch normalization • Activation: ReLU, softmax • Early stopping Parameters: • 146 thousand	Layers: • 6 convolutional • 3 fully connected • 3 max pooling • 3 dropouts • 6 batch normalization • Activation: ReLU, softmax • Early stopping Parameters: • 28 million

Modeling 1: Image Classification
ResNet50

```
Epoch: 150/150
loss: 0.6483
accuracy: 0.7450
val_loss: 0.8019
val_accuracy: 0.6585
Est. runtime: 50ms/step
```



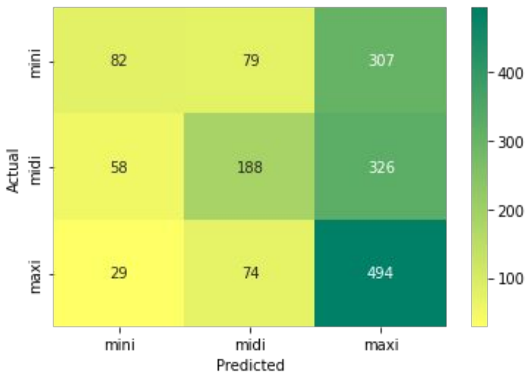


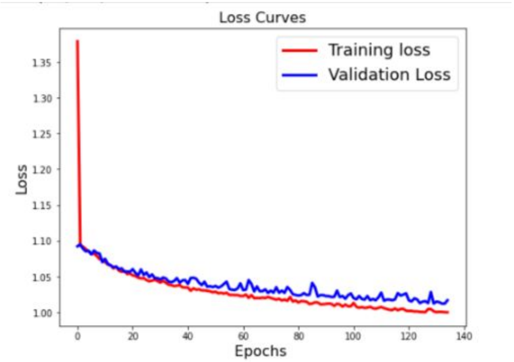


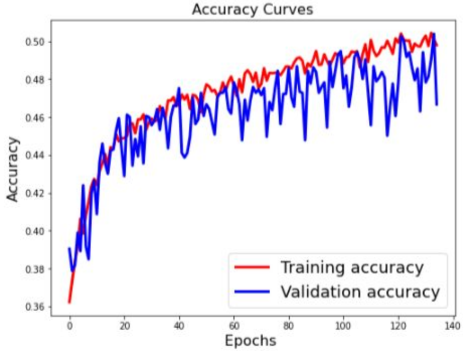
Good/acceptable accuracy, though validation set shows slower rate of improvement in each epoch

Modeling 1: Image Classification
VGG16

Epoch: 135/150
(early stopping)
loss: 1.0002
accuracy: 0.4980
val_loss: 1.0169
val_accuracy: 0.4667
Est. runtime: 50ms/step



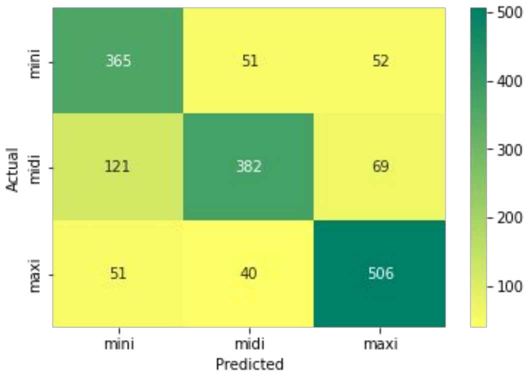




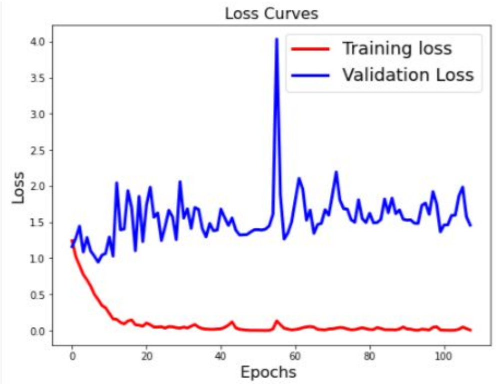
Lower accuracy, yet consistent performance across training and validation (minimized overfitting)

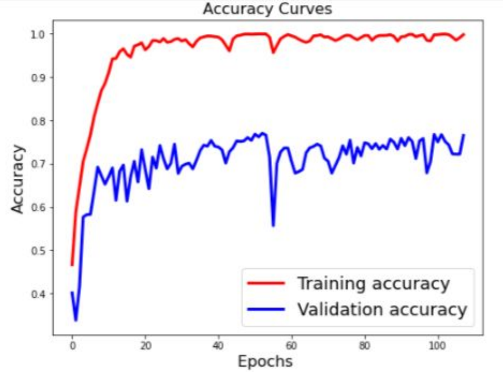
Modeling 1: Image Classification
AlexNet

Epoch: 108/150
(early stopping)
loss: 0.0056
accuracy: 0.9985
val_loss: 1.4573
val_accuracy: 0.7654
Est. runtime: 40ms/step



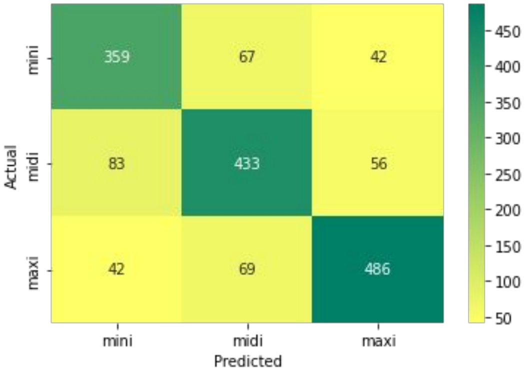
High accuracy, yet large gap / overfitting

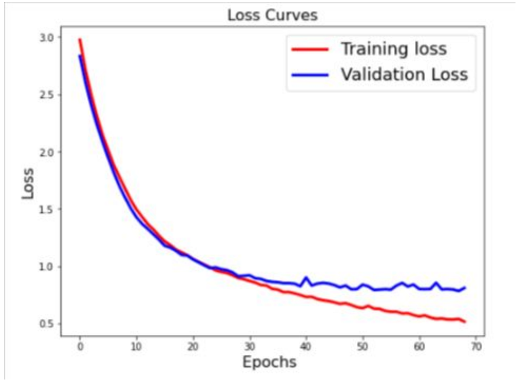


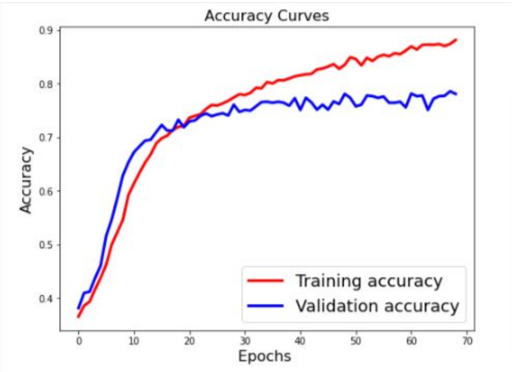


Modeling 1: Image Classification
Custom Model 1

```
Epoch: 69/150
(early stopping)
loss: 0.5162
accuracy: 0.8812
val_loss: 0.8094
val_accuracy: 0.7807
Est. runtime: 30ms/step
```



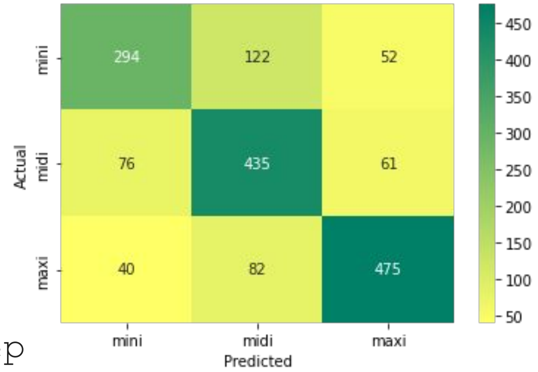


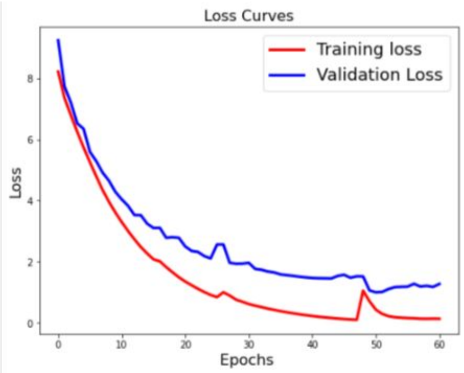


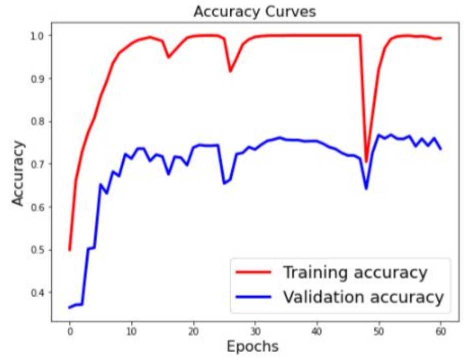
Simpler customized architecture with good/acceptable performance with decent consistency across training and validation

Modeling 1: Image Classification
Custom Model 2

```
Epoch: 61/150
(early stopping)
loss: 0.1309
accuracy: 0.9934
val_loss: 1.2711
val_accuracy: 0.7355
Est. runtime: 103ms/step
```







More complex customized architecture with good/acceptable performance but produce overfitting results

Modeling 1: Image Classification

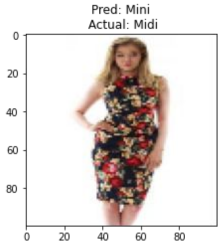
Comparison and Evaluation

ResNet50	VGG16	AlexNet	Custom Model 1	Custom Model 2
Loss • Training: 0.65 • Validation: 0.80 Accuracy: • Training: 75% • Validation: 66% # Layers: 50 # Params: 23mn Est. runtime: 50ms/step	Loss • Training: 1.00 • Validation: 1.02 Accuracy: • Training: 50% • Validation: 47% # Layers: 16 # Params: 138mn Est. runtime: 50ms/step	Loss • Training: 0.01 • Validation: 1.46 Accuracy: • Training: 99% • Validation: 77% # Layers: 8 # Params: 62mn Est. runtime: 30ms/step	Loss • Training: 0.52 • Validation: 0.81 Accuracy: • Training: 88% • Validation: 78% # Layers: 7 # Params: 146k Est. runtime: 40ms/step	Loss • Training: 0.13 • Validation: 0.74 Accuracy: • Training: 99% • Validation: 74% # Layers: 9 # Params: 28mn Est. runtime: 103ms/step

Modeling 1: Image Classification
Limitations and Further Improvements

Data:

- More distinction on ‘mini’ & ‘midi’ classifications
- Further clean-up on labeling (handling mislabeled data)
- Use higher image resolutions



Execution:

- Use larger hardware/software capacity
- Try deeper layers
- Further experiment with grayscale images (initial assessment: no difference)
- Add noise for better model generalization

Modeling 2: Object Detection

YOLO v3 - Summary

Modeling Design

- Tried to fit to a YOLOv3 custom object detection using ImageAI API
- A smaller subset was was used (only about 230 images for each category)
- Created annotations for all images
    - Annotations are in PASCAL Visual Object Classes (VOC) format
    - Used the bounding boxes provided

Execution

- Training took a very long time, 13 epochs took 2 hours
- Ran out of Colab GPU runtime allocation

```
Epoch 1/13
1120/1120 [==============================] - 612s 546ms/step - loss: 115.6004 - yolo_layer_6_loss: 13.8674 - yolo_layer_7_loss: 26.4391
Epoch 2/13
1120/1120 [==============================] - 595s 531ms/step - loss: 23.4449 - yolo_layer_6_loss: 2.4267 - yolo_layer_7_loss: 5.4595 - 
Epoch 3/13
1120/1120 [==============================] - 607s 541ms/step - loss: 20.3098 - yolo_layer_6_loss: 2.5715 - yolo_layer_7_loss: 5.3965 - 
Epoch 4/13
1120/1120 [==============================] - 594s 530ms/step - loss: 18.9928 - yolo_layer_6_loss: 2.5498 - yolo_layer_7_loss: 5.3762 - 
Epoch 5/13
1120/1120 [==============================] - 597s 533ms/step - loss: 17.8712 - yolo_layer_6_loss: 2.2763 - yolo_layer_7_loss: 5.2335 - 
Epoch 6/13
1120/1120 [==============================] - 604s 539ms/step - loss: 17.2642 - yolo_layer_6_loss: 2.4341 - yolo_layer_7_loss: 5.1159 - 
Epoch 7/13
1120/1120 [==============================] - 591s 527ms/step - loss: 16.6548 - yolo_layer_6_loss: 2.5199 - yolo_layer_7_loss: 4.6554 - 
Epoch 8/13
1120/1120 [==============================] - 601s 536ms/step - loss: 16.0430 - yolo_layer_6_loss: 2.4591 - yolo_layer_7_loss: 4.5810 - 
Epoch 9/13
1120/1120 [==============================] - 599s 534ms/step - loss: 15.8214 - yolo_layer_6_loss: 2.3161 - yolo_layer_7_loss: 4.7079 - 
Epoch 10/13
1120/1120 [==============================] - 599s 535ms/step - loss: 15.6048 - yolo_layer_6_loss: 2.3639 - yolo_layer_7_loss: 4.4318 - 
Epoch 11/13
1120/1120 [==============================] - 597s 533ms/step - loss: 15.2422 - yolo_layer_6_loss: 2.1936 - yolo_layer_7_loss: 4.4298 - 
Epoch 12/13
1120/1120 [==============================] - 603s 538ms/step - loss: 15.3286 - yolo_layer_6_loss: 2.5839 - yolo_layer_7_loss: 4.3280 - 
Epoch 13/13
1120/1120 [==============================] - 602s 537ms/step - loss: 14.7197 - yolo_layer_6_loss: 2.2407 - yolo_layer_7_loss: 3.9781 - 
```

Modeling 2: Object Detection
YOLO v3 - Video Detection: Model Creation & Engineering

- Full images used (uncropped to bounding boxes)
- Full resolution used for each image
- Bounding boxes were already part of dataset
- Had to retrain the model from scratch according to using ImageAI API


```xml
<?xml version="1.0"?>
- <annotation>
    <folder>train</folder>
    <filename>27.jpg</filename>
    <path>./train/images/27.jpg</path>
  - <source>
      <database>MMM</database>
    </source>
  - <size>
      <width>300</width>
      <height>300</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>Mini</name>
      <pose>Frontal</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
      <occluded>0</occluded>
    - <bndbox>
        <xmin>83</xmin>
        <xmax>235</xmax>
        <ymin>67</ymin>
        <ymax>297</ymax>
      </bndbox>
    </object>
</annotation>
```


```
>> train     >> images      >> img_1.jpg  (shows Object_1)
             >> images      >> img_2.jpg  (shows Object_2)
             >> images      >> img_3.jpg  (shows Object_1, Object_3 and Object_n)
             >> annotations >> img_1.xml  (describes Object_1)
             >> annotations >> img_2.xml  (describes Object_2)
             >> annotations >> img_3.xml  (describes Object_1, Object_3 and Object_n)

>> validation   >> images      >> img_151.jpg (shows Object_1, Object_3 and Object_n)
                >> images      >> img_152.jpg (shows Object_2)
                >> images      >> img_153.jpg (shows Object_1)
                >> annotations >> img_151.xml (describes Object_1, Object_3 and Object_n)
                >> annotations >> img_152.xml (describes Object_2)
                >> annotations >> img_153.xml (describes Object_1)
```

```python
trainer = DetectionModelTrainer()
trainer.setGpuUsage(1)
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory('./drive/MyDrive/Project YOLO')
trainer.setTrainConfig(object_names_array=["Mini","Midi",'Maxi'], batch_size=4, num_experiments=30)
trainer.trainModel()
```

Modeling 2: Object Detection

YOLO v3 - Custom Object Detection Video

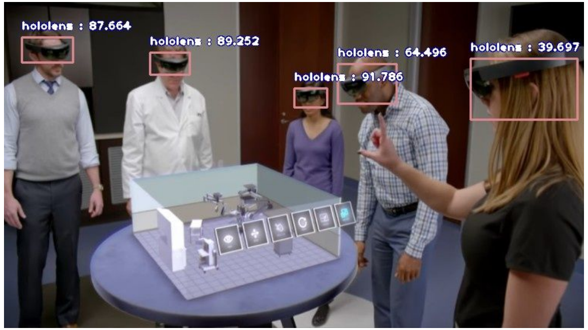
- Pinterest video of a woman trying out different dresses
- Tried all mini, midi and maxi dresses
- Text showing type of dress is from the video and not annotated by the model
- Model prediction is unfortunately annotated out of frame
  - Due to training images lacking much noise

https://www.pinterest.com/pin/mini-midi-maxis-oh-my-video--209628557647083293/

Modeling 2: Object Detection
YOLO v3 - Evaluation

- Unable to create robust model due to lacking GPU allocation
- The model was most confident with Maxi dresses, and almost unable to detect Mini dress
- Very low mAP score (Mean Average Precision)

```
Evaluation samples:  140
Using IoU:  0.5
Using Object Threshold:  0.3
Using Non-Maximum Suppression:  0.5
Maxi: 0.4467
Midi: 0.1051
Mini: 0.0320
mAP: 0.1946
```



- Our training images lack any noise
- All images are highly focused on the dresses themselves
- Might cause the model to assume the dress should take up the whole screen
- Since label annotations are placed above detection bounds, our demo video did not show any annotations

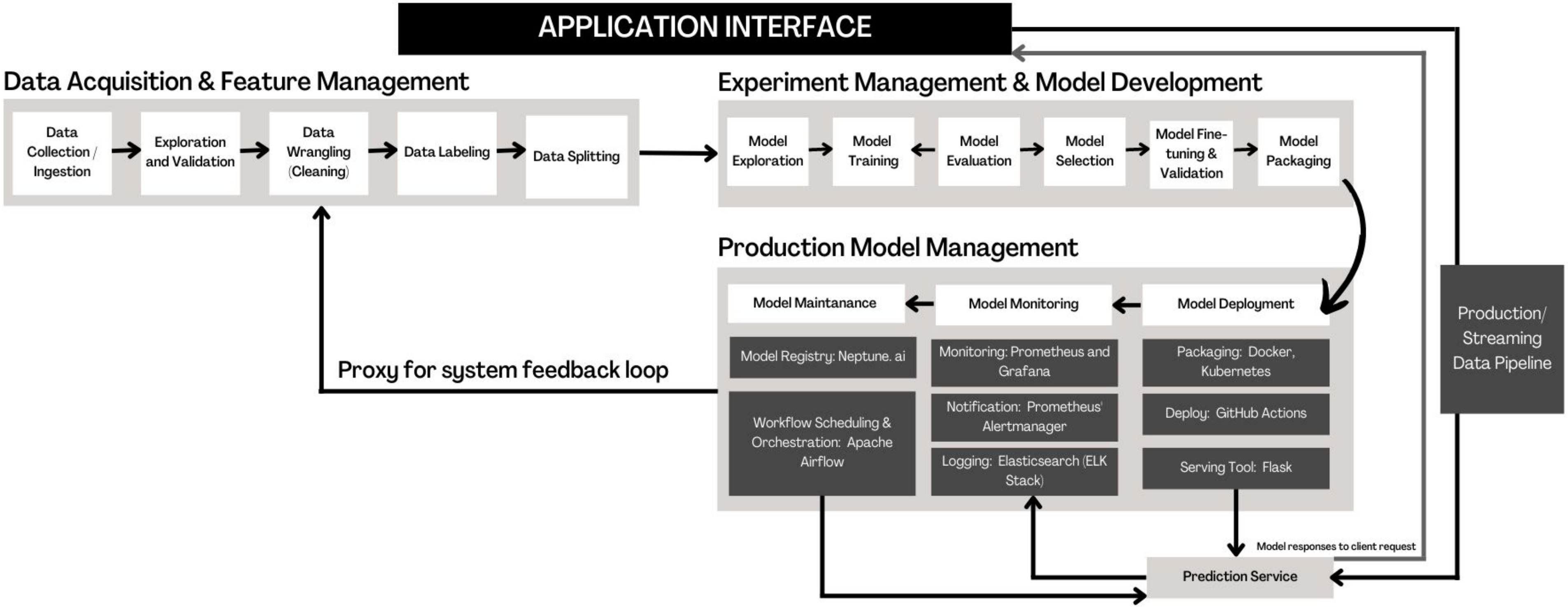
Modeling 2: Object Detection
YOLO v3 - Limitations and Further Improvements

- Use images that has a background instead of the whole image (noisy images)
- Run more epochs
- Try transfer learning with trained detection model or other general models
- Updated version of YOLOv5 can be used for improvement

Use Case

- Image Classification:
  - Fashion Recommendation Powered By DL
  - Attributes extracted from image classification can help enhance the fashion companies' recommendation to consumers based on purchase or search history
  - User can hover on image and see specific article label and this hover action can micro-logged and used for building fine-tuned recommendation as fashion outfit photos generally contain many items in one outfit.
- Object Detection
  - Video scene labeling is a popular DL application. Now fashion object detection can add more complex information to the description of the video scene. (ie. A woman with leopard maxi dress and blue eyes are carrying her weekender bag)
  - Tiktok and Reels can use this algorithm to extract information from videos and read the real-time fashion trends globally

Model Deployment/Management



https://towardsdatascience.com/deploying-an-image-classification-web-app-with-python-3753c46bb79
https://www.analyticsvidhya.com/blog/2020/07/deploy-an-image-classification-model-using-flask/
https://www.fairwinds.com/blog/heroku-vs.-kubernetes-the-big-differences-you-should-know
https://machinelearningmastery.com/update-neural-network-models-with-more-data/
https://neptune.ai/blog/mlops-architecture-guide

Model Deployment/Management

- Given that new input data will come in rapidly (fashion evolves rapidly), dynamic training architecture is optimal where model is re-trained with constant data feed into the data pipeline
- Integration of automated data versioning, monitoring, and continuous deployment of the model is essential using a scheduling tool/ trigger-based orchestration tool
- Kubernetes will be a good scalable option for our runtime environment as it is portable, executable anywhere and model-agnostic. Also, it can optimize the resources for ML workload

Conclusion

- Selecting and preparing the right training data is crucial
  - Ensure enough variance
  - Correct labeling
  - Use image augmentation when possible

- Pre-trained models sometimes are not the way to go
  - Need to balance computing power and expected accuracy
  - Always need to retrain the model and fine-tune the parameters and layers for a better fit

- Careful adjustment to manage model tendency to overfitting

Thank You!

References:

- https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion/AttributePrediction.html
- https://towardsdatascience.com/deploying-an-image-classification-web-app-with-python-3753c46bb79
- https://www.analyticsvidhya.com/blog/2020/07/deploy-an-image-classification-model-using-flask/
- https://www.fairwinds.com/blog/heroku-vs.-kubernetes-the-big-differences-you-should-know
- https://machinelearningmastery.com/update-neural-network-models-with-more-data/

- https://towardsdatascience.com/deploying-an-image-classification-web-app-with-python-3753c46bb79
- https://www.analyticsvidhya.com/blog/2020/07/deploy-an-image-classification-model-using-flask/
- https://www.fairwinds.com/blog/heroku-vs.-kubernetes-the-big-differences-you-should-know
- https://machinelearningmastery.com/update-neural-network-models-with-more-data/

- https://neptune.ai/blog/mlops-architecture-guide