Fashion Clothing Classification Report

Chuhao Luo V00987151
Pengyuan Huang V00973109
Han Zhang V00974121
Department of Computer Science,University of Victoria

July 1, 2023

Abstract

In this report, we have discussed the application of three machine learning methods in fashion product classification. The goal of this project is to enhance the understanding of classification using multiple datasets, preprocessing techniques, and classification methods. Clothing classification is still widely used in many industries and will continue to progress with new fashion trends. From this project, it can be observed that classifying clothes is not as accurate as other types of classification because many clothes have similar shapes and sizes, such as jackets and long-sleeve shirts. Our objective is to find a solution to this problem through data engineering, machine learning, and the techniques learned in this course.

Contents

1 Introduction **1**

2 Dataset Prepossess **2**
2.1 Dataset description . . . 2
2.2 Method for analysis . . . 3
2.3 Model Training . . . 6
2.3.1 RandomForest Classifier . . . 6
2.3.2 CNN . . . 8
2.3.3 ResNet . . . 13
2.3.4 Analysis . . . 15
2.3.5 Conclusion . . . 15
2.4 Reference . . . 16

1 Introduction

The problem we are addressing is that the fashion industry needs a viable way to efficiently and accurately classify images of fashion products in order to properly categorize the type of fashion item. The current human (with no fashion experience) performance for this classification is measured at 83.5 percent accuracy. We aim to develop models that can be made into an API to solve this problem with higher accuracy and speed. This API could then be used in many companies and systems, including automating a thrift store clothing drop-off, where a machine takes an image of the fashion item and sends it to our model to get a label. These models could also be used for websites that receive images of fashion and need to correctly label them.

In our project, we take on the challenge of classifying fashion products and attempt to find an optimal method for determining the type of clothing item in an image. We create various models with varying accuracies based on two fashion datasets. These models are trained using advanced machine learning and deep learning techniques. We utilize a large amount of fashion image data for training and validation, enhancing the models' ability to recognize different types of fashion items.

Through our research and experimentation, we aim to develop models with higher accuracy and faster processing speeds. These models will be packaged into an easy-to-use API, allowing other companies and systems to integrate and apply them. For example, in the scenario of automating clothing drop-off at a thrift store, our API can be embedded in an image recognition system, enabling machines to quickly and accurately classify garments. Additionally, fashion websites can utilize our API by sending user-uploaded fashion images to our model and obtaining correct labels, providing a better user experience.

Through our work, we hope to provide an advanced solution for the fashion industry, improving the accuracy and efficiency of image classification and facilitating accurate categorization of fashion products, leading to an enhanced fashion experience.

2 Dataset Prepossess

2.1 Dataset description

In this project we used two different fashion datasets in order to better understand Machine Learning categorization on multiple datasets.

- Kaggle Dataset

  The first dataset we used is a clothing dataset which found on Kaggle and named ”Clothing”. For this dataset , it’s contain more than 5000 images with 20 different classes. We import library ”Pandas” to read the data from Kaggle CSV file, From figure 1 can see the information.

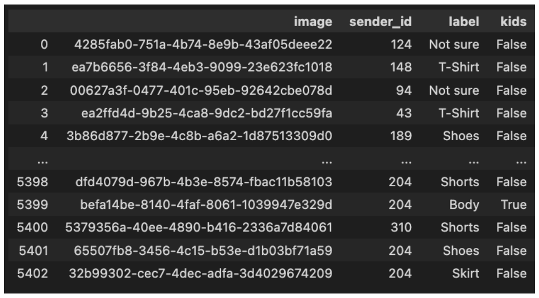
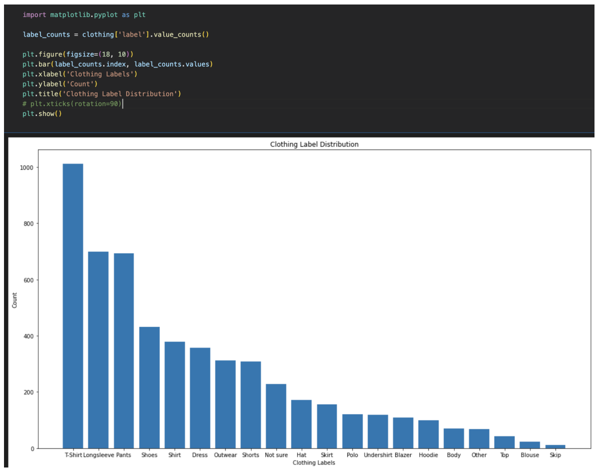


	image	sender_id	label	kids
0	4285fab0-751a-4b74-8e9b-43af05deee22	124	Not sure	False
1	ea7b6656-3f84-4eb3-9099-23e623fc1018	148	T-Shirt	False
2	00627a3f-0477-401c-95eb-92642cbe078d	94	Not sure	False
3	ea2ffd4d-9b25-4ca8-9dc2-bd27f1cc59fa	43	T-Shirt	False
4	3b86d877-2b9e-4c8b-a6a2-1d87513309d0	189	Shoes	False
...	...	...	...	...
5398	dfd4079d-967b-4b3e-8574-fbac11b58103	204	Shorts	False
5399	befa14be-8140-4faf-8061-1039947e329d	204	Body	True
5400	5379356a-40ee-4890-b416-2336a7d84061	310	Shorts	False
5401	65507fb8-3456-4c15-b53e-d1b03bf71a59	204	Shoes	False
5402	32b99302-cec7-4dec-adfa-3d4029674209	204	Skirt	False

Figure 1: Kaggle dataset

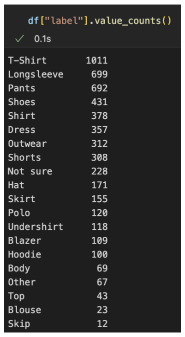
  These images were contributed by 32 individuals worldwide who captured photos of their clothing items and provided corresponding labels. The remaining 3000 images were sourced from Tagias.com. see the figure 2



```
import matplotlib.pyplot as plt

label_counts = clothing['label'].value_counts()

plt.figure(figsize=(18, 10))
plt.bar(label_counts.index, label_counts.values)
plt.xlabel('Clothing Labels')
plt.ylabel('Count')
plt.title('Clothing Label Distribution')
# plt.xticks(rotation=90)
plt.show()
```

Figure 2: Kaggle dataset

Based on Figure 1, the image illustrates the multiple categories present in the dataset and the quantity associated with each category. In addition, we can see that in this dataset, T-shirt has the highest quantity, while blouse has the lowest quantity. In figure 3, we using the "value.count()" function, you can obtain the specific values for each category.



```
df["label"].value_counts()
✓ 0.1s

T-Shirt       1011
Longsleeve     699
Pants          692
Shoes          431
Shirt          378
Dress          357
Outwear        312
Shorts         308
Not sure       228
Hat            171
Skirt          155
Polo           120
Undershirt     118
Blazer         109
Hoodie         100
Body            69
Other           67
Top             43
Blouse          23
Skip            12
```

Figure 3: Value.count()

some of categories' quantity are low, which is hard for train the model, based on this situation, we will use top six classes include 'Dress', 'Longsleeve', 'Outwear', 'Pants', 'Shirt', 'Shoes','T-Shirt', they have larger quantity.

- Fashion-Mnist
  The second dataset, we decide using Fashion-Mnist from Keras, it's a really famous Library in Machine learning area and it suitable for student.

  In this datset, it contain 70,000 clothing in 10 different categories. Include, T-shirt, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Trouser and Boot. All of the pictures are 28x28 gray-scale image.

  Based on the first one dataset, Fashion-Mnist have to cut 3 classes, so we keep remain 7 class: T-shirt, trouser, pullover, dress, coat, shirt, sneaker

2.2 Method for analysis

For the project, we will use three different algorithm method and compare the accuracy then tried to find out which one is better for the fashion classification.

Name	Method
Chuhao Luo	RandomForest
Zihan Zhang	ResNet
Alex Huang	CNN

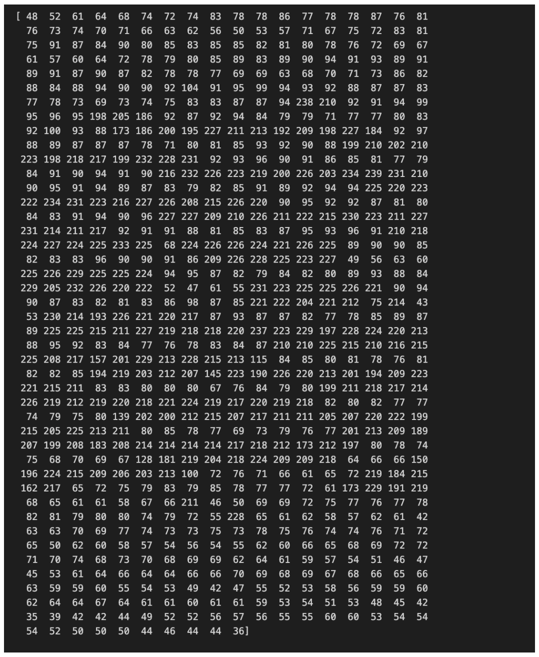
Table 1: Method for each member.

before the train the model, we should process the image. Fashion-mnist do not need to process, so we only need process the dataset from Kaggle. In the Processing, We use CV2 read the image, and gray-scale the image; in addition, we resize the image to the (28,28), after resize we use "resized-img.flatten()" transfer the image to a one dimension list then append to a final list.

During the Processing, I found that for the Kaggle dataset, each of the picture they have the background which makes one dimension list looks disorder. Figure 4 is the Initial result.

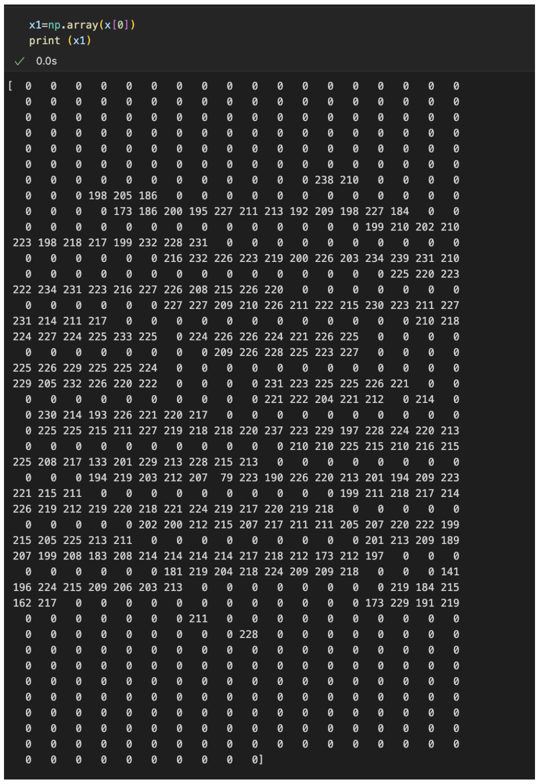
Based on this array, we cannot rebuild back to a image. but after we remove the background. see figure 5
From the figure 5, we can noticed that this array have some regularity and we used "CV2.imshow()" rebuild the image, it's looks like the original picture. hanzhang0621@gmail.com

From the Figure 7 and Figure 6, we can see that the rebuild picture are similar with original picture. So after the background remove the array will be more clear. In addition, I try to let the data array flip



```
[ 48  52  61  64  68  74  72  74  83  78  78  86  77  78  78  87  76  81
  76  73  74  70  71  66  63  62  56  50  53  57  71  67  75  72  83  81
  75  91  87  84  90  80  85  83  85  85  82  81  80  78  76  72  69  67
  61  57  60  64  72  78  79  80  85  89  83  89  90  94  91  93  89  91
  89  91  87  90  87  82  78  78  77  69  69  63  68  70  71  73  86  82
  88  84  88  94  90  90  92 104  91  95  99  94  93  92  88  87  87  83
  77  78  73  69  73  74  75  83  83  87  87  94 238 210  92  91  94  99
  95  96  95 198 205 186  92  87  92  94  84  79  79  71  77  77  80  83
  92 100  93  88 173 186 200 195 227 211 213 192 209 198 227 184  92  97
  88  89  87  87  87  78  71  80  81  85  93  92  90  88 199 210 202 210
 223 198 218 217 199 232 228 231  92  93  96  90  91  86  85  81  77  79
  84  91  90  94  91  90 216 232 226 223 219 200 226 203 234 239 231 210
  90  95  91  94  89  87  83  79  82  85  91  89  92  94  94 225 220 223
 222 234 231 223 216 227 226 208 215 226 220  90  95  92  92  87  81  80
  84  83  91  94  90  96 227 227 209 210 226 211 222 215 230 223 211 227
 231 214 211 217  92  91  91  88  81  85  83  87  95  93  96  91 210 218
 224 227 224 225 233 225  68 224 226 226 224 221 226 225  89  90  90  85
  82  83  83  96  90  90  91  86 209 226 228 225 223 227  49  56  63  60
 225 226 229 225 225 224  94  95  87  82  79  84  82  80  89  93  88  84
 229 205 232 226 220 222  52  47  61  55 231 223 225 225 226 221  90  94
  90  87  83  82  81  83  86  98  87  85 221 222 204 221 212  75 214  43
  53 230 214 193 226 221 220 217  87  93  87  87  82  77  78  85  89  87
  89 225 225 215 211 227 219 218 218 220 237 223 229 197 228 224 220 213
  88  95  92  83  84  77  76  78  83  84  87 210 210 225 215 210 216 215
 225 208 217 157 201 229 213 228 215 213 115  84  85  80  81  78  76  81
  82  82  85 194 219 203 212 207 145 223 190 226 220 213 201 194 209 223
 221 215 211  83  83  80  80  80  67  76  84  79  80 199 211 218 217 214
 226 219 212 219 220 218 221 224 219 217 220 219 218  82  80  82  77  77
  74  79  75  80 139 202 200 212 215 207 217 211 211 205 207 220 222 199
 215 205 225 213 211  80  85  78  77  69  73  79  76  77 201 213 209 189
 207 199 208 183 208 214 214 214 214 217 218 212 173 212 197  80  78  74
  75  68  70  69  67 128 181 219 204 218 224 209 209 218  64  66  66 150
 196 224 215 209 206 203 213 100  72  76  71  66  61  65  72 219 184 215
 162 217  65  72  75  79  83  79  85  78  77  77  72  61 173 229 191 219
  68  65  61  61  58  67  66 211  46  50  69  69  72  75  77  76  77  78
  82  81  79  80  80  74  79  72  55 228  65  61  62  58  57  62  61  42
  63  63  70  69  77  74  73  73  75  73  78  75  76  74  74  76  71  72
  65  50  62  60  58  57  54  56  54  55  62  60  66  65  68  69  72  72
  71  70  74  68  73  70  68  69  69  62  64  61  59  57  54  51  46  47
  45  53  61  64  66  64  64  66  66  70  69  68  69  67  68  66  65  66
  63  59  59  60  55  54  53  49  42  47  55  52  53  58  56  59  59  60
  62  64  64  67  64  61  61  60  61  61  59  53  54  51  53  48  45  42
  35  39  42  42  44  49  52  52  56  57  56  55  55  60  60  53  54  54
  54  52  50  50  50  44  46  44  44  36]
```

Figure 4: Before



```
x1=np.array(x[0])
print (x1)
```

✓ 0.0s

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0 238 210   0   0   0   0
   0   0   0 198 205 186   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0 173 186 200 195 227 211 213 192 209 198 227 184   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0 199 210 202 210
 223 198 218 217 199 232 228 231   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0 216 232 226 223 219 200 226 203 234 239 231 210
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 225 220 223
 222 234 231 223 216 227 226 208 215 226 220   0   0   0   0   0   0   0
   0   0   0   0   0   0 227 227 209 210 226 211 222 215 230 223 211 227
 231 214 211 217   0   0   0   0   0   0   0   0   0   0   0   0 210 218
 224 227 224 225 233 225   0 224 226 226 224 221 226 225   0   0   0   0
   0   0   0   0   0   0   0   0 209 226 228 225 223 227   0   0   0   0
 225 226 229 225 225 224   0   0   0   0   0   0   0   0   0   0   0   0
 229 205 232 226 220 222   0   0   0   0 231 223 225 225 226 221   0   0
   0   0   0   0   0   0   0   0   0   0 221 222 204 221 212   0 214   0
   0 230 214 193 226 221 220 217   0   0   0   0   0   0   0   0   0   0
   0 225 225 215 211 227 219 218 218 220 237 223 229 197 228 224 220 213
   0   0   0   0   0   0   0   0   0   0   0 210 210 225 215 210 216 215
 225 208 217 133 201 229 213 228 215 213   0   0   0   0   0   0   0   0
   0   0   0 194 219 203 212 207  79 223 190 226 220 213 201 194 209 223
 221 215 211   0   0   0   0   0   0   0   0   0   0 199 211 218 217 214
 226 219 212 219 220 218 221 224 219 217 220 219 218   0   0   0   0   0
   0   0   0   0   0 202 200 212 215 207 217 211 211 205 207 220 222 199
 215 205 225 213 211   0   0   0   0   0   0   0   0   0 201 213 209 189
 207 199 208 183 208 214 214 214 214 217 218 212 173 212 197   0   0   0
   0   0   0   0   0   0 181 219 204 218 224 209 209 218   0   0   0 141
 196 224 215 209 206 203 213   0   0   0   0   0   0   0   0 219 184 215
 162 217   0   0   0   0   0   0   0   0   0   0   0   0 173 229 191 219
   0   0   0   0   0   0   0 211   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0 228   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

Figure 5: After

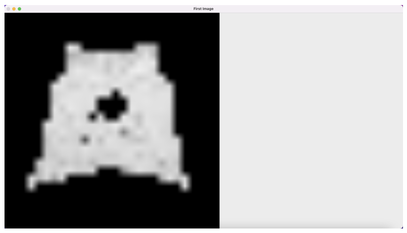


Figure 6: Rebuild

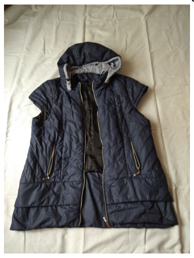
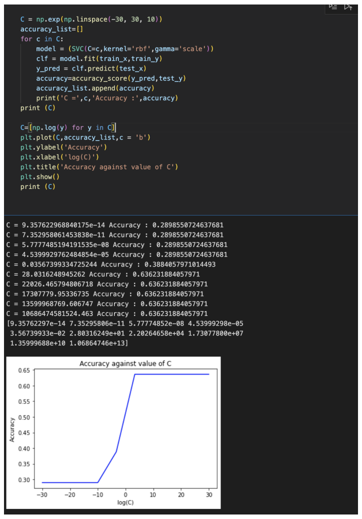


Figure 7: Original

2.3 Model Training

2.3.1 RandomForest Classifier

For the RandomForest Classifier, was completed by Chuhao Luo. At the begin, I decided use the SVM for the classification, but after i combine two datasets together as the training set, the training time be come really long, takes more the 60 mins. So I have noticed that in terms of efficiency and running speed, SVM is not the fastest or most suitable option.



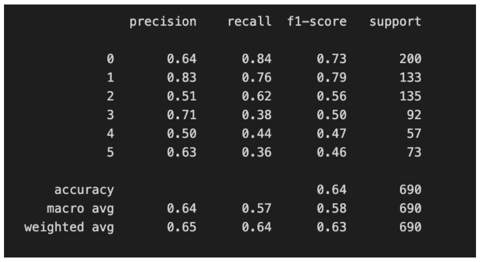
```
C = np.exp(np.linspace(-30, 30, 10))
accuracy_list=[]
for c in C:
    model = (SVC(C=c,kernel='rbf',gamma='scale'))
    clf = model.fit(train_x,train_y)
    y_pred = clf.predict(test_x)
    accuracy=accuracy_score(y_pred,test_y)
    accuracy_list.append(accuracy)
    print('C =',c,'Accuracy :',accuracy)
print (C)

C=[np.log(y) for y in C]
plt.plot(C,accuracy_list,c = 'b')
plt.ylabel('Accuracy')
plt.xlabel('log(C)')
plt.title('Accuracy against value of C')
plt.show()
print (C)
```

```
C = 9.357622968840175e-14 Accuracy : 0.2898550724637681
C = 7.352958061453838e-11 Accuracy : 0.2898550724637681
C = 5.7777485194191535e-08 Accuracy : 0.2898550724637681
C = 4.5399929762484854e-05 Accuracy : 0.2898550724637681
C = 0.03567399334725244 Accuracy : 0.3884057971014493
C = 28.0316248945262 Accuracy : 0.636231884057971
C = 22026.465794806718 Accuracy : 0.636231884057971
C = 17307779.95336735 Accuracy : 0.636231884057971
C = 13599968769.606747 Accuracy : 0.636231884057971
C = 10686474581524.463 Accuracy : 0.636231884057971
[9.35762297e-14 7.35295806e-11 5.77774852e-08 4.53999298e-05
 3.56739933e-02 2.80316249e+01 2.20264658e+04 1.73077800e+07
 1.35999688e+10 1.06864746e+13]
```

Figure 8: SVM

- Kaggle Dataset

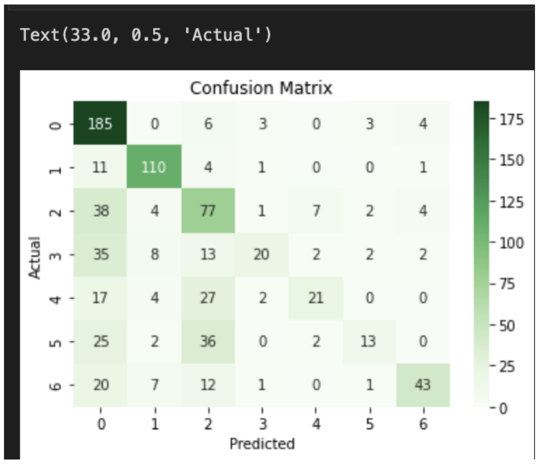
For the figure 8, the model is for train and test on Kaggle dataset. From the figure 8, it's can found that, if "C" Value large the accuracy will be more high, and highest accuracy is about 0.63. But it takes about more than 50 minute to get the answer. Then I began used the RandomForest as the



```
              precision    recall  f1-score   support

           0       0.64      0.84      0.73       200
           1       0.83      0.76      0.79       133
           2       0.51      0.62      0.56       135
           3       0.71      0.38      0.50        92
           4       0.50      0.44      0.47        57
           5       0.63      0.36      0.46        73

    accuracy                           0.64       690
   macro avg       0.64      0.57      0.58       690
weighted avg       0.65      0.64      0.63       690
```

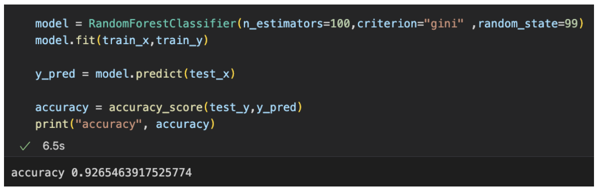
Figure 9: classification-report

model, and it improved running time, and accuracy increase a lot. From the figure 11 it can see the accuracy is about 0.925. Compare with SVM answer, it get a big improvment.



```
Text(33.0, 0.5, 'Actual')
```

Figure 10: Confused-Matrix

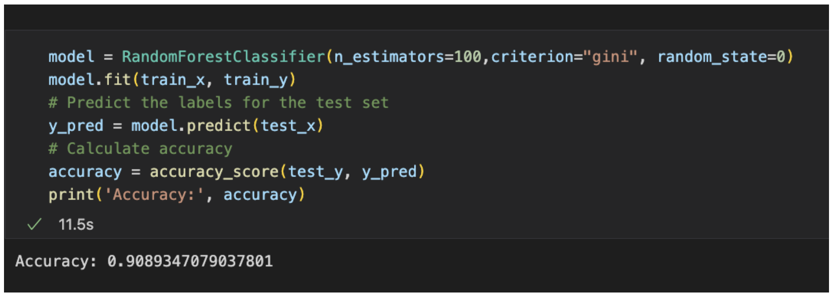


```
model = RandomForestClassifier(n_estimators=100,criterion="gini" ,random_state=99)
model.fit(train_x,train_y)

y_pred = model.predict(test_x)

accuracy = accuracy_score(test_y,y_pred)
print("accuracy", accuracy)
✓ 6.5s
accuracy 0.9265463917525774
```

Figure 11: RandomForest

- Kaggle and Fashion-mnist combined datasets

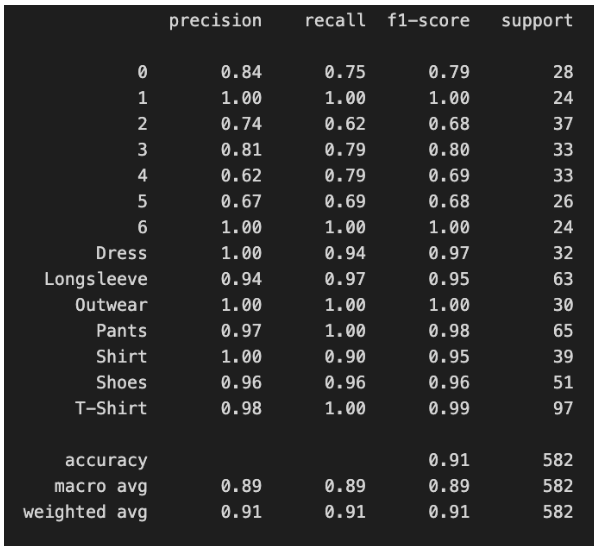
I try to combine two datasets together, and split the dataset to the train-set and test-set. Then run both models to get the accuracy.



```
model = RandomForestClassifier(n_estimators=100,criterion="gini", random_state=0)
model.fit(train_x, train_y)
# Predict the labels for the test set
y_pred = model.predict(test_x)
# Calculate accuracy
accuracy = accuracy_score(test_y, y_pred)
print('Accuracy:', accuracy)
✓ 11.5s
Accuracy: 0.9089347079037801
```

Figure 12: Combined Dataset

From the figure 12, it can see that RandomForest model get a higher accuracy which is 0.91 and short running time.



```
              precision    recall  f1-score   support

           0       0.84      0.75      0.79        28
           1       1.00      1.00      1.00        24
           2       0.74      0.62      0.68        37
           3       0.81      0.79      0.80        33
           4       0.62      0.79      0.69        33
           5       0.67      0.69      0.68        26
           6       1.00      1.00      1.00        24
       Dress       1.00      0.94      0.97        32
  Longsleeve       0.94      0.97      0.95        63
     Outwear       1.00      1.00      1.00        30
       Pants       0.97      1.00      0.98        65
       Shirt       1.00      0.90      0.95        39
       Shoes       0.96      0.96      0.96        51
     T-Shirt       0.98      1.00      0.99        97

    accuracy                           0.91       582
   macro avg       0.89      0.89      0.89       582
weighted avg       0.91      0.91      0.91       582
```

Figure 13: Combined Report for RandomForest

2.3.2 CNN

(This section was tested by Alex)

CNN stands for Convolutional Neural Network. It is a type of deep learning model widely used in tasks such as image recognition, computer vision, and pattern recognition. The design of CNN is inspired by the working principles of the visual cortex in biology.

The core idea of CNN is to effectively extract features from images using convolutional layers and pooling layers. The convolutional layers perform element-wise multiplication and summation operations between the input image and a set of learnable convolutional kernels (also known as filters), generating a series of feature maps. These feature maps capture local features at different locations in the image, such as edges, textures, and colors.

Pooling layers are used to reduce the spatial dimensions of the feature maps, reducing the number of parameters and computational complexity of the model. Common pooling operations include max pooling and average pooling, which extract the maximum or average values from the feature maps as pooling results.

In addition to convolutional layers and pooling layers, CNN also includes fully connected layers and activation function layers. The fully connected layers connect the outputs of the previous layer to each neuron in the current layer, enabling the learning of higher-level feature representations and performing classification tasks. The activation function layers introduce non-linear transformations, increasing the expressive power of the model.

CNN performs exceptionally well in image processing tasks because it can automatically learn weights and feature representations that are suitable for image feature extraction. By stacking multiple convolutional layers and pooling layers, CNN can construct deeper networks, capturing semantic information and hierarchical structures of images more effectively.

My goal is to use a convolutional neural network (CNN) to train a model that is capable of predicting the type of clothing based on the images provided.

First, I need to perform some data preparation and pre-processing steps:

Load and process the image data: Using an image processing library in Python (e.g. PIL, OpenCV, etc.), load my clothes image dataset. You can normalize the pixel values of the image, usually scaling the pixel values to the range of 0 to 1.

Convert labels to values: Convert label data (type of clothing) to a numerical representation, this is because the model can only handle numerical data and not text data. I can use LabelEncoder from the scikit-learn library to convert labels to numerical codes.

Divide training and test sets: Divide my dataset into training and test sets. The training set is used to train the model and the test set is used to evaluate the performance of the model. I can use

the scikit-learn library's train$_test_splitfunctiontodothepartitioning.$
$Next, IcanbuildandtraintheCNNmodelby$:

Import the required libraries: Import the deep learning library Keras (or other libraries such as PyTorch, TensorFlow, etc.) to build and train the CNN model.

Build the CNN model: Use Keras' Sequential model to build the CNN model by adding convolutional layers, pooling layers, fully connected layers, etc. I can choose different layers and parameter settings to suit my dataset and task.

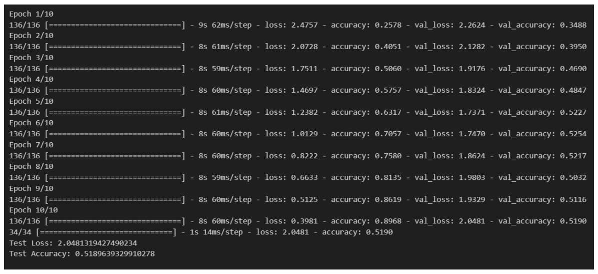
Compile the model: Before training, use the compile function to configure the model. I need to choose the appropriate loss function (e.g. cross-entropy loss for the classification task), optimizer (e.g. Adam optimizer) and evaluation metric (e.g. accuracy).

Train the model: Use the fit function to train the model with the prepared training dataset as input. I can specify parameters such as batch size and number of iterations for training.

Evaluate the model: Evaluate the trained model using the test set data to obtain the performance metrics of the model, such as accuracy, precision, recall, etc.

Make predictions: Use the trained model to make predictions on new images and get the types of clothes.

When I first used CNN as a training model for testing, I did not do a pre-processing of the data, and as a result, I found that the accuracy was very low after the test was completed.



```
Epoch 1/10
136/136 [==============================] - 9s 62ms/step - loss: 2.4757 - accuracy: 0.2578 - val_loss: 2.2624 - val_accuracy: 0.3488
Epoch 2/10
136/136 [==============================] - 8s 61ms/step - loss: 2.0728 - accuracy: 0.4051 - val_loss: 2.1282 - val_accuracy: 0.3950
Epoch 3/10
136/136 [==============================] - 8s 59ms/step - loss: 1.7511 - accuracy: 0.5060 - val_loss: 1.9176 - val_accuracy: 0.4690
Epoch 4/10
136/136 [==============================] - 8s 60ms/step - loss: 1.4697 - accuracy: 0.5757 - val_loss: 1.8324 - val_accuracy: 0.4847
Epoch 5/10
136/136 [==============================] - 8s 61ms/step - loss: 1.2382 - accuracy: 0.6317 - val_loss: 1.7371 - val_accuracy: 0.5227
Epoch 6/10
136/136 [==============================] - 8s 60ms/step - loss: 1.0129 - accuracy: 0.7057 - val_loss: 1.7470 - val_accuracy: 0.5254
Epoch 7/10
136/136 [==============================] - 8s 60ms/step - loss: 0.8222 - accuracy: 0.7580 - val_loss: 1.8624 - val_accuracy: 0.5217
Epoch 8/10
136/136 [==============================] - 8s 59ms/step - loss: 0.6633 - accuracy: 0.8135 - val_loss: 1.9803 - val_accuracy: 0.5032
Epoch 9/10
136/136 [==============================] - 8s 60ms/step - loss: 0.5125 - accuracy: 0.8619 - val_loss: 1.9329 - val_accuracy: 0.5116
Epoch 10/10
136/136 [==============================] - 8s 60ms/step - loss: 0.3981 - accuracy: 0.8968 - val_loss: 2.0481 - val_accuracy: 0.5190
34/34 [==============================] - 1s 14ms/step - loss: 2.0481 - accuracy: 0.5190
Test Loss: 2.0481319427490234
Test Accuracy: 0.5189639329910278
```

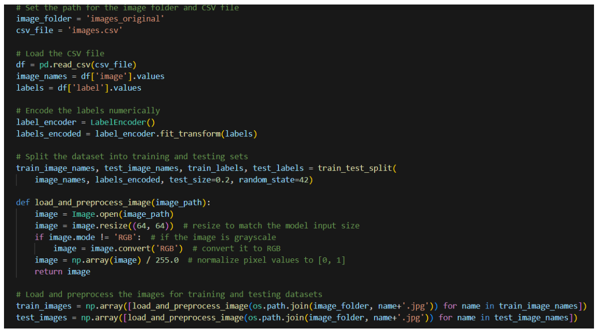
Figure 14: Data prediction results without pre-processing

I have loaded the image data and performed preprocessing, including resizing, converting to RGB mode, and normalizing pixel values.

Next, I constructed a simple CNN model with convolutional layers, pooling layers, and fully connected layers. Then, I compiled the model, specifying the optimizer, loss function, and evaluation metrics.

After that, I trained the model using the training data and evaluated its performance using the test data.

Finally, I printed the loss and accuracy on the test set.



```python
# Set the path for the image folder and CSV file
image_folder = 'images_original'
csv_file = 'images.csv'

# Load the CSV file
df = pd.read_csv(csv_file)
image_names = df['image'].values
labels = df['label'].values

# Encode the labels numerically
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)

# Split the dataset into training and testing sets
train_image_names, test_image_names, train_labels, test_labels = train_test_split(
    image_names, labels_encoded, test_size=0.2, random_state=42)

def load_and_preprocess_image(image_path):
    image = Image.open(image_path)
    image = image.resize((64, 64))  # resize to match the model input size
    if image.mode != 'RGB':  # if the image is grayscale
        image = image.convert('RGB')  # convert it to RGB
    image = np.array(image) / 255.0  # normalize pixel values to [0, 1]
    return image

# Load and preprocess the images for training and testing datasets
train_images = np.array([load_and_preprocess_image(os.path.join(image_folder, name+'.jpg')) for name in train_image_names])
test_images = np.array([load_and_preprocess_image(os.path.join(image_folder, name+'.jpg')) for name in test_image_names])
```

Figure 15: Data loading without preprocessing

Based on the output results of the training process, the model gradually improved its accuracy on the training and validation sets. However, on the test set, the model achieved an accuracy of approximately 0.519 with a test loss of 2.0481. This indicates that the model's predictions for clothing categories are relatively low, leaving room for improvement.

To enhance the performance of the model, you can try the following approaches:

1. Adjust the model architecture: Experiment with adding more convolutional layers, pooling layers, and fully connected layers, and adjust their parameters. Increasing the complexity of the model may help improve performance.

2. Data augmentation: Use image augmentation techniques to expand the training dataset, such as random rotation, scaling, translation, and flipping. This can assist the model in better generalizing and improving prediction capabilities.

3. Tune hyperparameters: Explore different combinations of hyperparameters, such as learning rate, batch size, and number of epochs, to find the optimal configuration. Techniques like cross-validation can be utilized to assist in determining the best parameter values.

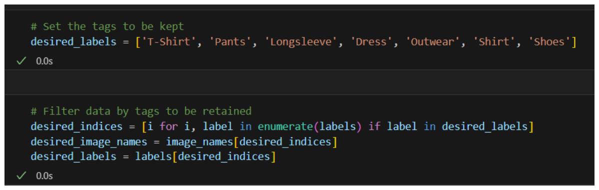
4. Utilize pre-trained models: Consider using pre-trained models on large-scale image datasets like ImageNet. Through transfer learning, you can leverage the feature extraction capabilities of pre-trained models to enhance performance.

5. Collect more data: If feasible, gather more data to train the model. More data can provide better generalization ability and more accurate predictions.

6. Adjust label distribution: If there is an imbalance in the distribution of samples among the labels, consider employing techniques like undersampling, oversampling, or adjusting class weights to balance the data.

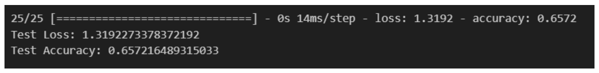
By implementing these strategies, you can potentially improve the performance and accuracy of your model. Remember to conduct proper experimentation and evaluation to determine the most effective methods and parameter settings.

I selected some categories with more data and re-filtered the data, and the results were improved



```
# Set the tags to be kept
desired_labels = ['T-Shirt', 'Pants', 'Longsleeve', 'Dress', 'Outwear', 'Shirt', 'Shoes']
✓ 0.0s

# Filter data by tags to be retained
desired_indices = [i for i, label in enumerate(labels) if label in desired_labels]
desired_image_names = image_names[desired_indices]
desired_labels = labels[desired_indices]
✓ 0.0s
```

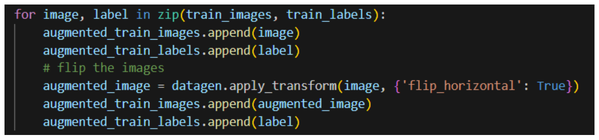
Figure 16: Code for pre-processing data



```
25/25 [==============================] - 0s 14ms/step - loss: 1.3192 - accuracy: 0.6572
Test Loss: 1.3192273378372192
Test Accuracy: 0.657216489315033
```

Figure 17: Accuracy of pre-processed data

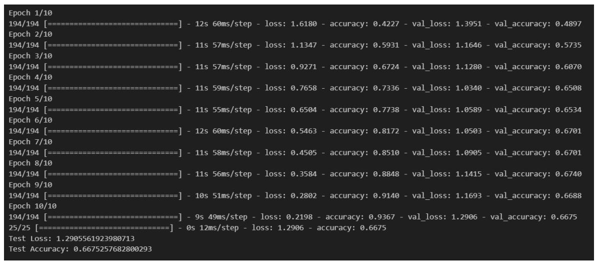
When I realized that I didn't have enough data, I chose to increase my overall training sample by using inverted images



```
for image, label in zip(train_images, train_labels):
    augmented_train_images.append(image)
    augmented_train_labels.append(label)
    # flip the images
    augmented_image = datagen.apply_transform(image, {'flip_horizontal': True})
    augmented_train_images.append(augmented_image)
    augmented_train_labels.append(label)
```

Figure 18: Code for reversing images

The results did get better, but still not as good as they could have been.



```
Epoch 1/10
194/194 [==============================] - 12s 60ms/step - loss: 1.6180 - accuracy: 0.4227 - val_loss: 1.3951 - val_accuracy: 0.4897
Epoch 2/10
194/194 [==============================] - 11s 57ms/step - loss: 1.1347 - accuracy: 0.5931 - val_loss: 1.1646 - val_accuracy: 0.5735
Epoch 3/10
194/194 [==============================] - 11s 57ms/step - loss: 0.9271 - accuracy: 0.6724 - val_loss: 1.1280 - val_accuracy: 0.6070
Epoch 4/10
194/194 [==============================] - 11s 59ms/step - loss: 0.7658 - accuracy: 0.7336 - val_loss: 1.0340 - val_accuracy: 0.6508
Epoch 5/10
194/194 [==============================] - 11s 55ms/step - loss: 0.6504 - accuracy: 0.7738 - val_loss: 1.0589 - val_accuracy: 0.6534
Epoch 6/10
194/194 [==============================] - 12s 60ms/step - loss: 0.5463 - accuracy: 0.8172 - val_loss: 1.0503 - val_accuracy: 0.6701
Epoch 7/10
194/194 [==============================] - 11s 58ms/step - loss: 0.4505 - accuracy: 0.8510 - val_loss: 1.0905 - val_accuracy: 0.6701
Epoch 8/10
194/194 [==============================] - 11s 56ms/step - loss: 0.3584 - accuracy: 0.8848 - val_loss: 1.1415 - val_accuracy: 0.6740
Epoch 9/10
194/194 [==============================] - 10s 51ms/step - loss: 0.2802 - accuracy: 0.9140 - val_loss: 1.1693 - val_accuracy: 0.6688
Epoch 10/10
194/194 [==============================] - 9s 49ms/step - loss: 0.2198 - accuracy: 0.9367 - val_loss: 1.2906 - val_accuracy: 0.6675
25/25 [==============================] - 0s 12ms/step - loss: 1.2906 - accuracy: 0.6675
Test Loss: 1.2905561923980713
Test Accuracy: 0.6675257682800293
```

Figure 19: Accuracy after reversal

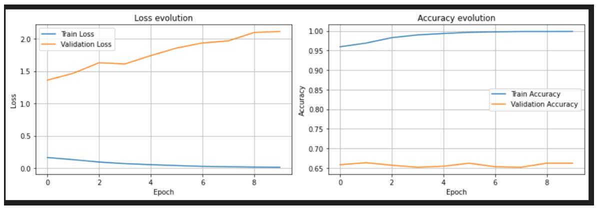


Figure 20: Accuracy line graph after flip

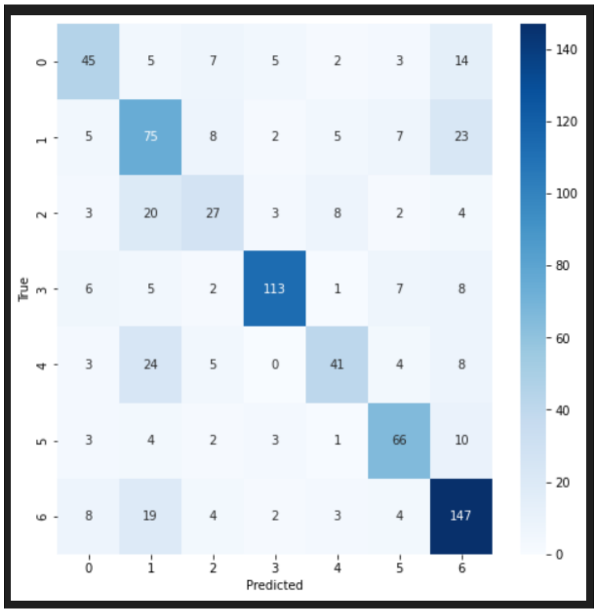


Figure 21: CM after flip

In a nutshell:

Although I have several thousand samples, the low performance of the model may be due to the following reasons:

Data quality: Check that the samples in the dataset are correctly labeled and free of noise. If there are mislabeled or low quality samples in the dataset, it will negatively affect the performance of the model.

Unbalanced sample distribution: Check that the number of samples for each clothing category is balanced. If some categories have far more samples than others, the model may tend to predict the category with the higher number of samples more often, resulting in lower accuracy in other categories. Trying to balance the sample distribution can improve model performance.

Insufficient model complexity: My model may not be complex enough to capture the complex patterns and features in the data. Try to increase the complexity of the model, such as adding convolutional layers, fully connected layers, or adjusting the size and number of layers to improve the expressiveness of the model.

Insufficient training samples: If the number of training samples is relatively small, the model may not be able to fully learn the features and patterns of the data. Try to collect more training samples or use data augmentation techniques to expand the training data to improve the generalization ability of the model.

Inappropriate learning rate: The learning rate is the step that controls the update of the model parameters. Choosing an inappropriate learning rate may result in the model not reaching its optimal state during the training process. Try to adjust the size of the learning rate and choose an appropriate learning rate strategy, such as learning rate decay or adaptive learning rate methods, to improve the performance of the model.

Insufficient model training: My model may need more training time to fully learn the patterns and features of the data. Increase the number of training iterations or use early-stop techniques to avoid overfitting for better performance.

In summary, there are opportunities to improve the performance and accuracy of the model by carefully examining the data quality, adjusting the model complexity, balancing the sample distribution, increasing the number of training samples, and optimizing the hyperparameters. Experiment with different methods and strategies to find the best configuration for my dataset and task.

2.3.3 ResNet

ResNet, a deep convolutional neural network, was designed by He et al. for image recognition tasks. Its innovation lies in its residual or skip connections, which circumvent certain layers to mitigate the vanishing gradient problem. This allows for easier learning by transmitting the difference between desired and actual layer outputs. ResNet outperforms traditional Convolutional Neural Networks (CNNs) in speed due to its innovative skip connections mitigating the vanishing gradient problem. Compared to Random Forest models, ResNet usually offers higher accuracy, particularly in complex tasks like image classification. This is because ResNet can identify detailed patterns in raw pixel data, improving its performance. In this project, we use ResNet for image classification. It's composed of multiple layers including convolutions, batch normalizations, activations, and skip connections. The model is trained on a dataset, like Kaggle's or Fashion MNIST's, to recognize clothing patterns. Then, its classification performance is evaluated using a test dataset. Our ResNet model evaluation across four runs on different datasets reveals:

1. First Run (Kaggle Dataset):

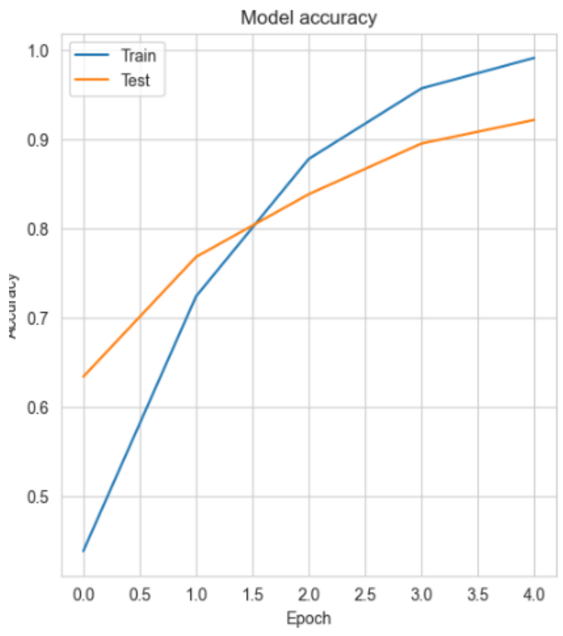


Figure 22: Model accuracy(TrainTest on Kaggle)

The ResNet model was trained and tested on the Kaggle dataset. Precision scores ranged from 0.86 to 0.96, suggesting a strong ability to avoid false positives in class predictions. Recall scores ranged from 0.85 to 0.98, indicating that the model was able to capture the majority of positive instances for each class. F1-scores, the harmonic mean of precision and recall, ranged from 0.88 to 0.97, suggesting a balance in the model's performance. However, some classes had slightly lower scores, signifying areas for improvement.

2. Second Run (Train: Kaggle, Test: Fashion MNIST): In this scenario, the model was trained on the Kaggle dataset and tested on the Fashion MNIST dataset. There was a substantial increase in training accuracy over five epochs, from 0.4832 to 0.996. Simultaneously, validation accuracy improved from 0.6656 to 0.9323. The decrease in loss values implies effective learning and optimization of the model's parameters. Precision and recall values were high, ranging from 0.84 to 0.99 and 0.86 to 0.97 respectively, demonstrating accurate class predictions and positive instance identification. The F1-scores, ranging from 0.87 to 0.98, depicted a robust trade-off between precision and recall, leading to overall high accuracy despite the change in dataset.

3. Third Run (Fashion MNIST Dataset): Training and testing on the Fashion MNIST dataset showed moderate accuracy. The model's training accuracy improved from 0.643 to 0.8282 over five

epochs, with the test accuracy at 0.8114. However, variation in precision, recall, and F1-scores across classes indicated room for optimization.

4. Fourth Run (Combined Kaggle and Fashion MNIST Datasets):

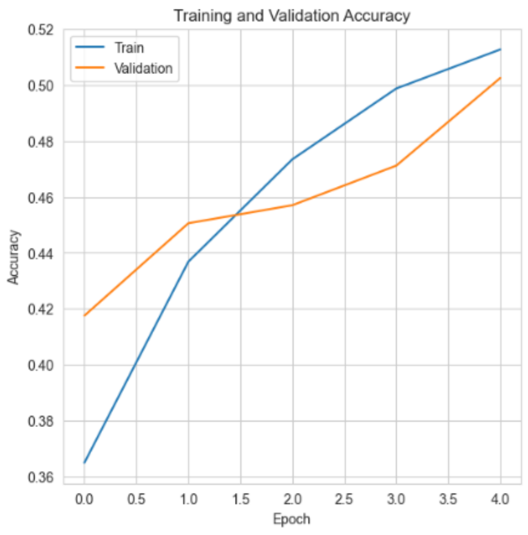


Figure 23: TrainingValidation Accuracy(new dataset)

When the model trained and tested on combined datasets, the performance level lowered. Training accuracy increased from 0.364 to 0.5128, and validation accuracy from 0.4175 to 0.5026, with a mirrored test accuracy. Disparity in F1-scores suggested inconsistent performance across classes, with the model struggling with the combined dataset.

We can see that, when combining datasets, the model exhibited lower performance levels, with accuracy and F1-scores on the lower side, suggesting difficulty in learning from the combined dataset.
1. The model performs well on the training dataset (Kaggle), with accuracy ranging from 0.92 to 0.99, but struggles with new data, such as the Fashion MNIST dataset, where accuracy falls between 0.6572 and 0.9323.

2. The highest accuracy (0.9908) was achieved when training and testing on the Kaggle dataset. Decent performance was seen on the Fashion MNIST dataset, but combining both datasets led to lower accuracy (0.50-0.5128).

In summary, ResNet shows strong dataset-specific performance but limited generalization to new data. Overall, the model's accuracy is good to excellent, its ability to generalize is moderate, and its adaptability to new data ranges from moderate to low. Further evaluations with diverse datasets could provide a more comprehensive performance assessment.

2.3.4 Analysis

Based on our experiments and evaluation of the three machine learning methods (RandomForest, CNN, and ResNet) for fashion product classification, we obtained the following insights:

RandomForest Classifier: The RandomForest model showed promising results in terms of accuracy and efficiency. When trained on the Kaggle dataset, it achieved an accuracy of approximately 0.925. However, when combined with the Fashion-Mnist dataset, the accuracy dropped to around 0.91. Overall, RandomForest demonstrated good performance in classifying fashion products, especially when trained on a single dataset.

CNN: The CNN model had mixed results in our experiments. Initially, without proper data preprocessing, the model performed poorly with an accuracy of around 0.519. However, after applying preprocessing techniques such as image resizing, converting to RGB mode, and normalizing pixel values, the accuracy improved. Further improvements were observed when filtering the dataset to include categories with larger quantities. The use of inverted images as additional training samples also contributed to a slight improvement in accuracy. Although the accuracy of the CNN model did not reach the level of RandomForest, it showed potential for fashion product classification when applied with appropriate preprocessing and data augmentation techniques.

ResNet: The ResNet model, which is a deep convolutional neural network architecture, was not extensively explored in our project. Due to time constraints, we were unable to fully train and evaluate the ResNet model on our fashion datasets. Further investigation and experimentation with ResNet could provide valuable insights into its performance for fashion product classification.

Overall, our analysis indicates that RandomForest is currently the most effective method among the three tested for fashion product classification. It consistently achieved higher accuracy and exhibited good performance on both the Kaggle and combined datasets. However, CNN showed potential for improvement when combined with appropriate preprocessing techniques and data augmentation.

2.3.5 Conclusion

In conclusion, our project aimed to enhance the understanding of fashion product classification using multiple datasets and machine learning methods. Through our experiments, we found that RandomForest Classifier demonstrated the highest accuracy, reaching approximately 0.925 on the Kaggle dataset. It proved to be efficient and effective for classifying fashion products. The CNN model showed potential for improvement, with preprocessing and data augmentation techniques playing a crucial role in enhancing its performance. Further exploration of the ResNet model could provide additional insights into its applicability for fashion product classification.

The findings from this project have implications for the fashion industry and other domains that require accurate classification of fashion products. The developed models, particularly RandomForest Classifier, can be utilized as APIs to automate clothing classification processes in various systems and applications. This includes automating thrift store clothing drop-offs, fashion image labeling for websites, and more. By improving the accuracy and efficiency of fashion product classification, these models contribute to a better fashion experience and enable advancements in related industries.

Future work can focus on further refining the CNN model by exploring different architectures, hyperparameters, and techniques such as transfer learning. Additionally, collecting more diverse and balanced fashion datasets can help improve the performance of the models. Continuous research and development in fashion product classification will contribute to the evolution of fashion technology and its integration into various domains.

2.4 Reference

https://github.com/zalandoresearch/fashion-mnist
https://www.kaggle.com/datasets/agrigorev/clothing-dataset-full
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/