ISY5002 PATTERN RECOGNITION SYSTEMS

Group Project Report

FASHION APPAREL GENERATION

NACHIKETH DORAISWAMY (A0215523N)

LAKSHMI SUBRAMANIAN (A0215255L)

YALAVARTI DHARMA TEJA (A0215457A)

Contents

Introduction

In today's competitive fashion industry, designers are looking for newer and better ways for designing apparel and launching them into the market.
Designing new garments as per the preferences of the designer can pose a big challenge as the designer may run of out of new ideas for ready to market garment designs.
Due to rapid pace of change in fashion trends, automated generation of apparel design has become popular.
Using the generative adversarial network, we train the model with large garment dataset with which we expect to receive new garment designs.

Tools/Technique Used

Tools used

- Jupyter Notebook/Colab
- Pycharm
- VS Code

Packages Used

- Tensorflow/Keras
- CV2
- Pytorch
- Scikit

Techniques Used

- Convolutional Neural Network
- VGG16
- ResNet
- Inception Resnet V2
- Stacked Ensemble model
- GAN
- DCGAN
- Color Detection using OpenCV

Convolutional Neural Networks

CNNs are widely used for image classification. Each image is passed through a series of convolutional layers with filters, MaxPooling, flattening and dense layer. The extraction of features and learning is done in the initial few layers.

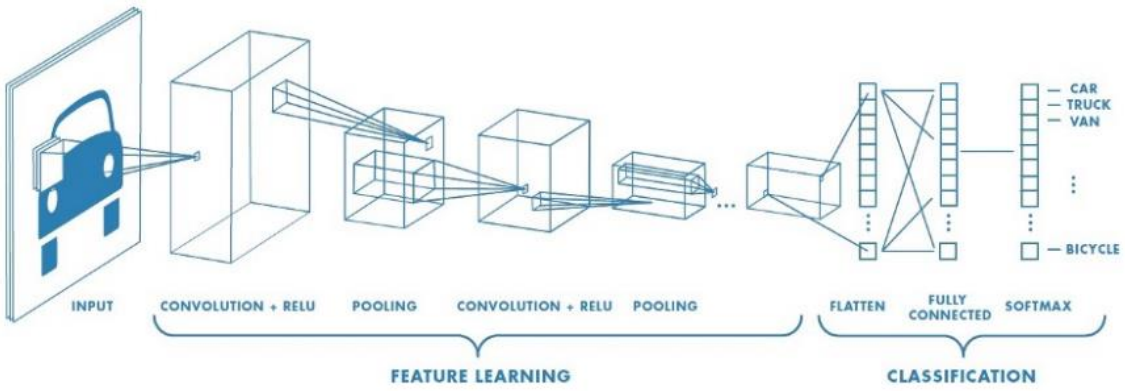

Fig - Convolutional Neural Network

Resnet

A deep neural network is difficult to train because of the vanishing gradient problem and performance gets saturated or even starts degrading. A deeper network is ideally expected to work at least as good as a shallow network if we assume many layers have not learnt features, we could simply stack identity mappings. Researchers hypothesize that letting the stacked layers fit a residual mapping is easier than letting them directly fit the desired underlaying mapping. The residual block below explicitly allows it to do precisely that by introducing skip connections.

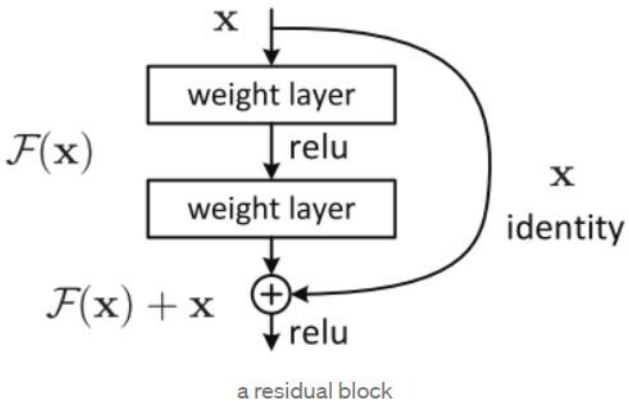

Fig – ResNet Block

Inception Resnet V2

Choosing right kernel size for convolution becomes tough, deep networks is prone to overfitting and randomly stacking convolution layers comes with a heavy cost. Hence the idea of inception net was born to utilize filters of multiple size operating on the same level, which drastically reduces the computation cost. Inception network combined with the concept of residual connections was introduced which helps in boosting the performance.

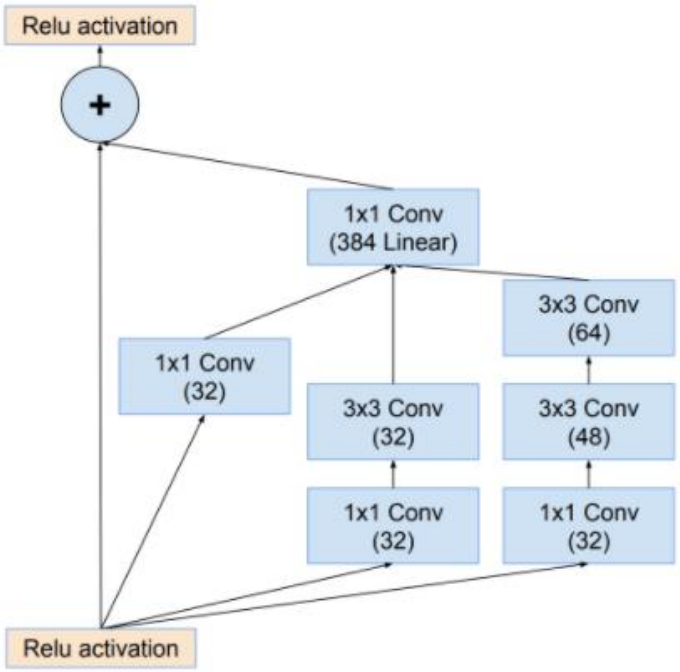

Fig – Inception ResNet - A

VGG net

The used VGG 16 is much deeper which consists of 16 weight layers including thirteen convolutional layers with filter size of 3 X 3, and fully connected layers with filter size of 3 X 3, and fully connected layers. The stride and padding of all convolutional layers are fixed to 1 pixel. All convolutional layers are divided into 5 groups and each group is followed by a max-pooling layer.

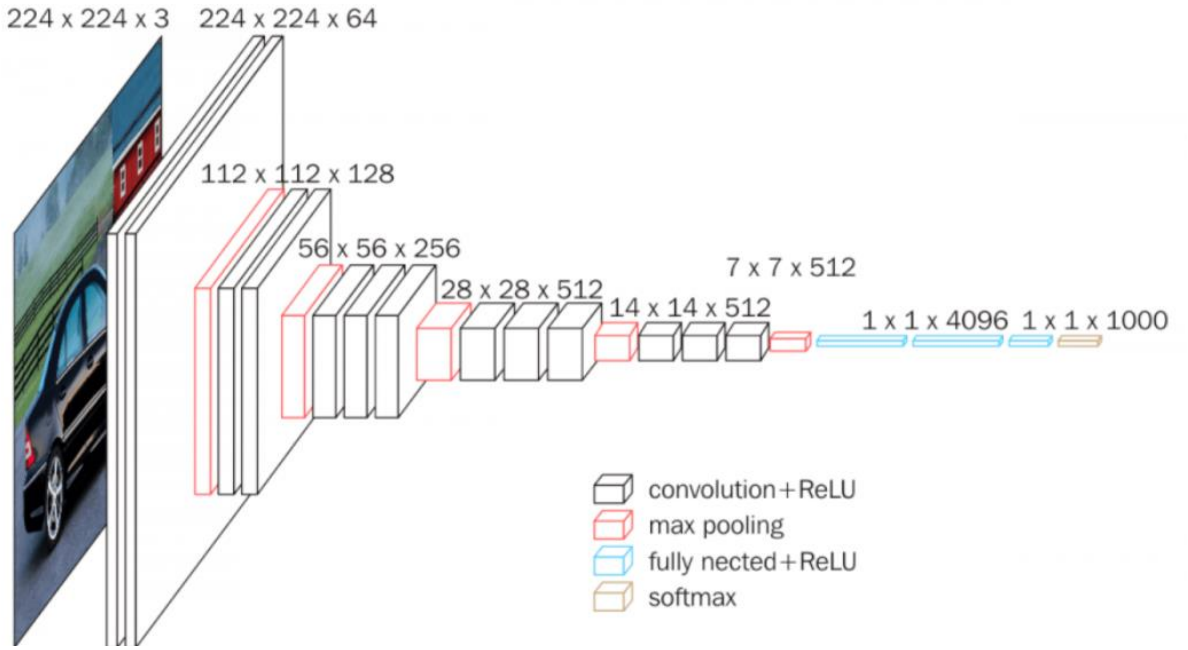

Fig – VGG net

GAN / DCGAN

GAN comprises of two neural networks, the discriminator and the generator. The purpose of the discriminator is to identify if the given image is from the original dataset or if it is newly generated image, and the generator generates new images in order to cheat the discriminator while the discriminator tries to increase the performance of guessing it right. The two components work hand in hand to give out more realistic images as output.

DCGAN is a simple variation of GAN where there is use of convolutional and transpose convolutional layers in discriminator and generator.

The difference between GAN and DCGAN is as follows –

- Use of convolutional stride instead of MaxPooling
- Use of transposed convolution for upsampling
- Eliminating fully connected layers
- Use of BatchNorm except the output layer for the generator and the input layer of the discriminator
- Use of ReLU in the generator except for the output which uses tanh
- Use LeakyReLU in the discriminator

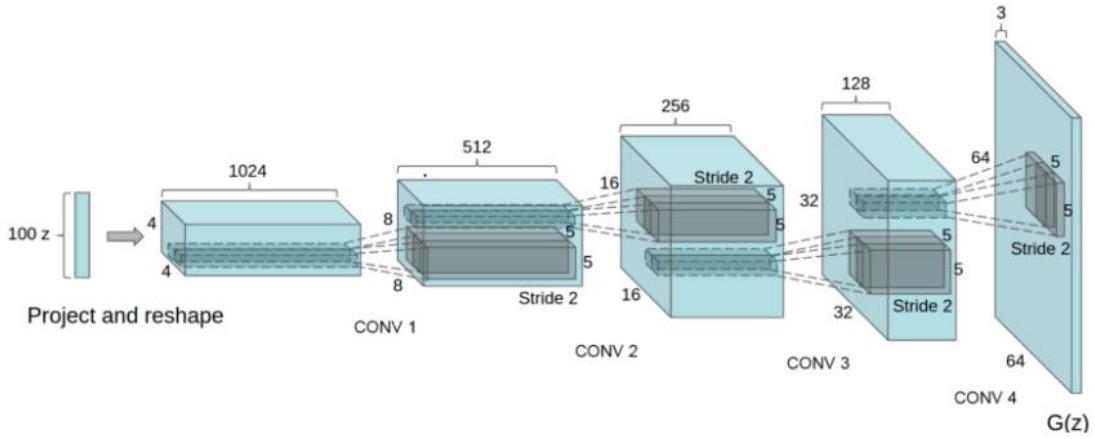

Fig – Generator block of DCGAN

System Design / Models

System Design

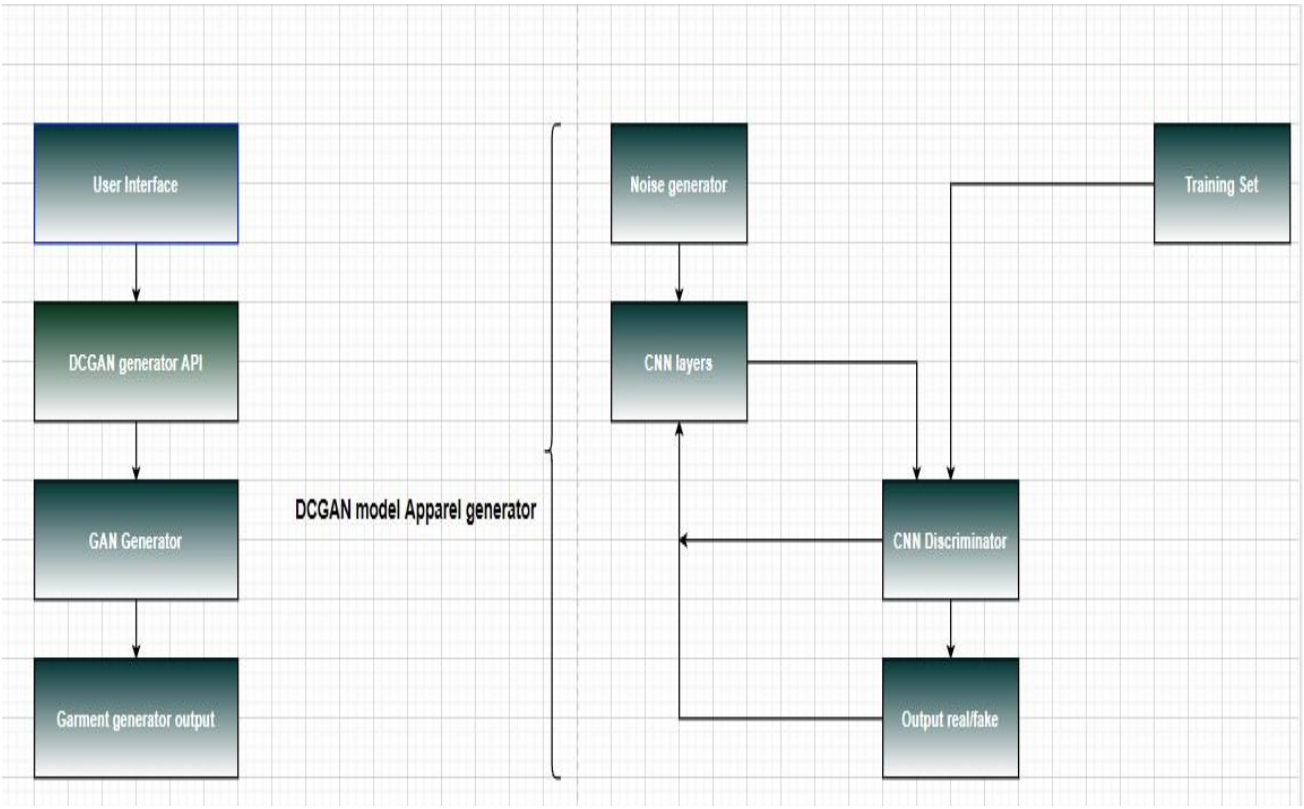

Fig – System Design for Apparel Generation

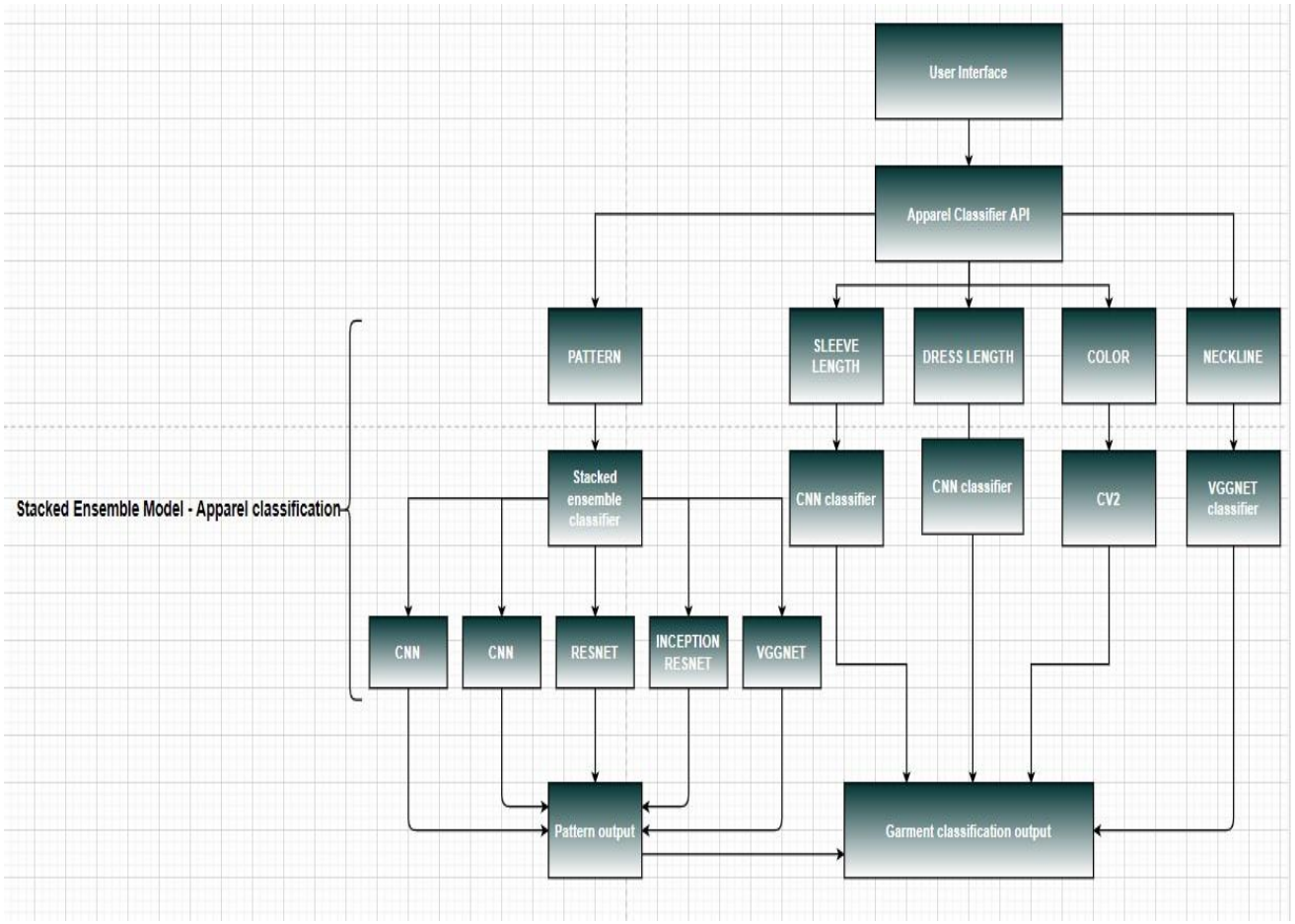

Fig – System Design for Apparel Classification

The system architecture is mainly divided into two parts:

- **DCGAN** – Apparel generation
- **Stacked Ensemble Model** – Apparel classification

We have used the **DCGAN** for apparel generation during the training phase. The API will in turn call the DCGAN model. This model uses a combination of GAN generator along with a series of convolutional layers. The CNN model is used as a discriminator that will classify the apparel as fake or real image.

In the testing and validation phase with the help of the user interface, we can call the apparel generator API. The generated garments are then sent as output to the fashion designer who can then verify the designs that have been auto generated.

The second part of the system architecture is the **stacked ensemble model**.

The fashion designer can select any apparel of his/her choice and find out the features he/she would like to know about the garment. This will help him/her analyze the data and help in making quick decisions.

The garment is fed to different models for classification. There are five models which are used as weak learners and stacked against each other to extract features.

- 2 CNN models
- Resnet
- Resnet-Inception
- VGGNET

Some of the features extracted are:

- Sleeve length
- Cloth pattern
- Dress length
- Neckline
- Color

Each model will perform a separate classification on the pre-trained model weights. The pre-trained model weights are obtained from training a separate CNN model for pattern recognition, sleeve length recognition etc.

For the cloth pattern attribute classification each weak learner model has distinct features and different accuracy and loss values which when stacked together will contribute to provide an enhanced accuracy.
The output of all the models is added together and provided as an input to the stacked ensemble model which is tasked with the classification of the garment.

The classified output from the ensemble model is provided to be used in the front-end.

Dataset

The dataset contains around 15k images with details of dress patters. The dataset was taken from the following https://s3.eu-central-1.amazonaws.com/fashion-gan/images.zip.

Our project consists of 2 modules, the first being the classification of clothes based on pattern on the dress and the second is the generation of garments.

Image Classification

In the first module we have developed classifier model to detect various attributes from a given image(256x256). The following are the few attributes we obtain from the image –

1. Pattern
2. Sleeve Length
3. Length
4. Color
5. Fit
6. Neckline

Dress Generation

We have used GAN and DCGAN to generate new dress by training on a wide range of different dress images. The training was done on colab with nearly 2k different dress images of resolution (256x256) and (64x64) respectively for both the models.

System Performance

Pattern Classification

There are 6 output classes for the design patterns of the clothes. We developed multiple CNN models as base models over which a stacked ensemble model is built for classification. The distribution of data in the class is ununiform.

Class	No. of image data
Floral	2591
Lace	1018
Polkadots	428
Print	1220
Stripes	1010
Unicolors	6098

Base Model-1

We have trained the dataset on a sequential Convolutional neural network with the following architecture and observed a validation accuracy of 78.01%.

The hyper parameters set were RMSprop optimizer with a learning rate of 0.0001, there are many dropout layers with 25% rate and 50% in one of the last layers. Two convolution layers before the dense layers were set to L2 kernel regularization of penalty 0.001.
The following is the plot for training and test set loss and accuracy over 50 epochs.

The architecture of all the models are in Appendix and results of individual base models are as follows.

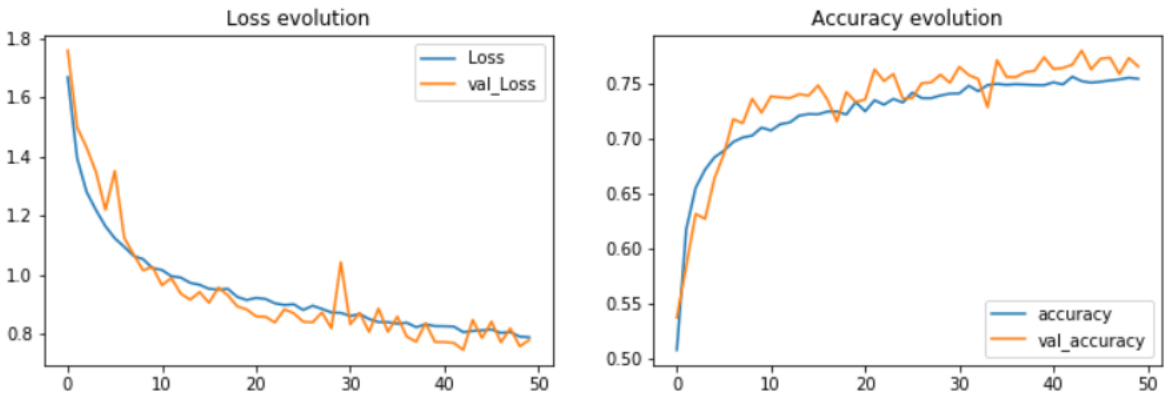

Fig – Pattern Classification Base Model 1 – loss and accuracy evolution over 50 epochs.

The following is the confusion matrix for the above trained model.

Best accuracy (on testing dataset): 78.01%

	precision	recall	f1-score	support
floral	0.7315	0.8490	0.7859	629
lace	0.2609	0.1233	0.1674	146
polkadots	0.5920	0.5968	0.5944	124
print	0.4730	0.1070	0.1746	327
stripes	0.8848	0.7967	0.8384	241
unicolors	0.8306	0.9594	0.8903	1625
micro avg	0.7801	0.7801	0.7801	3092
macro avg	0.6288	0.5720	0.5752	3092
weighted avg	0.7404	0.7801	0.7433	3092

Fig – Pattern Classification Base Model 1 – Classification Report

Base Model-2

We have trained the dataset on a VGG Net (VGG 16) and observed a validation accuracy of 79.66%.

The hyper parameters set were Adam optimizer with a learning rate of 0.00005. The following is the plot for training and test set loss :

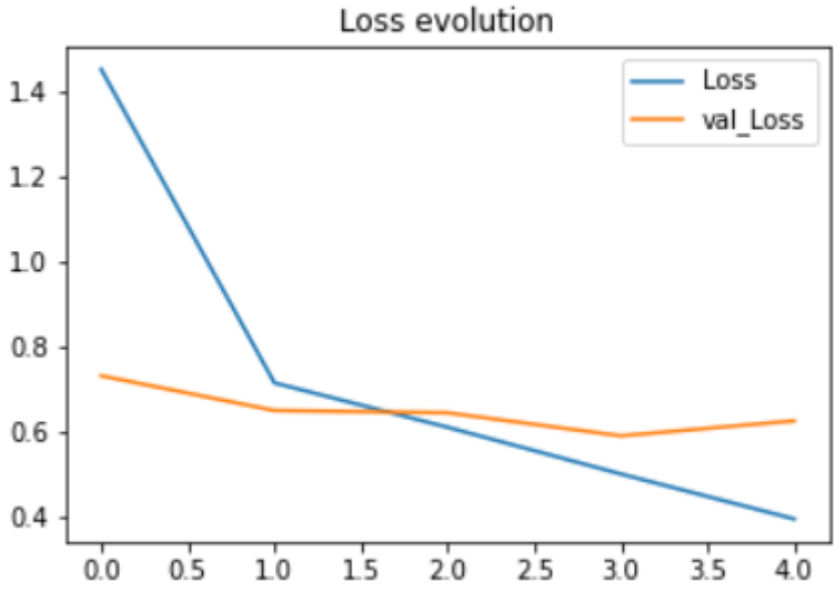

Fig – Pattern Classification Base Model 2 – loss evolution over 5 epochs.

Best accuracy (on testing dataset): 79.66%

	precision	recall	f1-score	support
floral	0.7945	0.8728	0.8318	629
lace	0.3125	0.3425	0.3268	146
polkadots	0.6972	0.6129	0.6524	124
print	0.5856	0.3242	0.4173	327
stripes	0.7586	0.8216	0.7888	241
unicolors	0.8781	0.9132	0.8953	1625
accuracy			0.7966	3092
macro avg	0.6711	0.6479	0.6521	3092
weighted avg	0.7869	0.7966	0.7870	3092

Fig – Pattern Classification Base Model 2 – Classification Report

Base Model-3

The third CNN model used in ensemble model is Residual Networks, we have used 3 residual blocks containing 3 sub-blocks of multiple residual layers with the setting of Adam optimizer and he_normal for initializing weight matrix. There are no Maxpooling and Dropout

layers, only layers of Convolution, BatchNorm, Activation and AveragePooling was used in the architecture. This model was trained on 4000 images (30 epochs) and 15K images (10 epochs) separately and showed similar results with an accuracy of around 76.58% and 75.16% respectively. The architecture (in Appendix) and results are shown below.

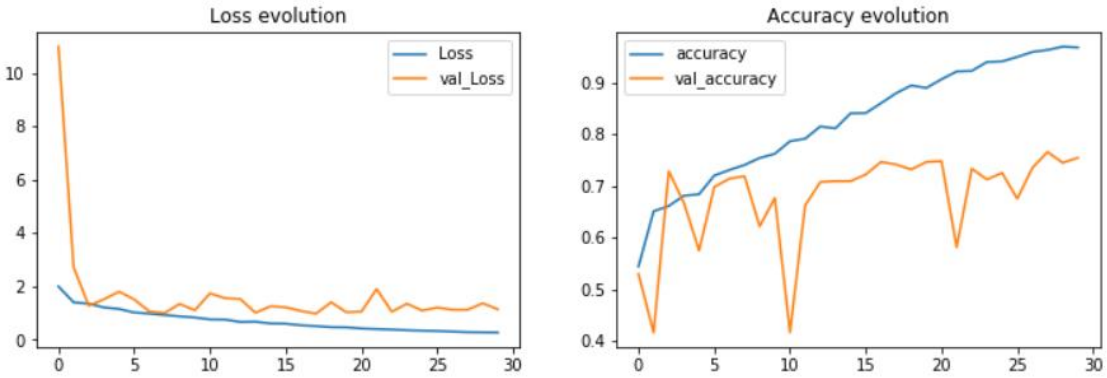

Fig – Pattern Classification Base Model 3 – loss and accuracy evolution over 30 epochs with dataset of 4000 images.

Best accuracy (on testing dataset): 76.58%

	precision	recall	f1-score	support
floral	0.6325	0.6016	0.6167	123
lace	0.0000	0.0000	0.0000	9
polkadots	0.6667	0.1111	0.1905	18
print	0.4935	0.4578	0.4750	83
stripes	1.0000	0.6087	0.7568	46
unicolors	0.8535	0.9765	0.9108	340
micro avg	0.7658	0.7658	0.7658	619
macro avg	0.6077	0.4593	0.4916	619
weighted avg	0.7543	0.7658	0.7483	619

Fig – Pattern Classification Base Model 3 Classification Report

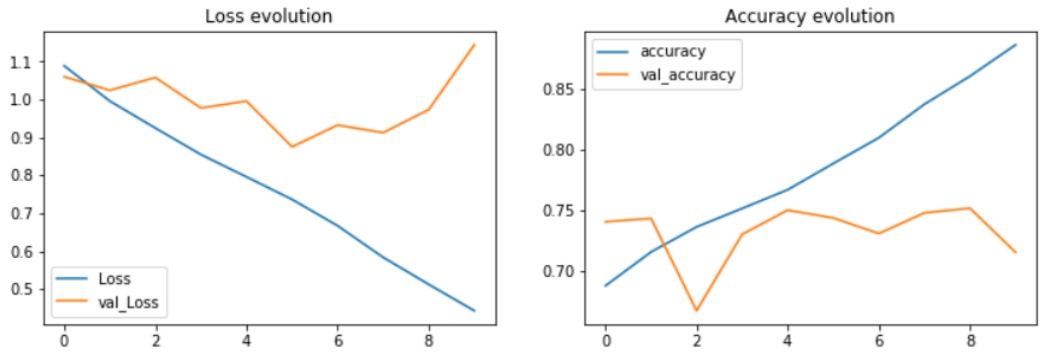

Fig – Pattern Classification Base Model 3 – loss and accuracy evolution over 30 epochs with dataset of 15k images.

Best accuracy (on testing dataset): 75.16%

	precision	recall	f1-score	support
floral	0.7266	0.7901	0.7570	629
lace	0.1984	0.1712	0.1838	146
polkadots	0.5612	0.4435	0.4955	124
print	0.3716	0.1682	0.2316	327
stripes	0.8169	0.7220	0.7665	241
unicolors	0.8327	0.9342	0.8805	1625
micro avg	0.7516	0.7516	0.7516	3092
macro avg	0.5846	0.5382	0.5525	3092
weighted avg	0.7203	0.7516	0.7295	3092

Fig – Pattern Classification Base Model 3 – Classification Report

Base Model-4

The fourth model used for ensemble is a simpler version of InceptionResNet V2, where only Inception ResNet-A was used for training due to system constraints. This model was trained on 4000 images (10 epochs) and 15K images (5 epochs) separately and showed similar results with an accuracy of around 77.87% and 74.84% respectively. The architecture and results are shown below.

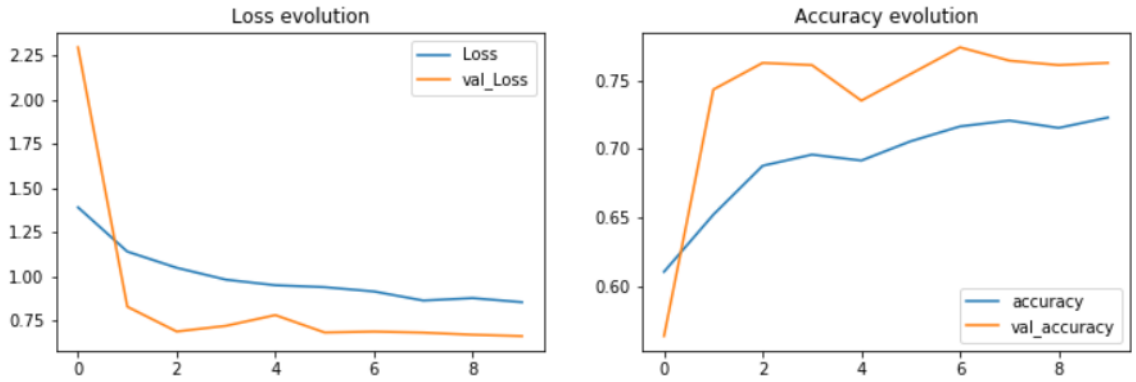

Fig – Pattern Classification Base Model 4 – loss and accuracy evolution over 10 epochs with dataset of 4k images.

Best accuracy (on testing dataset): 77.87%

	precision	recall	f1-score	support
floral	0.6444	0.7073	0.6744	123
lace	0.0000	0.0000	0.0000	9
polkadots	0.4286	0.3333	0.3750	18
print	0.4773	0.2530	0.3307	83
stripes	0.9048	0.8261	0.8636	46
unicolors	0.8594	0.9706	0.9116	340
accuracy			0.7787	619
macro avg	0.5524	0.5151	0.5259	619
weighted avg	0.7438	0.7787	0.7542	619

Fig – Pattern Classification Base Model 4 – Classification Report

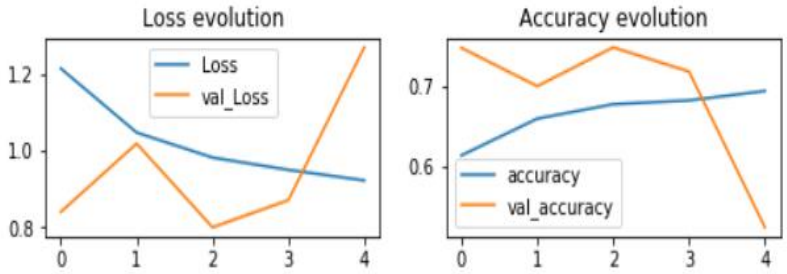

Fig – Pattern Classification Base Model 4 – loss and accuracy evolution over 5 epochs with dataset of 15k images.

Best accuracy (on testing dataset): 74.84%

	precision	recall	f1-score	support
floral	0.6903	0.7901	0.7368	629
lace	0.0000	0.0000	0.0000	146
polkadots	0.5043	0.4677	0.4854	124
print	0.3158	0.1651	0.2169	327
stripes	0.9416	0.5353	0.6825	241
unicolors	0.8086	0.9698	0.8819	1625
micro avg	0.7484	0.7484	0.7484	3092
macro avg	0.5434	0.4880	0.5006	3092
weighted avg	0.6924	0.7484	0.7090	3092

Fig – Pattern Classification Base Model 4 – Classification Report

Base Model-5

A sequential convolutional neural network was trained, and we observed an accuracy of 79.81%.

The hyper parameters that were set with many dropout layers with 25% rate in all the layers. Two convolution layers before the dense layers were set to L2 kernel regularization of penalty 0.001. In this base model we also used an Adam optimizer with a learning rate of 0.0001 which gave a better accuracy than the base model 1.

Ensemble model

The above models were concatenated together in a stacked ensemble model method and trained, which gave the following accuracy.

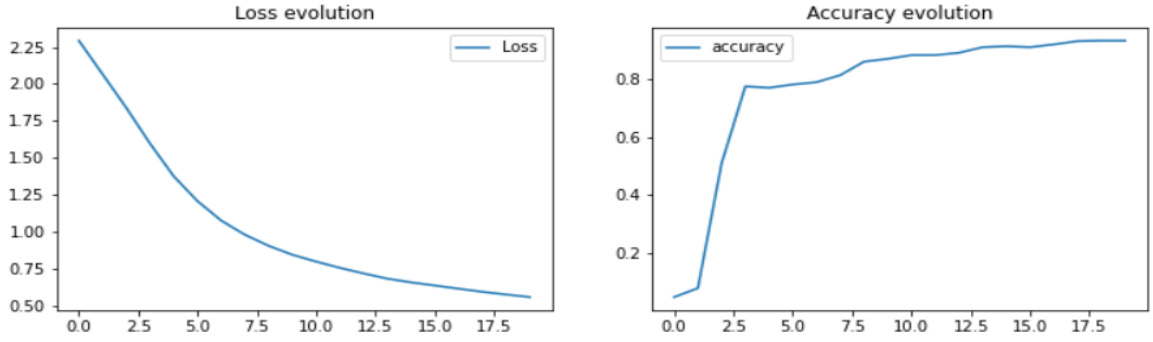

Fig – Pattern Classification Ensemble Model – loss and accuracy evolution over 20 epochs

```
Best accuracy (on testing dataset): 93.70%
              precision    recall  f1-score   support

       floral     0.9344    0.9268    0.9306       123
         lace     0.0000    0.0000    0.0000         9
    polkadots     1.0000    0.7222    0.8387        18
        print     0.9079    0.8313    0.8679        83
      stripes     1.0000    0.9565    0.9778        46
    unicolors     0.9341    1.0000    0.9659       340

     accuracy                         0.9370       619
    macro avg     0.7961    0.7395    0.7635       619
 weighted avg     0.9239    0.9370    0.9289       619
```

Fig – Pattern Classification Ensemble model – Classification Report

```
[[114   0   0   3   0   6]
 [  0   0   0   4   0   5]
 [  0   0  13   0   0   5]
 [  8   0   0  69   0   6]
 [  0   0   0   0  44   2]
 [  0   0   0   0   0 340]]
```

Fig – Ensemble Model – Confusion Matrix

Sleeve length Classification

There are 4 output classes for the sleeve length of the clothes dataset. We have developed CNN models for classification of same. The data used for training and test has the following distribution.

Class	No. of image data
Half	1500
Long	1500
Short	1500
Sleeveless	1500

There are 4 classes, the dataset used for training the CNN is around 1500 images per class.

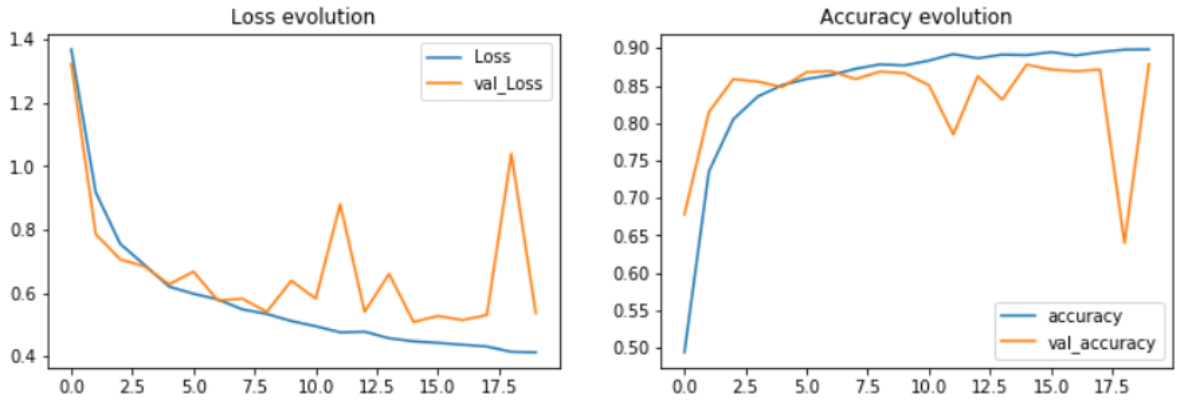

Fig – Sleeve length Classification CNN Model – loss and accuracy evolution over 20 epochs

Best accuracy (on testing dataset): 87.87%

	precision	recall	f1-score	support
half	0.8047	0.8536	0.8284	362
long	0.9405	0.8518	0.8939	371
short	0.8372	0.8989	0.8669	366
sleeveless	0.9406	0.9077	0.9239	401
accuracy			0.8787	1500
macro avg	0.8807	0.8780	0.8783	1500
weighted avg	0.8825	0.8787	0.8795	1500

Fig – Sleeve length Classification CNN model – Classification Report

Dress Length Classification

There are 5 classes used for the classification of dress length in the dataset. We preprocessed the data to have a balanced dataset. Each class must have equal data. For classification, we added a CNN model. The distribution of data for each class is as follows.

Class	No. of image data
3-4	1200
knee	1200
long	1200
normal	1200
short	1200

We have also plotted the training accuracy and loss.

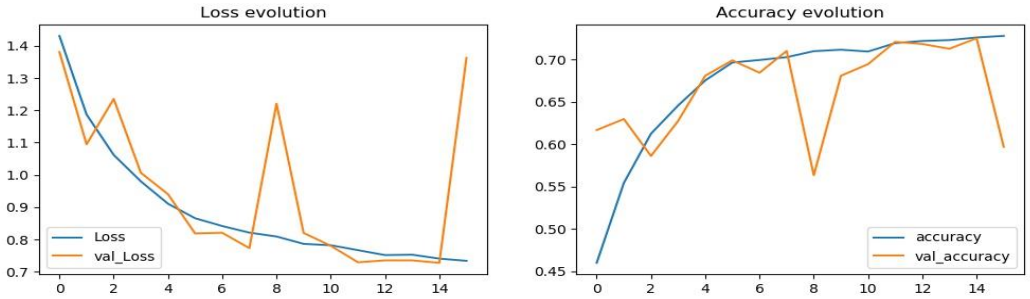

Fig – Sleeve length Classification CNN Model – loss and accuracy evolution over 20 epochs

Neckline Classification

There are 6 output classes for the neckline of the clothes dataset that are named as neckline-round, neckline-v, neckline-deep, neckline-wide, neckline-lined and neckline-back. We have incorporated VGG Net (VGG 16) model for classification of the same. Trained the model on 9319 images belonging to the above mentioned 6 classes.

Accuracy achieved on test set is: 78.8%
The accuracy and loss plots are as shown below:

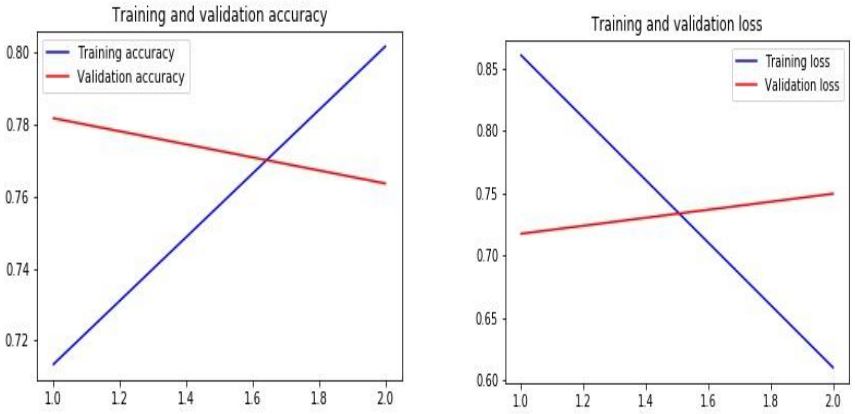

Fig – Neckline Classification VGG Net Model – loss and accuracy evolution over 2 epochs

Color Prediction

Using CV2 we have extracted the hex-code of predominant color in the image using k-means color cluster with number of clusters as 2 and the most predominant color is taken as the color of the dress.

Extraction of color is obtained based on the k means clustering of the pixel intensities of the given RGB Image, the number of clusters is set as 2 and the most frequent is taken the color of the dress. The nearest color name is also calculated using Euclidean distance from the smaller list of hex-codes and name.

```
most frequent is [ 53.88091796 104.12197326 102.49783159] (#356866)
356866 Avocado
Peaked color name: Avocado
```

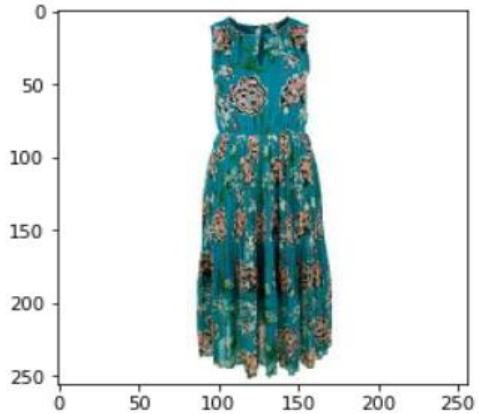

Fig – Color detection

Dress Generation

GAN

Generation of new dress was achieved by training the Generative Adversarial Network where the generator has multiple dense layers with tanh activation function in the final dense layer and discriminator also has multiple dense layers with sigmoid activation function in the final dense layer. The training was done for 200 epochs and received the following output:

Fig – GAN 256x256 Images generated

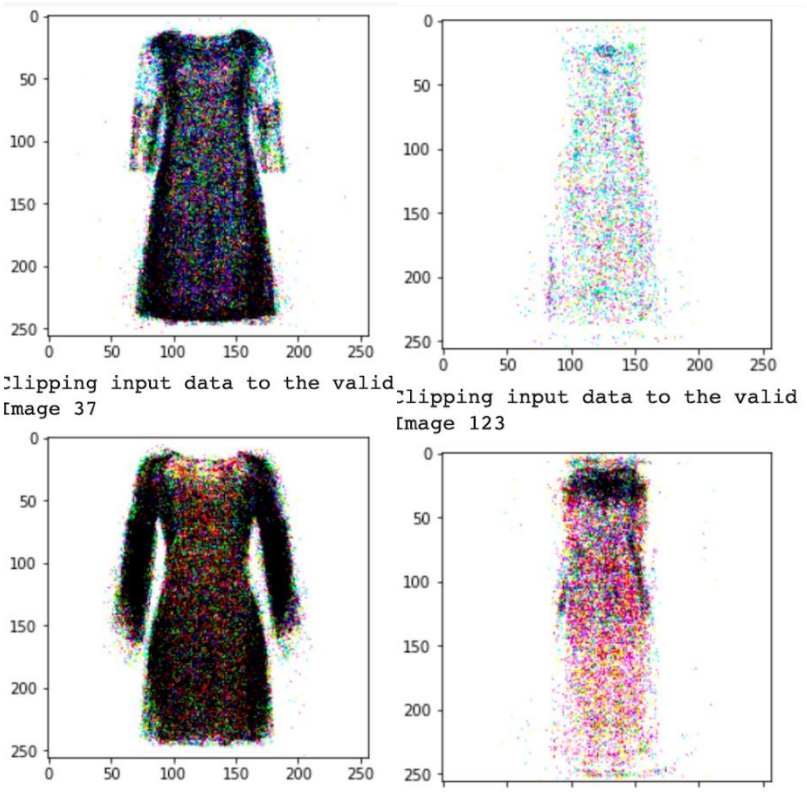

Fig – GAN 256x256 Images generated

DCGAN

Generation of new dress was achieved by training the deep convolutional Generative Adversarial Network where there are 5 convolution layers in discriminator and 5 transpose convolution layers in the generator block. The training was done for 100 epochs and the generator and discriminator loss are as follows.

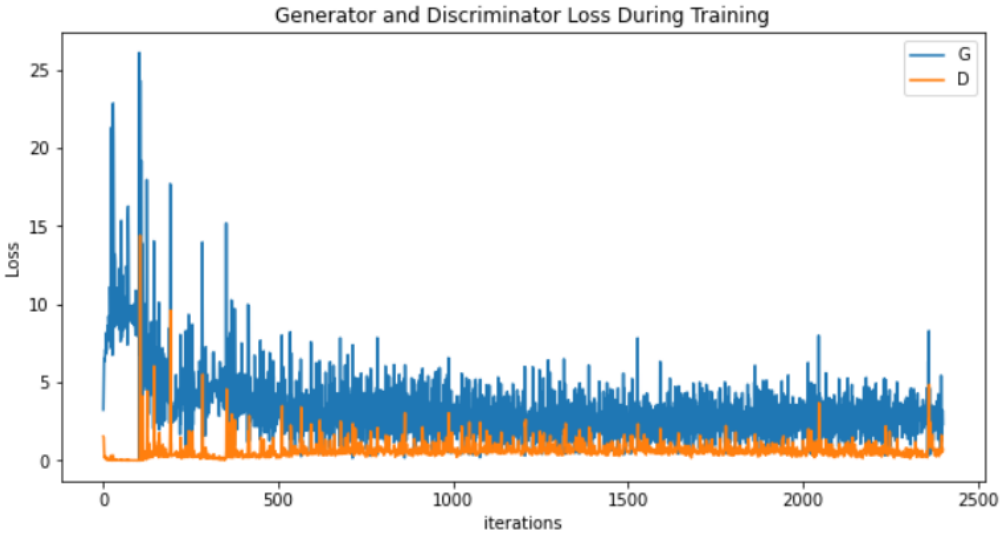

Fig – DCGAN Generator and Discriminator Loss over 100 epochs, the graphs show for every batch over 100 epochs

Fig – DCGAN 64x64 Images generated

Findings and Discussions

The shape and dynamics of a learning curve can be used to diagnose the behaviour of a machine learning model. The training and validation accuracy and loss indicate fitting of the model in the training and testing phase. Close curves of the training and accuracy loss show that the training was fairing good like in the case of CNN model used in pattern prediction. But for models like Resnet and Inception Resnet the curve seems to diverge after few epochs, indicating poorly fit model, where the model training should be stopped after reaching point of inflation. Also models like Resnet require lesser training because ResNet architecture is for reducing the complexity and solving the degradation while keeping good performance. By reducing complexity, a smaller number of parameters need to be trained and spending less time on training as well.

Models used for Pattern Classification	Model Accuracy
CNN-1	78.01%
CNN-2	79.81%
ResNet	76.58%
Inception Resnet	77.87%
VGG Net	79.66%

As it can be seen in the table above, we were able to achieve highest accuracy for CNN-2 model. The hyperparameters were optimized to accommodate the changes. Instead of using the RMSprop optimizer which was used in CNN-1 model we used the Adam optimizer as it is computationally efficient even with large datasets and works well on problems with normal or sparse gradients where the dataset could be sparse. We also maintained a learning rate of 0.0001 % which helped us in increasing the accuracy even further. With the help of the learning rate the gradients did not explode.

We also maintained a constant dropout rate of 25% in all the

convolutional layers which as compared to the dropout rate of 50% maintained in the last layers of CNN-1 model. This helped us in removing only those weights which did not actually contribute to increasing the overall accuracy of the model.

We had additional convolutional layers which included having a batch normalization layer, max pooling layer and a dropout layer which refined our accuracy by a factor of 0.12%.

We also used an L1 regularization kernel instead of the L2 regularization kernel used in the CNN-1 model. This also helped us as L1 is computationally

The overall accuracy of the pattern classification model has improved with the use of stacked ensemble model. We used a concatenate layer and two dense layers as a secondary meta learner giving weighted contribution of each sub models. The Precision and recall values for all the model for the class Lace seems to be low and therefore it is observed that the models are unable to decipher these patterns to the best compared to the other classes. This inability to identify this class has also taken a toll on the ensemble classification this is probably because of merge amount of data available in the dataset for these classes especially for lace.

Data augmentation and pre-processing of images was not required for classification of Pattern, Sleeve-Length, Length and Neckline as we have considered on the front view of the dress images and without and human model posing with the apparel. Cropping the input image for finding the colour of the dress was adopted as the background of the dress was also considered as one of the major colour contributors in the dress, therefore the centre of the image was considered for obtaining the colour. Here clustering was performed with cluster size as 2, although any number of clusters can be implemented, to get the major colour in the dress the most frequent is considered. To get the names of the colour from hex-code, we have taken a list of colour names and the corresponding hex-code and performed Euclidean

distance between the two hex-codes. The accuracy of getting the names correct is not high because the code, colour name considered is very less compared to the number of hex-codes present in each channel (16*16).

The results for GAN trained on 256x256 are found to be a little spotty compared to that of the results produced by DCGAN. The use of convolutional layers and transpose convolutional layers instead of fully connected layers has helped in improving the image clarity. Use of convnet tries to find the areas of correlation within the images looking for spatial correlations.

Transfer Learning (TL) is a concept related to machine learning (ML) which focuses on saving or storing the knowledge gained while solving for one problem and then applying it to a new but related problem. For example, the knowledge that is gained while learning to identify or recognize cars can be applied when trying to recognize buses and trucks. Using transfer learning, instead of starting the learning or training process from scratch, we can start from patterns that have already been learned when solving a different problem. In this way we can leverage earlier learnings and avoid beginning from scratch again. From the practical perspective, reusing information from previously learned tasks in order to the learn new tasks has the potential to highly improve the efficiency of a reinforcement learning agent.

Appendix

CNN_model_1.png

Fig – Base Model 1 – CNN Pattern Prediction Architecture

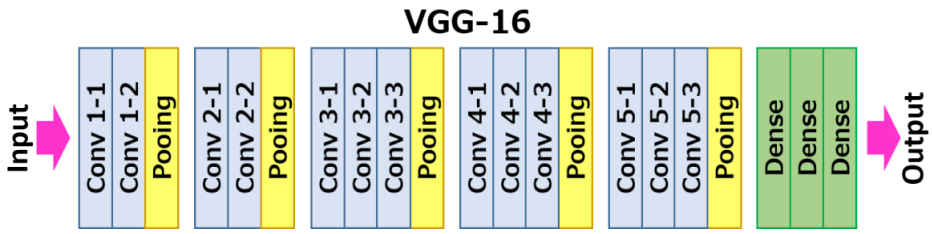

Fig – Base Model 2 – VGG Pattern Prediction Architecture

Resnet_model.png

Fig – Base Model 3 – Pattern Prediction Architecture Resnet

inception_resnet_v2.png

Fig – Base Model 4 – Pattern Prediction Architecture Inception Resnet V2

Ensemble Model.png

Fig – 5 Pattern Prediction Stacked Ensemble Model

CNN_model_sleevelength.png

Fig – 6 – CNN Architecture for Sleeve length Prediction

**Note – Images are too large to be displayed. We have embedded the images in the icons. Please double click on the icons to view the image properly.

```
D(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Fig – 8 – DCGAN Architecture for Discriminator

```
G(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Fig – 9 – DCGAN Architecture for Generator

Bibliography

https://medium.com/the-owl/building-inception-resnet-v2-in-keras-from-scratch-a3546c4d93f0