Neckline Classifier

Esma Zeynep Sayılı - Esra Cesur

Problem

- In online shopping, large fashion datasets or e-commerce platforms manually tagging and classifiying each clothing can be **a time-consuming task.** Since consumers will have preferences in the neckline styles.

- For marketers and designers, **analyzing the fashion trends** and consumer preferences is an important detail. Also, manufacterers will need **the market demands to produce clothing.**

Solution

- We will build a model using a **labeled dataset** to classify clothing items based on their neckline styles. In the process of creating this model, we will employ various architectures and follow a comparative approach to reach a conclusion. By the end of the project, **we will have categorized the photos as round, V-neck, and scoop neck styles.**

Dataset

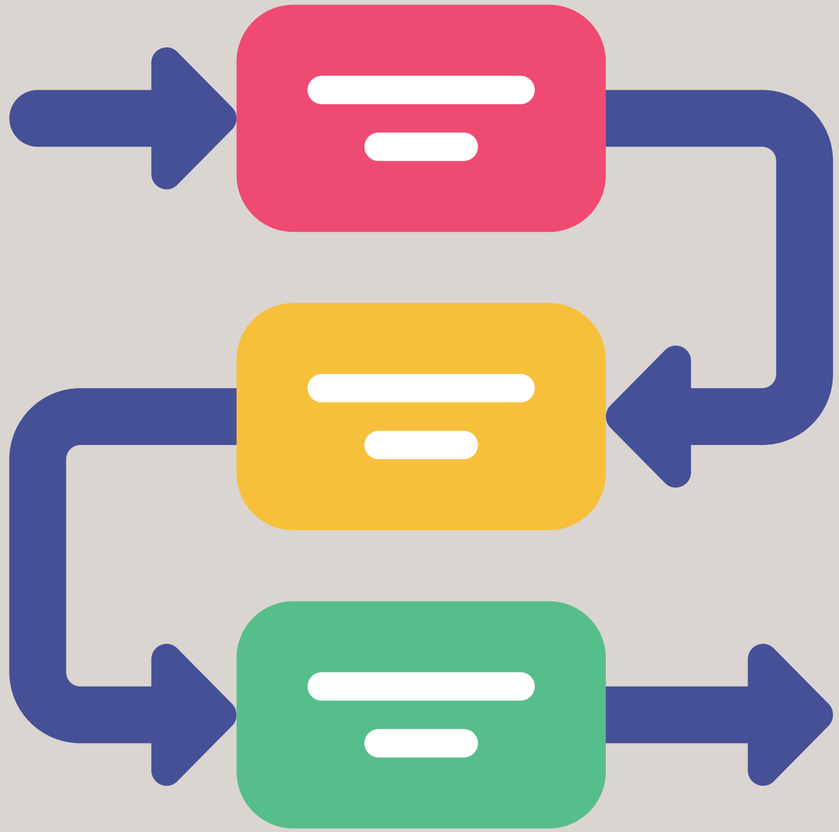
The dataset has been obtained from the Kaggle website. The data contains three types of necklines in clothing.

The dataset contains **flatlay images (images without human models) of upper/full body clothes.** We are required to classify if the cloth's region around the neck falls under one of these three categories i.e. **round, scoop or v-neck.**

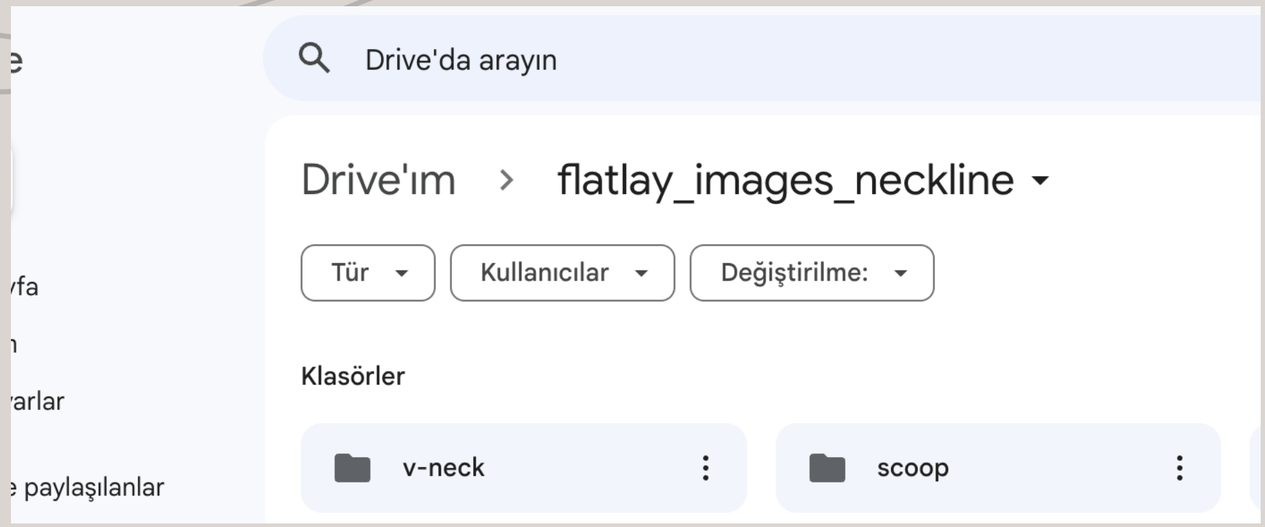
The following image gives **an understanding of the three different types of necklines mentioned above.**

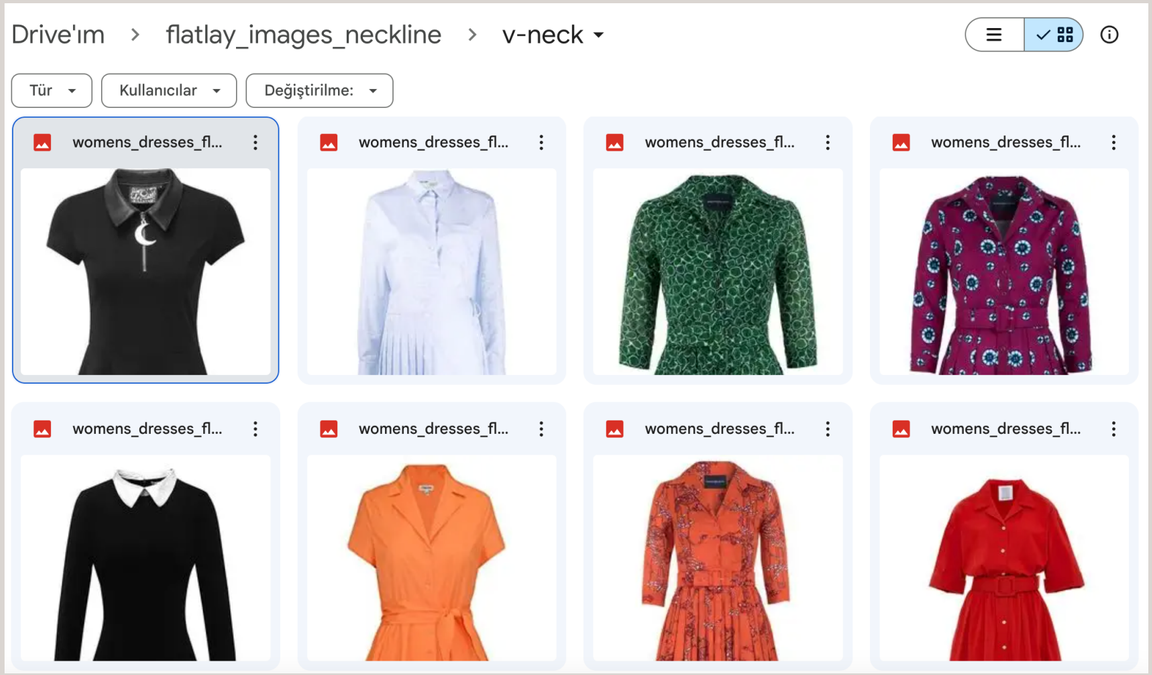
Methodology

1. **Data Collection**
2. **Deciding libraries we will use**
3. **Reviewing Architectures**
4. **Model Setup**



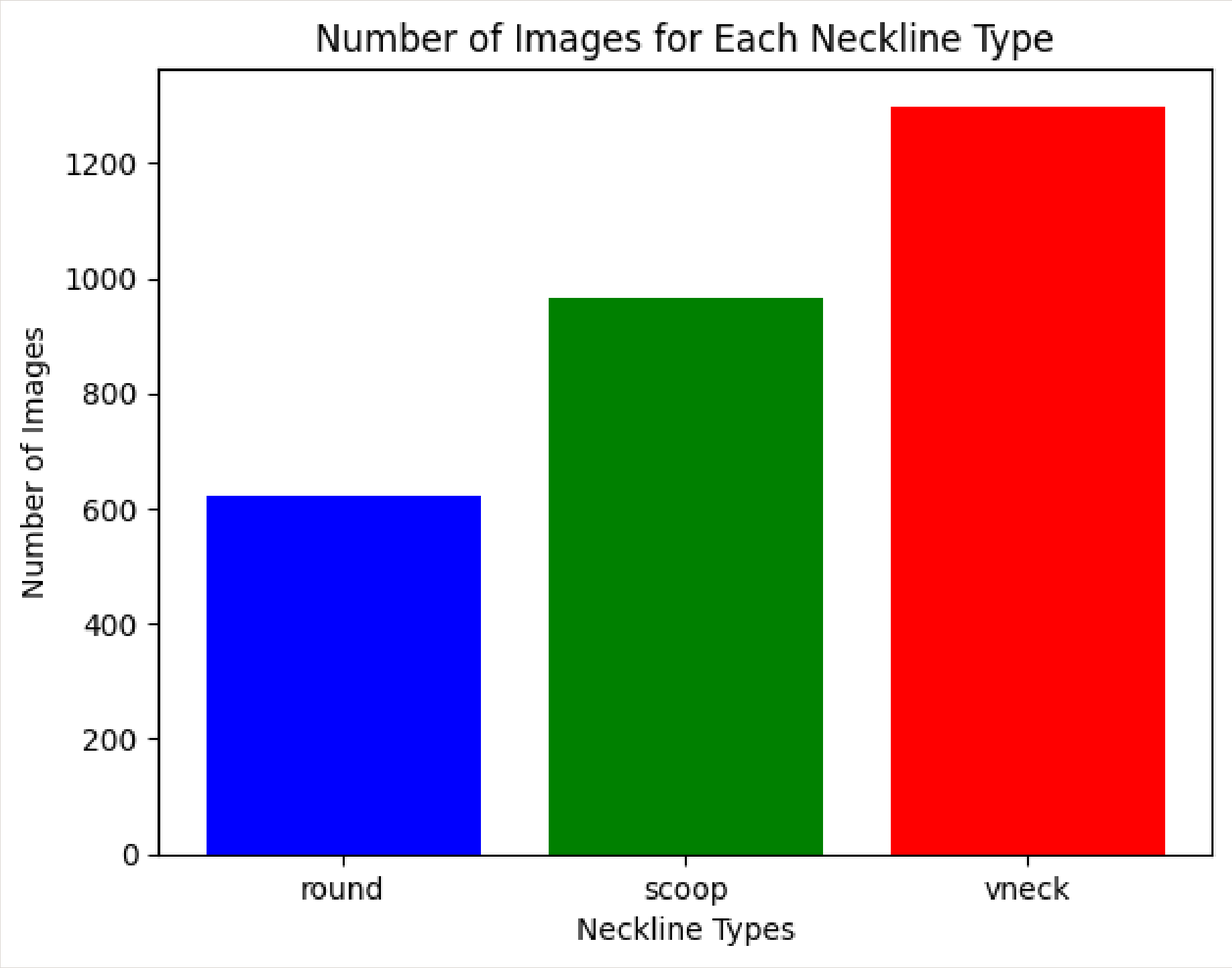
Data Collection






```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

Visualization

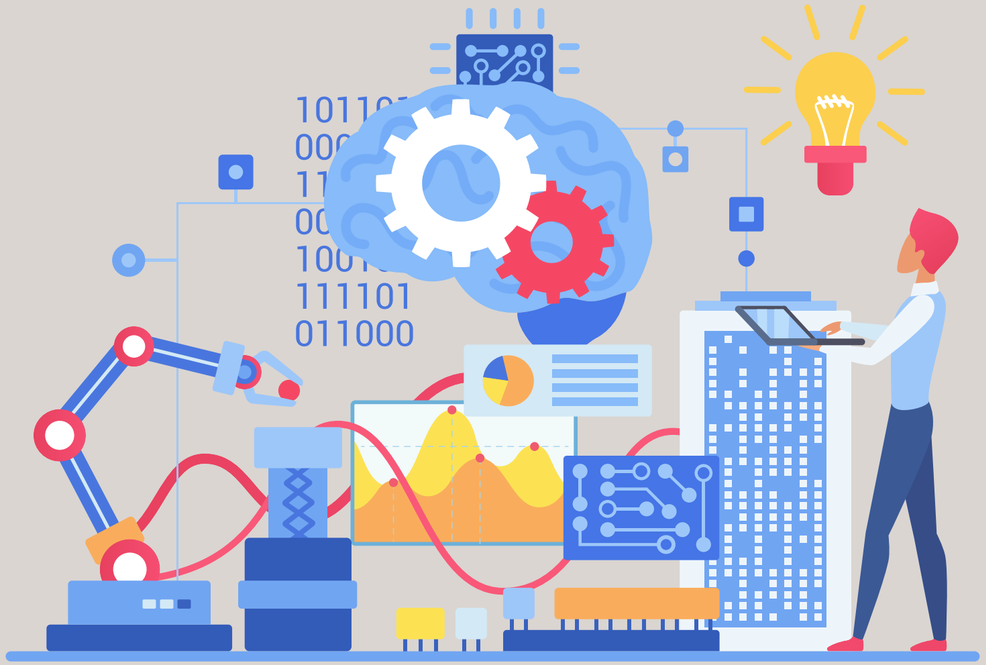


Importing Libraries

```
[ ]  import numpy as np
     import pandas as pd
     import cv2
     from google.colab.patches import cv2_imshow
     from skimage import io
     from PIL import Image
     import matplotlib.pylab as plt
     import os
     from os.path import isfile, join
     from os import listdir
```

Reviewing Architectures

1. **VGG16**
2. **ResNet**
3. **Mobilenet**



1. VGG16

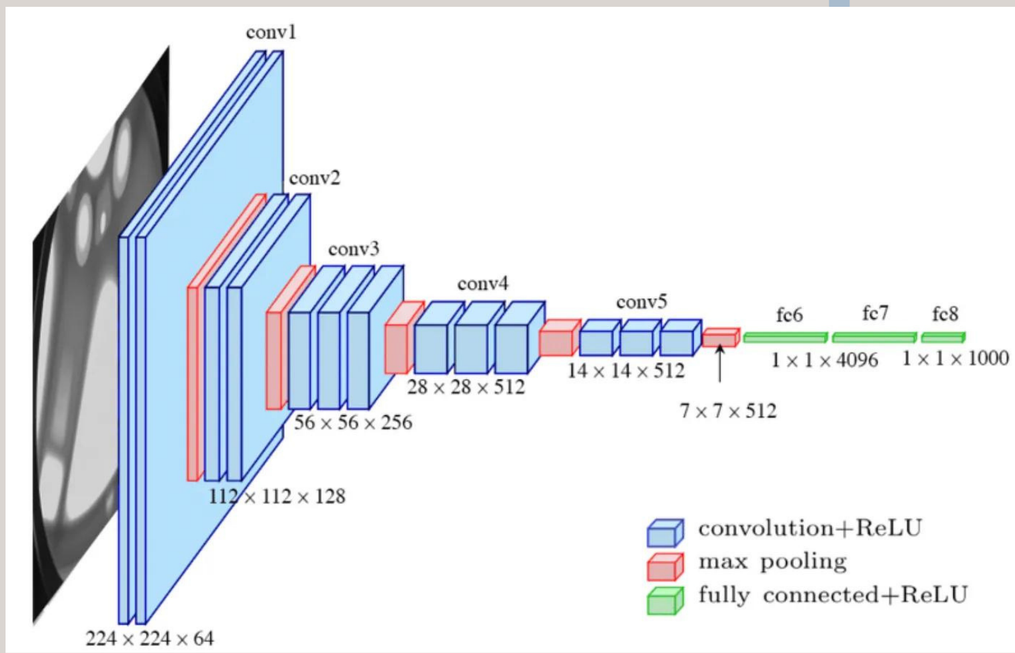
- VGG is a deep convolutional neural network that was proposed by Karen Simonyan and Andrew Zisserman.

- The VGG model investigates the depth of layers with a very small convolutional filter size (3 × 3) to deal with large-scale images.

1. VGG16

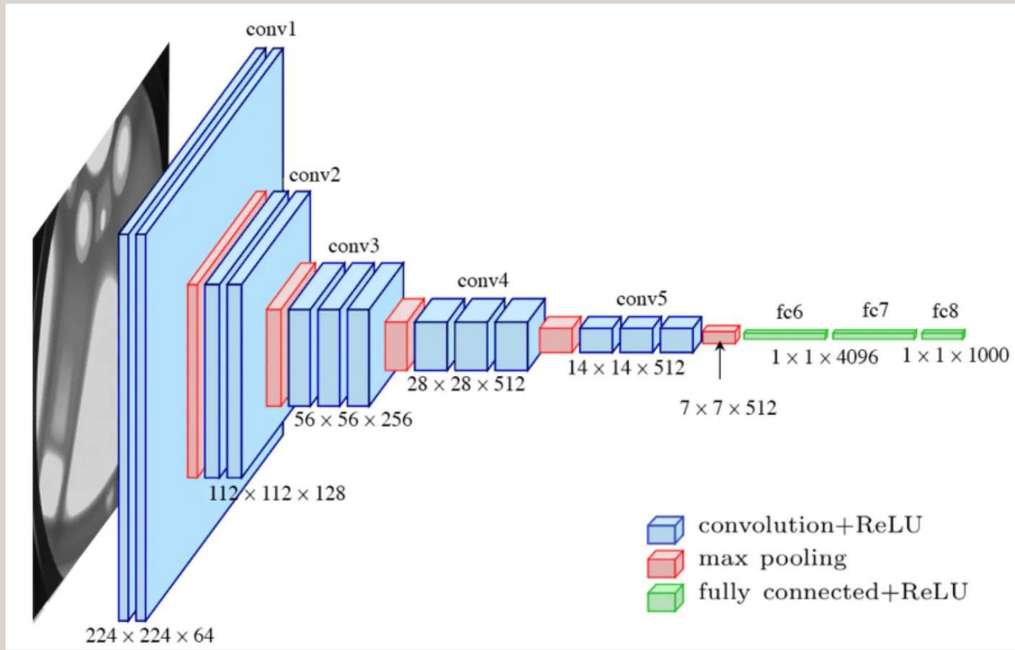
VGG16 is composed of

- **13 convolutional layers**,
- **5 max-pooling layers**,
- **3 fully connected layers.**
- The number of layers having **tunable parameters is 16** (13 convolutional layers and 3 fully connected layers).



1.VGG16

- The number of filters **in the first block is 64**, then this number **is doubled in the later blocks until it reaches 512.**
- This model is finished by two fully connected hidden layers and one output layer. The two fully connected layers have **the same neuron numbers which are 4096.**
- **The output layer consists of 1000 neurons** corresponding to the number of categories of the Imagenet dataset.



```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 3)                 75267

=================================================================
Total params: 14789955 (56.42 MB)
Trainable params: 75267 (294.01 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Model Setup

- VGG16

VGG16 Model

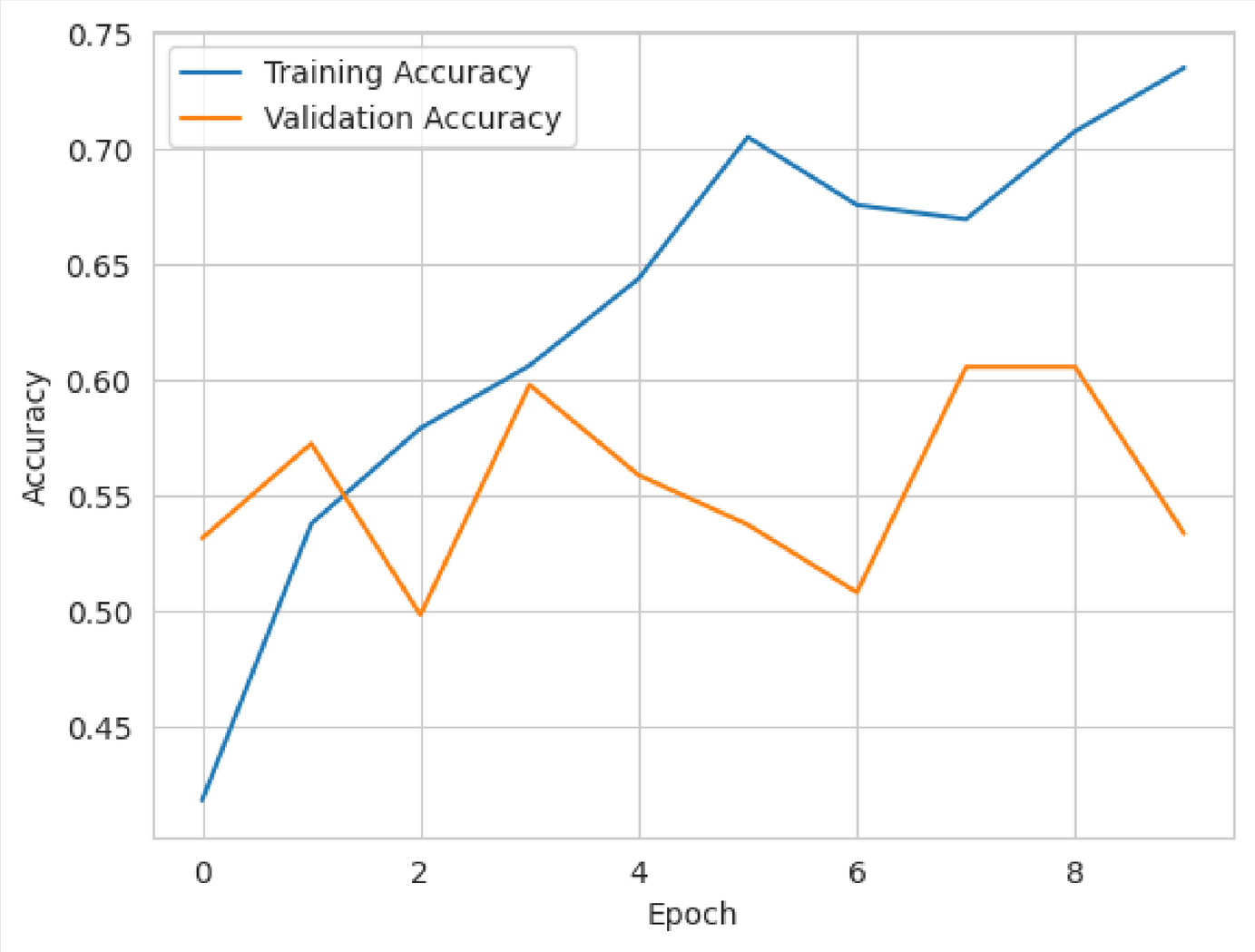
```
[ ]  from keras.models import Model
     from keras.layers import Flatten, Dense
     from keras.applications import VGG16


     IMAGE_SIZE = [224, 224]


     vgg = VGG16(input_shape = IMAGE_SIZE + [3], weights = 'imagenet', include_top = False)


     for layer in vgg.layers:
         layer.trainable = False

     x = Flatten()(vgg.output)

     x = Dense(3, activation = 'softmax')(x)

     model = Model(inputs = vgg.input, outputs = x)

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

Model Setup

- VGG16

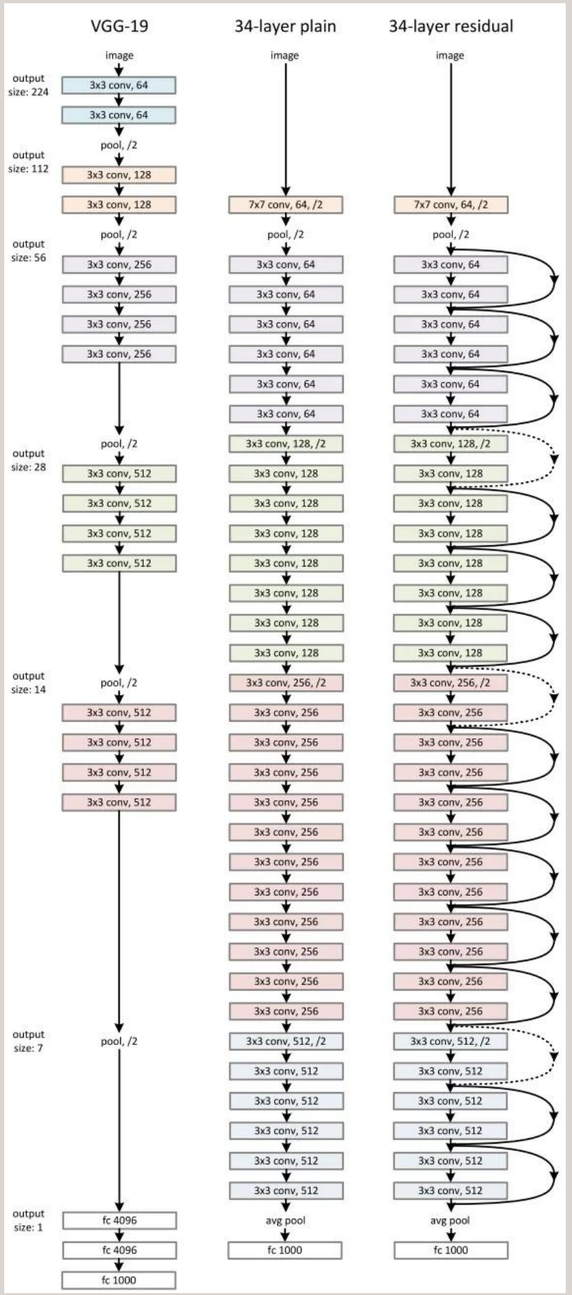
```
<ipython-input-7-bfe12322dac2>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
  history = model.fit_generator(training_generator,
Epoch 1/10
20/20 [==============================] - 319s 15s/step - loss: 1.5331 - accuracy: 0.4177 - val_loss: 1.0582 - val_accuracy: 0.5312
Epoch 2/10
20/20 [==============================] - 108s 5s/step - loss: 0.9657 - accuracy: 0.5375 - val_loss: 0.9058 - val_accuracy: 0.5723
Epoch 3/10
20/20 [==============================] - 54s 3s/step - loss: 0.8930 - accuracy: 0.5789 - val_loss: 1.0563 - val_accuracy: 0.4980
Epoch 4/10
20/20 [==============================] - 36s 2s/step - loss: 0.9091 - accuracy: 0.6061 - val_loss: 0.8805 - val_accuracy: 0.5977
Epoch 5/10
20/20 [==============================] - 32s 2s/step - loss: 0.7817 - accuracy: 0.6437 - val_loss: 0.9527 - val_accuracy: 0.5586
Epoch 6/10
20/20 [==============================] - 31s 2s/step - loss: 0.7005 - accuracy: 0.7052 - val_loss: 0.9837 - val_accuracy: 0.5371
Epoch 7/10
20/20 [==============================] - 27s 1s/step - loss: 0.7446 - accuracy: 0.6757 - val_loss: 1.1000 - val_accuracy: 0.5078
Epoch 8/10
20/20 [==============================] - 28s 1s/step - loss: 0.7382 - accuracy: 0.6695 - val_loss: 0.8845 - val_accuracy: 0.6055
Epoch 9/10
20/20 [==============================] - 28s 1s/step - loss: 0.6726 - accuracy: 0.7076 - val_loss: 0.9102 - val_accuracy: 0.6055
Epoch 10/10
20/20 [==============================] - 29s 1s/step - loss: 0.6114 - accuracy: 0.7352 - val_loss: 1.1217 - val_accuracy: 0.5332
```

Validation Train Graph



1. ResNet

- The fundamental feature of ResNet is the inclusion of a connection type known as "residual learning" or "skip connection." These connections directly add the input from the previous layer to the next layer. As a result, it prevents information loss in deeper layers of the network, making training easier.



Model Setup

- ResNet

```
history = model.fit_generator(
    training_generator,
    epochs= 10,
    validation_data = validation_generator,
    validation_steps = 10,
    steps_per_epoch = 20,
    class_weight=class_weights)
```

```
<ipython-input-40-16d0f521c88d>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Plea
  history = model.fit_generator(
Epoch 1/10
20/20 [==============================] - ETA: 0s - loss: 1.4623 - accuracy: 0.3333WARNING:tensorflow:Your input ran out of data;
20/20 [==============================] - 36s 1s/step - loss: 1.4623 - accuracy: 0.3333 - val_loss: 1.2498 - val_accuracy: 0.3351
Epoch 2/10
20/20 [==============================] - 26s 1s/step - loss: 1.2389 - accuracy: 0.3195
Epoch 3/10
20/20 [==============================] - 22s 1s/step - loss: 1.1272 - accuracy: 0.3187
Epoch 4/10
20/20 [==============================] - 24s 1s/step - loss: 1.1183 - accuracy: 0.3516
Epoch 5/10
20/20 [==============================] - 22s 1s/step - loss: 1.0985 - accuracy: 0.3719
Epoch 6/10
20/20 [==============================] - 22s 1s/step - loss: 1.1039 - accuracy: 0.3742
Epoch 7/10
20/20 [==============================] - 22s 1s/step - loss: 1.1022 - accuracy: 0.3079
Epoch 8/10
20/20 [==============================] - 21s 1s/step - loss: 1.1053 - accuracy: 0.3915
Epoch 9/10
20/20 [==============================] - 21s 1s/step - loss: 1.0918 - accuracy: 0.3805
Epoch 10/10
20/20 [==============================] - 21s 1s/step - loss: 1.0867 - accuracy: 0.3867
```

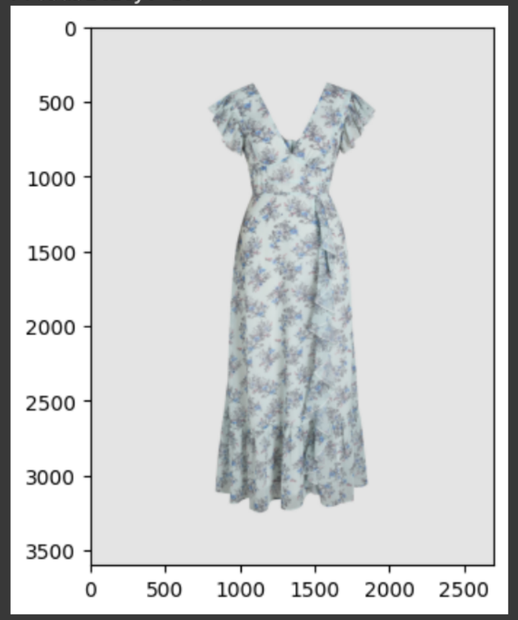
1. MobileNet

```
history33 = model_mb.fit(training_generator, epochs=10)
```

```
Epoch 1/10
73/73 [==============================] - 42s 564ms/step - loss: 0.5369 - accuracy: 0.7713
Epoch 2/10
73/73 [==============================] - 43s 582ms/step - loss: 0.5129 - accuracy: 0.7839
Epoch 3/10
73/73 [==============================] - 40s 545ms/step - loss: 0.4715 - accuracy: 0.8103
Epoch 4/10
73/73 [==============================] - 39s 536ms/step - loss: 0.4405 - accuracy: 0.8203
Epoch 5/10
73/73 [==============================] - 41s 556ms/step - loss: 0.3992 - accuracy: 0.8372
Epoch 6/10
73/73 [==============================] - 39s 540ms/step - loss: 0.3906 - accuracy: 0.8333
Epoch 7/10
73/73 [==============================] - 43s 590ms/step - loss: 0.3870 - accuracy: 0.8432
Epoch 8/10
73/73 [==============================] - 40s 545ms/step - loss: 0.3770 - accuracy: 0.8445
Epoch 9/10
73/73 [==============================] - 41s 555ms/step - loss: 0.3560 - accuracy: 0.8528
Epoch 10/10
73/73 [==============================] - 39s 533ms/step - loss: 0.3193 - accuracy: 0.8748
```

VGG16

```
1: round - Probability: 9.844415947824633e-28
2: scoop - Probability: 8.307911935257099e-34
3: v-neck - Probability: 1.0
Predicted Class: v-neck
Probability: 1.0
```



MobileNet

```
1/1 [==============================] - 1s 798ms/step
1: Round - Probability: 0.004099790472537279
2: Scoop - Probability: 0.48157933354377747
3: V-neck - Probability: 0.5143209099769592
Predicted Class: V-neck
Probability: 0.5143209099769592
```

ResNet

```
1/1 [==============================] - 0s 22ms/step
1: Round - Probability: 0.0788685753941536
2: Scoop - Probability: 0.12228689342737198
3: V-neck - Probability: 0.7988445162773132
Predicted Class: V-neck
Probability: 0.7988445162773132
```