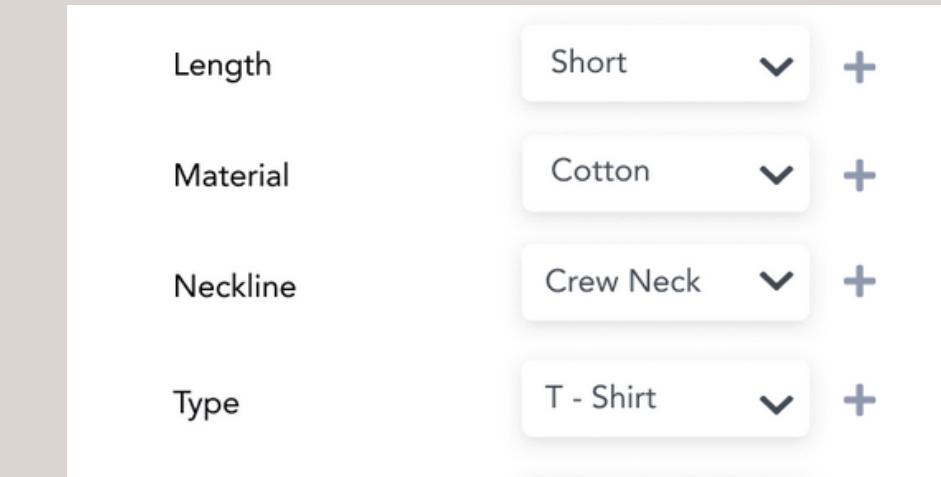


Neckline Classifier

Esma Zeynep Sayılı - Esra Cesur

Problem



- In online shopping, large fashion datasets or e-commerce platforms manually tagging and classifying 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, manufacturers 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.

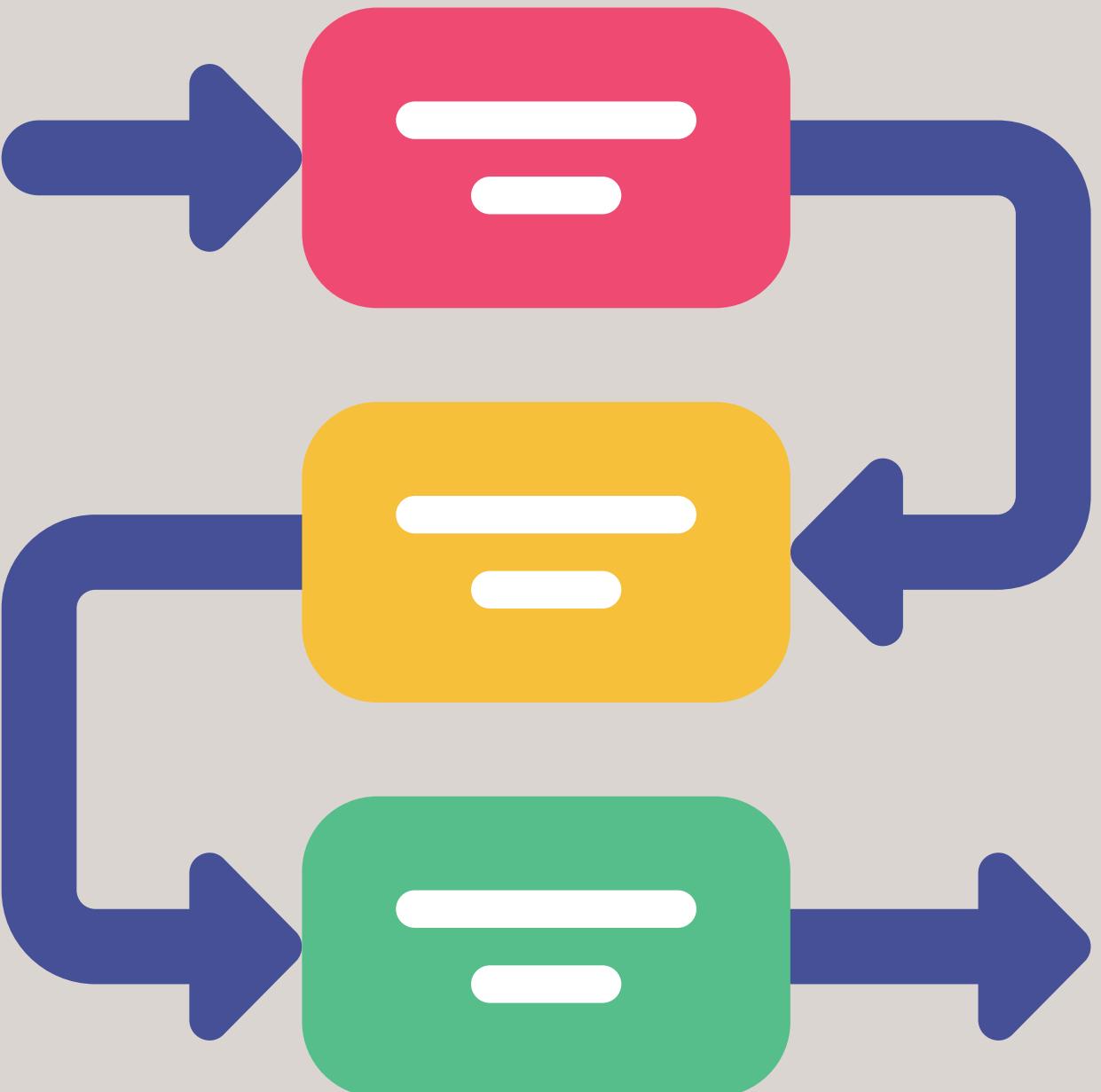
The dataset contains **flatlay images (images without human models) of upper/full body clothes**. We are required to classify if the cloth's region around the neck falls under one of these three categories i.e. **round, scoop or v-neck**.

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



Methodology

1. Data Collection
2. Data Balancing
3. Reviewing Architectures
4. Model Setup



Data Collection

Drive'da arayın

Drive'im > flatlay_images_neckline ▾

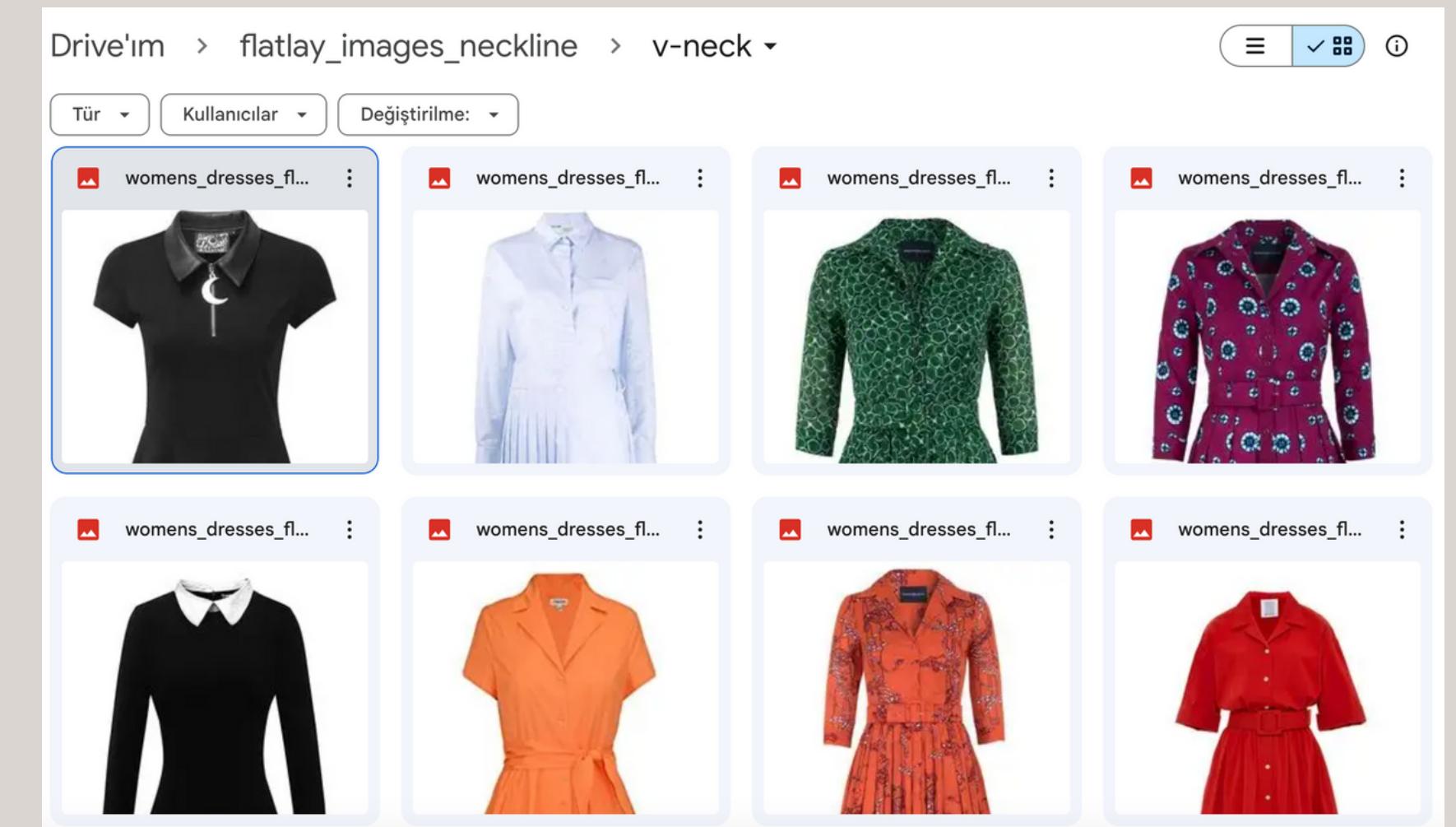
Tür ▾ Kullanıcılar ▾ Değiştirilme: ▾

Klasörler

- v-neck
- scoop
- round

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

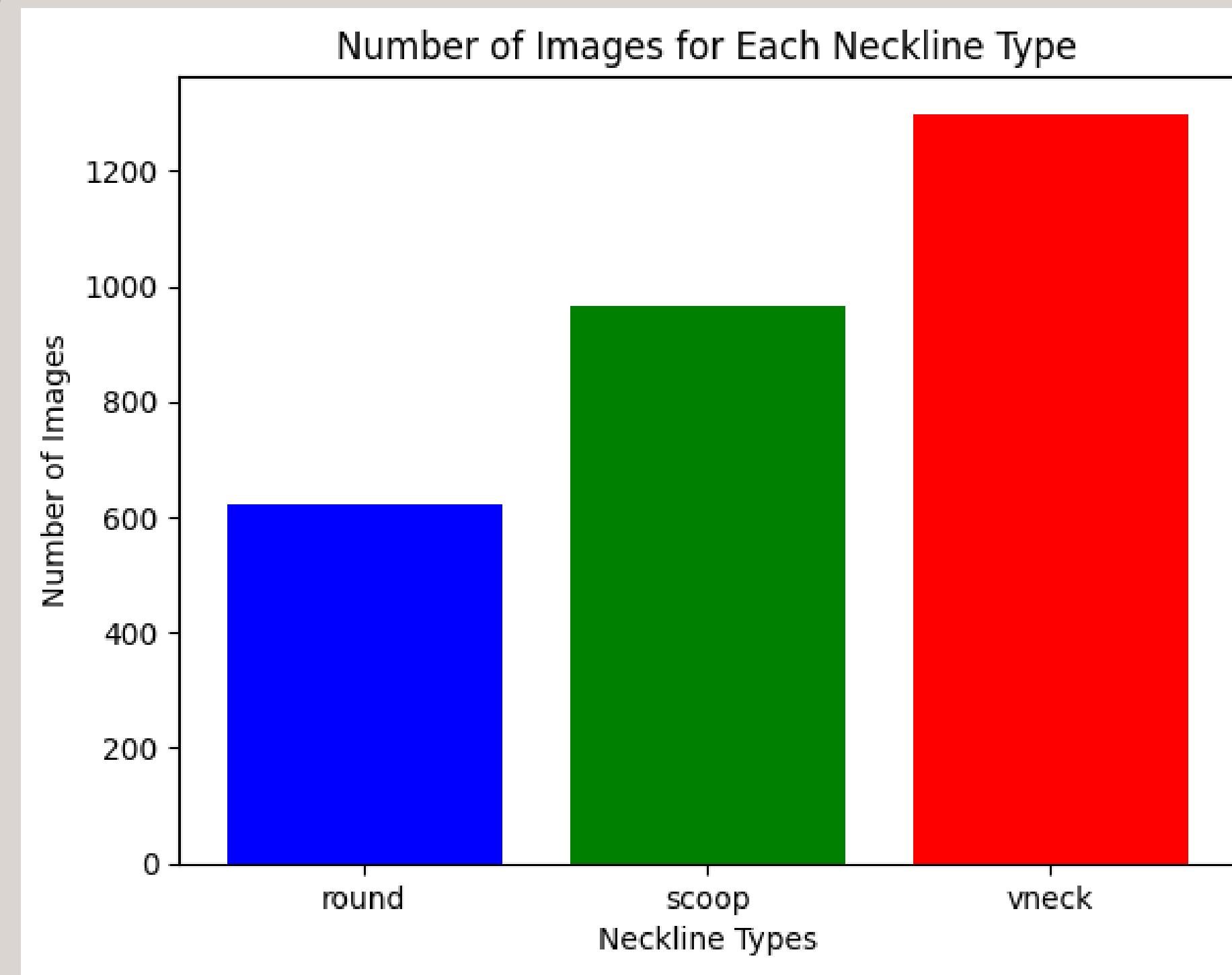
Mounted at /content/drive
```



Visualization of Dataset



Visualization of “Unbalanced Data”

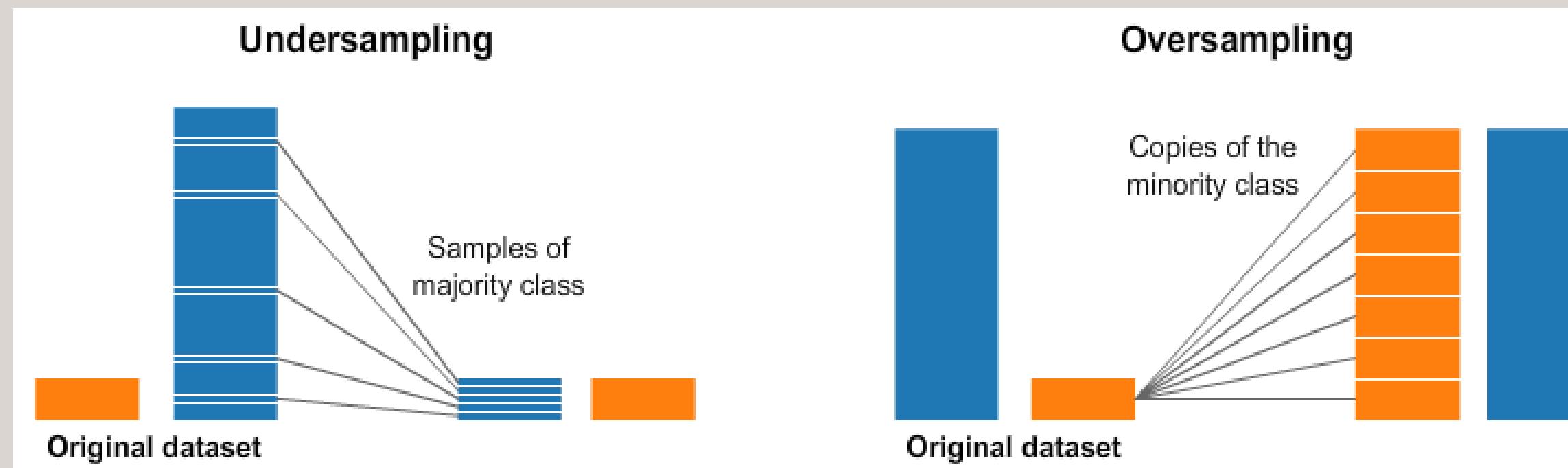


Round = 621
Scoup = 966
V-Neck = 1298

Data Balancing

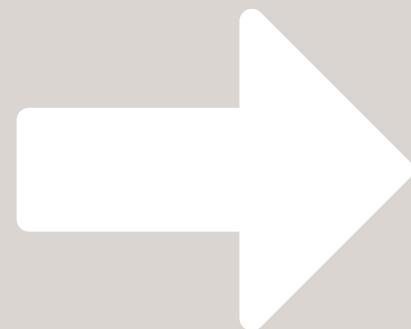
Resampling

- Undersampling : Remove samples from the majority class until the class distribution is balanced.
- Oversampling: Duplicate or generate synthetic samples for the minority class until the class distribution is balanced.



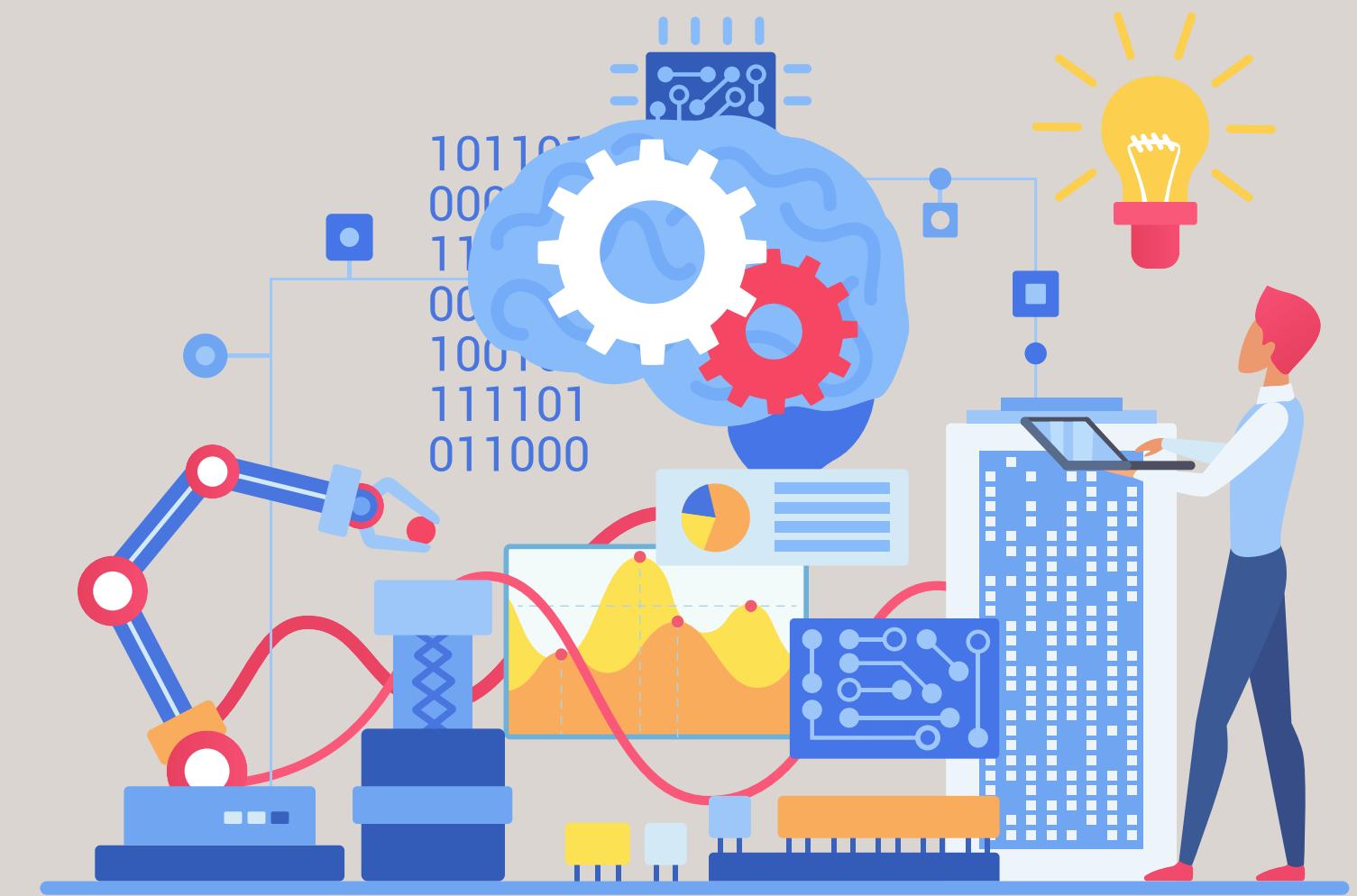
Resizing The Images

All images were in different sizes in the base dataset.
We cropped and later resized these images to (224,224)



Reviewing Architectures

1. VGG16
2. ResNet
3. Mobilenet



1. VGG16

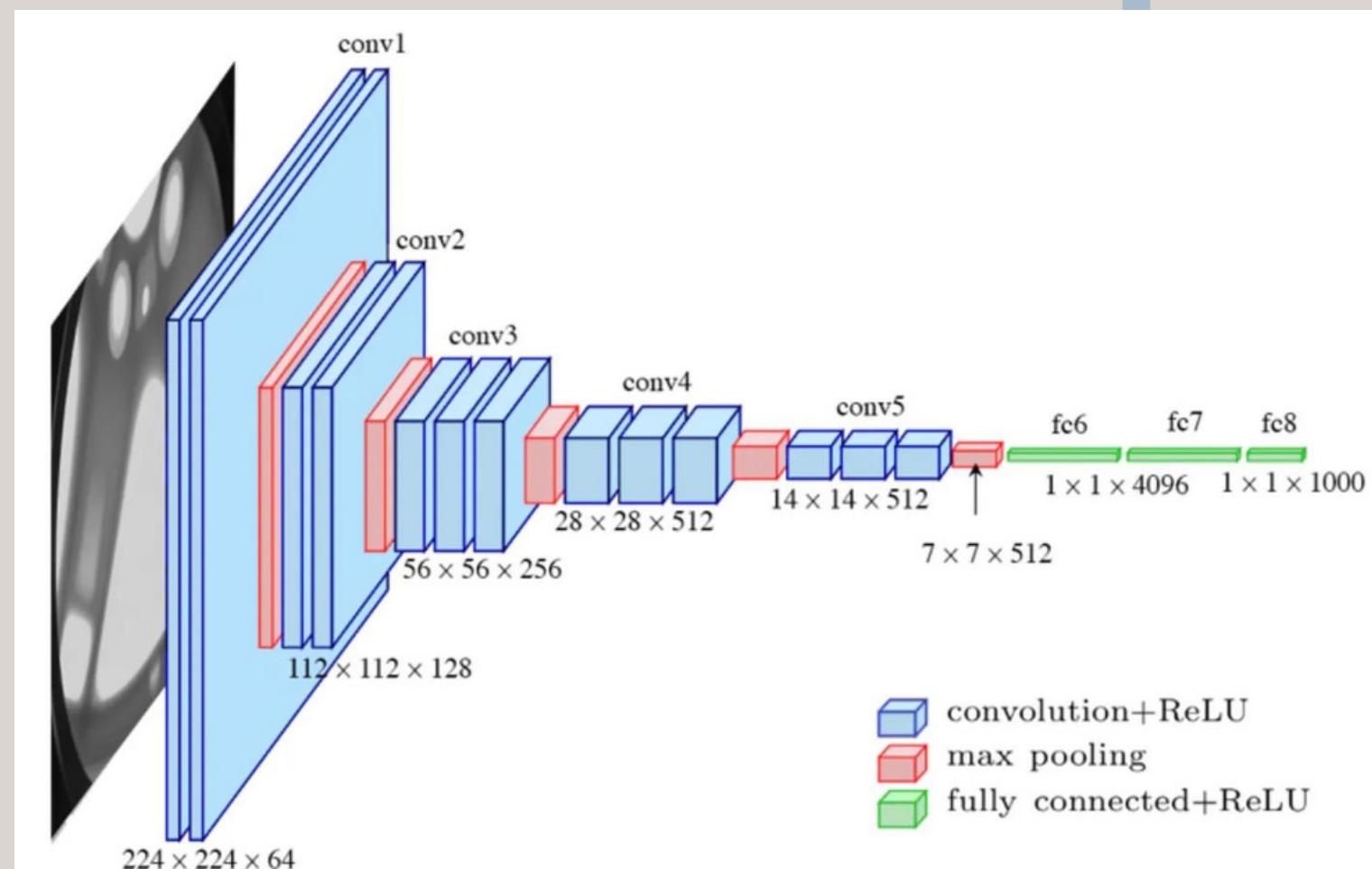


- VGG is a deep convolutional neural network that was proposed by Karen Simonyan and Andrew Zisserman.
- The VGG model investigates the depth of layers with a very small convolutional filter size (3×3) to deal with large-scale images.

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)

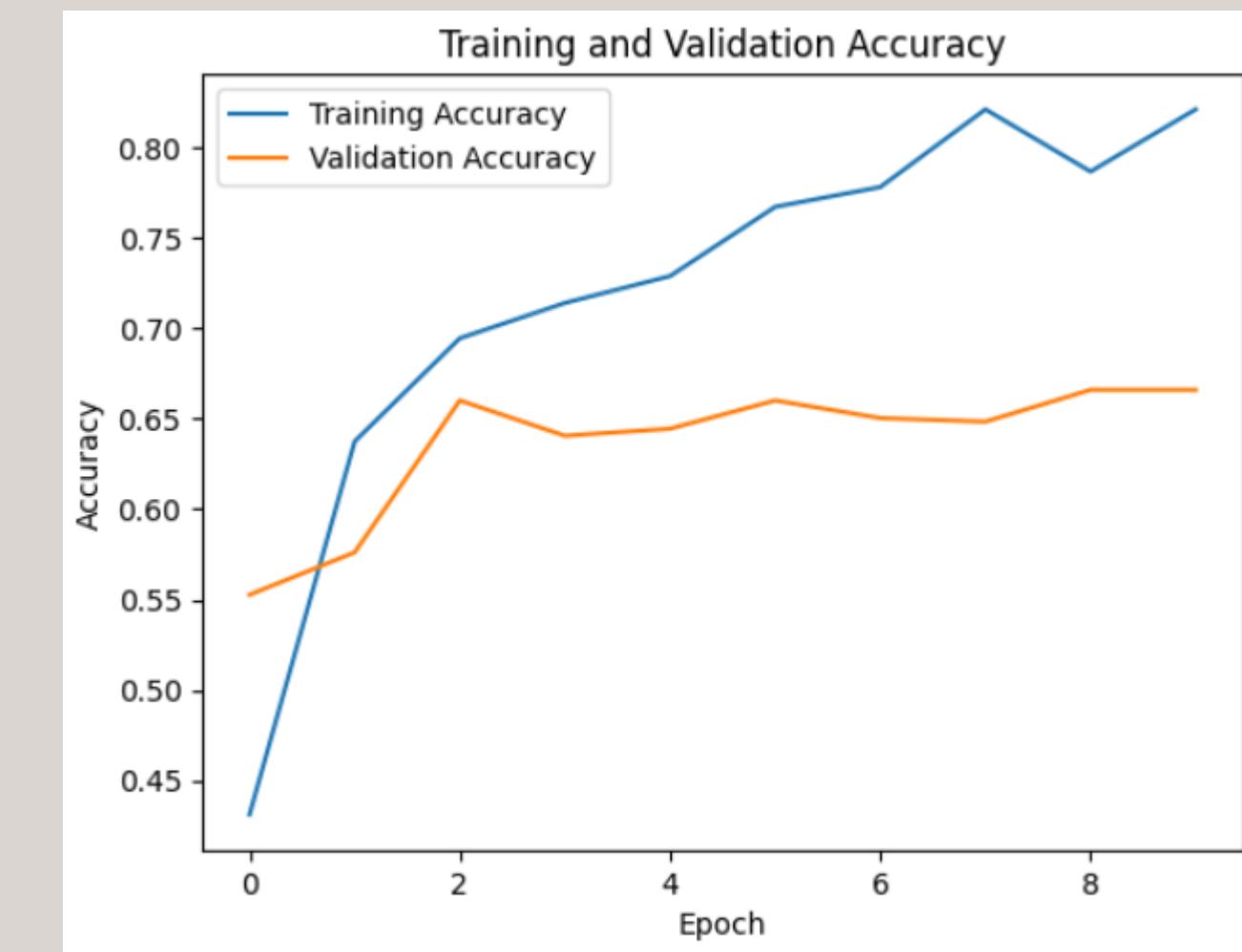
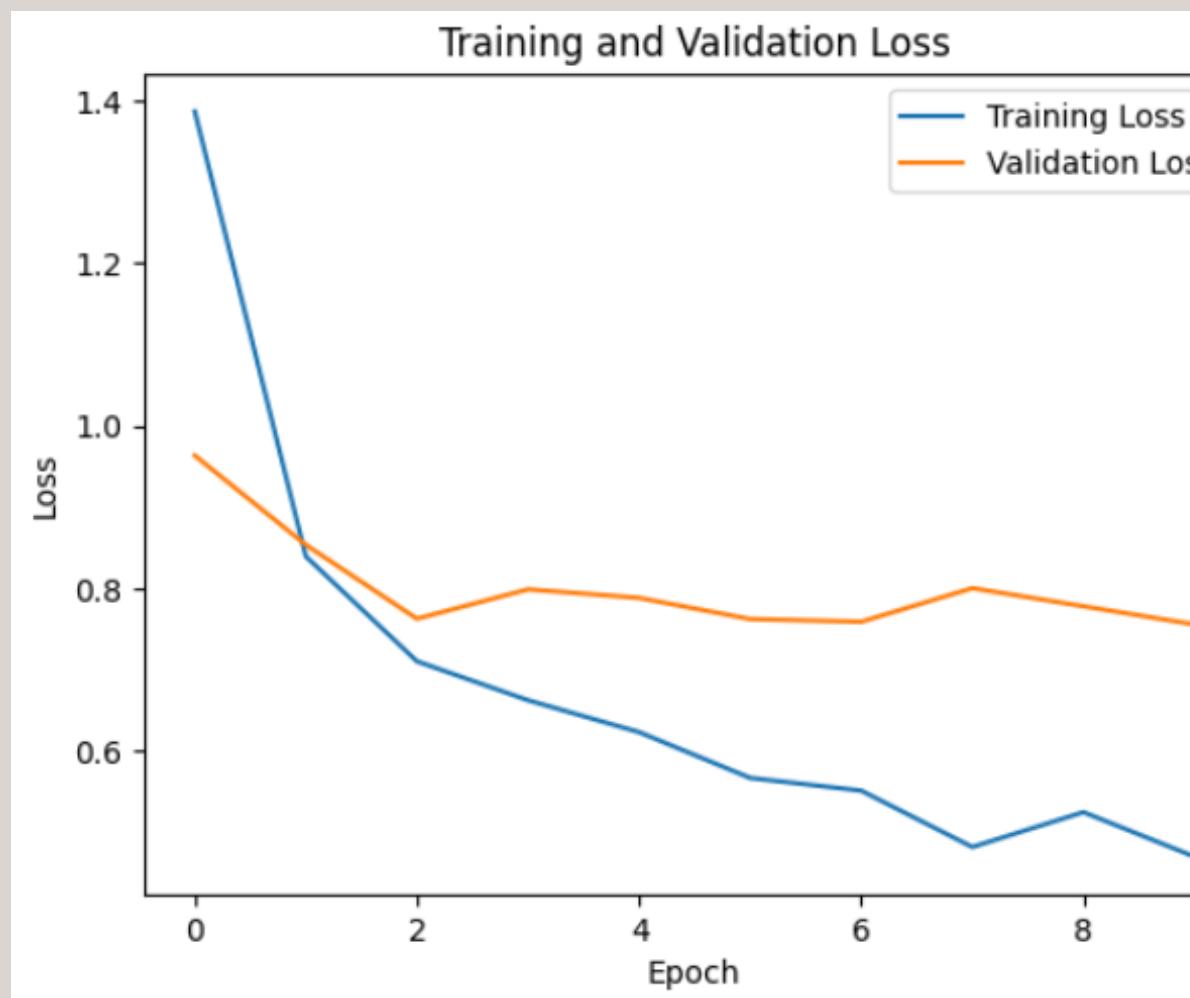
model = Model(inputs = vgg.input, outputs = x)

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

Train - Validation Loss and Accuracy Graph

VGG16

Oversampling



2. MobileNet

MobileNet consists of 86 layers.

Block 1

Model: "mobilenet_1.00_224"		
Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[None, 224, 224, 3]	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalizati on)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormali zation)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
conv_pw_1_bn (BatchNormali zation)	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0

Block 2

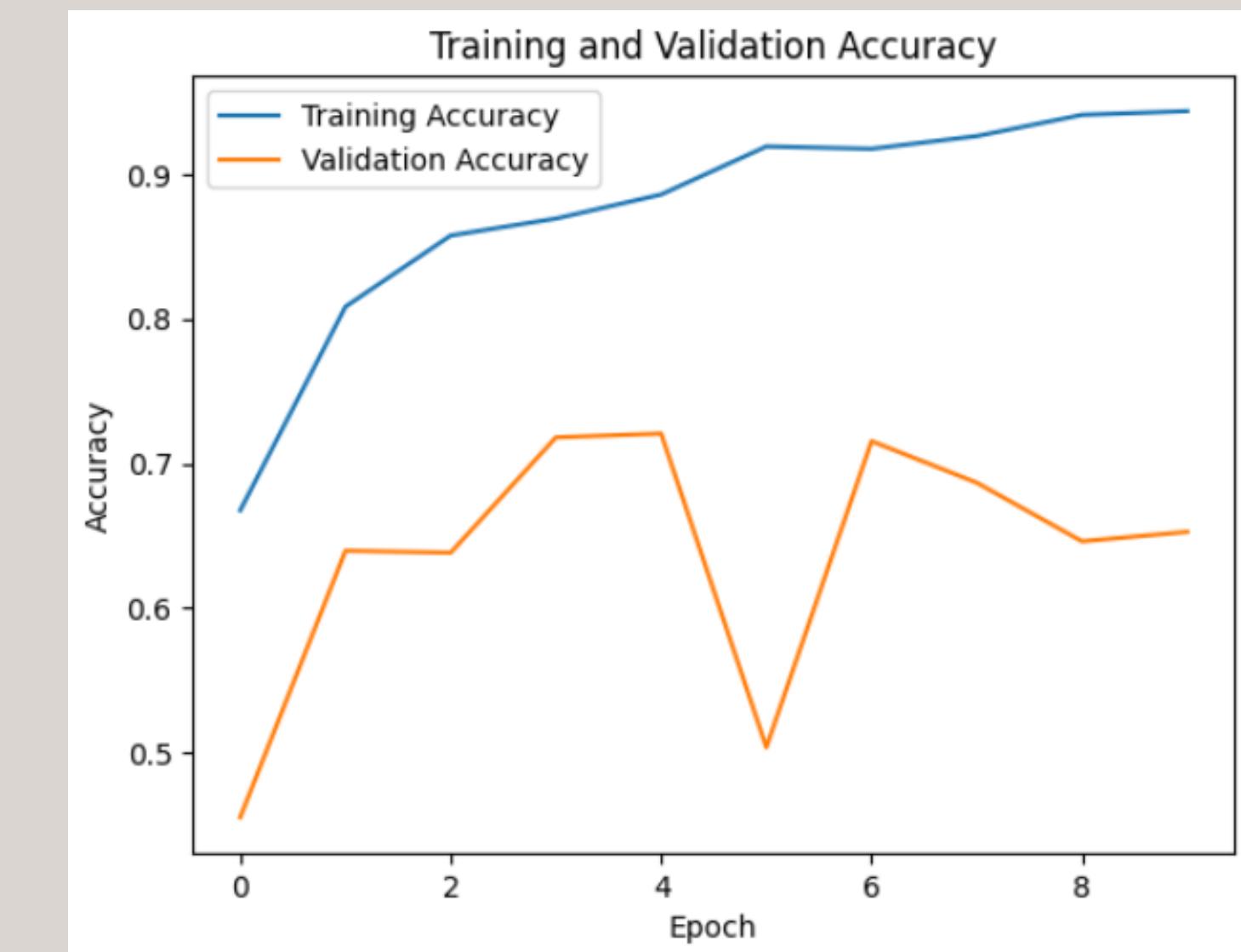
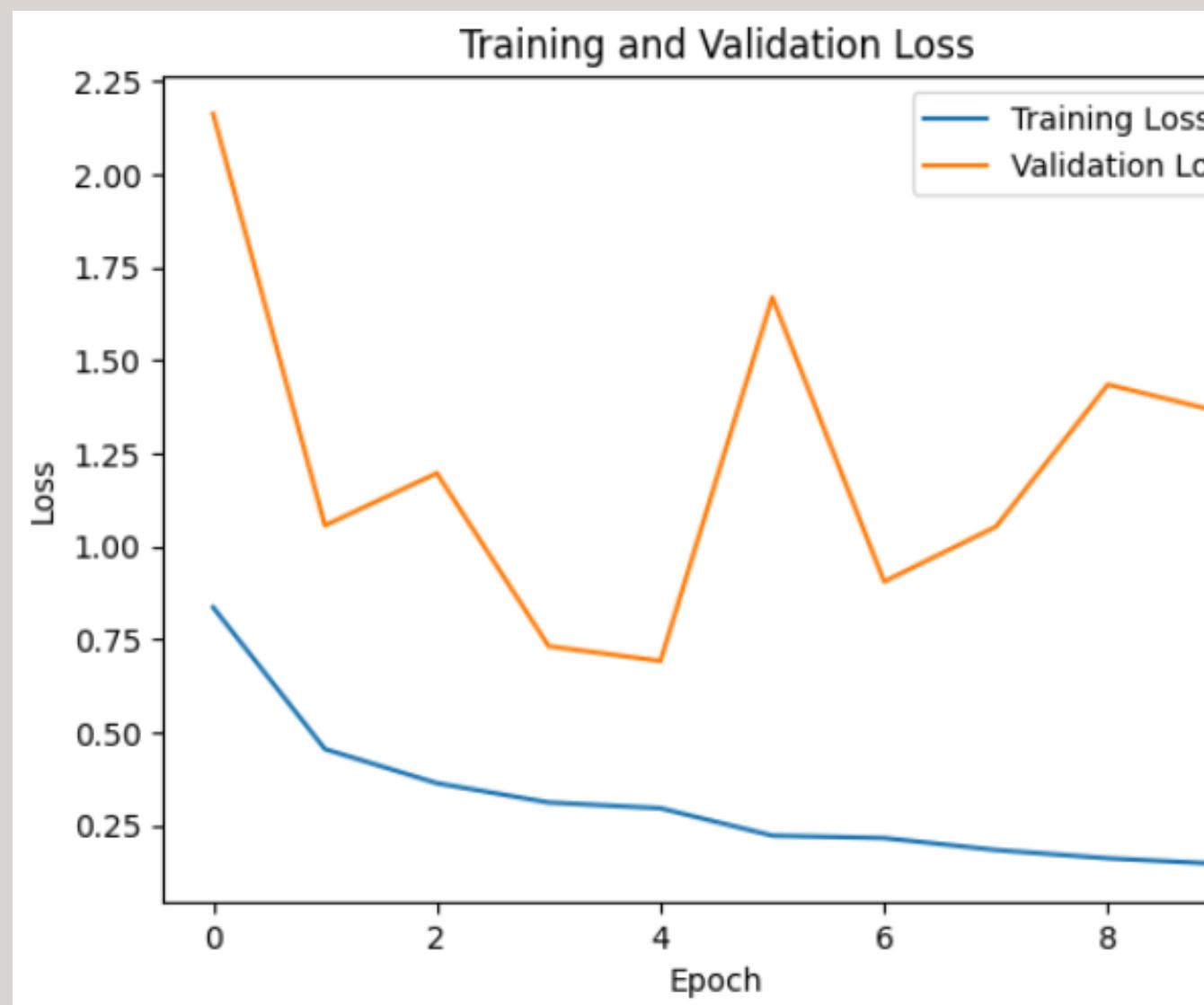
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormali zation)	(None, 56, 56, 64)	256
conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8192
conv_pw_2_bn (BatchNormali zation)	(None, 56, 56, 128)	512
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	0

These blocks repeat for 13 times.

Train - Validation Loss and Accuracy Graph

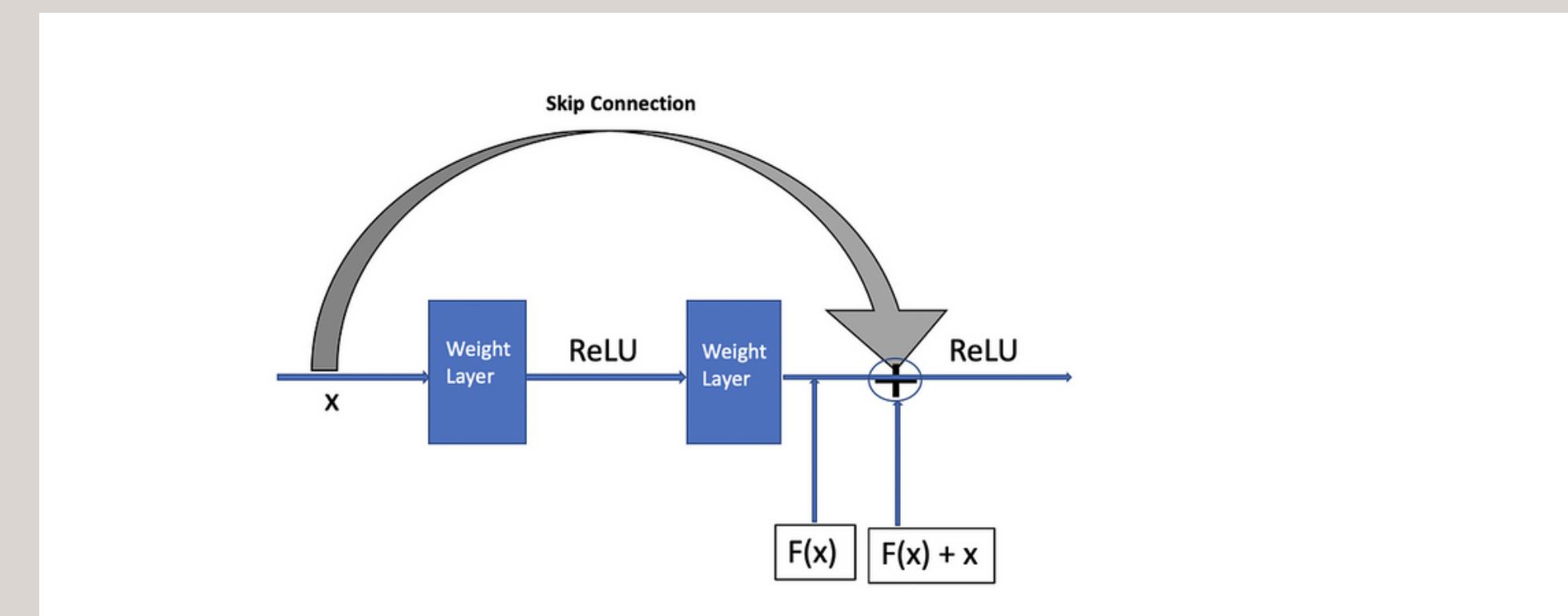
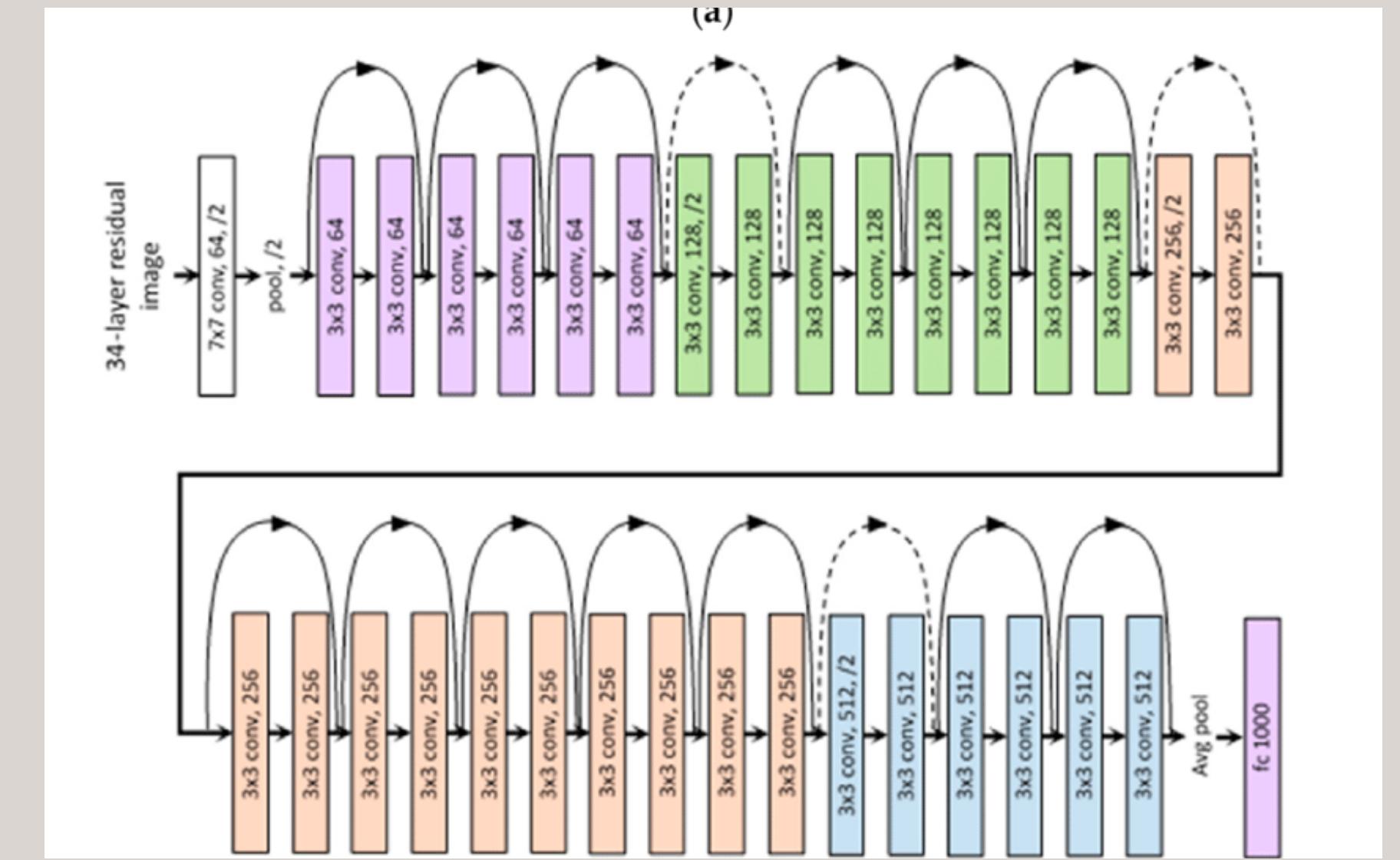
MobileNet

Oversampling



3. ResNet50

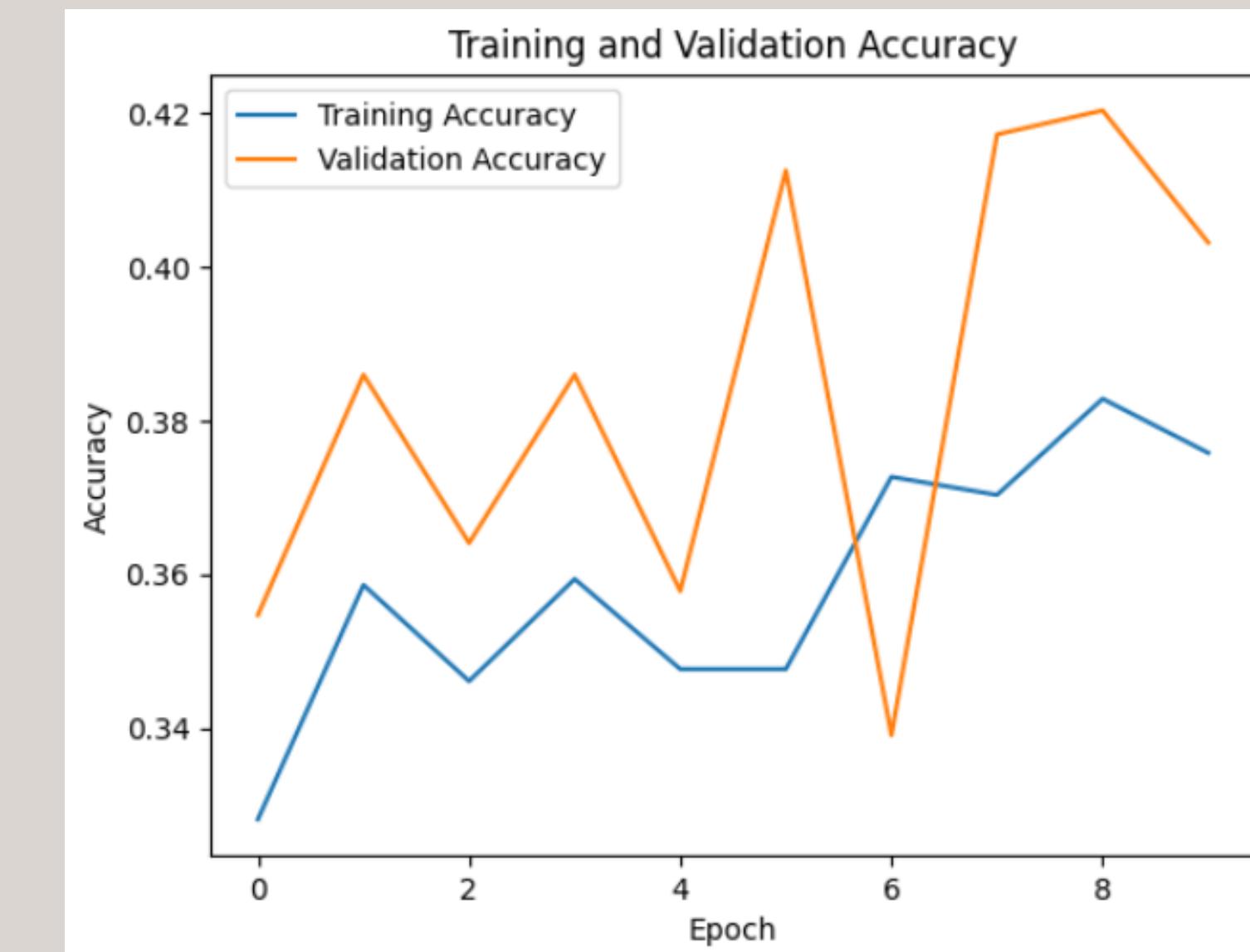
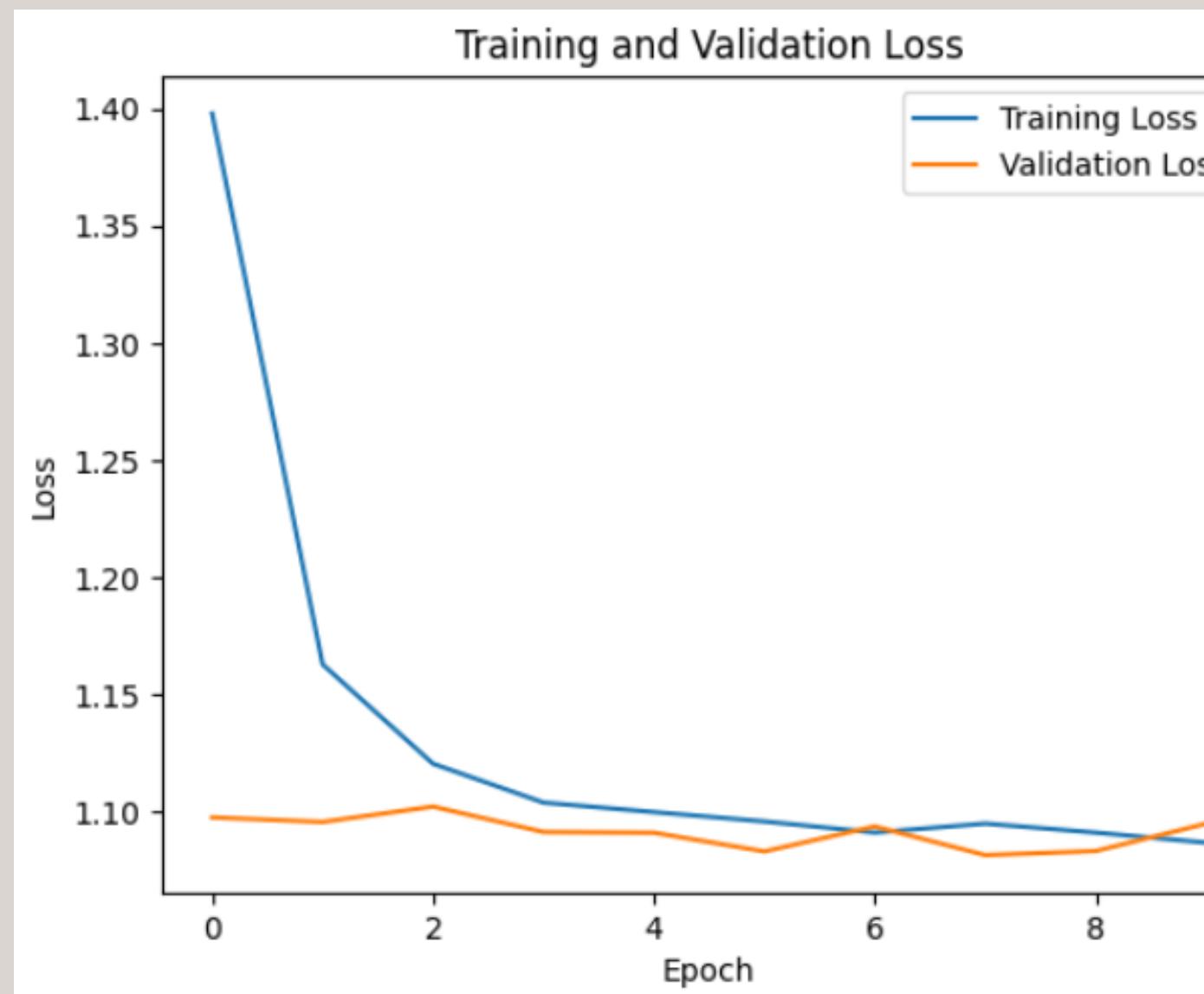
- The fundamental feature of ResNet is the inclusion of a connection type known as "residual learning" or "skip connection." These connections directly add the input from the previous layer to the next layer. As a result, it prevents information loss in deeper layers of the network, making training easier.



Train - Validation Loss and Accuracy Graph

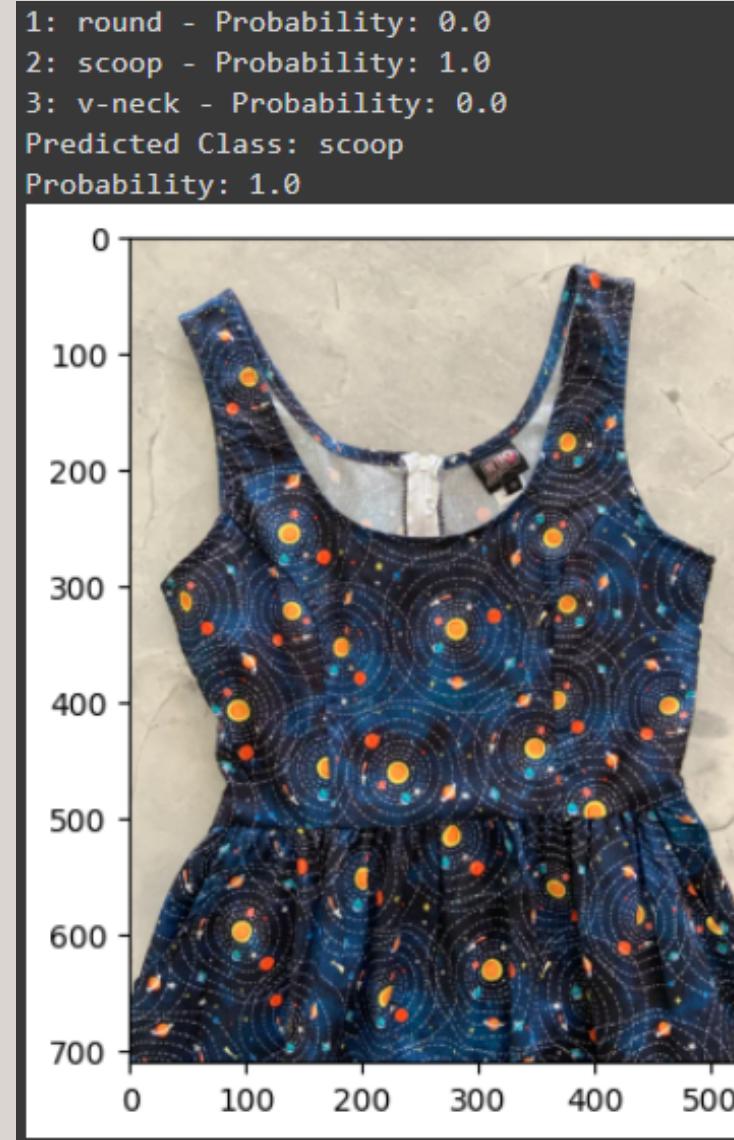
Oversampling

ResNet50



Oversampling

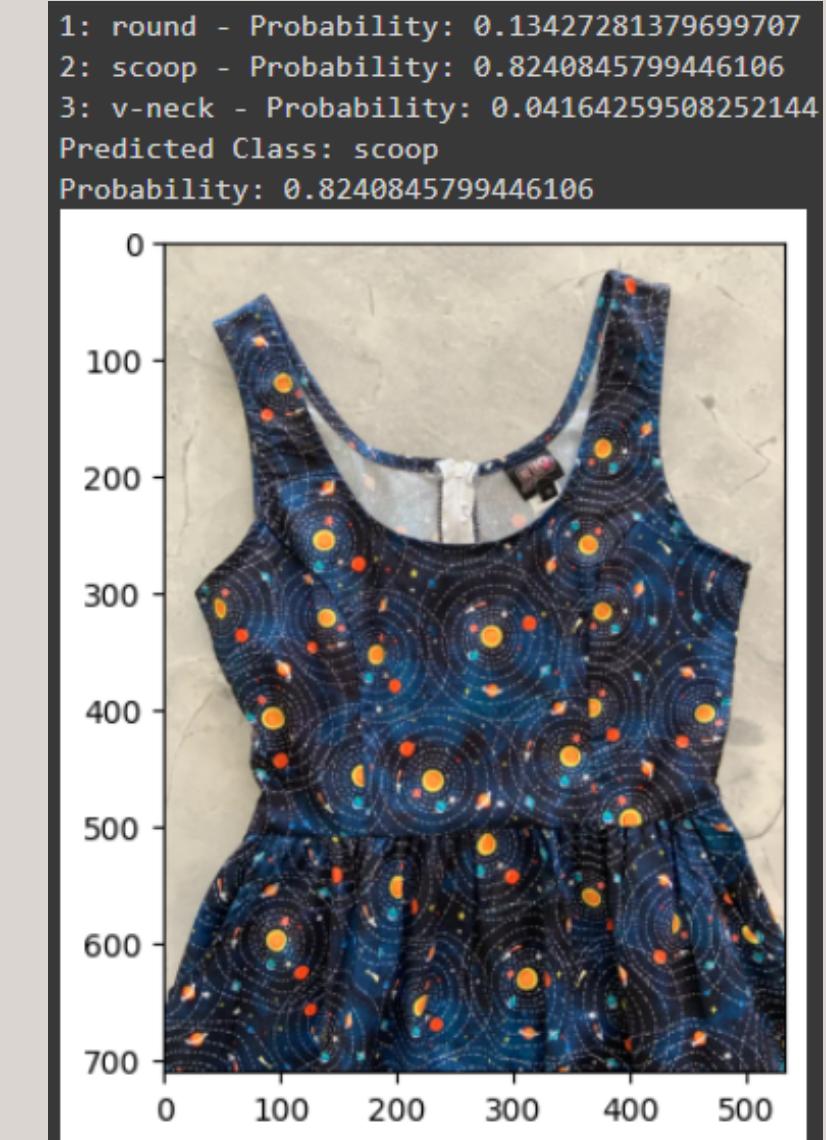
VGG16



MobileNet



ResNet50



Probability of Classes	Oversampling		
	VGG16	MobileNet	ResNet50
Round	0.0	0.0000001	0.13
Scoop	1.0	0.96	0.82
V-Neck	0.0	0.03	0.04
Predicted Class	Scoop	Scoop	Scoop