Image Classification

Submitted by:

Ali Asgar

ACKNOWLEDGMENT

- https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- https://keras.io
- https://www.thinkautomation.com/eli5/eli5-what-is-image-classification-in-deep-learning/
- https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy
- https://keras.io/guides/sequential_model/
- https://keras.io/guides/functional_api/
- https://neurohive.io/en/popular-networks/resnet/
- https://neurohive.io/en/popular-networks/vgg16/
- https://keras.io/guides/transfer_learning/

INTRODUCTION

Business Problem Framing

We are trying to build an image classification model using deep learning techniques. Images play a crucial role are more effective than text data. AI is using images to gather various information and advance in the field of visual data interpretation. We are here to build a classification model which can certainly classify between three categories which are Saree, Jeans(men), Trousers(men). We need to scrape images for the following three categories from e-commerce websites and build the model using them for training.

Conceptual Background of the Domain Problem

Image classification in Deep learning!

Image classification is a technique in deep learning where a system is trained to classify images for various categories. Several images are used to train the model so that it can differentiate between them. More precisely, Image classification is where a computer can analyze an image and identify the 'class' the image falls under. (Or a probability of the image being part of a 'class'.) A class is essentially a label, for instance, 'car', 'animal', 'building', and so on. For us, classifying images is no big deal. But it's a perfect example of Moravec's

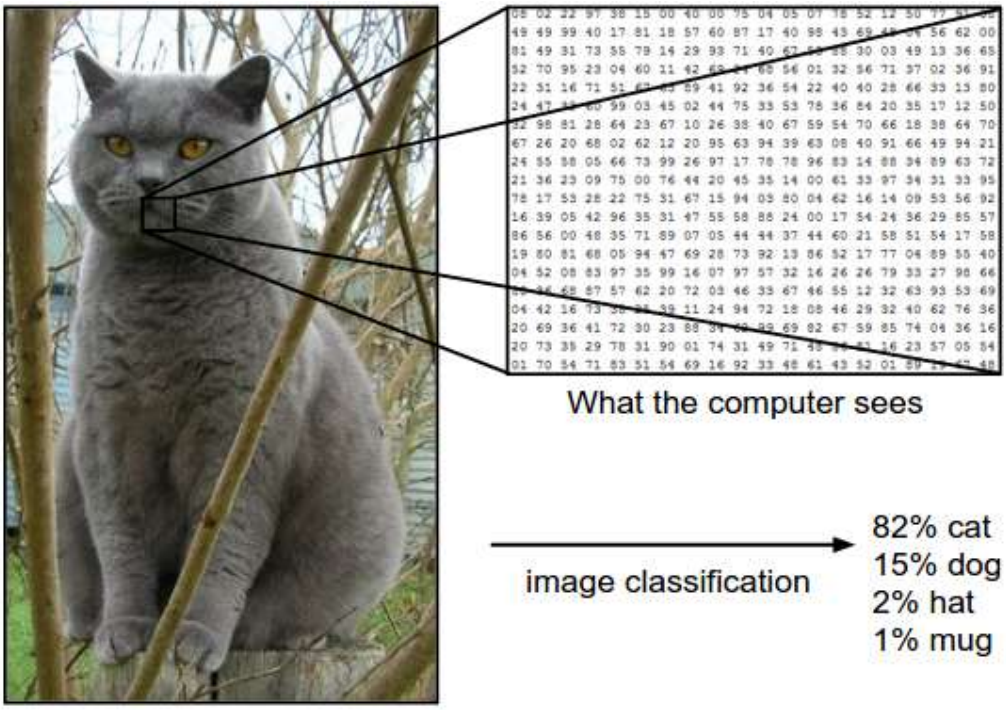
paradox when it comes to machines. (That is, the things we find easy are difficult for AI.)
Early image classification relied on raw pixel data. This meant that computers would break down images into individual pixels. The problem is that two pictures of the same thing can look very different. They can have different backgrounds, angles, poses, etcetera. This made it quite the challenge for computers to correctly 'see' and categorize images.

Image classification has a few uses — and vast potential as it grows in reliability. Here are just a few examples of what makes it useful.
Self-driving cars use image classification to identify what's around them. I.e. trees, people, traffic lights, and so on.
Image classification can also help in healthcare. For instance, it could analyze medical images and suggest whether they classify as depicting a symptom of illness.
Or, for example, image classification could help people organize their photo collections.

Review of Literature

What is Deep Learning?

Deep learning is a type of machine learning; a subset of artificial intelligence (AI) that allows machines to learn from data. Deep learning involves the use of computer systems known as neural networks.
In neural networks, the input filters through hidden layers of nodes. These nodes each process the input and communicate their results to the next layer of nodes. This repeats until it reaches an output layer, and the machine provides its answer.
There are different types of neural networks based on how the hidden layers work. Image classification with deep learning most often involves convolutional neural networks or CNN. In CNN, the nodes in the hidden layers don't always share their output with every node in the next layer (known as convolutional layers).
Deep learning allows machines to identify and extract features from images. This means they can learn the features to look for in images by analyzing lots of pictures. So, programmers don't need to enter these filters by hand.



Motivation for the Problem Undertaken

Image classification and identification are one of the most challenging and required topics in the field of Artificial Intelligence. It comprises a technique that analyses an image and extracts features to understand its orientation and various features. It helps in the classification of things and even helps in various fields like medical, security, entertainment, etc. Machine learning together with deep learning enables the system to read the image and make the system understand which can, later on, make use of it for various purposes. It is a matter of interest to work towards this field to make most of it and make use of it to make things better and more efficient.

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem

Our objective for the given problem is to create an image identification model which certainly can differentiate between the given categories. For the task to be done, we are required to fetch image data of Sarees, Jeans(men), and Trousers(men) from e-commerce websites.

We extracted around 1600 images of these categories. Images fetched have different orientations and types of each category. We will further increase the data with data augmentation techniques while importing it into python.

We separated the images into train and test datasets where train images are used for training purposes while images in the test dataset are used for validation.

Data Sources and their formats

For the given problem we collected the data from various e-commerce websites. We scraped images of sarees, jeans(men), and trousers (men). We then create a final dataset that was used to train the machine learning model.

We used Selenium web driver to scrape data and defined functions to scrape data from various websites:

```
url_amzn=('https://www.amazon.in/')
```

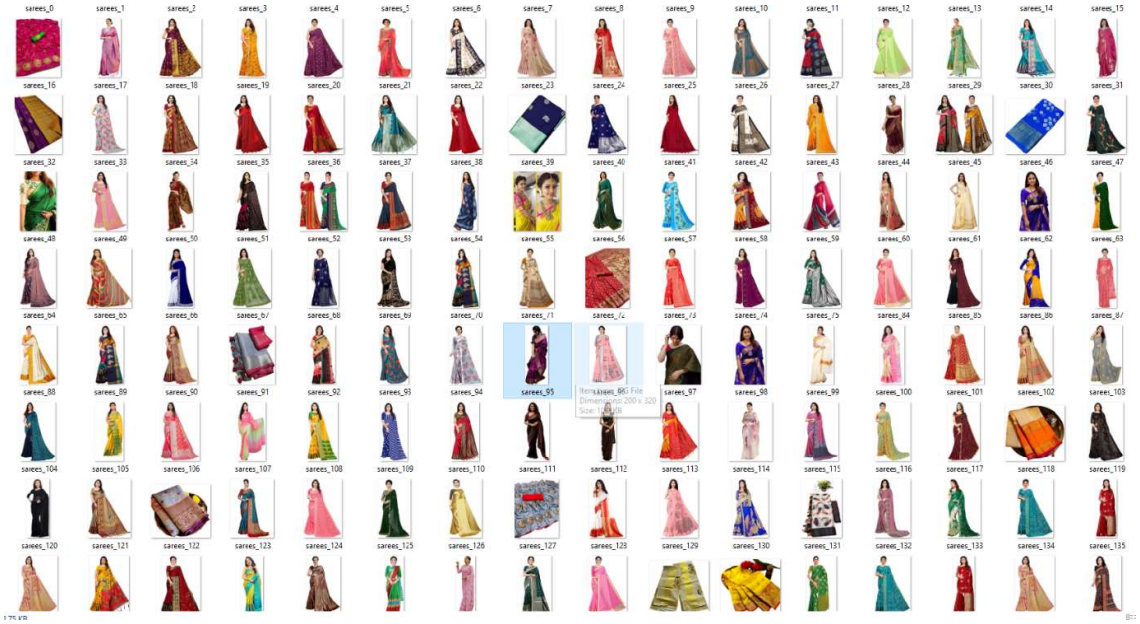
```
def amazon_buy(url,product1,product2,product3):

    # Establishing Connection with the website
    driver.get(url)
    time.sleep(2) # pausing the code execution for 2 seconds for website to load

    product_list=[product1,product2,product3]

    for product in product_list:
        # Creating object of the search bar by finding it through ID
        srch_bar=driver.find_element_by_id('twotabsearchtextbox')

        # Clearing the bar if there is any text in it
        srch_bar.clear()

        # Sending required product to the search bar
        srch_bar.send_keys(product)

        # Creating onject of the search button and sending click commamnd
        srch_btn=driver.find_element_by_id('nav-search-submit-button')
        srch_btn.click()
        time.sleep(3)
```

```
        # For extracting image urls
        img_urls=[]

        for i in range (0,8):

            # Extracting URLs of all the images on the page

            image=driver.find_elements_by_xpath("//img[@class='s-image']")
            for img in image:
                source=img.get_attribute('src')
                if source is not None:
                    if (source[0:4]=='http'):
                        img_urls.append(source)
            try:
                driver.find_element_by_xpath("//li[@class='a-last']/a").click()
                time.sleep(2)
            except NoSuchElementException:
                print('End of pages...')
        time.sleep(1)

        # Creating Directory
        prod_name = str(product).split()
        prod_name = '_'.join(prod_name)

        directory = "Image_Classification/"+str(prod_name)

        if not os.path.exists(directory):
            os.makedirs(directory)

        # Retrieving images in a directory we created
        cnt=0
        for image in img_urls:
            response=requests.get(image)
            file=open(r""+str(directory)+"/"+str(prod_name)+"_"+str(cnt)+".jpg","wb")
            file.write(response.content)
            cnt+=1

        # Printing Image count

        print('********************')
        print(f"\n In {str.upper(prod_name)} folder there are {cnt} images. \n")
        print('********************')
```

```
amazon_buy(url_amzn,'sarees','men jeans','men trousers')
```

```
End of pages...
End of pages...
********************

 In SAREES folder there are 444 images.

********************
End of pages...
End of pages...
********************

 In MEN_JEANS folder there are 447 images.

********************
End of pages...
End of pages...
********************

 In MEN_TROUSERS folder there are 439 images.

********************
```

We then collected all images and separated them to train and test dataset. We kept 400 images of each category for training purposes and keep extra images for testing.



Preprocessing Done

Image Size correction

Scraping the image data from the internet provides you with images of varying sizes. To feed the images as input data to our deep learning model we need to convert the size and pixels to a predefined format suitable for the machine learning model.

We defined width and height which can be used at the time of data import to convert the images to relevant sizes.

```
# defining Width and Height
width,height = 224,224
```

We then created ImageDataGenerator instances for importing data with defined scaling and also data augmentation techniques.

```
# Initializing ImageDataGenerator to import data with 1/255 scalling, and data augmentation for Train Dataset and Test Dataset

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
```

Creating Train and Test dataset using Flow_From _Directory function of ImagaDataGenerator

```
# Importing train data using flow from directory

train_set = train_datagen.flow_from_directory('Image_Classification/Train/',target_size=(width,height),batch_size=32,
                                              class_mode='categorical')
```

```
Found 1200 images belonging to 3 classes.
```

```
test_set = test_datagen.flow_from_directory('Image_Classification/Test/',target_size=(width,height),batch_size=5,
                                            class_mode='categorical')
```

```
Found 104 images belonging to 3 classes.
```

For output layer units we used Glob library to enumerate the number of folders in the train dataset and count them as categories.

```
# useful for getting number of classes
folders = glob('Image_Classification/Train/*')
```

```
folders
```

```
['Image_Classification/Train\\men_jeans',
 'Image_Classification/Train\\men_trousers',
 'Image_Classification/Train\\sarees']
```

Hardware and Software Requirements and Tools Used

Software Requirement:

1. Anaconda Navigator software.

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.

2. Jupyter Notebook - as an IDE for writing Python codes for analyzing data and machine learning purposes.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents that combine explanatory text, mathematics, computations, and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects

3. Python Programming

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and

code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed

4. Libraries Used

Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms basic linear algebra, basic statistical operations, random simulation and much more.

Pandas

Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data-wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState's ActivePython.

Matplotlib

Matplotlib is a cross-platform, data visualization, and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

- The pyplot API is a hierarchy of Python code objects topped by *matplotlib.pyplot*

- An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

Selenium

Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. It provides extensions to emulate user interaction with browsers, a distribution server for scaling browser allocation, and the infrastructure for implementations of the W3C WebDriver specification that lets you write interchangeable code for all major web browsers.

This project is made possible by volunteer contributors who have put in thousands of hours of their own time and made the source code freely available for anyone to use, enjoy, and improve. Selenium brings together browser vendors, engineers, and enthusiasts to further an open discussion around the automation of the web platform. The project organizes an annual conference to teach and nurture the community. At the core of Selenium is WebDriver, an interface to write instruction sets that can be run interchangeably in many browsers.

Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Tensorflow - Keras

TensorFlow is an open-source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient

front-end API for building applications with the framework while executing those applications in high-performance C++.

TensorFlow 2 is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities:

- Efficiently executing low-level tensor operations on CPU, GPU, or TPU.
- Computing the gradient of arbitrary differentiable expressions.
- Scaling computation to many devices, such as clusters of hundreds of GPUs.
- Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile, and embedded devices.

Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

Keras empowers engineers and researchers to take full advantage of the scalability and cross-platform capabilities of TensorFlow 2: you can run Keras on TPU or large clusters of GPUs, and you can export your Keras models to run in the browser or on a mobile device.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

The given scenario challenges us to create a model that can predict the category of the given image. Predicting the image category is one of the favorite and challenging areas in the field of machine learning. For the task to be done, we scraped the required data from e-commerce websites.

Testing of Identified Approaches (Algorithms)

After performing various preprocessing tasks on our dataset we made our data suitable for passing through the algorithms to train the best predicting model.

We split the data into train and test data. We kept 400 images from each category to go for training and put the remaining images in the test folder to be used for validation. After importing the data using flow_from_directory with data augmentation we created a model for data to fit.

Algorithms used for training are :

1. Sequential API

The sequential API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs. A Sequential model is appropriate for **a plain stack of layers** where each layer has **exactly one input tensor and one output tensor**.

A Sequential model is **not appropriate** when:

- Your model has multiple inputs or multiple outputs
- Any of your layers have multiple inputs or multiple outputs
- You need to do layer sharing

- You want non-linear topology (e.g. a residual connection, a multi-branch model)

2. Functional API

The Keras functional API is a way to create models that are more flexible than the Sequential API. The functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs.

The main idea is that a deep learning model is usually a directed acyclic graph (DAG) of layers. So the functional API is a way to build graphs of layers. The functional API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers. You can connect layers to (literally) any other layer. As a result, creating complex networks such as siamese networks and residual networks become possible.

3. Transfer Learning

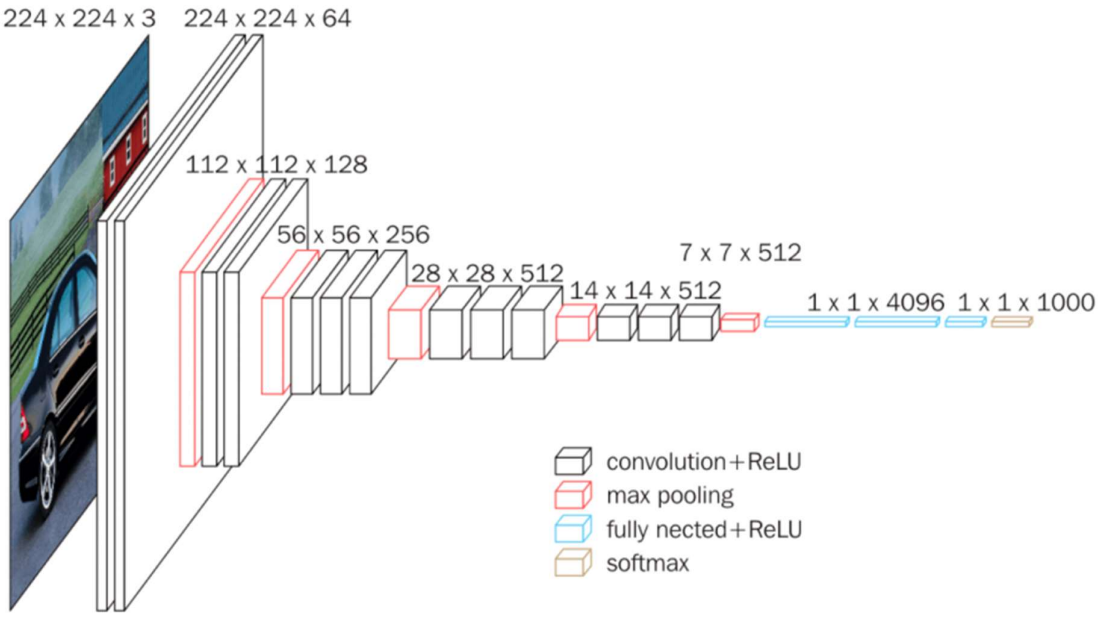
Transfer learning consists of taking features learned on one problem and leveraging them on a new, similar problem. For instance, features from a model that has learned to identify raccoons may be useful to kick-start a model meant to identify tanukis. Transfer learning is usually done for tasks where your dataset has too little data to train a full-scale model from scratch.

The most common incarnation of transfer learning in the context of deep learning is the following workflow:

- Take layers from a previously trained model.
- Freeze them, to avoid destroying any of the information they contain during future training rounds.
- Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
- Train the new layers on your dataset.
- A last, optional step, is fine-tuning, which consists of unfreezing the entire model you obtained above (or part of it) and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pre-trained features to the new data.

Architectures used

1. VGG16



VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous models submitted to ILSVRC-2014. It improves AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPUs.

The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which have a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class).

The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time.

Unfortunately, there are two major drawbacks with VGGNet:

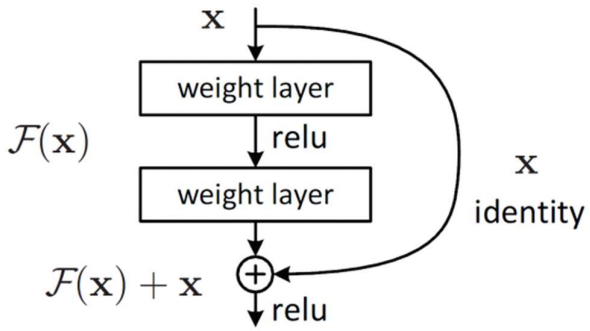
- It is painfully slow to train.
- The network architecture weights themselves are quite large (concerning disk/bandwidth).

Due to its depth and number of fully connected nodes, VGG16 is over 533MB. This makes deploying VGG a tiresome task.VGG16 is used in many deep learning image classification problems; however, smaller network architectures are often more desirable (such as SqueezeNet, GoogLeNet, etc.). But it is a great building block for learning purposes as it is easy to implement.

2. Resnet50

ResNet is a short name for a residual network, but what's residual learning?

Deep convolutional neural networks have achieved the human-level image classification result. Deep networks extract low, middle, and high-level features and classifiers in an end-to-end multi-layer fashion, and the number of stacked layers can enrich the "levels" of features. The stacked layer is of crucial importance, look at the ImageNet result.



When the deeper network starts to converge, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Such degradation is not caused by overfitting or by adding more layers to a deep network leads to higher training error. The deterioration of training accuracy shows that not all systems are easy to optimize.

To overcome this problem, Microsoft introduced a deep residual learning framework. Instead of hoping every few stacked layers directly fit a desired underlying mapping, they explicitly let these layers fit a residual mapping. The formulation of F(x)+x can be realized by feedforward neural networks with shortcut connections. Shortcut connections are those skipping one or more layers shown in Figure 1. The shortcut connections perform identity mapping, and their outputs are added to the outputs of the stacked layers. By using the residual network, many problems can be solved such as:

- ResNets are easy to optimize, but the “plain” networks (that simply stack layers) show higher training error when the depth increases.
- ResNets can easily gain accuracy from greatly increased depth, producing results that are better than previous networks.

We created a model with sequential API adding various convolution layers and max-pooling layers followed by fully connected layers.

```
# Creating a sequential model
model_seq = Sequential()

# adding convolution layers

model_seq.add(Conv2D(input_shape=(224,224,3), filters=64,kernel_size=(3,3), padding='same', activation='relu'))

model_seq.add(Conv2D(filters=64, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(MaxPool2D(pool_size=(2,2),strides =(2,2)))

model_seq.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))

model_seq.add(Conv2D(filters=128, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(Conv2D(filters=128, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(MaxPool2D(pool_size=(2,2),strides =(2,2)))

model_seq.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='relu'))

model_seq.add(Conv2D(filters=256, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(Conv2D(filters=256, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(MaxPool2D(pool_size=(2,2),strides =(2,2)))

model_seq.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', activation='relu'))

model_seq.add(Conv2D(filters=512, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(Conv2D(filters=512, kernel_size=(3,3),padding="same", activation="relu"))

model_seq.add(MaxPool2D(pool_size=(2,2),strides =(2,2)))
# Flattening the data
model_seq.add(Flatten())

# Adding fully connected layers

model_seq.add(Dense(256, activation = 'relu'))

model_seq.add(Dense(512, activation = 'relu'))

model_seq.add(Dense(1024, activation = 'relu'))

model_seq.add(Dense(2048, activation = 'relu'))

model_seq.add(Dense(4096, activation = 'relu'))

model_seq.add(Dense(len(folders), activation = 'softmax'))
```

We then compiled the model with ADAM optimizer having a learning rate of 0.001.

```
from tensorflow.keras.optimizers import Adam
opt = Adam(learning_rate=0.001)
model_seq.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model_seq.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_90 (Conv2D)           (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_91 (Conv2D)           (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_92 (Conv2D)           (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_93 (Conv2D)           (None, 112, 112, 128)     147584
_________________________________________________________________
conv2d_94 (Conv2D)           (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_95 (Conv2D)           (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_96 (Conv2D)           (None, 56, 56, 256)       590080
_________________________________________________________________
conv2d_97 (Conv2D)           (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_98 (Conv2D)           (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_99 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_100 (Conv2D)          (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 14, 14, 512)       0
_________________________________________________________________
flatten_9 (Flatten)          (None, 100352)            0
```

We also used transfer learning technique and used VGG 16 and Resnet50 model to train on our dataset using pre-trained weights.

```
from tensorflow.keras.applications.vgg16 import VGG16
```

```
IMAGE_SIZE = [224, 224]


# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
  layer.trainable = False



  # useful for getting number of classes
folders = glob('Image_Classification/Train/*')


# our layers - you can add more if you want
x = Flatten()(vgg.output)
#x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders),activation='softmax')(x)

# create a model object
model_vgg = Model(inputs=vgg.input, outputs=prediction)
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (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
```

Using Resnet Architecture

```
from tensorflow.keras.applications.resnet50 import ResNet50
```

```
IMAGE_SIZE = [224, 224]


# add preprocessing layer to the front of VGG
res = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
  layer.trainable = False



  # useful for getting number of classes
folders = glob('Image_Classification/Train/*')


# our layers - you can add more if you want
x = Flatten()(vgg.output)
#x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders),activation='softmax')(x)

# create a model object
model_res = Model(inputs=vgg.input, outputs=prediction)
```

Layer (type)	Output Shape	Param #
input_1 (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

Run and Evaluate selected models

After initiating the models we fit the training dataset and used the test dataset for validation purposes.

```
# fit the model
seq_model = model_seq.fit(train_set,validation_data=test_set,epochs=30)
```

```
Epoch 1/30
38/38 [==============================] - 313s 8s/step - loss: 1.2840 - accuracy: 0.3333 - val_loss: 1.0974 - val_accuracy: 0.34
62
Epoch 2/30
38/38 [==============================] - 312s 8s/step - loss: 1.1187 - accuracy: 0.3425 - val_loss: 1.1044 - val_accuracy: 0.34
62
Epoch 3/30
38/38 [==============================] - 310s 8s/step - loss: 0.9399 - accuracy: 0.4725 - val_loss: 0.7013 - val_accuracy: 0.58
65
Epoch 4/30
38/38 [==============================] - 309s 8s/step - loss: 0.6442 - accuracy: 0.6158 - val_loss: 0.5939 - val_accuracy: 0.65
38
Epoch 5/30
38/38 [==============================] - 307s 8s/step - loss: 0.5935 - accuracy: 0.6383 - val_loss: 0.5496 - val_accuracy: 0.65
38
Epoch 6/30
38/38 [==============================] - 306s 8s/step - loss: 0.6330 - accuracy: 0.6350 - val_loss: 0.6931 - val_accuracy: 0.59
62
Epoch 7/30
38/38 [==============================] - 305s 8s/step - loss: 0.6866 - accuracy: 0.6250 - val_loss: 0.6249 - val_accuracy: 0.63
46
Epoch 8/30
38/38 [==============================] - 305s 8s/step - loss: 0.5774 - accuracy: 0.6283 - val_loss: 0.5517 - val_accuracy: 0.61
54
Epoch 9/30
38/38 [==============================] - 304s 8s/step - loss: 0.5301 - accuracy: 0.6600 - val_loss: 0.5566 - val_accuracy: 0.67
31
Epoch 10/30
38/38 [==============================] - 305s 8s/step - loss: 0.5579 - accuracy: 0.6383 - val_loss: 0.5296 - val_accuracy: 0.61
54
Epoch 11/30
38/38 [==============================] - 312s 8s/step - loss: 0.5228 - accuracy: 0.6325 - val_loss: 0.5098 - val_accuracy: 0.74
04
```

We increased the epochs step by step. For 30 epochs only we achieved an accuracy of 63% with a loss of 0.51. Further increasing the epochs can enhance the accuracy.

For Vgg16 we ran for 10 Epochs and achieved an accuracy of 97.1%.

```
vgg_model = model_vgg.fit(train_set,validation_data=test_set,epochs=10)
```

```
Epoch 1/10
38/38 [==============================] - 79s 2s/step - loss: 0.5342 - accuracy: 0.7600 - val_loss: 0.4958 - val_accuracy: 0.750
0
Epoch 2/10
38/38 [==============================] - 80s 2s/step - loss: 0.2686 - accuracy: 0.8750 - val_loss: 0.1607 - val_accuracy: 0.942
3
Epoch 3/10
38/38 [==============================] - 82s 2s/step - loss: 0.1908 - accuracy: 0.9158 - val_loss: 0.1227 - val_accuracy: 0.961
5
Epoch 4/10
38/38 [==============================] - 82s 2s/step - loss: 0.1947 - accuracy: 0.9092 - val_loss: 0.1446 - val_accuracy: 0.951
9
Epoch 5/10
38/38 [==============================] - 80s 2s/step - loss: 0.1414 - accuracy: 0.9392 - val_loss: 0.0938 - val_accuracy: 0.980
8
Epoch 6/10
38/38 [==============================] - 81s 2s/step - loss: 0.1289 - accuracy: 0.9508 - val_loss: 0.0831 - val_accuracy: 0.971
2
Epoch 7/10
38/38 [==============================] - 81s 2s/step - loss: 0.1179 - accuracy: 0.9508 - val_loss: 0.0966 - val_accuracy: 0.961
5
Epoch 8/10
38/38 [==============================] - 82s 2s/step - loss: 0.1130 - accuracy: 0.9542 - val_loss: 0.0717 - val_accuracy: 0.990
4
Epoch 9/10
38/38 [==============================] - 84s 2s/step - loss: 0.1008 - accuracy: 0.9600 - val_loss: 0.1179 - val_accuracy: 0.932
7
Epoch 10/10
38/38 [==============================] - 85s 2s/step - loss: 0.1066 - accuracy: 0.9583 - val_loss: 0.0824 - val_accuracy: 0.971
2
```

For Resnet as well we ran the model for 10 epochs and achieved 100% accuracy which might be a result of overfitting.

```
res_model = model_res.fit(train_set,validation_data=test_set,epochs=10)
```

```
Epoch 1/10
38/38 [==============================] - 84s 2s/step - loss: 0.6093 - accuracy: 0.7342 - val_loss: 0.2338 - val_accuracy: 0.932
7
Epoch 2/10
38/38 [==============================] - 84s 2s/step - loss: 0.2555 - accuracy: 0.8833 - val_loss: 0.2325 - val_accuracy: 0.894
2
Epoch 3/10
38/38 [==============================] - 83s 2s/step - loss: 0.2104 - accuracy: 0.9117 - val_loss: 0.1467 - val_accuracy: 0.951
9
Epoch 4/10
38/38 [==============================] - 83s 2s/step - loss: 0.1723 - accuracy: 0.9258 - val_loss: 0.1405 - val_accuracy: 0.923
1
Epoch 5/10
38/38 [==============================] - 82s 2s/step - loss: 0.1452 - accuracy: 0.9467 - val_loss: 0.1042 - val_accuracy: 0.961
5
Epoch 6/10
38/38 [==============================] - 87s 2s/step - loss: 0.1138 - accuracy: 0.9583 - val_loss: 0.0893 - val_accuracy: 0.971
2
Epoch 7/10
38/38 [==============================] - 84s 2s/step - loss: 0.1291 - accuracy: 0.9467 - val_loss: 0.1828 - val_accuracy: 0.894
2
Epoch 8/10
38/38 [==============================] - 83s 2s/step - loss: 0.1259 - accuracy: 0.9550 - val_loss: 0.1080 - val_accuracy: 0.942
3
Epoch 9/10
38/38 [==============================] - 83s 2s/step - loss: 0.1285 - accuracy: 0.9433 - val_loss: 0.1058 - val_accuracy: 0.942
3
Epoch 10/10
38/38 [==============================] - 82s 2s/step - loss: 0.1022 - accuracy: 0.9642 - val_loss: 0.0388 - val_accuracy: 1.000
0
```

We prefer using the VGG16 based trained model as it has the best accuracy with less chances of overfitting.

Key Metrics for success in solving the problem under consideration

Metrics used for solving the current scenario:

Loss: Categorical cross-entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one. Formally, it is designed to quantify the difference between two probability distributions.

Accuracy: The accuracy of any machine learning model is defined as the closeness of a measured value to a true value. That means the closer the measured value to the true value, the better is the accuracy.

Our less complex sequential model has an accuracy of 63% while VGG16 and Resnet through transfer learning have 97.1% and 100% accuracy.

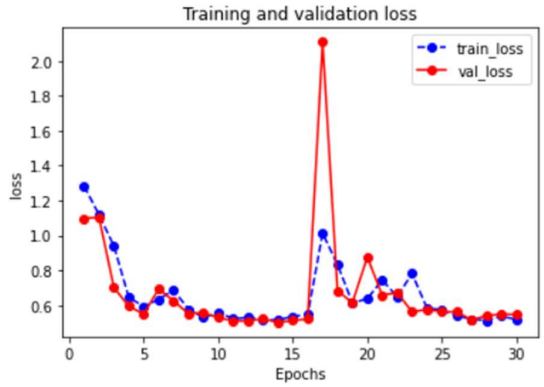
Visualizations

We plotted the model loss and accuracy graphs by creatin g a function

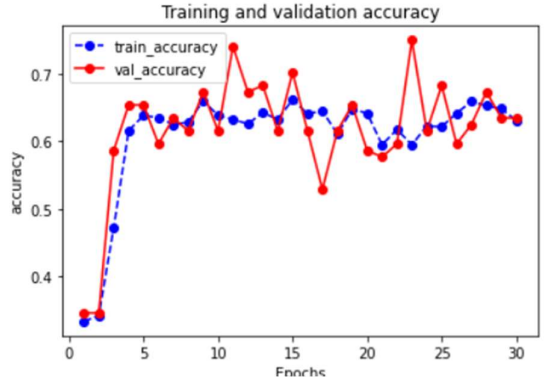
```
def plot_metric(model, metric):
    train_metrics = model.history[metric]
    val_metrics = model.history['val_'+metric]
    epochs = range(1, len(train_metrics) + 1)
    plt.plot(epochs, train_metrics, 'bo--')
    plt.plot(epochs, val_metrics, 'ro-')
    plt.title('Training and validation '+ metric)
    plt.xlabel("Epochs")
    plt.ylabel(metric)
    plt.legend(["train_"+metric, 'val_'+metric])
    plt.show()
```

Sequential Model

```
plot_metric(seq_model, 'loss')
```

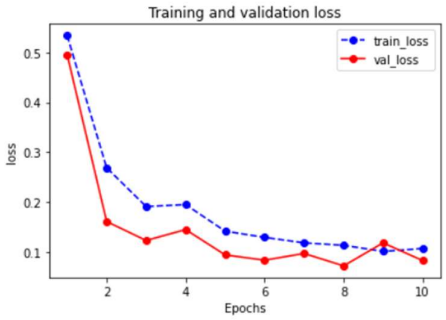



```
plot_metric(seq_model, 'accuracy')
```

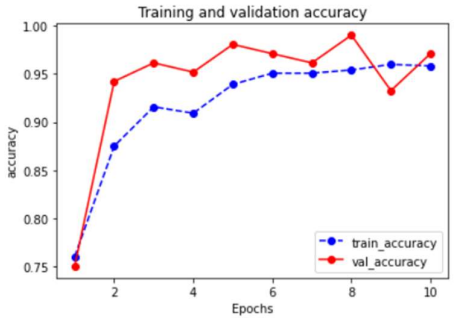


VGG16 Model

```
plot_metric(vgg_model, 'loss')
```

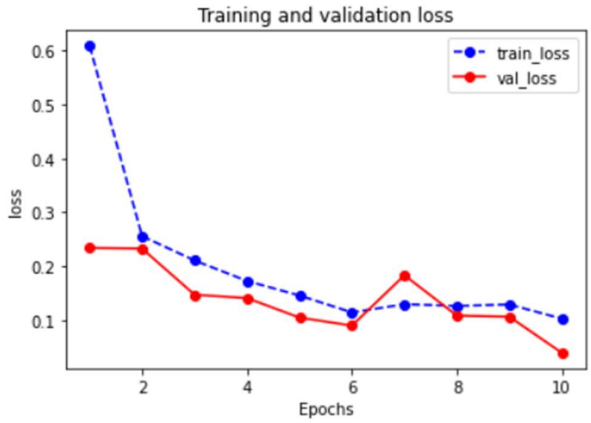



```
plot_metric(vgg_model, 'accuracy')
```

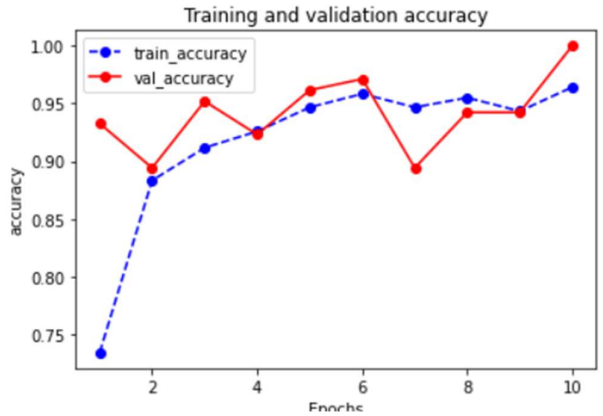


Resnet50

```
plot_metric(res_model, 'loss')
```




```
plot_metric(res_model, 'accuracy')
```



Interpretation of the Results

We created models for the image classification. We created a custom model using Sequential API and also used transfer learning techniques to use the pre-trained models and weights. With the less complex model, we still managed to get an accuracy of 63% with only 30 epochs while with pre-trained weights we got 97.1% accuracy in 10 epochs with VGG16 and 100% accuracy with Reanet50 in 10 epochs. There might be some overfitting in the later one so the best model will be VGG16.

CONCLUSION

Key Findings and Conclusions of the Study

Image classification depends on the ability of the system to extract as many features as possible from the image. More we extract more accurate is the prediction of the machine. We use convolution layers to extract features from the image and then pass it on to the max-pooling layers to reduce the dimensions. Then we use a network of fully connected layers to train our model on the extracted features. Here we use the technique to train our model to predict images between Saree, Jeans(men), and Trousers(men).

Learning Outcomes of the Study in respect of Data Science

As they say, human beings are visual beings. Visualizations hold the power to convey the information most effectively and efficiently. Whether it be analytical visualization or graphical visualization, both play an important role in the analysis. Visualization is a great method to easily sight the information and various relation between the features. Using visualization also gives a quick glimpse of the situation.

Images play a very crucial role in our daily life. With time as we have developed intelligent techniques to enhance the skill to work with images, it has become an important aspect to make computers work with images and make them understand and learn to create differentiation between them as we humans do.

In our case, we observe the best performance from the VGG16 supported model using the transfer learning technique.

Limitations of this work and Scope for Future Work

In the given dataset we observe that several features are available for us to train the model. Extracting features from the image and training the model is a complex and computationally expensive task. Also, deep learning requires a huge amount of data

to be trained for generalization. We use models trained data from imagenet but creating a custom model remains a time-consuming and cumbersome task.