Eluvio project report

In this project, I want to distinguish the car brands using an imagery dataset which includes the pictures of the brands. For this purpose, convolutional neural network (CNN) is chosen according to its satisfactory performance for image-based data classification. The first step is explanatory data analysis (EDA) to get some information about the given dataset.

EDA:

In [1]:

```
from keras.callbacks import ModelCheckpoint
from keras.preprocessing import image
from PIL import ImageFile
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from keras import layers
import io
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob
import os
import tensorflow as tf
from PIL import Image
import os.path
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dropout, Dense,
from keras.models import Sequential
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.layers import GaussianNoise
from livelossplot import PlotLossesKeras
from keras.callbacks import EarlyStopping
```

In [16]:

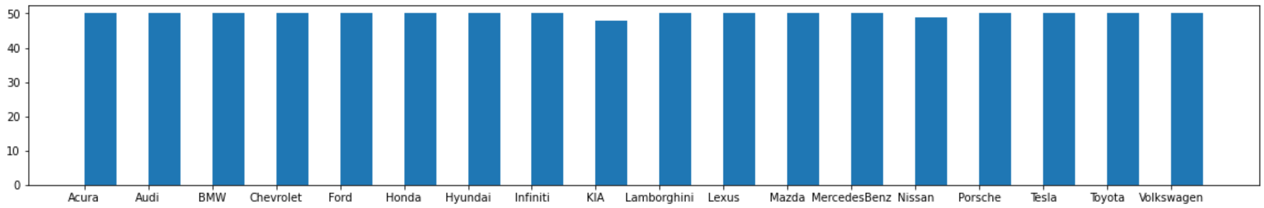
```
image_size=224
epoch_number=32

train_batchsize = 64
val_batchsize = 16
```

The below plot shows that the car logos dataset is a balanced dataset and we do not need to apply any oversampling or undersampling method on this dataset.

In [3]:

```
cars = glob('C:\\Users\\Negar Shakiba\\Desktop\\EluvioProject\\dataset_car_logo\\Train\
cars = [os.path.basename(item) for item in cars]
cars_photo_counts = {car:len(glob(os.path.join(os.path.dirname('C:\\Users\\Negar Shakib
                                               f"{car}/*"))) for car in cars}
car_names=list(cars_photo_counts.keys())
car_numbers=list(cars_photo_counts.values())
```

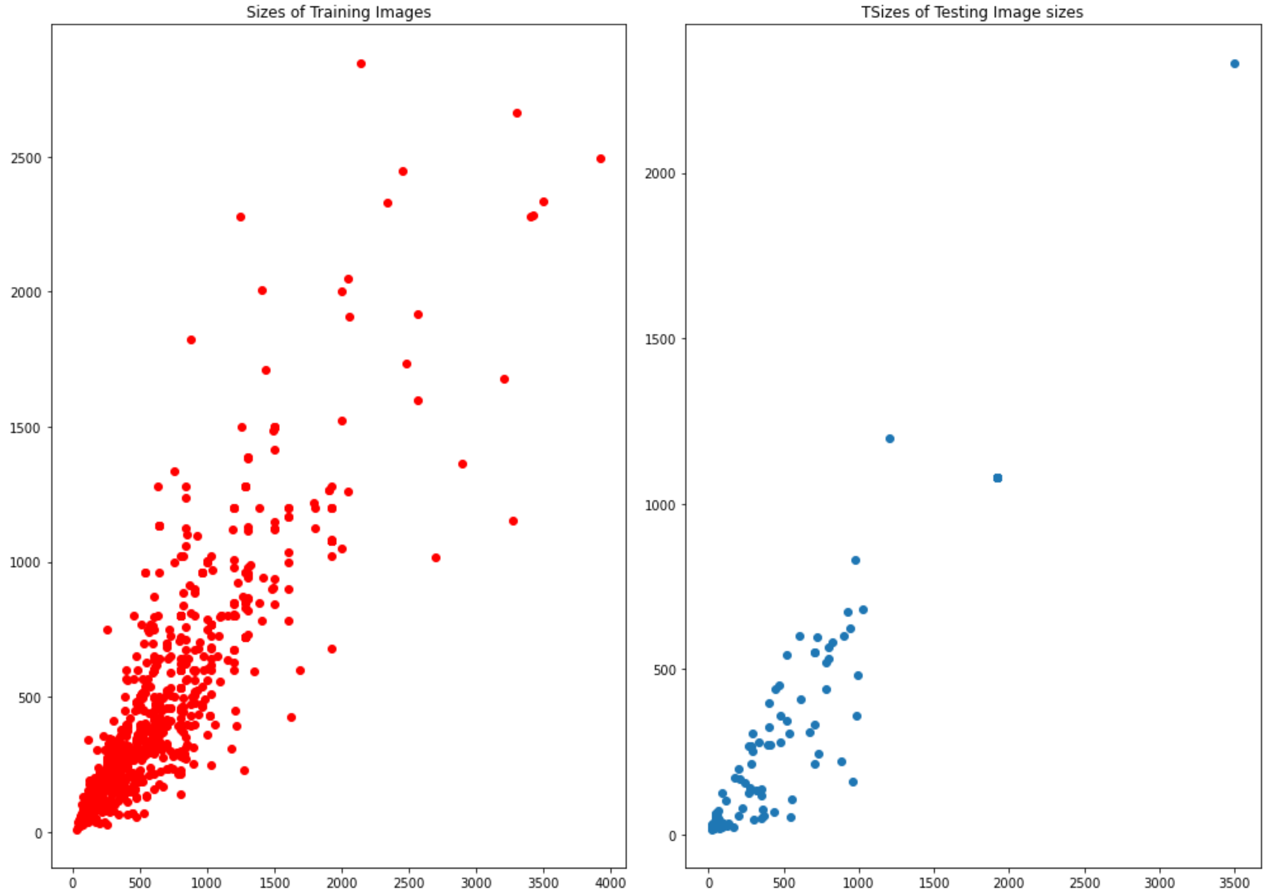
```
plt.figure(figsize=(20, 3))
plt.bar(range(len(cars_photo_counts)), car_numbers, tick_label=car_names, align = 'edge
plt.show()
```



Below, we draw a diagram which shows the sizes of the pictures. In this diagram each dot shows and image.

In [4]:

```
train_dir='C:\\Users\\Negar Shakiba\\Desktop\\EluvioProject\\dataset_car_logo\\Train'
test_dir='C:\\Users\\Negar Shakiba\\Desktop\\EluvioProject\\dataset_car_logo\\Test'

data_train = load_files(train_dir)
logo_files_train = np.array(data_train['filenames'])
data_test = load_files(test_dir)
logo_files_test = np.array(data_test['filenames'])

size_images_train = dict()
size_images_test = dict()

for path_image in logo_files_train:
    image = os.path.abspath(os.path.join(train_dir, path_image))
    with Image.open(image) as img:
        width, heigth = img.size
        size_images_train[path_image] = [width, heigth]
for path_image in logo_files_test:
    image = os.path.abspath(os.path.join(test_dir, path_image))
    with Image.open(image) as img:
        width, heigth = img.size
        size_images_test[path_image] = [width, heigth]
x_1,y_1 = zip(*size_images_train.values())
x_2,y_2 = zip(*size_images_test.values())

f, (ax1, ax2) = plt.subplots(1, 2,figsize=(14,10))
ax1.plot(x_1, y_1,'o',c='r')
ax1.set_title('Sizes of Training Images')

ax2.scatter(x_2, y_2, marker='o')
ax2.set_title('TSizes of Testing Image sizes')

plt.tight_layout()
plt.show()
```



For the second step, I applied a CNN with 9 layers to the given dataset to classify the datasets. This model generated only 30% accuracy which is not sufficient. Below the basic cnn model is shown:

Basic CNN with 9 layers:

In [5]:

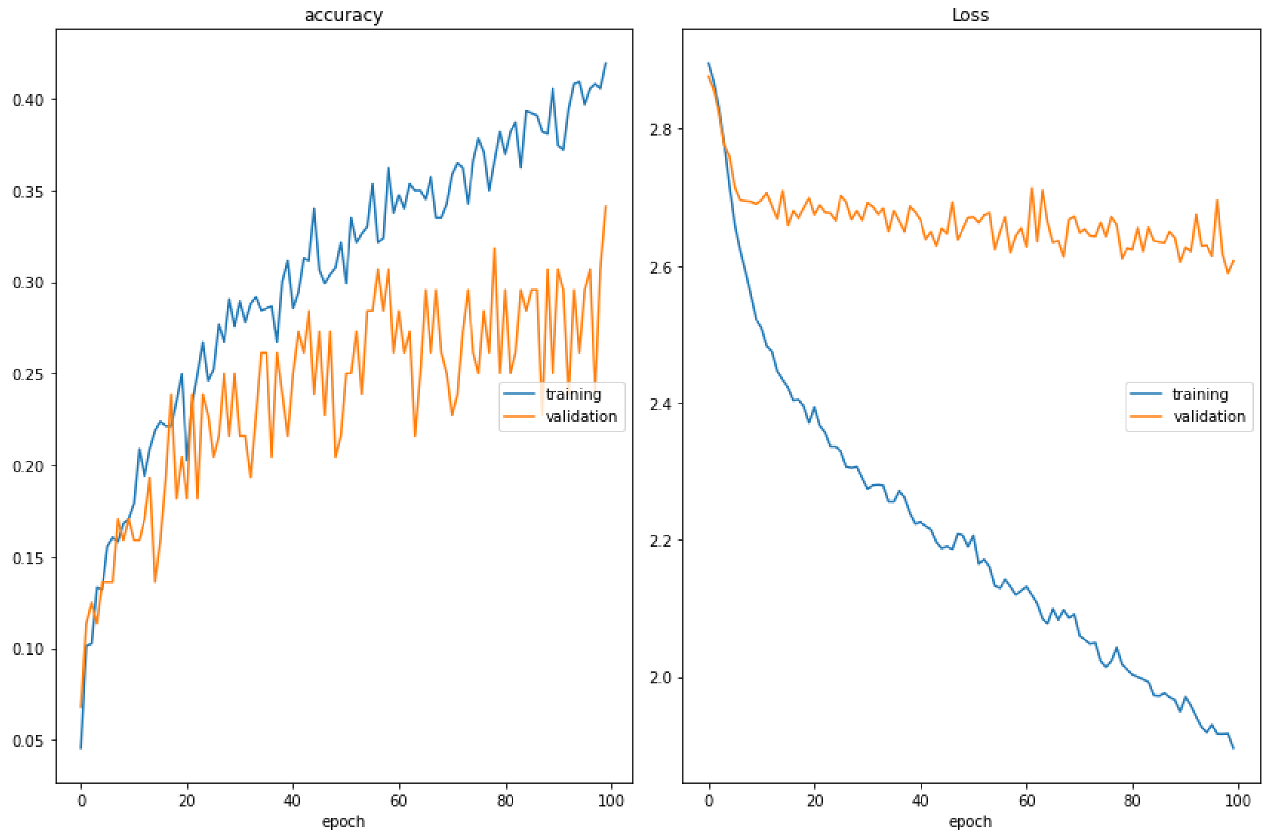
```
# Load the normalized images
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.1)
test_datagen = ImageDataGenerator(rescale=1./255)



# Data generator for training data,validation data, and test data
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='categorical',
        subset='training')

validation_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='categorical',
        subset='validation')

test_generator = test_datagen.flow_from_directory(
    test_dir,
```

```
    target_size=(image_size, image_size),
    batch_size=val_batchsize,
    class_mode='categorical')
```

```
Found 809 images belonging to 18 classes.
Found 88 images belonging to 18 classes.
Found 105 images belonging to 18 classes.
<tensorflow.python.keras.preprocessing.image.DirectoryIterator object at 0x000001C6AC529
7F0>
```

In [7]:

```
basic_model = Sequential()
basic_model.add(Conv2D (kernel_size = (2,2), filters = 32,
                        input_shape=(image_size, image_size, 3), activation='relu'))
basic_model.add(MaxPooling2D(pool_size=2))
basic_model.add(Conv2D(kernel_size = 2, filters = 64, activation='relu'))
basic_model.add(MaxPooling2D(pool_size=2))
basic_model.add(Conv2D(kernel_size = 2, filters = 128, activation='relu'))
basic_model.add(Dropout(0.2))
basic_model.add(MaxPooling2D(pool_size = 2))
basic_model.add(GlobalAveragePooling2D())
basic_model.add(Dense(18, activation = 'softmax'))

basic_model.summary()
basic_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

#monitor_val_acc = EarlyStopping(monitor = 'val_loss', patience = 5)
history =basic_model.fit(train_generator,
                         epochs=epoch_number,
                         verbose=0 ,
                         validation_data=validation_generator,
                         callbacks=[PlotLossesKeras()])
test_loss, test_acc = basic_model.evaluate_generator(test_generator, verbose = 0)
print('Test loss: {} Test Acc: {}'.format(test_loss, test_acc))
```




```
accuracy
	training             	 (min:    0.046, max:    0.419, cur:    0.419)
	validation           	 (min:    0.068, max:    0.341, cur:    0.341)
Loss
	training             	 (min:    1.896, max:    2.895, cur:    1.896)
	validation           	 (min:    2.589, max:    2.876, cur:    2.607)
WARNING:tensorflow:From <ipython-input-7-1e572ce728ad>:21: Model.evaluate_generator (fro
m tensorflow.python.keras.engine.training) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use Model.evaluate, which supports generators.
Test loss: 2.926781177520752 Test Acc: 0.18095238506793976
```

The above graphs shows an unrepresentative Train dataset. An unrepresentative training dataset means that the training dataset does not provide sufficient information to learn the features, relative to the validation dataset used to evaluate it. This may occur if the training dataset does not have sufficient data as compared to the validation dataset. This situation can be identified by a learning curve for training loss that shows improvement and similarly a learning curve for validation loss that shows improvement, but a large gap remains between both curves which can be seen in the above loss graph.

To increase the accuracy of the system, two different approach are presented:

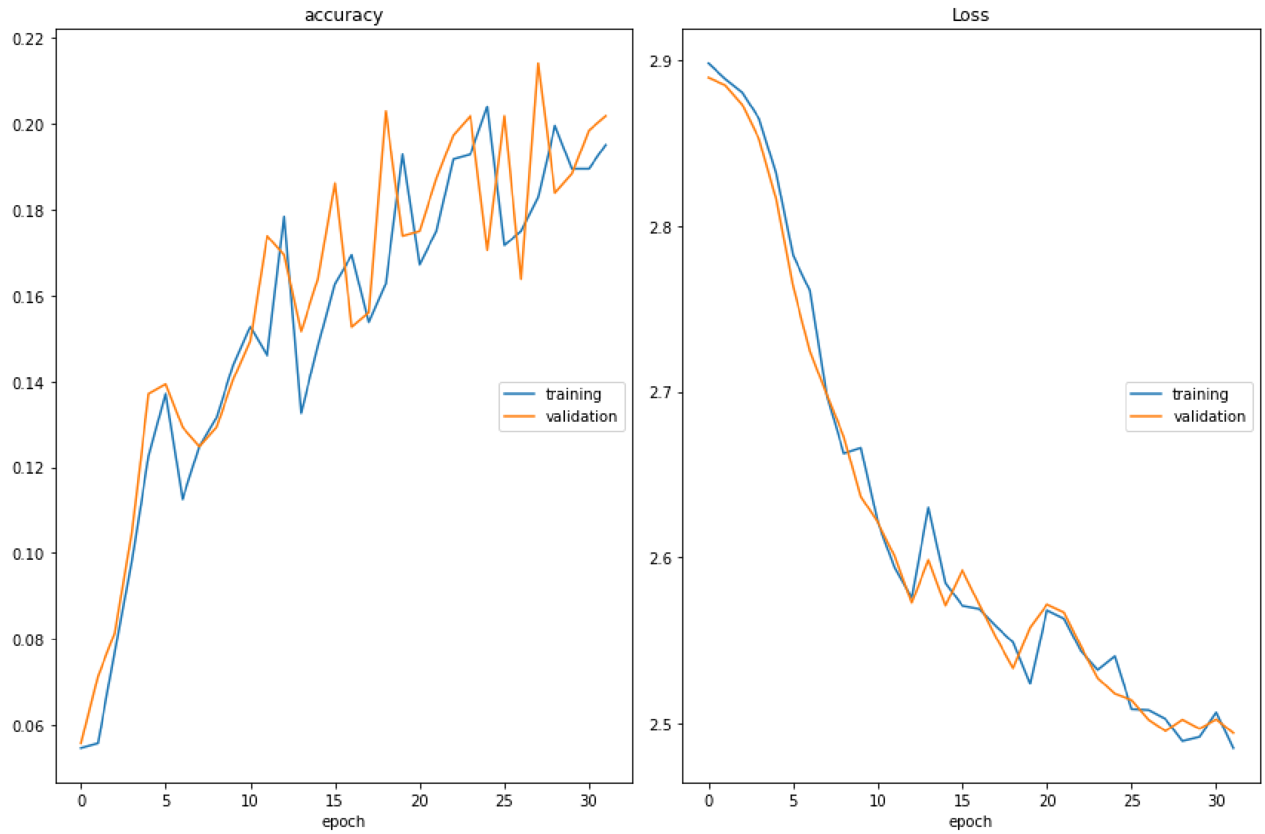
1. Data Augmentation
2. Adding Noises such as Gaussian Noise (white noise)

Data Augmentation:

In [18]:

```
# create the class object
datagen = ImageDataGenerator(rescale=1./255,
```

```
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
brightness_range=[0.4, 1.0],
fill_mode='nearest')


train_aug_generator = datagen.flow_from_directory(
        'C:\\Users\\Negar Shakiba\\Desktop\\EluvioProject\\dataset_car_logo\\Train',
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='categorical',
        subset='training')
validation_aug_generator = datagen.flow_from_directory(
        'C:\\Users\\Negar Shakiba\\Desktop\\EluvioProject\\dataset_car_logo\\Train',
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='categorical',
        subset='training')

aug_model = Sequential()
aug_model.add(Conv2D (kernel_size = (2,2), filters = 32,
                      input_shape=(image_size, image_size, 3), activation='relu'))
aug_model.add(MaxPooling2D(pool_size=2))
aug_model.add(Conv2D(kernel_size = 2, filters = 64, activation='relu'))
aug_model.add(MaxPooling2D(pool_size=2))
aug_model.add(Conv2D(kernel_size = 2, filters = 128, activation='relu'))
aug_model.add(Dropout(0.2))
aug_model.add(MaxPooling2D(pool_size = 2))
aug_model.add(GlobalAveragePooling2D())
aug_model.add(Dense(18, activation = 'softmax'))

aug_model.summary()
aug_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
history =aug_model.fit(train_aug_generator,
                       epochs=epoch_number,
                       verbose=1,
                       validation_data=validation_aug_generator,
                         callbacks=[PlotLossesKeras()])
test_loss, test_acc = aug_model.evaluate_generator(test_generator, verbose = 0)
print('Test loss: {} Test Acc: {}'.format(test_loss, test_acc))
```




```
accuracy
	training          	 (min:    0.055, max:    0.204, cur:    0.195)
	validation        	 (min:    0.056, max:    0.214, cur:    0.202)
Loss
	training          	 (min:    2.485, max:    2.898, cur:    2.485)
	validation        	 (min:    2.494, max:    2.890, cur:    2.494)
15/15 [==============================] - 74s 5s/step - loss: 2.4847 - accuracy: 0.1951 - val_loss: 2.4938 - val_accuracy: 0.2018
Test loss: 2.907989263534546 Test Acc: 0.13333334028720856
```

The above graphs show that the model is overfitting and probably that is the reason that the accuracy is not as good as other models.

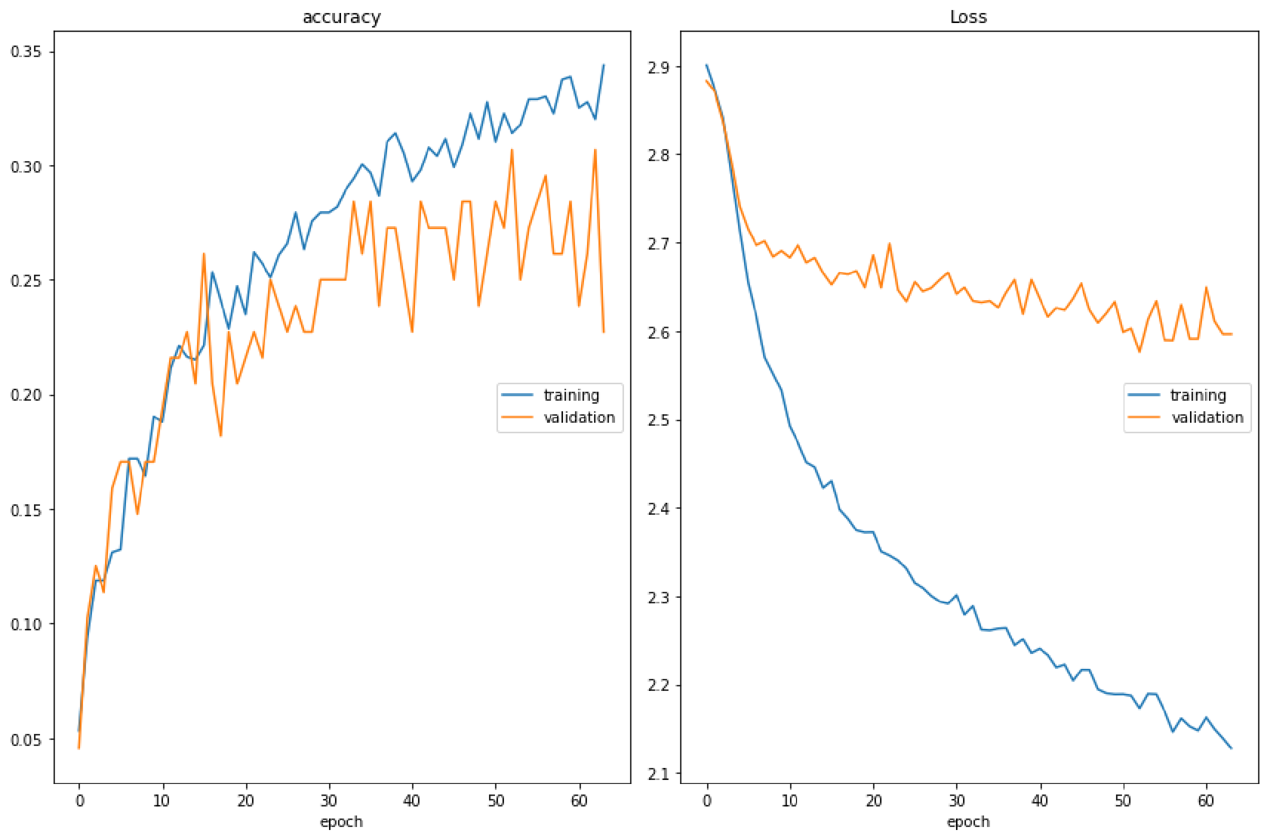
Now, we want to add a layer of noise to the system to remove the overfitting problem.

Adding Gaussian Noise Layer

In [11]:

```
noisy_model = Sequential()
noisy_model.add(Conv2D (kernel_size = (2,2), filters = 32,
                        input_shape=(image_size, image_size, 3), activation='relu'))
noisy_model.add(GaussianNoise(0.01, input_shape=(2,)))
noisy_model.add(MaxPooling2D(pool_size=2))
noisy_model.add(Conv2D(kernel_size = 2, filters = 64, activation='relu'))
noisy_model.add(MaxPooling2D(pool_size=2))
noisy_model.add(Conv2D(kernel_size = 2, filters = 128, activation='relu'))
noisy_model.add(Dropout(0.2))
noisy_model.add(MaxPooling2D(pool_size = 2))
noisy_model.add(GlobalAveragePooling2D())
noisy_model.add(Dense(18, activation = 'softmax'))

noisy_model.summary()
noisy_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
```

```
history =noisy_model.fit(train_generator,
                         epochs=epoch_number,
                         verbose=1 ,
                         validation_data=validation_generator,
                         callbacks=[PlotLossesKeras()])
test_loss, test_acc = noisy_model.evaluate_generator(test_generator, verbose = 0)
print('Test loss: {} Test Acc: {}'.format(test_loss, test_acc))
```




```
accuracy
	training          	 (min:    0.053, max:    0.344, cur:    0.344)
	validation        	 (min:    0.045, max:    0.307, cur:    0.227)
Loss
	training          	 (min:    2.128, max:    2.901, cur:    2.128)
	validation        	 (min:    2.576, max:    2.883, cur:    2.597)
13/13 [==============================] - 52s 4s/step - loss: 2.1278 - accuracy: 0.3436 - 
val_loss: 2.5965 - val_accuracy: 0.2273
Test loss: 2.8633577823638916 Test Acc: 0.190476194024086
```

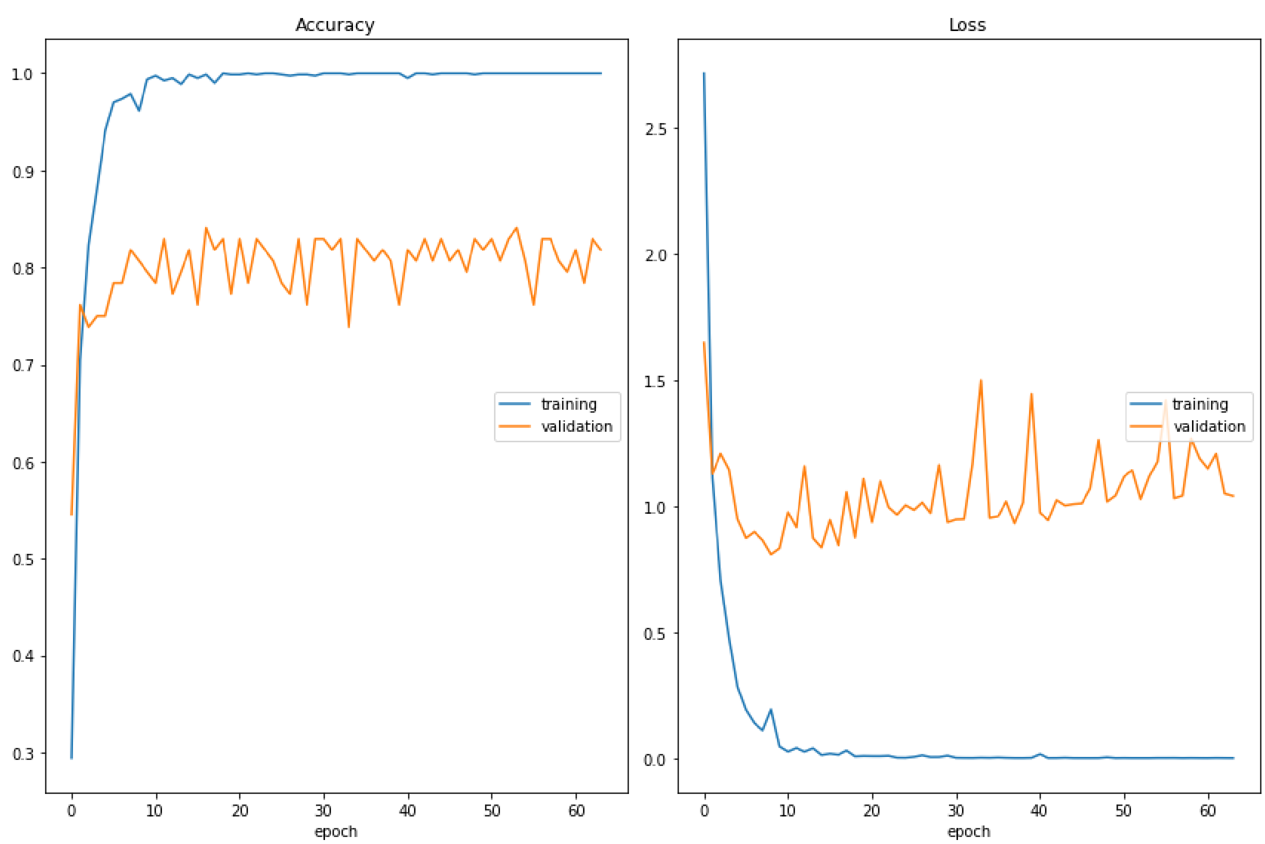
Considering the data augmentation and adding gaussian noise, it can be concluded that another efficient methodology is needed. In other words, we need to use other datasets and architectures to help our models in training step . Transfer learning is the methodology which I used in this project to generate better results.

To perform the transfer learning method firstly, I used the imagenet dataset and VGG16 neural network. In other words, the weights from trained neural network using imagenet, are transferred to a new neural network (a CNN) and then I added 4 new layers for the purpose of fine tuning to this new CNN. These new layers which are trainable, are added to match the features from the imagenet dataset to car brands dataset.

VGG16 and ImageNet

In [14]:

```
vgg_layer =VGG16(weights='imagenet', include_top=False,
                 input_shape=(image_size, image_size, 3))

for layer in vgg_layer.layers[:]:
    layer.trainable = False


vgg_model = Sequential()
# Add the vgg convolutional base model
vgg_model.add(vgg_layer)
# Add new layers
vgg_model.add(Flatten())
vgg_model.add(Dense(1024, activation='relu'))
vgg_model.add(Dropout(0.5))
vgg_model.add(Dense(18, activation='softmax'))

# Show a summary of the model and layers parameters
vgg_model.summary()
# Configure the model for training
vgg_model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])

# Train the model
history = vgg_model.fit(
          train_generator,
          validation_data = validation_generator,
          steps_per_epoch=train_generator.samples/train_generator.batch_size,
          validation_steps = validation_generator.samples / validation_generator.batch_
          epochs=epoch_number, verbose=1,
          callbacks=[PlotLossesKeras()])
test_loss, test_acc = vgg_model.evaluate_generator(test_generator, verbose = 0)
print('Test loss: {} Test Acc: {}'.format(test_loss, test_acc))
```



```
Accuracy
	training             (min:    0.294, max:    1.000, cur:    1.000)
	validation           (min:    0.545, max:    0.841, cur:    0.818)
Loss
	training             (min:    0.000, max:    2.716, cur:    0.000)
	validation           (min:    0.809, max:    1.647, cur:    1.040)
13/12 [==============================] - 147s 11s/step - loss: 2.5073e-05 - acc: 1.0000
- val_loss: 1.0400 - val_acc: 0.8182
Test loss: 2.9830410480499268 Test Acc: 0.4952380955219269
```

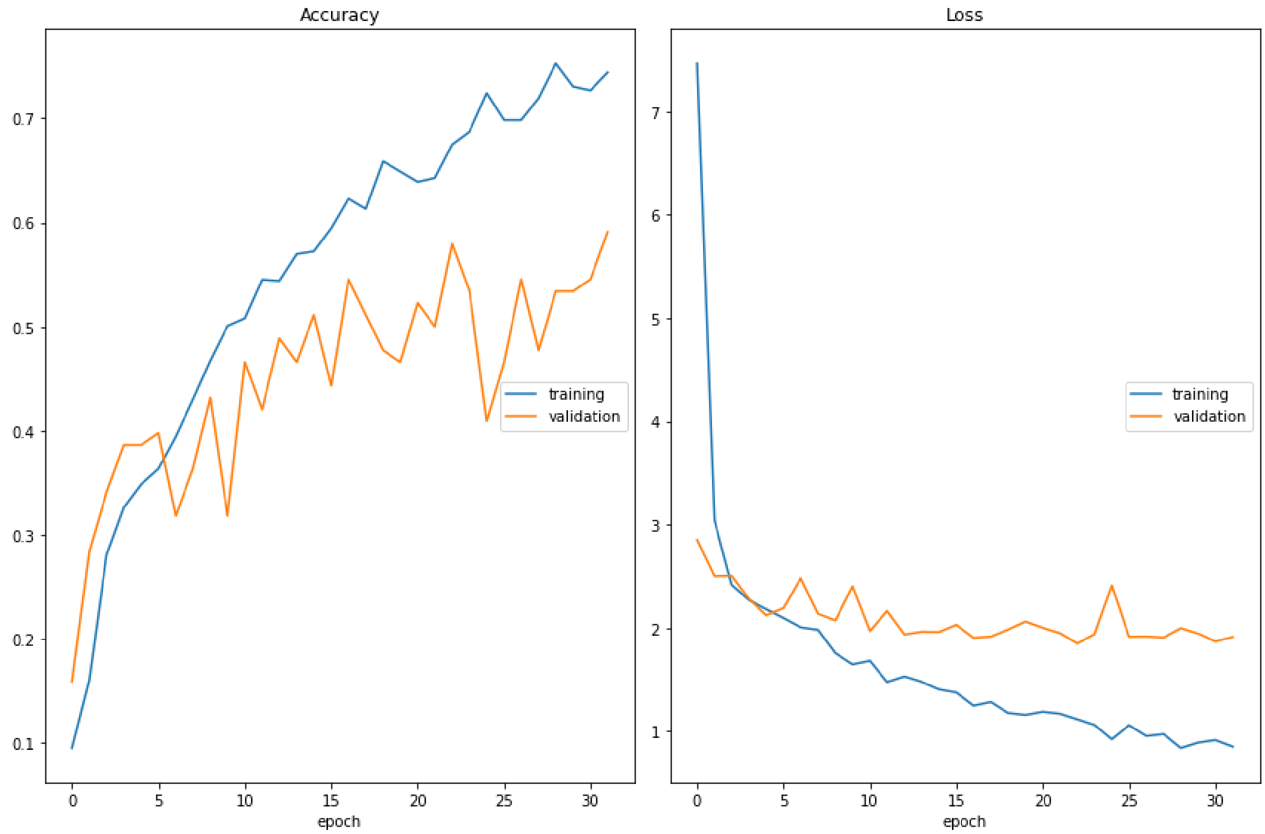
The above graphs show the improvement in validation and testing accuracy.

Below I used another architecture for CNN which is called ResNet50 and used its pretrained features based on ImageNet dataset.

ResNet50 and ImageNet

In [17]:

```
res_layer =ResNet50(weights='imagenet', include_top=False,
                    input_shape=(image_size, image_size, 3))

for layer in res_layer.layers[:]:
    layer.trainable = False

for layer in res_layer.layers:
    print(layer, layer.trainable)



ResNet50_model = Sequential()
# Add the ResNet50 convolutional base model
ResNet50_model.add(res_layer)
# Add new layers
ResNet50_model.add(Flatten())
ResNet50_model.add(Dense(1024, activation='relu'))
ResNet50_model.add(Dropout(0.5))
ResNet50_model.add(Dense(18, activation='softmax'))

# Show a summary of the model and layers parameters
ResNet50_model.summary()
# Configure the model for training
ResNet50_model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])

# Train the model
history = ResNet50_model.fit(
          train_generator,
          validation_data = validation_generator,
          steps_per_epoch= train_generator.samples/train_generator.batch_size,
          validation_steps = validation_generator.samples / validation_generator.batch_
          epochs=epoch_number, verbose=1,
          callbacks=[PlotLossesKeras()])
test_loss, test_acc = ResNet50_model.evaluate_generator(test_generator, verbose = 0)
print('Test loss: {} Test Acc: {}'.format(test_loss, test_acc))
```




```
Accuracy
	training           	 (min:    0.095, max:    0.753, cur:    0.744)
	validation         	 (min:    0.159, max:    0.591, cur:    0.591)
Loss
	training           	 (min:    0.836, max:    7.465, cur:    0.850)
	validation         	 (min:    1.850, max:    2.852, cur:    1.911)
13/12 [==============================] - 88s 7s/step - loss: 0.8499 - acc: 0.7441 - val_
loss: 1.9110 - val_acc: 0.5909
Test loss: 3.251791477203369 Test Acc: 0.21904762089252472
```

Results

Methods	Basic CNN	Using Augmentation	Noise Added	Transfer Learning VGG16 with ImageNet	Transfer Learning ResNet50 with ImageNet
Accuracy	18%	13%	19%	49.5%	21%

Bonus question:

Of course there is a trade-off between the time of running a deeplearning model and its accuracy. The bigger the data becomes, the more important challenge shows itself and we always need to be careful about it. Regarding gathering the dataset, I would suggest to balance the data gathering system in a way that each class has an equal portion of the whole data. Although there are solutions for imbalanced datasets but they usually sacrify some precision. Different data augmentation can have different influences on the dataset. In case,our system does not work pretty well, we can combine some of the augmentaion models to remove probable overfitting problem. Because current existing datasets include very straight and high quality images, which is dissimilar to real world taken pictures, the number of given pictures should be sufficient and techniques like blaring or change the brightness of the pictures for training purposes can help us generating more accurate results.

In [ ]: