In [4]:

```
capstone_dir = '/home/ec2-user/SageMaker/capstone'
```

Data.zip contains category wise csv, which internally contains images URL

```
Let's first UNPACK the data file and create a directories structure for ou
r images
```

DIRECTORY STRUCTURE

- ./data/
  - cars/img_N.jpg
  - motorcycle/img_N.jpg
  - mobile/img_N.jpg
  - books/img_N.jpg
  - furniture/img_N.jpg

In [83]:

```
!unzip -o data.zip
```

```
Archive:  data.zip
   creating: data/
   creating: data/motorcycle/
  inflating: data/motorcycle/data.csv
   creating: data/books/
  inflating: data/books/data.csv
   creating: data/furniture/
  inflating: data/furniture/data.csv
   creating: data/mobile/
  inflating: data/mobile/data.csv
   creating: data/cars/
  inflating: data/cars/data.csv
```

In [90]:

```
# import the necessary packages
from skimage.io import imread

def url_to_image(url):
    image = imread(url)
    return image
```

Now lets download the images given in the CSV class wise

```
Convert those images into 112x112 pixel and in RGB color mode and save it
 in given class path
```

In [91]:

```
from PIL import Image
import urllib.request
from skimage.transform import resize

def img_url_to_pil_image(img_url, img_pwd):

    urllib.request.urlretrieve(img_url, img_pwd)
    pil_im = Image.open(img_pwd)
    pil_im = pil_im.resize((112,112))
    pil_im.save(img_pwd)
    return pil_im
```

In [86]:

```
import pandas
import numpy as np
from pathlib import Path

def importImages(path):
    filename = path + '/data.csv'
    data = pandas.read_csv(filename,names=['images'])
    img_path = path + "/"
    disk_dir = Path(img_path)
    disk_dir.mkdir(parents=True, exist_ok=True)
    counter = 0
    for index, row in data.iterrows():
        try:
            img_pwd = img_path + '/img_%d.jpg' % ( counter )
            img = img_url_to_pil_image(row['images'],img_pwd)
            counter +=1
        except:
            pass
```

In [87]:

```
#importImages("data/books")
#importImages("data/cars")
#importImages("data/mobile")
#importImages("data/motorcycle")
#importImages("data/furniture")
```

In [88]:

```
# To download images faster by running in parallel
from threading import Thread
books = Thread(target=importImages, args=["data/books"]).start()
cars = Thread(target=importImages, args=["data/cars"]).start()
mobile = Thread(target=importImages, args=["data/mobile"]).start()
motorcycle = Thread(target=importImages, args=["data/motorcycle"]).start()
furniture = Thread(target=importImages, args=["data/furniture"]).start()
```

Creating the RecordIO list files

Right, now that we have all the images in their respective directories, one per class, it is time to create the RecordIO file. RecordIO is a optimized file format that will feed our images to the Neural Network during training.

We will split the dataset into training (70%) and testing (30%). To do that, we'll run a python script (im2rec), which is the best tool for this job.

In [5]:

```
# Here we will search for the python script im2rec
import sys,os

suffix='/mxnet/tools/im2rec.py'
im2rec = list(filter( (lambda x: os.path.isfile(x + suffix )), sys.path))[0] + s
uffix
%env IM2REC=$im2rec
%env DATA_DIR=$capstone_dir/data
```

```
env: IM2REC=/home/ec2-user/anaconda3/envs/amazonei_mxnet_p36/lib/pyt
hon3.6/site-packages/mxnet/tools/im2rec.py
env: DATA_DIR=/home/ec2-user/SageMaker/capstone/data
```

In [7]:

```
%%bash

# Ok. Here, im2rec will read all the images and create two .lst files, one for t
raining and other for validation
# this files will then be used for creating the RecordIO files

cd $DATA_DIR
python $IM2REC --list --recursive --test-ratio=0.3 --train-ratio=0.7 data ./
ls *.lst
```

```
books 0
cars 1
furniture 2
mobile 3
motorcycle 4
data_test.lst
data_train.lst
```

Then, using the .lst files, let's create both RecordIO files (train and test)

In [8]:

```
%%bash
#jupyter notebook --NotebookApp.iopub_data_rate_limit=2000000.0
cd $DATA_DIR
python $IM2REC --num-thread=16 --pass-through data_train.lst .
python $IM2REC --num-thread=16 --pass-through data_test.lst .
ls *.rec
```

```
Creating .rec file from /home/ec2-user/SageMaker/capstone/data/data_
train.lst in /home/ec2-user/SageMaker/capstone/data
time: 0.022768259048461914  count: 0
time: 0.06211709976196289  count: 1000
time: 0.029352426528930664  count: 2000
time: 0.031106233596801758  count: 3000
time: 0.03186511993408203  count: 4000
time: 0.03950929641723633  count: 5000
time: 0.04929399490356445  count: 6000
time: 0.04783749580383301  count: 7000
time: 0.1012272834777832  count: 8000
time: 0.035082340240478516  count: 9000
time: 0.041769981384277344  count: 10000
time: 0.03869986534118652  count: 11000
time: 0.032672882080078125  count: 12000
time: 0.0313572883605957  count: 13000
time: 0.03845787048339844  count: 14000
time: 0.10934972763061523  count: 15000
time: 0.03928995132446289  count: 16000
time: 0.04845857620239258  count: 17000
time: 0.046961307525634766  count: 18000
time: 0.03354835510253906  count: 19000
time: 0.03111124038696289  count: 20000
time: 0.03715991973876953  count: 21000
time: 0.10791778564453125  count: 22000
time: 0.050415992736816406  count: 23000
time: 0.02806711196899414  count: 24000
time: 0.03856658935546875  count: 25000
time: 0.05252981185913086  count: 26000
time: 0.09858202934265137  count: 27000
time: 0.04912519454956055  count: 28000
time: 0.0484464168548584  count: 29000
time: 0.032926082611083984  count: 30000
time: 0.03611135482788086  count: 31000
time: 0.04447627067565918  count: 32000
time: 0.04082608222961426  count: 33000
time: 0.09200215339660645  count: 34000
time: 0.04424643516540527  count: 35000
time: 0.03304004669189453  count: 36000
time: 0.036547183990478516  count: 37000
time: 0.04447317123413086  count: 38000
time: 0.03732800483703613  count: 39000
time: 0.035239458084106445  count: 40000
time: 0.044387102127075195  count: 41000
time: 0.11842012405395508  count: 42000
time: 0.03775620460510254  count: 43000
time: 0.04204702377319336  count: 44000
time: 0.04280519485473633  count: 45000
time: 0.05238747596740723  count: 46000
time: 0.10181403160095215  count: 47000
time: 0.03858757019042969  count: 48000
time: 0.03208112716674805  count: 49000
time: 0.03276538848876953  count: 50000
time: 0.04111146926879883  count: 51000
time: 0.02914595603942871  count: 52000
time: 0.04794812202453613  count: 53000
time: 0.03353452682495117  count: 54000
time: 0.10127878189086914  count: 55000
time: 0.03548383712768555  count: 56000
time: 0.04704165458679199  count: 57000
time: 0.025027751922607422  count: 58000
```

```
time: 0.019660472869873047  count: 59000
time: 0.017006397247314453  count: 60000
Creating .rec file from /home/ec2-user/SageMaker/capstone/data/data_
test.lst in /home/ec2-user/SageMaker/capstone/data
time: 0.008723258972167969  count: 0
time: 0.08196115493774414  count: 1000
time: 0.040810585021972656  count: 2000
time: 0.03532266616821289  count: 3000
time: 0.1103365421295166  count: 4000
time: 0.04393768310546875  count: 5000
time: 0.04094362258911133  count: 6000
time: 0.06603217124938965  count: 7000
time: 0.04163408279418945  count: 8000
time: 0.03358793258666992  count: 9000
time: 0.11974549293518066  count: 10000
time: 0.0437619686126709  count: 11000
time: 0.035685062408447266  count: 12000
time: 0.03744363784790039  count: 13000
time: 0.043944597244262695  count: 14000
time: 0.050086021423339844  count: 15000
time: 0.10187172889709473  count: 16000
time: 0.04066872596740723  count: 17000
time: 0.043926239013671875  count: 18000
time: 0.032441139221191406  count: 19000
time: 0.04982447624206543  count: 20000
time: 0.04440712928771973  count: 21000
time: 0.03332853317260742  count: 22000
time: 0.033257246017456055  count: 23000
time: 0.018323659896850586  count: 24000
time: 0.017594099044799805  count: 25000
time: 0.016998291015625  count: 26000
data_test.rec
data_train.rec
```

Great now upload *.rec files to S3

In [10]:

```
import boto3
import sagemaker

# Get the current Sagemaker session
sagemaker_session = sagemaker.Session()

bucket=sagemaker_session.default_bucket()
```

In [11]:

```
train_path = sagemaker_session.upload_data(path=capstone_dir + '/data/data_trai
n.rec', key_prefix='top-category-olx/train')
test_path = sagemaker_session.upload_data(path=capstone_dir + '/data/data_test.r
ec', key_prefix='top-category-olx/test')
```

Environment SetUp

1. Role attached to sagemaker instance that will access to data present in s3
2. S3 bucket in which training and model data to be stored
3. The Amazon sagemaker image classification docker image

In [12]:

```
%%time
import boto3
import re
import os
import time

from time import gmtime, strftime
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri


# 1. Obtaining the role you already configured for Sagemaker when you setup
# your Instance notebook (https://docs.aws.amazon.com/sagemaker/latest/dg/gs-set
up-working-env.html)
role = get_execution_role()

# 3. Select the correct Docker image with the Image Classification algorithm
training_image = get_image_uri(boto3.Session().region_name, 'image-classificatio
n','latest')
print(training_image)
```

```
475088953585.dkr.ecr.ap-southeast-1.amazonaws.com/image-classificati
on:latest
CPU times: user 81.3 ms, sys: 0 ns, total: 81.3 ms
Wall time: 134 ms
```

Hyper-parameters tuning. These parameters will determine how model will be trained and, consequently, how trained model will behave.

In [18]:

```
# The algorithm supports multiple network depth (number of layers). They are 18,
34, 50, 101, 152 and 200
# For this training, we will use 152 layers
num_layers = 34
# we need to specify the input image shape for the training data
image_shape = "3,112,112"
# we also need to specify the number of training samples in the training set
num_training_samples = 60964
# specify the number of output classes
num_classes = 5
# batch size for training
mini_batch_size = 1024
# number of epochs
epochs = 7
# learning rate
learning_rate = 0.05
# Since we are using transfer learning, we set use_pretrained_model to 1 so that
weights can be
# initialized with pre-trained weights
use_pretrained_model = 1
# Training algorithm/optimizer. Default is SGD
optimizer = 'sgd'
```

After defining all the parameters, now lets start start job in SageMaker

In [19]:

```
dataset_prefix='top-category-olx'
# create unique job name
job_name_prefix = 'top-category-olx'
timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
job_name = job_name_prefix + timestamp

training_params = {}

# Here we set the reference for the Image Classification Docker image, stored on
ECR (https://aws.amazon.com/pt/ecr/)
training_params["AlgorithmSpecification"] = {
    "TrainingImage": training_image,
    "TrainingInputMode": "File"
}

# The IAM role with all the permissions given to Sagemaker
training_params["RoleArn"] = role

# Here Sagemaker will store the final trained model
training_params["OutputDataConfig"] = {
    "S3OutputPath": 's3://{}/{}/output'.format(bucket, job_name_prefix)
}

# This is the config of the instance that will execute the training
training_params["ResourceConfig"] = {
    "InstanceCount": 1,
    "InstanceType": "ml.p3.8xlarge",
    "VolumeSizeInGB": 100
}

# The job name. You'll see this name in the Jobs section of the Sagemaker's cons
ole
training_params["TrainingJobName"] = job_name

# Here you will configure the hyperparameters used for training your model.
training_params["HyperParameters"] = {
    "image_shape": image_shape,
    "num_layers": str(num_layers),
    "num_training_samples": str(num_training_samples),
    "num_classes": str(num_classes),
    "mini_batch_size": str(mini_batch_size),
    "epochs": str(epochs),
    "learning_rate": str(learning_rate),
    #"use_pretrained_model": str(use_pretrained_model),
    "use_pretrained_model": str(use_pretrained_model),
    "optimizer": optimizer
}

# Training timeout
training_params["StoppingCondition"] = {
    "MaxRuntimeInSeconds": 360000
}

# The algorithm currently only supports fullyreplicated model (where data is cop
ied onto each machine)
training_params["InputDataConfig"] = []

# Please notice that we're using application/x-recordio for both
# training and validation datasets, given our dataset is formated in RecordIO
```

```
# Here we set training dataset
# Training data should be inside a subdirectory called "train"
training_params["InputDataConfig"].append({
    "ChannelName": "train",
    "DataSource": {
        "S3DataSource": {
            "S3DataType": "S3Prefix",
            "S3Uri": 's3://{}/{}/train/'.format(bucket, dataset_prefix),
            "S3DataDistributionType": "FullyReplicated"
        }
    },
    "ContentType": "application/x-recordio",
    "CompressionType": "None"
})

# Here we set validation dataset
# Validation data should be inside a subdirectory called "validation"
training_params["InputDataConfig"].append({
    "ChannelName": "validation",
    "DataSource": {
        "S3DataSource": {
            "S3DataType": "S3Prefix",
            "S3Uri": 's3://{}/{}/test/'.format(bucket, dataset_prefix),
            "S3DataDistributionType": "FullyReplicated"
        }
    },
    "ContentType": "application/x-recordio",
    "CompressionType": "None"
})

print('Training job name: {}'.format(job_name))
print('\nInput Data Location: {}'.format(training_params['InputDataConfig'][0][
'DataSource']['S3DataSource']))
```

```
Training job name: top-category-olx-2020-03-23-13-54-38

Input Data Location: {'S3DataType': 'S3Prefix', 'S3Uri': 's3://sagem
aker-ap-southeast-1-361886290651/top-category-olx/train/', 'S3DataDi
stributionType': 'FullyReplicated'}
```

Model creation

Steps in which i will create my model:

1. First I will submit job for sagemaker to train model using dataset on S3
2. Pack the job output into the model
3. Create an Endpoint Configuration, which is the metadata used by Sagemaker to deploy model
4. Finally deploy model using the Endpoint Configuration

In [20]:

```
sagemaker = boto3.client(service_name='sagemaker')
```

Submit a job , to train your model

In [21]:

```
# create the Amazon SageMaker training job

sagemaker.create_training_job(**training_params)

# confirm that the training job has started
status = sagemaker.describe_training_job(TrainingJobName=job_name)['TrainingJobS
tatus']
print('Training job current status: {}'.format(status))

try:
    # wait for the job to finish and report the ending status
    sagemaker.get_waiter('training_job_completed_or_stopped').wait(TrainingJobNa
me=job_name)
    training_info = sagemaker.describe_training_job(TrainingJobName=job_name)
    status = training_info['TrainingJobStatus']
    print("Training job ended with status: " + status)
except:
    print('Training failed to start')
     # if exception is raised, that means it has failed
    message = sagemaker.describe_training_job(TrainingJobName=job_name)['Failure
Reason']
    print('Training failed with the following error: {}'.format(message))
```

```
Training job current status: InProgress
Training job ended with status: Completed
```

In [103]:

```
print(training_info['TrainingJobStatus'])
```

```
Completed
```

Now it's time convert job output into the model

Just make sure if you are running this for first time, set use_pretrained_model=False

In [22]:

```
%%time
import boto3
from time import gmtime, strftime

use_pretrained_model=False

model_name="top-category-olx" + time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime
())
print(model_name)
if use_pretrained_model:
    prefix="top-category-olx/model/model.tar.gz"
    model_data="s3://{}/{}".format(bucket, prefix)
    s3 = boto3.client('s3')
    resp = s3.list_objects(Bucket=bucket, Prefix=prefix)
else:
    info = sagemaker.describe_training_job(TrainingJobName=job_name)
    model_data = info['ModelArtifacts']['S3ModelArtifacts']
    print(model_data)

primary_container = {
    'Image': training_image,
    'ModelDataUrl': model_data,
}

try:
    create_model_response = sagemaker.create_model(
        ModelName = model_name,
        ExecutionRoleArn = role,
        PrimaryContainer = primary_container)
    print(create_model_response['ModelArn'])
except Exception as e:
    print(e)
```

```
top-category-olx-2020-03-23-18-03-14
s3://sagemaker-ap-southeast-1-361886290651/top-category-olx/output/t
op-category-olx-2020-03-23-13-54-38/output/model.tar.gz
arn:aws:sagemaker:ap-southeast-1:361886290651:model/top-category-olx
-2020-03-23-18-03-14
CPU times: user 20.9 ms, sys: 0 ns, total: 20.9 ms
Wall time: 459 ms
```

Endpoint configuration for model

Endpoint will be used by lambda function.

In [26]:

```
from time import gmtime, strftime

timestamp = time.strftime('%Y-%m-%d-%H-%M-%S', time.gmtime())
endpoint_config_name = job_name_prefix + '-epc-' + timestamp
endpoint_config_response = sagemaker.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.c4.2xlarge',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])

print('Endpoint configuration name: {}'.format(endpoint_config_name))
print('Endpoint configuration arn:  {}'.format(endpoint_config_response['Endpoin
tConfigArn']))
```

```
Endpoint configuration name: top-category-olx-epc-2020-03-23-18-27-1
5
Endpoint configuration arn:  arn:aws:sagemaker:ap-southeast-1:361886
290651:endpoint-config/top-category-olx-epc-2020-03-23-18-27-15
```

In [27]:

```
### Now it's time to deploy the model
```

In [28]:

```
%%time
import time

timestamp = time.strftime('%Y-%m-%d-%H-%M-%S', time.gmtime())
endpoint_name = job_name_prefix + '-ep-' + timestamp
print('Endpoint name: {}'.format(endpoint_name))

endpoint_params = {
    'EndpointName': endpoint_name,
    'EndpointConfigName': endpoint_config_name,
}
endpoint_response = sagemaker.create_endpoint(**endpoint_params)
print('EndpointArn = {}'.format(endpoint_response['EndpointArn']))


# get the status of the endpoint
response = sagemaker.describe_endpoint(EndpointName=endpoint_name)
status = response['EndpointStatus']
print('EndpointStatus = {}'.format(status))


# wait until the status has changed
sagemaker.get_waiter('endpoint_in_service').wait(EndpointName=endpoint_name)


# print the status of the endpoint
endpoint_response = sagemaker.describe_endpoint(EndpointName=endpoint_name)
status = endpoint_response['EndpointStatus']
print('Endpoint creation ended with EndpointStatus = {}'.format(status))

if status != 'InService':
    raise Exception('Endpoint creation failed.')
```

```
Endpoint name: top-category-olx-ep-2020-03-23-18-27-18
EndpointArn = arn:aws:sagemaker:ap-southeast-1:361886290651:endpoin
t/top-category-olx-ep-2020-03-23-18-27-18
EndpointStatus = Creating
Endpoint creation ended with EndpointStatus = InService
CPU times: user 176 ms, sys: 0 ns, total: 176 ms
Wall time: 6min 33s
```

Time to test model

I have trained an Image classifier model using OLX data. Now we have a endpoint which has the power to classify 5 different types of classes 1) Car 2) Furniture 3) Mobile 4) Books 5) Motorcyle

Now let's test our model. In **test** directory you will find 5 images of 5 different categories

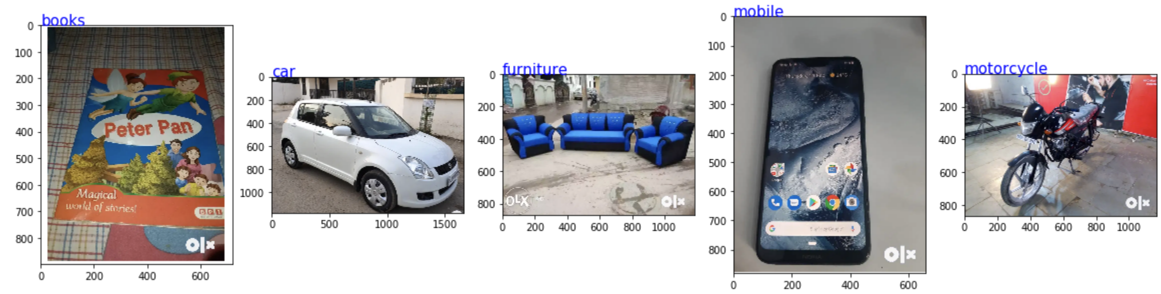
In [29]:

```
%matplotlib inline

import matplotlib.pyplot as plt
from PIL import Image

#test_categories = ['books','car', 'furniture', 'mobile', 'motorcycle']
test_categories = ['books','car', 'furniture', 'mobile', 'motorcycle']

f, axarr = plt.subplots(1, 5, figsize=(20,12))
col = 0
for i in range(5):
    im = Image.open(capstone_dir + '/test/test%d.png' % (i+1))
    axarr[col].text(0, 0, '%s' %(test_categories[i] ), fontsize=15, color='blue'
)
    frame = axarr[col].imshow(im)
    col += 1
plt.show()
```



In [30]:

```
### Now lets test our prediction
```

In [31]:

```
%matplotlib inline

import json
import numpy as np
from io import BytesIO

import matplotlib.pyplot as plt
from PIL import Image

runtime = boto3.Session().client(service_name='sagemaker-runtime')
#object_categories = ['books','car', 'furniture', 'mobile', 'motorcycle']
object_categories = ['books','car', 'furniture', 'mobile', 'motorcycle']

_, axarr = plt.subplots(1, 5, figsize=(20,12))
col = 0
for i in range(5):

    # Load the image bytes
    img = open(capstone_dir + '/test/test%d.png' % (i+1), 'rb').read()

    # Call your model for predicting which object appears in this image.
    response = runtime.invoke_endpoint(
        EndpointName=endpoint_name,
        ContentType='application/x-image',
        Body=bytearray(img)
    )
    # read the prediction result and parse the json
    result = response['Body'].read()
    result = json.loads(result)
    print(result)
    # which category has the highest confidence?
    pred_label_id = np.argmax(result)

    # Green when our model predicted correctly, otherwise, Red
    text_color = 'red'
    if object_categories[pred_label_id] == test_categories[i]:
        text_color = 'green'

    # Render the text for each image/prediction
    output_text = '%s (%f)' %(object_categories[pred_label_id], result[pred_labe
l_id] )
    axarr[col].text(0, 0, output_text, fontsize=15, color=text_color)
    print( output_text )

    # Render the image
    img = Image.open(BytesIO(img))
    frame = axarr[col].imshow(img)

    col += 1
plt.show()
```

```
[0.9998500347137451, 4.873583293374395e-06, 0.00013053567090537 4, 1.
4367052244779188e-05, 1.0494022006923842e-07]
books (0.999850)
[6.676772024150068e-09, 0.999997615814209, 4.4591327963416916e-08,
2.440585376461968e-06, 1.6897381271974155e-08]
car (0.999998)
[4.932523278711187e-10, 1.0844769526840992e-09, 1.0, 3.1677288347964
35e-08, 3.601324100044323e-11]
furniture (1.000000)
[1.0361862192928561e-12, 1.7355898619328403e-12, 2.100476476779578e-
12, 1.0, 8.295634144894759e-13]
mobile (1.000000)
[3.4090340210565957e-15, 1.3726294406657402e-12, 2.2773260707764642e
-11, 4.242176475008873e-14, 1.0]
motorcycle (1.000000)
```

In [113]:

```
sagemaker.delete_endpoint(EndpointName=endpoint_name)
```

Out[113]:

```
{'ResponseMetadata': {'RequestId': '787e61f0-6c61-4cd6-a079-f074d5a2
b3cf',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '787e61f0-6c61-4cd6-a079-f074d
5a2b3cf',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '0',
   'date': 'Sat, 07 Mar 2020 20:33:02 GMT'},
  'RetryAttempts': 0}}
```

In [ ]: