Part 0: Import Packages

```
!pip install git+git://github.com/alok-ai-lab/DeepInsight.git#egg=DeepInsight
```

```
Collecting DeepInsight
  Cloning git://github.com/alok-ai-lab/DeepInsight.git to /tmp/pip-install-egnfdqxr/deepinsig
  Running command git clone -q git://github.com/alok-ai-lab/DeepInsight.git /tmp/pip-install-
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from DeepIns
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from s
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from s
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from p
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from pytho
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from google.colab import drive
from os import listdir
from os.path import isfile, join
import glob

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline

from pyDeepInsight import ImageTransformer, LogScaler
from sklearn.model_selection import train_test_split

import time
import argparse



#import pytorch
import torch
import torchvision
import torchvision.transforms as transforms
from torch import nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
import torch.nn.init as init
from torch.utils.data import TensorDataset, DataLoader
import pickle
from torch.utils.data import Dataset, TensorDataset

import torchvision
```

```
import torchvision.transforms as transforms
```

Part 1: Load Data & Data Cleaning

```
#load content from google drive
drive.mount('/content/drive')

#get the file path
p = "/content/drive/MyDrive/2021 Fall/Intro to DS/"

app_df = pd.read_csv(p+"application_data.csv")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c
```

```
#missing data
missing_fractions = app_df.isnull().mean().sort_values(ascending=False)
drop_list = sorted(list(missing_fractions[missing_fractions > 0.3].index))
app_df.drop(labels=drop_list, axis=1, inplace=True)

app_df.drop(labels=["SK_ID_CURR"], axis=1, inplace=True)

# Behavior of the applicant before the loan application
Behavior=['AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_HOUR',
                    'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUR
                    'AMT_REQ_CREDIT_BUREAU_YEAR']

#Different Behavior variables
Behavior_2=['DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'DAYS_LAST_PHONE_CHANGE', 'DAYS_REGISTRATION','W
              'HOUR_APPR_PROCESS_START']

# indicator (dummy variable) whether the applicant provided ...
Flag=['FLAG_MOBIL',
              'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
              'FLAG_EMAIL','FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
              'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
              'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
              'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
              'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
              'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
              'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21']

# does not match variable (fraud?)
notmatch=['REG_REGION_NOT_LIVE_REGION',
              'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
              'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
              'LIVE_CITY_NOT_WORK_CITY']

#alarming columns ,past default record
```

```
Default_record=['OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
              'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE']

value_column=[]
for i in range(app_df.shape[1]):
    if type(app_df.iloc[1,i])!=str:
        value_column.append(i)

app_df=pd.get_dummies(app_df)

app_df = app_df.dropna()
```

Part 2: Transfer Tabulaer Data to Image Data

```
df_0 = app_df[app_df['TARGET'] == 0]
df_1 = app_df[app_df['TARGET'] == 1]

df_0 = df_0.sample(len(df_1))

df = pd.concat([df_0,df_1]).reset_index(drop= True)
```

```
y = df[df.columns[0]].astype(int)
X = df[df.columns[1:]]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=23,


ln = LogScaler()
X_train_norm = ln.fit_transform(X_train).fillna(0)
X_test_norm = ln.transform(X_test).fillna(0)

it = ImageTransformer(feature_extractor='tsne',
                                            pixels=(32,32), random_state=1701,
                                            n_jobs=-1)



X_train_img = it.fit_transform(X_train_norm)
X_test_img = it.transform(X_test_norm)

print(X_train_img.shape)
print(X_test_img.shape)
```

```
X_test_img  =  np.transpose(X_test_img,  (0,3,1,2))
X_train_img  =  np.transpose(X_train_img,  (0,3,1,2))

X_train  =  torch.Tensor(X_train_img)
y_train  =  torch.Tensor(y_train.to_numpy()).int()

X_test  =  torch.Tensor(X_test_img)
y_test  =  torch.Tensor(y_test.to_numpy()).int()
```

```
/usr/local/lib/python3.7/dist-packages/pyDeepInsight/image_transformer.py:309: RuntimeWarning
  X_norm = np.log(X + np.abs(self._min0) + 1).clip(0, None)
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The def
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The def
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:827: FutureWarning: 'square
  FutureWarning,
(26699, 32, 32, 3)
(11443, 32, 32, 3)
```

Part 3: Train the models in Pytorch

```
transform_train  =  transforms.Compose([
        transforms.RandomCrop(32,  padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.Normalize((0.4914,  0.4822,  0.4465),  (0.2023,  0.1994,  0.2010)),
])

transform_test  =  transforms.Compose([

        transforms.Normalize((0.4914,  0.4822,  0.4465),  (0.2023,  0.1994,  0.2010)),
])

class  CustomTensorDataset(Dataset):
        """TensorDataset  with  support  of  transforms.
        """
        def  __init__(self,  tensors,  transform=None):
                assert  all(tensors[0].size(0)  ==  tensor.size(0)  for  tensor  in  tensors)
                self.tensors  =  tensors
                self.transform  =  transform

        def  __getitem__(self,  index):
                x  =  self.tensors[0][index]

                if  self.transform:
                        x  =  self.transform(x)

                y  =  self.tensors[1][index]
                return  x,  y

        def  __len__(self):
```

```
        return self.tensors[0].size(0)




trainset = CustomTensorDataset((X_train, y_train))
trainloader = DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = CustomTensorDataset((X_test, y_test))
testloader = DataLoader(testset, batch_size=128, shuffle=True, num_workers=2)
```

```
testset.tensors[0].shape
```

```
torch.Size([11443, 3, 32, 32])
```

```
# VGG
cfg = {
    'VGG11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'VGG13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512,
    'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M',
    'VGG19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512,
}


class VGG(nn.Module):
    def __init__(self, vgg_name):
        super(VGG, self).__init__()
        self.features = self._make_layers(cfg[vgg_name])
        self.classifier = nn.Linear(512, 2)

    def forward(self, x):
        out = self.features(x)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)
        return out

    def _make_layers(self, cfg):
        layers = []
        in_channels = 3
        for x in cfg:
            if x == 'M':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1)
                           nn.BatchNorm2d(x),
                           nn.ReLU(inplace=True)]
                in_channels = x
        layers += [nn.AvgPool2d(kernel_size=1, stride=1)]
```

```
        return nn.Sequential(*layers)
```

```
#set up gpu
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)
```

```
cuda:0
```

```
def train_model(model,learning_rate = 0.001,num_epochs = 80):

    if device == 'cuda':
        model = torch.nn.DataParallel(model)
        cudnn.benchmark = True

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    # For updating learning rate
    def update_lr(optimizer, lr):
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr

    # Train the model
    total_step = len(trainloader)
    curr_lr = learning_rate
    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(trainloader):
            images = images.to(device)
            labels = labels.to(device)
            labels = labels.to(torch.long)


            # Forward pass
            outputs = model(images)

            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if (i+1) % 100 == 0:
                print ("Epoch [{}/{}], Step [{}/{}] Loss: {:.4f}"
                       .format(epoch+1, num_epochs, i+1, total_step, loss.item

        # Decay learning rate
        if (epoch+1) % 20 == 0:
            curr_lr /= 3
            update_lr(optimizer, curr_lr)
```

```
    return model
```

```
# Test the model
def test_model(model):
    model.eval()
    with torch.no_grad():
            correct = 0
            total = 0
            for images, labels in testloader:
                    images = images.to(device)
                    labels = labels.to(device)
                    labels = labels.to(torch.long)
                    outputs = model(images)
                    _, predicted = torch.max(outputs.data, 1)
                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()

            print('Accuracy of the model on the test images: {} %'.format(100 * correct
```

```
# VGG11

vggnet = VGG('VGG11').to(device)
vggnet_trained = train_model(model=vggnet)
```

```
Epoch [1/80], Step [100/209] Loss: 0.6580
Epoch [1/80], Step [200/209] Loss: 0.7746
Epoch [2/80], Step [100/209] Loss: 0.6549
Epoch [2/80], Step [200/209] Loss: 0.6643
Epoch [3/80], Step [100/209] Loss: 0.6018
Epoch [3/80], Step [200/209] Loss: 0.6491
Epoch [4/80], Step [100/209] Loss: 0.6084
Epoch [4/80], Step [200/209] Loss: 0.6295
Epoch [5/80], Step [100/209] Loss: 0.6522
Epoch [5/80], Step [200/209] Loss: 0.6083
Epoch [6/80], Step [100/209] Loss: 0.6547
Epoch [6/80], Step [200/209] Loss: 0.6326
Epoch [7/80], Step [100/209] Loss: 0.6001
Epoch [7/80], Step [200/209] Loss: 0.6616
Epoch [8/80], Step [100/209] Loss: 0.6133
Epoch [8/80], Step [200/209] Loss: 0.6350
Epoch [9/80], Step [100/209] Loss: 0.6522
Epoch [9/80], Step [200/209] Loss: 0.6375
Epoch [10/80], Step [100/209] Loss: 0.6295
Epoch [10/80], Step [200/209] Loss: 0.6291
Epoch [11/80], Step [100/209] Loss: 0.6764
Epoch [11/80], Step [200/209] Loss: 0.6136
Epoch [12/80], Step [100/209] Loss: 0.6530
Epoch [12/80], Step [200/209] Loss: 0.5998
Epoch [13/80], Step [100/209] Loss: 0.6254
Epoch [13/80], Step [200/209] Loss: 0.6405
Epoch [14/80], Step [100/209] Loss: 0.6573
Epoch [14/80], Step [200/209] Loss: 0.6234
Epoch [15/80], Step [100/209] Loss: 0.6264
```

```
Epoch [15/80], Step [200/209] Loss: 0.6559
Epoch [16/80], Step [100/209] Loss: 0.6281
Epoch [16/80], Step [200/209] Loss: 0.6134
Epoch [17/80], Step [100/209] Loss: 0.6172
Epoch [17/80], Step [200/209] Loss: 0.6447
Epoch [18/80], Step [100/209] Loss: 0.6570
Epoch [18/80], Step [200/209] Loss: 0.5910
Epoch [19/80], Step [100/209] Loss: 0.6137
Epoch [19/80], Step [200/209] Loss: 0.6412
Epoch [20/80], Step [100/209] Loss: 0.6570
Epoch [20/80], Step [200/209] Loss: 0.5728
Epoch [21/80], Step [100/209] Loss: 0.5956
Epoch [21/80], Step [200/209] Loss: 0.6057
Epoch [22/80], Step [100/209] Loss: 0.6547
Epoch [22/80], Step [200/209] Loss: 0.6833
Epoch [23/80], Step [100/209] Loss: 0.6588
Epoch [23/80], Step [200/209] Loss: 0.5836
Epoch [24/80], Step [100/209] Loss: 0.6444
Epoch [24/80], Step [200/209] Loss: 0.6165
Epoch [25/80], Step [100/209] Loss: 0.5981
Epoch [25/80], Step [200/209] Loss: 0.6079
Epoch [26/80], Step [100/209] Loss: 0.6400
Epoch [26/80], Step [200/209] Loss: 0.5335
Epoch [27/80], Step [100/209] Loss: 0.6355
Epoch [27/80], Step [200/209] Loss: 0.6224
Epoch [28/80], Step [100/209] Loss: 0.6151
Epoch [28/80], Step [200/209] Loss: 0.6062
Epoch [29/80], Step [100/209] Loss: 0.5416
Epoch [29/80], Step [200/209] Loss: 0.6166
```

```
#VGG  test
test_model(vggnet_trained)
```

```
Accuracy of the model on the test images: 58.428733723673865 %
```

```
#VGG  13

vggnet  =  VGG('VGG13').to(device)
vggnet_trained  =  train_model(model=vggnet)
```

```
Epoch [1/80], Step [100/209] Loss: 0.6820
Epoch [1/80], Step [200/209] Loss: 0.7306
Epoch [2/80], Step [100/209] Loss: 0.6715
Epoch [2/80], Step [200/209] Loss: 0.6982
Epoch [3/80], Step [100/209] Loss: 0.6290
Epoch [3/80], Step [200/209] Loss: 0.6430
Epoch [4/80], Step [100/209] Loss: 0.6414
Epoch [4/80], Step [200/209] Loss: 0.6489
Epoch [5/80], Step [100/209] Loss: 0.6395
Epoch [5/80], Step [200/209] Loss: 0.6545
Epoch [6/80], Step [100/209] Loss: 0.6482
Epoch [6/80], Step [200/209] Loss: 0.7221
Epoch [7/80], Step [100/209] Loss: 0.6779
Epoch [7/80], Step [200/209] Loss: 0.6201
Epoch [8/80], Step [100/209] Loss: 0.6715
```

```
Epoch [8/80], Step [200/209] Loss: 0.6336
Epoch [9/80], Step [100/209] Loss: 0.6854
Epoch [9/80], Step [200/209] Loss: 0.6207
Epoch [10/80], Step [100/209] Loss: 0.6088
Epoch [10/80], Step [200/209] Loss: 0.6937
Epoch [11/80], Step [100/209] Loss: 0.6260
Epoch [11/80], Step [200/209] Loss: 0.6369
Epoch [12/80], Step [100/209] Loss: 0.6324
Epoch [12/80], Step [200/209] Loss: 0.6420
Epoch [13/80], Step [100/209] Loss: 0.6869
Epoch [13/80], Step [200/209] Loss: 0.6215
Epoch [14/80], Step [100/209] Loss: 0.5916
Epoch [14/80], Step [200/209] Loss: 0.6825
Epoch [15/80], Step [100/209] Loss: 0.6806
Epoch [15/80], Step [200/209] Loss: 0.6807
Epoch [16/80], Step [100/209] Loss: 0.5895
Epoch [16/80], Step [200/209] Loss: 0.6356
Epoch [17/80], Step [100/209] Loss: 0.7120
Epoch [17/80], Step [200/209] Loss: 0.5983
Epoch [18/80], Step [100/209] Loss: 0.6221
Epoch [18/80], Step [200/209] Loss: 0.6013
Epoch [19/80], Step [100/209] Loss: 0.5640
Epoch [19/80], Step [200/209] Loss: 0.6024
Epoch [20/80], Step [100/209] Loss: 0.6088
Epoch [20/80], Step [200/209] Loss: 0.5739
Epoch [21/80], Step [100/209] Loss: 0.6571
Epoch [21/80], Step [200/209] Loss: 0.5859
Epoch [22/80], Step [100/209] Loss: 0.5930
Epoch [22/80], Step [200/209] Loss: 0.6107
Epoch [23/80], Step [100/209] Loss: 0.6101
Epoch [23/80], Step [200/209] Loss: 0.5255
Epoch [24/80], Step [100/209] Loss: 0.6736
Epoch [24/80], Step [200/209] Loss: 0.6218
Epoch [25/80], Step [100/209] Loss: 0.6190
Epoch [25/80], Step [200/209] Loss: 0.6426
Epoch [26/80], Step [100/209] Loss: 0.6149
Epoch [26/80], Step [200/209] Loss: 0.5543
Epoch [27/80], Step [100/209] Loss: 0.6332
Epoch [27/80], Step [200/209] Loss: 0.5772
Epoch [28/80], Step [100/209] Loss: 0.5934
Epoch [28/80], Step [200/209] Loss: 0.5672
Epoch [29/80], Step [100/209] Loss: 0.5844
Epoch [29/80], Step [200/209] Loss: 0.5700
```

```
#VGG  test
test_model(vggnet_trained)
```

```
Accuracy of the model on the test images: 57.72087739229223 %
```

```
#VGG  16

vggnet  =  VGG('VGG16').to(device)
vggnet_trained  =  train_model(model=vggnet)
```

```
Epoch [1/80], Step [100/209] Loss: 0.6832
Epoch [1/80], Step [200/209] Loss: 0.6900
Epoch [2/80], Step [100/209] Loss: 0.6637
Epoch [2/80], Step [200/209] Loss: 0.6096
```

```
Epoch [3/80], Step [100/209] Loss: 0.6113
Epoch [3/80], Step [200/209] Loss: 0.6554
Epoch [4/80], Step [100/209] Loss: 0.6629
Epoch [4/80], Step [200/209] Loss: 0.6385
Epoch [5/80], Step [100/209] Loss: 0.7083
Epoch [5/80], Step [200/209] Loss: 0.6695
Epoch [6/80], Step [100/209] Loss: 0.6226
Epoch [6/80], Step [200/209] Loss: 0.6289
Epoch [7/80], Step [100/209] Loss: 0.6411
Epoch [7/80], Step [200/209] Loss: 0.6615
Epoch [8/80], Step [100/209] Loss: 0.6499
Epoch [8/80], Step [200/209] Loss: 0.6444
Epoch [9/80], Step [100/209] Loss: 0.6533
Epoch [9/80], Step [200/209] Loss: 0.6156
Epoch [10/80], Step [100/209] Loss: 0.5998
Epoch [10/80], Step [200/209] Loss: 0.6146
Epoch [11/80], Step [100/209] Loss: 0.6294
Epoch [11/80], Step [200/209] Loss: 0.6544
Epoch [12/80], Step [100/209] Loss: 0.5423
Epoch [12/80], Step [200/209] Loss: 0.6381
Epoch [13/80], Step [100/209] Loss: 0.6490
Epoch [13/80], Step [200/209] Loss: 0.6043
Epoch [14/80], Step [100/209] Loss: 0.6070
Epoch [14/80], Step [200/209] Loss: 0.7285
Epoch [15/80], Step [100/209] Loss: 0.6384
Epoch [15/80], Step [200/209] Loss: 0.6546
Epoch [16/80], Step [100/209] Loss: 0.6624
Epoch [16/80], Step [200/209] Loss: 0.7037
Epoch [17/80], Step [100/209] Loss: 0.6725
Epoch [17/80], Step [200/209] Loss: 0.6211
Epoch [18/80], Step [100/209] Loss: 0.6481
Epoch [18/80], Step [200/209] Loss: 0.6781
Epoch [19/80], Step [100/209] Loss: 0.6293
Epoch [19/80], Step [200/209] Loss: 0.6266
Epoch [20/80], Step [100/209] Loss: 0.6121
Epoch [20/80], Step [200/209] Loss: 0.6826
Epoch [21/80], Step [100/209] Loss: 0.6281
Epoch [21/80], Step [200/209] Loss: 0.6227
Epoch [22/80], Step [100/209] Loss: 0.5660
Epoch [22/80], Step [200/209] Loss: 0.6538
Epoch [23/80], Step [100/209] Loss: 0.5983
Epoch [23/80], Step [200/209] Loss: 0.6428
Epoch [24/80], Step [100/209] Loss: 0.5819
Epoch [24/80], Step [200/209] Loss: 0.6287
Epoch [25/80], Step [100/209] Loss: 0.5680
Epoch [25/80], Step [200/209] Loss: 0.6440
Epoch [26/80], Step [100/209] Loss: 0.6265
Epoch [26/80], Step [200/209] Loss: 0.6049
Epoch [27/80], Step [100/209] Loss: 0.6156
Epoch [27/80], Step [200/209] Loss: 0.5480
Epoch [28/80], Step [100/209] Loss: 0.6075
Epoch [28/80], Step [200/209] Loss: 0.5973
Epoch [29/80], Step [100/209] Loss: 0.6689
Epoch [29/80], Step [200/209] Loss: 0.6598
```

```
#VGG  test
test_model(vggnet_trained)
```

```
#VGG  19
```

```
vggnet = VGG('VGG19').to(device)
vggnet_trained = train_model(model=vggnet
```

```
Epoch [1/80], Step [100/209] Loss: 0.6343
Epoch [1/80], Step [200/209] Loss: 0.6324
Epoch [2/80], Step [100/209] Loss: 0.6907
Epoch [2/80], Step [200/209] Loss: 0.6888
Epoch [3/80], Step [100/209] Loss: 0.6660
Epoch [3/80], Step [200/209] Loss: 0.6190
Epoch [4/80], Step [100/209] Loss: 0.6442
Epoch [4/80], Step [200/209] Loss: 0.6344
Epoch [5/80], Step [100/209] Loss: 0.6608
Epoch [5/80], Step [200/209] Loss: 0.6622
Epoch [6/80], Step [100/209] Loss: 0.6304
Epoch [6/80], Step [200/209] Loss: 0.5803
Epoch [7/80], Step [100/209] Loss: 0.6017
Epoch [7/80], Step [200/209] Loss: 0.6487
Epoch [8/80], Step [100/209] Loss: 0.6715
Epoch [8/80], Step [200/209] Loss: 0.6300
Epoch [9/80], Step [100/209] Loss: 0.5982
Epoch [9/80], Step [200/209] Loss: 0.6179
Epoch [10/80], Step [100/209] Loss: 0.6234
Epoch [10/80], Step [200/209] Loss: 0.6521
Epoch [11/80], Step [100/209] Loss: 0.5972
Epoch [11/80], Step [200/209] Loss: 0.5787
Epoch [12/80], Step [100/209] Loss: 0.6205
Epoch [12/80], Step [200/209] Loss: 0.6228
Epoch [13/80], Step [100/209] Loss: 0.5999
Epoch [13/80], Step [200/209] Loss: 0.6129
Epoch [14/80], Step [100/209] Loss: 0.6010
Epoch [14/80], Step [200/209] Loss: 0.6643
Epoch [15/80], Step [100/209] Loss: 0.6335
Epoch [15/80], Step [200/209] Loss: 0.6539
Epoch [16/80], Step [100/209] Loss: 0.6380
Epoch [16/80], Step [200/209] Loss: 0.5658
Epoch [17/80], Step [100/209] Loss: 0.6076
Epoch [17/80], Step [200/209] Loss: 0.6137
Epoch [18/80], Step [100/209] Loss: 0.6014
Epoch [18/80], Step [200/209] Loss: 0.6033
Epoch [19/80], Step [100/209] Loss: 0.6072
Epoch [19/80], Step [200/209] Loss: 0.5326
Epoch [20/80], Step [100/209] Loss: 0.5780
Epoch [20/80], Step [200/209] Loss: 0.6125
Epoch [21/80], Step [100/209] Loss: 0.6083
Epoch [21/80], Step [200/209] Loss: 0.5708
Epoch [22/80], Step [100/209] Loss: 0.5318
Epoch [22/80], Step [200/209] Loss: 0.5703
Epoch [23/80], Step [100/209] Loss: 0.5849
Epoch [23/80], Step [200/209] Loss: 0.5571
Epoch [24/80], Step [100/209] Loss: 0.5830
Epoch [24/80], Step [200/209] Loss: 0.5388
Epoch [25/80], Step [100/209] Loss: 0.6266
Epoch [25/80], Step [200/209] Loss: 0.5477
Epoch [26/80], Step [100/209] Loss: 0.5807
Epoch [26/80], Step [200/209] Loss: 0.5581
Epoch [27/80], Step [100/209] Loss: 0.5892
Epoch [27/80], Step [200/209] Loss: 0.5453
Epoch [28/80], Step [100/209] Loss: 0.5721
```

```
Epoch [28/80], Step [200/209] Loss: 0.5780
Epoch [29/80], Step [100/209] Loss: 0.5758
```

```
#VGG  test
test_model(vggnet_trained)
```

```
Accuracy of the model on the test images: 60.0279646945731 %
```

Double-click (or enter) to edit