PROJECT

Sports image classification

Abstract

In this Project we have used Convolutional Neural Network to recognise the sport from its image. We have considered 22 sports as "badminton", "baseball", "basketball", "boxing", "chess", "cricket", "fencing", "football", "formula1", "gymnastics", "hockey", "ice_hockey", "kabaddi", "motogp", "shooting", "swimming", "table_tennis", "tennis", "volleyball", "weight_lifting", "wrestling", "wwe". 'sports-image-dataset', consists of 14141 observations/images with sports name as labels. Firstly, the data is divided into training set, validation set and test set with ratio 8:1:1. Then from the training set we found out the number of observations corresponding to each label. Next, we trained the training data on different models and based on accuracy from the Validation set , we have chosen the best model and found its test accuracy. Lastly, we test our model with some real examples. Here, we have only provided the best of all the models which we have tried.

INTRODUCTION

In the era of Artificial Intelligence many unbelievable things become possible. Using a machine we can find out the person in front of you are telling lie or truth, just seeing a image of person Facebook can detect its name and details, hearing our voice Google understands and shows us the relevant websites etc. CNN in Deep Learning is one such tool to achieve perfection in image recognition. Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although CNNs trained by backpropagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs with max-pooling on GPUs in the style of Ciresan and colleagues were needed to progress on computer vision. In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image segmentation contest. Until 2011, CNNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR showed how max-pooling CNNs on GPU can dramatically improve many vision benchmark records. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection.

Here we have also used CNN to recognise sports name by seeing the picture of the sport.

OBJECTIVE

Objective of the project is to develop a sport recogniser based on 14.1K sample images.

DATA INFORMATION

This 'sports-image-dataset' data is collected from Kaggle. There are 14141 examples in the data and each example is rbg image. Data is divided into 80%, 10% and 10% respectively for train set, dev set and test set. Thus we get 11312 examples in the train set, 1414 examples the dev set and 1415 images in the test set. The data consists of 22 folders with label name as sports name. Each folder consists of around 800-900 images. This dataset is collected from Google Images using Images Scrapper.

It consists of 22 folders with label name as sports name. Each folder consists of around 800-900 images. This dataset is collected from Google Images using Images Scrapper.

ANALYSIS

In the whole analysis various models are used. These models vary by hyperparameters such as number of

convnet layers, number of fully connected layers, number of filters, size of filter, activation functions, dropout and even training data size. As there are twentytwo categories , 'softmax' activation is in the final layer instead of 'sigmoid' (which is used for binary category). Keras is used in the analysis with Tensorflow at the backend.

In [ ]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

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

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

Necessary modules to Import

In [2]:

```
! pip install imutils
```

```
Collecting imutils
  Downloading imutils-0.5.4.tar.gz (17 kB)
Building wheels for collected packages: imutils
  Building wheel for imutils (setup.py) ... done
  Created wheel for imutils: filename=imutils-0.5.4-py3-none-any.whl size=25860 sha256=9b
6294d8ee7cc962e8c91a34d40958dfd86ab0ad581cf213daafe4ab4bcb13d3
  Stored in directory: /root/.cache/pip/wheels/86/d7/0a/4923351ed1cec5d5e24c1eaf8905567b0
2a0343b24aa873df2
Successfully built imutils
Installing collected packages: imutils
Successfully installed imutils-0.5.4
WARNING: Running pip as root will break packages and permissions. You should install pack
ages reliably by using venv: https://pip.pypa.io/warnings/venv
```

In [3]:

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

# import the necessary packages

from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
```

```
from keras import backend as K
from keras.callbacks import EarlyStopping
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import argparse
import random
import pickle
import cv2
import os
%matplotlib inline
```

Visiting the data

Let's check the data.

In [4]:

```
# initialize the data and labels
print("[INFO] loading images...")
random.seed(13)
data = []
labels = []
#Enter the path of your image data folder
image_data_folder_path = "../input/sports-image-dataset/data"

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(image_data_folder_path)))

total_number_of_images = len(imagePaths)
print("\n")
print("Total number of images----->",total_number_of_images)
random.shuffle(imagePaths)

# loop over the input images
for imagePath in imagePaths:
    # load the image, resize it to 84x84 pixels (the required input
    # spatial dimensions of SmallVGGNet), and store the image in the
    # data list
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (84,84))
    data.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    labels.append(label)
print ("data",data[0].shape)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
print(labels)
print(labels.shape)
```

```
[INFO] loading images...


Total number of images-----> 14141
data (84, 84, 3)
['gymnastics' 'tennis' 'badminton' ... 'table_tennis' 'formula1' 'motogp']
(14141,)
```

In [5]:

```
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, val_testX, trainY, val_testY) = train_test_split(data, labels, test_size=0.2, r
andom_state=42)
```

```
print ("trainX.shape------>>",trainX.shape)
(testX, valX, testY, valY) = train_test_split(val_testX, val_testY, test_size=0.5, rando
m_state=42)

# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
valY = lb.transform(valY)

height = 84
width = 84
depth =3

inputShape = (height, width, depth)

classes = len(lb.classes_)
print("Number of classes:",classes)
print("Number of training images:", len(trainX))
print("Number of validation images:", len(valY))
print("Number of test images:", len(testX))
```

```
trainX.shape------>> (11312, 84, 84, 3)
Number of classes: 22
Number of training images: 11312
Number of validation images: 1415
Number of test images: 1414
```

Dividing data into training set, validation(developement) set and testing set

In [6]:

```
classes_name =  ["badminton", "baseball", "basketball", "boxing",
                "chess", "cricket", "fencing", "football", "formula1",
                "gymnastics", "hockey", "ice_hockey", "kabaddi", "motogp",
                "shooting", "swimming", "table_tennis", "tennis", "volleyball",
                "weight_lifting", "wrestling", "wwe"]
```

Plotting some images from data randomly

In [7]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(trainX[i], cmap=plt.cm.binary)
    plt.xlabel(classes_name[np.argmax(trainY[i])])
plt.show()
```

In [8]:

```
# initialize number of epochs to train for, and batch size
EPOCHS = 50
BS = 32
```

number of epochs and batch size

In [9]:

```
random_state = 99
```

MODEL1

In [10]:

```
chanDim=3
model1 = Sequential()
# CONV => RELU => POOL layer set
model1.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model1.add(Activation("relu"))
model1.add(BatchNormalization(axis=chanDim))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

# (CONV => RELU) * 2 => POOL layer set
model1.add(Conv2D(64, (3, 3), padding="same"))
model1.add(Activation("relu"))
model1.add(BatchNormalization(axis=chanDim))
model1.add(Conv2D(64, (3, 3), padding="same"))
model1.add(Activation("relu"))
model1.add(BatchNormalization(axis=chanDim))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

# (CONV => RELU) * 3 => POOL layer set
model1.add(Conv2D(128, (3, 3), padding="same"))
model1.add(Activation("relu"))
model1.add(BatchNormalization(axis=chanDim))
model1.add(Conv2D(128, (3, 3), padding="same"))
model1.add(Activation("relu"))
model1.add(BatchNormalization(axis=chanDim))
model1.add(Conv2D(128, (3, 3), padding="same"))
model1.add(Activation("relu"))
model1.add(BatchNormalization(axis=chanDim))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
model1.add(Flatten())
model1.add(Dense(512))
```

```python
model1.add(Activation("relu"))
model1.add(BatchNormalization())
model1.add(Dropout(0.5))
model1.add(Dense(128))
model1.add(Activation("relu"))
model1.add(BatchNormalization())
model1.add(Dropout(0.3))


# softmax classifier
model1.add(Dense(classes))
model1.add(Activation("softmax"))


model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 84, 84, 32)        896       
_________________________________________________________________
activation (Activation)      (None, 84, 84, 32)        0         
_________________________________________________________________
batch_normalization (BatchNo (None, 84, 84, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 42, 42, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 42, 42, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 42, 42, 64)        18496     
_________________________________________________________________
activation_1 (Activation)    (None, 42, 42, 64)        0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 42, 42, 64)        256       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 42, 42, 64)        36928     
_________________________________________________________________
activation_2 (Activation)    (None, 42, 42, 64)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 42, 42, 64)        256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 21, 21, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 21, 21, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 128)       73856     
_________________________________________________________________
activation_3 (Activation)    (None, 21, 21, 128)       0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 21, 21, 128)       512       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 21, 21, 128)       147584    
_________________________________________________________________
activation_4 (Activation)    (None, 21, 21, 128)       0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 21, 21, 128)       512       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 21, 21, 128)       147584    
_________________________________________________________________
activation_5 (Activation)    (None, 21, 21, 128)       0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 21, 21, 128)       512       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 128)       0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 10, 128)       0         
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0         
_________________________________________________________________
dense (Dense)                (None, 512)               6554112   
_________________________________________________________________
```

```
activation_6 (Activation)    (None, 512)               0
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
activation_7 (Activation)    (None, 128)               0
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 22)                2838
_________________________________________________________________
activation_8 (Activation)    (None, 22)                0
=================================================================
Total params: 7,052,694
Trainable params: 7,050,326
Non-trainable params: 2,368
_________________________________________________________________
```

In [11]:

```
# initialize the model and optimizer
model1.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

In [12]:

```
# train the network
H1 = model1.fit(trainX, trainY, batch_size=BS,
    validation_data=(valX, valY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS)
```

```
Epoch 1/50
353/353 [==============================] - 14s 19ms/step - loss: 3.0802 - accuracy: 0.177
5 - val_loss: 5.3347 - val_accuracy: 0.1251
Epoch 2/50
353/353 [==============================] - 6s 16ms/step - loss: 2.1995 - accuracy: 0.3474
- val_loss: 3.5301 - val_accuracy: 0.2424
Epoch 3/50
353/353 [==============================] - 6s 16ms/step - loss: 1.9304 - accuracy: 0.4348
- val_loss: 2.4070 - val_accuracy: 0.3555
Epoch 4/50
353/353 [==============================] - 6s 16ms/step - loss: 1.6910 - accuracy: 0.4974
- val_loss: 1.7497 - val_accuracy: 0.4876
Epoch 5/50
353/353 [==============================] - 6s 16ms/step - loss: 1.5757 - accuracy: 0.5235
- val_loss: 1.7241 - val_accuracy: 0.5166
Epoch 6/50
353/353 [==============================] - 6s 16ms/step - loss: 1.4441 - accuracy: 0.5565
- val_loss: 1.6150 - val_accuracy: 0.5293
Epoch 7/50
353/353 [==============================] - 6s 16ms/step - loss: 1.3080 - accuracy: 0.6050
- val_loss: 1.6397 - val_accuracy: 0.5230
Epoch 8/50
353/353 [==============================] - 5s 16ms/step - loss: 1.2177 - accuracy: 0.6315
- val_loss: 1.6879 - val_accuracy: 0.5159
Epoch 9/50
353/353 [==============================] - 6s 16ms/step - loss: 1.0760 - accuracy: 0.6775
- val_loss: 1.6258 - val_accuracy: 0.5329
Epoch 10/50
353/353 [==============================] - 6s 16ms/step - loss: 0.9910 - accuracy: 0.6963
- val_loss: 1.5016 - val_accuracy: 0.5802
Epoch 11/50
353/353 [==============================] - 6s 16ms/step - loss: 0.8423 - accuracy: 0.7402
- val_loss: 1.7954 - val_accuracy: 0.5187
Epoch 12/50
353/353 [==============================] - 6s 16ms/step - loss: 0.7501 - accuracy: 0.7743
- val_loss: 1.5323 - val_accuracy: 0.5852
Epoch 13/50
```

```
353/353 [==============================] - 6s 16ms/step - loss: 0.6388 - accuracy: 0.7975
- val_loss: 1.6843 - val_accuracy: 0.5519
Epoch 14/50
353/353 [==============================] - 6s 16ms/step - loss: 0.5516 - accuracy: 0.8299
- val_loss: 1.6153 - val_accuracy: 0.5830
Epoch 15/50
353/353 [==============================] - 6s 17ms/step - loss: 0.5328 - accuracy: 0.8331
- val_loss: 1.8388 - val_accuracy: 0.5399
Epoch 16/50
353/353 [==============================] - 6s 16ms/step - loss: 0.4432 - accuracy: 0.8583
- val_loss: 1.8456 - val_accuracy: 0.5654
Epoch 17/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3682 - accuracy: 0.8786
- val_loss: 1.9624 - val_accuracy: 0.5435
Epoch 18/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3398 - accuracy: 0.8920
- val_loss: 1.6957 - val_accuracy: 0.5965
Epoch 19/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3372 - accuracy: 0.8898
- val_loss: 1.7947 - val_accuracy: 0.5986
Epoch 20/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2947 - accuracy: 0.9061
- val_loss: 1.9320 - val_accuracy: 0.5611
Epoch 21/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3374 - accuracy: 0.8940
- val_loss: 1.6511 - val_accuracy: 0.6134
Epoch 22/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3283 - accuracy: 0.8966
- val_loss: 1.7747 - val_accuracy: 0.6106
Epoch 23/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2522 - accuracy: 0.9157
- val_loss: 1.6884 - val_accuracy: 0.6191
Epoch 24/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2362 - accuracy: 0.9235
- val_loss: 1.7494 - val_accuracy: 0.6382
Epoch 25/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2351 - accuracy: 0.9203
- val_loss: 1.9272 - val_accuracy: 0.5816
Epoch 26/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2518 - accuracy: 0.9175
- val_loss: 1.9984 - val_accuracy: 0.5809
Epoch 27/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2188 - accuracy: 0.9258
- val_loss: 1.7327 - val_accuracy: 0.6184
Epoch 28/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1895 - accuracy: 0.9361
- val_loss: 1.8945 - val_accuracy: 0.5986
Epoch 29/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1774 - accuracy: 0.9441
- val_loss: 2.4399 - val_accuracy: 0.5293
Epoch 30/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1775 - accuracy: 0.9391
- val_loss: 1.8464 - val_accuracy: 0.6092
Epoch 31/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1832 - accuracy: 0.9411
- val_loss: 2.1333 - val_accuracy: 0.5859
Epoch 32/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1773 - accuracy: 0.9426
- val_loss: 1.9801 - val_accuracy: 0.5965
Epoch 33/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1711 - accuracy: 0.9442
- val_loss: 2.4122 - val_accuracy: 0.5491
Epoch 34/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1662 - accuracy: 0.9440
- val_loss: 2.0352 - val_accuracy: 0.5943
Epoch 35/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1672 - accuracy: 0.9466
- val_loss: 2.0189 - val_accuracy: 0.5915
Epoch 36/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1316 - accuracy: 0.9577
- val_loss: 2.0009 - val_accuracy: 0.6035
Epoch 37/50
```

```
353/353 [==============================] - 6s 16ms/step - loss: 0.1342 - accuracy: 0.9582
- val_loss: 1.9545 - val_accuracy: 0.6184
Epoch 38/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1576 - accuracy: 0.9498
- val_loss: 2.1410 - val_accuracy: 0.5781
Epoch 39/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1465 - accuracy: 0.9533
- val_loss: 1.9889 - val_accuracy: 0.6212
Epoch 40/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1971 - accuracy: 0.9375
- val_loss: 1.9968 - val_accuracy: 0.6155
Epoch 41/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1550 - accuracy: 0.9517
- val_loss: 2.1492 - val_accuracy: 0.5689
Epoch 42/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1361 - accuracy: 0.9555
- val_loss: 2.0047 - val_accuracy: 0.6057
Epoch 43/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1208 - accuracy: 0.9598
- val_loss: 2.1573 - val_accuracy: 0.5979
Epoch 44/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2916 - accuracy: 0.9125
- val_loss: 1.9357 - val_accuracy: 0.6078
Epoch 45/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1401 - accuracy: 0.9543
- val_loss: 1.8209 - val_accuracy: 0.6269
Epoch 46/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1302 - accuracy: 0.9583
- val_loss: 1.9421 - val_accuracy: 0.6205
Epoch 47/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1008 - accuracy: 0.9688
- val_loss: 1.9988 - val_accuracy: 0.6148
Epoch 48/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1075 - accuracy: 0.9680
- val_loss: 1.9213 - val_accuracy: 0.6134
Epoch 49/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1134 - accuracy: 0.9648
- val_loss: 1.9158 - val_accuracy: 0.6170
Epoch 50/50
353/353 [==============================] - 6s 16ms/step - loss: 0.0931 - accuracy: 0.9705
- val_loss: 2.0435 - val_accuracy: 0.6177
```

In [13]:

```
preds1=model1.evaluate(testX, testY, batch_size=32)
print('loss = '+str(preds1[0]))
print('test accuracy = '+str(preds1[1]))
```

```
45/45 [==============================] - 0s 7ms/step - loss: 2.0623 - accuracy: 0.6025
loss = 2.0623135566711426
test accuracy = 0.602545976638794
```

We got 63.02% test accuracy using 'Adam' optimizer.

In [14]:

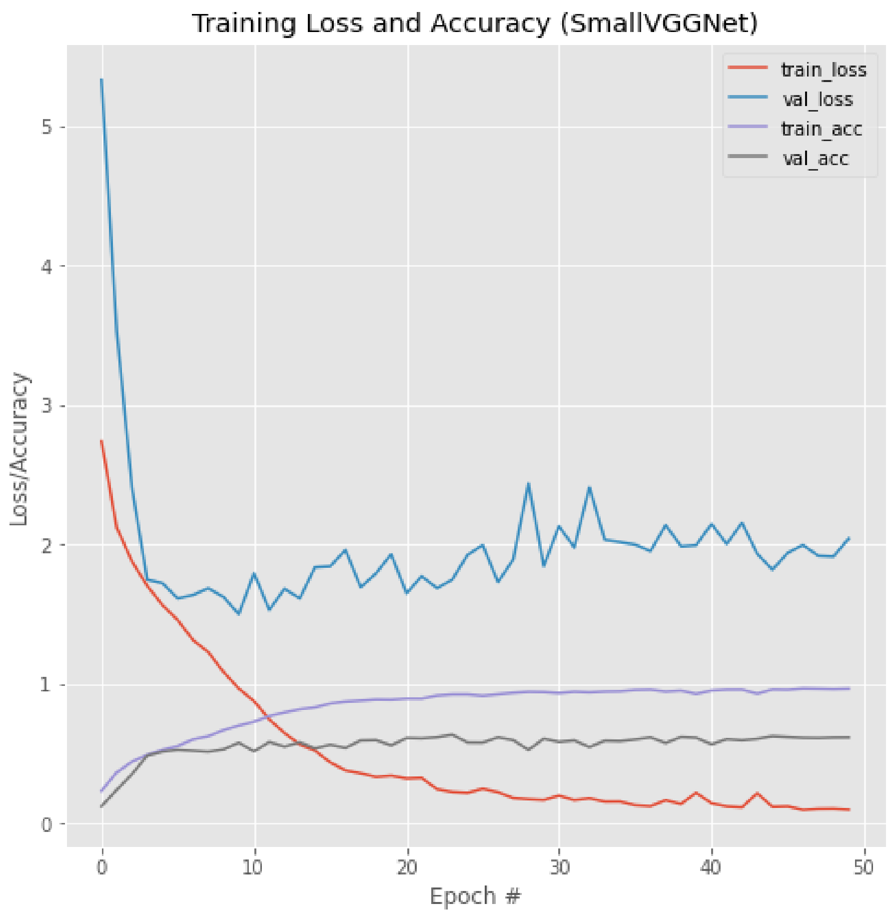
```
# evaluate the network
print("[INFO] evaluating network...")
predictions1 = model1.predict(valX, batch_size=32)
print(classification_report(valY.argmax(axis=1),
    predictions1.argmax(axis=1), target_names=lb.classes_))
```

```
[INFO] evaluating network...
                precision    recall  f1-score   support

     badminton       0.50      0.57      0.53        79
      baseball       0.82      0.62      0.71        89
    basketball       0.58      0.37      0.45        49
        boxing       0.72      0.65      0.68        77
         chess       0.65      0.27      0.38        49
       cricket       0.69      0.70      0.70        74
```

fencing	0.67	0.65	0.66	65
football	0.72	0.66	0.69	77
formula1	0.61	0.60	0.61	68
gymnastics	0.47	0.52	0.50	69
hockey	0.62	0.53	0.57	45
ice_hockey	0.88	0.68	0.77	76
kabaddi	0.72	0.62	0.67	37
motogp	0.74	0.82	0.78	62
shooting	0.39	0.57	0.46	40
swimming	0.90	0.78	0.83	68
table_tennis	0.38	0.69	0.49	74
tennis	0.58	0.54	0.56	84
volleyball	0.68	0.67	0.68	76
weight_lifting	0.64	0.53	0.58	51
wrestling	0.44	0.48	0.46	48
wwe	0.54	0.83	0.65	58
accuracy			0.62	1415
macro avg	0.63	0.61	0.61	1415
weighted avg	0.64	0.62	0.62	1415

In [15]:

```
# plot the training loss and accuracy
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure(figsize=(8,8))
plt.plot(N, H1.history["loss"], label="train_loss")
plt.plot(N, H1.history["val_loss"], label="val_loss")
plt.plot(N, H1.history["accuracy"], label="train_acc")
plt.plot(N, H1.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy (SmallVGGNet)")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig("smallvggnet_plot.png")
plt.show()
```



Clearly the model is overfit. Now we want to do some regualrization, such as try some different optimizer.

MODEL 2

In [16]:

```
chanDim=3
model2 = Sequential()
# CONV => RELU => POOL layer set
model2.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model2.add(Activation("relu"))
model2.add(BatchNormalization(axis=chanDim))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

# (CONV => RELU) * 2 => POOL layer set
model2.add(Conv2D(64, (3, 3), padding="same"))
model2.add(Activation("relu"))
model2.add(BatchNormalization(axis=chanDim))
model2.add(Conv2D(64, (3, 3), padding="same"))
model2.add(Activation("relu"))
model2.add(BatchNormalization(axis=chanDim))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

# (CONV => RELU) * 3 => POOL layer set
model2.add(Conv2D(128, (3, 3), padding="same"))
model2.add(Activation("relu"))
model2.add(BatchNormalization(axis=chanDim))
model2.add(Conv2D(128, (3, 3), padding="same"))
model2.add(Activation("relu"))
model2.add(BatchNormalization(axis=chanDim))
model2.add(Conv2D(128, (3, 3), padding="same"))
model2.add(Activation("relu"))
model2.add(BatchNormalization(axis=chanDim))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
model2.add(Flatten())
model2.add(Dense(512))
model2.add(Activation("relu"))
model2.add(BatchNormalization())
model2.add(Dropout(0.5))
model2.add(Dense(128))
model2.add(Activation("relu"))
model2.add(BatchNormalization())
model2.add(Dropout(0.3))



# softmax classifier
model2.add(Dense(classes))
model2.add(Activation("softmax"))


model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 84, 84, 32)        896
_________________________________________________________________
activation_9 (Activation)    (None, 84, 84, 32)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 84, 84, 32)        128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 42, 42, 32)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 42, 42, 32)        0
_________________________________________________________________
```

```
conv2d_7 (Conv2D)            (None, 42, 42, 64)        18496     
_________________________________________________________________
activation_10 (Activation)   (None, 42, 42, 64)        0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 42, 42, 64)        256       
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 42, 42, 64)        36928     
_________________________________________________________________
activation_11 (Activation)   (None, 42, 42, 64)        0         
_________________________________________________________________
batch_normalization_10 (Batc (None, 42, 42, 64)        256       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 21, 21, 64)        0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 21, 21, 64)        0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 21, 21, 128)       73856     
_________________________________________________________________
activation_12 (Activation)   (None, 21, 21, 128)       0         
_________________________________________________________________
batch_normalization_11 (Batc (None, 21, 21, 128)       512       
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 21, 21, 128)       147584    
_________________________________________________________________
activation_13 (Activation)   (None, 21, 21, 128)       0         
_________________________________________________________________
batch_normalization_12 (Batc (None, 21, 21, 128)       512       
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 21, 21, 128)       147584    
_________________________________________________________________
activation_14 (Activation)   (None, 21, 21, 128)       0         
_________________________________________________________________
batch_normalization_13 (Batc (None, 21, 21, 128)       512       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 10, 10, 128)       0         
_________________________________________________________________
dropout_7 (Dropout)          (None, 10, 10, 128)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 12800)             0         
_________________________________________________________________
dense_3 (Dense)              (None, 512)               6554112   
_________________________________________________________________
activation_15 (Activation)   (None, 512)               0         
_________________________________________________________________
batch_normalization_14 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664     
_________________________________________________________________
activation_16 (Activation)   (None, 128)               0         
_________________________________________________________________
batch_normalization_15 (Batc (None, 128)               512       
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 22)                2838      
_________________________________________________________________
activation_17 (Activation)   (None, 22)                0         
=================================================================
Total params: 7,052,694
Trainable params: 7,050,326
Non-trainable params: 2,368
_________________________________________________________________
```

In [17]:

```
# initialize the model and optimizermodel2 = model()
model2.compile(loss="categorical_crossentropy", optimizer="RMSprop", metrics=["accuracy"
])
```

In [18]:

```
H2 = model2.fit(trainX, trainY, batch_size=BS,
    validation_data=(valX, valY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS)
```

```
Epoch 1/50
353/353 [==============================] - 8s 19ms/step - loss: 3.0339 - accuracy: 0.1776
- val_loss: 3.6086 - val_accuracy: 0.2000
Epoch 2/50
353/353 [==============================] - 6s 18ms/step - loss: 2.1988 - accuracy: 0.3553
- val_loss: 2.3803 - val_accuracy: 0.3491
Epoch 3/50
353/353 [==============================] - 6s 17ms/step - loss: 1.8960 - accuracy: 0.4481
- val_loss: 1.9390 - val_accuracy: 0.4269
Epoch 4/50
353/353 [==============================] - 6s 18ms/step - loss: 1.6920 - accuracy: 0.5030
- val_loss: 1.5539 - val_accuracy: 0.5265
Epoch 5/50
353/353 [==============================] - 6s 17ms/step - loss: 1.5140 - accuracy: 0.5409
- val_loss: 1.5667 - val_accuracy: 0.5484
Epoch 6/50
353/353 [==============================] - 6s 17ms/step - loss: 1.3652 - accuracy: 0.5932
- val_loss: 1.6742 - val_accuracy: 0.5272
Epoch 7/50
353/353 [==============================] - 6s 17ms/step - loss: 1.2352 - accuracy: 0.6344
- val_loss: 1.7707 - val_accuracy: 0.5102
Epoch 8/50
353/353 [==============================] - 6s 17ms/step - loss: 1.1184 - accuracy: 0.6701
- val_loss: 1.3241 - val_accuracy: 0.6028
Epoch 9/50
353/353 [==============================] - 6s 18ms/step - loss: 1.0125 - accuracy: 0.6903
- val_loss: 1.4464 - val_accuracy: 0.5901
Epoch 10/50
353/353 [==============================] - 6s 17ms/step - loss: 0.8838 - accuracy: 0.7342
- val_loss: 1.3798 - val_accuracy: 0.6127
Epoch 11/50
353/353 [==============================] - 6s 17ms/step - loss: 0.8107 - accuracy: 0.7518
- val_loss: 1.2968 - val_accuracy: 0.6318
Epoch 12/50
353/353 [==============================] - 6s 17ms/step - loss: 0.7206 - accuracy: 0.7782
- val_loss: 1.3880 - val_accuracy: 0.6332
Epoch 13/50
353/353 [==============================] - 6s 18ms/step - loss: 0.6246 - accuracy: 0.8064
- val_loss: 1.4591 - val_accuracy: 0.6078
Epoch 14/50
353/353 [==============================] - 6s 18ms/step - loss: 0.5388 - accuracy: 0.8344
- val_loss: 1.4453 - val_accuracy: 0.6233
Epoch 15/50
353/353 [==============================] - 6s 18ms/step - loss: 0.4752 - accuracy: 0.8525
- val_loss: 1.5222 - val_accuracy: 0.6170
Epoch 16/50
353/353 [==============================] - 6s 17ms/step - loss: 0.4373 - accuracy: 0.8618
- val_loss: 1.4639 - val_accuracy: 0.6325
Epoch 17/50
353/353 [==============================] - 6s 18ms/step - loss: 0.4035 - accuracy: 0.8710
- val_loss: 1.8003 - val_accuracy: 0.5562
Epoch 18/50
353/353 [==============================] - 6s 18ms/step - loss: 0.3736 - accuracy: 0.8832
- val_loss: 1.6134 - val_accuracy: 0.6042
Epoch 19/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3348 - accuracy: 0.8911
- val_loss: 1.7778 - val_accuracy: 0.6049
Epoch 20/50
353/353 [==============================] - 6s 18ms/step - loss: 0.3004 - accuracy: 0.9033
- val_loss: 1.7558 - val_accuracy: 0.6035
Epoch 21/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2952 - accuracy: 0.9044
- val_loss: 1.6414 - val_accuracy: 0.6198
Epoch 22/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2863 - accuracy: 0.9121
- val_loss: 1.8720 - val_accuracy: 0.6007
Epoch 23/50
```

```
353/353 [==============================] - 6s 17ms/step - loss: 0.2684 - accuracy: 0.9098
- val_loss: 1.8393 - val_accuracy: 0.6035
Epoch 24/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2549 - accuracy: 0.9216
- val_loss: 1.6995 - val_accuracy: 0.6269
Epoch 25/50
353/353 [==============================] - 6s 18ms/step - loss: 0.2233 - accuracy: 0.9268
- val_loss: 1.8961 - val_accuracy: 0.6014
Epoch 26/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2162 - accuracy: 0.9321
- val_loss: 1.6940 - val_accuracy: 0.6346
Epoch 27/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2139 - accuracy: 0.9295
- val_loss: 1.7730 - val_accuracy: 0.6261
Epoch 28/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1953 - accuracy: 0.9340
- val_loss: 1.9871 - val_accuracy: 0.5689
Epoch 29/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1907 - accuracy: 0.9394
- val_loss: 1.8544 - val_accuracy: 0.6148
Epoch 30/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2016 - accuracy: 0.9375
- val_loss: 1.8612 - val_accuracy: 0.6191
Epoch 31/50
353/353 [==============================] - 6s 18ms/step - loss: 0.1871 - accuracy: 0.9402
- val_loss: 1.8953 - val_accuracy: 0.6113
Epoch 32/50
353/353 [==============================] - 6s 18ms/step - loss: 0.1917 - accuracy: 0.9393
- val_loss: 2.0313 - val_accuracy: 0.6099
Epoch 33/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1762 - accuracy: 0.9440
- val_loss: 2.4494 - val_accuracy: 0.5604
Epoch 34/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1646 - accuracy: 0.9458
- val_loss: 1.9310 - val_accuracy: 0.6113
Epoch 35/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1500 - accuracy: 0.9494
- val_loss: 1.7899 - val_accuracy: 0.6226
Epoch 36/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1483 - accuracy: 0.9487
- val_loss: 1.9512 - val_accuracy: 0.6042
Epoch 37/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1640 - accuracy: 0.9491
- val_loss: 2.4294 - val_accuracy: 0.5208
Epoch 38/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1413 - accuracy: 0.9541
- val_loss: 1.9309 - val_accuracy: 0.6198
Epoch 39/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1524 - accuracy: 0.9521
- val_loss: 2.2494 - val_accuracy: 0.5767
Epoch 40/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1277 - accuracy: 0.9591
- val_loss: 2.0878 - val_accuracy: 0.5972
Epoch 41/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1325 - accuracy: 0.9544
- val_loss: 2.0889 - val_accuracy: 0.6205
Epoch 42/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1406 - accuracy: 0.9535
- val_loss: 2.0477 - val_accuracy: 0.6155
Epoch 43/50
353/353 [==============================] - 6s 18ms/step - loss: 0.1209 - accuracy: 0.9621
- val_loss: 1.9550 - val_accuracy: 0.6134
Epoch 44/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1349 - accuracy: 0.9579
- val_loss: 2.1300 - val_accuracy: 0.5880
Epoch 45/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1364 - accuracy: 0.9575
- val_loss: 1.9820 - val_accuracy: 0.6071
Epoch 46/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1093 - accuracy: 0.9646
- val_loss: 2.1836 - val_accuracy: 0.5852
Epoch 47/50
```

```
353/353 [==============================] - 6s 17ms/step - loss: 0.1220 - accuracy: 0.9608
- val_loss: 2.0208 - val_accuracy: 0.6276
Epoch 48/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1191 - accuracy: 0.9573
- val_loss: 2.1534 - val_accuracy: 0.6127
Epoch 49/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1134 - accuracy: 0.9649
- val_loss: 2.2334 - val_accuracy: 0.6014
Epoch 50/50
353/353 [==============================] - 6s 17ms/step - loss: 0.1105 - accuracy: 0.9651
- val_loss: 2.1490 - val_accuracy: 0.6247
```

In [19]:

```
preds2=model2.evaluate(testX, testY, batch_size=32)
print('loss = '+str(preds2[0]))
print('test accuracy = '+str(preds2[1]))
```

```
45/45 [==============================] - 0s 6ms/step - loss: 2.0817 - accuracy: 0.6124
loss = 2.081742286682129
test accuracy = 0.6124469637870789
```

We got 62.02% test accuracy using 'RMSprop' optimizer.

In [20]:

```
# evaluate the network
print("[INFO] evaluating network...")
predictions2 = model2.predict(valX, batch_size=32)
print(classification_report(valY.argmax(axis=1),
    predictions2.argmax(axis=1), target_names=lb.classes_))
```

```
[INFO] evaluating network...
                precision    recall  f1-score   support

     badminton       0.49      0.59      0.54        79
      baseball       0.90      0.63      0.74        89
    basketball       0.64      0.37      0.47        49
        boxing       0.66      0.68      0.67        77
         chess       0.51      0.47      0.49        49
       cricket       0.73      0.61      0.66        74
       fencing       0.56      0.69      0.62        65
      football       0.60      0.69      0.64        77
      formula1       0.61      0.68      0.64        68
    gymnastics       0.49      0.54      0.51        69
        hockey       0.52      0.36      0.42        45
    ice_hockey       0.87      0.79      0.83        76
       kabaddi       0.88      0.62      0.73        37
        motogp       0.77      0.81      0.79        62
      shooting       0.45      0.62      0.53        40
      swimming       0.96      0.76      0.85        68
  table_tennis       0.60      0.55      0.58        74
        tennis       0.58      0.50      0.54        84
    volleyball       0.67      0.63      0.65        76
weight_lifting       0.43      0.57      0.49        51
     wrestling       0.46      0.60      0.52        48
           wwe       0.57      0.81      0.67        58

      accuracy                           0.62      1415
     macro avg       0.64      0.62      0.62      1415
  weighted avg       0.65      0.62      0.63      1415
```

In [21]:

```
# plot the training loss and accuracy for model 2
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure(figsize=(8,8))
plt.plot(N, H2.history["loss"], label="train_loss")
```

```
plt.plot(N, H2.history["val_loss"], label="val_loss")
plt.plot(N, H2.history["accuracy"], label="train_acc")
plt.plot(N, H2.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy (SmallVGGNet)")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig("smallvggnet_plot.png")
plt.show()
```

Still we can observe high overfitting. So I want to try another optimizer.

MODEL 3

In [22]:

```
chanDim=3
model3 = Sequential()
# CONV => RELU => POOL layer set
model3.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model3.add(Activation("relu"))
model3.add(BatchNormalization(axis=chanDim))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

# (CONV => RELU) * 2 => POOL layer set
model3.add(Conv2D(64, (3, 3), padding="same"))
model3.add(Activation("relu"))
model3.add(BatchNormalization(axis=chanDim))
model3.add(Conv2D(64, (3, 3), padding="same"))
model3.add(Activation("relu"))
model3.add(BatchNormalization(axis=chanDim))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

# (CONV => RELU) * 3 => POOL layer set
model3.add(Conv2D(128, (3, 3), padding="same"))
model3.add(Activation("relu"))
```

```
model3.add(BatchNormalization(axis=chanDim))
model3.add(Conv2D(128, (3, 3), padding="same"))
model3.add(Activation("relu"))
model3.add(BatchNormalization(axis=chanDim))
model3.add(Conv2D(128, (3, 3), padding="same"))
model3.add(Activation("relu"))
model3.add(BatchNormalization(axis=chanDim))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
model3.add(Flatten())
model3.add(Dense(512))
model3.add(Activation("relu"))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
model3.add(Dense(128))
model3.add(Activation("relu"))
model3.add(BatchNormalization())
model3.add(Dropout(0.3))

# softmax classifier
model3.add(Dense(classes))
model3.add(Activation("softmax"))


model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 84, 84, 32)        896
_________________________________________________________________
activation_18 (Activation)   (None, 84, 84, 32)        0
_________________________________________________________________
batch_normalization_16 (Batc (None, 84, 84, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 42, 42, 32)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 42, 42, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 42, 42, 64)        18496
_________________________________________________________________
activation_19 (Activation)   (None, 42, 42, 64)        0
_________________________________________________________________
batch_normalization_17 (Batc (None, 42, 42, 64)        256
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 42, 42, 64)        36928
_________________________________________________________________
activation_20 (Activation)   (None, 42, 42, 64)        0
_________________________________________________________________
batch_normalization_18 (Batc (None, 42, 42, 64)        256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 21, 21, 64)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 21, 21, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 21, 21, 128)       73856
_________________________________________________________________
activation_21 (Activation)   (None, 21, 21, 128)       0
_________________________________________________________________
batch_normalization_19 (Batc (None, 21, 21, 128)       512
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 21, 21, 128)       147584
_________________________________________________________________
activation_22 (Activation)   (None, 21, 21, 128)       0
_________________________________________________________________
batch_normalization_20 (Batc (None, 21, 21, 128)       512
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 21, 21, 128)       147584
_________________________________________________________________
```

```
activation_23 (Activation)   (None, 21, 21, 128)      0
_________________________________________________________________
batch_normalization_21 (Batc (None, 21, 21, 128)      512
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 10, 10, 128)      0
_________________________________________________________________
dropout_12 (Dropout)         (None, 10, 10, 128)      0
_________________________________________________________________
flatten_2 (Flatten)          (None, 12800)            0
_________________________________________________________________
dense_6 (Dense)              (None, 512)              6554112
_________________________________________________________________
activation_24 (Activation)   (None, 512)              0
_________________________________________________________________
batch_normalization_22 (Batc (None, 512)              2048
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)              65664
_________________________________________________________________
activation_25 (Activation)   (None, 128)              0
_________________________________________________________________
batch_normalization_23 (Batc (None, 128)              512
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)              0
_________________________________________________________________
dense_8 (Dense)              (None, 22)               2838
_________________________________________________________________
activation_26 (Activation)   (None, 22)               0
=================================================================
Total params: 7,052,694
Trainable params: 7,050,326
Non-trainable params: 2,368
_________________________________________________________________
```

In [23]:

```
# initialize the model and optimizermodel2 = model()
model3.compile(loss="categorical_crossentropy", optimizer="adamax", metrics=["accuracy"]
)
```

In [24]:

```
H3 = model3.fit(trainX, trainY, batch_size=BS,
    validation_data=(valX, valY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS)
```

```
Epoch 1/50
353/353 [==============================] - 7s 17ms/step - loss: 3.1226 - accuracy: 0.1811
- val_loss: 3.3959 - val_accuracy: 0.0862
Epoch 2/50
353/353 [==============================] - 6s 16ms/step - loss: 2.3298 - accuracy: 0.3215
- val_loss: 2.3043 - val_accuracy: 0.3512
Epoch 3/50
353/353 [==============================] - 6s 16ms/step - loss: 2.1021 - accuracy: 0.3775
- val_loss: 1.9625 - val_accuracy: 0.4205
Epoch 4/50
353/353 [==============================] - 6s 16ms/step - loss: 1.9147 - accuracy: 0.4309
- val_loss: 1.9648 - val_accuracy: 0.4509
Epoch 5/50
353/353 [==============================] - 6s 16ms/step - loss: 1.8126 - accuracy: 0.4569
- val_loss: 1.7927 - val_accuracy: 0.4792
Epoch 6/50
353/353 [==============================] - 6s 16ms/step - loss: 1.7201 - accuracy: 0.4835
- val_loss: 1.6636 - val_accuracy: 0.5208
Epoch 7/50
353/353 [==============================] - 6s 16ms/step - loss: 1.6107 - accuracy: 0.5114
- val_loss: 1.5944 - val_accuracy: 0.5449
Epoch 8/50
353/353 [==============================] - 6s 16ms/step - loss: 1.4525 - accuracy: 0.5641
- val_loss: 1.6243 - val_accuracy: 0.5102
Epoch 9/50
```

```
353/353 [==============================] - 6s 16ms/step - loss: 1.3877 - accuracy: 0.5792
- val_loss: 1.5798 - val_accuracy: 0.5519
Epoch 10/50
353/353 [==============================] - 6s 16ms/step - loss: 1.2927 - accuracy: 0.6168
- val_loss: 1.5157 - val_accuracy: 0.5703
Epoch 11/50
353/353 [==============================] - 6s 16ms/step - loss: 1.1721 - accuracy: 0.6446
- val_loss: 1.5469 - val_accuracy: 0.5590
Epoch 12/50
353/353 [==============================] - 6s 16ms/step - loss: 1.1009 - accuracy: 0.6660
- val_loss: 1.6210 - val_accuracy: 0.5385
Epoch 13/50
353/353 [==============================] - 6s 16ms/step - loss: 1.0327 - accuracy: 0.6850
- val_loss: 1.5041 - val_accuracy: 0.5802
Epoch 14/50
353/353 [==============================] - 6s 16ms/step - loss: 0.9343 - accuracy: 0.7103
- val_loss: 1.4747 - val_accuracy: 0.5823
Epoch 15/50
353/353 [==============================] - 6s 16ms/step - loss: 0.8620 - accuracy: 0.7380
- val_loss: 1.4820 - val_accuracy: 0.5901
Epoch 16/50
353/353 [==============================] - 6s 16ms/step - loss: 0.7757 - accuracy: 0.7611
- val_loss: 1.4791 - val_accuracy: 0.5972
Epoch 17/50
353/353 [==============================] - 6s 16ms/step - loss: 0.7238 - accuracy: 0.7801
- val_loss: 1.4401 - val_accuracy: 0.6028
Epoch 18/50
353/353 [==============================] - 6s 16ms/step - loss: 0.6746 - accuracy: 0.7932
- val_loss: 1.5149 - val_accuracy: 0.6078
Epoch 19/50
353/353 [==============================] - 6s 16ms/step - loss: 0.6044 - accuracy: 0.8100
- val_loss: 1.4631 - val_accuracy: 0.6155
Epoch 20/50
353/353 [==============================] - 6s 17ms/step - loss: 0.5295 - accuracy: 0.8428
- val_loss: 1.4680 - val_accuracy: 0.6205
Epoch 21/50
353/353 [==============================] - 6s 16ms/step - loss: 0.4925 - accuracy: 0.8463
- val_loss: 1.5347 - val_accuracy: 0.6148
Epoch 22/50
353/353 [==============================] - 6s 16ms/step - loss: 0.4750 - accuracy: 0.8476
- val_loss: 1.5065 - val_accuracy: 0.6191
Epoch 23/50
353/353 [==============================] - 6s 16ms/step - loss: 0.4127 - accuracy: 0.8702
- val_loss: 1.5191 - val_accuracy: 0.6304
Epoch 24/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3736 - accuracy: 0.8861
- val_loss: 1.6796 - val_accuracy: 0.5943
Epoch 25/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3543 - accuracy: 0.8896
- val_loss: 1.5925 - val_accuracy: 0.6212
Epoch 26/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3410 - accuracy: 0.8934
- val_loss: 1.6016 - val_accuracy: 0.6085
Epoch 27/50
353/353 [==============================] - 6s 16ms/step - loss: 0.3064 - accuracy: 0.9017
- val_loss: 1.6877 - val_accuracy: 0.6092
Epoch 28/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2919 - accuracy: 0.9102
- val_loss: 1.8145 - val_accuracy: 0.5809
Epoch 29/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2578 - accuracy: 0.9183
- val_loss: 1.8201 - val_accuracy: 0.6000
Epoch 30/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2459 - accuracy: 0.9205
- val_loss: 1.7075 - val_accuracy: 0.6155
Epoch 31/50
353/353 [==============================] - 6s 16ms/step - loss: 0.2164 - accuracy: 0.9300
- val_loss: 1.6813 - val_accuracy: 0.6269
Epoch 32/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2120 - accuracy: 0.9329
- val_loss: 1.7258 - val_accuracy: 0.6240
Epoch 33/50
```

```
353/353 [==============================] - 6s 16ms/step - loss: 0.2074 - accuracy: 0.9322
- val_loss: 1.9025 - val_accuracy: 0.5802
Epoch 34/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1916 - accuracy: 0.9362
- val_loss: 1.9438 - val_accuracy: 0.5901
Epoch 35/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1720 - accuracy: 0.9467
- val_loss: 1.7941 - val_accuracy: 0.6092
Epoch 36/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1826 - accuracy: 0.9425
- val_loss: 1.8942 - val_accuracy: 0.6085
Epoch 37/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1546 - accuracy: 0.9510
- val_loss: 1.8726 - val_accuracy: 0.6205
Epoch 38/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1654 - accuracy: 0.9518
- val_loss: 1.8174 - val_accuracy: 0.6184
Epoch 39/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1641 - accuracy: 0.9458
- val_loss: 1.7618 - val_accuracy: 0.6148
Epoch 40/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1527 - accuracy: 0.9505
- val_loss: 1.8944 - val_accuracy: 0.6021
Epoch 41/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1467 - accuracy: 0.9523
- val_loss: 1.9252 - val_accuracy: 0.6120
Epoch 42/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1557 - accuracy: 0.9501
- val_loss: 1.8154 - val_accuracy: 0.6177
Epoch 43/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1396 - accuracy: 0.9569
- val_loss: 1.8789 - val_accuracy: 0.6148
Epoch 44/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1358 - accuracy: 0.9530
- val_loss: 1.8381 - val_accuracy: 0.6141
Epoch 45/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1261 - accuracy: 0.9593
- val_loss: 1.8937 - val_accuracy: 0.6205
Epoch 46/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1258 - accuracy: 0.9608
- val_loss: 2.0122 - val_accuracy: 0.6014
Epoch 47/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1312 - accuracy: 0.9580
- val_loss: 2.0802 - val_accuracy: 0.5943
Epoch 48/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1038 - accuracy: 0.9642
- val_loss: 1.9475 - val_accuracy: 0.6177
Epoch 49/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1126 - accuracy: 0.9642
- val_loss: 1.9367 - val_accuracy: 0.6276
Epoch 50/50
353/353 [==============================] - 6s 16ms/step - loss: 0.1009 - accuracy: 0.9684
- val_loss: 2.0242 - val_accuracy: 0.6191
```

In [25]:

```
preds3=model3.evaluate(testX, testY, batch_size=32)
print('loss = '+str(preds3[0]))
print('test accuracy = '+str(preds3[1]))
```

```
45/45 [==============================] - 0s 6ms/step - loss: 1.8782 - accuracy: 0.6068
loss = 1.8782297372817993
test accuracy = 0.606789231300354
```

We got 64.28% test accuracy using 'Adamax' optimizer.

In [26]:

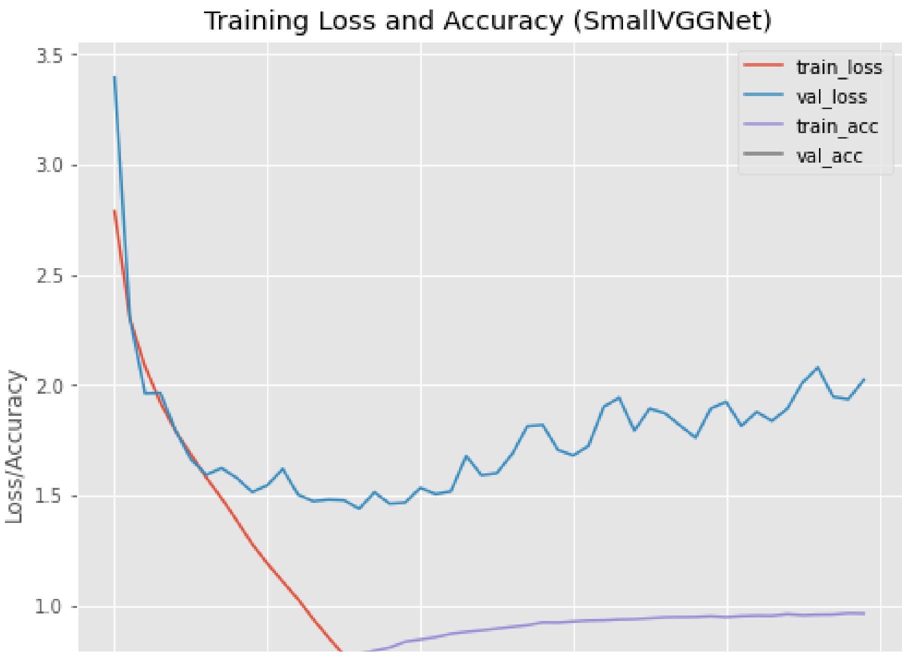
```
# evaluate the network
print("[INFO] evaluating network...")
predictions3 = model3.predict(valX, batch_size=32)
```

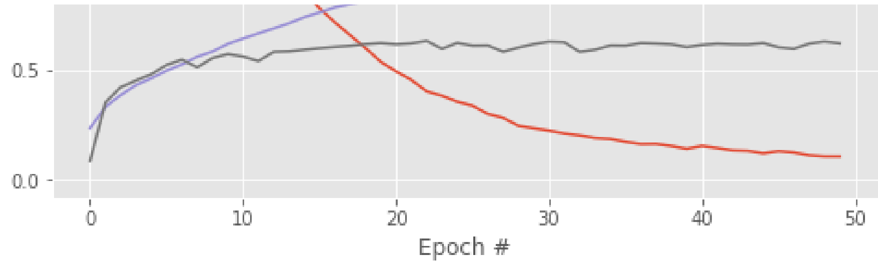
```
print(classification_report(valY.argmax(axis=1),
    predictions3.argmax(axis=1), target_names=lb.classes_))
```

```
[INFO] evaluating network...
                precision    recall  f1-score   support

     badminton       0.61      0.54      0.58        79
      baseball       0.75      0.67      0.71        89
    basketball       0.48      0.47      0.47        49
        boxing       0.66      0.58      0.62        77
         chess       0.47      0.31      0.37        49
       cricket       0.65      0.72      0.68        74
       fencing       0.64      0.63      0.64        65
      football       0.76      0.57      0.65        77
      formula1       0.77      0.63      0.69        68
    gymnastics       0.40      0.55      0.46        69
        hockey       0.49      0.38      0.42        45
    ice_hockey       0.82      0.76      0.79        76
       kabaddi       0.66      0.68      0.67        37
        motogp       0.82      0.79      0.80        62
      shooting       0.47      0.53      0.49        40
      swimming       0.95      0.88      0.92        68
  table_tennis       0.60      0.58      0.59        74
        tennis       0.49      0.57      0.53        84
    volleyball       0.59      0.64      0.62        76
weight_lifting       0.48      0.61      0.54        51
     wrestling       0.42      0.58      0.49        48
           wwe       0.61      0.72      0.66        58

      accuracy                           0.62      1415
     macro avg       0.62      0.61      0.61      1415
  weighted avg       0.63      0.62      0.62      1415
```

In [27]:

```
# plot the training loss and accuracy for model 2
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure(figsize=(8,8))
plt.plot(N, H3.history["loss"], label="train_loss")
plt.plot(N, H3.history["val_loss"], label="val_loss")
plt.plot(N, H3.history["accuracy"], label="train_acc")
plt.plot(N, H3.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy (SmallVGGNet)")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig("smallvggnet_plot.png")
plt.show()
```





Still we are facing the problem of overfitting. Now we will try some other regularization like adding or increasing Dropouts.

MODEL 4

In [28]:

```
chanDim=3
model4 = Sequential()
# CONV => RELU => POOL layer set
model4.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model4.add(Activation("relu"))
model4.add(BatchNormalization(axis=chanDim))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.5))

# (CONV => RELU) * 2 => POOL layer set
model4.add(Conv2D(64, (3, 3), padding="same"))
model4.add(Activation("relu"))
model4.add(BatchNormalization(axis=chanDim))
model4.add(Conv2D(64, (3, 3), padding="same"))
model4.add(Activation("relu"))
model4.add(BatchNormalization(axis=chanDim))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.5))

# (CONV => RELU) * 3 => POOL layer set
model4.add(Conv2D(128, (3, 3), padding="same"))
model4.add(Activation("relu"))
model4.add(BatchNormalization(axis=chanDim))
model4.add(Conv2D(128, (3, 3), padding="same"))
model4.add(Activation("relu"))
model4.add(BatchNormalization(axis=chanDim))
model4.add(Conv2D(128, (3, 3), padding="same"))
model4.add(Activation("relu"))
model4.add(BatchNormalization(axis=chanDim))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.5))

# first (and only) set of FC => RELU layers
model4.add(Flatten())
model4.add(Dense(512))
model4.add(Activation("relu"))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))
model4.add(Dense(128))
model4.add(Activation("relu"))
model4.add(BatchNormalization())
model4.add(Dropout(0.3))

# softmax classifier
model4.add(Dense(classes))
model4.add(Activation("softmax"))


model4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 84, 84, 32)        896
_________________________________________________________________
activation_27 (Activation)   (None, 84, 84, 32)        0
_________________________________________________________________
batch_normalization_24 (Batc (None, 84, 84, 32)        128
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 42, 42, 32)        0
_________________________________________________________________
dropout_15 (Dropout)         (None, 42, 42, 32)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 42, 42, 64)        18496
_________________________________________________________________
activation_28 (Activation)   (None, 42, 42, 64)        0
_________________________________________________________________
batch_normalization_25 (Batc (None, 42, 42, 64)        256
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 42, 42, 64)        36928
_________________________________________________________________
activation_29 (Activation)   (None, 42, 42, 64)        0
_________________________________________________________________
batch_normalization_26 (Batc (None, 42, 42, 64)        256
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 21, 21, 64)        0
_________________________________________________________________
dropout_16 (Dropout)         (None, 21, 21, 64)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 21, 21, 128)       73856
_________________________________________________________________
activation_30 (Activation)   (None, 21, 21, 128)       0
_________________________________________________________________
batch_normalization_27 (Batc (None, 21, 21, 128)       512
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 21, 21, 128)       147584
_________________________________________________________________
activation_31 (Activation)   (None, 21, 21, 128)       0
_________________________________________________________________
batch_normalization_28 (Batc (None, 21, 21, 128)       512
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 21, 21, 128)       147584
_________________________________________________________________
activation_32 (Activation)   (None, 21, 21, 128)       0
_________________________________________________________________
batch_normalization_29 (Batc (None, 21, 21, 128)       512
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 10, 10, 128)       0
_________________________________________________________________
dropout_17 (Dropout)         (None, 10, 10, 128)       0
_________________________________________________________________
flatten_3 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               6554112
_________________________________________________________________
activation_33 (Activation)   (None, 512)               0
_________________________________________________________________
batch_normalization_30 (Batc (None, 512)               2048
_________________________________________________________________
dropout_18 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               65664
_________________________________________________________________
activation_34 (Activation)   (None, 128)               0
_________________________________________________________________
batch_normalization_31 (Batc (None, 128)               512
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 22)                2838
_________________________________________________________________
activation_35 (Activation)   (None, 22)                0
=================================================================
Total params: 7,052,694
```

```
Total params: 7,052,694
Trainable params: 7,050,326
Non-trainable params: 2,368
_________________________________________________________________
```

In [29]:

```python
# initialize the model and optimizermodel2 = model()
model4.compile(loss="categorical_crossentropy", optimizer="RMSprop", metrics=["accuracy"
])
```

In [30]:

```python
H4 = model4.fit(trainX, trainY, batch_size=BS,
    validation_data=(valX, valY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS)
```

```
Epoch 1/50
353/353 [==============================] - 9s 19ms/step - loss: 3.1843 - accuracy: 0.1609
- val_loss: 9.0867 - val_accuracy: 0.1343
Epoch 2/50
353/353 [==============================] - 6s 17ms/step - loss: 2.2586 - accuracy: 0.3302
- val_loss: 2.1115 - val_accuracy: 0.4191
Epoch 3/50
353/353 [==============================] - 6s 17ms/step - loss: 1.9323 - accuracy: 0.4227
- val_loss: 2.1673 - val_accuracy: 0.3965
Epoch 4/50
353/353 [==============================] - 6s 18ms/step - loss: 1.7679 - accuracy: 0.4804
- val_loss: 1.6807 - val_accuracy: 0.5018
Epoch 5/50
353/353 [==============================] - 6s 17ms/step - loss: 1.6160 - accuracy: 0.5168
- val_loss: 1.7004 - val_accuracy: 0.5102
Epoch 6/50
353/353 [==============================] - 6s 17ms/step - loss: 1.4727 - accuracy: 0.5509
- val_loss: 1.6524 - val_accuracy: 0.5244
Epoch 7/50
353/353 [==============================] - 6s 17ms/step - loss: 1.4192 - accuracy: 0.5745
- val_loss: 1.4335 - val_accuracy: 0.5767
Epoch 8/50
353/353 [==============================] - 6s 17ms/step - loss: 1.3018 - accuracy: 0.6128
- val_loss: 1.5694 - val_accuracy: 0.5590
Epoch 9/50
353/353 [==============================] - 6s 17ms/step - loss: 1.2145 - accuracy: 0.6337
- val_loss: 1.3581 - val_accuracy: 0.6071
Epoch 10/50
353/353 [==============================] - 6s 18ms/step - loss: 1.1478 - accuracy: 0.6524
- val_loss: 1.3677 - val_accuracy: 0.6049
Epoch 11/50
353/353 [==============================] - 6s 17ms/step - loss: 1.0633 - accuracy: 0.6782
- val_loss: 1.3953 - val_accuracy: 0.6057
Epoch 12/50
353/353 [==============================] - 6s 17ms/step - loss: 0.9738 - accuracy: 0.7058
- val_loss: 1.4877 - val_accuracy: 0.5880
Epoch 13/50
353/353 [==============================] - 6s 17ms/step - loss: 0.9200 - accuracy: 0.7149
- val_loss: 1.4292 - val_accuracy: 0.6057
Epoch 14/50
353/353 [==============================] - 6s 17ms/step - loss: 0.8719 - accuracy: 0.7337
- val_loss: 1.5141 - val_accuracy: 0.5880
Epoch 15/50
353/353 [==============================] - 6s 17ms/step - loss: 0.8292 - accuracy: 0.7465
- val_loss: 1.3110 - val_accuracy: 0.6360
Epoch 16/50
353/353 [==============================] - 6s 17ms/step - loss: 0.7702 - accuracy: 0.7663
- val_loss: 1.3723 - val_accuracy: 0.6353
Epoch 17/50
353/353 [==============================] - 6s 17ms/step - loss: 0.7242 - accuracy: 0.7727
- val_loss: 1.6628 - val_accuracy: 0.5724
Epoch 18/50
353/353 [==============================] - 6s 17ms/step - loss: 0.6675 - accuracy: 0.7935
- val_loss: 1.4307 - val_accuracy: 0.6078
Epoch 19/50
353/353 [==============================] - 6s 17ms/step - loss: 0.6536 - accuracy: 0.7982
```

```
353/353 [==============================] - 6s 17ms/step - loss: 0.6536 - accuracy: 0.7982
- val_loss: 1.4706 - val_accuracy: 0.6276
Epoch 20/50
353/353 [==============================] - 6s 17ms/step - loss: 0.5957 - accuracy: 0.8125
- val_loss: 1.6887 - val_accuracy: 0.6021
Epoch 21/50
353/353 [==============================] - 6s 17ms/step - loss: 0.5694 - accuracy: 0.8217
- val_loss: 1.4682 - val_accuracy: 0.6325
Epoch 22/50
353/353 [==============================] - 6s 18ms/step - loss: 0.5350 - accuracy: 0.8278
- val_loss: 1.7454 - val_accuracy: 0.5845
Epoch 23/50
353/353 [==============================] - 6s 18ms/step - loss: 0.5222 - accuracy: 0.8359
- val_loss: 1.5360 - val_accuracy: 0.6035
Epoch 24/50
353/353 [==============================] - 6s 17ms/step - loss: 0.4992 - accuracy: 0.8372
- val_loss: 1.4726 - val_accuracy: 0.6382
Epoch 25/50
353/353 [==============================] - 6s 17ms/step - loss: 0.4677 - accuracy: 0.8548
- val_loss: 1.5751 - val_accuracy: 0.6021
Epoch 26/50
353/353 [==============================] - 6s 17ms/step - loss: 0.4475 - accuracy: 0.8599
- val_loss: 1.5934 - val_accuracy: 0.6276
Epoch 27/50
353/353 [==============================] - 6s 18ms/step - loss: 0.4395 - accuracy: 0.8612
- val_loss: 1.7088 - val_accuracy: 0.6106
Epoch 28/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3999 - accuracy: 0.8723
- val_loss: 1.4430 - val_accuracy: 0.6417
Epoch 29/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3987 - accuracy: 0.8694
- val_loss: 1.6036 - val_accuracy: 0.6445
Epoch 30/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3795 - accuracy: 0.8785
- val_loss: 1.5338 - val_accuracy: 0.6445
Epoch 31/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3795 - accuracy: 0.8790
- val_loss: 1.6039 - val_accuracy: 0.6261
Epoch 32/50
353/353 [==============================] - 6s 18ms/step - loss: 0.3706 - accuracy: 0.8789
- val_loss: 1.6397 - val_accuracy: 0.6283
Epoch 33/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3496 - accuracy: 0.8874
- val_loss: 1.5531 - val_accuracy: 0.6311
Epoch 34/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3402 - accuracy: 0.8900
- val_loss: 1.6175 - val_accuracy: 0.6346
Epoch 35/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3330 - accuracy: 0.8956
- val_loss: 1.8492 - val_accuracy: 0.6042
Epoch 36/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3268 - accuracy: 0.8935
- val_loss: 1.7163 - val_accuracy: 0.6325
Epoch 37/50
353/353 [==============================] - 6s 18ms/step - loss: 0.3097 - accuracy: 0.9016
- val_loss: 1.7128 - val_accuracy: 0.6495
Epoch 38/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3127 - accuracy: 0.9023
- val_loss: 1.4746 - val_accuracy: 0.6721
Epoch 39/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3033 - accuracy: 0.9067
- val_loss: 1.7576 - val_accuracy: 0.6304
Epoch 40/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3053 - accuracy: 0.9017
- val_loss: 1.7236 - val_accuracy: 0.6438
Epoch 41/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2758 - accuracy: 0.9103
- val_loss: 1.7491 - val_accuracy: 0.6332
Epoch 42/50
353/353 [==============================] - 6s 17ms/step - loss: 0.3059 - accuracy: 0.9006
- val_loss: 1.5666 - val_accuracy: 0.6516
Epoch 43/50
353/353 [==============================] - 6s 18ms/step - loss: 0.2691 - accuracy: 0.9158
```

```
353/353 [==============================] - 6s 18ms/step - loss: 0.2691 - accuracy: 0.9158
- val_loss: 1.6208 - val_accuracy: 0.6459
Epoch 44/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2611 - accuracy: 0.9180
- val_loss: 1.6277 - val_accuracy: 0.6629
Epoch 45/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2529 - accuracy: 0.9166
- val_loss: 1.7146 - val_accuracy: 0.6565
Epoch 46/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2344 - accuracy: 0.9242
- val_loss: 1.6766 - val_accuracy: 0.6572
Epoch 47/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2521 - accuracy: 0.9220
- val_loss: 1.8954 - val_accuracy: 0.6163
Epoch 48/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2409 - accuracy: 0.9188
- val_loss: 1.6561 - val_accuracy: 0.6516
Epoch 49/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2443 - accuracy: 0.9213
- val_loss: 1.7629 - val_accuracy: 0.6396
Epoch 50/50
353/353 [==============================] - 6s 17ms/step - loss: 0.2359 - accuracy: 0.9244
- val_loss: 1.6794 - val_accuracy: 0.6622
```

In [31]:

```
preds4=model4.evaluate(testX, testY, batch_size=32)
print('loss = '+str(preds4[0]))
print('test accuracy = '+str(preds4[1]))
```

```
45/45 [==============================] - 0s 6ms/step - loss: 1.6740 - accuracy: 0.6436
loss = 1.67401921749115
test accuracy = 0.6435643434524536
```

We got 63.43% test accuracy when we increased the dropouts.

In [32]:

```
# evaluate the network
print("[INFO] evaluating network...")
predictions4 = model4.predict(valX, batch_size=32)
print(classification_report(valY.argmax(axis=1),
    predictions4.argmax(axis=1), target_names=lb.classes_))
```

```
[INFO] evaluating network...
               precision    recall  f1-score   support

     badminton       0.58      0.63      0.61        79
      baseball       0.78      0.70      0.73        89
    basketball       0.66      0.55      0.60        49
        boxing       0.82      0.64      0.72        77
         chess       0.52      0.51      0.52        49
       cricket       0.71      0.69      0.70        74
       fencing       0.62      0.74      0.67        65
      football       0.66      0.78      0.71        77
      formula1       0.64      0.60      0.62        68
    gymnastics       0.62      0.49      0.55        69
        hockey       0.57      0.51      0.54        45
    ice_hockey       0.85      0.80      0.82        76
       kabaddi       0.78      0.68      0.72        37
        motogp       0.79      0.81      0.80        62
      shooting       0.52      0.70      0.60        40
      swimming       0.77      0.94      0.85        68
  table_tennis       0.68      0.49      0.57        74
        tennis       0.61      0.60      0.60        84
    volleyball       0.57      0.75      0.65        76
weight_lifting       0.58      0.49      0.53        51
     wrestling       0.53      0.54      0.54        48
           wwe       0.65      0.78      0.71        58

      accuracy                           0.66      1415
     macro avg       0.66      0.65      0.65      1415
```

```
weighted avg       0.67      0.66      0.66      1415
```

In [33]:

```
# plot the training loss and accuracy for model 2
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure(figsize=(8,8))
plt.plot(N, H4.history["loss"], label="train_loss")
plt.plot(N, H4.history["val_loss"], label="val_loss")
plt.plot(N, H4.history["accuracy"], label="train_acc")
plt.plot(N, H4.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy (SmallVGGNet)")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig("smallvggnet_plot.png")
plt.show()
```

We can observe overfitting is reduced but we are loosing accuracy at the same time. To tackle both these problems together we will increase our training data. We will do so by using Data Augmentation.

MODEL 5

In [34]:

```
# Data Augmentation
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True, fill_mode="nearest")
```

In [35]:

```
chanDim=3
model5 = Sequential()
# CONV => RELU => POOL layer set
```

```
model5.add(Conv2D(64, (3, 3), padding="same", input_shape=inputShape))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(Conv2D(64, (3, 3), padding="same"))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(MaxPooling2D(pool_size=(2, 2)))
model5.add(Dropout(0.25))

# (CONV => RELU) * 2 => POOL layer set
model5.add(Conv2D(128, (3, 3), padding="same"))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(Conv2D(128, (3, 3), padding="same"))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(MaxPooling2D(pool_size=(2, 2)))
model5.add(Dropout(0.3))

# (CONV => RELU) * 3 => POOL layer set
model5.add(Conv2D(256, (3, 3), padding="same"))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(Conv2D(256, (3, 3), padding="same"))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(Conv2D(256, (3, 3), padding="same"))
model5.add(Activation("relu"))
model5.add(BatchNormalization(axis=chanDim))
model5.add(MaxPooling2D(pool_size=(2, 2)))
model5.add(Dropout(0.5))

# first (and only) set of FC => RELU layers
model5.add(Flatten())
model5.add(Dense(512))
model5.add(Activation("relu"))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))
model5.add(Dense(128))
model5.add(Activation("relu"))
model5.add(BatchNormalization())
model5.add(Dropout(0.3))

# softmax classifier
model5.add(Dense(classes))
model5.add(Activation("softmax"))


model5.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_24 (Conv2D)           (None, 84, 84, 64)        1792      
_________________________________________________________________
activation_36 (Activation)   (None, 84, 84, 64)        0         
_________________________________________________________________
batch_normalization_32 (Batc (None, 84, 84, 64)        256       
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 84, 84, 64)        36928     
_________________________________________________________________
activation_37 (Activation)   (None, 84, 84, 64)        0         
_________________________________________________________________
batch_normalization_33 (Batc (None, 84, 84, 64)        256       
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 42, 42, 64)        0         
_________________________________________________________________
dropout_20 (Dropout)         (None, 42, 42, 64)        0         
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 42, 42, 128)       73856     
_________________________________________________________________
```

```
activation_38 (Activation)    (None, 42, 42, 128)       0
_________________________________________________________________
batch_normalization_34 (Batc (None, 42, 42, 128)       512
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 42, 42, 128)       147584
_________________________________________________________________
activation_39 (Activation)    (None, 42, 42, 128)       0
_________________________________________________________________
batch_normalization_35 (Batc (None, 42, 42, 128)       512
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 21, 21, 128)       0
_________________________________________________________________
dropout_21 (Dropout)         (None, 21, 21, 128)       0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 21, 21, 256)       295168
_________________________________________________________________
activation_40 (Activation)    (None, 21, 21, 256)       0
_________________________________________________________________
batch_normalization_36 (Batc (None, 21, 21, 256)       1024
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 21, 21, 256)       590080
_________________________________________________________________
activation_41 (Activation)    (None, 21, 21, 256)       0
_________________________________________________________________
batch_normalization_37 (Batc (None, 21, 21, 256)       1024
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 21, 21, 256)       590080
_________________________________________________________________
activation_42 (Activation)    (None, 21, 21, 256)       0
_________________________________________________________________
batch_normalization_38 (Batc (None, 21, 21, 256)       1024
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 10, 10, 256)       0
_________________________________________________________________
dropout_22 (Dropout)         (None, 10, 10, 256)       0
_________________________________________________________________
flatten_4 (Flatten)          (None, 25600)             0
_________________________________________________________________
dense_12 (Dense)             (None, 512)               13107712
_________________________________________________________________
activation_43 (Activation)    (None, 512)               0
_________________________________________________________________
batch_normalization_39 (Batc (None, 512)               2048
_________________________________________________________________
dropout_23 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               65664
_________________________________________________________________
activation_44 (Activation)    (None, 128)               0
_________________________________________________________________
batch_normalization_40 (Batc (None, 128)               512
_________________________________________________________________
dense_14 (Dense)             (None, 22)                2838
_________________________________________________________________
activation_45 (Activation)    (None, 22)                0
=================================================================
Total params: 14,918,870
Trainable params: 14,915,286
Non-trainable params: 3,584
_________________________________________________________________
```

In [36]:

```
# initialize the model and optimizermodel2 = model()
model5.compile(loss="categorical_crossentropy", optimizer="RMSprop", metrics=["accuracy"
])
```

In [37]:

```
H5 = model5.fit(aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(valX, valY), steps_per_epoch=len(trainX) // BS,
```

```
    epochs=100)
```

```
Epoch 1/100
353/353 [==============================] - 29s 76ms/step - loss: 2.8851 - accuracy: 0.184
2 - val_loss: 4.5902 - val_accuracy: 0.1710
Epoch 2/100
353/353 [==============================] - 24s 69ms/step - loss: 2.2415 - accuracy: 0.318
9 - val_loss: 2.1956 - val_accuracy: 0.3661
Epoch 3/100
353/353 [==============================] - 27s 76ms/step - loss: 2.0346 - accuracy: 0.387
6 - val_loss: 2.1567 - val_accuracy: 0.4078
Epoch 4/100
353/353 [==============================] - 25s 72ms/step - loss: 1.9197 - accuracy: 0.417
2 - val_loss: 2.3255 - val_accuracy: 0.3724
Epoch 5/100
353/353 [==============================] - 26s 73ms/step - loss: 1.8322 - accuracy: 0.435
5 - val_loss: 1.9710 - val_accuracy: 0.4219
Epoch 6/100
353/353 [==============================] - 26s 75ms/step - loss: 1.7291 - accuracy: 0.477
3 - val_loss: 1.6340 - val_accuracy: 0.5159
Epoch 7/100
353/353 [==============================] - 27s 76ms/step - loss: 1.6500 - accuracy: 0.495
2 - val_loss: 1.8281 - val_accuracy: 0.4777
Epoch 8/100
353/353 [==============================] - 24s 69ms/step - loss: 1.5920 - accuracy: 0.520
7 - val_loss: 1.6552 - val_accuracy: 0.5067
Epoch 9/100
353/353 [==============================] - 26s 75ms/step - loss: 1.5544 - accuracy: 0.525
8 - val_loss: 1.7079 - val_accuracy: 0.5145
Epoch 10/100
353/353 [==============================] - 26s 73ms/step - loss: 1.4892 - accuracy: 0.543
0 - val_loss: 1.6118 - val_accuracy: 0.5322
Epoch 11/100
353/353 [==============================] - 24s 69ms/step - loss: 1.4422 - accuracy: 0.561
6 - val_loss: 1.8565 - val_accuracy: 0.4721
Epoch 12/100
353/353 [==============================] - 27s 77ms/step - loss: 1.3914 - accuracy: 0.577
2 - val_loss: 1.3938 - val_accuracy: 0.5993
Epoch 13/100
353/353 [==============================] - 25s 72ms/step - loss: 1.3603 - accuracy: 0.585
4 - val_loss: 1.6665 - val_accuracy: 0.5265
Epoch 14/100
353/353 [==============================] - 26s 74ms/step - loss: 1.3308 - accuracy: 0.596
2 - val_loss: 1.6917 - val_accuracy: 0.5201
Epoch 15/100
353/353 [==============================] - 26s 75ms/step - loss: 1.3146 - accuracy: 0.598
9 - val_loss: 1.7259 - val_accuracy: 0.5258
Epoch 16/100
353/353 [==============================] - 25s 70ms/step - loss: 1.2429 - accuracy: 0.622
8 - val_loss: 1.4794 - val_accuracy: 0.5640
Epoch 17/100
353/353 [==============================] - 27s 77ms/step - loss: 1.2118 - accuracy: 0.625
2 - val_loss: 1.4771 - val_accuracy: 0.5774
Epoch 18/100
353/353 [==============================] - 25s 71ms/step - loss: 1.2070 - accuracy: 0.628
4 - val_loss: 1.3845 - val_accuracy: 0.6007
Epoch 19/100
353/353 [==============================] - 25s 72ms/step - loss: 1.1733 - accuracy: 0.645
7 - val_loss: 1.4032 - val_accuracy: 0.6000
Epoch 20/100
353/353 [==============================] - 27s 76ms/step - loss: 1.1244 - accuracy: 0.654
7 - val_loss: 1.3771 - val_accuracy: 0.6177
Epoch 21/100
353/353 [==============================] - 27s 78ms/step - loss: 1.0788 - accuracy: 0.674
9 - val_loss: 1.6074 - val_accuracy: 0.5604
Epoch 22/100
353/353 [==============================] - 25s 71ms/step - loss: 1.0689 - accuracy: 0.675
9 - val_loss: 1.3053 - val_accuracy: 0.6403
Epoch 23/100
353/353 [==============================] - 26s 73ms/step - loss: 1.0770 - accuracy: 0.674
3 - val_loss: 1.3173 - val_accuracy: 0.6205
Epoch 24/100
```

```
353/353 [==============================] - 26s 74ms/step - loss: 1.0320 - accuracy: 0.684
0 - val_loss: 1.5225 - val_accuracy: 0.5894
Epoch 25/100
353/353 [==============================] - 25s 71ms/step - loss: 1.0134 - accuracy: 0.688
6 - val_loss: 1.3856 - val_accuracy: 0.6360
Epoch 26/100
353/353 [==============================] - 27s 75ms/step - loss: 0.9642 - accuracy: 0.703
8 - val_loss: 1.3457 - val_accuracy: 0.6311
Epoch 27/100
353/353 [==============================] - 25s 72ms/step - loss: 0.9539 - accuracy: 0.707
8 - val_loss: 1.2662 - val_accuracy: 0.6558
Epoch 28/100
353/353 [==============================] - 26s 74ms/step - loss: 0.9210 - accuracy: 0.712
1 - val_loss: 1.5227 - val_accuracy: 0.6049
Epoch 29/100
353/353 [==============================] - 26s 72ms/step - loss: 0.9203 - accuracy: 0.717
3 - val_loss: 1.3392 - val_accuracy: 0.6276
Epoch 30/100
353/353 [==============================] - 25s 71ms/step - loss: 0.9122 - accuracy: 0.721
8 - val_loss: 1.3453 - val_accuracy: 0.6085
Epoch 31/100
353/353 [==============================] - 27s 77ms/step - loss: 0.9008 - accuracy: 0.724
3 - val_loss: 1.3664 - val_accuracy: 0.6367
Epoch 32/100
353/353 [==============================] - 27s 77ms/step - loss: 0.8918 - accuracy: 0.722
7 - val_loss: 1.1976 - val_accuracy: 0.6551
Epoch 33/100
353/353 [==============================] - 26s 74ms/step - loss: 0.8632 - accuracy: 0.736
4 - val_loss: 1.4212 - val_accuracy: 0.6473
Epoch 34/100
353/353 [==============================] - 25s 71ms/step - loss: 0.8393 - accuracy: 0.741
5 - val_loss: 1.2230 - val_accuracy: 0.6544
Epoch 35/100
353/353 [==============================] - 27s 77ms/step - loss: 0.8057 - accuracy: 0.747
1 - val_loss: 1.7504 - val_accuracy: 0.5710
Epoch 36/100
353/353 [==============================] - 25s 72ms/step - loss: 0.8036 - accuracy: 0.746
9 - val_loss: 1.4655 - val_accuracy: 0.6184
Epoch 37/100
353/353 [==============================] - 26s 74ms/step - loss: 0.7905 - accuracy: 0.752
6 - val_loss: 1.4016 - val_accuracy: 0.6438
Epoch 38/100
353/353 [==============================] - 26s 74ms/step - loss: 0.7761 - accuracy: 0.755
1 - val_loss: 1.1792 - val_accuracy: 0.6799
Epoch 39/100
353/353 [==============================] - 25s 72ms/step - loss: 0.7807 - accuracy: 0.763
5 - val_loss: 1.2699 - val_accuracy: 0.6587
Epoch 40/100
353/353 [==============================] - 27s 76ms/step - loss: 0.7479 - accuracy: 0.765
8 - val_loss: 1.2164 - val_accuracy: 0.6636
Epoch 41/100
353/353 [==============================] - 25s 72ms/step - loss: 0.7534 - accuracy: 0.761
2 - val_loss: 1.3046 - val_accuracy: 0.6452
Epoch 42/100
353/353 [==============================] - 26s 75ms/step - loss: 0.7221 - accuracy: 0.780
7 - val_loss: 1.2179 - val_accuracy: 0.6714
Epoch 43/100
353/353 [==============================] - 26s 74ms/step - loss: 0.7032 - accuracy: 0.786
2 - val_loss: 1.2075 - val_accuracy: 0.6756
Epoch 44/100
353/353 [==============================] - 25s 71ms/step - loss: 0.7186 - accuracy: 0.776
6 - val_loss: 1.1889 - val_accuracy: 0.6728
Epoch 45/100
353/353 [==============================] - 27s 78ms/step - loss: 0.6901 - accuracy: 0.786
9 - val_loss: 1.1242 - val_accuracy: 0.6869
Epoch 46/100
353/353 [==============================] - 27s 78ms/step - loss: 0.6474 - accuracy: 0.798
9 - val_loss: 1.1993 - val_accuracy: 0.6806
Epoch 47/100
353/353 [==============================] - 25s 71ms/step - loss: 0.6554 - accuracy: 0.792
7 - val_loss: 1.3236 - val_accuracy: 0.6580
Epoch 48/100
```

```
353/353 [==============================] - 26s 74ms/step - loss: 0.6438 - accuracy: 0.801
1 - val_loss: 1.1040 - val_accuracy: 0.7230
Epoch 49/100
353/353 [==============================] - 27s 76ms/step - loss: 0.6445 - accuracy: 0.797
9 - val_loss: 1.1231 - val_accuracy: 0.7074
Epoch 50/100
353/353 [==============================] - 25s 70ms/step - loss: 0.6263 - accuracy: 0.813
7 - val_loss: 1.3294 - val_accuracy: 0.6862
Epoch 51/100
353/353 [==============================] - 27s 77ms/step - loss: 0.6398 - accuracy: 0.801
8 - val_loss: 1.2345 - val_accuracy: 0.6869
Epoch 52/100
353/353 [==============================] - 26s 73ms/step - loss: 0.5919 - accuracy: 0.817
0 - val_loss: 1.3198 - val_accuracy: 0.6636
Epoch 53/100
353/353 [==============================] - 25s 71ms/step - loss: 0.5880 - accuracy: 0.810
7 - val_loss: 1.1021 - val_accuracy: 0.7131
Epoch 54/100
353/353 [==============================] - 27s 77ms/step - loss: 0.6083 - accuracy: 0.809
7 - val_loss: 1.2010 - val_accuracy: 0.6841
Epoch 55/100
353/353 [==============================] - 26s 72ms/step - loss: 0.6045 - accuracy: 0.807
6 - val_loss: 1.1883 - val_accuracy: 0.6961
Epoch 56/100
353/353 [==============================] - 28s 78ms/step - loss: 0.5615 - accuracy: 0.820
2 - val_loss: 1.2125 - val_accuracy: 0.6961
Epoch 57/100
353/353 [==============================] - 27s 76ms/step - loss: 0.5588 - accuracy: 0.822
8 - val_loss: 1.0776 - val_accuracy: 0.7314
Epoch 58/100
353/353 [==============================] - 27s 77ms/step - loss: 0.5642 - accuracy: 0.824
1 - val_loss: 1.2805 - val_accuracy: 0.6763
Epoch 59/100
353/353 [==============================] - 25s 72ms/step - loss: 0.5491 - accuracy: 0.824
6 - val_loss: 1.1751 - val_accuracy: 0.7088
Epoch 60/100
353/353 [==============================] - 27s 78ms/step - loss: 0.5503 - accuracy: 0.831
5 - val_loss: 1.1705 - val_accuracy: 0.7124
Epoch 61/100
353/353 [==============================] - 26s 73ms/step - loss: 0.5272 - accuracy: 0.828
2 - val_loss: 1.0464 - val_accuracy: 0.7265
Epoch 62/100
353/353 [==============================] - 26s 72ms/step - loss: 0.5146 - accuracy: 0.836
2 - val_loss: 1.1325 - val_accuracy: 0.7159
Epoch 63/100
353/353 [==============================] - 26s 75ms/step - loss: 0.5172 - accuracy: 0.833
5 - val_loss: 1.2019 - val_accuracy: 0.7011
Epoch 64/100
353/353 [==============================] - 26s 73ms/step - loss: 0.5286 - accuracy: 0.836
1 - val_loss: 1.1504 - val_accuracy: 0.7201
Epoch 65/100
353/353 [==============================] - 27s 76ms/step - loss: 0.5107 - accuracy: 0.838
5 - val_loss: 1.1438 - val_accuracy: 0.7152
Epoch 66/100
353/353 [==============================] - 27s 76ms/step - loss: 0.5198 - accuracy: 0.836
3 - val_loss: 1.0477 - val_accuracy: 0.7237
Epoch 67/100
353/353 [==============================] - 27s 78ms/step - loss: 0.4872 - accuracy: 0.846
2 - val_loss: 1.1346 - val_accuracy: 0.7279
Epoch 68/100
353/353 [==============================] - 28s 79ms/step - loss: 0.4873 - accuracy: 0.849
2 - val_loss: 1.1554 - val_accuracy: 0.7081
Epoch 69/100
353/353 [==============================] - 25s 70ms/step - loss: 0.4686 - accuracy: 0.854
3 - val_loss: 1.1359 - val_accuracy: 0.7194
Epoch 70/100
353/353 [==============================] - 27s 77ms/step - loss: 0.4776 - accuracy: 0.845
4 - val_loss: 1.0762 - val_accuracy: 0.7329
Epoch 71/100
353/353 [==============================] - 27s 76ms/step - loss: 0.4694 - accuracy: 0.852
0 - val_loss: 1.5194 - val_accuracy: 0.6721
Epoch 72/100
```

```
353/353 [==============================] - 25s 72ms/step - loss: 0.4650 - accuracy: 0.852
5 - val_loss: 1.2580 - val_accuracy: 0.7124
Epoch 73/100
353/353 [==============================] - 26s 75ms/step - loss: 0.4435 - accuracy: 0.860
8 - val_loss: 1.4359 - val_accuracy: 0.6933
Epoch 74/100
353/353 [==============================] - 27s 76ms/step - loss: 0.4362 - accuracy: 0.857
8 - val_loss: 1.2392 - val_accuracy: 0.7102
Epoch 75/100
353/353 [==============================] - 25s 71ms/step - loss: 0.4360 - accuracy: 0.865
8 - val_loss: 1.2436 - val_accuracy: 0.7145
Epoch 76/100
353/353 [==============================] - 27s 76ms/step - loss: 0.4166 - accuracy: 0.869
2 - val_loss: 1.2883 - val_accuracy: 0.7166
Epoch 77/100
353/353 [==============================] - 26s 73ms/step - loss: 0.4275 - accuracy: 0.860
1 - val_loss: 1.0872 - val_accuracy: 0.7519
Epoch 78/100
353/353 [==============================] - 25s 72ms/step - loss: 0.4257 - accuracy: 0.863
3 - val_loss: 1.3019 - val_accuracy: 0.6905
Epoch 79/100
353/353 [==============================] - 27s 76ms/step - loss: 0.4072 - accuracy: 0.873
3 - val_loss: 1.1743 - val_accuracy: 0.7399
Epoch 80/100
353/353 [==============================] - 25s 72ms/step - loss: 0.4167 - accuracy: 0.862
3 - val_loss: 1.3770 - val_accuracy: 0.6982
Epoch 81/100
353/353 [==============================] - 27s 76ms/step - loss: 0.4265 - accuracy: 0.863
3 - val_loss: 1.4469 - val_accuracy: 0.6848
Epoch 82/100
353/353 [==============================] - 27s 76ms/step - loss: 0.4091 - accuracy: 0.865
4 - val_loss: 1.0983 - val_accuracy: 0.7505
Epoch 83/100
353/353 [==============================] - 27s 75ms/step - loss: 0.4050 - accuracy: 0.869
9 - val_loss: 1.1795 - val_accuracy: 0.7343
Epoch 84/100
353/353 [==============================] - 25s 70ms/step - loss: 0.3913 - accuracy: 0.877
3 - val_loss: 1.1310 - val_accuracy: 0.7364
Epoch 85/100
353/353 [==============================] - 27s 77ms/step - loss: 0.3824 - accuracy: 0.881
2 - val_loss: 1.4119 - val_accuracy: 0.7046
Epoch 86/100
353/353 [==============================] - 25s 72ms/step - loss: 0.3820 - accuracy: 0.880
8 - val_loss: 1.2148 - val_accuracy: 0.7343
Epoch 87/100
353/353 [==============================] - 26s 72ms/step - loss: 0.3815 - accuracy: 0.882
9 - val_loss: 1.2143 - val_accuracy: 0.7329
Epoch 88/100
353/353 [==============================] - 27s 75ms/step - loss: 0.4121 - accuracy: 0.870
8 - val_loss: 1.4471 - val_accuracy: 0.6820
Epoch 89/100
353/353 [==============================] - 25s 72ms/step - loss: 0.3752 - accuracy: 0.880
7 - val_loss: 1.3390 - val_accuracy: 0.6989
Epoch 90/100
353/353 [==============================] - 26s 74ms/step - loss: 0.3678 - accuracy: 0.884
5 - val_loss: 1.2635 - val_accuracy: 0.7230
Epoch 91/100
353/353 [==============================] - 25s 72ms/step - loss: 0.3843 - accuracy: 0.874
2 - val_loss: 1.1428 - val_accuracy: 0.7435
Epoch 92/100
353/353 [==============================] - 25s 72ms/step - loss: 0.3607 - accuracy: 0.885
9 - val_loss: 1.3044 - val_accuracy: 0.7180
Epoch 93/100
353/353 [==============================] - 26s 75ms/step - loss: 0.3506 - accuracy: 0.889
8 - val_loss: 1.1725 - val_accuracy: 0.7519
Epoch 94/100
353/353 [==============================] - 27s 77ms/step - loss: 0.3527 - accuracy: 0.887
6 - val_loss: 1.1934 - val_accuracy: 0.7477
Epoch 95/100
353/353 [==============================] - 24s 69ms/step - loss: 0.3566 - accuracy: 0.889
0 - val_loss: 1.5172 - val_accuracy: 0.7067
Epoch 96/100
```

```
353/353 [==============================] - 26s 74ms/step - loss: 0.3461 - accuracy: 0.889
5 - val_loss: 1.1918 - val_accuracy: 0.7258
Epoch 97/100
353/353 [==============================] - 26s 75ms/step - loss: 0.3386 - accuracy: 0.892
4 - val_loss: 1.4758 - val_accuracy: 0.6834
Epoch 98/100
353/353 [==============================] - 25s 70ms/step - loss: 0.3263 - accuracy: 0.893
9 - val_loss: 1.3135 - val_accuracy: 0.7180
Epoch 99/100
353/353 [==============================] - 27s 75ms/step - loss: 0.3310 - accuracy: 0.897
9 - val_loss: 1.2855 - val_accuracy: 0.7201
Epoch 100/100
353/353 [==============================] - 25s 72ms/step - loss: 0.3345 - accuracy: 0.893
4 - val_loss: 1.2698 - val_accuracy: 0.7477
```

In [38]:

```
preds5 = model5.evaluate(testX, testY, batch_size=32)
print('loss = '+str(preds5[0]))
print('test accuracy = '+str(preds5[1]))
```

```
45/45 [==============================] - 1s 13ms/step - loss: 1.2949 - accuracy: 0.7284
loss = 1.2949408292770386
test accuracy = 0.7284299731254578
```

We got 72.98% test accuracy which is higher than all the model used.

In [39]:

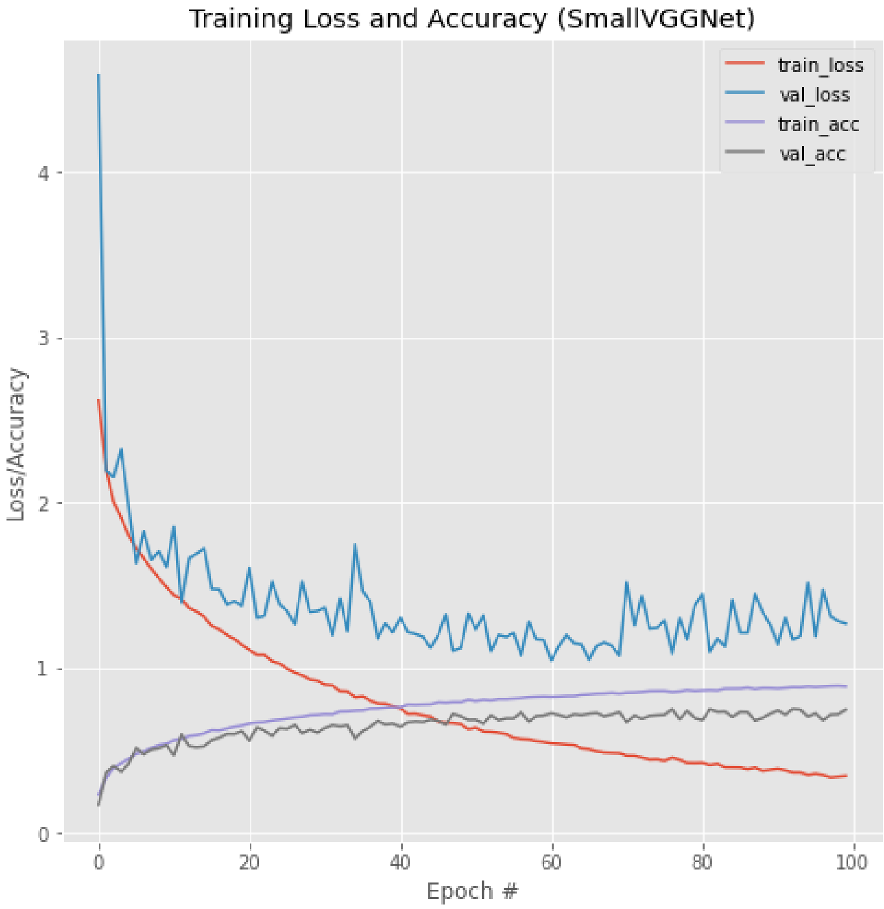
```
# evaluate the network
print("[INFO] evaluating network...")
predictions5 = model5.predict(valX, batch_size=32)
print(classification_report(valY.argmax(axis=1),
    predictions5.argmax(axis=1), target_names=lb.classes_))
```

```
[INFO] evaluating network...
                precision    recall  f1-score   support

     badminton       0.79      0.70      0.74        79
      baseball       0.88      0.79      0.83        89
    basketball       0.49      0.84      0.62        49
        boxing       0.87      0.68      0.76        77
         chess       0.62      0.63      0.63        49
       cricket       0.93      0.73      0.82        74
       fencing       0.77      0.71      0.74        65
      football       0.84      0.66      0.74        77
      formula1       0.82      0.75      0.78        68
    gymnastics       0.78      0.61      0.68        69
        hockey       0.48      0.64      0.55        45
    ice_hockey       0.78      0.91      0.84        76
       kabaddi       0.91      0.81      0.86        37
        motogp       0.80      0.85      0.83        62
      shooting       0.76      0.70      0.73        40
      swimming       0.89      0.91      0.90        68
  table_tennis       0.78      0.61      0.68        74
        tennis       0.71      0.73      0.72        84
    volleyball       0.72      0.88      0.79        76
weight_lifting       0.55      0.63      0.59        51
     wrestling       0.68      0.81      0.74        48
           wwe       0.71      0.86      0.78        58

      accuracy                           0.75      1415
     macro avg       0.75      0.75      0.74      1415
  weighted avg       0.77      0.75      0.75      1415
```

In [40]:

```
# plot the training loss and accuracy for model 2
N = np.arange(0, 100)
plt.style.use("ggplot")
```

```
plt.figure(figsize=(8,8))
plt.plot(N, H5.history["loss"], label="train_loss")
plt.plot(N, H5.history["val_loss"], label="val_loss")
plt.plot(N, H5.history["accuracy"], label="train_acc")
plt.plot(N, H5.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy (SmallVGGNet)")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig("smallvggnet_plot.png")
plt.show()
```



We can see that overfitting is reduced by a large extent. So, Model 5 is the best, and this is our final model. Now we test some real life pictures with this model to check its accuracy.

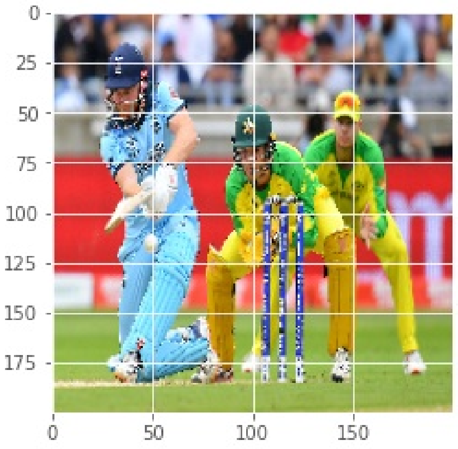
Prediction on internet images

In [52]:

```
from skimage import io
import PIL
from keras.preprocessing import image
show_img=image.load_img('../input/images2/Jonny-Bairstow-batting-semifinal-match-England-
Australia-2019.jpg', grayscale=False, target_size=(200, 200,3))
plt.imshow(show_img);

imagePath="../input/images2/Jonny-Bairstow-batting-semifinal-match-England-Australia-2019
.jpg"
image = cv2.imread(imagePath)
image = cv2.resize(image, (84,84))
#plt.imshow(image)

x = np.expand_dims(image, axis = 0)
x = np.array(x, dtype="float") / 255.0


prediction=model5.predict(x)
index=np.argmax(prediction[0])
```

```
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(classes_name[index], 100 * np.max(prediction[0]))
)
```

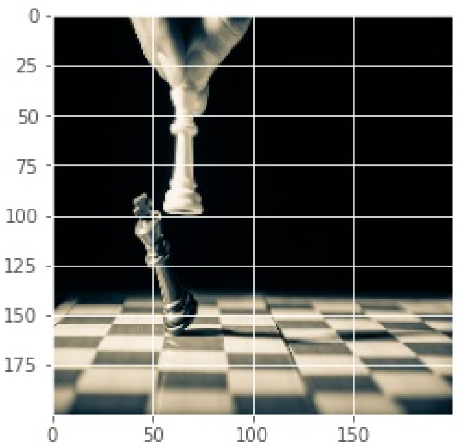
This image most likely belongs to cricket with a 100.00 percent confidence.



In [53]:

```
from skimage import io
import PIL
from keras.preprocessing import image
show_img=image.load_img('../input/images2/photo-1604948501466-4e9c339b9c24.jpg', grayscal
e=False, target_size=(200, 200,3))
plt.imshow(show_img);

imagePath="../input/images2/photo-1604948501466-4e9c339b9c24.jpg"
image = cv2.imread(imagePath)
image = cv2.resize(image, (84,84))
#plt.imshow(image)

x = np.expand_dims(image, axis = 0)
x = np.array(x, dtype="float") / 255.0


prediction=model5.predict(x)
index=np.argmax(prediction[0])
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(classes_name[index], 100 * np.max(prediction[0]))
)
```

This image most likely belongs to chess with a 99.76 percent confidence.



In [ ]:

```
from skimage import io
import PIL
from keras.preprocessing import image
show_img=image.load_img('../input/images/image5.jpg', grayscale=False, target_size=(200,
```

```
200,3))
plt.imshow(show_img);

imagePath="../input/images/image5.jpg"
image = cv2.imread(imagePath)
image = cv2.resize(image, (84,84))
plt.imshow(image)

x = np.expand_dims(image, axis = 0)
x = np.array(x, dtype="float") / 255.0


prediction=model5.predict(x)
index=np.argmax(prediction[0])
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(classes_name[index], 100 * np.max(prediction[0]))
)
```

We can see that model 5 is recognizing and classifying all images very nicely.

Conclusion

We finally conclude that Model with specification

Conv2D0 -> Conv2D1 -> Batchnormalization0 -> Maxpool0 -> Dropout -> Cov2D3 -> Conv2D4 -> batchnormalization -> Maxpool1 -> Dropout -> Conv2D5 -> Conv2D6 -> Conv2D7 -> Batchnormalization -> Maxpool2 -> Dropout -> FC0 -> Dropout -> FC1 -> Dropout ->Softmax with RMSprop optimizer, 100 epoches and 32 batch-size is giving 72.98% accuracy on our test data. We can further improve our accuracy with more regularizations.

In [ ]:

```

```