Human Action Recognition using CNN

Nagasharan Sathish

May 3, 2024

Abstract

This project focuses on Human Action Recognition (HAR) using a dataset that consists of 15 different human actions. We utilize Convolutional Neural Networks (CNNs) for image classification to categorize the human actions captured in the images. By extracting features directly from the images, CNNs help in distinguishing between various activities and enable the model to learn the intricacies of human poses and movements within the activity. Our goal is to develop a robust system that can accurately classify different human actions captured in images by leveraging CNNs.

1 Introduction

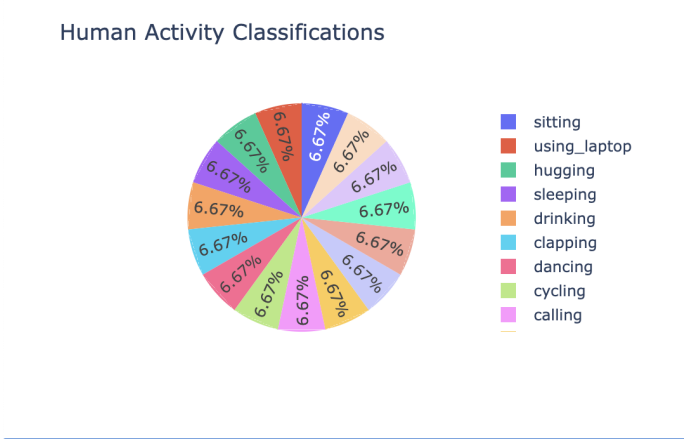
Still image based action recognition has many applications such as surveillance, robotic applications, annotating images using verbs, searching the image database using verbs, etc[1]. Human activity recognition involves identifying specific movements or actions based on some data from the sensor. These movements can include dancing, walking, sitting, and more.

1.1 Problem Statement

The goal of Human Action Recognition is to understand human actions through images and their associated labels. The dataset comprises over 12,000 labeled images, with 15 different action classifications such as walking, dancing, sitting, etc. This problem falls into the category of classification. Our objective is to build a Convolutional Neural Network (CNN) that can identify human actions. We aim to create a robust model by experimenting with various parameters to achieve optimal results.

1.2 Data Interpretation/Analysis

In the dataset, we have 12600 data in the training dataset. Also, we have a CSV file containing image names and their corresponding labels. The images are equally divided into 15 different classifications of the actions. The below figure represents the various classifications and data available in the training dataset.




```
label
sitting                 840
using_laptop            840
hugging                 840
sleeping                840
drinking                840
clapping                840
dancing                 840
cycling                 840
calling                 840
laughing                840
eating                  840
fighting                840
listening_to_music      840
running                 840
texting                 840
Name: count, dtype: int64
```

By using the csv file we can match the images and the labels for the sample data. Below are some sample images in the training dataset.

On the other hand, the test dataset doesn't have the labels available in the Excel file, so we will validate them manually with a few images to check if the model performing well with the new data outside of the training dataset. Also, we will split the training dataset into test and train datasets to check accuracy.

2 Solution and Technical Implementation

2.1 Data Prepossessing

In the sample images, it is evident that the images are of different sizes, so images need to be resized, and data is split into images and their corresponding labels. The images are resized to 256*256. Below is the code snippet.

```
x_train = []
y_train = []
length = len(train_data)
for i in (range(len(train_data))):
    t = './Human Action Recognition/train/' + filename[i]
    temp_img = Image.open(t)
    x_train.append(np.asarray(temp_img.resize((256,256))))
x_train = x_train
x_train = np.asarray(x_train)
```

```
y_train = to_categorical(np.asarray(train_data["label"].factorize()[0]))
```

```
print(len(x_train))
print(len(y_train))
```

```
12600
12600
```

2.2 Model Architecture

We used Convolutional Neural Network (CNN) for our image classification. The model will assign the probability to each of the 15 classifications. Lets breakdown the architecture.

- Input Layer
  - shape: (256,256,3) which represents 256*256 pixels and 3 color channels (RGB)
- Convolutional Layers
  - Four convolutional layers with ReLu activation
    - Layer 1: 64 filters
    - Layer 2: 64 filters
    - Layer 3: 128 filters
    - Layer 4: 256 filters
  - Each layer extracts local features from the input image
- Max-Pooling Layers
  - Each convolutional layer followed by max pooling layer which will reduce the spacial dimensions, capturing the essential features

- Flatten layer: converts 3D feature maps to 1D vector
- Fully Connected Layers
  - Two Dense Layers with ReLu activation
    - Layer 1: 128 neurons
    - Layer 2: 256 neurons
  - These layers learn high-level representations
- Output Layer
  - units : 15 (number of classes)
  - softmax activation

Below image specifies the layers and the hyper parameters used in the model.

Model: "sequential_3"

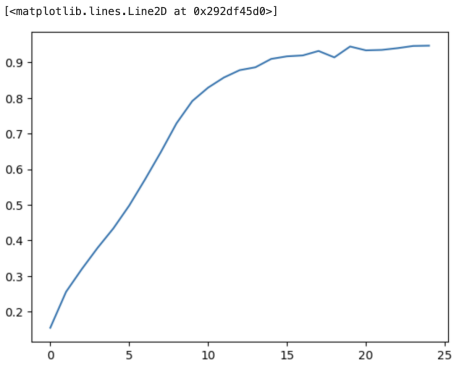
Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_8 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_9 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_8 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_10 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_9 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_11 (Conv2D)	(None, 12, 12, 256)	295,168
max_pooling2d_10 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten_2 (Flatten)	(None, 9216)	0
dense_4 (Dense)	(None, 128)	1,179,776
dense_5 (Dense)	(None, 256)	33,024
dense_6 (Dense)	(None, 15)	3,855

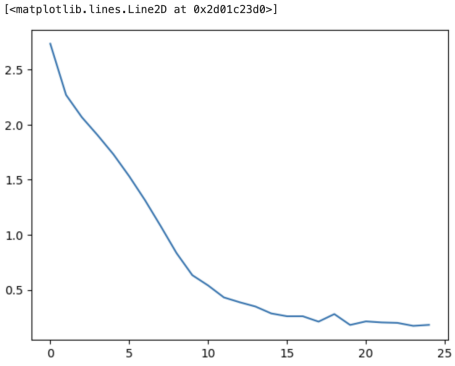
Total params: 1,641,999 (6.26 MB)
Trainable params: 1,641,999 (6.26 MB)
Non-trainable params: 0 (0.00 B)

2.3 Training the Model

The Adam optimizer and categorical cross-entropy loss function are used in the training of the model. For training, batch size used is 30 and the number of epochs used is 25. After a few iterations, it is evident that the model was effective around the epoch of 20.

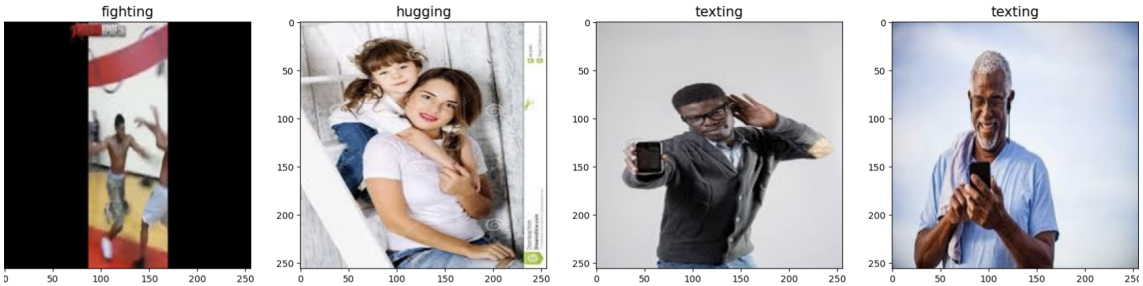
Below is the accuracy and loss of the model for the training data set.





3 Result Discussion

For the training dataset, the accuracy of 95% and loss of 0.163 is observed after 25 epochs. This suggests the model is performing well for the training dataset. However, when tested for the test dataset manually due to the unavailability of the labels in the data, the accuracy was observed around 50-70%. Below is the sample results for some random images.



Initial models were also troubled by the overfitting issue. After trying various models, the final architecture and hyperparameters were chosen because they performed the best among the tried combinations.

Since the labels were not available for the test dataset, initially tain dataset was divided into test and train datasets. However, the efficiency of the model was reduced to 30% with this approach. This might be due to the less training data because of partition. Hence in the final model training dataset is not split and the test dataset is validated manually for a few images.

4 Conclusion

Using CNN we are able to solve Human actions recognition which is an image classification problem. The accuracy of the model is around 50-70%. We can have more accurate model by using the other open-source models as the backbone of the network [2].

5 Code

[12]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import glob
import random
import numpy as np
import pandas as pd
#import tensorflow_addons as tfa
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Activation, Dropout,␣
↪Flatten, Dense, Input
#from keras.preprocessing.image import ImageDataGenerator
from tqdm import tqdm
from PIL import Image
from tensorflow.keras.utils import to_categorical
import seaborn as sns
import matplotlib.image as img
import matplotlib.pyplot as plt
import plotly.express as plotty
from tensorflow.keras.preprocessing.image import img_to_array,load_img
import torch
from torch.utils.data import TensorDataset, DataLoader
from keras.models import load_model
from tensorflow.keras.optimizers import Adam
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

[ ]:
```
train_data = pd.read_csv("./Human Action Recognition/Training_set.csv")
test_data = pd.read_csv("./Human Action Recognition/Testing_set.csv")
#print(train_data)
print(train_data.label.value_counts())
#actions = train_data.label.value_counts()
#plt.pie(actions.values,actions.index)
```

[ ]:
```
def show_data_graph():
    actions_train = train_data.label.value_counts()
    fig = plotty.pie(train_data, values=actions_train.values,␣
 ↪names=actions_train.index, title='Human Activity Classifications')
    fig.show()
show_data_graph()
```

[7]:
```
filename = train_data['filename']
action = train_data['label']
def train_data_random_images():
    for i in range(6):
        num = random.randint(1,12600)
        img_name = "Image_{}.jpg".format(num)
        train_fldr = "./Human Action Recognition/train/"
        test_image = img.imread(train_fldr+img_name)
        plt.subplot(2, 3, i+1)
        plt.imshow(test_image)
        plt.axis('off')
        plt.title("{}".format(train_data.loc[train_data['filename'] ==␣
 ↪"{}".format(img_name), 'label'].item()))
```

[ ]:
```
train_data_random_images()
```

[ ]:
```
x_train = []
y_train = []
length = len(train_data)
for i in (range(len(train_data))):
    t = './Human Action Recognition/train/' + filename[i]
    temp_img = Image.open(t)
    x_train.append(np.asarray(temp_img.resize((256,256))))
x_train = x_train
x_train = np.asarray(x_train)
```

[ ]:
```
y_train = to_categorical(np.asarray(train_data["label"].factorize()[0]))
```

[115]:
```
def read_img(fn):
    img = Image.open(fn)
    return np.asarray(img.resize((256,256)))
```

[116]:
```
actions_ =␣
 ↪['sitting','using_laptop','hugging','sleeping','drinking','clapping','dancing','cy
```

[ ]:
```
classes=train_data.label.value_counts()
num_classes=len(classes)
```

[ ]:
```
model=Sequential()

model.add(Input(shape=(256,256,3)))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))


model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))


model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))


model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(num_classes,activation='softmax'))

model.summary()
```

[43]:
```
model.compile(optimizer='adam',␣
 ↪loss='categorical_crossentropy',metrics=['accuracy'])
```

[44]:
```
history=model.fit(x_train,y_train, epochs=25,batch_size=30)
```

```
Epoch 1/25
420/420  269s 638ms/step -
accuracy: 0.1122 - loss: 3.6877
Epoch 2/25
420/420  299s 711ms/step -
accuracy: 0.2412 - loss: 2.3106
Epoch 3/25
420/420  475s 1s/step -
accuracy: 0.3064 - loss: 2.0769
Epoch 4/25
420/420  476s 1s/step -
accuracy: 0.3853 - loss: 1.9013
Epoch 5/25
420/420  297s 706ms/step -
```

```
accuracy: 0.4313 - loss: 1.7344
Epoch 6/25
420/420  314s 748ms/step -
accuracy: 0.5110 - loss: 1.5012
Epoch 7/25
420/420  322s 767ms/step -
accuracy: 0.5945 - loss: 1.2541
Epoch 8/25
420/420  318s 756ms/step -
accuracy: 0.6659 - loss: 1.0233
Epoch 9/25
420/420  307s 730ms/step -
accuracy: 0.7480 - loss: 0.7768
Epoch 10/25
420/420  278s 662ms/step -
accuracy: 0.8180 - loss: 0.5548
Epoch 11/25
420/420  276s 657ms/step -
accuracy: 0.8529 - loss: 0.4618
Epoch 12/25
420/420  279s 664ms/step -
accuracy: 0.8767 - loss: 0.3812
Epoch 13/25
420/420  279s 665ms/step -
accuracy: 0.8888 - loss: 0.3559
Epoch 14/25
420/420  277s 659ms/step -
accuracy: 0.9005 - loss: 0.3076
Epoch 15/25
420/420  267s 635ms/step -
accuracy: 0.9176 - loss: 0.2624
Epoch 16/25
420/420  261s 622ms/step -
accuracy: 0.9280 - loss: 0.2256
Epoch 17/25
420/420  265s 631ms/step -
accuracy: 0.9356 - loss: 0.2052
Epoch 18/25
420/420  266s 632ms/step -
accuracy: 0.9417 - loss: 0.1763
Epoch 19/25
420/420  273s 651ms/step -
accuracy: 0.9363 - loss: 0.2063
Epoch 20/25
420/420  269s 640ms/step -
accuracy: 0.9457 - loss: 0.1662
Epoch 21/25
420/420  270s 642ms/step -
```

```
accuracy: 0.9367 - loss: 0.1981
Epoch 22/25
420/420  266s 634ms/step -
accuracy: 0.9432 - loss: 0.1703
Epoch 23/25
420/420  268s 638ms/step -
accuracy: 0.9477 - loss: 0.1783
Epoch 24/25
420/420  276s 657ms/step -
accuracy: 0.9500 - loss: 0.1501
Epoch 25/25
420/420  283s 673ms/step -
accuracy: 0.9511 - loss: 0.1643
```

[ ]:
```
accuracy = history.history['accuracy']
plt.plot(accuracy)
```

[ ]:
```
plt.plot(history.history['loss'])
```

[ ]:
```
random_images = []
for i in range(100, 110):
    imgg = "Image_{}.jpg".format(i)
    train = "./Human Action Recognition/train/"
    result = model.predict(np.asarray([read_img(train+imgg)]))
    itemindex = np.where(result==np.max(result))
    prediction = itemindex[1][0]

    r = random.randint(1, 60000)

    random_images.append((read_img(train+imgg), actions_[prediction]))
```

[ ]:
```
show_images(list(map(lambda x: x[0], random_images)), list(map(lambda x:
 ↪ x[1], random_images)))
```

[266]:
```
x_train_1 = x_train[250:]
x_test_1 = x_train[:250]
y_train_1 = y_train[250:]
y_test_1 = y_train[:250]
```

[ ]:
```
model_2=Sequential()

model_2.add(Input(shape=(256,256,3)))
model_2.add(Conv2D(32,(3,3),activation='relu'))
model_2.add(MaxPooling2D((2,2)))

model_2.add(Conv2D(64,(3,3),activation='relu'))
model_2.add(MaxPooling2D((2,2)))


model_2.add(Conv2D(64,(3,3),activation='relu'))
model_2.add(MaxPooling2D((2,2)))


model_2.add(Conv2D(128,(3,3),activation='relu'))
model_2.add(MaxPooling2D((2,2)))


model_2.add(Conv2D(256,(3,3),activation='relu'))
model_2.add(MaxPooling2D((2,2)))

model_2.add(Flatten())

model_2.add(Dense(128,activation='relu'))
model_2.add(Dense(256,activation='relu'))
model_2.add(Dense(num_classes,activation='softmax'))

model_2.summary()
```

[268]:
```
model_2.compile(optimizer='adam',␣
↪loss='categorical_crossentropy',metrics=['accuracy'])
```

[273]:
```
history_2=model_2.fit(x_train_1,y_train_1, validation_data=(x_test_1,␣
↪y_test_1),epochs=15,batch_size=30)
```

```
Epoch 1/15
412/412  242s 587ms/step -
accuracy: 0.1587 - loss: 2.5372 - val_accuracy: 0.2280 - val_loss: 2.3308
Epoch 2/15
412/412  253s 615ms/step -
accuracy: 0.2438 - loss: 2.2878 - val_accuracy: 0.2440 - val_loss: 2.2633
Epoch 3/15
412/412  266s 645ms/step -
accuracy: 0.3115 - loss: 2.1388 - val_accuracy: 0.3000 - val_loss: 2.1479
Epoch 4/15
412/412  271s 657ms/step -
accuracy: 0.3747 - loss: 1.9319 - val_accuracy: 0.3040 - val_loss: 2.1059
Epoch 5/15
```

```
412/412  276s 671ms/step -
accuracy: 0.4344 - loss: 1.7435 - val_accuracy: 0.2680 - val_loss: 2.1503
Epoch 6/15
412/412  278s 675ms/step -
accuracy: 0.4978 - loss: 1.5234 - val_accuracy: 0.3080 - val_loss: 2.2180
Epoch 7/15
412/412  277s 673ms/step -
accuracy: 0.5911 - loss: 1.2325 - val_accuracy: 0.3200 - val_loss: 2.4345
Epoch 8/15
412/412  282s 683ms/step -
accuracy: 0.6715 - loss: 1.0028 - val_accuracy: 0.2760 - val_loss: 2.5676
Epoch 9/15
412/412  282s 685ms/step -
accuracy: 0.7295 - loss: 0.8283 - val_accuracy: 0.3080 - val_loss: 2.8211
Epoch 10/15
412/412  281s 681ms/step -
accuracy: 0.8156 - loss: 0.5677 - val_accuracy: 0.2880 - val_loss: 3.5052
Epoch 11/15
412/412  286s 694ms/step -
accuracy: 0.8526 - loss: 0.4476 - val_accuracy: 0.3040 - val_loss: 3.6634
Epoch 12/15
412/412  283s 686ms/step -
accuracy: 0.8794 - loss: 0.3843 - val_accuracy: 0.3520 - val_loss: 3.6453
Epoch 13/15
412/412  281s 681ms/step -
accuracy: 0.8954 - loss: 0.3133 - val_accuracy: 0.3200 - val_loss: 3.9441
Epoch 14/15
412/412  289s 701ms/step -
accuracy: 0.9118 - loss: 0.2731 - val_accuracy: 0.3040 - val_loss: 4.5765
Epoch 15/15
412/412  285s 691ms/step -
accuracy: 0.9253 - loss: 0.2357 - val_accuracy: 0.2880 - val_loss: 4.8586
```

[ ]:
```
random_images2 = []
for i in range(1, 10):
    imgg = "Image_{}.jpg".format(i)
    train = "./Human Action Recognition/train/"
    result = model.predict(np.asarray([read_img(train+imgg)]))
    itemindex = np.where(result==np.max(result))
    prediction = itemindex[1][0]

    r = random.randint(1, 60000)

    random_images2.append((read_img(train+imgg), actions_[prediction]))
```

[ ]:
```
show_images(list(map(lambda x: x[0], random_images2)), list(map(lambda␣
 ↪x: x[1], random_images2)))
```

6 References

1 Deeptha Girish, Vineeta Singh, Anca Ralescu; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020, pp. 370-371
https://openaccess.thecvf.com/content_CVPRW_2020/papers/w23/Girish_Understanding_Action_Recognition_in_Still_Images_CVPRW_2020_paper.pdf

2 Seyed Rohollah Hosseyni, Sanaz Seyedin, Hasan Taheri;Human Action Recognition in Still Images Using ConViT https://doi.org/10.48550/arXiv.2307.08994

3 https://medium.com/@khang.pham.exxact/image-classification-with-dcnns-35b9108fb782

4 https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8