image-classification

December 5, 2022

title: ”Image_Classification” author: Aarushi Pandey date: 11/21/2022

1 Image Classification

The data I will be classifying is from Kaggle. It is images of people doing various yoga poses, and the aim is to correctly classify the test images to the right yoga poses.

1.0.1 1. Read the data

Since the original dataset has each yoga pose in its own separate folder, I will need to go through each folder. This Kaggle notebook was a great resource in doing so.

First, I need to import all the required packages. Then, I will store the names of all the possible poses in the labels variable.

[24]:
```python
import numpy as np
import pandas as pd
import tensorflow as tf
import os,cv2

#import warnings
#warnings.filterwarnings("ignore")

data_path = '../input/yoga-pose-image-classification-dataset/dataset'

labels=[]
#print(os.listdir(data_path))
for folder in os.listdir(data_path):
    labels.append(folder)
#print(labels)
labels.sort() #len = 107
#print(labels)
#labels = labels[0:13] #len =13
```

Now, I will get all the possible images to append it to train_images variable. The poses will be indexed according to the order they were accessed before (to add to the labels list), and these indexes will be added to the train_labels variable. These will then be converted to arrays.

[25]:

```
train_images=[]
train_labels=[]

for i,folder in enumerate(labels):
    try:
        for image in os.listdir(data_path+'/'+folder):
            img = os.path.join(data_path+'/'+folder+'/'+image)
            img = cv2.imread(img)
            img = cv2.resize(img,(256,256))
            #print(img)
            train_images.append(img)
            train_labels.append(i)
    except:
        print(i,folder,image,img)

# convert lists to arrays
train_images = np.asarray(train_images)
#print(train_labels)
train_labels = np.asarray(train_labels).astype('int64')
```

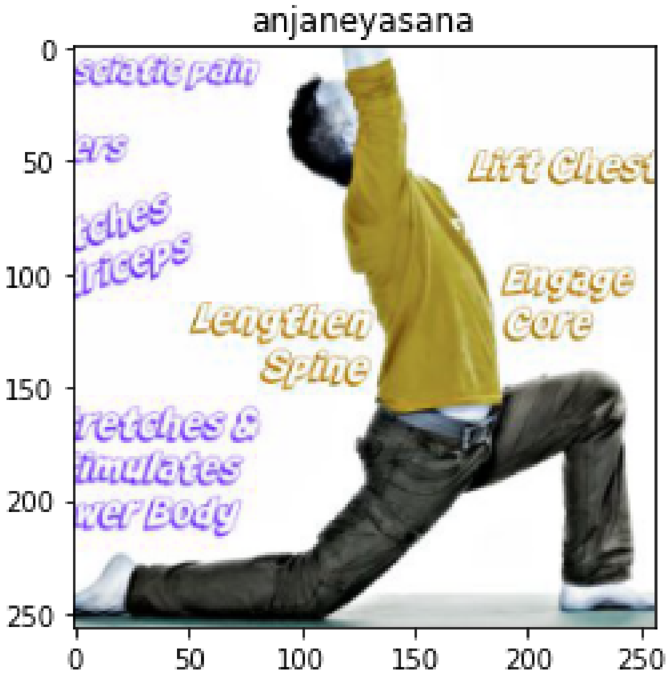
```
libpng warning: Ignoring incorrect cHRM white(.34575,.35855)
r(.6485,.33088)g(.32121,.59787)b(.15589,.06604) when sRGB is also present
```

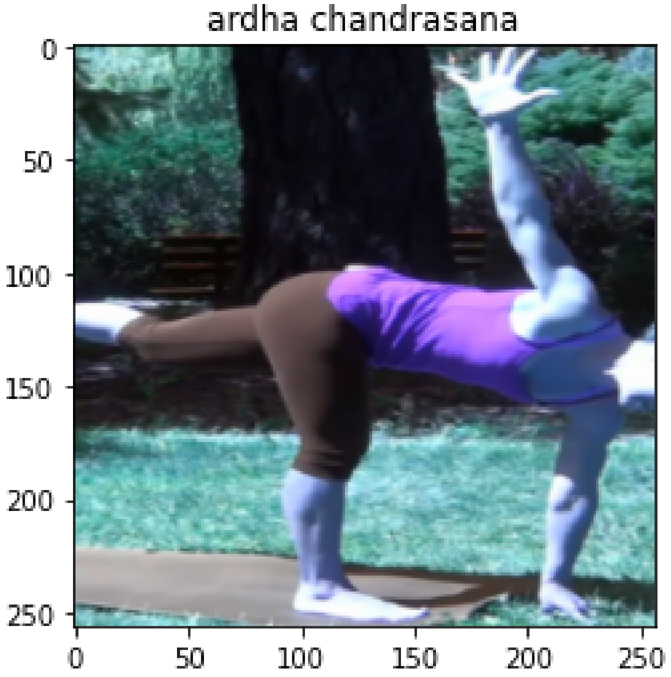
```
100 virabhadrasana i File62.gif None
101 virabhadrasana ii File36.gif None
```

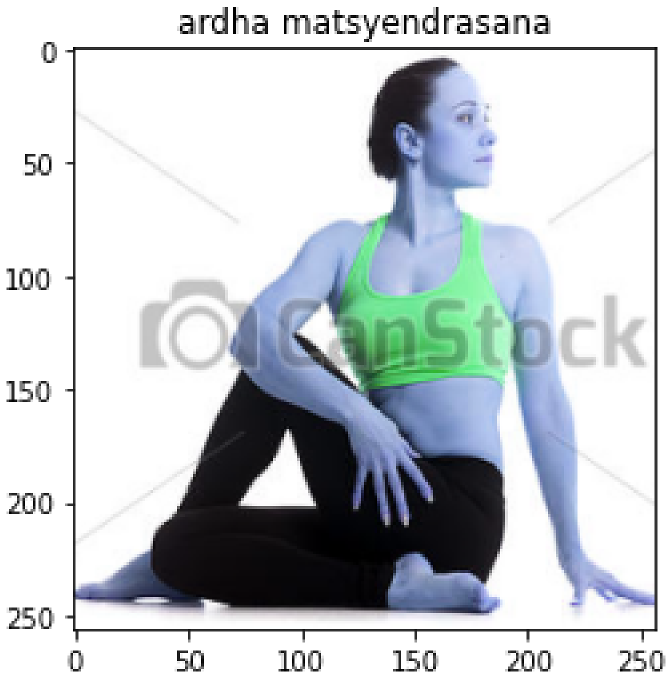
Then I plotted some random images to ensure that they were the same size and had the right labels.

[6]:

```
import matplotlib.pyplot as plt
plt.imshow(train_images[300])
#print(train_labels[300])
plt.title(labels[train_labels[300]])
plt.show()

plt.imshow(train_images[400])
plt.title(labels[train_labels[400]])
plt.show()

plt.imshow(train_images[500])
plt.title(labels[train_labels[500]])
plt.show()
```


anjaneyasana
sciatic pain
ers
tches
riceps
Lift Chest
Engage
Core
Lengthen
Spine
retches &
timulates
wer Body
0
50
100
150
200
250
0
50
100
150
200
250


ardha chandrasana
0
50
100
150
200
250
0
50
100
150
200
250



Now, I will print the dimensions of the data and then split it 80-20.

[26]:

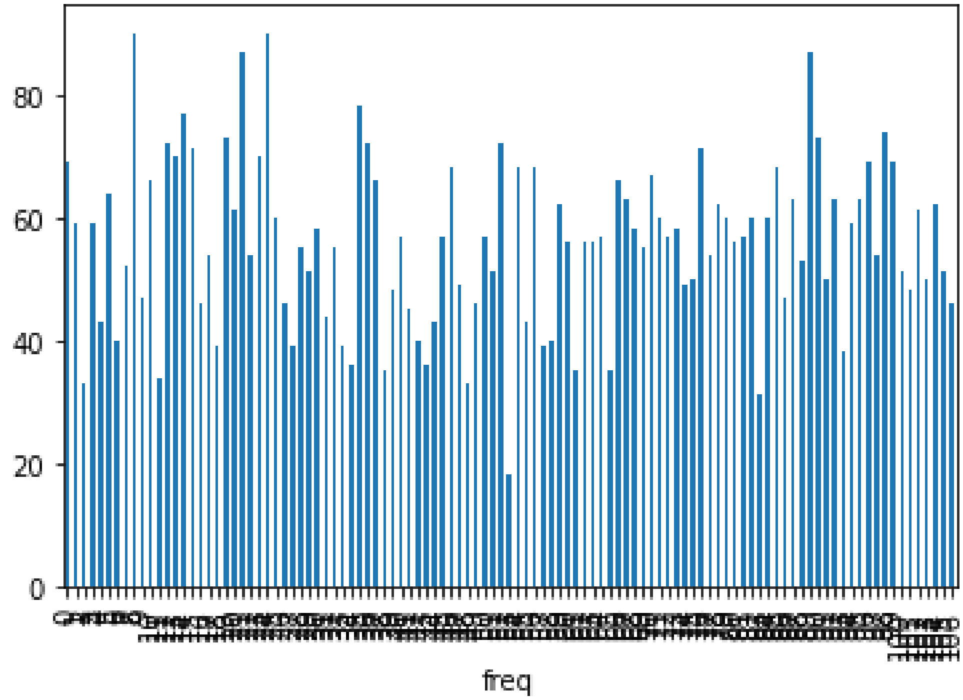
```
train_labels2 = train_labels

from tensorflow.keras.utils import to_categorical
train_labels = to_categorical(train_labels, 107)

print(f'After preprocessing, our dataset has {train_images.shape[0]} images␣
 ↪with shape {train_images.shape[1:]}')
print(f'After preprocessing, our dataset has {train_labels.shape[0]} rows with␣
 ↪{train_labels.shape[1]} labels')
#print(train_labels2)

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =␣
 ↪train_test_split(train_images,train_labels,test_size=0.15,shuffle=True)

print(f'After splitting, shape of our train dataset: {X_train.shape}')
print(f'After splitting, labels of our train dataset: {y_train.shape}')
print(f'After splitting, shape of our test dataset: {X_test.shape}')
print(f'After splitting, labels of our test dataset: {y_test.shape}')
```

```
After preprocessing, our dataset has 5982 images with shape (256, 256, 3)
After preprocessing, our dataset has 5982 rows with 107 labels
After splitting, shape of our train dataset: (5084, 256, 256, 3)
```

```
After splitting, labels of our train dataset: (5084, 107)
After splitting, shape of our test dataset: (898, 256, 256, 3)
After splitting, labels of our test dataset: (898, 107)
```

I will make a graph showing the distribution of the target classes. Since there are 107 possible poses, the graph is not interpretable.

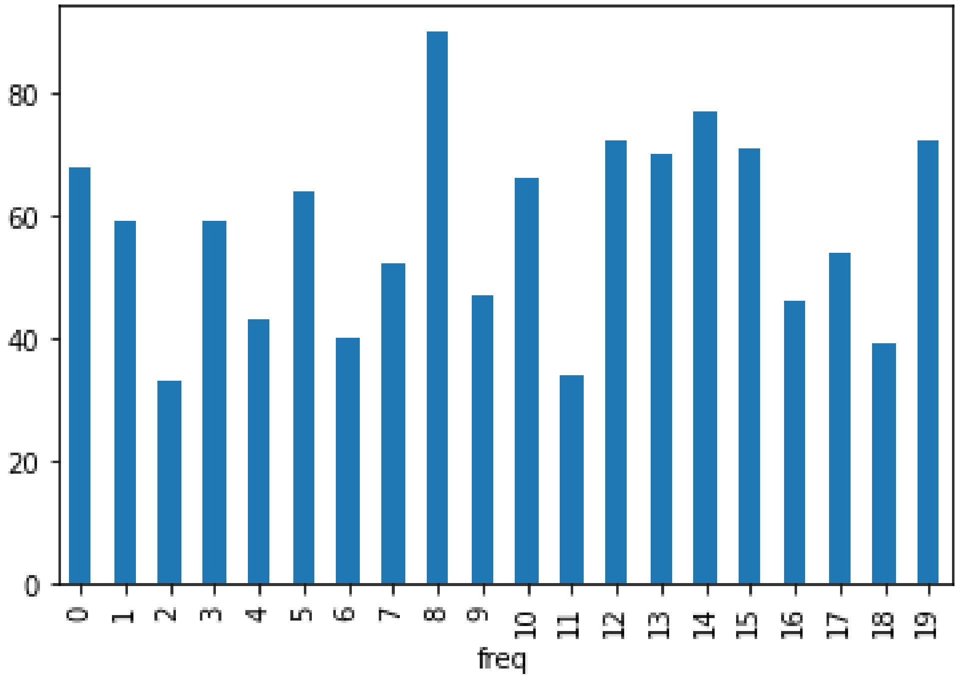
[8]:
```
#print(train_labels2)
df = pd.DataFrame({'freq': train_labels2})
df.groupby('freq').size().plot(kind='bar')
plt.show()
```

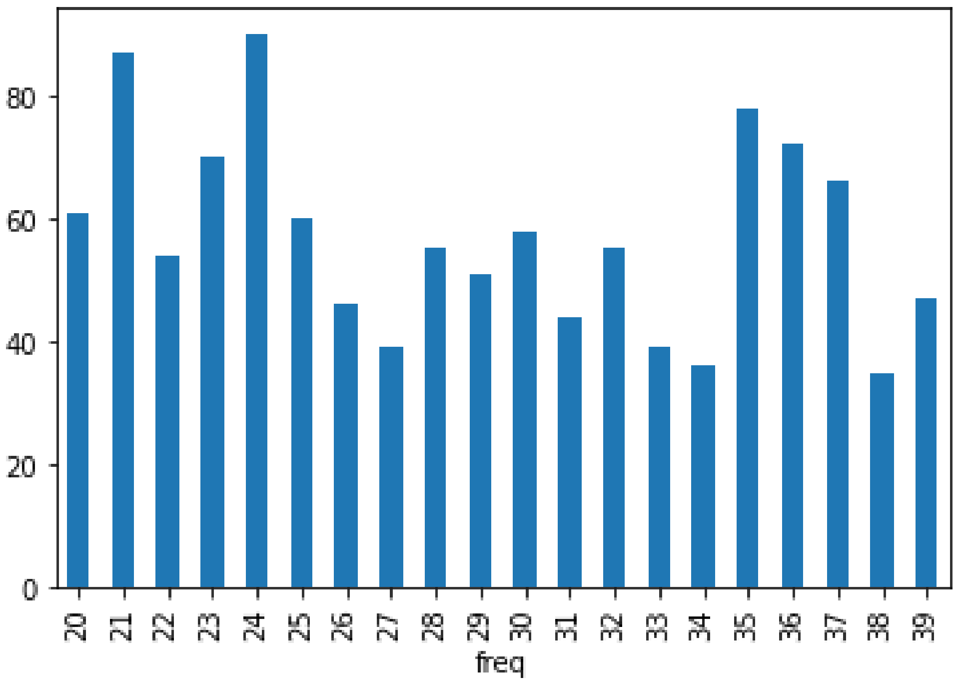


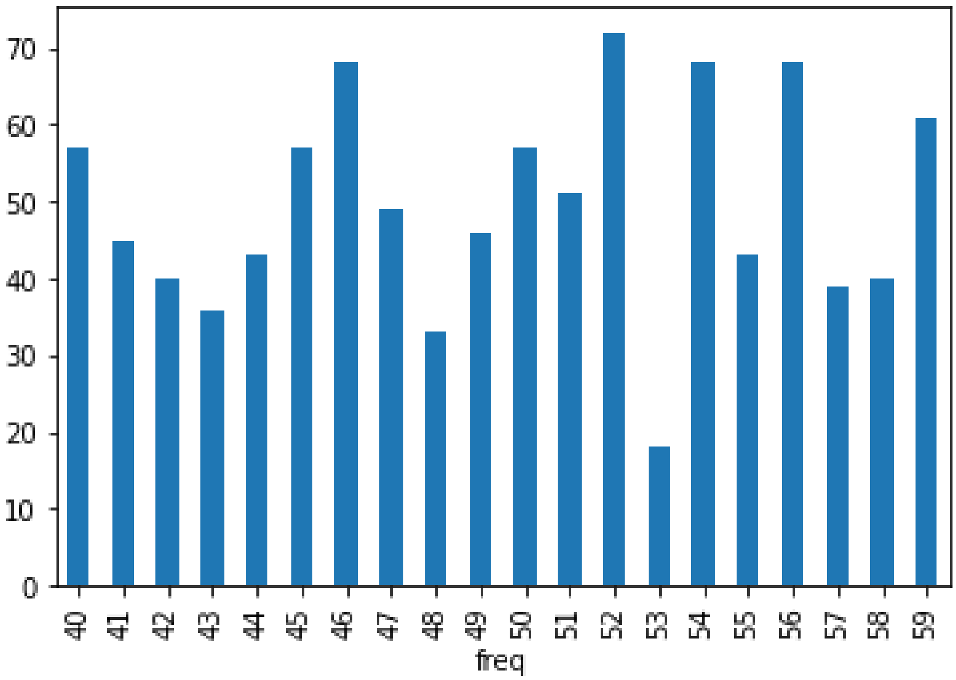
In this case, I will split the graphs into 5 different ones (with ~20 poses each). The starting indices will be 1158, 2302, 3294, and 4419. The last plot will contain 27 poses. I will make a new list, poses_freq, to store the number of pictures of each pose.

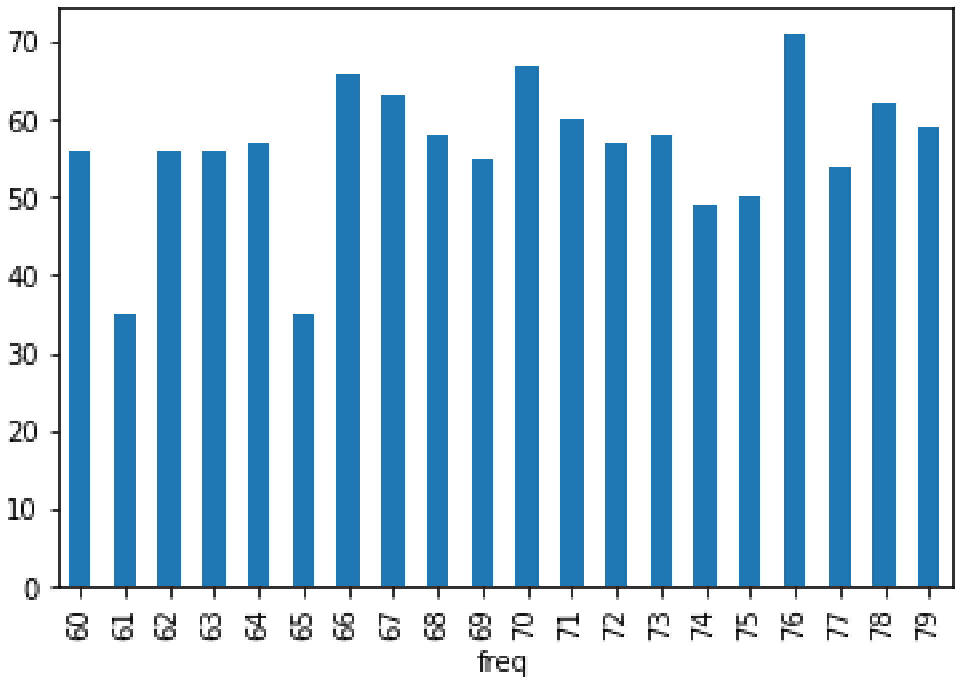
[9]:
```
#for i,label in enumerate(train_labels2):
#    if label!=0 and label%20==0:
#        print(i)

poses_freq = []
df = pd.DataFrame({'freq': train_labels2[1:1157]})
df.groupby('freq').size().plot(kind='bar')
#print(df.groupby('freq').size())
poses_freq.extend(df.groupby('freq').size())
```

```
plt.show()

df = pd.DataFrame({'freq': train_labels2[1158:2301]})
df.groupby('freq').size().plot(kind='bar')
poses_freq.extend(df.groupby('freq').size())
plt.show()

df = pd.DataFrame({'freq': train_labels2[2302:3293]})
df.groupby('freq').size().plot(kind='bar')
poses_freq.extend(df.groupby('freq').size())
plt.show()

df = pd.DataFrame({'freq': train_labels2[3294:4418]})
df.groupby('freq').size().plot(kind='bar')
poses_freq.extend(df.groupby('freq').size())
plt.show()

df = pd.DataFrame({'freq': train_labels2[4419:]})
df.groupby('freq').size().plot(kind='bar')
poses_freq.extend(df.groupby('freq').size())
plt.show()

#print(poses_freq)
#print(len(poses_freq))
```

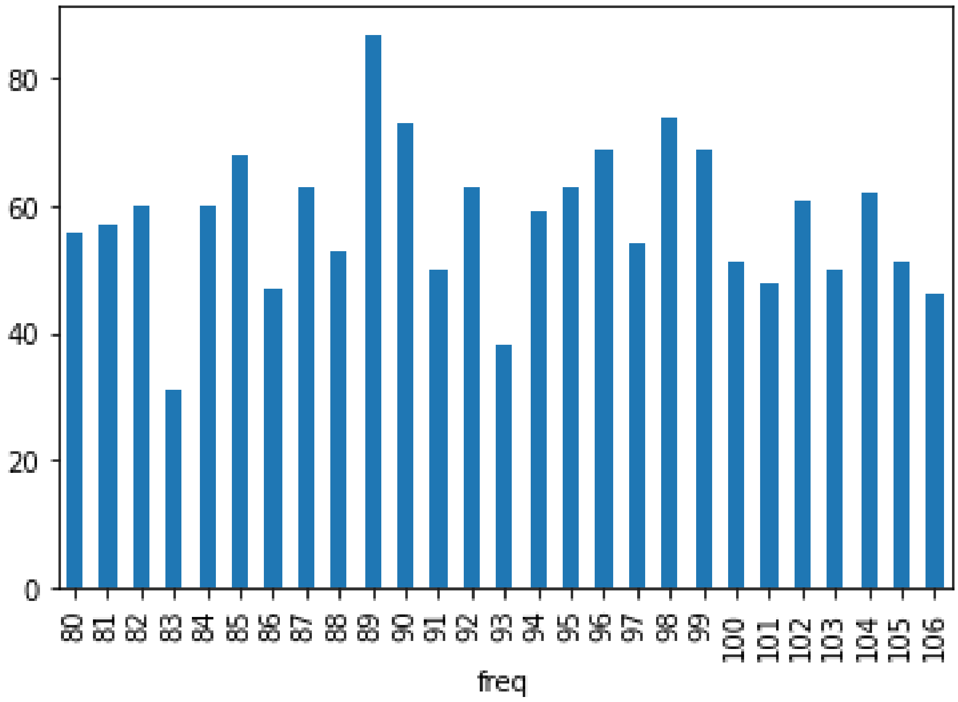



0
20
40
60
80
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
freq


0
10
20
30
40
50
60
70
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
freq



0
10
20
30
40
50
60
70
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
freq



0
20
40
60
80
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
freq

[10]:

```
import statistics
print(f"The maximum frequency is {max(poses_freq)} for the {labels[poses_freq.
 ↪index(max(poses_freq))]} pose and the minimum is {min(poses_freq)} for the␣
 ↪{labels[poses_freq.index(min(poses_freq))]} pose")
print(f"Range= {max(poses_freq)} - {min(poses_freq)} = {max(poses_freq) -␣
 ↪min(poses_freq)}")
print(f"Mean= {statistics.mean(poses_freq)}")
print(f"Median= {statistics.median(poses_freq)}")
print(f"Mode= {statistics.mode(poses_freq)}")
```

```
The maximum frequency is 90 for the ardha matsyendrasana pose and the minimum is
18 for the padangusthasana pose
Range= 90 - 18 = 72
Mean= 55.85981308411215
Median= 57
Mode= 57
```

After looking at all the graphs, there are at most 90 pictures for the same pose, with one having 18 pictures (the least). The range is 72, which is greater than the number of pictures for most of the poses. This difference in frequency might be because some poses are easier to distinguish than others and so might not need as many pictures identifying them. The mean is ~56 pictures, and the median and mode is 57 pictures. Here, the number of pictures is the number of observations (for each pose).

This dataset contains the name and images of 107 different yoga poses. The pictures are of people doing the pose and (sometimes) text describing features of the pose being done. There might be backgrounds in these pictures too. The model should be able to predict the name of the pose after viewing the image depicting it.

1.0.2 2. Sequential model and evaluate on test data

[27]:

```
from tensorflow.keras.utils import to_categorical

# convert class vectors to binary class matrices
#X_test = to_categorical(X_test, 107)
#y_test = to_categorical(y_test, 107)

#print(f'After converting to categorial type, shape of our train dataset:␣
 ↪{X_train.shape}')
#print(f'After converting to categorial type, labels of our train dataset:␣
 ↪{y_train.shape}')
#print(f'After converting to categorial type, shape of our test dataset:␣
 ↪{X_test.shape}')
#print(f'After converting to categorial type, labels of our test dataset:␣
 ↪{y_test.shape}')

sequential_model = tf.keras.models.Sequential(
```

```
    [
        tf.keras.layers.Flatten(input_shape=(256,256,3)),
        tf.keras.layers.Dense(512,activation='relu'),
        #tf.keras.layers.Dense(512,activation='relu'),
        tf.keras.layers.Dense(107,activation='softmax')
    ]
)

sequential_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 196608)            0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               100663808
_________________________________________________________________
dense_7 (Dense)              (None, 107)               54891
=================================================================
Total params: 100,718,699
Trainable params: 100,718,699
Non-trainable params: 0
_________________________________________________________________
```

[28]:

```
sequential_model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

batch_size = 32
num_classes = 10
epochs = 20
num_filters = 8
filter_size = 3
pool_size = 2

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
#X_train = X_train.reshape((15252, 256 * 256))
#X_train = X_train.astype("float32") / 255
#X_test = X_test.reshape((543988, 256 * 256))
#X_test = X_test.astype("float32") / 255

sequential_history = sequential_model.fit(X_train, y_train,
                    batch_size=batch_size,
```

```
                    epochs=10,
                    #verbose=1,
                    validation_split=0.2)
                    #validation_data=(X_test, y_test))
```

```
(5084, 256, 256, 3)
(898, 256, 256, 3)
(5084, 107)
(898, 107)
Epoch 1/10
128/128 [==============================] - 30s 232ms/step - loss: 26253.1367 -
accuracy: 0.0111 - val_loss: 5.0296 - val_accuracy: 0.0177
Epoch 2/10
128/128 [==============================] - 29s 226ms/step - loss: 5.5362 -
accuracy: 0.0148 - val_loss: 4.8783 - val_accuracy: 0.0177
Epoch 3/10
128/128 [==============================] - 28s 220ms/step - loss: 4.6674 -
accuracy: 0.0150 - val_loss: 4.8767 - val_accuracy: 0.0177
Epoch 4/10
128/128 [==============================] - 29s 223ms/step - loss: 4.6650 -
accuracy: 0.0150 - val_loss: 4.8751 - val_accuracy: 0.0177
Epoch 5/10
128/128 [==============================] - 28s 221ms/step - loss: 4.6627 -
accuracy: 0.0150 - val_loss: 4.8738 - val_accuracy: 0.0177
Epoch 6/10
128/128 [==============================] - 28s 221ms/step - loss: 4.6606 -
accuracy: 0.0150 - val_loss: 4.8728 - val_accuracy: 0.0187
Epoch 7/10
128/128 [==============================] - 29s 225ms/step - loss: 4.6586 -
accuracy: 0.0152 - val_loss: 4.8716 - val_accuracy: 0.0187
Epoch 8/10
128/128 [==============================] - 28s 223ms/step - loss: 4.6568 -
accuracy: 0.0152 - val_loss: 4.8708 - val_accuracy: 0.0187
Epoch 9/10
128/128 [==============================] - 29s 229ms/step - loss: 4.6551 -
accuracy: 0.0152 - val_loss: 4.8702 - val_accuracy: 0.0187
Epoch 10/10
128/128 [==============================] - 29s 224ms/step - loss: 4.6536 -
accuracy: 0.0152 - val_loss: 4.8695 - val_accuracy: 0.0187
```

[29]:

```
sequential_score = sequential_model.evaluate(X_test, y_test, verbose=0)
print("Test loss:", sequential_score[0])
print("Test accuracy:", sequential_score[1])
```

```
Test loss: 5.299704074859619
Test accuracy: 0.01336302887648344
```

Considering the fact that there were 107 possible classifications, an accuracy of ~1.34% should coincide with the probability when the model predicts all the pictures to be one pose.

1.0.3 3. CNN, RNN, and LSTM model and evaluate on test data

CNN

[30]:

```
from tensorflow.keras.utils import to_categorical

# convert class vectors to binary class matrices
#X_test = to_categorical(X_test, 107)
#y_test = to_categorical(y_test, 107)

#print(f'After converting to categorial type, shape of our train dataset:␣
 ↪{X_train.shape}')
#print(f'After converting to categorial type, labels of our train dataset:␣
 ↪{y_train.shape}')
#print(f'After converting to categorial type, shape of our test dataset:␣
 ↪{X_test.shape}')
#print(f'After converting to categorial type, labels of our test dataset:␣
 ↪{y_test.shape}')

cnn_model = tf.keras.models.Sequential(
    [
        tf.keras.Input(shape=(256,256,3)),
        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(0.1),
        tf.keras.layers.Rescaling(1.0 / 255),
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Flatten(input_shape=(256,256,3)),
        tf.keras.layers.Dense(512,activation='relu'),
        tf.keras.layers.Dense(107,activation='softmax')
    ]
)

cnn_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_flip (RandomFlip)     (None, 256, 256, 3)       0
_________________________________________________________________
random_rotation (RandomRotat (None, 256, 256, 3)       0
```

```
_________________________________________________________________
rescaling (Rescaling)        (None, 256, 256, 3)       0

_________________________________________________________________
conv2d (Conv2D)              (None, 254, 254, 32)      896

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0

_________________________________________________________________
dropout (Dropout)            (None, 127, 127, 32)      0

_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 64)      18496

_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 64)        0

_________________________________________________________________
dropout_1 (Dropout)          (None, 62, 62, 64)        0

_________________________________________________________________
flatten_3 (Flatten)          (None, 246016)            0

_________________________________________________________________
dense_8 (Dense)              (None, 512)               125960704

_________________________________________________________________
dense_9 (Dense)              (None, 107)               54891
=================================================================
Total params: 126,034,987
Trainable params: 126,034,987
Non-trainable params: 0

_________________________________________________________________
```

[35]:

```
cnn_model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

batch_size = 32
num_classes = 10
epochs = 20
num_filters = 8
filter_size = 3
pool_size = 2

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
#X_train = X_train.reshape((15252, 256 * 256))
#X_train = X_train.astype("float32") / 255
#X_test = X_test.reshape((543988, 256 * 256))
#X_test = X_test.astype("float32") / 255

cnn_history = cnn_model.fit(X_train, y_train,
```

```
                        batch_size=batch_size,
                        epochs=10,
                        #verbose=1,
                        validation_split=0.2)
                        #validation_data=(X_test, y_test))
```

```
(5084, 256, 256, 3)
(898, 256, 256, 3)
(5084, 107)
(898, 107)
Epoch 1/10
128/128 [==============================] - 230s 2s/step - loss: 4.0296 -
accuracy: 0.1015 - val_loss: 3.9712 - val_accuracy: 0.1268
Epoch 2/10
128/128 [==============================] - 213s 2s/step - loss: 3.8064 -
accuracy: 0.1330 - val_loss: 3.9342 - val_accuracy: 0.1219
Epoch 3/10
128/128 [==============================] - 187s 1s/step - loss: 3.5587 -
accuracy: 0.1773 - val_loss: 3.6529 - val_accuracy: 0.1829
Epoch 4/10
128/128 [==============================] - 191s 1s/step - loss: 3.3509 -
accuracy: 0.2134 - val_loss: 3.4945 - val_accuracy: 0.2252
Epoch 5/10
128/128 [==============================] - 194s 2s/step - loss: 3.1293 -
accuracy: 0.2604 - val_loss: 3.3866 - val_accuracy: 0.2222
Epoch 6/10
128/128 [==============================] - 197s 2s/step - loss: 2.8894 -
accuracy: 0.3007 - val_loss: 3.3277 - val_accuracy: 0.2360
Epoch 7/10
128/128 [==============================] - 196s 2s/step - loss: 2.7059 -
accuracy: 0.3418 - val_loss: 3.1913 - val_accuracy: 0.2704
Epoch 8/10
128/128 [==============================] - 213s 2s/step - loss: 2.5350 -
accuracy: 0.3737 - val_loss: 3.1827 - val_accuracy: 0.2881
Epoch 9/10
128/128 [==============================] - 210s 2s/step - loss: 2.3613 -
accuracy: 0.4170 - val_loss: 3.2229 - val_accuracy: 0.2999
Epoch 10/10
128/128 [==============================] - 201s 2s/step - loss: 2.2546 -
accuracy: 0.4345 - val_loss: 3.2406 - val_accuracy: 0.2802
```

[36]:

```
cnn_score = cnn_model.evaluate(X_test, y_test, verbose=0)
print("Test loss:", cnn_score[0])
print("Test accuracy:", cnn_score[1])
```

```
Test loss: 3.1400697231292725
Test accuracy: 0.3051224946975708
```

In this case, the accuracy reaches ~30%, which is more than the sequential model. (This could be because this model has more hidden layers.)

[11]:

```
from tensorflow.keras.utils import to_categorical

# convert class vectors to binary class matrices
#X_test = to_categorical(X_test, 107)
#y_test = to_categorical(y_test, 107)

#print(f'After converting to categorial type, shape of our train dataset:␣
 ↪{X_train.shape}')
#print(f'After converting to categorial type, labels of our train dataset:␣
 ↪{y_train.shape}')
#print(f'After converting to categorial type, shape of our test dataset:␣
 ↪{X_test.shape}')
#print(f'After converting to categorial type, labels of our test dataset:␣
 ↪{y_test.shape}')

cnn_model2 = tf.keras.models.Sequential(
    [
        tf.keras.Input(shape=(256,256,3)),
        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(0.1),
        tf.keras.layers.Rescaling(1.0 / 255),
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Flatten(input_shape=(256,256,3)),
        tf.keras.layers.Dense(512,activation='relu'),
        tf.keras.layers.Dense(107,activation='softmax')
    ]
)

cnn_model2.summary()
```

```
2022-12-05 00:31:51.894826: I
tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool
with default inter op setting: 2. Tune using inter_op_parallelism_threads for
best performance.
```

```
Model: "sequential"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
random_flip (RandomFlip)     (None, 256, 256, 3)       0

_________________________________________________________________
random_rotation (RandomRotat (None, 256, 256, 3)       0

_________________________________________________________________
rescaling (Rescaling)        (None, 256, 256, 3)       0

_________________________________________________________________
conv2d (Conv2D)              (None, 254, 254, 32)      896

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0

_________________________________________________________________
dropout (Dropout)            (None, 127, 127, 32)      0

_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 64)      18496

_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 64)        0

_________________________________________________________________
dropout_1 (Dropout)          (None, 62, 62, 64)        0

_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 64)        36928

_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0

_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0

_________________________________________________________________
flatten (Flatten)            (None, 57600)             0

_________________________________________________________________
dense (Dense)                (None, 512)               29491712

_________________________________________________________________
dense_1 (Dense)              (None, 107)               54891
=================================================================
Total params: 29,602,923
Trainable params: 29,602,923
Non-trainable params: 0

_________________________________________________________________
```

[12]:

```
cnn_model2.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

batch_size = 32
num_classes = 10
epochs = 20
num_filters = 8
filter_size = 3
pool_size = 2
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
#X_train = X_train.reshape((15252, 256 * 256))
#X_train = X_train.astype("float32") / 255
#X_test = X_test.reshape((543988, 256 * 256))
#X_test = X_test.astype("float32") / 255

cnn_history2 = cnn_model2.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=5,
                    #verbose=1,
                    validation_split=0.2)
                    #validation_data=(X_test, y_test))
```

```
(5084, 256, 256, 3)
(898, 256, 256, 3)
(5084, 107)
(898, 107)
Epoch 1/5
```

```
2022-12-05 00:31:58.510783: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```

```
128/128 [==============================] - 209s 2s/step - loss: 4.7779 -
accuracy: 0.0125 - val_loss: 4.6687 - val_accuracy: 0.0157
Epoch 2/5
128/128 [==============================] - 206s 2s/step - loss: 4.6633 -
accuracy: 0.0155 - val_loss: 4.6625 - val_accuracy: 0.0157
Epoch 3/5
128/128 [==============================] - 210s 2s/step - loss: 4.6504 -
accuracy: 0.0187 - val_loss: 4.6285 - val_accuracy: 0.0334
Epoch 4/5
128/128 [==============================] - 204s 2s/step - loss: 4.5688 -
accuracy: 0.0344 - val_loss: 4.4349 - val_accuracy: 0.0796
Epoch 5/5
128/128 [==============================] - 202s 2s/step - loss: 4.2394 -
accuracy: 0.0821 - val_loss: 3.8694 - val_accuracy: 0.1426
```

[13]:
```
cnn_score2 = cnn_model2.evaluate(X_test, y_test, verbose=0)
print("Test loss:", cnn_score2[0])
print("Test accuracy:", cnn_score2[1])
```

```
Test loss: 3.8885796070098877
Test accuracy: 0.13474386930465698
```

Interestingly, adding another layer decreased the accuracy to 13.5%.

RNN Now, I'm trying a RNN model.

[5]:

```
rnn_model = tf.keras.models.Sequential()
rnn_model.add(tf.keras.layers.Flatten(input_shape=(256,256,3)))
rnn_model.add(tf.keras.layers.Embedding(10000, 32))
rnn_model.add(tf.keras.layers.SimpleRNN(512))
rnn_model.add(tf.keras.layers.Dense(107, activation='sigmoid'))

rnn_model.summary()
```

```
2022-12-05 00:53:56.529350: I
tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool
with default inter op setting: 2. Tune using inter_op_parallelism_threads for
best performance.
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
flatten (Flatten)           (None, 196608)            0

_________________________________________________________________
embedding (Embedding)       (None, 196608, 32)        320000

_________________________________________________________________
simple_rnn (SimpleRNN)      (None, 512)               279040

_________________________________________________________________
dense (Dense)               (None, 107)               54891
=================================================================
Total params: 653,931
Trainable params: 653,931
Non-trainable params: 0

_________________________________________________________________
```

[ ]:

```
"""
rnn_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train
rnn_history = rnn_model.fit(X_train, y_train,
                    batch_size=32,
                    epochs=5,
                    #verbose=1,
                    validation_split=0.2)
                    #validation_data=(X_test, y_test))
"""
```

```
2022-12-05 00:54:19.326056: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```

```
Epoch 1/5
```

This model takes up more memory than is accessible so I am unable to determine how well it works.

LSTM Now I will try a LSTM model.

[9]:

```
# build a model with LSTM
LSTM_model = tf.keras.models.Sequential()
LSTM_model.add(tf.keras.layers.Flatten(input_shape=(256,256,3)))
LSTM_model.add(tf.keras.layers.Embedding(10000, 32))
LSTM_model.add(tf.keras.layers.LSTM(512))
LSTM_model.add(tf.keras.layers.Dense(107, activation='softmax'))

LSTM_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 196608)            0

_________________________________________________________________
embedding_3 (Embedding)      (None, 196608, 32)        320000

_________________________________________________________________
lstm_3 (LSTM)                (None, 512)               1116160

_________________________________________________________________
dense_3 (Dense)              (None, 107)               54891
=================================================================
Total params: 1,491,051
Trainable params: 1,491,051
Non-trainable params: 0

_________________________________________________________________
```

[ ]:

```
"""
LSTM_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train
LSTM_history = LSTM_model.fit(X_train, y_train,
                    batch_size=32,
                    epochs=5,
                    #verbose=1,
                    validation_split=0.2)
                    #validation_data=(X_test, y_test))
```

```
"""
```

```
Epoch 1/5
```

This model also takes up more memory than is accessible so I am unable to determine how well it works.

1.0.4 4. Pretrained model and transfer learning

I will use the VGG16 model as a pretrained model with my input (aka yoga pose images and labels).

[9]:

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

vgg_model = VGG16(weights="imagenet", include_top=False,␣
 ↪input_shape=(256,256,3))
vgg_model.trainable = False

X_train1 = preprocess_input(X_train)
X_test1 = preprocess_input(X_test)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

[14]:

```
from tensorflow.keras import layers, models

model_tl = tf.keras.models.Sequential([
    vgg_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(107, activation='softmax')
])
```

[16]:

```
from tensorflow.keras.callbacks import EarlyStopping

model_tl.compile( optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)



es = EarlyStopping(monitor='val_accuracy', mode='max',
                    patience=5,  restore_best_weights=True)
```

```
history_model_tl = model_tl.fit(X_train, y_train, epochs=5,
                                validation_split=0.2, batch_size=32,␣
 ↪callbacks=[es])
```

```
Epoch 1/5
128/128 [==============================] - 1807s 14s/step - loss: 7.7418 -
accuracy: 0.1151 - val_loss: 4.5148 - val_accuracy: 0.1318
Epoch 2/5
128/128 [==============================] - 1658s 13s/step - loss: 3.0744 -
accuracy: 0.3280 - val_loss: 3.8422 - val_accuracy: 0.2527
Epoch 3/5
128/128 [==============================] - 1653s 13s/step - loss: 2.0786 -
accuracy: 0.5208 - val_loss: 3.8104 - val_accuracy: 0.2989
Epoch 4/5
128/128 [==============================] - 1711s 13s/step - loss: 1.6206 -
accuracy: 0.6339 - val_loss: 3.7658 - val_accuracy: 0.3520
Epoch 5/5
128/128 [==============================] - 1691s 13s/step - loss: 1.0090 -
accuracy: 0.7603 - val_loss: 3.7276 - val_accuracy: 0.3540
```

[18]:
```
score_tl = model_tl.evaluate(X_test, y_test, verbose=0)
print("Test loss:", score_tl[0])
print("Test accuracy:", score_tl[1])
```

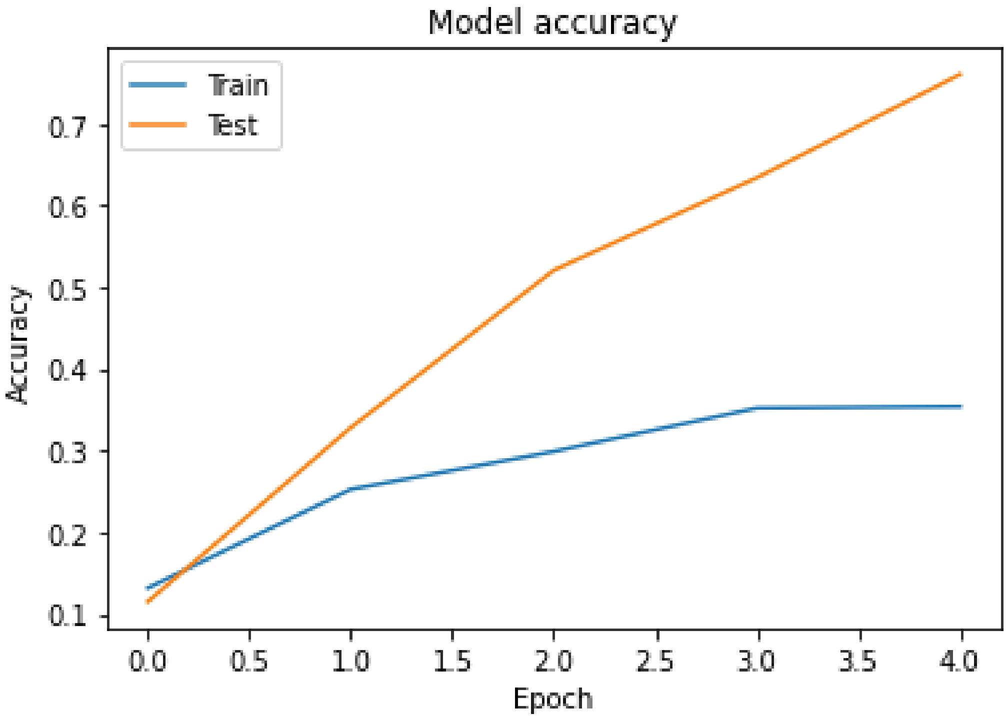
```
Test loss: 3.2549431324005127
Test accuracy: 0.38752785325050354
```

Although it took much longer to run this model, it gives an accuracy of ~40%, which is impressive considering that there are 107 poses the images can be classified into.

1.0.5 5. Analysis

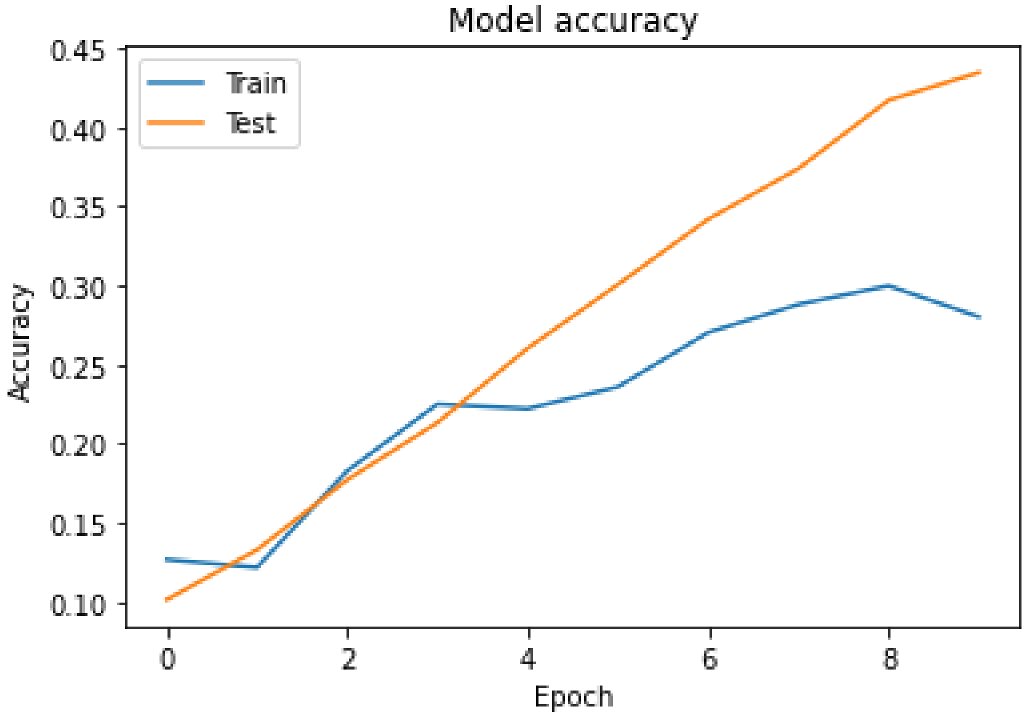
The model with the highest accuracy was the pretrained model, with an accuracy of 38.75%. There are many layers and steps in that model, which explains the long runningtime. If I had (a lot) more time, I could have gotten better accuracy by increasing the number of epochs. This can be seen in the graph below. The test loss was ~3.25.

[21]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_model_tl.history['val_accuracy'])
plt.plot(history_model_tl.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



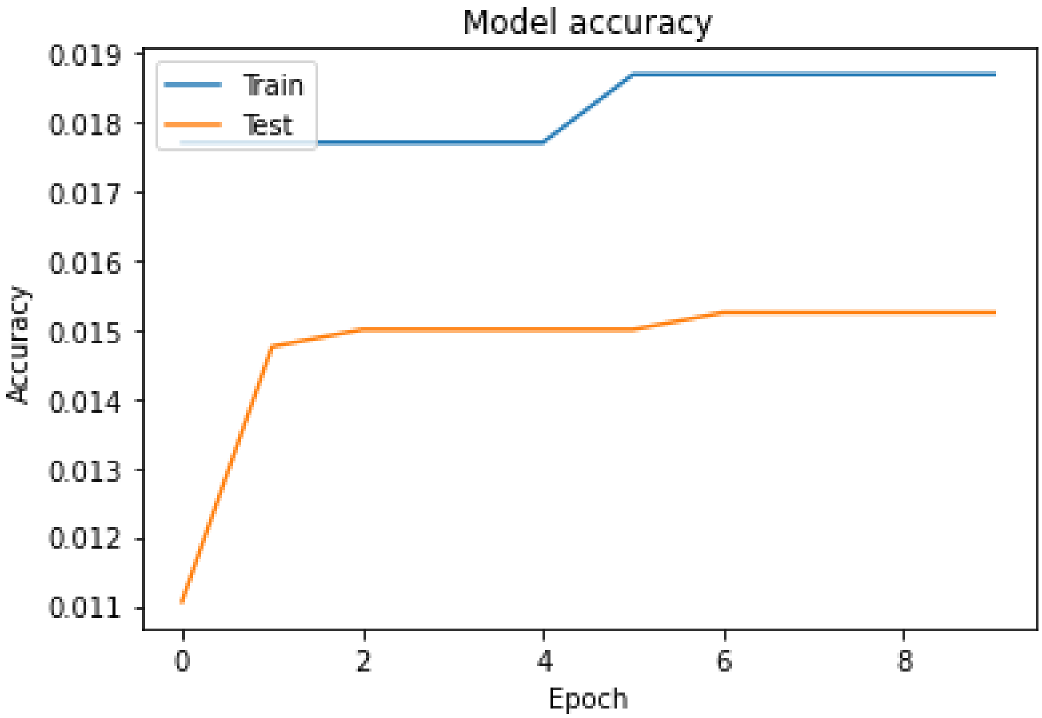
The next best model is the CNN model, with an accuracy of 30.5% and a loss of 3.14. I saw some potential in this model, but the next one I made had an accuracy of 13.5% and loss of 3.89. It seems like increasing the epochs might not have helped in increasing the accuracy.

[37]:
```
# Plot training & validation accuracy values
plt.plot(cnn_history.history['val_accuracy'])
plt.plot(cnn_history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



The worst model is the sequential model, with test loss of about 5.30 and an accuracy of 1.33%. This was by far the simplest model, which proved to be insufficient in classifying my dataset. Increasing the epochs in this case would not necessarily increase the accuracy.

[32]:
```
# Plot training & validation accuracy values
plt.plot(sequential_history.history['val_accuracy'])
plt.plot(sequential_history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Since I was unable to create and run a RNN and LSTM model, I am unsure as to how it would have done with my dataset. I can make an educated guess that since RNN works better on text processing and CNN is better at classifying images, any RNN model I would've made would not have done better than the CNN models I have.