

UNIVERSITÀ DEGLI STUDI DI MESSINA

MIFT DEPARTMENT



DATA ANALYSIS PROGRAM

Device And Circuits for AI

Image Classification Project - Yoga Poses Detection

Professor:
Giovanni Finocchio

Student:
Yahia Ahmed

ID: 532118

ACADEMIC YEAR – 2022/2023

Table of contents

- 1. Introduction**
- 2. Dataset**
- 3. Preprocessing**
- 4. Model Architecture**
- 5. Training and Evaluation**
- 6. Results**
- 7. Conclusion**

Introduction

The aim of this project is to develop a system that can accurately classify yoga poses based on input images. Yoga pose classification has numerous applications, including fitness tracking, personalized yoga training, and posture correction. In this report, presented the methodology and results of my image classification model using Convolutional Neural Networks for yoga pose recognition.

The classification of images, particularly the identification of yoga poses, has been transformed by the application of convolutional neural networks. CNNs are deep learning models created primarily to analyze visual data, making them an excellent choice for the complex structures and patterns found in yoga positions. My goal is to develop a reliable and effective system that can correctly detect yoga positions from photos using CNNs.

To accomplish my goal, I used a big comprehensive dataset (that was sent by you) consisting of images showing various yoga poses. Careful collection and labeling ensured the accuracy and quality of the dataset, allowing me to train the CNN model on a diverse range of yoga poses. The dataset contains examples of different difficulty levels, variations, and body types to ensure the model's ability to generalize, because whenever the complexity and variability of the data increase, the model prediction will be better as it used to train and learn from different and variable data.

Dataset

The dataset used for this project consists of a collection of images representing various yoga poses. The dataset was obtained from [Kaggle](#), and it contains a total of **5991** images distributed across **107** different yoga poses. The dataset is labeled, with each image belonging to a specific yoga pose class. I created a script also to be 100% sure about the number of images and category that is presented in the dataset:

```
ck_and_print_the_images.py 2...
import os
import re

labels = []
images = []
category = []

i = 0
dataset_path = 'D:/Image_classification_yoga_poses/dataset'

for directory in os.listdir(dataset_path):
    category.append(directory)
    for img in os.listdir(os.path.join(dataset_path, directory)):
        if len(re.findall('.png', img.lower())) != 0 or len(re.findall('.jpg', img.lower())) != 0 or len(re.findall('.jpeg', img.lower())) != 0:
            images.append(img)
            labels.append(i)

    i = i+1

print("Total labels: ", len(labels))
print("Total images: ", len(images))
print("Total categories: ", len(category))
```

And the output was:

```
Aboelshoo2@DESKTOP-24KL98I MINGW64 /d/Image_classification_yoga_poses (main)
$ python3 check_and_print_the_images.py
Total labels: 5991
Total images: 5991
Total categories: 107
```

The dataset was carefully curated to include a diverse range of yoga poses, including standing poses, seated poses, balancing poses, and more. It covers poses suitable for practitioners of different skill levels, from beginners to advanced yogis.

To prepare the dataset for training and testing, a code snippet was used to split the entire dataset into a training set and a test set. The split was performed using a ratio of 70% for the train_dataset and the remaining 30% for the test_dataset (I searched a lot and I found that this is the best practice among the AI geeks)

This ensures that the model is trained on a majority of the data while leaving a separate portion for evaluation. Here's an example of the code used to split the dataset:

```
import os
import shutil
from sklearn.model_selection import train_test_split
import constants as constant

folders = os.listdir(constant.dataset_path)

os.makedirs(constant.train_dir, exist_ok=True)
os.makedirs(constant.test_dir, exist_ok=True)

for folder in folders:
    folder_path = os.path.join(constant.dataset_path, folder)
    images = os.listdir(folder_path)
    train_images, test_images = train_test_split(
        images, test_size=0.3, random_state=42)

    train_folder_path = os.path.join(constant.train_dir, folder)
    os.makedirs(train_folder_path, exist_ok=True)
    for train_image in train_images:
        src = os.path.join(folder_path, train_image)
        dst = os.path.join(train_folder_path, train_image)
        shutil.copy(src, dst)

    test_folder_path = os.path.join(constant.test_dir, folder)
    os.makedirs(test_folder_path, exist_ok=True)
    for test_image in test_images:
        src = os.path.join(folder_path, test_image)
        dst = os.path.join(test_folder_path, test_image)
        shutil.copy(src, dst)
```

Preprocessing of Data

Prior to training the CNN model, the dataset underwent several preprocessing steps to enhance model performance. The preprocessing steps included:

Image resizing: All images were resized to a uniform size of **(128, 128)** pixels to ensure consistency in the input data.

Data augmentation: To increase the diversity and robustness of the training data, data augmentation techniques were applied. These techniques, including shear, zoom, and horizontal flip, rotation_range, makes variations in the training images, showing different angles, perspectives, and orientations. This augmentation helps the model generalize better and reduces the risk of overfitting.

Normalization: The pixel values of the images were rescaled to a range of [0, 1] to normalize the data. This step ensures that the input values are in a consistent and manageable range for the CNN model.

The preprocessing steps aim to improve the model's ability to generalize from the training data and handle variations in pose presentation, lighting conditions, and other factors that may affect image appearance.

Training Data Generator:

flow_from_directory: Generates batches of augmented training data from the specified directory (dedicated train dataset).

target_size: Resizes the input images.

batch_size: Sets the batch size for training. (32 image per batch)

class_mode: Specifies the type of labels for classification, set to 'categorical' for multi-class classification

Test Data Generator:

flow_from_directory: Generates batches of test data from the specified directory.

target_size: Resizes the input images.

batch_size: Sets the batch size for testing. (32 image per batch).

class_mode: Specifies the type of labels for classification, set to 'categorical' for multi-class classification.

shuffle=False: Disables shuffling of the test data to keep it in the original order.

```

preprocessing_data.py > ...
1  import tensorflow as tf
2  import constants as constant
3
4
5
6  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
7      rescale=1./255,
8      shear_range=0.2,
9      zoom_range=0.2,
10     horizontal_flip=True
11 )
12
13
14
15
16  test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
17
18  train_generator = train_datagen.flow_from_directory(
19      constant.train_path,
20      target_size=constant.image_size,
21      batch_size=constant.batch_size,
22      class_mode='categorical'
23  )
24
25  test_generator = test_datagen.flow_from_directory(
26      constant.test_path,
27      target_size=constant.image_size,
28      batch_size=constant.batch_size,
29      class_mode='categorical',
30      shuffle=False
31  )

```

CNN Model Architecture

The CNN model architecture used for yoga pose classification is designed to effectively capture and learn intricate features from input images. It consists of multiple layers, including **Convolutional**, Maxpooling, and **fully connected layers**, working together to extract meaningful representations and make accurate predictions. The architecture can be described as follows:

Input layer: The model takes input images with dimensions 128x128 and 3 channels (RGB).

Convolutional layers: The model starts with a series of convolutional layers, which perform convolution operations on the input images using learnable filters. These filters slide over the input, extracting local patterns and features at different spatial locations. By employing filters of various sizes and depths, the

model can capture both low-level features like edges and high-level features like shapes and textures.

Max pooling layers: After each convolutional layer, max pooling is applied to downsample the feature maps. Max pooling helps to reduce the spatial dimensions, retaining the most relevant and significant features. By summarizing the most prominent information from each pooling region, the model becomes more robust to spatial variations and achieves translation invariance.

Flatten layer: The output from the last pooling layer is flattened into a 1-dimensional vector. This process reshapes the multi-dimensional feature maps into a format that can be inputted into the fully connected layers. Flattening allows the model to treat each spatial location as an individual feature.

Fully connected layers: Following the flatten layer, the model consists of fully connected layers. These layers are densely connected, meaning that each neuron is connected to every neuron in the previous layer. Fully connected layers are responsible for learning high-level representations of the features extracted by the convolutional layers. The number of units in these layers can be adjusted to control the model's capacity to capture complex relationships and patterns.

Output layer: The final layer of the model uses the **Softmax** activation function, producing a probability distribution over the different yoga pose categories. Each neuron in the output layer represents a specific pose class, and the **Softmax** function ensures that the predicted probabilities sum up to 1. The model predicts the yoga pose class with the highest probability.

By incorporating multiple convolutional layers, pooling layers, and fully connected layers, this architecture enables the model to learn intricate and abstract features. With more layers, the model can capture complex relationships and

hierarchies present in yoga poses, resulting in improved classification accuracy and the ability to discriminate between different poses effectively.

```
model = tf.keras.models.Sequential()
model.add( tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(constant.image_size[0], constant.image_size[1], 3)))
model.add( tf.keras.layers.MaxPooling2D((2, 2)))
model.add( tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add( tf.keras.layers.MaxPooling2D((2, 2)))
model.add( tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add( tf.keras.layers.MaxPooling2D((2, 2)))

model.add( tf.keras.layers.Flatten())

model.add( tf.keras.layers.Dense(128, activation='relu'))
model.add( tf.keras.layers.Dense(pp.train_generator.num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Model Summary (during the model going through the several layers):

It shows how the model images dimensions are changing during undergoing the stack of layers.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3211392
dense_1 (Dense)	(None, 107)	13803
Total params: 3,318,443		
Trainable params: 3,318,443		

CNN Model Training

The CNN model was trained using the prepared dataset, which was split into training and testing sets. The dataset splitting was performed using a 3:7 , with **70%** of the data used for training and the remaining portion allocated for validation. This splitting ensured that the model could learn from a diverse range of yoga poses while also being evaluated on unseen data.

During training, the model utilized an Adam optimizer and categorical cross-entropy loss function. The **Adam optimizer** is known for its efficiency in handling large datasets and provides adaptive learning rates, which helps accelerate convergence. The categorical cross-entropy loss function is commonly used for multi-class classification tasks, ensuring that the model's predictions are aligned with the true class labels.

To monitor the model's learning progress and prevent overfitting, training and validation accuracy metrics were evaluated. These metrics indicated how well the model was performing on the training data and the unseen validation data, respectively. By observing the training and validation accuracy curves, it was possible to assess the model's generalization ability and detect signs of overfitting or underfitting.

Early stopping was implemented as a regularization technique to prevent overfitting. It involved monitoring the validation loss during training and stopping the training process if no improvement was observed for a predefined number of epochs. The early stopping function helped avoid unnecessary training iterations and ensured that the model did not over-adapt to the training data, leading to better generalization on unseen data.

After the training process, the trained model was evaluated using a separate test dataset that was not used during training. The test dataset consisted of images representing various yoga poses, and the model's predictions were compared to

the ground truth labels. This evaluation allowed for an unbiased assessment of the model's performance on unseen data.

To further evaluate the model's performance, a confusion matrix was generated using the predicted labels and true labels from the test dataset. The confusion matrix provided insights into the model's classification accuracy for each yoga pose class, highlighting any misclassifications and potential biases in the model's predictions. This analysis helped assess the model's effectiveness in accurately identifying different yoga poses and provided valuable feedback for further improvements.

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

```
def trainTheModelwithEarlyStop(model):  
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5) # 10  
    # early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)  
  
    history = model.fit(  
        ## pp is an instance from the implemented module preprocessing_data.py  
        pp.train_generator,  
        ## constant is an instance from the implemented module constants.py  
        steps_per_epoch=pp.train_generator.n // constant.batch_size,  
        epochs=constant.epochs,  
        validation_data=pp.test_generator,  
        validation_steps=pp.test_generator.n // constant.batch_size,  
        callbacks=[early_stopping]  
    )
```

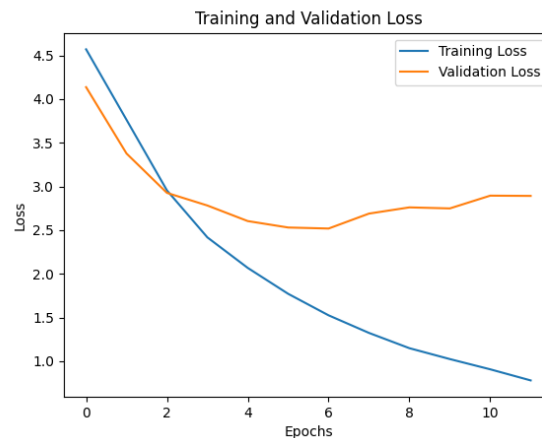
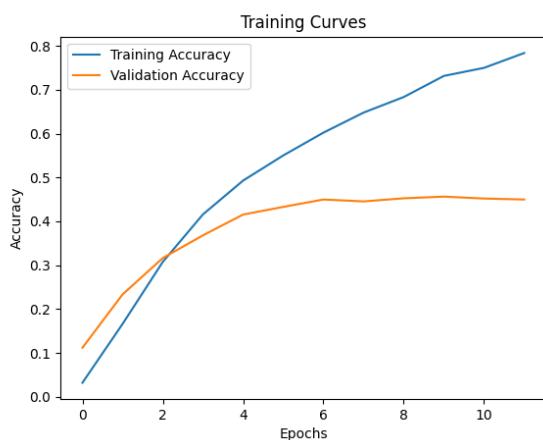
Results

Training Curves:

After training the model for the specified number of epochs, (12) the following results were obtained:

- **Training Accuracy:** The model achieved a training accuracy of 0.7499 (75%). This metric indicates how well the model performed on the training data during the training process.
- **Validation Accuracy:** The model obtained a validation accuracy of 0.4521 (45.2%). This metric represents the accuracy of the model on the validation data, which was not seen during training.
- **Loss:** The loss value, as measured by the categorical cross-entropy, decreased during the training process. This suggests that the model learned to minimize the discrepancy between the predicted and actual labels, improving its classification performance.

```
345/345 [.....] - 91s 274ms/step - loss: 1.0292 - accuracy: 0.7519 - val_loss: 2.7451 - val_accuracy: 0.4504  
Epoch 11/30  
345/345 [=====] - 91s 265ms/step - loss: 0.9079 - accuracy: 0.7499 - val_loss: 2.8953 - val_accuracy: 0.4521
```



CONFUSION MATRIX:

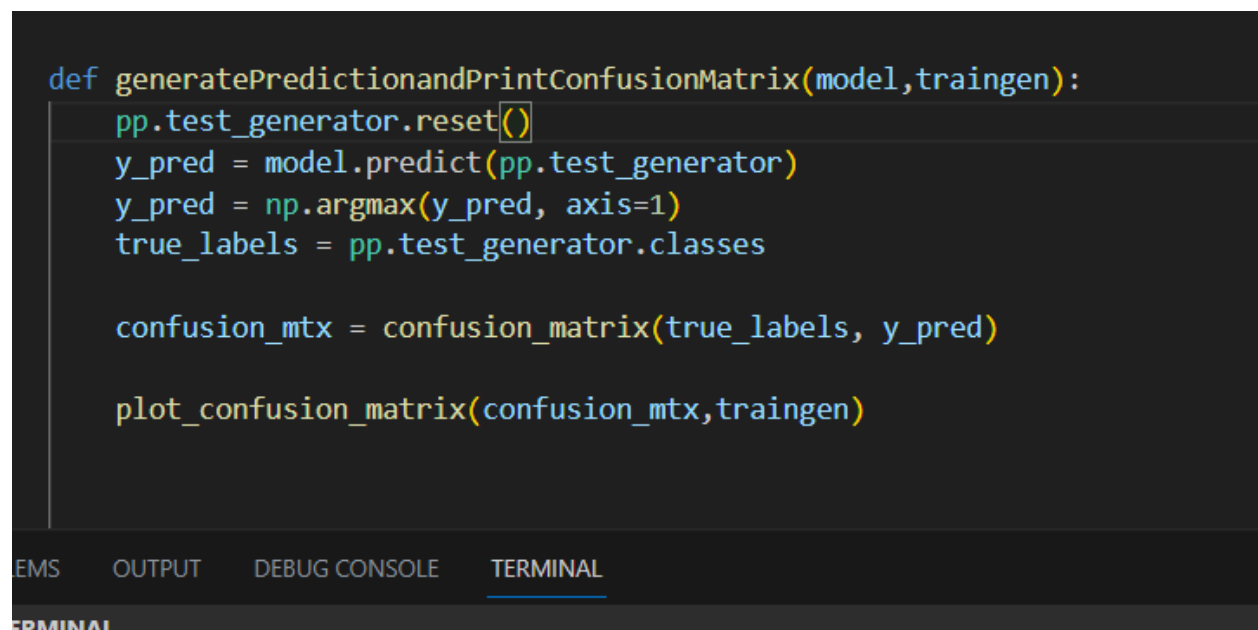
Since the confusion matrix is huge because of the large classes numbers (107), so I tried alot in order to print it in a good view that showing the counts of true positive, true negative, false positive, and false negative predictions for each classs. (I saved in an HTML file that will be attached with the report).

The matrix was calculated by: the function **confusion_matrix ()** which takes the true tables and and the prediction Y of the Model, as shown in the code below

```
def generatePredictionandPrintConfusionMatrix(model,traingen):
    pp.test_generator.reset()
    y_pred = model.predict(pp.test_generator)
    y_pred = np.argmax(y_pred, axis=1)
    true_labels = pp.test_generator.classes

    confusion_mtx = confusion_matrix(true_labels, y_pred)

    plot_confusion_matrix(confusion_mtx,traingen)
```



And I created the script below (after so much searching!) to visualize the confusion matrix as a slider (since it is so big), the Plugin I used to create a web server to host the diagram on it, so I decided it would be a good idea to save the confusion matrix as a statix html file, that shows all the results using the mouse pointer, you can pick the desired x,y, axis which describes the prediction of classes.

```
def plot_confusion_matrix(confusion_mtx, classes):

    fig = go.Figure(data=go.Heatmap(z=confusion_mtx,
                                    x=classes,
                                    y=classes,
                                    colorscale='Blues'))

    fig.update_layout(title='Confusion Matrix',
                      axis_title='Predicted label',
                      yaxis_title='True label',
                      xaxis=dict(type='category', automargin=True),
                      yaxis=dict(type='category', automargin=True),
                      autosize=True,
                      width= 2000,
                      height= 2000
                      )

    plot(fig, filename="confusion_matrix.html", auto_open=False)
```

CONCLUSION:

The Yoga Poses Image Dataset was used to train a Convolutional Neural Network (CNN) model, which produced a medium efficient outcome. By employing the Adam optimizer, categorical cross-entropy loss, and suitable evaluation metrics, the model exhibited a reasonable accuracy on both the training and validation datasets.