LE BAO QUAN - SE160758 - AI1601

MAI391 Final Project

Link to project on Github

ROCK, PAPER, SCISSORS CLASSIFICATION WITH MACHINE-LEARNING

Overview:

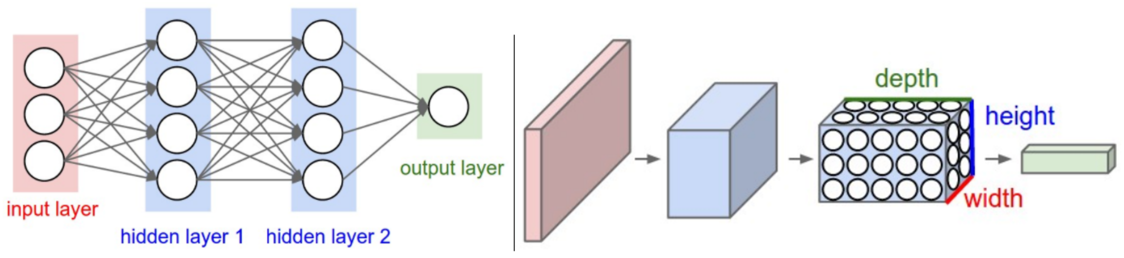
Purpose of this research paper is to apply the learn and apply Convolutional Neural Network (CNN) model into image classification problem. In particular, to classify between Rock - Paper - Scissors. While doing so, this paper also discussed why CNN model is needed for image processing over basic fully-connected network.

This paper consists of 7 parts in total:
Firstly the purposes of this project is mentioned. Secondly, technical background requirements is listed. Then, data exploration. Next, the method of approach is discussed. Build and refine model comes next. Results and in depth discussion will be followed. And lastly, appendix.

Background:

Basic knowledge for this research includes basic Artificial Neural Network (ANN - fully connected) and Convolutional Neural Network (CNN).

```
- comparison between ANN and CNN
```



Basic ANN:

- fully connected layers in ANN is sometimes called “densely connected”.
- All possible connections layer to layer are present, meaning every input of the input vector influences every output of the output vector.
- However, not all weights affect all outputs.
    - Which leads to the needs of CNN, especially in image processing

  Basic CNN:

    - Instead of doing all dot products between all weights and inputs, convolutional layer introduces a 'kernel'
    - The kernel shifts along the input matrix (i.e. the image), and get the dot product as usual.

    - By doing do, CNN can effectively perform dot product for input images (which often have very high dimensional input data) in a much shorter time, giving it much more flexibility.
    - A common CNN consists of the following layers
        - Convolutional layers: results in an output of similar width and height, but change in depth (depth = number of filters)
        - ReLU layer: activation function for thresholding at 0, keep the size the same
        - Pool layer: to reduce the size, to prevent overfitting
        - Lastly, a fully connected (Dense) layer to produce the output (shape 1x1xN where N is the number of categories)

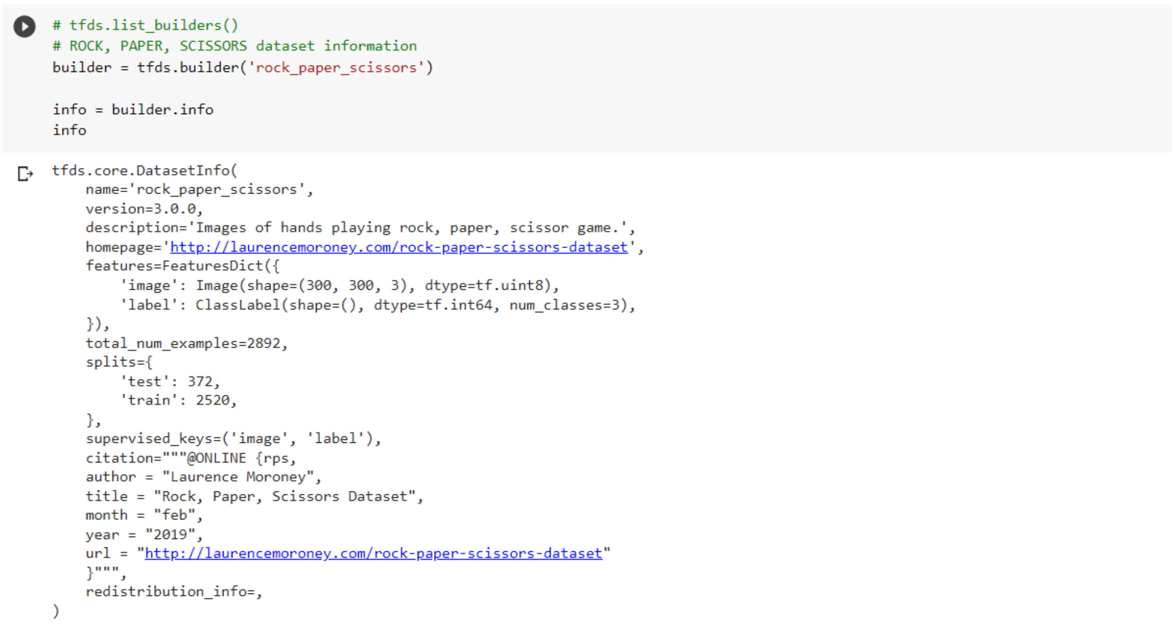
Data Preparation:

- *Where to get the data:*

  The dataset is taken from Tensorflow Datasets

  ```
  import tensorflow_datasets as tfds
  builder = tfds.builder('rock_paper_scissors')
  ```

    - Detail information of the dataset is as below:



```
# tfds.list_builders()
# ROCK, PAPER, SCISSORS dataset information
builder = tfds.builder('rock_paper_scissors')

info = builder.info
info
```

```
tfds.core.DatasetInfo(
    name='rock_paper_scissors',
    version=3.0.0,
    description='Images of hands playing rock, paper, scissor game.',
    homepage='http://laurencemoroney.com/rock-paper-scissors-dataset',
    features=FeaturesDict({
        'image': Image(shape=(300, 300, 3), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=3),
    }),
    total_num_examples=2892,
    splits={
        'test': 372,
        'train': 2520,
    },
    supervised_keys=('image', 'label'),
    citation="""@ONLINE {rps,
    author = "Laurence Moroney",
    title = "Rock, Paper, Scissors Dataset",
    month = "feb",
    year = "2019",
    url = "http://laurencemoroney.com/rock-paper-scissors-dataset"
    }""",
    redistribution_info=,
)
```

        - The dataset includes 2520 (test) and 372 (train) images of resolution 300x300 showing rocks, papers or scissors

- some sample images:

- *Data pre-processing:*
  - Split into train and test dataset

```
ds_train = tfds.load(name='rock_paper_scissors', split = 'train')
ds_test = tfds.load(name='rock_paper_scissors', split = 'test')
```

  - Convert images into numpy array
  - Since colors isn't an important factor in this classification problem, we only need 1 color channel

```
train_images = np.array([example['image'].numpy()[:,:,0] for example in ds_train])
train_labels = np.array([example['label'].numpy() for example in ds_train])

test_images = np.array([example['image'].numpy()[:,:,0] for example in ds_test])
test_labels = np.array([example['label'].numpy() for example in ds_test])
```

  - Reshape (add another column (depth) as color channel)

```
train_images = train_images.reshape(2520,300,300,1)
test_images = test_images.reshape(372,300,300,1)
```

  - Convert data to float values in range (0-1)

```
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

train_images /= 255
test_images /= 255
```

Approach:

- Try both the basic approach Fully-connected Network and CNN (Convolutional Neural Network)
- Compare the result and explain why CNN is better to be used for working with images
- Refine the CNN model to get a better result
- For both fully-connected ANN and CNN approach, first a Flatten layer should be applied to convert the image into 2d matrices. And in the output layers, a Fully-Connected (Dense) layer using Softmax classification (to return the classified percentage of each categories (rock, paper, scissors)).
  -
- The loss function for this model is mainly composed of Categorical Cross-Entropy because this is a multi categories classification

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

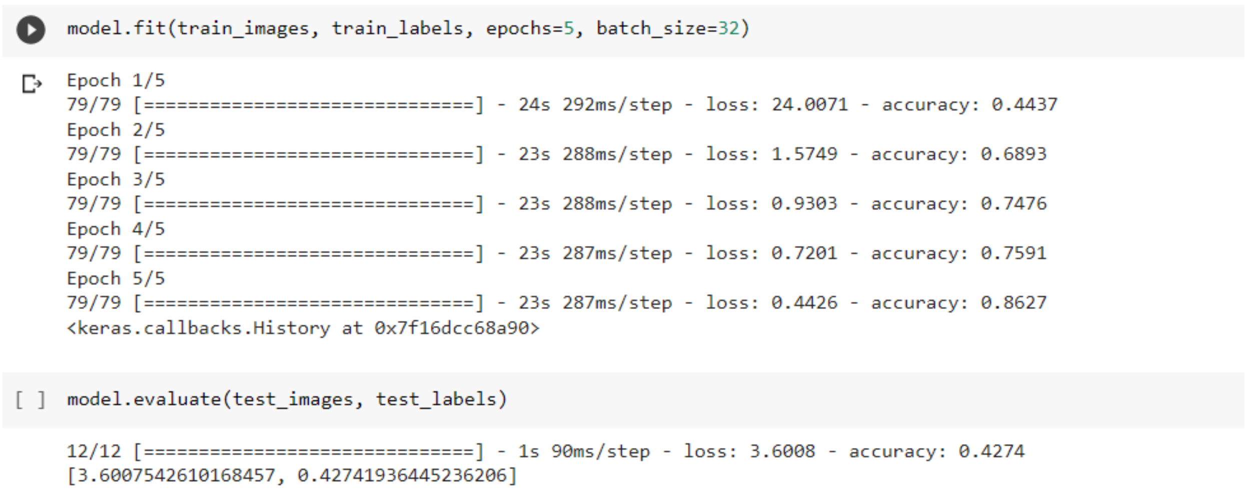
  - In case of binary classification problem (as studied during 391 course), output size is 2. In case of this particular problem (rock, paper, scissors), output size is 3
- Lastly, Gradient Descent with Momentum is used across all layers as the optimizer

Implementation and Refining Model:

- *basic ANN model:*

```
model = keras.Sequential([
  keras.layers.Flatten(),
  keras.layers.Dense(512, activation='relu'),
  keras.layers.Dense(256, activation='relu'),
  keras.layers.Dense(3, activation='softmax')
])

model.compile(
  optimizer='adam',
  loss=keras.losses.SparseCategoricalCrossentropy(),
  metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5, batch_size=32)
```

```
 - Result and Observation:
```



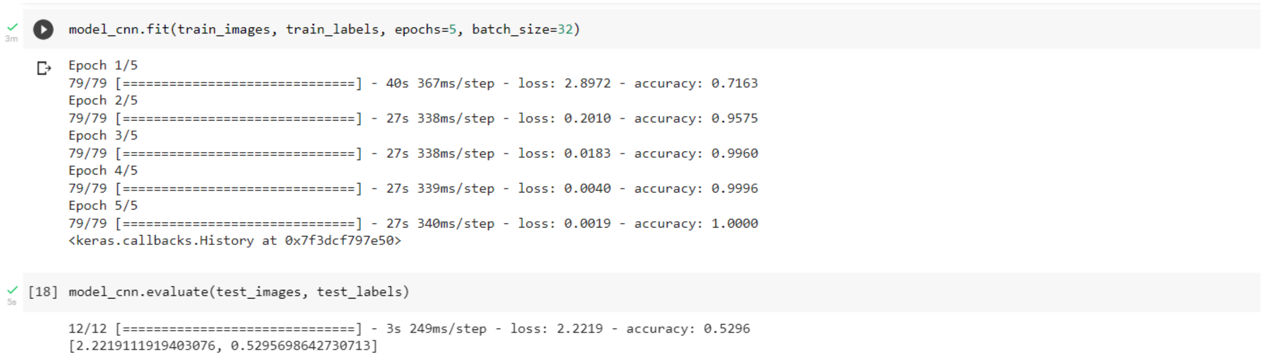
```
model.fit(train_images, train_labels, epochs=5, batch_size=32)

Epoch 1/5
79/79 [==============================] - 24s 292ms/step - loss: 24.0071 - accuracy: 0.4437
Epoch 2/5
79/79 [==============================] - 23s 288ms/step - loss: 1.5749 - accuracy: 0.6893
Epoch 3/5
79/79 [==============================] - 23s 288ms/step - loss: 0.9303 - accuracy: 0.7476
Epoch 4/5
79/79 [==============================] - 23s 287ms/step - loss: 0.7201 - accuracy: 0.7591
Epoch 5/5
79/79 [==============================] - 23s 287ms/step - loss: 0.4426 - accuracy: 0.8627
<keras.callbacks.History at 0x7f16dcc68a90>

[ ] model.evaluate(test_images, test_labels)

12/12 [==============================] - 1s 90ms/step - loss: 3.6008 - accuracy: 0.4274
[3.6007542610168457, 0.42741936445236206]
```

```
     - Even though the accuracy when training the model is great (86%), the actually accuracy of test model is only 42
     - Next, CNN model will be used to test how much improvement can be achieved
```

- *basic CNN model:*

```
model_cnn = keras.Sequential([
  layers.Conv2D(filters=64, kernel_size=3, strides=(1,1), activation='relu', input_shape=(300,300,1)),
  layers.Conv2D(filters=32, kernel_size=3, activation='relu'),
  keras.layers.Flatten(),
  keras.layers.Dense(3, activation='softmax')
])
```

```
 - Result and Observation:
```



```
model_cnn.fit(train_images, train_labels, epochs=5, batch_size=32)

Epoch 1/5
79/79 [==============================] - 40s 367ms/step - loss: 2.8972 - accuracy: 0.7163
Epoch 2/5
79/79 [==============================] - 27s 338ms/step - loss: 0.2010 - accuracy: 0.9575
Epoch 3/5
79/79 [==============================] - 27s 338ms/step - loss: 0.0183 - accuracy: 0.9960
Epoch 4/5
79/79 [==============================] - 27s 339ms/step - loss: 0.0040 - accuracy: 0.9996
Epoch 5/5
79/79 [==============================] - 27s 340ms/step - loss: 0.0019 - accuracy: 1.0000
<keras.callbacks.History at 0x7f3dcf797e50>

[18] model_cnn.evaluate(test_images, test_labels)

12/12 [==============================] - 3s 249ms/step - loss: 2.2219 - accuracy: 0.5296
[2.2219111919403076, 0.5295698642730713]
```

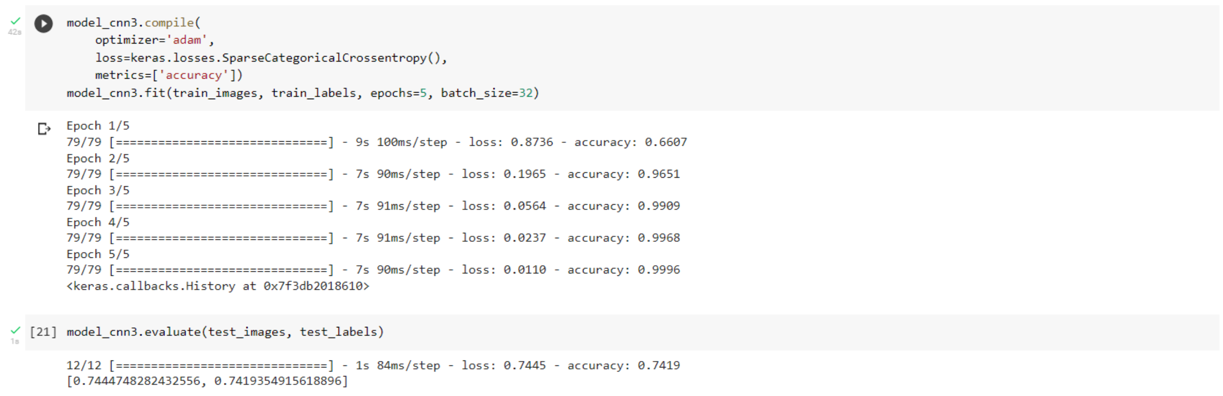
- Accuracy on the test dataset has been increase but still 52%
- This probably due to overfitting problems (as accuracy is perfect with train dataset but mediocre with test dataset)
- Attempt to solve this problem by introducing pooling layers into the model

- *improved CNN model:*

```
from keras.layers.pooling import MaxPool2D
model_cnn3 = keras.Sequential([
   layers.AveragePooling2D(pool_size=4,strides=2, input_shape=(300,300,1)),
   layers.Conv2D(filters=64, kernel_size=3, strides=(1,1), activation='relu'),
   layers.Conv2D(filters=32, kernel_size=3, activation='relu'),
   layers.MaxPool2D(pool_size=2, strides=2),
   keras.layers.Flatten(),
   keras.layers.Dense(3, activation='softmax')
])
```

`

- *Result and Observation:*



```
model_cnn3.compile(
    optimizer='adam',
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])
model_cnn3.fit(train_images, train_labels, epochs=5, batch_size=32)

Epoch 1/5
79/79 [==============================] - 9s 100ms/step - loss: 0.8736 - accuracy: 0.6607
Epoch 2/5
79/79 [==============================] - 7s 90ms/step - loss: 0.1965 - accuracy: 0.9651
Epoch 3/5
79/79 [==============================] - 7s 91ms/step - loss: 0.0564 - accuracy: 0.9909
Epoch 4/5
79/79 [==============================] - 7s 91ms/step - loss: 0.0237 - accuracy: 0.9968
Epoch 5/5
79/79 [==============================] - 7s 90ms/step - loss: 0.0110 - accuracy: 0.9996
<keras.callbacks.History at 0x7f3db2018610>

[21] model_cnn3.evaluate(test_images, test_labels)

12/12 [==============================] - 1s 84ms/step - loss: 0.7445 - accuracy: 0.7419
[0.7444748282432556, 0.7419354915618896]
```

  - As can be seen, the accuracy is now almost 75%
  - In the next part, confusion matrix will be used to further evaluate and understand the model

- *Other methods to improve model:*

  - Using multiple filters and pooling layers alternating between the CNN layers
  - Modern architectures (ResNet, VGGnet, etc) may be applied to achieve better results.

Final Results and Discussion:

- *result & evaluate:*

```
predictions = np.argmax(model_cnn3.predict(test_images), axis=1)
```

- confusion matrix

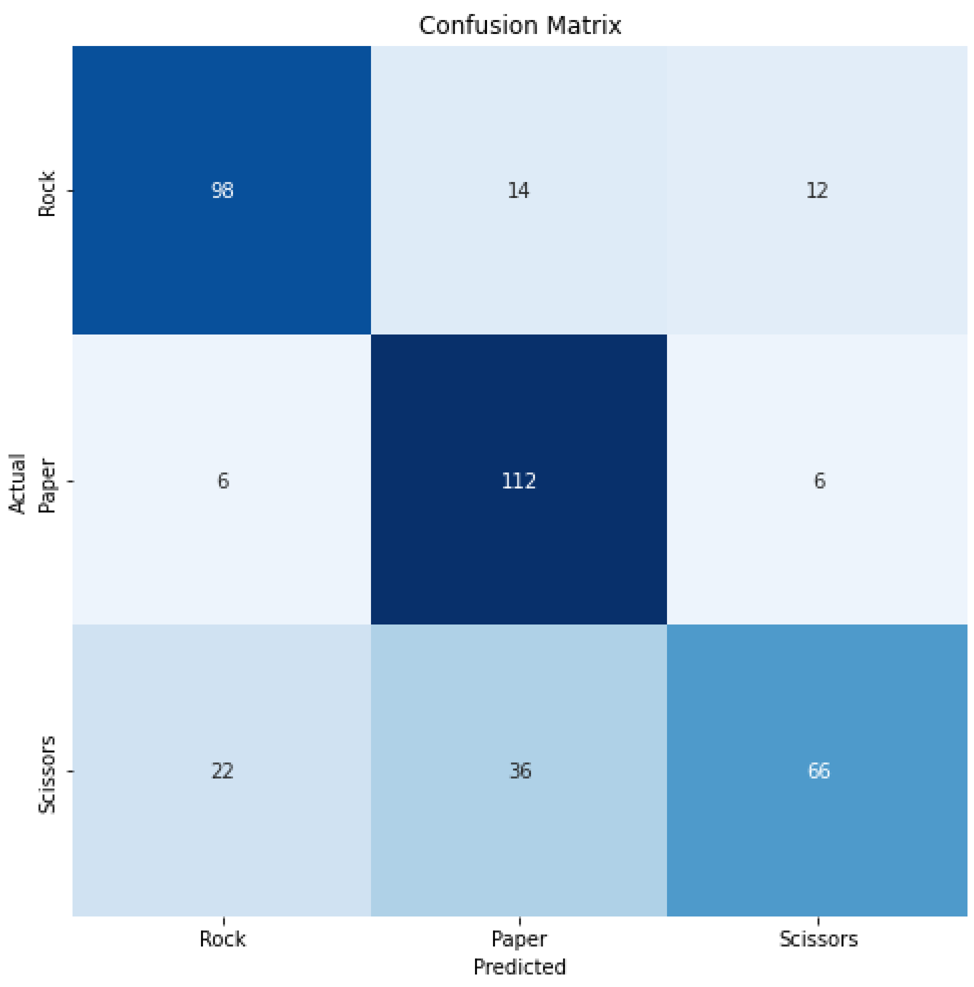
```
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(test_labels, predictions, labels=[0, 1, 2])
clr = classification_report(test_labels, predictions, labels=[0, 1, 2], target_names=["Rock", "Paper", "Scissors"])
```

- *show the result*

```
  import seaborn as sns

  plt.figure(figsize=(8, 8))
  sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
```

```
    plt.xticks(ticks=[0.5, 1.5, 2.5], labels=["Rock", "Paper", "Scissors"])
    plt.yticks(ticks=[0.5, 1.5, 2.5], labels=["Rock", "Paper", "Scissors"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()
    print(clr)
```

- *Evaluate the model:*



```
Classification Report:
----------------------
              precision    recall  f1-score   support

        Rock       0.78      0.79      0.78       124
       Paper       0.69      0.90      0.78       124
    Scissors       0.79      0.53      0.63       124

    accuracy                           0.74       372
   macro avg       0.75      0.74      0.73       372
weighted avg       0.75      0.74      0.73       372
```

  - *Discussion:*
    - Paper is the most accurately predicted, follows by Rock and lastly, scissors

- This observation makes sense because paper and rock are at the 2 extreme case (all fingers stretch, or no finger extends)
- The use of a proper CNN model has tremendously increase the accuracy (upto 78%) compared to the basic ANN model in image classification
- This model is also better than many similar others found in Kaggle (which has accuracy around 70%)
- Further tinkering with the model may achieve even better result.

Reference and Appendix:

1. ANN vs CNN: https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b
2. basic CNN: https://cs231n.github.io/convolutional-networks/
3. Link to google colab: https://colab.research.google.com/drive/1pw2_jrYVWmddWbZDS8xE0COzlrLNb3WR#scrollTo=Gj1fqnEqqpzv
4. Similar projects on Kaggle: https://www.kaggle.com/code/gcdatkin/rock-paper-scissors-image-recognition/notebook