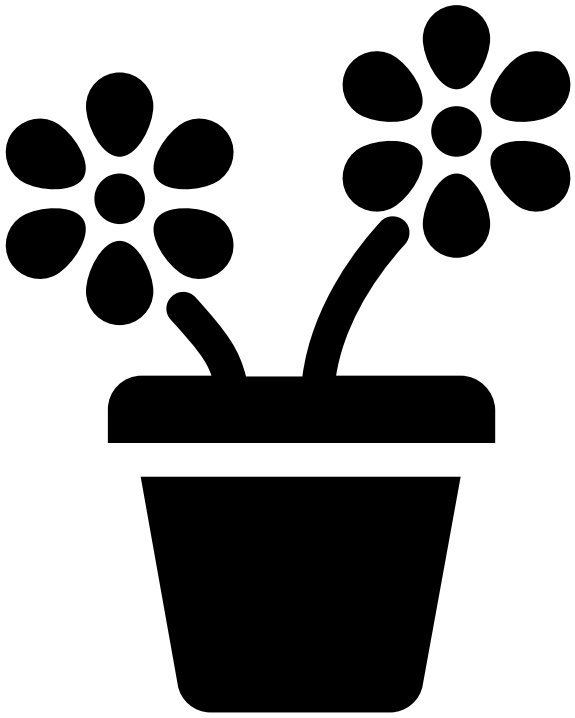
IMAGE RECOGNITION USING ARTIFICIAL INTELLIGENCE & CONVOLUTIONAL NEURAL NETWORKS



Cole M. Helmer

Widener University | 1675690 | 12/02/2024

Abstract:

The objective of this project is to develop an AI-based system within Python. Using a '*computer vision*' model in a sense. Our algorithm would be capable of importing a folder from the computer memory. The folder contains multiple images, for this project it contains almost 4,000 images of flowers. It would then use a certain percentage of the images in order to train itself. This would be done through a CNN, or a Convolution Neural Network. This breaks down the image based on certain attributes to multiple layers and analyzes what it is left with.[1] From these layers, the AI algorithm will in turn, conclude and make an educated guess as to what exactly the image is showing. It will the leftover percentage of images to test itself, to see how accurate it was. A second algorithm will then be created, also using *Python*, to put the CNN to the test. This test algorithm will then be able to output, what flower it believes the image is showing, along with a percent confidence value.

Theory & Principle:

The project tasked students with constructing an AI algorithm. The structure had to be capable of identifying images of flowers with a confidence percentage value. The theory behind the project was mainly focused on a Convolution Neural Network. A Neural Network is a subset of machine learning and are the fundamental basis for most learning algorithms.[2] CNN's are commonly used within AI, finding their home within the realm of deep learning.[2] These CNN's perform incredibly well with image, audio, and speech signal inputs.

A Convolutional Neural Network is normally made up of four layers, the input layer, the convolutional layer, the pooling layer, and the fully connected layer. Layer by layer, the CNN increases in complexity.[3] The input layer being the input signals. Early layers focus on colors on edges while later layers may focus on the overall silhouette of the object. The convolution layer is the main layer within a CNN, as this is where the network associates values with certain attributes in the images. This is then followed by the pooling layer, which condenses the data received from the convolutional layer. The pooling layers simplifies the representation of its computer computation in a sense. These are then followed by the fully connected layer. This occurs when the feature maps created from previous convolution and pooling layers, are flattened, and then passed through the FC layer. This layer, in turn is then responsible for making the final output prediction.

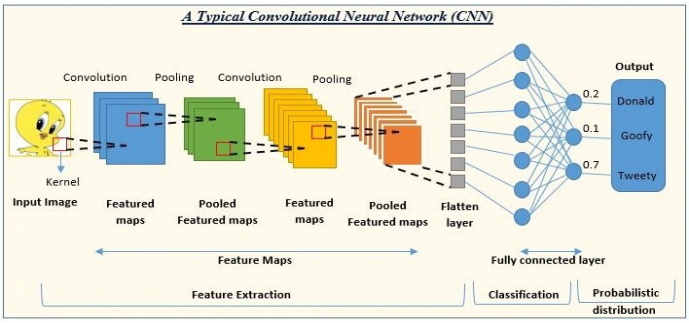


***Figure** 1.* Above, pictures an example of the steps taken by a convolutional neural network. The multiple layers are repeatedly gone through then in a training/testing process, discussed in the next section. [4]

Methods:

As mentioned before, the overall goal was to create an AI algorithm capable of recognizing what it has been shown in an image. The system would refer to a folder stored in the same location as our Python file. This folder contains 3,681 images of specific flowers. The system would then proceed to train itself using eighty percent of the images and the remaining twenty percent shall be used to test itself. These images would be known as our *input layer*.[3] The first part of our algorithm takes all the input images and resizes them to 180x180 pixels, so the image size is kept constant.

The next goal was to then develop and train two Convolutional Neural Networks. The first one, found in *Appendix 1*, consisted of three convolutional layers, and was completed using the ReLU activation function. The number of filters for the three layers went as follows, 16, 32, 64.[3] Along with a batch size of 32 and a total of 15 epochs. The batch size, being how many images are analyzed by the system at once, while an epoch is when the system goes through the entirety of the input layer images and proceeds to cycle through again. As stated

above, eighty percent of images used for training and twenty percent for validation by the system.

The second CNN would consist of four total convolutional layers. With filter values 8, 16, 32, 64. The added layer being the 8-filter level. Everything else consisting with this system, remained constant with the first one.

In order for the system to be able to produce a prediction, a second Python structure needed to be developed. These partner, test files can be found within *Appendix 3 & 4*. These test files would call in the specific CNN model needed at the time, either 'CNN_Model_L3_Ep15' or 'CNN_Model_L4_Ep15'.[3] This would determine which system was being tested. The ten images, stored within the *Test zip file*, were then inputted into the CNN, and the AI algorithm would use the input layer, analyze it within the convolutional and pooling layers, and then feed it through the FC layer, in order to come to final output prediction, along with its percent confidence.

Results:

The structure developed, turned out to be a success. The system was able to be fed ten input images and return a decently accurate response for majority of the images. The accuracy and effectiveness of the different CNN's will be discussed throughout this section. Below, in *Figure 2,* the resulting graphs from the three-layer CNN can be seen.

Figure 2. Resulting graphs from the three-layer CNN training and validation process.

Looking at the training & validation accuracy graph first, it can easily be seen that as the epochs went on, the system showed quite proficient growth, especially within the training aspect, where the validation aspect would sometimes take a dip. Now looking at the training and validation loss of the three-layer model. It can also be seen, as an inverse relationship with the other graph, that as epochs progress, the system would output incorrect predictions less and less. Now moving onto the four-layer CNN, which the resulting graphs can be found below in *Figure 3*.

These graphs, as expected are a little better than that of the three-layer CNN. The

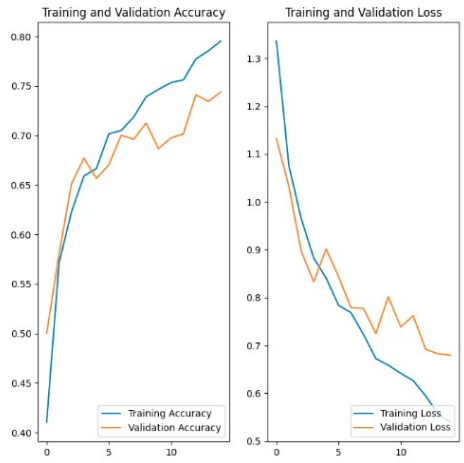


Figure 3. Resulting graphs from the four-layer CNN

T&V accuracy graph is somewhat steeper, with less jagged dips, as is the T&V loss graph lines. The reasoning behind the more jagged movements with the validation tracking, in comparison to the smoothness of the training tracking is because the system has eighty percent of images for training, giving it a larger sample size to garnish a better average, whereas the validation aspect is only twenty percent of the images. These graphs, help to visualize how exactly the CNN model progresses and learns from practice. This process, is discussed more in full in the *Discussion section* of the report. These CNN graphs, also play a crucial role when it comes to the eleven test images.

RE: 601 | Project 3 Test Results

			Three Layer CNN		Four Layer CNN	
Path #	IMG Address	Flower Type	Type Prediction	Confidence %	Type Prediction	Confidence %
0	daisy_1.jpg	DAISY	DAISY	100%	DAISY	100%
1	daisy_2.jpg	DAISY	DAISY	100%	DAISY	99.94%
2	dandelion_1.jpg	DANDELION	DANDELION	56.53%	DANDELION	51.19%
3	dandelion_2.jpg	DANDELION	DAISY	50.28%	DANDELION	43.27%
4	red_sunflower.jpg	SUNFLOWER	SUNFLOWER	99.09%	SUNFLOWER	97.76%
5	rose_1.jpg	ROSE	ROSE	99.62%	ROSE	99.24%
6	rose_2.jpg	ROSE	ROSE	98.08%	ROSE	98.51%
7	sunflower_1.jpg	SUNFLOWER	SUNFLOWER	100%	SUNFLOWER	99.96%
8	sunflower_2.jpg	SUNFLOWER	SUNFLOWER	99.98%	SUNFLOWER	99.02%
9	tulip_1.jpg	TULIP	TULIP	97.80%	TULIP	94.47%
10	tulip_2.jpg	TULIP	TULIP	99.81%	TULIP	98.45%
			10 OUT OF 11	AVG = 91.79%	11 OUT OF 11	AVG = 89.26%

Figure 4. Chart showing individual image test results.

Now, moving into the results of the test structure. For this the system was individually fed eleven input images, in which it would return its prediction along with the percent confidence value. The results for individual image testing can be found above in the charts in *Figure 4*. As seen, for the three-layer CCN, the system scored a 10 out of 11, when looking at whether or not it outputted the correct flower type. It also, associated on average for the eleven trials a confidence percentage of 91.79%. As for the four-layer CNN, it scored a 100% accuracy rating on the test, going 11 out of 11 on predictions for flower type. The average output confidence percentage however was an 89.26%.

Discussion:

Overall, the AI algorithm system was capable and succeeded in its main purpose. I was able to develop and train two separate CNN's, one a three-layer and one a four-layer. I was able to then display four total output graphs, two for each CNN. One showing its T&V accuracy, while the other showed the T&V loss. Quite easily seen, it is noticed that these two have an inverse relationship as epochs progressed. It was quite a steep progression as well. This is understandable, as this basically means that as the system saw more images and had more practice testing with the images, it in turn became better at predicting what was stored in these images.

Another aspect tested in this project was the effect of convolutional layers in an AI system like this. As we saw, when going from a three-layer to a four-layer, our classification went from a 10/11 to a 11/11, however our percentage of confidence dropped by almost three percent. This can be visualized as the system learns more about the images and obtains more variables/attributes/filters it begins to almost-second guess itself more often. A term often used in test taking when somebody believes they know the answer immediately, but then later on begin to contemplate whether or not that is the correct answer after all.

Conclusion:

Project three proved to be a great representation of an AI algorithm tasked with image recognition. The developed structure seen below, was required to accept almost four-thousand images in order to train itself on flower images. Eighty percent used for training; twenty percent used for validating its training. We looked at two graphs showing the inverse correlation between the T&V accuracy and loss for both CNN's. We then tested the CNN's to see how our algorithm would perform when given a single image at a time, with both the three-layer and four-layer CNN's performing at high levels. As mentioned in the *Results* section, the three-layer CNN went 10 out of 11 with an average confidence percentage value of 91.79%, while the four-layer went 11 out of 11 and had an average confidence percentage value of 89.26, a slight a decrease from the three-layer.

I had a few questions arise during the project as I worked. Such as what would happen if we entered in more complicated images? Would simply adding more convolution layers improve the ability of your algorithm with those images? In theory, that should be the case. Would just increasing the epochs a good amount be enough? These are all questions I would like to further dive into after taking on this project.

References

[1] GeeksforGeeks. (2024, March 14). *Introduction to Convolution Neural Network*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-convolution-neural-network/

[2] IBM. (2024). *What are Convolutional Neural Networks? | IBM*. Www.ibm.com; IBM. https://www.ibm.com/topics/convolutional-neural-networks

[3] Song, Dr. (2024, November). *RE: 601 | Project 3*. Canvas; Widener University.

[4] (2024). Analyticsvidhya.com. https://cdn.analyticsvidhya.com/wp-

content/uploads/2024/10/59954intro-to- CNN.web

Appendix 1:

CNN_Model_L3_Ep15

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

batch_size=32
img_height=180
img_width=180
data_dir="flower_photos"

train_ds=tf.keras.utils.image_dataset_from_directory(data_dir,
                                    validation_split=0.2,
                                    subset="training",
                                    seed=123,
                                    image_size=(img_height,img_width),
                                    batch_size=batch_size)
val_ds=tf.keras.utils.image_dataset_from_directory(data_dir,
                                  validation_split=0.2,
                                  subset="validation",
                                  seed=123,
                                  image_size=(img_height, img_width),
                                  batch_size=batch_size)

class_names=train_ds.class_names
print(class_names)

AUTOTUNE=tf.data.AUTOTUNE
train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds=val_ds.cache().prefetch(buffer_size=AUTOTUNE)

normalization_layer=layers.Rescaling(1./255)

normalized_ds=train_ds.map(lambda x, y: (normalization_layer(x),y))
image_batch, labels_batch=next(iter(normalized_ds))
first_image=image_batch[0]

num_classes=len(class_names)
data_augmentation=keras.Sequential([layers.RandomFlip("horizontal",input_shape=(img_height,img_wi
dth,3)),
                    layers.RandomRotation(0.1),
                    layers.RandomZoom(0.1)])
```

```
model=Sequential([data_augmentation,layers.Rescaling(1./255),
                layers.Conv2D(16,3,padding='same',activation='relu'),
                layers.MaxPooling2D(),
                layers.Conv2D(32,3,padding='same',activation='relu'),
                layers.MaxPooling2D(),
                layers.Conv2D(64,3,padding='same',activation='relu'),
                layers.MaxPooling2D(),
                layers.Dropout(0.2),
                layers.Flatten(),
                layers.Dense(128,activation='relu'),
                layers.Dense(num_classes,name="outputs")])

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

model.summary()
epochs=15;
history=model.fit(train_ds,validation_data=val_ds,epochs=epochs)

acc=history.history['accuracy']
val_acc=history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range=range(epochs)

model.save("cnn_model_L3_ep15.keras")
plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(epochs_range,acc,label='Training Accuracy')
plt.plot(epochs_range,val_acc,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1,2,2)
plt.plot(epochs_range,loss,label='Training Loss')
plt.plot(epochs_range,val_loss,label='Validation Loss')
plt.legend(loc='lower right')
plt.title('Training and Validation Loss')
plt.show()
```

Appendix 2:

CNN_Model_L4_Ep15

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

batch_size=32
img_height=180
img_width=180
data_dir="flower_photos"

train_ds=tf.keras.utils.image_dataset_from_directory(data_dir,
                                        validation_split=0.2,
                                        subset="training",
                                        seed=123,
                                        image_size=(img_height,img_width),
                                        batch_size=batch_size)
val_ds=tf.keras.utils.image_dataset_from_directory(data_dir,
                                      validation_split=0.2,
                                      subset="validation",
                                      seed=123,
                                      image_size=(img_height, img_width),
                                      batch_size=batch_size)

class_names=train_ds.class_names
print(class_names)

AUTOTUNE=tf.data.AUTOTUNE
train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds=val_ds.cache().prefetch(buffer_size=AUTOTUNE)

normalization_layer=layers.Rescaling(1./255)

normalized_ds=train_ds.map(lambda x, y: (normalization_layer(x),y))
image_batch, labels_batch=next(iter(normalized_ds))
first_image=image_batch[0]

num_classes=len(class_names)
data_augmentation=keras.Sequential([layers.RandomFlip("horizontal",input_shape=(img_height,img_wi
dth,3)),
                    layers.RandomRotation(0.1),
                    layers.RandomZoom(0.1)])

model=Sequential([data_augmentation,layers.Rescaling(1./255),
          layers.Conv2D(8,3,padding='same',activation='relu'),
```

```
            layers.MaxPooling2D(),
            layers.Conv2D(16,3,padding='same',activation='relu'),
            layers.MaxPooling2D(),
            layers.Conv2D(32,3,padding='same',activation='relu'),
            layers.MaxPooling2D(),
            layers.Conv2D(64,3,padding='same',activation='relu'),
            layers.MaxPooling2D(),
            layers.Dropout(0.2),
            layers.Flatten(),
            layers.Dense(128,activation='relu'),
            layers.Dense(num_classes,name="outputs")])

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

model.summary()
epochs=15;
history=model.fit(train_ds,validation_data=val_ds,epochs=epochs)

acc=history.history['accuracy']
val_acc=history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range=range(epochs)

model.save("cnn_model_L4_ep15.keras")
plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(epochs_range,acc,label='Training Accuracy')
plt.plot(epochs_range,val_acc,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1,2,2)
plt.plot(epochs_range,loss,label='Training Loss')
plt.plot(epochs_range,val_loss,label='Validation Loss')
plt.legend(loc='lower right')
plt.title('Training and Validation Loss')
plt.show()
```

Appendix 3:

Test Structure | cnn_model_L3_ep15
(individualy entered image addresses and recorded the outputs on side paper for the chart)

```
import numpy as np
import tensorflow as tf
from tensorflow import keras

img_height=180
img_width=180

class_names=['daisy','dandelion','roses','sunflowers','tulips']
tes_path="C:/test/Red_sunflower.jpg"

img=tf.keras.utils.load_img(test_path,target_size=(img_height,img_width))
img_array=tf.keras.utils.img_to_array(img)
img_array=tf.expand_dims(img_array,0)

test_model=keras.models.load_model("cnn_model_L3_ep15")
predictions=test_model.predict(img_array)
score=tf.nn.softmax(predictions[0])

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

Appendix 4:

Test Structure | cnn_model_L3_ep15
(individualy entered image addresses and recorded the outputs on side paper for the chart)

```
import numpy asnp
import tensorflow as tf
from tensorflow import keras

img_height=180
img_width=180

class_names=['daisy','dandelion','roses','sunflowers','tulips']
tes_path="C:/test/Red_sunflower.jpg"

img=tf.keras.utils.load_img(test_path,target_size=(img_height,img_width))
img_array=tf.keras.utils.img_to_array(img)
img_array=tf.expand_dims(img_array,0)

test_model=keras.models.load_model("cnn_model_L4_ep15")
predictions=test_model.predict(img_array)
score=tf.nn.softmax(predictions[0])

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