Henry Puma

Professor Erik Grimmelmann

CSC 44700 : Introduction to Machine Learning Final Paper

December 17, 2020

Final Paper: Convolutional Neural Networks

Introduction

Since the inception of computational devices, humanity has accomplished so much to what would be considered unfathomable to the generations before them. With each generation, computing has evolved in a way that slowly embeds itself into our daily lives while improving the day to day operations for everyone. Daily necessities such as food, transportation, and communication have become a major contributor to the surge of raw data being generated. The best candidate to process this supply of information would be none other than the modern computer; a machine where even in its infancy could process large amounts of data in a fraction amount of time it would take dozens of people to process. Every few years, the boundaries of computation expands itself, opening a new realm of possibilities. After a few decades of hovering around, machine learning has recently become increasingly popular these past few years. **Image 1**

This semester I've had the opportunity to take a machine learning course. Given that the concept of machine learning has been around for a few decades. It is only these past few years that machine learning has become widespread and accessible to ordinary people. The ability of possessing a machine capable of large amounts of computation has become much easier. When deciding on taking this course, I came to realize that the concept of machine learning is

constantly evolving but by taking this course I have been able to establish a foundation that will aid me in understanding the ongoing development of machine learning.

Since our birth, our developing mind has been capable of capturing images and interpreting them without us consciously aware about it. The millions of neurons that symphonize together to create what we know as vision go unnoticed by the majority. For a long period of time a machine's capability of processing real time video has greatly sparked my interest. However, the extensive understanding requirement creates a temporary barrier due to my limited knowledge. I figured, I'd start with something more primitive such as 2D images and with that move my way up. And what better way to tackle this than through the use of modern day computing. But if you were to get a machine to replicate our exact method of vision, it would be another story. The machine's ability to replicate the same thing is not hindered by the technological requirement but instead its ability to match one of the most complex entities known to man; the brain.

What are Convolutional Neural Networks? (CNN)

For my final paper, I have decided to tackle the concept of Convolutional neural networks. It is one of the closest approaches I've encountered that closely resemble our method of visual processing. Additionally, I consider it the best approach of using "what we have" with modern computing. **Image 2** depicts the layer structure of a typical Convolutional Neural Network (CNN).

To get an understanding of the methodology used in this experiment, I will need to explain the components that are usually found within a typical Convolutional Neural Network (CNN). The following components are what best describe the overall structure of my project:

Convolutional Layers: This is the bread and butter of the entire neural network. This layer is in charge of detecting recognizable features of an image, this is possible through the application of filters. Each filter is best at recognizing specific features ("patterns") such as edges, corners, circles, squares, and many geometric properties. These layers can be organized in many variations. Again it is up to the programmer to design and test which configuration best fits the scenario.

Pooling Layers: Addresses the need for downsampling. It is in charge of reducing the size of activation maps while also preserving the same detected features. This is possible due to the pooling operation that is applied to the feature map. The size of the pooling operation could be something such as a 2 x 2 matrix with a stride of 2 pixels, the stride describes how much of disticance the pooling operation shifts as it convolutes'. There are many types of pooling such as Max, Average, and L2-norm. Each operation is applied to the pooling operation where each type of pooling has their advantages/disadvantages, this feature would be up to the data scientist which takes into account the current situation. **Image 3**

Feature Map: The extracted values from the filters that have been applied to the image. This means that the feature depicts the most prominent features.

Fully Connected Layers: These layers usually occur at the end of the network. It takes in the previous layer and outputs an N dimensional vector where N is the number of all classes**.** For instance, if there are 5 classes in total then N=5 and a vector outputted from the connected

layer may look something like [0 , 0.3, 0.82, 0.5, 0.11], each index represents the probability an image could be [index] class so there is:

0% for class 1, 30% for class 2, 82% for class 3 etc … **Image 2 (Second to last section of image)**

These listed components are the ones usually found in a Convolutional Networks. I cannot stress it enough that these components ("layers") are customizable by the programmer. The customizability of our machine learning model allows us to make a large variety of tweaks that would produce the overall "best" result. With a brief overview of what a CNN consists of, I was able to move on with tackling the main computational ML task which in turn compliments the recently reviewed knowledge acquired from this paper.

Computational Task: Constructing a simple CNN in python

In my machine learning course, I have gathered enough information to "get the ball rolling" and have ventured out and pursued completing a small project which would help me get a firsthand understanding of how CNNs works.

For my project I decided to use the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 color images with 10 classes, where there are 6000 images per class. There are 50000 training images and 10000 test images **Image 4**. This is a fairly large dataset and was a bit lengthy when it came time to train a model. Fortunately, my system has been able to take advantage of my GPU which is the NVIDIA GTX 1080, which drastically reduced the amount of time it took to train a model. The main reason why I chose this dataset was because it is really easy for one to obtain the "prepared" dataset from the keras.dataset library. This essentially removes the need for one to figure out how to appropriately prepare the data for training. The

reason why I bring this up is because an ill prepared dataset, even minor faults could cause unexpected/inaccurate results to the true data's representation when it's being modeled.

After looking through CNN models with the same dataset in the keras library, I realized that most of the model setups were more complex than anticipated. But it forced me to familiarize myself with the structure of a CNN model setup using Keras. This in turn helped me set up my own barebones CNN. **Image 5.** I set up my CNN with 2 convolution layers with a (2 x 2) max pooling layer after the convolutions layers. I learned that the deeper you go in a convolution layer, the more filters you need to add. The reason being is because as you go deeper, detecting more complex patterns require additional filters. I've decided to increase the amount of filters in each convolutional layer by 2 fold. I haven't added additional convolutional layers because the images set doesn't have large enough images to hold very complex patterns because each image is 32 x 32 Pixels. Finally I added a dense layer that implements the ReLU activation function. The ReLU activation function outputs only positive values and zeros anything negative. I've decided to go with this activation function because it has been talked a lot about within the course and also because I've learned that using the ReLU function as a default tends to provide better performance. Finally, I've added the corresponding Dense layers that outputs the vector of size N=(10 classes) which are normalized by the last dense layer through the softmax function. With this, I have described my CNN model setup. I know there is room for improvement and I look forward to any feedback.

When compiling the model, **(Image 6)** I used the Adam algorithm optimizer, and this is because it has been covered in class and works best most of the time with models. But, for the loss function I decided to go with the "sparse_categorical_crossentropy". The sparse takes into

consideration that of the 0-9 classes since there are 10 classes in total. Categorical, because there is obviously more than one class (non-binary).

I fit the model with the imported training data which undergoes 10 epochs **Image 7**. The lowest accuracy was 0.4533 and highest 0.7624. Getting this improved accuracy was a sign of relief because it is far better than the .30 accuracy I had with my previous model during the presentation. The moment of truth came when I finally evaluated the model with the test data **Image 8** and endup up with a **69% accuracy!**!. But in order to look further into the data, I have taken the liberty to create auxiliary functions that take a random sample and check it's predictions for each class. In **Image 9** You will see how well my model predicts animals, but when a random non-animate object is generated, a large portion of the accuracies aren't really that accurate because most of them were less than 50%. With my final submission, I have also included my .ipynb file with the necessary documentation. I feel that the code itself would tell the story in greater detail. You can also call the test_value() function to see more random samples and their accuracies along with their percent predictions for other classes.

Conclusion

It amazes me that something part of nature (vision) became applicable for something inanimate to possess. I always keep in mind that unfathomable today could be something possible way beyond our time, and I am more than satisfied to have been a tiny part of that journey through this paper.Although I feel like I haven't expanded enough on CNNs extensively as I would have liked. I have made the effort to invest in a machine learning related book published by O'reilly "ML with Sci-kit learn, tensorflow and keras". I feel like this book will serve as a great reinforcement to what I've learned throughout the course.

Appendix

Image 1

The internet of things: A brief illustration of how modern technology is embedded in our everyday lives

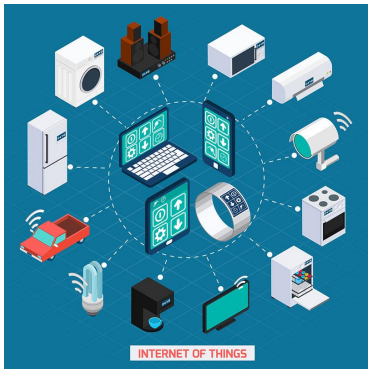


Image 2

An illustration depicting the layers within a typical Convolutional Neural Network

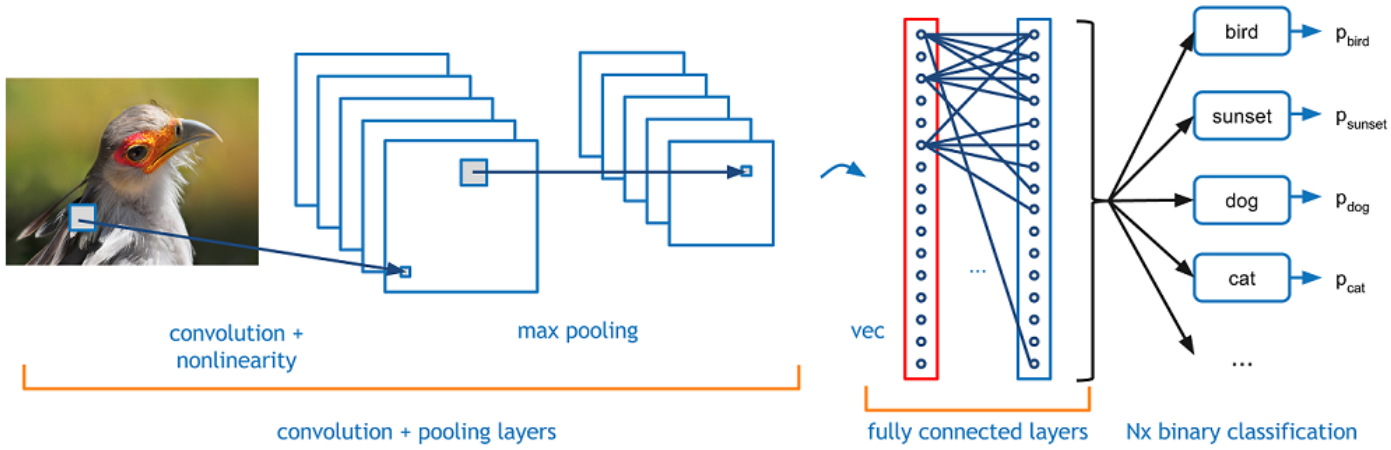


Image 3

Illustration that depicts the pooling operation and examples of max and average pooling

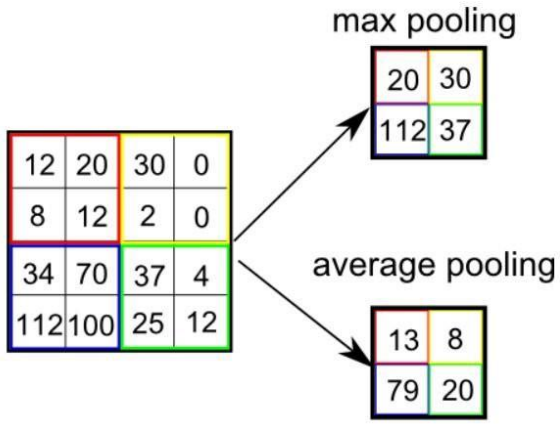


Image 4

CIFAR-10 dataset class illustration with samples of each class

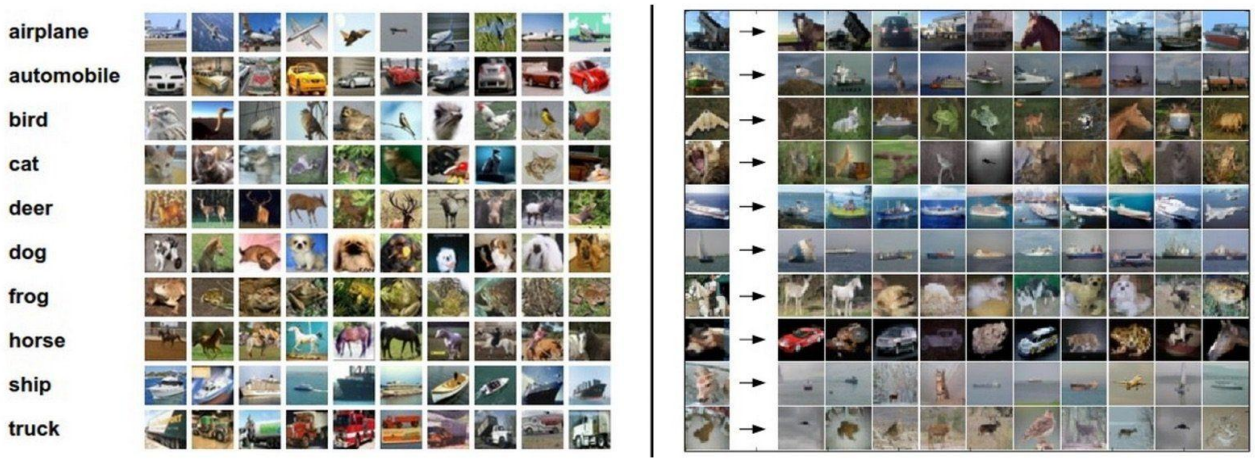


Image 5

CNN setup using keras

```
In [8]: # Convolution Neural network implementation, much more concise and more raw that will have
        # more detail adjustments in paper, start simple then move forward
        cnn_model = Sequential()
        cnn_model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)))
        cnn_model.add(MaxPooling2D((2,2)))

        cnn_model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
        cnn_model.add(MaxPooling2D((2,2)))

        cnn_model.add(Flatten())
        cnn_model.add(Dense(64,activation='relu'))
        cnn_model.add(Dense(num_classes, activation='softmax'))
```

Image 6

Setup when compiling model

```
In [9]: cnn_model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=['accuracy'])
```

Image 7

Progress of fitting the model

```
In [10]: # Fitting and saving to callbacks_list for tensoboard
         callbacks_list = []
         cnn_model.fit(train_X, train_Y, callbacks=callbacks_list, epochs=10)

Epoch 1/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.5095 - accuracy: 0.4533
Epoch 2/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.1456 - accuracy: 0.5966
Epoch 3/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0141 - accuracy: 0.6467
Epoch 4/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.9343 - accuracy: 0.6746
Epoch 5/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.8776 - accuracy: 0.6956
Epoch 6/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.8307 - accuracy: 0.7129
Epoch 7/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7861 - accuracy: 0.7275
Epoch 8/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7482 - accuracy: 0.7406
Epoch 9/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7199 - accuracy: 0.7507
Epoch 10/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6846 - accuracy: 0.7624
```

Image 8

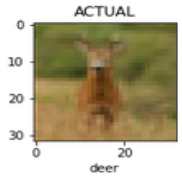
Actually testing model and displaying model summary

```
In [11]: cnn_model.evaluate(test_X, test_Y)

         313/313 [==============================] - 1s 2ms/step - loss: 0.9107 - accuracy: 0.6961

Out[11]: [0.91070157289505, 0.6960999965667725]

In [12]: cnn_model.summary()

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         conv2d (Conv2D)              (None, 30, 30, 32)        896
         _________________________________________________________________
         max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
         _________________________________________________________________
         conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
         _________________________________________________________________
         max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
         _________________________________________________________________
         flatten (Flatten)            (None, 2304)              0
         _________________________________________________________________
         dense (Dense)                (None, 64)                147520
         _________________________________________________________________
         dense_1 (Dense)              (None, 10)                650
         =================================================================
         Total params: 167,562
         Trainable params: 167,562
         Non-trainable params: 0
```

Image 9

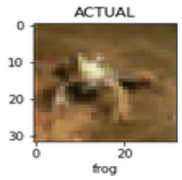
Outputs when picking a random index to see if our model is accurate when a random sample is picked

```
MODEL_PRED:       deer            0.8108045459

OTHER_PRED:       airplane        0.0005033406
OTHER_PRED:       automobile      0.0000035547
OTHER_PRED:       bird            0.1151704714
OTHER_PRED:       cat             0.0221642386
OTHER_PRED:       dog             0.0214709602
OTHER_PRED:       frog            0.0009594164
OTHER_PRED:       horse           0.0288763903
OTHER_PRED:       ship            0.0000164434
OTHER_PRED:       truck           0.0000306290
```



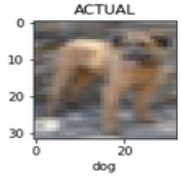

```
In [277]: test_value()
```

```
MODEL_PRED:       frog            0.4181624055

OTHER_PRED:       airplane        0.0011491725
OTHER_PRED:       automobile      0.0000196193
OTHER_PRED:       bird            0.1439086944
OTHER_PRED:       cat             0.1241742522
OTHER_PRED:       deer            0.1999398619
OTHER_PRED:       dog             0.0989187360
OTHER_PRED:       horse           0.0134862978
OTHER_PRED:       ship            0.0000160099
OTHER_PRED:       truck           0.0002250000
```




```
In [278]: test_value()
```

```
MODEL_PRED:       dog             0.7336249948

OTHER_PRED:       airplane        0.0000162709
OTHER_PRED:       automobile      0.0000034940
OTHER_PRED:       bird            0.1698070765
OTHER_PRED:       cat             0.0520992391
OTHER_PRED:       deer            0.0028986193
OTHER_PRED:       frog            0.0050806515
OTHER_PRED:       horse           0.0361725837
OTHER_PRED:       ship            0.0000118139
OTHER_PRED:       truck           0.0002852328
```



Resources

These links are a large portion of the sources I've used to research deep into the topic and create my model.

https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network

https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

https://towardsdatascience.com/visualising-filters-and-feature-maps-for-deep-learning-d814e13bd671#:~:text=The%20feature%20maps%20of%20a,what%20features%20our%20CNN%20detects.

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/