Deep Learning Quantitative Analysis

Sivaram Jawahar

Convolutional Neural Networks Case Study

Computer Vision: Classifying Dog and Cat Images

---

1 Methodology

Convolutional Neural Networks (CNN)

When looking at picture our brain is trying to identify features than through this features observe the image. Then depending on these features our brain classifies peaces of the image in a certain way. The same picture can be interpreted differently depending on the features that our brain identifies and processes. Let us look at the following picture which has two main interpretations. On one hand we can see a young lady with dark hair and a feather in it. She is wearing a black dress and is looking to the other way. On the other hand, in the same image one can see an old women wearing scarf on her had who is looking down. So, thee same image can be classified to two different categories depending on the features that are identified by your brain.

Figure 1: The young lady and old women

The idea behind CNN to mimic the process of evaluating and interpreting images by the brain such that computers can interpret images in a the same way as human brain does. During the last few years CNN became so popular that it is even taking over the ANN (Artificial Neural Network) and is being largely used by different industries. The self-driving cars are one of its applications where the CNN is used to identify people on the read, other cars, animals passing the road etc. Another application of CNN is the face-recognition tool in modern smartphones where the phone is being unlocked using the face instead of a password. CNN consists of three layers: **Input Image**, **Feature Extraction**, **Image Class** (Output Label). Figure 2 visualizes the process behind the CNN. The idea behind this process is that each picture has a digital representation which is used to extract features from it.

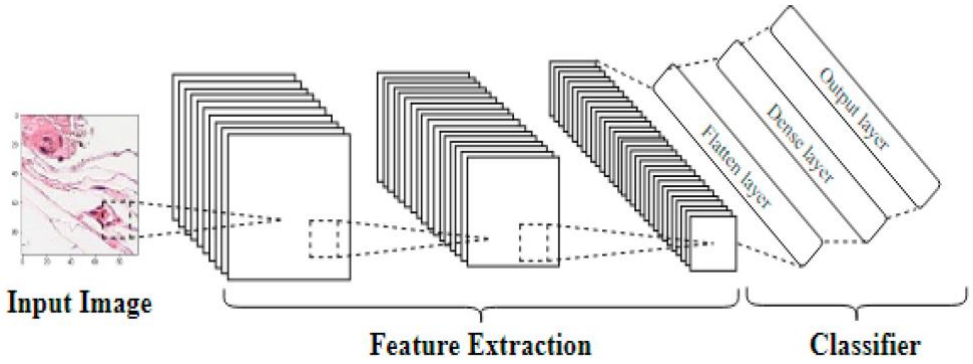


Figure 2: CNN layers

1.1 Convolutional Operation

The main idea behind convolution is to detect features in the image and put them into a feature map while keeping the pixel information (1). Feature detection helps to make image size smaller, which makes the processing faster and depending on the stride used in the detection process the amount of size reduction can differ. For instance, stride of 2 will lead to much larger reduction in the size than stride of 1. However, using larger strides might also lead to information loss. The idea is to detect and focus the important features while ignoring the unimportant part of the information. Feature detection is based on matrix multiplication. The result of this process is to create feature maps.
Different detected filters lead to different feature maps of the same image.

ReLU

The purpose behind applying the Rectifier function is to increase thee non-linearity in the CNN (2). This is being done to mimic the property of the images that are usually highly non-linear, with various elements within the image.

1.2 Pooling

The same object can appear in different positions in different pictures, the question is how the CNN can identify the same feature in different shapes, forms and positions. Namely, the neural network needs to be flexible enough to detect thee object independent of the position or shape. Pooling is responsible for this flexibility property. The pooling can be of different types: Max Pooling, Min Pooling, Mean Pooling, Sum Pooling etc. For instance, in case of the Max Pooling, from the convoluted layer, the feature map, the maximum

values per stride are selected to create a Pooled Feature Map (3). By doing this, the unimportant part of the features are filtered out meanwhile taking into account all types of possible distortions. After this step the size of the picture is reduced once again. One of the biggest benefits of pooling is the reduction of number of features which helps to avoid model overfitting.

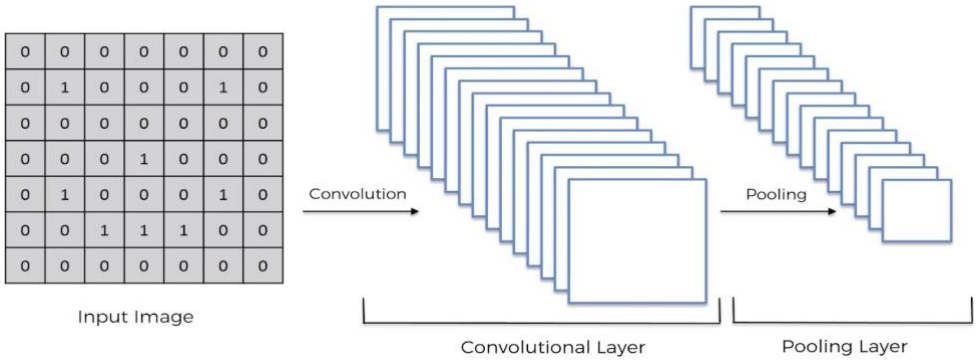


Figure 3: Pooling Process in CNN

1.3 Flattening

The flattening step is one of the most convenient in the CNN process. What flattening does is that it takes the Pooled Feature Map from the Pooling step and transforms it to a single vector with 1 column. The motivation behind this is to prepare this data as an a filed that can be used as an input for an Artificial Neural Network (ANN).

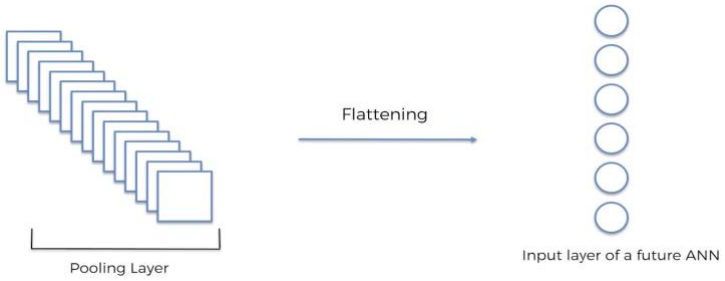


Figure 4: Flattening Process

1.4 Full Connection

This is a final of the entire CNN process where all pieces of the process come together. The Features have been detected and convoluted into a feature map which has been pooled and flattened and it is at this stage where this output from the Flattening process is being used as an input for the ANN. The goal of using ANN is to enrich the existing flattened feature representation of an image with additional attributes. These steps help us to get even better prediction of the picture. The right part of figure 5 is a simple representation of this final step where all peaches are being fully connected to get an output prediction.
Let us look at this final part of the CNN model, the training of ANN model in more detail. In the first step, the input layer features that are based on the pooled and flattened feature maps are used to transfer signal to the hidden layer. In the next step the final neurons learn about the specific hidden neurons that they need to focus on when classifying the image to a certain class in the output layer. In this final step the features are propagated through the network and conveyed to the output. Based on this the importance weights are assigned to the neurons.

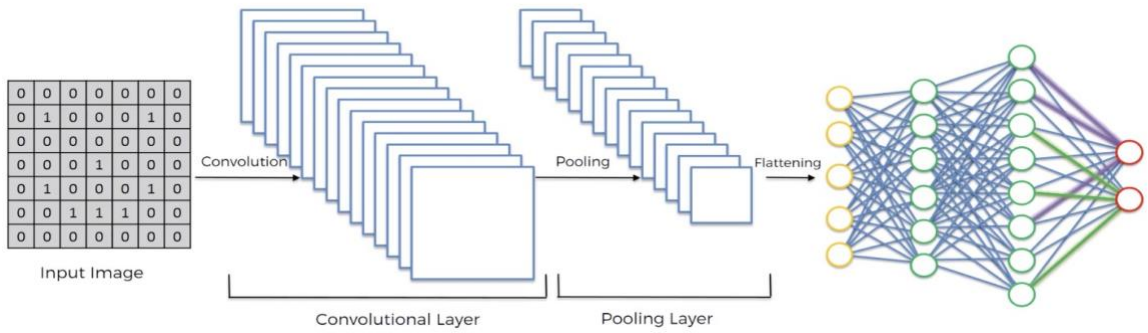


Figure 5: CNN Architecture

1.5 Loss Function: Softmax and Cross-Entropy

The goal behind the CNN process is to classify the image to a certain class. So, the idea is to get probabilities of the image belonging to each of the possible classes, where the sum of all these probabilities should be 1. For this purpose the *Softmax* function expressed by the following equation is used to achieve this. This Softmax function, also called Normalized Exponential function takes the k-dimensional output vector, where k refers to the number of classes that image can belong to, and transform it to a vector consisting of values in the range of [0,1] such and it sums up to 1.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

In order to perform optimization the CNN needs to use a value that it can optimize in each epoch to determine to upgrade the weights in ANN accordingly. To do this, CNN is using a *Cross-Entropy* function as a Loss Function which is similar to the accuracy measure *Mean Squared Error*, that is being minimized to optimize the training model and make accurate predictions. The Cross-Entropy Loss function can be represented by the following expression:

$$L_i = -log(\frac{e^{f_{y_j}}}{\sum_j e^{f_j}})$$

$$H(p,q) = -\sum_x p(x) * log(q(x))$$

During the training of CNN the goal is to minimize this loss function to improve the prediction accuracy of the model. Where the p is the vector containing the final labels for the classes with values wither 0 and 1, and the q is the probability vector that we get from the Softmax function. The reason behind using Cross Entropy, instead of Classification Error and Mean Squared Error, is because in the CNN it is likely to detect even small errors more accurately than the other evaluation functions which is thanks to the logarithmic part of this function. Consequently,identifying as much as possible errors helps ti train the neural network better and leads to an increased accuracy of the model. Overall, training CNN takes longer than training ANN simply because CNN includes all the previously mentioned steps before applying the ANN, where the final image classification process takes place. Figure 5 summarizes the entire CNN process. What is important in this process is that not only the weights but also the feature detection are being repeated and updated in each iteration or epoch unlike the usual ANN where the only changing factor is the weights.

1.5.1 Loss Function Optimizers

Our goal during the CNN training process is to minimize the amount of error we are making, the difference between the predicted value and the real value of the output variable (y). So, in order to learn about these different signals is to predict the output value of y, (y_{hat}) then compute the cost associated with this iteration or *epoch*, determine which set of weights at this stage will minimize the cost function, and use this to update the weights. This process is repeated as much that the cost function is minimized as much as possible. To determine which weights minimizes the cost function, different statistical algorithms can be used such as Stochastic Gradient Decent (SGD) or Adam Optimizer.

Stochastic Gradient Decent

SGD method, also known as Incremental Gradient Descent, is an iterative approach for solving optimization problems with a differential objective function (7). The SGD method is often referred as the stochastic approximation of the gradient descent which aims to find the extreme or zero points of the stochastic model containing parameters that cannot be directly estimated. Figure below demonstrates an example of an optimization problem where GD and SGD are used. From the two black patters in the graph, the straight pattern in the lower part corresponds to a standard Gradient Descent method where the parameters updates are done using all training points, whereas the black non-linear pattern in the upper side with a lot of fluctuations, corresponds to the SGD method where for the parameter updates only one training sample is used. With SGD reaching the global maximum is likely to happen faster, because one starts making updates and improving the direction of the pattern using a single training sample. Due to so many updates (fluctuations in the pattern in the direction of reaching global optimum point) SGD is likely to converge faster compared to the GD. Both SGD and GD suffer from the problem of reaching a local optimum instead of the global optimum. However, in case of SGD the likelihood of reaching a local optimum instead of global optimum is higher compared to the GD, because of its constant changes in moving direction. From Figure 6 we observe that it is likely that the fluctuating black pattern, corresponding to the SGD, will end up in L instead of G while changing his moving direction so often. Whereas, it is less likely that the straight pattern, corresponding to the GD, will end up in L, while both patterns have the same starting point.

Adam Optimizer

A popular technique for enhancing SGD optimization procedure is the Adaptive Moment Estimation (Adam) introduced by (5). Adam is the extended version of the SGD with momentum method. The main difference of it compared to the SGD with momentum, which uses a single learning rate for all parameter updates, Adam algorithm defines different learning rates for different parameters. The algorithm calculates the individual adaptive learning rates for each parameter based on the estimates of the first two moments of the gradients. So, each parameter has a unique learning rate, which is being updated using the exponential decaying average of the first moments (the mean) and second moments (the variance) of the gradients. We will use Adam Optimizer in this analysis as

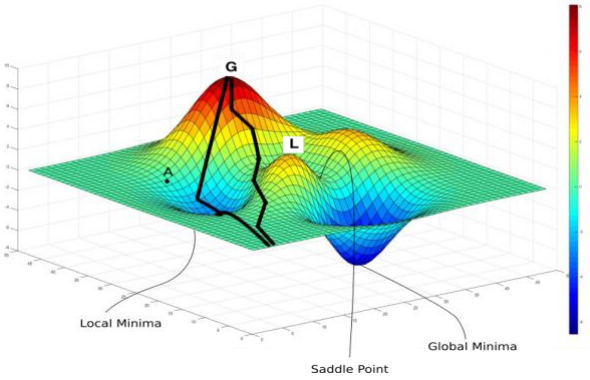


Figure 6: SGD and GD method. The lower straight black pattern in the direction of the global maxima, corresponds to the standard Gradient Descent method. The upper non-linear black pattern to the Stochastic Gradient Descent method. https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

a way to minimize the loss function and update the weights for all mentioned benefits of Adam optimizer compared to the GD and SGD.

2 Data

In this analysis we use a data consisting of dog and cat images. The data consists of three parts; training dataset, testing dataset and a pair of dog cat images where the training data will be used to train the CNN, the testing data will be used to obtain predictions and the single pair of images (1 dog image and 1 cat image) will be used to test the model in production. The training set consists of in total 8000 observations: 4000 pictures of dogs and 4000 images of cats. Moreover, the test set contains in total 2000 observations with 1000 pictures of dogs and 1000 images of cats. Following images are examples of few dog and cat images in the training data.

2.1 Data Preprocessing

As an initial step in the analysis we will prepare the data which will be different from the conventional data preprocessing approaches given that in this case we have a datasets based on images rather than numbers. For this we use two deep learning libraries: **Tensorflow version 2.4.1** and **Keras**. From the Keras library we will use one specific module called **ImageDataGenerator**.

Data Transformation or Image Augmentation

This filtering step will only be applied on the training set because we would like to avoid overfitting. We apply series of geometrical transformations such as shifting some of the

pixels, horizontal flips, zooming in and out to modify the images in the training set such they they become augmented [1].

3 Analysis and Results

For the entire analysis, from data preparation till evaluation of the results, we will use *Python* and we use the same Deep Learning Python libraries *Tensorflow*and *Keras* as before to train and validate teh CNN model.

Building CNN Model

To build the CNN model, firstly we initialize it by using *tf.keras* module and adding a *Sequential layer*. Then we start the process by calling the *Cov2D* function where we specify the following 4 parameters: **Filters, Kernel size, Activation function** and **Input shape** where *filters* are the feature detectors and we use a standard value of *32* which can be changed to any other more desirable value, *kernel size* represents the dimension of matrix that is used for each convolutional layer and in our case we want to have 3 by 3 matrices so the kernel size is 3, *activation function* is used in the ANN part of the model and as common practice we will use the *Rectifier* activation function. Finally, we use as an*image shape* [64, 64, 3] where we select 3 because the images in our dataset are colorful whereas in case of having black and white pictures, this value should be equal to 2.

In the next step we apply Max Pooling. We take the CNN model and add pooling layer to the convolutional model for which we use *MaxPool2D*. This requires the following parameters: *pooling size* for which we use a value 2 (recommended dimension of matrix when applying Max Pooling), *stride* the number of pixels that the frame is shifted to another for which we use again a default value of 2 that is the maximum of each 2 by 2 pixel/frame. In this step we also add a second convolutional layer with Max Pooling applied. Furthermore, we create a one dimensional vector that summarizes the pooled layer into a single vector that will be used as an input for the ANN. For which we use the *Flatten* class from the Keras Layers module. We then add fully connected Input Layer to the Sequential CNN by calling *Dense* class from Keras while specifying a Rectifier activation function. As a final step we classify each image to single class by using 1 hidden layer and a *Sigmoid* activation function.

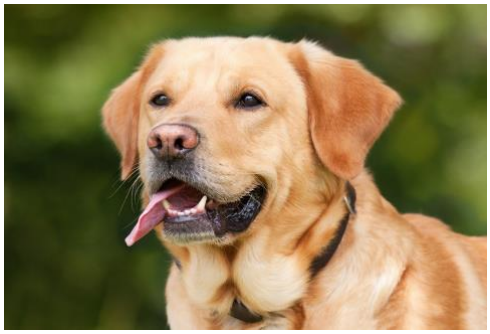
[1] https://keras.io/api/preprocessing/image/imagedatagenerator-class

Evaluation

Unlike many Machine Learning or Deep Learning models, in case of CNN the training and testing processes happen at the same time rather than firstly training the model and then using it to make the predictions. Firstly, we compile the CNN where we connect it to an *optimizer*, *loss function* and *evaluation metrics*. For the entire analysis, from data preparation till evaluation of the results, we will use Python.

Given that the output variable is a dummy variable, we use the *Binary Cross Entropy* as a cost function. Given that the focus of this study is of more descriptive nature, we don't prune the model parameters such as the number of epochs, therefore, 25 for the number of epochs.[2]

In order to evaluate our CNN model we use a pair of following images of a dog and a cat. If the model performs accurately then CNN model will classify the first picture as an image of a dog whereas the class of the second image will be class of cats.



Two functions that we use for this part are *load img*, to load the image, and *img to array* from **Keras's image** module. Following screenshot shows the results of a CNN model training and prediction when using as an input the dog's picture and the cat's image, respectively showing the accurately predicted classes; dog and cat, respectively.

[2] To train a CNN model takes usually much more time than an ANN hence we have chosen the minimum amount of epochs that will be just enough to give as proper prediction accuracy

```
251/251 [==============================] - 36s 142ms/step - loss: 0.3630 - accuracy: 0.8363 - val_loss: 0.4835 - val_accuracy: 0.783
Epoch 15/25
251/251 [==============================] - 36s 145ms/step - loss: 0.3493 - accuracy: 0.8390 - val_loss: 0.4801 - val_accuracy: 0.797
Epoch 16/25
251/251 [==============================] - 37s 146ms/step - loss: 0.3539 - accuracy: 0.8403 - val_loss: 0.4794 - val_accuracy: 0.800
Epoch 17/25
251/251 [==============================] - 36s 144ms/step - loss: 0.3329 - accuracy: 0.8550 - val_loss: 0.4808 - val_accuracy: 0.797
Epoch 18/25
251/251 [==============================] - 35s 138ms/step - loss: 0.3165 - accuracy: 0.8594 - val_loss: 0.5069 - val_accuracy: 0.808
Epoch 19/25
251/251 [==============================] - 34s 137ms/step - loss: 0.3128 - accuracy: 0.8688 - val_loss: 0.5349 - val_accuracy: 0.793
Epoch 20/25
251/251 [==============================] - 34s 137ms/step - loss: 0.2932 - accuracy: 0.8676 - val_loss: 0.4962 - val_accuracy: 0.790
Epoch 21/25
251/251 [==============================] - 34s 137ms/step - loss: 0.2658 - accuracy: 0.8876 - val_loss: 0.5106 - val_accuracy: 0.796
Epoch 22/25
251/251 [==============================] - 34s 137ms/step - loss: 0.2755 - accuracy: 0.8781 - val_loss: 0.5209 - val_accuracy: 0.798
Epoch 23/25
251/251 [==============================] - 35s 138ms/step - loss: 0.2416 - accuracy: 0.8989 - val_loss: 0.5025 - val_accuracy: 0.812
Epoch 24/25
251/251 [==============================] - 37s 149ms/step - loss: 0.2381 - accuracy: 0.8989 - val_loss: 0.5604 - val_accuracy: 0.787
Epoch 25/25
251/251 [==============================] - 37s 147ms/step - loss: 0.2371 - accuracy: 0.9056 - val_loss: 0.5098 - val_accuracy: 0.799
{'cats': 0, 'dogs': 1}
dog
cat
```

References

[1] Jay Kuo C.-C.(2016). Understanding Convolutional Neural Networks with A Mathematical Model 1–21.

[2] Kaiming H., Xiangyu Z., Shaoqing R., Jian S., (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Microsoft Research*, 1–11.

[3] Scherer D., Muller A., Behnk S., (2010). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. *20th International Conference on Artificial Neural Networks (ICANN)*,

[4] Glorot, X., Bordes, A., Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks *International Conference on Artificial Intelligence and Statistics, 15(15)*

[5] Rogerson, R. J. (2015). Adam: A Method For Stochastic Optimization. *3rd International Conference on Learning Representations (ICLR2015), 36(1)*, 1–13.

[6] LeCun, Y., Bengio, Y., Hinton, G., (2015). Deep learning. *Nature, 521*, 436 – 444.

[7] Wiegerinck, W. and Komoda, A. and Heskes, T. (1999). Stochastic dynamics of learning with momentum in neural networks. *Journal of Physics A: Mathematical and General, 27(13)*, 4425–4425.