Author: Ashley Arik (850904)

Comparing CNN and Fully connected Neural Networks in image Classification

Introduction

Using machine learning to recognize and classify images has been a popular discussion within in the technology industry, which has had billions of dollars poured into it's development. Understandable so, as getting a machine to recognize a complex image that is given to them is very useful. Some industries would be Telsa self-driving cars which feed real time videos of the surroundings into to determine the cars next action and recently China using facial recognition to identify what civilians (or criminals) are doing around the city.

This paper will contribute to the development of machine learning by comparing two model of machine learning (CNN and FCNN), on their performance in classifying a collection of images from the CIFAR100 dataset. Performance of the models can be measure in many ways, accuracy of model at identifying the images, speed of the model and processing and classifying the images and I would like to compare them on all these bases.

The reason why I chose these two models to compare is because Convolutional Neural Network (CNN) and a Fully connected Neural Network (FCNN) are very similar in design, which means their perfect for comparing. It also means their few differences can be more easily measured in their performance. The main difference between the models being that CNN has the addition of the convolutional layer. I predict that the CNN model will perform better then FCNN model due to the greater advantage CNN has with identifying images because of its Convolution layer that extracts features that otherwise would be lost then the data would be flatten by other models.

Methodology

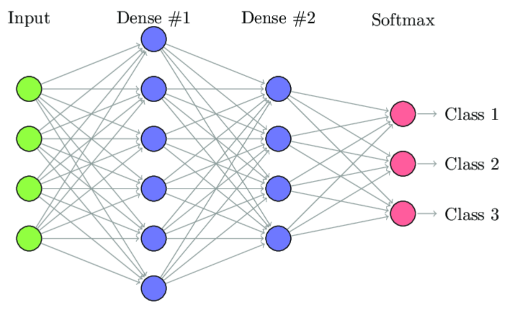
Routine of making a testing model:

- Loading all the data from files and setting them into variable
- Transpose Training and testing images into the right format
- Set layers of the model
- Compile the model
- Use “model.fit()” to train model using loaded train data and label
- Evaluate accuracy of the model using “model.summary()” and “model.evaluate()”
- Create Confusion matrix for model

We will be using 50,000 samples of 32*32-pixel RGB Images, to train the models and then 10,000 sample to test the data. Each sample will have 2 classes that will be used to classify what those images represent.

We'll be comparing two models in their performance in identifying images; that being CNN (convolutional Neural network) and a FCNN (Fully connected neural network).

FCNN



Fully Connected Neural Networks are made up of many layers of connected neurons. Each FCNN having 3 main types of layers, Input layer (colored green in Fig 1), Hidden layers (colored blue in Fig 1) and an output layer (colored in pink in Fig 1).

When creating the model, I set the “input_shape” as the shape of single image. This being the shape 32 * 32 * 3, where 32 is the height and width of image in pixels and the 3 is RBG of the pixel, where each value in the RGB array is a value from 0 to 1.

For my dense layers, I've decided to use Relu type because I've read that is reduces the impact of the vanishing problem. [1] [2]The Vanishing gradient refers the situation where the backpropagated error signal (gradient on the accuracy curve) decreases exponential as a function of the distance from the last layer. In simple terms, due to there being an error from layer to layer of the model it means that there will be a decrease of useful gradient information that will reach the beginning of the network from the end of the network as layers increases.

CNN Model

CNN is very similar to FCNN in the sense they both feedforward machine learning programs that have input, hidden and output layers.Where they differ is that CNN has a very important Convolutional layer, which contain

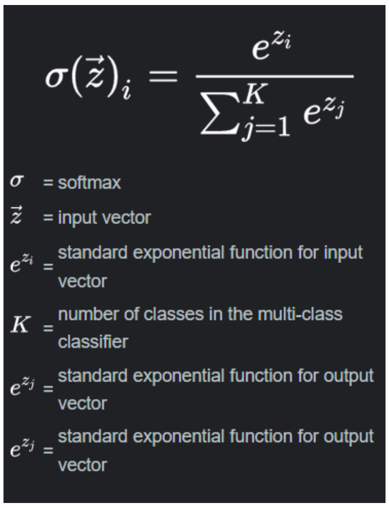
Convolution layer is the first layer to extract features from an inputted image. [3] This layer learns images features using small squares of input data, thus preserving the very important relationship between pixels. This is an important advantage, as most neural networks "Flatten" their input data (in the FC layer), which mean the inputted data gets converted into a 1-dimensional array so it can be more easily inputted into the next layer. This is where the Convolution layer is important, due to extracting the features into singular pixels.

Thus, the addition of this layer, gives CNN models a much greater advantage of FCNN which lose many of the images through flattening and additional layers.

After the Convolutional Layer there is the pooling layer which will reduce the number of parameter when images are too large. In simple in will condense e.g. a 8*8 feature map into a 2*2 feature map if the pooling has a filter of 2*2 with a stride of 2. Filter being the section of pixels that would be condensed, and the stride is the controls how the filter convolves around the input volume.

There is a chance of overfitting, where the model will highly be trained to solve the classes of exactly the images, I give it but will have a low score when classifying images/data that is new. In order to combat this, I have added a "Dropout layer" to my CNN which randomly sets input units to 0 with a frequency of rate at each step during training time. This forces the model to know get different inputs for images as we're training it, so it doesn't become too used to training for the exact same images.

For my final layer in the CNN I will be using a "softmax" layer. This assumes that each example is a member of exactely one class which is very useful when trying to train a mode to identify images as from one classes, and attibutes that images features to just that class. **In simple, for every picture it outs a value for every class of that likelyhood that that singluar images is of being in that class.**



$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax
$\vec{z}$ = input vector
e^{z_i} = standard exponential function for input vector
K = number of classes in the multi-class classifier
e^{z_j} = standard exponential function for output vector
e^{z_j} = standard exponential function for output vector

~~Convolutonal preserves the local features of the images that we can use to enhance the machine learning model due to better~~

<u>Measure Effectives of the model:</u>

There are many ways to measure the effeteness of machine learning Model. Firstly I used the "tensorflow.keras.models.summary" and "tensorflow.keras.models.evaluate" function to give a quick written summary. "Summary()" prints out a summary of the network, which gives information on number of neuron in each layers of model and shows the number of parameters passed in at each layer. "Evaluate(X_Train, y_Train, Verbose)” use very useful as it returns the loss value and m important accuracy when it came to the model predicting the c

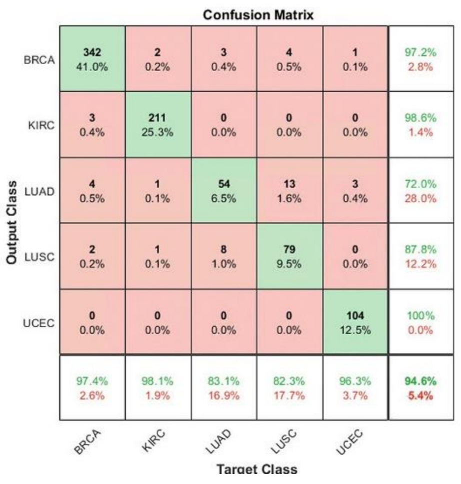


Figure 1: Confusion matrix example from "ResearchGat" website

Confusion Matrix

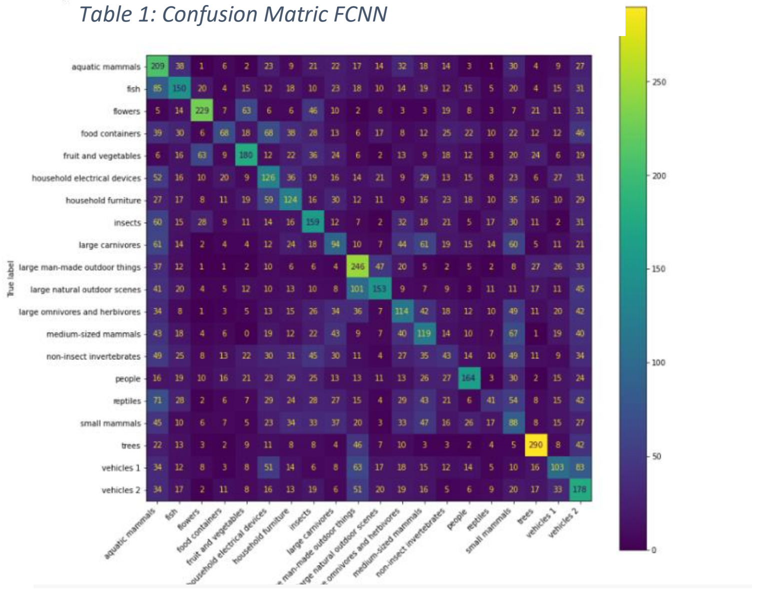
It terms of measure performance, by far the most useful metric is the Confusion Matrix. Which is a table that shows the number of time my machine learning model would have guessed the right label for the image. For example in figure 2 [4], when given 350 image with label BRCA the model guessed correctly 342 (Model has 97.2% success rate of guess right when given image with that label). In simple it predicts how likely the trained model is going to be at predicting the given image the right label

Results

FCNN

To my pleasant surprise, my FCNN models are greatly more efficient than I expected them to be reach accuracy of 0.5072. I have found that flattening the data was better saved to right at the second last layer, so the model can calculate predicting with the least abstraction/most feature data from the original data. Although over all this increased my accuracy of my model, I found it to be much slower than other models, as the time between each epochs was doubled. Through repeat experimentation I have found that having a large first layer.

Table 1: Confusion Matric FCNN



As you can tell from FCNN’s Confusion Matrix in Table 1, The model is doing well at correctly labeling the images (as can be seen from clear line going from the top left to the bottom right).

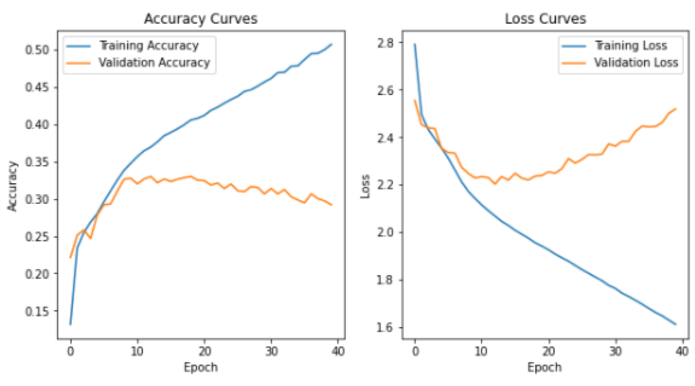
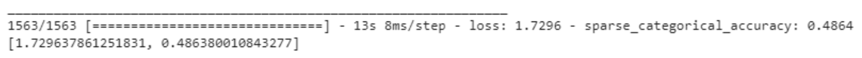


Table 2: Accuracy Line chart of FCNN

Although The model seems to have some confusion on specific classes (for good reason) such “small mammals”, its gets confused between that people, medium-sized mammals, large carnivores. Which makes complete sense because the shape features of these classes would be close since their all “animal shaped”. The model seems to be having trouble with animals in general, as it also confuses “non-insect invertebrates”, “retiles” which is understandable. Animals are very similar in nature, cause a lot of species evolve in the same features to survive, and for added complexity animals can move and stretch, meaning key features of them can be distorted.



```
1563/1563 [==============================] - 13s 8ms/step - loss: 1.7296 - sparse_categorical_accuracy: 0.4864
[1.729637861251831, 0.486380010843277]
```

Supporting theory is showing what the model is best at: “Trees”, “Flowers” and “Large man-made outdoor things” all being the most predicted correctly by a large margin. All inanimate objects that are unable to move and have very simple features/edge; trees for example can be very easily recognize with two large parallel lines heading, which is a feature noticed in every tree image, unless if the image was rotated.

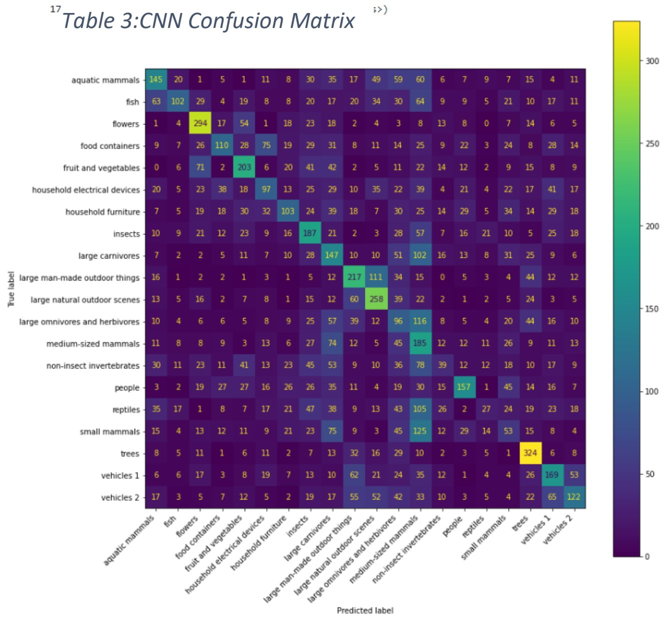
CNN

In the end, my CNN model had an accuracy of only 0.3145

Like the FCNN model, they seem to be having trouble and being good at predicting the same things. For example, "Trees", "Large man-mad outdoor things" and "Flowers" being the top correctly Predicted Items. The CNN model is also having trouble distinguish the different animals from each other.

Although, it is coming with the same strength and weakness in classes, what CNN get's right, it gets extremely right. E.g. Even though FCNN model has a high accuracy overall, the CNN is out preforming it ALL of it's top class predicts; "trees", "man-made…" etc.

Table 3:CNN Confusion Matrix



Comparing Accuracy Curves

Even comparing the Accuracy Curves overall, the CNN model looks be a lot smoother and more consistent with its training to validation accuracy. It consistent but has a lower accuracy curve overall, which suggests with the right training could outperform the inconsistent FCNN (as seen by the diverging Loss curves).

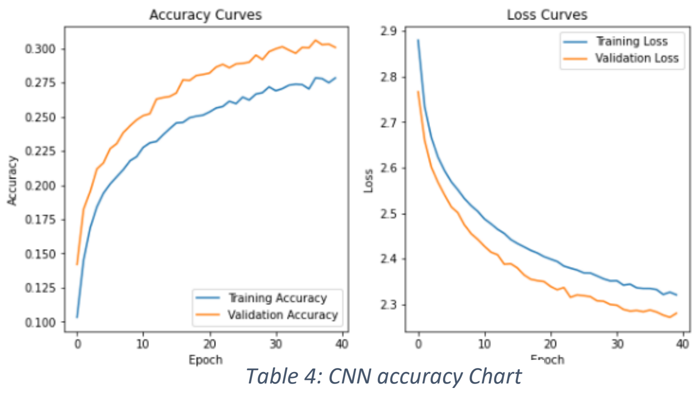


Table 4: CNN accuracy Chart

Conclusions

In conclusion, I was unable to show how CNN could be better overall as I've been able to create a better FCNN model (in overall accuracy). Although I have shown the power of the CNN when it does work, for example when the CNN model was got the simple features for Trees it got the right prediction almost every time 324/350 which is a 92.6 rating whilst FCNN could only get a high rating of 290/350 a 8.28 percentage rating. Even though the FCNN model had an overall rating of 0.18 higher than CNN (0.49 to 0.31).

As stated in my Methodology, CNN has every reason to have an advantage in image classification. It convolutions would have should have given it every advantage against the FCNN. The only conclusion is that what was wrong with the model was the creator, user error. In future work, I aim to increase variety of neurons in each layer to test the change in overall accuracy. It is also possible that I set the dropout layer too high, at it set too many of the data to 0, thus the model couldn't properly train on complete Images. Finally, I wonder how the increase in Epoch would effect the results, as at the beginning I chose to work with only 40 because accuracy was not increasing at a significant rate when it passed that threshold. But I believe it I had done the same experiment with more Epoch, then the CNN would have more times to reach its full potential that could surpass the FCNN whilst the FCNN accuracy would flatline on the graph.

Author: Ashley Arik (850904)

References

[1] "https://github.com/chetnakhanna16/CIFAR100_ImageRecognition/blob/master/Project_Paper_001081074.pdf," [Online].

[2] "https://medium.com/analytics-vidhya/how-batch-normalization-and-relu-solve-vanishing-gradients-3f1a8ace1c88," [Online].

[3] "https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148," [Online].

[4] https://www.researchgate.net/figure/Confusion-matrix-for-60-training-and-40-testing-strategy_fig4_338909223. [Online].