**Image Classification: Who’s that Doggie in the Window?**

Introduction

One of the biggest topics right now in technology is how do we create programs that think for us. Artificial intelligence (AI) has been on the tips of everyone’s tongue for years. One of the components that makes up AI is neural networks. Neural networks are very much like human neurons, which take an input, process it, then sends an output. This system of virtual neurons is used to learn from multiple inputs and predict an output. A great example of this process is computer vision (CV). An image is sent through a network, with predictions about what is in the image as an output. This project will take a real-world example of this to show how neural networks can be created in a multitude of ways.

Abstract

There have been lots of advancements when it comes to computer vision. As technology improves, computer vision applications are able to identify lots of different objects. For this project an image classifier will be created to help distinguish different dog breeds. The dataset being used has 120 different breeds that need to be identified. A convolution neural network (CNN) will be created to identify these images. Also, three different pre-trained models will be used to determine if transfer learning is more effective than a custom model. These three models are VGG19, ResNet152V2, and MobileNetV2.

Research Questions

The two main research questions for this project are:

1. How can a custom neural network be created in Keras to identify one of 120 dog breeds from an image?
2. Is using a pre-trained network trained via transfer learning more effective than a custom-built model?

Methods

In order to create an image classification network, images that have already been labeled needs to be used. The dataset that will be used is the Dog Breed dataset from Stanford University (Khosla, Jayadevaprakash, Yao, & Fei-Fei, n.d.). Overall, this dataset contains almost 1GB of images belonging to 120 different classes (in this case dog breeds). The images do not require further processing, more of a rearrangement. In order to make the model creation steps easier three separate directories must be created: train, test, and validation. The dataset contains 120 different directors, one for each label. In order to create the three different directories a program must be created to randomly select images for each. Luckily, there is already a library written for this task. The library is called split\_folders, it takes a few parameters: directory of images, random seed, and a tuple of ration for train, test, and validation folders. With the images reorganized its time to process and load them into the Python program.

To create a neural network to classify images, the Keras framework will be used. Keras is described as an “open source neural network library written in Python (Heller, 2019).” It offers easy APIs that work with TensorFlow in deep learning applications. This makes it pretty easy to create a network, save it, and predict from it. Before creating a model, the images need to be loaded first. In Keras there is a function called ImageDataGenerator that can handle any preprocessing needs for images. In this instance it will be used to scale the images down so the pixels have a value between 0 and 1 instead of 0 to 255. Then, the ImageDataGenerator object can be used to flow the images into the Python program by way of flow\_from\_directory. Now with the images loaded it’s time to create the network!

The first network that will be created is a custom one, meaning it is not reliant on any transfer learning from a pre-trained network. The specific network being used here is a Convolution Neural Network (CNN). CNN’s are effective in computer vision applications and typical consist of Conv2D, MaxPooling2D, and Dense layers (Saha, 2018). For this model four Conv2D and MaxPooling2D layers are used along with one Dense layer (with 2048 neurons) once the data has been flattened. Two dropout layers with a ratio of 0.4 will be used to help prevent overfitting.

Using a pre-trained model is somewhat similar to creating one from scratch, there are a few caveats that need to be remembered. In this case, the top layer has been excluded because that is where the predictions are made. In order to ensure training goes smoothly, the layers in the pre-trained model need to be frozen. When playing around with creating some networks I discovered the loss and accuracy values were awful. Once I discovered I was trying to retrain the pre-trained layers, things started to look much better. For this project three different pre-trained models were investigated: VGG19, ResNet152V2, and MobileNetV2. The code that creates the networks in pretty much the same for all three, the only change is the base model that’s used.

Results

Since four different neural networks are being evaluated, a table comparing all of the loss and accuracy values will be produced. Table 1 (below) contains these results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Network** | **Training**  **(Loss)** | **Training**  **(Accuracy)** | **Validation**  **(Loss)** | **Validation**  **(Accuracy)** | **Testing (Loss)** | **Testing**  **(Accuracy)** |
| Scratch | 4.78 | 0.67% | 4.82 | 1.24% | 4.80 | 1.21% |
| VGG19 | 3.09 | 26.25% | 4.42 | 13.34% | 4.44 | 13.1% |
| ResNet152V2 | 3.29 | 22.08% | 4.30 | 19.41% | 8.66 | 18.44% |
| MobileNetV2 | 1.44 | 61.97% | 4.51 | 46.89% | 2.23 | 45.01% |

Table 1 - Training, Validation, and Testing Loss and Accuracy for Four Different Neural Networks

The results were definitely surprising to me! I knew going into this project that a model built from scratch would not perform well. However, I expected some pre-trained networks to perform a lot better than they did. When training all of these networks the one that trained the fastest and used the least amount of computer resources was MobileNetV2 (the network that performed the best!)

Conclusion

Even after training four different networks using different architectures, there is still no model in this batch that can be labeled as a high performer. Out of these four the clear winner was MobileNetV2. It achieved much higher accuracy while having the lowest loss value. Because of how time-consuming image classification can be each of these models was trained with the same number of epochs and steps. From this information gathered, more research should be made into creating networks from MobileNetV2. Its ability to train quickly with less resources, while providing the best results makes it the clear winner!

Bibliography

Heller, M. (2019, January 18). *What is Keras? The deep neural network API explained*. Retrieved from InfoWorld: https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html

Khosla, A., Jayadevaprakash, N., Yao, B., & Fei-Fei, L. (n.d.). *Stanford Dogs Dataset*. Retrieved from Stanford University: http://vision.stanford.edu/aditya86/ImageNetDogs/

Saha, S. (2018, December 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Retrieved from Towards Data Science: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53