Transfer Learning and Residual Networks

Pavan Sekhar Naidu

July 13, 2024

1 Introduction

In this project, we utilize transfer learning to develop efficient image classifiers by leveraging the pre-trained models VGG16 and InceptionNet. Transfer learning allows us to use models that have already been trained on large datasets, thereby significantly reducing the training time required for our task. These models have already learned to extract general features from images, making them highly effective for various image classification tasks.

2 Why Transfer Learning?

- **Reduced Training Time**: Since VGG16 and InceptionNet are pre-trained on large datasets, they have already learned to identify general features in images. This eliminates the need for extensive training from scratch.
- **Improved Performance**: Pre-trained models typically provide better performance due to their ability to extract meaningful features from images.
- **Efficient Resource Utilization**: Utilizing GPUs for training these models allows for parallel execution, significantly speeding up the training process compared to using CPUs.

3 Custom Model with Residual Networks

In addition to using pre-trained models, we design a custom Convolutional Neural Network (CNN) model incorporating residual networks. Residual networks address common issues faced by deep networks, such as vanishing gradients and increasing training loss as the number of layers increases.

3.1 Key Points from Research

- **Degradation Problem**: As networks become deeper, they often face degradation where adding more layers results in higher training error.

- **Vanishing Gradients**: With more layers, gradients can become very small, making it difficult for the network to learn.
- **Residual Learning**: By using residual connections, where the input to a layer is added to its output, the network can learn residual functions instead of direct mappings, which simplifies the learning process and improves performance.

3.2 Residual Block

Consider a simple neural network block where x is passed through two convolutional layers to compute $F(x)$:

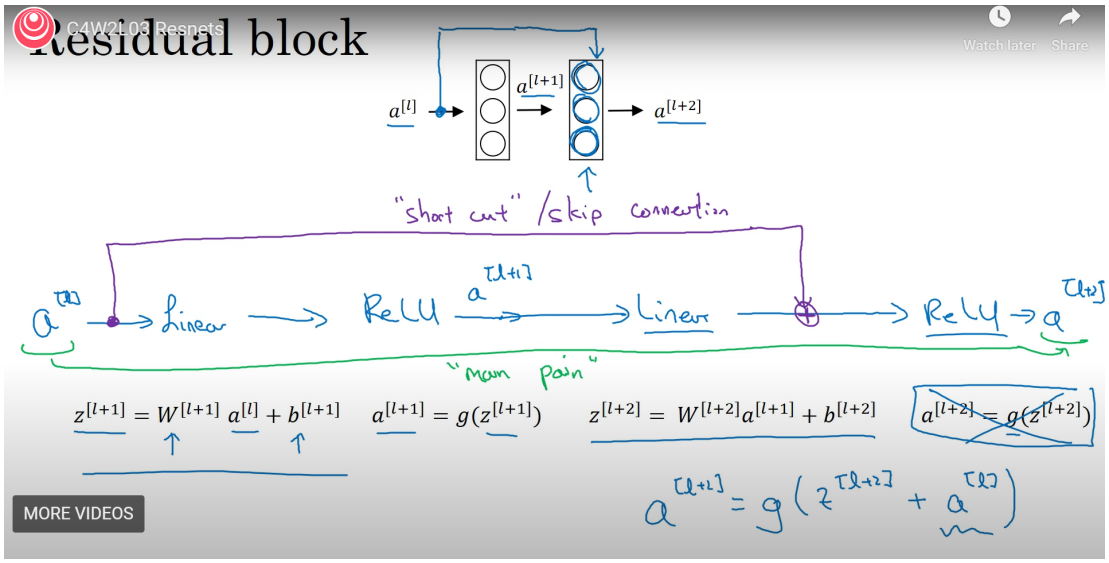

Figure 1: Resdual Block

- **Convolutional Layers**: Let W_1 and W_2 be the weights of the first and second convolutional layers, respectively. The residual function $F(x)$ is computed as:

  $$F(x) = \sigma(W_2 \cdot \sigma(W_1 \cdot x))$$

  Here, σ denotes the activation function (e.g., ReLU).

- **Residual Block Output**: The output of the residual block is then:

  $$y = F(x) + x$$

4 Challenges

- **Dataset Handling**: Ensuring proper downloading and handling of the dataset from Kaggle.
- **Shape mismatch**: adding x to the F(x), shapes mismatched so again I convoluted x to change it into shape of F(x) and then proceeded

5 Running the Code

5.1 Step-by-Step Guide

For guidance on setting up Kaggle datasets in Google Colab, refer to the following tutorial: Kaggle Dataset Setup on Google Colab.