Kolby Boyd

ITAI 1378

Prof. McManus

L08

Chihuahua or Muffin? Model Improvement & Reflection

In this project, the goal is to develop a neural network capable of differentiating between images of chihuahuas and muffins. This task will serve as a practical demonstration of image classification, a fundamental problem in computer vision. By training and optimizing a model to correctly identify these categories, I aim to show the capabilities of deep learning algorithms in discerning the subtle differences within these images.

Setting up the environment

1. **MatPlotLib.pyplot** is imported as a graphical library to plot images. Then there is a special Jupyter notebook command to show the plots inline instead of in a new window for ease of use.
2. **PyTorch** is a deep learning framework
3. **TorchVision** is an extension to PyTorch for dataset management
4. **Torch.NN** is the neural networks module of PyTorch that lets us define neural network layers
5. **Torch.NN** import functional brings in some special functions
6. **Torch.Optim** brings in some optimizers

These libraries are essential for building and training deep learning models using PyTorch. Using these libraries simplifies the process of building, training, and evaluating deep learning models in PyTorch, making it more efficient and accessible for us to use.

Data Preparation

The data comes from Module 08. It is in a zip folder called ‘data’.

Inside the zip folder, there is a folder called ‘train’, and one called ‘validation’. Inside of each are subfolders named ‘muffin’, and ‘chihuahua’. The training subfolders contain thousands of images of either chihuahuas or muffins, while the validation contains hundreds of these images. One subset will be used to train the model and the other will be used to validate the model before setting it loose on unseen data.

For both training and validation data, we resize the images to the ‘input_height’ and ‘input_width’ specified in the beginning of the lab. This makes sure that the input sizes are all uniform. Then we converted the images into PyTorch tensor objects. This converts the image data into a compatible format with PyTorch, allowing the neural network to process it. Finally, we normalized the pixel values of the images. The mean and standard deviation values used for this normalization are ‘[0.5, 0.5, 0.5]’ which means that the red, green, and blue channels are normalized independently to have a mean of 0.5 and a standard deviation of 0.5. This ensures that the pixel values are within a standardized range, which aids in the training process.

NN Architecture

The model chosen for this task is a simple neural network known as a multilayer perceptron. MLPs are very simple, yet they can still perform well for basic image classification tasks when combined with appropriate preprocessing techniques.

The architecture consists of an input layer for flattened input images, three hidden layers with decreasing numbers of neurons and ReLU activation functions, and an output layer with two neurons using softmax

activation function to generate class probabilities. Also, the code uses the Cross-Entropy Loss function and Stochastic Gradient Descent optimizer for training the model.

Training the Model

Loss Function: Cross Entropy Loss, which measures the difference between the predicted probabilities and the actual labels.

Optimizer: Stochastic Gradient Descent (SGD) updates the weights of the network in the direction that minimizes the loss, which improves the model's performance.

Epochs: 3 epoch training sessions give the model sufficient opportunity to learn the patterns in the data.

Evaluating the Model

Metrics: We use accuracy and the error rate (loss). These are calculated during the training process for both training and validation datasets. The error rate shows how well the model is performing, while accuracy measures the proportion of samples that are correctly classified.

Results: After each epoch the code prints the error rate and accuracy for both the training and validation datasets. During the first epoch, the model had an accuracy of 61.34% on the training set with an error rate of 0.6756, while on the validation set, the accuracy was 62.42% with an error rate of 0.6610. In the second epoch, the model improved slightly, reaching an accuracy of 71.67% (error rate 0.6411) on the training dataset and 74.83% (error rate 0.6304) on the validation set. Finally, the third epoch shows the model continuing to improve accuracy with the achievement of 76.40% accuracy and an error rate of 0.6206 on the training set and 75.84% accuracy with an error rate of 0.5782 on the validation dataset. Overall, the model continued to progress in its learning through the training process.

Improving the Model: Tweaks & Adjustments

Architecture: I increased the number of neurons in the initial layers and removed one of the hidden layers to allow for more complexity in the learned features.

Results: This improved the model's ability to capture the complex patterns in the data. Both the training and validation datasets resulted in higher accuracy and lower loss.

Image Resizing: I changed the image resizing function to resize the images to 250 pixels instead of 50 pixels.

Results: This increased the pixel amounts that the model can work with and potentially improved its ability to capture finer details and features in the images.

Learning Rate Optimization: I changed the learning rate from 0.1 to 0.06 in the hopes that the training would be slower but more stable.

Results: This has a significant impact on the training process. Convergence was more stable and the optimal solution was prevented from being overshot. It was definitely a little slower but the performance over the epochs improves consistently without much overfitting.

Theoretical Concepts

Neural Networks: MySkyNet class defines a neural network with multiple neuron layers. It includes an input layer, multiple hidden layers, and an output layer. Each layer is connected to the next layer, allowing information to propagate through the network during both the forward pass and the backward pass during training.

Convolutional Layers: Convolutional layers are commonly used in image classification tasks, where the input consists of images represented by pixel values. Pooling layers are used to extract features from the input layers after the convolution process.

Activation Functions: The model uses ReLU activation functions. They are applied after each linear layer to introduce non-linearity to the network's outputs and this enables it to learn complex representations of the input data.

Backpropagation: This process is used during the training process, where the optimizer updates the weights of the network based on the computed gradients using the loss function (Cross Entropy). The backward method of PyTorch's tensors is used to compute gradients during the backward pass and the optimizer updates the weights based on these gradients.

Differences Between Image Classification & Object Detection

Image Classification involves assigning a label (known as a class) to an image based on its content. This aims to identify the main object or scene depicted in the image from a predefined set of categories. For example, given the image of a dog, an image classification model would predict that the image belongs in the "dog" category. Image classification is best suited for tasks where the goal is to classify images into a single category. It is commonly used in applications like autonomous driving or in medical imaging where the presence or absence of specific objects needs to be determined. Image classification models usually consist of a CNN architecture which learns to extract features from input images and classifies them into different categories. The output layer of the network usually uses a softmax activation function to produce probabilities of the classes and the model is trained using loss functions such as the cross-entropy loss function.

Object Detection is the process of identifying and localizing multiple objects within an image by drawing boundary boxes around them and assigning a class label to each detected object. Image classification classifies the entire image, while object detection provides more nuanced and detailed information about the location and class of objects present together in a single image. Object detection is perfect for tasks where the goal is to detect and localize multiple objects within an image along with classifying them into different categories. Autonomous driving also uses this to detect pedestrians, vehicles and traffic signs. Other applications include surveillance systems for identifying people, and in retail for tracking and analyzing behavior of customers. Object detection models often use architectures like Faster R-CNN or YOLO (You Only Look Once). These models are composed of a region proposal network (this generates candidate object regions or loose boxes) and a classification network (for refining the boundary boxes). Object detection models are trained by the use of labeled datasets which contain the boundary box labels for each instance of an object and they are optimized using classification and localization loss functions.

Conclusion

In this project, we aimed to build a neural network capable of differentiating between images of chihuahuas and muffins. We began by setting up our environment and preparing our dataset. Training the model involved monitoring metrics like accuracy and error rate across epochs, with gradual improvements indicating successful learning. Through these efforts, we achieved a model capable of semi-accurate classification, which highlights the potential of deep learning and computer vision applications.