Project: 'Deep Learning: Image Classification with CNN'

Report

Dani Siaj & Carlos R. Vidondo

October, 2024

1. Loading the data set ‘CIFAR-10’

We tried to different options:

- We downloaded the CIFAR-10 dataset from cs.toronto.edu website, and used the function ‘unpickle’ provided by the website. This way, the dataset comes divided into 7 batches (5 for training, 1 for testing, and 1 for the metadata that has the labeling).

- Without any download. We imported it directly into the jupyter notebook using ‘tensorflow.keras.datasets’, which allowed us to even separate both training and testing sets in a single code line.

2. Preprocessing Steps

Then we completed two mandatory steps for the pixel data and class labels to be more manageable by the model:

- Normalization: Images were converted into floats and scaled to have pixel values between 0 and 1, which helps the model converge faster.

- One-hot encoding: Labels were converted to categorical format to suit the categorical cross-entropy loss function.

3. Model building and hyperparameters

We work on two different models: **Pytorch** and **Tensorflow.**

For each model we have built various **blocks of convolutional layers** with different filters, using ReLu as the activation function and padding set on ‘same’, so we don’t lose any dimensions.

In between each Conv2D layers we added a **batch normalization** that helps to stabilize and faster the training.

Each block of Conv2D is separated by **Max-pooling** to reduce spatial dimensions and a **Dropout (**that increases its size rate in each block) to help reduce the overfitting.

In the end, we **flattened** and added another Dense layer to fully connect the layers before the ‘softmax’ function that ensures the output

4. Training Process

While **compiling** the model, we used the Adam optimizer but lowering as much as possible the learning rate to 0.001 (helps the model learn finer details) and we chose the accuracy metric to track the performance of our model. As the loss function we used ‘categorical_crossentropy’.

In the beginning, we **trained** the model using a sample data of 8.000 images. We set the number of epochs to 60 to define how many times the model will see the entire training set; we set an image batch size of 32, and the validation split was 10% of the training data.

5. Model performance (evaluation metrics)

After computing the accuracy score, the CNN tensorflow model achieved a test accuracy of 72.2% and a test loss of 1.31% on unseen data. But actually, as the plotting shows, after the 30th epoch the model stabilizes on a plateau.

The CNN Pytorch model achieved a test accuracy of 84%, showing a training curve that slowly flattens after the 15th epoch.

Confusion matrix revealed:

- Strengths in categories like frogs, trucks and ships.
- Confusion between classes like cats and dogs or trucks and automobiles, which visually are very similar.

6. Transfer Learning

We chose Xception and VGG16 because they are both widely used architecture for image classification tasks compared to models like Inception or ResNet, making it easier to fine-tune for a dataset like CIFAR-10, which contains smaller 32x32 images. Both are pre-trained on weights from ImageNet, that offer robust feature extraction.

Xception

- Test loss: 0.15
- Test accuracy: 0.82

VGG16

- Test loss: 1.716
- Test accuracy: 0.652

6. Save and load

We saved each CNN model (both Tensorflow and Pytorch) and each Pre-trained Model (Xception and VGG16) using pickle files. Then anyone can use these trained models later.