Artificial Intelligence
CNN

Fall 2024
Parsa Darban 810100141
This file contains the report and results of the simulations conducted.

Abstract

In this project, we explore convolutional neural networks (CNNs) and implement the VGG16 model. The model is trained and tested on the CIFAR-10 dataset and classify the data accordingly.

A Convolutional Neural Network (CNN) architecture is a deep learning model designed for processing structured grid-like data, such as images. It consists of multiple layers, including convolutional, pooling, and fully connected layers. CNNs are highly effective for tasks like image classification, object detection, and image segmentation due to their hierarchical feature extraction capabilities.

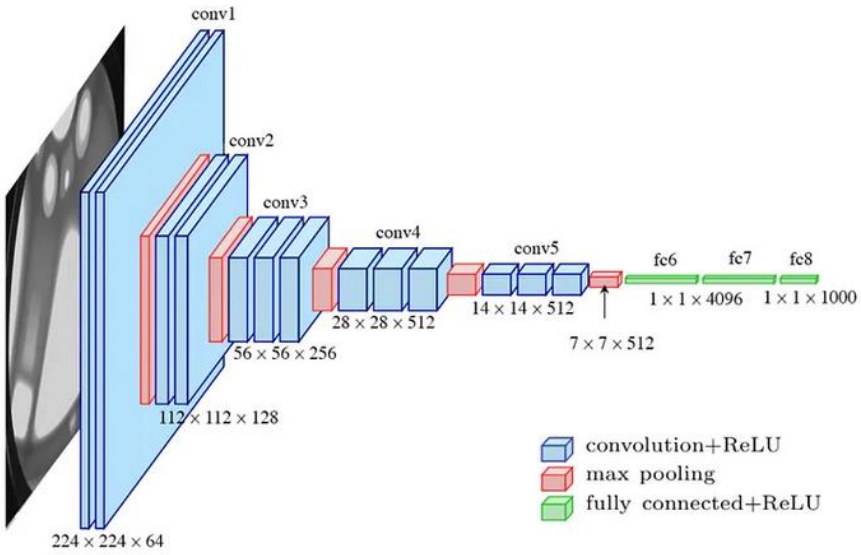
VCG-16

The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model's architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth. This design enables the model to learn intricate hierarchical representations of visual features, leading to robust and accurate predictions. Despite its simplicity compared to more recent architectures, VGG-16 remains a popular choice for many deep learning applications due to its versatility and excellent performance.

The VGG-16 architecture is a deep convolutional neural network designed for image classification tasks, consisting of 16 weighted layers. It begins with convolutional layers that extract features from input images using small 3×3 filters and employs same padding to preserve spatial dimensions. These are grouped into blocks with increasing filter depths—64, 128, 256, and 512 filters—across the network. Each block ends with a 2×2 max-pooling layer, reducing spatial dimensions while retaining important features.

The architecture includes three fully connected layers after the convolutional blocks. The first two layers have 4096 neurons each, and the final layer maps to the number of classes (10 for CIFAR10), applying a softmax activation to produce class probabilities. ReLU is used as the activation function throughout the network for non-linearity. Flattening is performed before entering the fully connected layers to transition from spatial data to dense layers.

VGG-16's design emphasizes simplicity and depth, with fixed-size filters, consistent structure, and progressively deeper feature extraction. This hierarchical approach captures both low- and high-level features, making the model highly effective for image classification, though computationally expensive due to its large number of parameters.



However, after training this simple model on the data, the accuracy converges to around 10%. Therefore, we consider improving the model using a series of techniques.

Batch Normalization

Batch Normalization (BN) is a technique to normalize the inputs of each layer in a neural network. It stabilizes and accelerates the training process by ensuring that the inputs to each layer maintain a consistent distribution. BN reduces the problem of internal covariate shift, which occurs when the distribution of inputs to a layer changes due to updates in preceding layers during training.

It's first compute mean and variance for a given mini-batch of data with activations x:

$$\mu_B = 1/m \sum_{i=1}^{m} x_i \text{ and } \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

Here, m is the batch size, μ_B is the mean, and σ_B^2 is the variance of the batch. After that normalize the input by the computed value and then scale and shift to make the next layer input:

$$\overline{x_i} = \frac{x_i - \mu_B}{\sigma_B}$$

$$x_i = \gamma \overline{x_i} + \beta$$

BN learns to optimize the normalization parameters (γ and β) during training, allowing the network to retain representational power.

Batch Normalization (BN) accelerates training by stabilizing the distribution of inputs to layers, allowing higher learning rates and reducing sensitivity to initialization. It introduces a regularization effect, slightly reducing overfitting due to batch-wise statistics. BN also mitigates vanishing /exploding gradients by normalizing intermediate activations, ensuring gradients remain within a manageable range during backpropagation. These factors contribute to improved generalization, helping the network perform better on unseen data, making BN a crucial technique for enhancing the efficiency and performance of deep neural networks. It is particularly useful in deep networks, where gradient instability such as vanishing or exploding gradients can be a problem. It is widely applied in convolutional layers, especially for tasks involving datasets like CIFAR-10 and ImageNet, to improve network stability. BN can also be used before fully connected layers to stabilize dense layers, although Dropout may suffice in smaller networks. Additionally, BN enables the use of higher learning rates, facilitating faster convergence during training. This makes BN an essential tool in improving the performance and efficiency of deep learning models.

Batch Normalization (BN) has some limitations, including its dependence on batch size, as performance may degrade with very small batches due to noisy statistics. It also introduces additional computational overhead and memory usage for maintaining the mean and variance statistics. Furthermore, BN is not always the best choice for sequence models like RNNs, where alternative normalization techniques such as Layer Normalization or Group Normalization may perform better in certain cases. These limitations should be considered when choosing the appropriate normalization technique for specific tasks.

Dropout

Dropout is a regularization technique used in neural networks to prevent overfitting. During training, it randomly "drops out" (sets to zero) a certain percentage of neurons in the network. This prevents the model from becoming overly reliant on specific neurons and helps it generalize better to unseen data. Essentially, Dropout forces the network to learn redundant representations and reduces the risk of overfitting by preventing co-adaptation of neurons.

During training, for each mini-batch, a random subset of neurons is ignored (set to zero). This is done independently for each training step. The dropout rate (usually between 20% and 50%) determines the fraction of neurons to be dropped. During inference (testing), no neurons are dropped, but the output of each neuron is scaled by the inverse of the dropout rate to maintain the correct output distribution.

In this project, p=0.5. During training, half of the neurons are dropped, and the other half are scaled by $\frac{1}{1-p} = 2$. During inference, all neurons are kept, but their outputs are scaled by 0.5 to match the expected activations seen during training. The reason for the scaling factor $\frac{1}{1-p}$ is to ensure that the expected output during training is equal to the output during inference. Without this scaling, the model would produce smaller outputs during training, which would affect the learning process. By scaling the outputs during training, we maintain the same expected sum of activations, thus ensuring stable learning. This simple mechanism allows Dropout to act as a form of regularization, forcing the model to learn robust features and preventing it from overfitting to the training data.

So, Dropout is a regularization technique that prevents overfitting by randomly disabling a fraction of neurons during training, forcing the model to learn more robust and generalized features. This approach helps the model generalize better to unseen data, especially in large networks like VGG16, where overfitting is a concern due to the large number of parameters. It is typically applied after fully connected layers, where the model is more likely to overfit due to the dense nature of the connections.

However, Dropout does have limitations. It introduces additional noise during training, which can slow down convergence and make the optimization process more challenging. The use of Dropout can also lead to longer training times, as the model has to learn with fewer active neurons at each iteration. Additionally, while it helps prevent overfitting, it may not be sufficient for every problem, and alternative regularization methods may be more effective in certain scenarios, such as for sequence-based models or very small datasets.

The summary of our model is as follows:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	1,792
BatchNorm2d-2	[-1, 64, 32, 32]	128
Conv2d-3	[-1, 64, 32, 32]	36,928
BatchNorm2d-4	[-1, 64, 32, 32]	128
Conv2d-5	[-1, 128, 16, 16]	73,856
BatchNorm2d-6	[-1, 128, 16, 16]	256
Conv2d-7	[-1, 128, 16, 16]	147,584
BatchNorm2d-8	[-1, 128, 16, 16]	256
Conv2d-9	[-1, 256, 8, 8]	295,168
BatchNorm2d-10	[-1, 256, 8, 8]	512
Conv2d-11	[-1, 256, 8, 8]	590,080
BatchNorm2d-12	[-1, 256, 8, 8]	512
Conv2d-13	[-1, 256, 8, 8]	590,080
BatchNorm2d-14	[-1, 256, 8, 8]	512
Conv2d-15	[-1, 512, 4, 4]	1,180,160
BatchNorm2d-16	[-1, 512, 4, 4]	1,024
Conv2d-17	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-18	[-1, 512, 4, 4]	1,024
Conv2d-19	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-20	[-1, 512, 4, 4]	1,024
Conv2d-21	[-1, 512, 2, 2]	2,359,808
BatchNorm2d-22	[-1, 512, 2, 2]	1,024
Conv2d-23	[-1, 512, 2, 2]	2,359,808
BatchNorm2d-24	[-1, 512, 2, 2]	1,024
Conv2d-25	[-1, 512, 2, 2]	2,359,808
BatchNorm2d-26	[-1, 512, 2, 2]	1,024
Linear-27	[-1, 4096]	2,101,248
Dropout-28	[-1, 4096]	0
Linear-29	[-1, 4096]	16,781,312
Dropout-30	[-1, 4096]	0
Linear-31	[-1, 10]	40,970

Total params: 33,646,666
Trainable params: 33,646,666

Classification

Next, we load the data using transformers within the code and split it into test and training datasets. Our dataset is as follows:

Criterion

CrossEntropyLoss is a commonly used loss function in multi-class classification tasks. It combines two main operations: Softmax, which transforms raw output logits into probabilities by exponentiating each logit and normalizing them, and Log Loss, which calculates the difference between the predicted probabilities and the true class labels. The loss function penalizes the model more when its predictions deviate significantly from the actual class. The formula for CrossEntropyLoss involves summing the negative logarithms of the predicted probabilities, weighted by the true class labels, and is typically used with class indices as labels (not one-hot encoded).

Optimizer

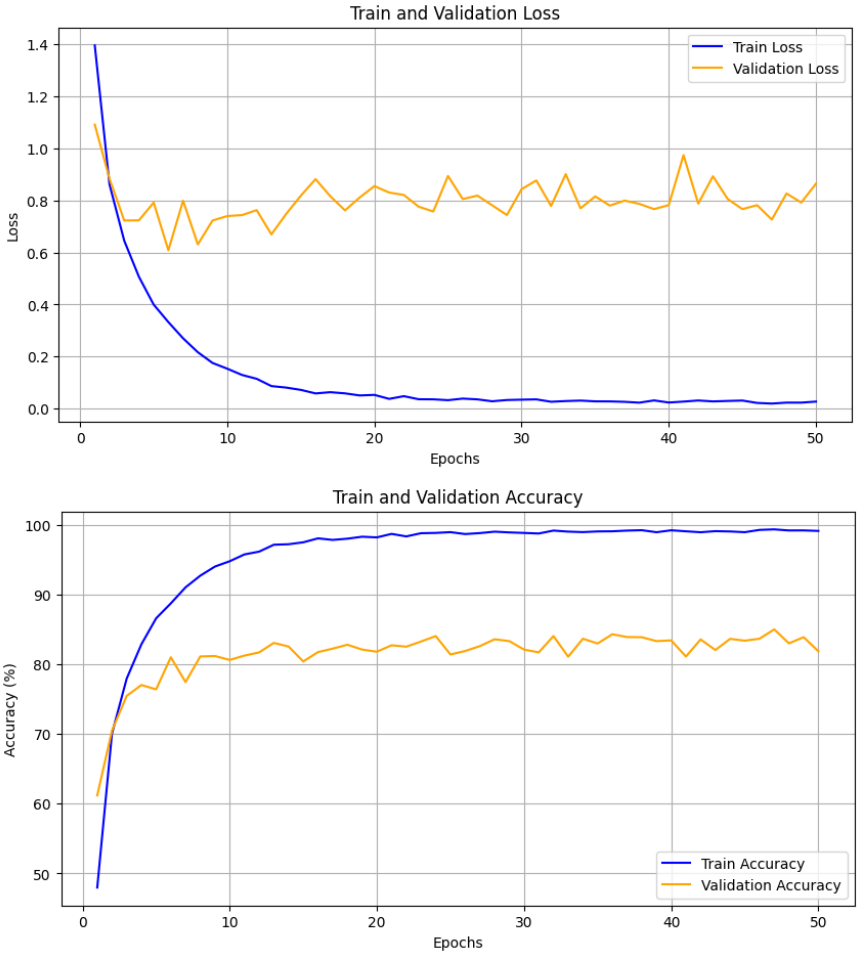
Stochastic Gradient Descent (SGD) is a widely used optimization algorithm that updates model parameters based on the gradient of the loss function using a random subset of the dataset at each step. Key components of SGD include the learning rate, which controls the step size of updates, momentum, which helps accelerate the process in the relevant direction and dampen oscillations by incorporating previous updates, and weight decay, which is an L2 regularization technique that penalizes large weights to prevent overfitting. The weight update rule combines these components to improve the optimization process.

In the VGG16 model, weight decay (also referred to as L2 regularization) is applied with a parameter value of 0.0005. This penalty term added to the loss function discourages the model from assigning excessively large values to its weights, helping to prevent overfitting, especially when training on a large dataset with many parameters. The small value (0.0005) is chosen to ensure that regularization doesn't dominate the model's training, allowing the network to still learn complex patterns, but with some constraint on the magnitude of the weights.

After defining several functions to train our model, we begin the learning process with 50 epochs. The accuracy for epochs is as follows:

```
Epoch 46/50
Train Loss: 0.0221, Train Accuracy: 99.29%
Validation Loss: 0.7818, Validation Accuracy: 83.68%
Epoch 47/50
Train Loss: 0.0197, Train Accuracy: 99.37%
Validation Loss: 0.7266, Validation Accuracy: 85.02%
Epoch 48/50
Train Loss: 0.0234, Train Accuracy: 99.21%
Validation Loss: 0.8266, Validation Accuracy: 83.02%
Epoch 49/50
Train Loss: 0.0233, Train Accuracy: 99.22%
Validation Loss: 0.7920, Validation Accuracy: 83.90%
Epoch 50/50
Train Loss: 0.0273, Train Accuracy: 99.14%
Validation Loss: 0.8643, Validation Accuracy: 81.88%
```

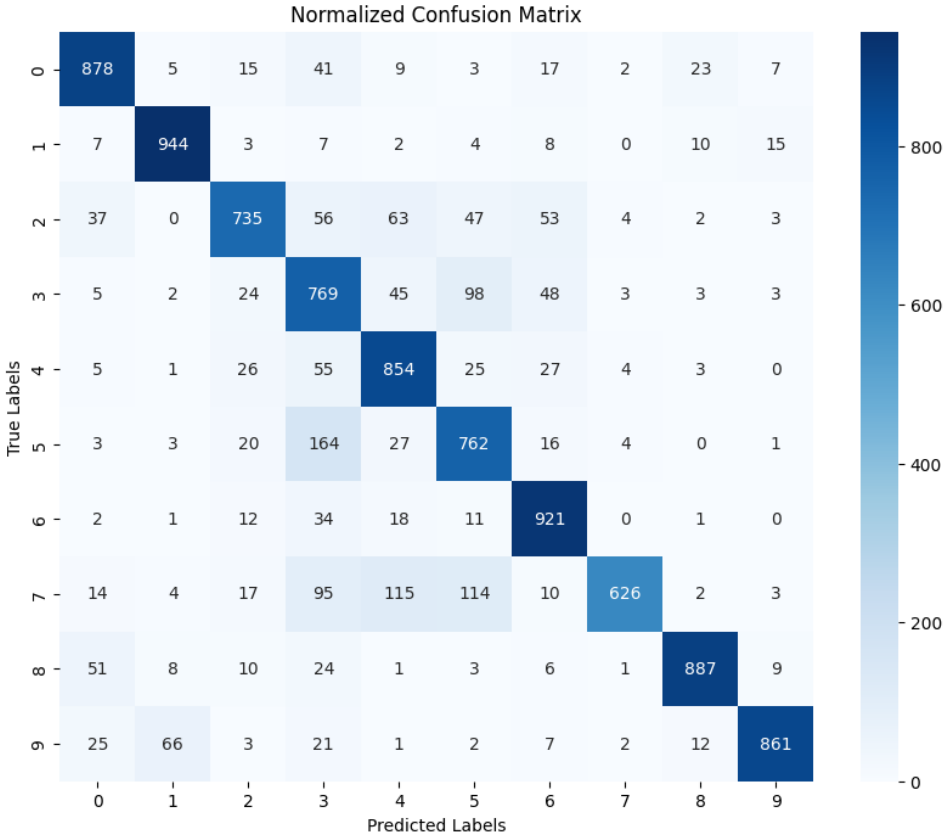
Additionally, the changes in loss and accuracy are reported in the graph below:



The result of test dataset is:

```
Test Loss: 0.8151
Test Accuracy: 82.37%
```

And the confusion matrix is:



Conclusion

In Convolutional Neural Networks (CNNs), the convolutional layers play a fundamental role in extracting features from input data, typically images. These layers use small filters (kernels) applied locally to the image, extracting various features such as edges, textures, and shapes. The features obtained in the convolutional layers are usually low-level features, so in the early layers, simpler features like edges and colors are identified, and in deeper layers, more complex features like objects and abstract patterns are recognized.

After the convolutional layers, fully connected layers are used to convert the extracted features into a final prediction. Fully connected layers can combine all the features extracted by the convolutional layers and extract more complex information from these features. These layers perform the classification task, where the model decides which category the image belongs to. Since the convolutional layers only learn local features of the image, the fully connected layers are used to learn global and abstract patterns, ultimately producing the final result.

Fully connected layers in neural networks, unlike convolutional layers, are not aware of the spatial structure of the data. They process inputs independently and do not pay attention to the location of features or local patterns in an image. This makes them unable to recognize important features like edges and textures, which are crucial in image processing. Furthermore, fully connected layers consume significant resources due to the large number of parameters and computational complexity, leading to slower training processes.

While convolutional layers have the ability to generalize features to different scales and positions in an image, fully connected layers lack this capability. Convolutional layers apply shared filters across the entire image and learn similar features at different locations, reducing dependence on the position of the features. This improves the model's performance and efficiency in feature recognition, whereas fully connected layers are less effective at extracting features from images.