Subject: Image Processing and Computer Vision - II Laboratory (DJ19DSL702)

AY: 2022-23

Experiment 1

(Image Classification using CNN)

Aim: Apply CNN to develop a Computer Vision model to classify images containing cancer cells.

Theory:

1. Introduction

Image classification is a fundamental task in computer vision, which involves assigning labels or categories to images based on their visual content. Convolutional Neural Networks (CNNs) have revolutionized image classification by enabling the automatic extraction of relevant features directly from the raw pixel data. In this lab, you will learn how to implement image classification using CNNs and gain hands-on experience in training and evaluating a CNN model.

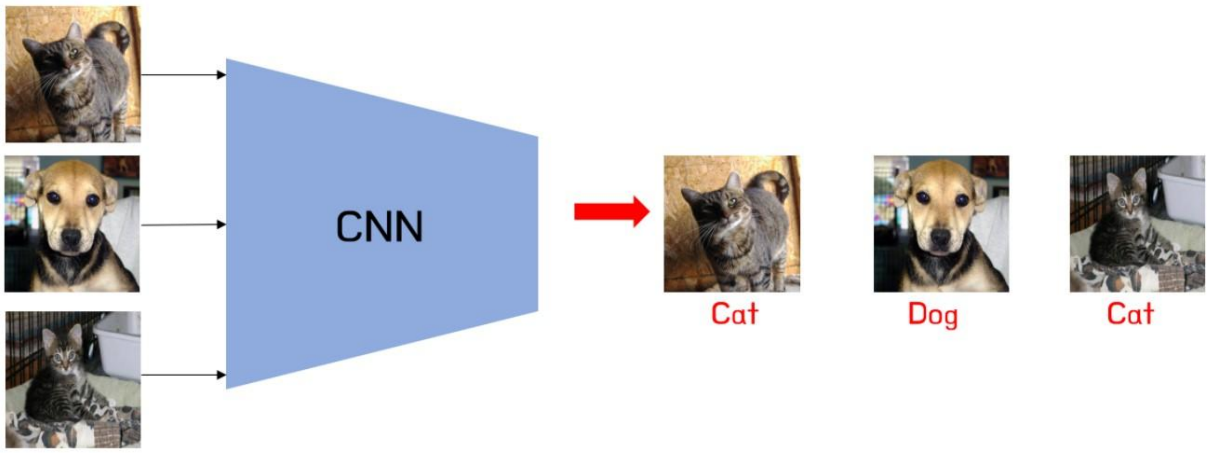
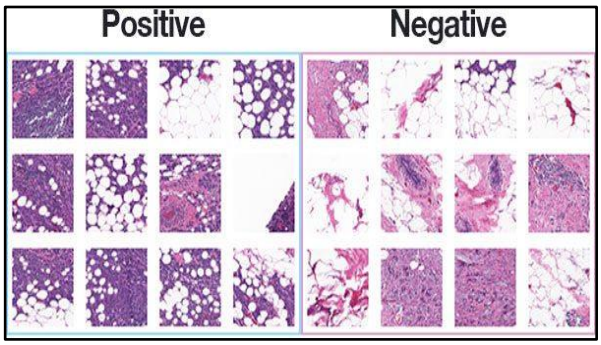


Fig 1. (a) Image Classification



(b)

(b) Classification of cancer cells into benign/ malignant(cancerous).

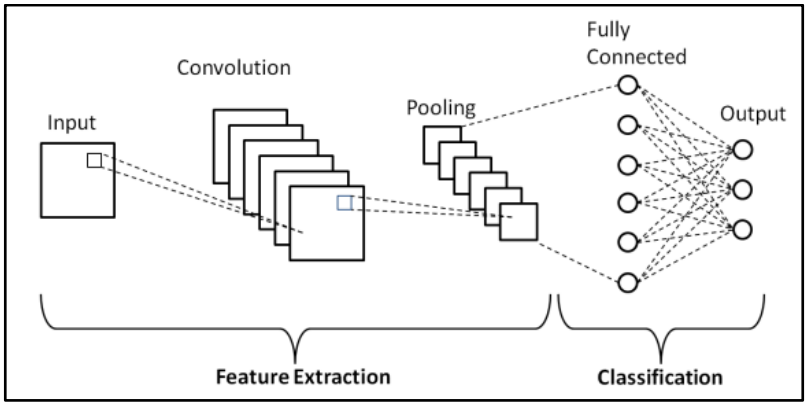


Figure 2. Basic CNN architecture overview

2. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are a specialized type of neural network architecture designed for processing grid-like data, such as images. CNNs are inspired by the organization of the visual cortex in the human brain. They consist of three main components:

2.1 Convolutional Layers

Convolutional layers are responsible for extracting local features from the input image. Each layer consists of multiple filters (also known as kernels) that slide over the input image, performing element-wise multiplication and summation operations to produce feature maps. These feature maps capture spatial hierarchies of features.

2.2 Activation Functions

Activation functions introduce non-linearities into the CNN architecture, allowing the model to learn complex relationships between the input image and the corresponding class labels. Common activation functions used in CNNs include Rectified Linear Unit (ReLU), sigmoid, and Softmax.

2.3 Pooling Layers

Pooling layers downsample the feature maps obtained from the convolutional layers, reducing the spatial dimensions while retaining the most salient information. Max pooling is a commonly used pooling technique, which selects the maximum value from a neighborhood window.

3. CNN Architecture for Image Classification

In the context of image classification, a typical CNN architecture consists of multiple convolutional layers followed by pooling layers, eventually leading to one or more fully connected layers and a final output layer. The output layer applies a softmax activation function to produce class probabilities.

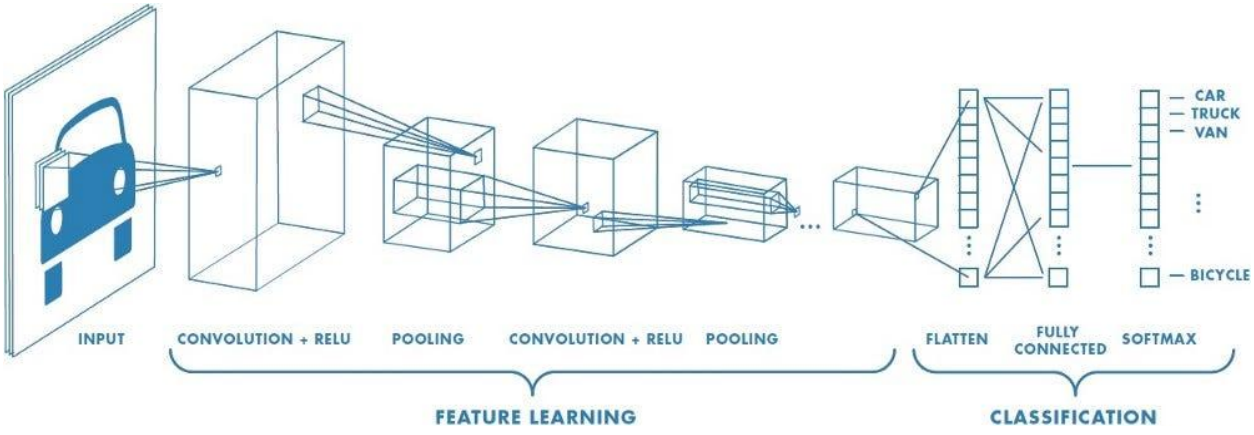


Figure 2. Traditional CNN architecture

4. Dataset Preparation

To perform image classification experiments, it is essential to have a labeled dataset. Choose an appropriate dataset that suits your experiment's requirements, such as CIFAR-10, ImageNet, or a custom dataset. Ensure that the dataset is split into training, validation (optional), and testing subsets.

5. Building and Training a CNN Model

In this lab, you will use a deep learning framework (e.g., TensorFlow, PyTorch, Keras) to implement a CNN model for image classification. The steps involved are as follows:

5.1 Model Architecture Design

Define the architecture of your CNN model by specifying the number and type of layers, their connectivity, and the activation functions to be used. Consider the depth, width, and complexity of the model based on the dataset size and complexity.

5.2 Model Compilation

Compile the model by specifying the optimizer, loss function, and evaluation metrics. The choice of loss function depends on the classification problem, such as categorical cross-entropy for multi-class classification.

5.3 Model Training

Train the CNN model using the training dataset. Perform forward and backward passes to compute gradients and update the model's weights iteratively. Monitor the training process by observing the loss and accuracy metrics on the training and validation datasets.

5.4 Hyperparameter Tuning

Experiment with different hyperparameters, such as learning rate, batch size, number of epochs, and network architecture, to optimize the model's performance. (Ideally use the validation dataset to choose the best set of hyperparameters.)
We experiment with the number of epochs, loss function, early stopping callback and optimizer.

- ➔ Epochs: The model was trained for 4 epochs initially, allowing it to iterate over the entire dataset multiple times and improve its performance through successive iterations. It was then tuned to 2 epochs to test the convergence of the model and analyze the change. Changing epochs is a

common practice of hyperparameter tuning since it generally affects the model performance and convergence.

- ➔ Callbacks (EarlyStopping): The 'EarlyStopping' callback was utilized to monitor the validation loss during training. Deep learning models are prone to overfitting, where they perform well on the training data but fail to generalize to unseen data. EarlyStopping helps to avoid overfitting by monitoring the model's performance on a validation set. When the validation performance stops improving, EarlyStopping interrupts the training process, preventing the model from overfitting to the training data.
- ➔ Loss (CategoricalCrossentropy): The model is initially tested using the 'BinaryCrossentropy' since it is a binary classification problem. The 'CategoricalCrossentropy' loss function was also tested for the classification task (which is generally used for multiclass classification problems). It measured the dissimilarity between the predicted and actual class distributions, guiding the model to minimize the loss and make accurate predictions.
- ➔ Optimizer (RMSprop): The optimizer is responsible for updating the model's parameters during the training process, aiming to minimize the loss function and improve the model's accuracy. Initially, the Adam (Adaptive Moment Estimation) optimizer is used. It is an adaptive learning rate optimizer that combines the advantages of RMSprop and momentum. It dynamically adjusts the learning rate for each parameter during training. The 'RMSprop' optimizer was then chosen to update the model's parameters based on the gradients calculated during training. With a learning rate of 0.001, it adjusted the weights and biases to minimize the loss function and optimize the model's performance. RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimizer that divides the learning rate for a parameter by the square root of the moving average of the squared gradients for that parameter.

  For more information: https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be#

6. Model Evaluation and Testing

Once the training is complete, evaluate the trained CNN model using the testing dataset. Compute the accuracy, loss to assess the model's performance.

7. Fine-tuning and Transfer Learning

To improve the classification accuracy further, explore the concept of fine-tuning and transfer learning. Fine-tuning involves retraining only specific layers of a pre-trained CNN model on a new dataset, whereas transfer learning utilizes the knowledge gained from a pre-trained model to solve a related task.

Apply transfer learning using ResNet, MobileNet pre-trained models

8. Conclusion

In this lab, you have learned the fundamentals of image classification using Convolutional Neural Networks. You have gained hands-on experience in building, training, and evaluating a CNN model. With this knowledge, you can now apply CNNs to various image classification tasks and explore advanced techniques in computer vision and deep learning.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:

Dataset: https://drive.google.com/drive/folders/1a2FwghL4oIocb1kzSU-cwWp8c3EKePAU?usp=sharing

This dataset contains Malaria cell images classified into “Paratisized” and “Uninfected.”

The steps involved in the experiment are as follows:

a. Preparing the dataset: Download and preprocess the dataset. Resize the images and perform one-hot encoding.

b. Designing the model: Design the CNN model.

c. Training the model: Initialize the Custom CNN and transfer learning models (with pre-trained weights if available) and fine-tune them using the training dataset. Define the loss function and optimization algorithm. Monitor the training process and adjust 3-4 hyperparameters.

d. Post-processing and evaluation: After training, apply the trained model to test images. Calculate evaluation metrics such as to assess the model's performance.