AJEENKYA D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY
School of Engineering

A

Report of Certification for MLA-I

on

"Breast Cancer Detection Using Convolutional Neural Network"

submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Technology

by

Shivam Kaushik

URN NO: 2021-B-16092003C

Under the Guidance of

Prof. Balwant Shrikrishna S.

April 2024

School of Engineering

Ajeenkya D Y Patil University, Pune

AJEENKYA D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY
School of Engineering

Apr 2024

CERTIFICATE

This is to certify that the report entitled **"Breast Cancer Detection Using Convolutional Neural Network"** is a bonafide work of "**Shivam Kaushik" (URN No. 2021-B-16092003C)** submitted to the Ajeenkya D Y Patil University, Pune in partial fulfillment of the requirement for the award of the degree of "***Bachelor of Technology (B.Tech) of Information Technology (Data Science)***".

Prof. Balwant Shrikrishna S.

Internal-Examiner

INDEX

SR NO.	TITLE	PAGE
1	CHAPTER 1: Need for Breast Cancer Detection	3
2	CHAPTER 2: Convolutional Neural Network	4
3	CHAPTER 3: Classification Using CNN	5
4	CHAPTER 4: Sequential Neural Network	7
5	CHAPTER 5: RELU and Sigmoid Function	8
6	CHAPTER 6: Adam Algorithm	10
7	CHAPTER 7: Binary Crossentropy	11
8	CHAPTER 8: accuracy_score	13
9	CHAPTER 9: CODE	14

CHAPTER 1

Need For Breast Cancer Detection

Breast cancer detection is a critical area of medical research and practice, given the significant impact it has on public health worldwide. Early detection is key to improving treatment outcomes and reducing mortality rates associated with breast cancer. Conventional screening methods like mammography have been effective but can sometimes yield false positives or miss subtle signs of cancer. This has led to a growing need for more advanced and reliable detection techniques, and deep learning has emerged as a promising solution.

Deep learning algorithms, particularly Convolutional Neural Networks (CNNs), excel at extracting complex patterns and features from medical imaging data. By analyzing mammography images, these algorithms can identify subtle abnormalities indicative of breast cancer with high accuracy. Moreover, deep learning models can continuously learn and improve from large datasets, potentially enhancing their performance over time.

The integration of deep learning into breast cancer detection holds immense promise for revolutionizing screening programs and improving patient outcomes. These models have the potential to assist radiologists in interpreting mammography images more accurately and efficiently, leading to earlier detection of cancerous lesions. Ultimately, this could result in reduced morbidity and mortality rates associated with breast cancer.

In conclusion, the need for breast cancer detection using deep learning is paramount in advancing medical diagnostics and improving public health outcomes. By leveraging the power of artificial intelligence, we can enhance the effectiveness of screening programs and empower healthcare providers to make more informed decisions, ultimately saving lives.

CHAPTER 2

Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing and analyzing visual data such as images. They have revolutionized various fields, including computer vision, image recognition, and medical imaging. Here's how CNNs work and why they're crucial:

1. **Convolutional Layers**: CNNs are comprised of multiple layers, including convolutional layers. These layers apply convolution operations to input images, using learnable filters or kernels to extract features such as edges, textures, and patterns.

2. **Pooling Layers**: After convolution, pooling layers are often employed to downsample the feature maps, reducing their spatial dimensions while retaining important information. This helps in making the network more computationally efficient and invariant to small spatial translations in the input.

3. **Activation Functions**: Non-linear activation functions like ReLU (Rectified Linear Unit) are applied to introduce non-linearity into the network, enabling it to learn complex relationships between features.

4. **Fully Connected Layers**: Following the convolutional and pooling layers, CNNs often end with one or more fully connected layers, which perform classification based on the features extracted in earlier layers. These layers use techniques like softmax to generate probabilities for each class in a classification task.

5. **Training**: CNNs are trained using a supervised learning approach, where they are fed input images along with their corresponding labels. During training, the network adjusts its parameters (weights and biases) through backpropagation and gradient descent, minimizing a chosen loss function such as categorical cross-entropy.

6. **Transfer Learning**: CNNs can leverage pre-trained models on large datasets like ImageNet and then fine-tune them on specific tasks with smaller datasets. This approach, known as transfer learning, is particularly useful when limited labeled data is available for training.

7. **Applications**: CNNs have found applications in various domains, including image classification, object detection, facial recognition, medical image analysis, and more. In the context of medical imaging, CNNs are used for tasks such as disease detection, tumor segmentation, and treatment planning.

Overall, CNNs have significantly advanced the field of computer vision and have become indispensable tools in a wide range of applications, including those requiring the analysis of visual data like images and videos. Their ability to automatically learn hierarchical representations of data has made them particularly effective for tasks involving complex visual patterns.

CHAPTER 3

Classification Using Convolutional Neural Network

Classification using Convolutional Neural Networks (CNNs) is a powerful application of deep learning, particularly in the field of computer vision. Here's how CNNs are used for classification tasks:

1. Data Preparation: The first step is to gather and preprocess the data. For image classification tasks, this involves collecting a dataset of labeled images. Preprocessing may include resizing, normalization, and augmentation to ensure uniformity and improve generalization.

2. Model Architecture: A CNN architecture is designed based on the complexity of the classification task and the characteristics of the input data. Common architectures like VGG, ResNet, and Inception are often used as starting points. These architectures typically consist of multiple convolutional layers, pooling layers, and fully connected layers.

3. Training: The CNN model is trained on the labeled dataset using a supervised learning approach. During training, the model learns to extract relevant features from the input images and map them to their corresponding labels. This is achieved through forward propagation, where input images pass through the layers of the network, and backpropagation, where the error is calculated and used to update the model's parameters (weights and biases).

4. Loss Function: A loss function is used to measure the difference between the predicted labels and the ground truth labels. Common loss functions for classification tasks include categorical cross-entropy and softmax cross-entropy.

5. Optimization: An optimization algorithm such as stochastic gradient descent (SGD), Adam, or RMSprop is used to minimize the loss function and update the model's parameters iteratively.

6. Validation: Throughout the training process, the performance of the model is evaluated on a separate validation dataset to monitor its generalization ability and prevent overfitting.

7. Testing: Once training is complete, the trained model is evaluated on a separate test dataset to assess its performance on unseen data. This provides an estimate of the model's accuracy and effectiveness in real-world scenarios.

Types of Classification CNN Models

LeNet-5:

Explanation: LeNet-5, developed by Yann LeCun in the 1990s, is one of the earliest CNN architectures. It consists of convolutional layers followed by subsampling layers (pooling), and fully connected layers. LeNet-5 was primarily designed for handwritten digit recognition tasks.

AlexNet:

Explanation: AlexNet, proposed by Alex Krizhevsky et al. in 2012, gained significant attention for its success in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It consists of eight layers, including convolutional layers with large filter sizes, max-pooling layers, and fully connected layers with dropout regularization. AlexNet popularized the use of deep CNNs for image classification tasks.

VGGNet (Visual Geometry Group Network):

Explanation: VGGNet, introduced by the Visual Geometry Group at the University of Oxford in 2014, is known for its simplicity and uniform architecture. It consists of multiple convolutional layers with small 3x3 filters and max-pooling layers. VGGNet is available in

several variants with different depths (e.g., VGG16, VGG19), where the number indicates the total number of layers.

GoogLeNet/Inception:

Explanation: GoogLeNet, also known as Inception v1, was developed by Google researchers in 2014. It introduced the concept of inception modules, which use multiple filter sizes within the same layer to capture features at different scales efficiently. GoogLeNet aims to increase both depth and width of the network without significantly increasing computational cost.

ResNet (Residual Network):

Explanation: ResNet, proposed by Kaiming He et al. in 2015, addresses the problem of vanishing gradients in very deep networks. It introduces residual connections, or skip connections, which allow the network to learn residual mappings instead of directly learning the desired mappings. This enables the training of extremely deep networks (e.g., ResNet50, ResNet101) with improved accuracy.

DenseNet (Densely Connected Convolutional Network):

Explanation: DenseNet, introduced by Gao Huang et al. in 2017, extends the idea of skip connections in ResNet by densely connecting each layer to every other layer in a feed-forward fashion. This facilitates feature reuse and encourages feature propagation throughout the network, leading to improved parameter efficiency and feature extraction.

MobileNet:

Explanation: MobileNet, developed by Google in 2017, is designed for mobile and embedded vision applications with limited computational resources. It employs depthwise separable convolutions, which split standard convolutions into depthwise and pointwise convolutions, reducing the computational cost while maintaining accuracy. MobileNet models are lightweight and suitable for deployment on mobile devices.

These are just a few examples of classification CNN models, each with its own strengths and characteristics suited for different applications and computational constraints. Researchers and practitioners often choose models based on factors such as performance, model size, computational efficiency, and available resources.

CHAPTER 4

Sequential Neural Network

Sequential Neural Network refers to a specific type of neural network architecture where the layers are arranged sequentially, one after the other. This architecture is commonly used in deep learning frameworks like Keras and TensorFlow to create models for various machine learning tasks, including classification, regression, and sequence prediction.

Here's a breakdown of the key components and characteristics of a Sequential Neural Network:

1. Layer-by-Layer Structure: In a Sequential Neural Network, each layer is stacked sequentially, with the output of one layer serving as the input to the next layer. The sequential arrangement allows for the easy construction of feedforward neural networks.

2. Input Layer: The first layer of the network is the input layer, which receives the raw input data. Depending on the nature of the task, this layer may consist of neurons equal to the number of input features.

3. Hidden Layers: Between the input and output layers, there can be one or more hidden layers, where the network learns to extract hierarchical representations of the input data. These hidden layers typically consist of densely connected (fully connected) layers, convolutional layers, recurrent layers, or a combination thereof, depending on the architecture of the network.

4. Activation Functions: Each layer (except the output layer) is followed by an activation function, which introduces non-linearity into the network and enables it to learn complex relationships in the data. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, tanh, and softmax.

5. Output Layer: The final layer of the network is the output layer, which produces the predictions or outputs of the model. The structure of the output layer depends on the nature of the task:

6. For binary classification tasks, a single neuron with a sigmoid activation function is typically used.

7. For multi-class classification tasks, the output layer often consists of multiple neurons, one for each class, with a softmax activation function.

8. For regression tasks, the output layer may consist of a single neuron without an activation function or a linear activation function.

9. Loss Function and Optimization: During training, the network is trained to minimize a predefined loss function, which measures the difference between the predicted outputs and the true labels. Common loss functions include mean squared error (MSE) for regression tasks and categorical cross-entropy for classification tasks. Optimization algorithms such as stochastic gradient descent (SGD), Adam, or RMSprop are used to update the network's parameters (weights and biases) to minimize the loss function.

Overall, Sequential Neural Networks provide a straightforward and flexible framework for building and training neural network models for a wide range of machine learning tasks.

CHAPTER 5

RELU Activation and Sigmoid Function

ReLU (Rectified Linear Unit) and the sigmoid function are both activation functions commonly used in neural networks, each with its own characteristics and applications:

ReLU (Rectified Linear Unit):

Function: ReLU activation function is defined as

$f(x)=\max(0,x)$

where x is the input to the function.

Characteristics:

ReLU is simple and computationally efficient, as it only involves a thresholding operation.

It introduces non-linearity to the network, allowing it to learn complex patterns and representations in the data.

ReLU has been widely adopted in deep learning due to its effectiveness in training deep neural networks and mitigating the vanishing gradient problem.

However, ReLU can suffer from the "dying ReLU" problem, where neurons can become inactive (outputting zero) during training and fail to recover. This can slow down or hinder the training process.

Applications: ReLU is commonly used as the activation function in hidden layers of deep neural networks for tasks such as image classification, object detection, and natural language processing.

Sigmoid Function:

Function: The sigmoid function, also known as the logistic function, is defined as

$f(x)$**= 1/1+(e^(-x))**

Characteristics:

1. The sigmoid function maps the input to a value between 0 and 1, making it suitable for binary classification tasks where the output represents probabilities.

2. It has a smooth, S-shaped curve, which allows for gradient-based optimization techniques like backpropagation to be applied effectively.

3. However, the sigmoid function saturates for large positive or negative inputs, leading to vanishing gradients during training (especially in deep networks). This can slow down the convergence of the learning process.

4. The output of the sigmoid function can be interpreted as the probability of the input belonging to the positive class in binary classification tasks.

Applications: The sigmoid function is commonly used as the activation function in the output layer of binary classification neural networks, where it produces probabilities indicating the likelihood of a sample belonging to one class.

In summary, ReLU is widely used as an activation function in hidden layers of deep neural networks due to its simplicity and effectiveness in training, while the sigmoid function is commonly used in the output layer of binary classification networks for producing probability outputs. Each activation function has its advantages and limitations, and the choice depends on the specific requirements of the task and the characteristics of the data.

CHAPTER 6

Adam Algorithm

The Adam optimizer is an adaptive optimization algorithm commonly used in training neural networks. It stands for "Adaptive Moment Estimation" and combines the advantages of two other popular optimization algorithms: AdaGrad and RMSprop. The Adam optimizer was introduced by Diederik P. Kingma and Jimmy Ba in their paper titled "Adam: A Method for Stochastic Optimization" in 2014.

Here's how the Adam optimizer works and its key characteristics:

1. Adaptive Learning Rates: Adam adjusts the learning rate for each parameter individually based on estimates of the first and second moments of the gradients. This adaptivity allows for faster convergence and more stable training compared to fixed learning rate methods.

2. Momentum Optimization: Like RMSprop, Adam incorporates the concept of momentum, which helps accelerate the optimization process by accumulating past gradients. This momentum term smoothens the optimization trajectory and helps overcome local minima.

3. Bias Correction: Adam performs bias correction to account for the initialization bias and reduce the impact of the initial learning rate on the optimization process, especially during the early stages of training.

4. Parameter Updates: The update rule for parameters θ in Adam is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Where:

- η is the learning rate.
- $P_m = K_W \cdot \Delta_W [W]$ is the estimate of the first moment (mean) of the gradients.
- $\hat{v}_t$ is the estimate of the second moment (uncentered variance) of the gradients.
- ϵ is a small constant (usually on the order of 10^{-8}) to prevent division by zero.
- t is the current iteration.

5. Hyperparameters: Adam has hyperparameters that need to be set, including the learning rate η, the exponential decay rates for the moment estimates (β_1 for the first moment and β_2 for the second moment), and the small constant ϵ.

6. Advantages: Adam is known for its efficiency in practice and is widely used for training deep neural networks. It combines the benefits of adaptive learning rates and momentum optimization, making it well-suited for a variety of optimization tasks.

Overall, the Adam optimizer is a popular choice for training neural networks due to its adaptive nature, efficiency, and effectiveness in handling various optimization challenges. However, it may require careful tuning of hyperparameters for optimal performance in specific tasks and datasets.

CHAPTER 7

Binary Crossentropy

Binary crossentropy loss, also known as binary log loss or logistic loss, is a commonly used loss function in binary classification tasks. It measures the dissimilarity between the true labels and the predicted probabilities for binary classification problems, where each sample belongs to one of two classes (e.g., positive or negative, 1 or 0).

Here's how binary crossentropy loss is calculated and its key characteristics:

1. Definition: The binary crossentropy loss $L(y, \hat{y})$ for a single sample is defined as:

$$L(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

Where:

- y is the true label (0 or 1).
- $\hat{y}$ is the predicted probability that the sample belongs to class 1 (the positive class).
- $\log$ denotes the natural logarithm.

2. Interpretation:

- The binary crossentropy loss penalizes the model more heavily when it makes confident incorrect predictions (i.e., assigning high probability to the wrong class).
- It encourages the model to output high probabilities for the correct class (close to 1 for positive samples and close to 0 for negative samples) to minimize the loss.

3. Properties:

- The binary crossentropy loss is differentiable, making it suitable for training neural networks using gradient-based optimization algorithms such as stochastic gradient descent (SGD) or Adam.

- It is convex in nature, facilitating optimization and convergence to a global minimum during training.

- The loss function is symmetric with respect to the predicted probabilities, penalizing both overestimation and underestimation of the true class probability.

4. Applications:

- Binary crossentropy loss is commonly used as the loss function in the output layer of neural networks for binary classification tasks, such as spam detection, sentiment analysis, and medical diagnosis.

- It is particularly effective when dealing with imbalanced datasets, where one class is significantly more prevalent than the other, as it treats each sample equally regardless of class frequency.

Overall, binary crossentropy loss is a fundamental and widely used loss function in binary classification problems due to its simplicity, effectiveness, and suitability for training neural networks. By minimizing this loss during training, models can learn to make accurate predictions and effectively distinguish between the two classes.

CHAPTER 8

accuracy_score

‘accuracy_score’ is a function commonly used in machine learning for evaluating the performance of classification models. It measures the proportion of correctly classified instances out of the total number of instances in the dataset. Here's a detailed explanation of ‘**accuracy_score’**:

1. **Purpose**: The primary purpose of accuracy_score is to quantify the overall correctness of a classification model's predictions. It provides a simple and intuitive metric to assess how well the model performs on the given dataset.

2. **Inputs**: **accuracy_score** takes two parameters:

- **y_true**: The true labels or target values of the dataset. These are the actual classes to which each instance in the dataset belongs.
- **y_pred**: The predicted labels or target values generated by the classification model. These are the classes predicted by the model for each instance in the dataset.

3. **Calculation**: The accuracy score is calculated by comparing the true labels (**y_true**) with the predicted labels (**y_pred**) and counting the number of instances where they match. The accuracy is then computed as the ratio of the number of correctly classified instances to the total number of instances in the dataset:
Accuracy= Total Number of Predictions / Number of Correct Predictions

4. **Output**: The output of accuracy_score is a single scalar value representing the accuracy of the model on the given dataset. The accuracy score ranges from 0 to 1, where 1 indicates perfect classification (all predictions are correct), and 0 indicates no correct predictions.

5. **Usage**: accuracy_score is commonly used in conjunction with model evaluation in machine learning workflows. After training a classification model and making predictions on a test dataset, you would typically use accuracy_score to quantify how well the model performs on the test data.

6. **Considerations**: While accuracy is a useful metric for evaluating classification models, it may not always provide a complete picture, especially in the presence of class imbalance or misclassification costs.