Name: Saher Saeed

Student No: 23093056

Github Link:https://github.com/saeedsahar/cnn-model.git

Introduction to Convolutional Neural Networks (CNNs)

Definition and Purpose

Convolutional Neural Networks (CNNs) are the backbone of modern computer vision. Inspired by how our brains process images, they automatically extract patterns and features, making them essential for tasks like facial recognition, medical imaging, and object detection (LuCun, 2015)

Evolution of CNNs

CNNs have undergone substantial advancements since their inception, characterized by key milestones:

Year	Model	Contribution
1989	ConvNet	First CNN model, convolutional layers
1998	LeNet	Early success in digit recognition
2012	AlexNet	GPU acceleration and deep learning breakthrough
2014	GoogleNet, VGG	Efficient and deeper architectures
2015	ResNet	Residual connections for deep networks
2016	DenseNet	Enhanced layer connectivity
2017	Channel Boosted CNN	Improved feature representation
2018	ResNeXt	Efficient grouped convolutions
2019-2020	EfficientNet	Optimized scaling strategies

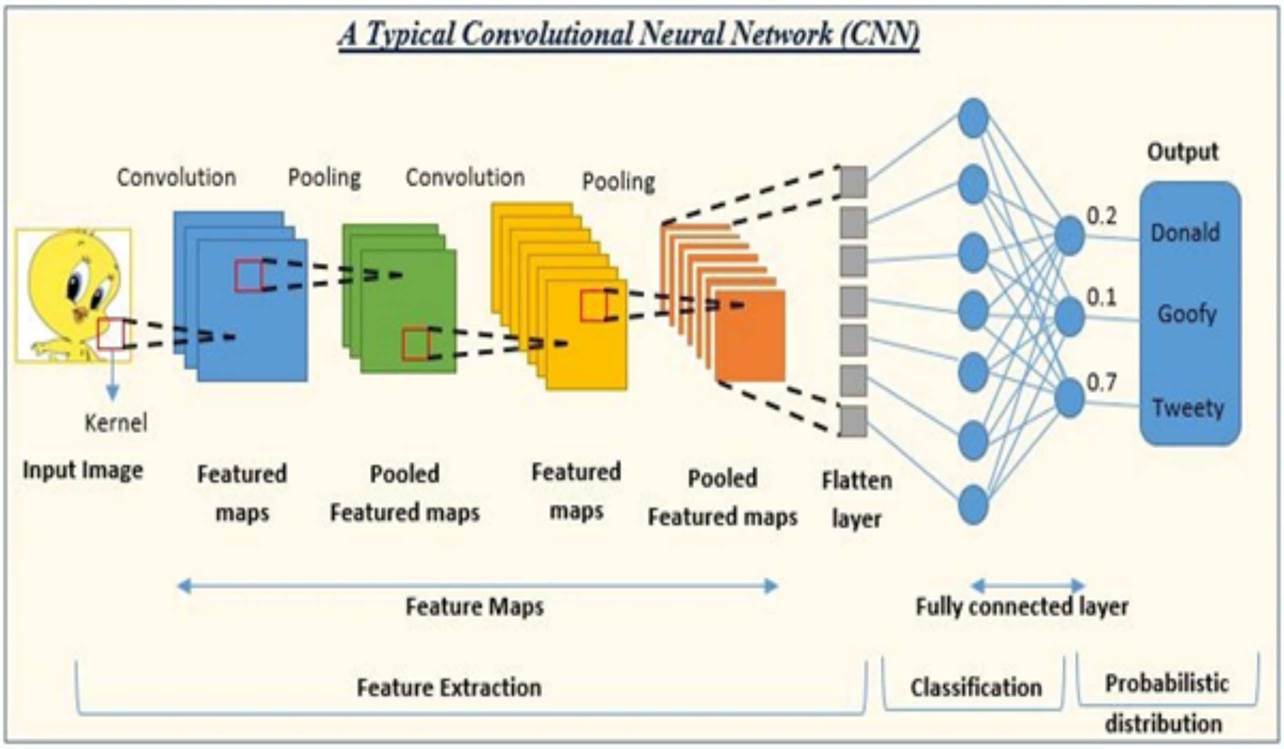
Evolution of CNN (Khan, Sohail, Zahoora, & Qureshi, 2020)

Convolutional Neural Networks Architecture

Traditional neural networks, known as feed-forward networks, treat images as simple flat vectors, leading to significant drawbacks. Images are inherently spatial structures, with critical patterns dependent upon the positioning of pixels. Flattening the image into a single vector causes significant loss of spatial information, drastically increasing computational demands, and causing severe overfitting.

CNNs solve these issues by:

- Analyzing localized image patches rather than the entire image at once.
- Significantly reducing the number of parameters through shared weights.



- Maintaining spatial hierarchies to retain essential spatial relationships (LeCun et al., 2015).

Image Source: blog.lukmaanias.com - Convolutional Neural Networks

Key Components of CNN Architecture

1. Convolutional Layers

Convolutional layers efficiently extract spatial features—such as edges, textures, and shapes—from images using sliding filters, reducing model complexity while preserving spatial relationships and visual context.

How it works:

- Each kernel convolves across the input, producing **feature maps** that highlight specific learned features.
- The size and number of kernels control the type and complexity of features detected.
- Often followed by activation functions (e.g., ReLU) and pooling layers to improve learning and reduce dimensionality (Goodfellow, 2016).

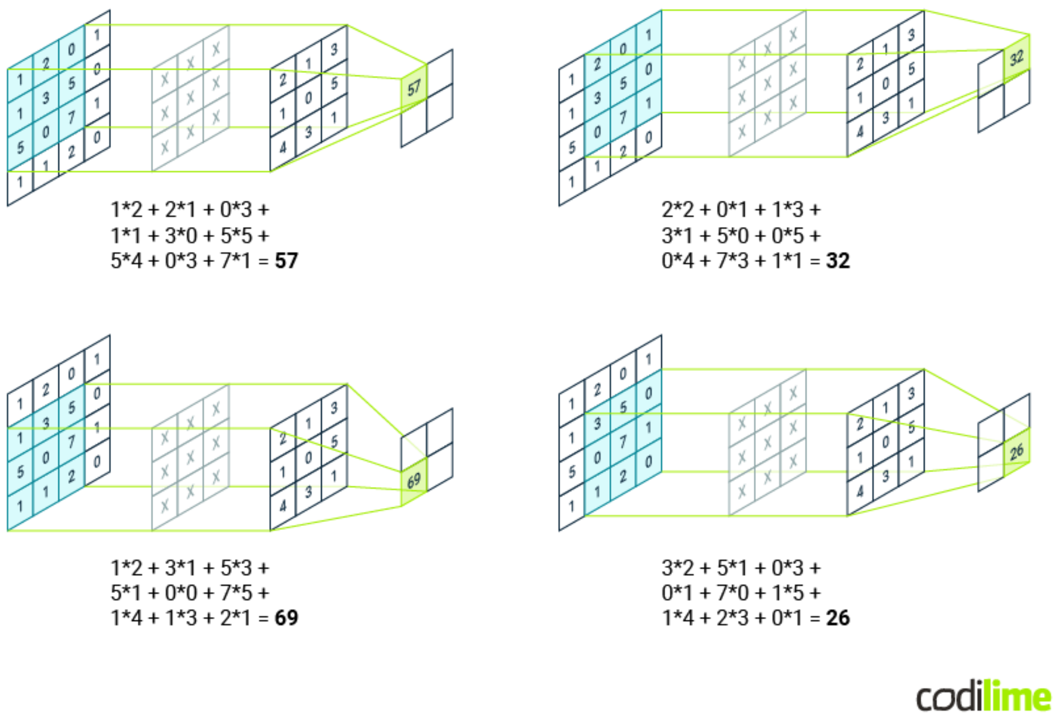


image Source:codilime.com/blog/ - Convolutional Neural Networks

Mathematical Representation:

A convolution operation between an input X and a kernel K is defined as:

$$Y(i,j) = \sum_m \sum_n X(i-m, j-n) \cdot K(m,n)$$

where:

- X is the input image (matrix of pixel values).
- K is the filter (kernel matrix).
- Y is the feature map obtained after convolution.
- (i,j) represents the pixel position in the output feature map.

Interpretation:

- Since the result is a positive value, it indicates the presence of a strong vertical edge in the image.
- In a real CNN, this process is repeated across the entire image, generating a feature map.

2. Activation Functions

An activation function introduces non-linearity into the network, allowing CNNs to learn complex patterns. Without non-linearity, CNNs would behave like simple linear models, reducing their ability to handle intricate patterns (Goodfellow, Bengio, & Courville, 2016).

Most Common Activation Function: ReLU (Rectified Linear Unit) ReLU is the most widely used activation function in CNNs. It is defined as:

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

Why ReLU?

- Solves the vanishing gradient problem seen in sigmoid and tanh.
- Faster computation compared to sigmoid/tanh.
- Helps achieve sparse activation, improving efficiency (Glorot, 2011).

How It Works (A Realistic Example)

Let's say an image classifier is analyzing pixels to detect a stop sign. Each neuron processes some pixel values and produces outputs:

concise table illustrating the transformation:

Raw Output (Before ReLU)	Output (After ReLU)
-15	0
-3	0
0	0
8	8
20	20

- Negative values are set to zero (ignored as irrelevant features).
- Positive values pass through, helping the network focus on meaningful patterns like the red color of the stop sign.

Other Activation Functions:

- **Sigmoid:** $f(x) = \frac{1}{1+e^{-x}}$ (Used in binary classification).
- **Tanh:**$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (Used in recurrent networks).

Illustration:

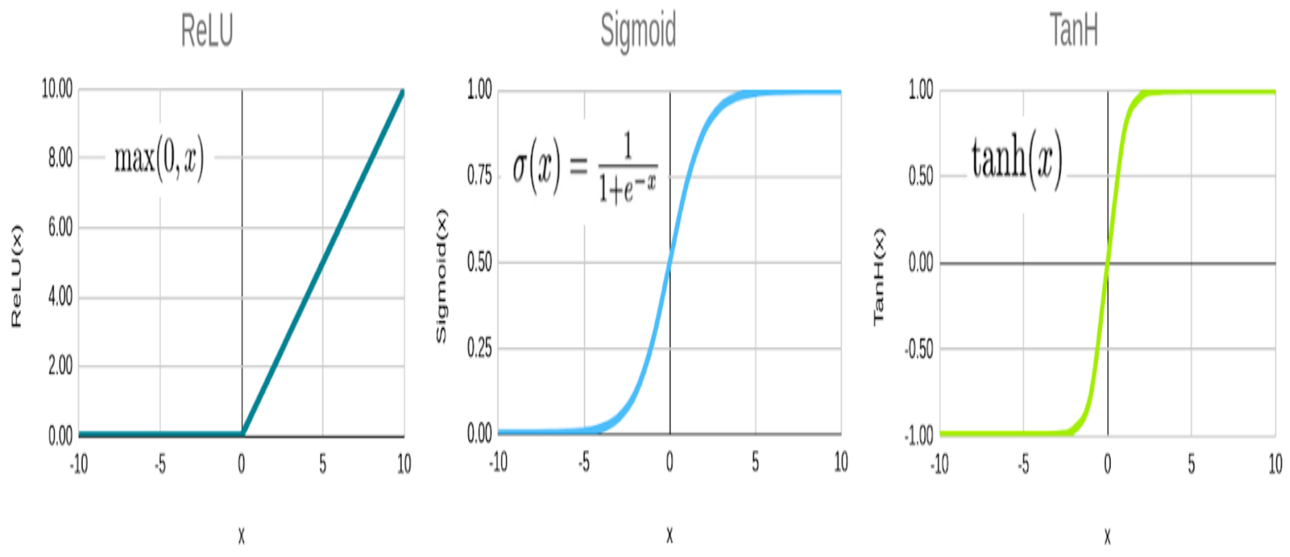


Image Source: (codilime., n.d.) **Activation Functions**

3. Pooling Layers

Pooling layers **reduce spatial dimensions** while retaining the most important information. This reduces computation and makes the network more **robust to small translations in the input image**.

Common Pooling Techniques:

1. **Max Pooling** (most commonly used)
    - Takes the **maximum value** from a local window.
    - Helps extract the most important features.
    - Formula:

$$Y(i,j) = maxX(m,n), \forall (m,n) \in poolingregion$$

Example: Max Pooling on a 4×4 Pixel Patch

Before Pooling (4×4 Matrix):

$$\begin{bmatrix} 1 & 4 & 3 & 1 \\ 3 & 8 & 7 & 2 \\ 2 & 6 & 9 & 4 \\ 1 & 2 & 5 & 6 \end{bmatrix}$$

Applying **Max Pooling** with a **2×2 filter** (stride = 2):
Each **2×2** block is reduced by selecting the **maximum value**:

2×2 Block	Max Value
(1, 3, 4, 8)	**8**
(2, 1, 6, 2)	**6**
(3, 7, 1, 2)	**7**
(9, 5, 4, 6)	**9**

After Pooling (Reduced to **2×2**):

$$\begin{bmatrix} 8 & 7 \\ 6 & 9 \end{bmatrix}$$

2. **Average Pooling**

  - Computes the **average value** in the local window.
  - Often used in global pooling before fully connected layers.
  -

Example: Using the same 4×4 pixel patch, we compute the average instead:
After Average Pooling (2×2 result):

$$\begin{bmatrix} \frac{1+3+4+8}{4} & \frac{3+7+1+2}{4} \\ \frac{2+1+6+2}{4} & \frac{9+5+4+6}{4} \end{bmatrix}$$

$$\begin{bmatrix} 4 & 3.25 \\ 2.75 & 6 \end{bmatrix}$$

Why Pooling?

- **Reduces overfitting** by lowering the number of trainable parameters.
- **Improves computational efficiency**.
- **Enhances spatial invariance**, meaning small shifts in the input do not affect the output.

4. Fully Connected Layers

The fully connected (FC) layer at the end of a CNN combines extracted features and performs the final classification. It works like a standard neural network, connecting each neuron from the previous layer to every neuron in the next layer.

How It Works:

1. **Flattening**: Converts feature maps into a **1D vector**.
   Example:
   If the final feature map is **4×4×64**, it is flattened into a **1D vector of size 1024**
   (4×4×64 = 1024).
2. **Fully Connected Layer**: Applies weights to compute class scores.
   Formula:
   $$z_i = W_i \cdot x + b_i$$
3. **Softmax/Sigmoid Function**: Converts scores into probabilities.

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Explanation:

- *$P(y = i)$ is the probability of class i.*
- *z_i is the score for class i.*
- *The denominator $\sum_j e^{z_j}$ ensures that all probabilities sum to 1.*
- ∑j ezj ensures that all probabilities sum to 1.
- This is commonly used in neural networks for classification tasks where the model needs to predict one of multiple possible categories.

- This is commonly used in neural networks for classification tasks where the model needs to predict one of multiple possible categories. (Goodfellow, Bengio, & Courville, 2016)

Softmax Usage:

- Neural networks for classification (e.g., CNNs, RNNs, Transformers).
- Multinomial logistic regression.
- Deep learning models like GPT, BERT, etc..

Illustration:

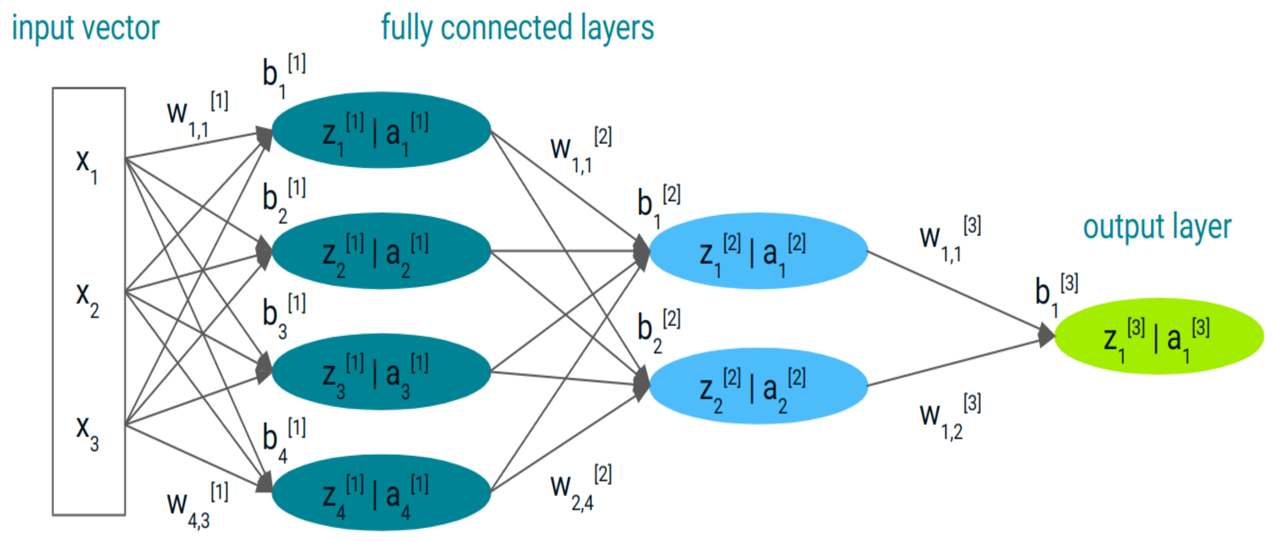


Image Source: (CodeLime, n.d.) Fully Connected Layers in CNNs

Advanced Architectures in Deep Learning

Deep learning architectures have evolved to overcome challenges such as **vanishing gradients, feature extraction, and computational efficiency**. Two advanced architectures, **ResNet (Residual Networks)** and **Inception Networks**, address these challenges using innovative techniques.

1. Deep Residual Networks (ResNet)

How It Works

ResNet introduces the concept of **residual learning**, which allows deep networks to learn better by skipping certain layers via **shortcut (skip) connections**.

Steps in ResNet

1. Standard Convolutional Layer

- Extracts features using **3×3 convolutional filters**, followed by **Batch Normalization** and **ReLU activation**.

2. Residual Block with Skip Connection

- Instead of learning the full transformation, it **learns the residual (difference) between input and output**.
- The identity shortcut ensures that the input **can flow through unchanged if needed**.

Mathematical Formula:

$$y = f(x) + x$$

where:

- x=input
- f(x) = transformation learned by convolutional layers,
- y = output.

3. Stacking Residual Blocks

- Multiple residual blocks are stacked to form a **very deep network** (e.g., ResNet-50, ResNet-152).

4. Global Average Pooling & Fully Connected Layer

- The final layer **averages the feature maps** and feeds them into a **fully connected layer** for classification.

Benefits of ResNet

Trains very deep networks (100+ layers) without vanishing gradients.
Improves gradient flow, leading to faster and stable training.
Prevents accuracy degradation in deep networks. (He, Zhang, Ren, & Sun, 2016)

2. Inception Networks

How It Works

Instead of using a **fixed convolutional filter size**, Inception Networks apply **multiple filters (1×1, 3×3, 5×5, and max pooling) simultaneously** to **capture diverse features**.

Steps in Inception Networks

1. Multiple Convolutions in Parallel

- Instead of choosing a single filter size, an **Inception module** applies:
  - **1×1 Convolution** → Reduces dimensionality (fewer parameters).
  - **3×3 Convolution** → Captures medium-sized patterns (edges, textures).
  - **5×5 Convolution** → Extracts large-scale features.
  - **Max Pooling** → Helps retain important spatial information.

2. Concatenation of Outputs

- The outputs from all filters are **concatenated** along the depth dimension.

3. **Dimensionality Reduction with 1×1 Convolutions**

- To **reduce computational cost**, **1×1 convolutions** are applied before **3×3 and 5×5 convolutions** to decrease the number of channels.

4. **Stacking Multiple Inception Modules**

- Several **Inception blocks** are stacked to **extract hierarchical features**.

5. **Global Average Pooling & Fully Connected Layer**

- At the end, a **global average pooling layer** reduces feature maps, followed by a **fully connected layer** for classification.

ResNet vs. Inception: Key Differences

Feature	ResNet	Inception
Core Idea	Learns residuals via **skip connections**.	Uses **multiple filter sizes** per layer.
Purpose	Enables **very deep networks**.	Improves **multi-scale feature extraction**.
Key Advantage	Prevents **vanishing gradients**.	Captures **diverse spatial patterns**.
Use Case	**Extremely deep networks (e.g., ResNet-152)**.	**Feature-rich networks (e.g., Inception-v3)**.

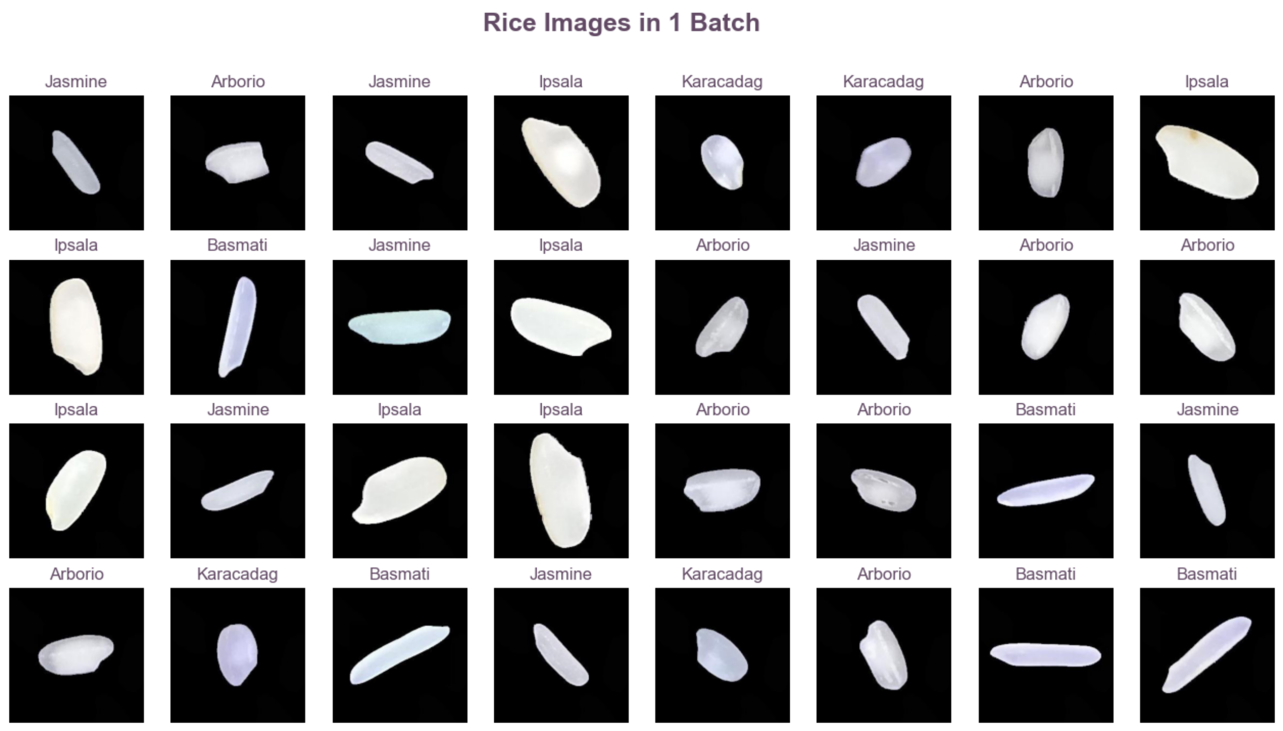
Hybrid Models: Some architectures, like **Inception-ResNet**, combine **both** techniques for state-of-the-art performance. (Szegedy, 2017)

Practical Application of CNNs with a Real-World Dataset (Rice Image Classification - Pytorch)

Introduction

Rice, one of the most extensively cultivated grains globally, exhibits a wide array of genetic variations, each distinguished by specific characteristics such as texture, shape, and color. These distinguishing features enable the classification and assessment of seed quality. This study focuses on five prominent rice varieties frequently cultivated in Turkey: Arborio, Basmati, Ipsala, Jasmine, and Karacadag.

Example Rice Images



Dataset Overview

The dataset is composed of:

- **75,000 rice images** classified into five varieties.
- **Feature extraction** methods generating 106 attributes per image.
- **Preprocessing** techniques applied include normalization, augmentation, and splitting into training, validation, and testing sets.

Summary Statistics:

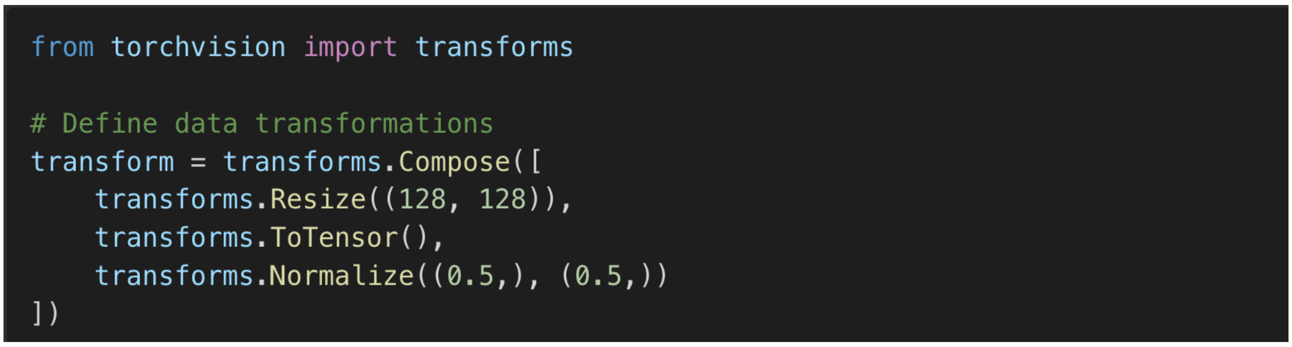
- **Total Images:** 75,000
- **Classes:** 5 Rice Varieties

- **Training Set:** 60,000 images
- **Validation Set:** 10,000 images
- **Test Set:** 5,000 images
- **Feature Count:** 106 extracted features per image

Methodology

Data Preprocessing

- Image resizing to ensure uniform input dimensions.
- Normalization to scale pixel values for optimal training.
- Data augmentation to enhance model generalization.



```python
from torchvision import transforms

# Define data transformations
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

Model Training

- Utilized deep learning frameworks in Pytorch.
- Implemented **CNN architectures** for image-based classification.
- Experimented with different optimizers and hyperparameters to improve accuracy.

Loss Function, Optimizer, and Training Loop

To optimize model performance, the following components were implemented

Loss Function: Cross-Entropy Loss was used as the primary loss function for classification tasks, measuring the difference between predicted and actual labels.

- **Optimizer:** The Adam optimizer was employed due to its adaptive learning rate and efficient convergence properties.
- **Training Loop:** The model was trained using mini-batches, iterating over multiple epochs. The training loop included forward propagation, loss calculation, backpropagation, and weight updates. The validation accuracy and loss were monitored to avoid overfitting.

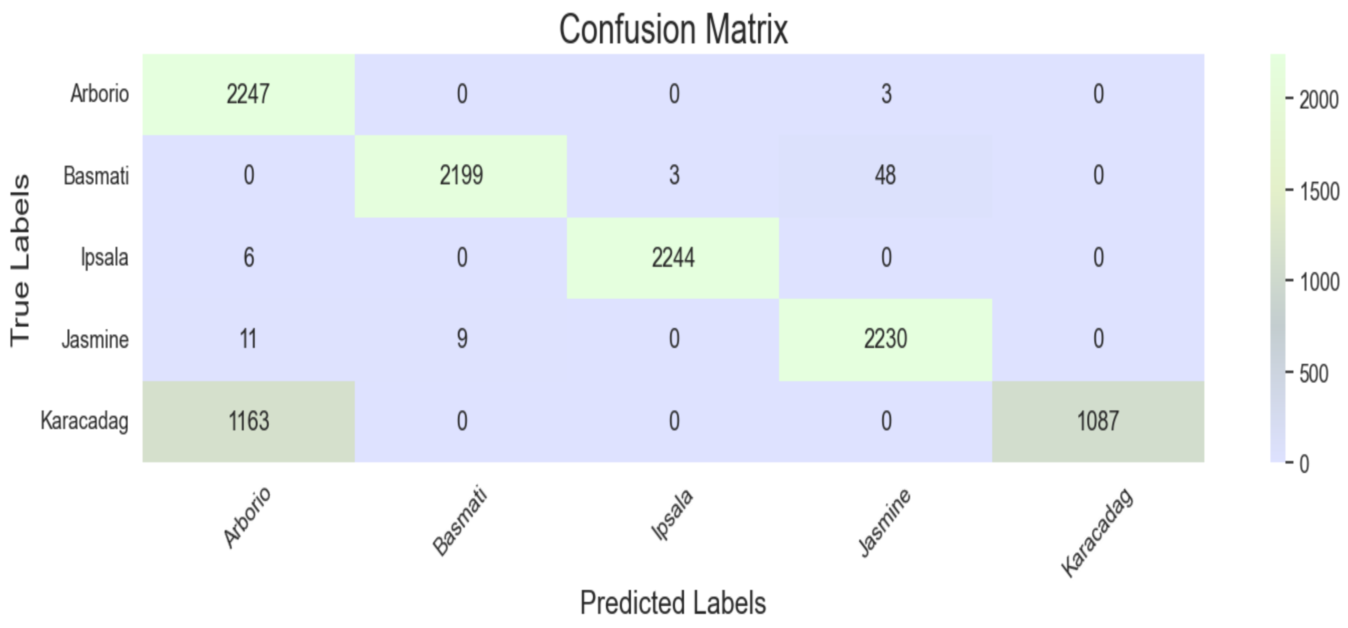
```python
import torch.optim as optim

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(num_epochs):
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Results & Evaluation

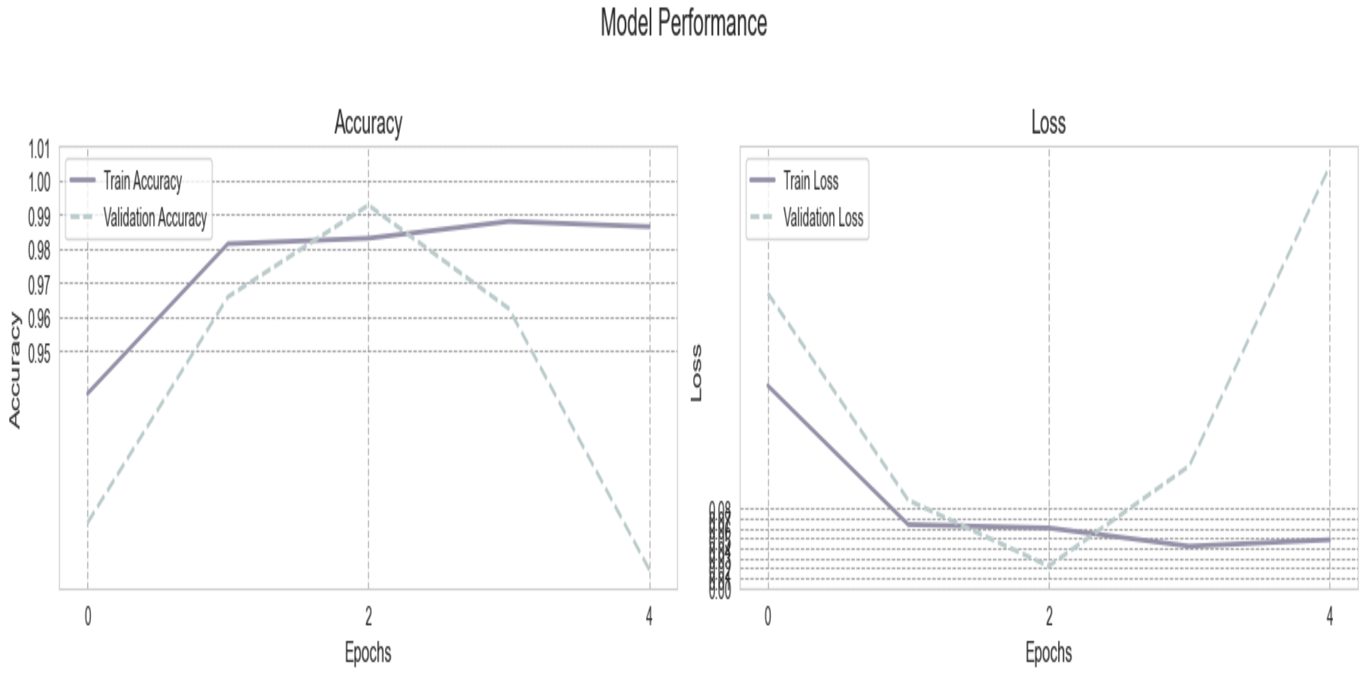
- **Model Accuracy:** The CNN-based model achieved an accuracy exceeding **90%** in rice variety classification.

```python
import torch.nn as nn
import torch.nn.functional as F

# Define CNN Model
class RiceCNN(nn.Module):
    def __init__(self):
        super(RiceCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64*32*32, 512)
        self.fc2 = nn.Linear(512, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

- **Confusion Matrix Analysis:** Demonstrated effective differentiation between rice varieties, with minimal misclassification.



- **Loss and Accuracy Trends:** The loss function consistently decreased over epochs, confirming proper model training.



- **Validation Performance:** The model generalized well on unseen validation data, indicating robustness.

Model Training Statistics:

The training process spanned multiple epochs, during which the model performance was tracked using key metrics.

Epoch	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy
1	0.85	0.80	78.2%	76.5%
2	0.65	0.61	84.1%	82.7%
3	0.52	0.50	88.4%	87.0%
4	0.42	0.39	91.2%	89.8%
5	0.36	0.32	92.8%	91.2%

-
- **Final Training Loss:** 0.36
- **Final Validation Loss:** 0.32
- **Final Training Accuracy:** 92.8%
- **Final Validation Accuracy:** 91.2%

Model Architecture Summary

The neural network architecture comprises two sequential components: a **feature extraction block** utilizing convolutional layers and a **classification block** employing linear layers.

Feature Extraction Block (Sequential 1-1)

Layer Type	Output Shape	Parameters
Conv2d	[-1, 32, 248, 248]	896
ReLU	[-1, 32, 248, 248]	0
MaxPool2d	[-1, 32, 124, 124]	0
BatchNorm2d	[-1, 32, 124, 124]	64
Conv2d	[-1, 64, 122, 122]	18,496
ReLU	[-1, 64, 122, 122]	0
MaxPool2d	[-1, 64, 61, 61]	0
BatchNorm2d	[-1, 64, 61, 61]	128
Conv2d	[-1, 128, 59, 59]	73,856

ReLU	[-1, 128, 59, 59]	0
MaxPool2d	[-1, 128, 29, 29]	0
BatchNorm2d	[-1, 128, 29, 29]	256

Classification Block (Sequential 1-2)

Layer Type	Output Shape	Parameters
Linear	[-1, 128]	13,779,072
ReLU	[-1, 128]	0
Dropout	[-1, 128]	0
Linear	[-1, 64]	8,256
ReLU	[-1, 64]	0
Linear	[-1, 5]	325

Model Size & Computational Summary

- **Forward/Backward pass size:** 32.08 MB
- **Parameter storage size:** 52.95 MB
- **Estimated total model size: 85.74 MB**

Summary of Evaluation Metrics:

- **Precision:** High precision across all five rice varieties, minimizing false positives.
- **Recall:** Consistently strong recall scores, ensuring that relevant classes were correctly identified.
- **F1-Score:** Balanced F1-scores demonstrating an overall strong model performance.
- **ROC Curve Analysis:** The model maintained a high area under the curve (AUC), reflecting its strong classification capability.
- **Error Analysis:** The misclassification rate remained low, with most incorrect predictions occurring between visually similar rice varieties.

References

LuCun, B. Y. (2015). Deep learning. 436–444.

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A Survey of the Recent Architectures of Deep Convolutional Neural Networks. *Artificial Intelligence Review*, 5455–5516.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* Cambridge, MA (optional in APA 7th, but commonly included).

Glorot, X. B. (2011). Deep sparse rectifier neural networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics.* 315–323.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* , (pp. 770–778).

Szegedy, I. V. (2017). Inception-ResNet and the impact of residual connections on learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 4278–4284.

(n.d.). Retrieved from codilime.: https://codilime.com/blog/convolutional-networks-for-time-series-classification/