CNN Vignette Primary Document

Kaitlyn, Oscar, Nini, Johanna

2025-12-03

Convolutional Neural Network Vignette

Introduction

Convolutional Neural Networks (CNNs) are a special type of neural networks designed for processing grid-structured data, particularly images. Unlike traditional fully-connected neural networks, CNNs use the spatial structure of images through localized connectivity and parameter sharing. This vignette explores CNN process and performance through implementation of a multi-class butterfly species classifier.

Limitations of Fully Connected NNs

Traditional neural networks face significant challenges when processing image data. For a 128×128 pixel RGB image (which is what our butterfly image dataset consists of), the input dimensionality is 49,152 (128 × 128 × 3). A single fully-connected hidden layer with 256 neurons requires 12,583,936 parameters. This parameter explosion leads to computational inefficiency and increases the risk of overfitting. Additionally, fully-connected networks lack translation equivariance. Identical patterns appearing in different spatial locations have to be learned independently, requiring substantially more training data to achieve comparable performance to CNNs.

CNN Architecture Principles

CNNs address these limitations through two fundamental design principles:

Parameter Sharing:

Small learnable filters, called kernels, are applied across the entire input via convolution operations. The same filter weights are used at every spatial location, drastically reducing parameter count while enabling position-invariant feature detection.

Hierarchical Feature Learning:

Stacked convolutional layers create hierarchical representations. Initial layers extract low-level features (edges, textures), intermediate layers identify mid-level patterns (shapes, object parts), and deep layers recognize high-level semantic concepts.

CNN Layer Components

Convolutional Layers

Convolutional layers apply learnable filters through discrete convolution operations. Each filter performs element-wise multiplication with the receptive field and sums the results, producing activation maps that indicate feature presence at different spatial locations.

Activation Functions

Non-linear activation functions, typically ReLU (Rectified Linear Unit: f(x) = max(0,x)), are applied element-wise following convolution operations. Non-linearity is essential for learning complex mappings beyond linear transformations.

Pooling Layers

Pooling operations perform spatial downsampling. Max pooling selects the maximum value within each pooling window, reducing spatial dimensions while retaining salient features. This provides computational efficiency, partial translation invariance, and regularization.

Flatten Layers

Flatten operations reshape multi-dimensional feature maps into one-dimensional vectors, enabling transition from convolutional feature extraction to fully-connected classification layers.

Fully-Connected Layers

Dense layers with full connectivity between consecutive layers aggregate extracted features for final classification. These layers function identically to traditional neural network layers.

Dropout

Dropout applies stochastic regularization by randomly deactivating neurons during training with specified probability. This prevents co-adaptation of features and reduces overfitting.

Output Layer

The final layer uses softmax activation to produce normalized probability distributions over classes:

Dataset Description

The dataset comprises butterfly images spanning 75 species classes. The training set contains 6,499 images, split into 5,199 training samples and 1,300 validation samples (80/20 ratio). The test set contains 2,786 images. All images are resized to 128×128 pixels and processed in batches of 32. Stratified sampling ensures proportional class representation across training and validation splits. This is critical for maintaining class balance in multi-class problems with potentially uneven class distributions.

Model Architecture

The implemented CNN follows a sequential architecture:

Input Layer: 128×128×3 (RGB images)

Convolutional Block 1: 32 filters (3×3), ReLU activation, followed by 2×2 max pooling

Convolutional Block 2: 64 filters (3×3), ReLU activation, followed by 2×2 max pooling

Convolutional Block 3: 128 filters (3×3), ReLU activation, followed by 2×2 max pooling

Flatten Layer: Converts 3D feature maps to 1D vector

Dense Layer: 256 units, ReLU activation Dropout Layer: 50% dropout rate

Output Layer: 75 units (one per class), softmax activation

The model contains 6,535,307 trainable parameters.

Depth Analysis

An experiment examining network depth trained models with 1, 2, 3, and 4 convolutional blocks:

1 Block: Validation accuracy of 31.6%. Insufficient representational capacity for complex multi-class classification.

2 Blocks: Validation accuracy of 55.7%. Additional depth enables more discriminative feature learning.

3 Blocks: Validation accuracy of 67.2%. Three-layer hierarchy provides adequate feature abstraction for this task.

These results demonstrate that network depth directly impacts feature abstraction capability. However, excessive depth may lead to overfitting or training difficulties without sufficient data or regularization.

Data Augmentation

Data augmentation applies random transformations to training images:

- Rotation: ±15 degrees
- Width shift: 10%
- Height shift: 10%
- Horizontal flip

Augmentation increases effective training set size and improves model generalization by exposing varied presentations of each class. Augmentation is applied only to training data; validation and test sets remain unmodified for accurate performance assessment.

Training Results

The model was trained for 10 epochs using the Adam optimizer and categorical cross-entropy loss. Training progression showed:

- Epoch 1: 17.8% validation accuracy
- Epoch 10: 67.2% validation accuracy

Training accuracy reached approximately 88-89% for the 2-block model, indicating some degree of overfitting. The training-validation accuracy gap suggests potential benefit from additional regularization or larger training datasets.

Prediction Pipeline

Test set predictions follow this procedure:

1. Forward pass through trained model
2. Softmax output produces probability distribution over 75 classes
3. Class with maximum probability selected via argmax operation
4. Numeric class indices mapped to species labels
5. Results exported to CSV with filename and predicted species

Conclusion

CNNs provide substantial advantages over fully-connected architectures for image classification tasks through parameter sharing and spatial structure exploitation. The butterfly classifier achieved 67.2% accuracy across 75 classes, demonstrating effective feature learning from training data. Network depth proved critical for performance, with deeper architectures enabling more sophisticated feature hierarchies. Further improvements could be obtained through transfer learning, additional regularization techniques, or larger training datasets.

Code Summary

Data Loading

```
#train_df = pd.read_csv(TRAIN_CSV)
#test_df  = pd.read_csv(TEST_CSV)
```

Here, we loaded the training and testing CSV files that contained the image filenames and their labels.

Data Preprocessing

```
#train_df_split, val_df_split = train_test_split(
#    train_df, test_size=0.2, stratify=train_df['class'], random_state=123
#)
```

Here, we split the training data set into 80% training and 20% validation sets.

Layers