1. Convolutional Neural Network (CNN) for Fashion-MNIST Classification

1.1. Introduction

Image classification is a fundamental task in computer vision. Convolutional Neural Networks (CNNs) have proven to be highly effective for image recognition due to their ability to automatically learn spatial features from images.

In this lab, a CNN is implemented using the KNIME Analytics Platform to classify images from the Fashion-MNIST dataset. This dataset contains grayscale images of clothing items belonging to ten different categories.

The objective of this work is to design, train, and evaluate a CNN model, analyze its performance using accuracy curves and a confusion matrix, and experiment with architectural improvements such as dropout and batch normalization.

1.2. Dataset Description and Exploration

1.2.1. Fashion-MNIST Dataset

The Fashion-MNIST dataset is a widely used benchmark dataset for image classification. It consists of grayscale images representing various clothing items.

- Number of images: 70,000
- Image resolution: 28 × 28 pixels
- Color format: Grayscale
- Number of classes: 10

The dataset includes the following classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

1.2.2. Loading the Dataset in KNIME

The dataset is loaded into KNIME using image processing nodes. Images are read directly from disk and converted into KNIME image cells. Class labels are extracted from file paths using string manipulation techniques.

The following KNIME nodes are used:
- List Files/Folders
- Path to String
- Image Reader (Table)
- String Manipulation

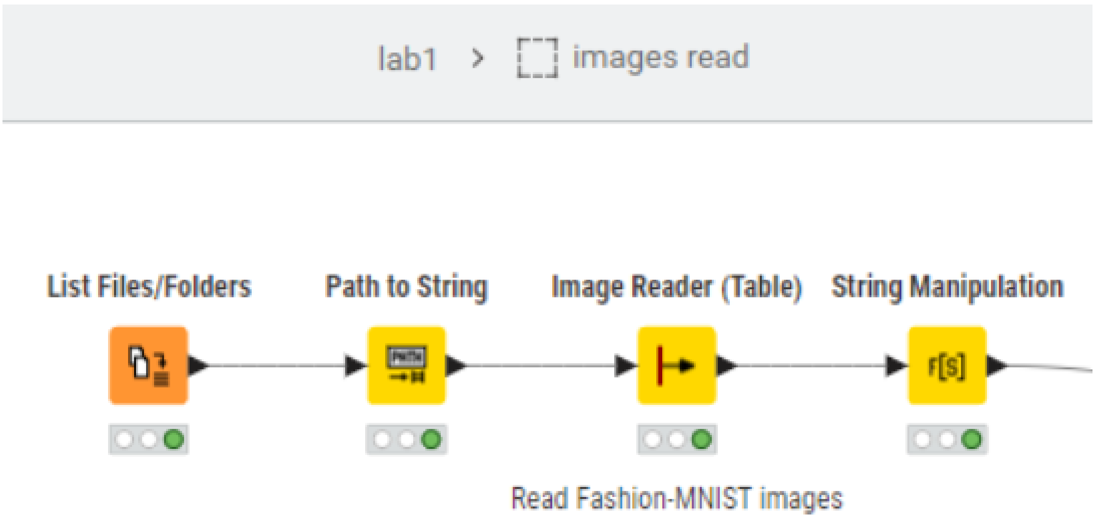
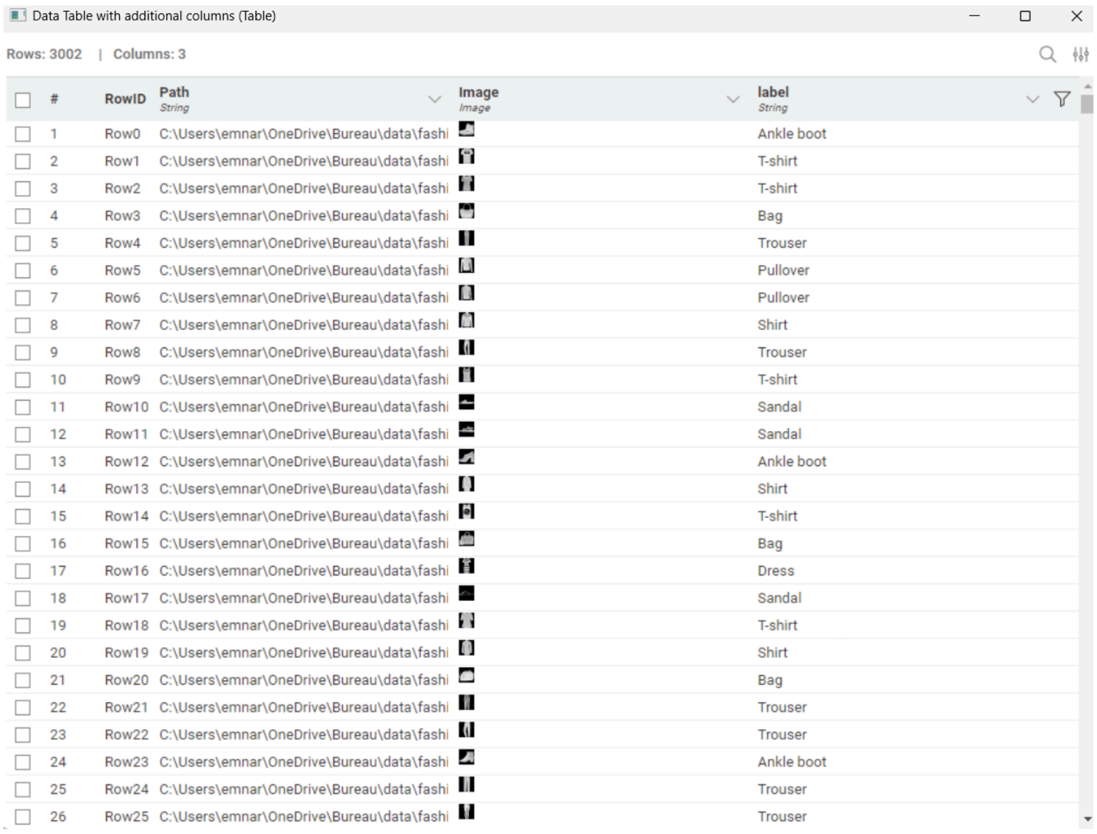


Figure 1: KNIME workflow for loading Fashion-MNIST images

1.2.3. Sample Visualization

Sample images from different categories are visualized to verify data integrity and label correctness. This step helps ensure that images and labels are correctly aligned before training.


Data Table with additional columns (Table)

Rows: 3002 | Columns: 3

#	RowID	Path (String)	Image (Image)	label (String)
1	Row0	C:\Users\emnar\OneDrive\Bureau\data\fashi		Ankle boot
2	Row1	C:\Users\emnar\OneDrive\Bureau\data\fashi		T-shirt
3	Row2	C:\Users\emnar\OneDrive\Bureau\data\fashi		T-shirt
4	Row3	C:\Users\emnar\OneDrive\Bureau\data\fashi		Bag
5	Row4	C:\Users\emnar\OneDrive\Bureau\data\fashi		Trouser
6	Row5	C:\Users\emnar\OneDrive\Bureau\data\fashi		Pullover
7	Row6	C:\Users\emnar\OneDrive\Bureau\data\fashi		Pullover
8	Row7	C:\Users\emnar\OneDrive\Bureau\data\fashi		Shirt
9	Row8	C:\Users\emnar\OneDrive\Bureau\data\fashi		Trouser
10	Row9	C:\Users\emnar\OneDrive\Bureau\data\fashi		T-shirt
11	Row10	C:\Users\emnar\OneDrive\Bureau\data\fashi		Sandal
12	Row11	C:\Users\emnar\OneDrive\Bureau\data\fashi		Sandal
13	Row12	C:\Users\emnar\OneDrive\Bureau\data\fashi		Ankle boot
14	Row13	C:\Users\emnar\OneDrive\Bureau\data\fashi		Shirt
15	Row14	C:\Users\emnar\OneDrive\Bureau\data\fashi		T-shirt
16	Row15	C:\Users\emnar\OneDrive\Bureau\data\fashi		Bag
17	Row16	C:\Users\emnar\OneDrive\Bureau\data\fashi		Dress
18	Row17	C:\Users\emnar\OneDrive\Bureau\data\fashi		Sandal
19	Row18	C:\Users\emnar\OneDrive\Bureau\data\fashi		T-shirt
20	Row19	C:\Users\emnar\OneDrive\Bureau\data\fashi		Shirt
21	Row20	C:\Users\emnar\OneDrive\Bureau\data\fashi		Bag
22	Row21	C:\Users\emnar\OneDrive\Bureau\data\fashi		Trouser
23	Row22	C:\Users\emnar\OneDrive\Bureau\data\fashi		Trouser
24	Row23	C:\Users\emnar\OneDrive\Bureau\data\fashi		Ankle boot
25	Row24	C:\Users\emnar\OneDrive\Bureau\data\fashi		Trouser
26	Row25	C:\Users\emnar\OneDrive\Bureau\data\fashi		Trouser

Figure 2: Sample images from the Fashion-MNIST dataset

1.3. Global KNIME Workflow Overview

The complete KNIME workflow used for Fashion-MNIST classification is illustrated in Figure. It provides an overview of the entire machine learning pipeline, from data loading to model evaluation.

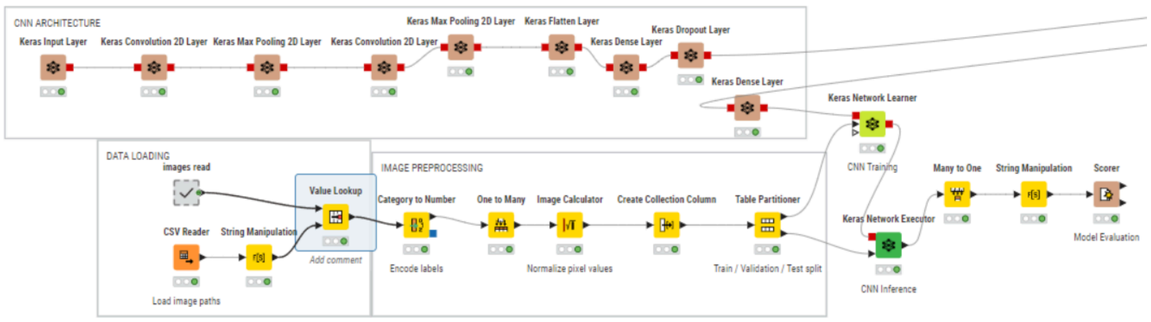


Figure 3: Complete KNIME workflow for Fashion-MNIST CNN classification

The workflow is organized into five main stages. First, image files are loaded and class labels are extracted from file paths. Second, images are normalized and reshaped to meet CNN input requirements. Third, the convolutional neural network architecture is constructed using Keras integration nodes. Fourth, the model is trained using the Keras Network Learner. Finally, the trained model is evaluated using accuracy metrics, a confusion matrix, and training curves.

1.4. Normalization and Label Encoding

1.4.1. Pixel Value Normalization

The images used in this project are already resized to a resolution of 28*28 pixels before being loaded into KNIME. Therefore, no additional image resizing or format conversion is required.

To improve the stability and convergence speed of the Convolutional Neural Network (CNN), pixel values are normalized to the range $[0, 1]$. This normalization is performed by dividing the original pixel intensities by 255.

The normalization process is implemented using the following KNIME node:

- **Image Calculator**: applies a mathematical operation to scale pixel values.

Since the images are grayscale, the input shape used by the CNN is: $28 \times 28 \times 1$

1.4.2. Label Encoding

The dataset contains categorical class labels such as **T-shirt**, **Trouser**, **Bag**, and **Ankle boot**. These labels must be converted into a numerical representation suitable for training a neural network.

The following KNIME nodes are used for label encoding:

- **Category to Number**: converts each categorical class label into a numeric index.
- **One to Many**: transforms the numeric labels into one-hot encoded vectors.

One-hot encoding is required for multi-class classification when using the categorical cross-entropy loss function during CNN training.

1.4.3. Dataset Splitting

The dataset is split into training, validation, and test sets using stratified sampling to preserve class distribution.

- Training set: 60%
- Validation set: 20%
- Test set: 20%

The partitioning is performed using the Table Partitioner node.

1.5. CNN Architecture Design

1.5.1. Network Structure

The CNN architecture is composed of the following layers:

- Two convolutional layers with 32 and 64 filters, kernel size 3 × 3, and ReLU activation
- Max pooling layers with pool size 2 × 2
- A flatten layer to convert feature maps into a vector
- A fully connected dense layer with 128 neurons
- A softmax output layer with 10 neurons corresponding to the dataset classes

This architecture allows the network to learn hierarchical spatial features efficiently.

1.5.2. CNN Implementation in KNIME

The CNN is implemented using Keras integration nodes in KNIME. The main nodes used are:

- Keras Input Layer
- Keras Convolution 2D Layer
- Keras Max Pooling 2D Layer
- Keras Flatten Layer
- Keras Dense Layer
- Keras Network Learner

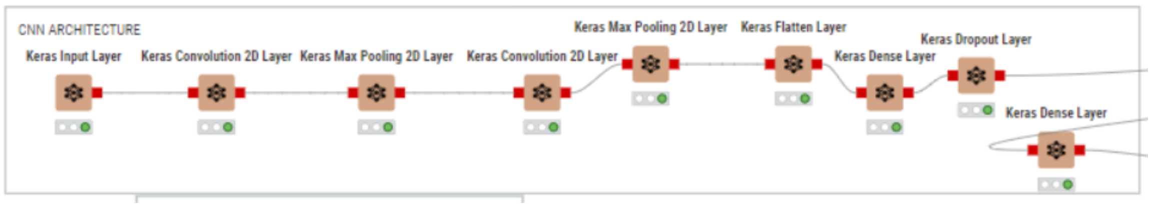


Figure 4: CNN architecture implemented using KNIME Keras nodes

1.6. Training Configuration

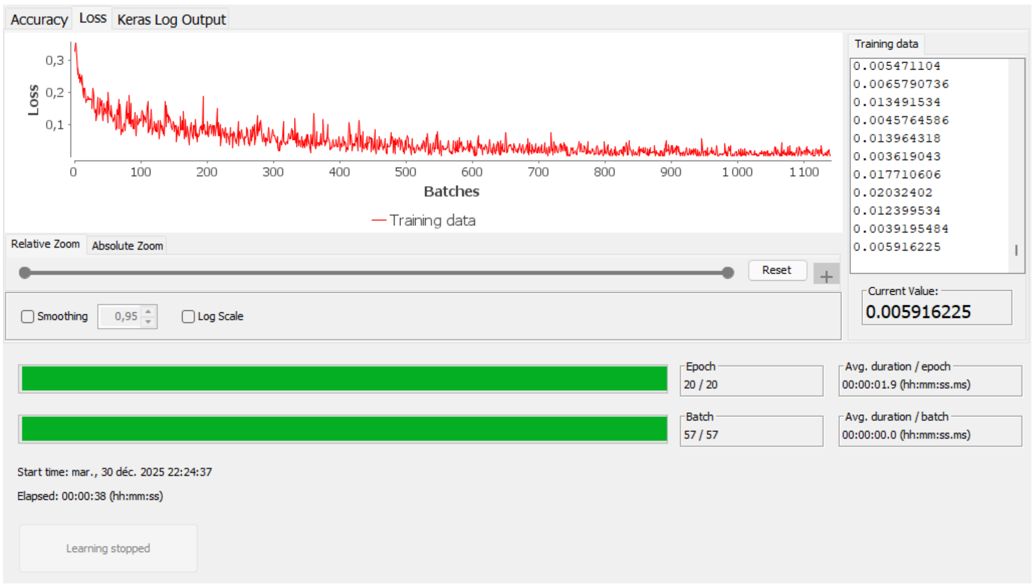
1.6.1. Training Parameters

The CNN is trained with the following configuration:

- Optimizer: Adam
- Loss function: Categorical Cross-Entropy
- Batch size: 32
- Number of epochs: 20

1.6.2. Training Curve

The training loss curve is monitored using the Keras Learning Monitor to evaluate the learning behavior and convergence of the CNN model.



```
Training data
0.005471104
0.0065790736
0.013491534
0.0045764586
0.013964318
0.003619043
0.017710606
0.02032402
0.012399534
0.0039195484
0.005916225

Current Value: 0.005916225

Smoothing 0,95   Log Scale

Epoch 20 / 20            Avg. duration / epoch 00:00:01.9 (hh:mm:ss.ms)
Batch 57 / 57            Avg. duration / batch 00:00:00.0 (hh:mm:ss.ms)

Start time: mar., 30 déc. 2025 22:24:37
Elapsed: 00:00:38 (hh:mm:ss)
```

Figure 5: Training loss evolution during CNN training

The curve shows a rapid decrease in loss during the initial batches, followed by a gradual stabilization, indicating effective learning and stable convergence of the model.

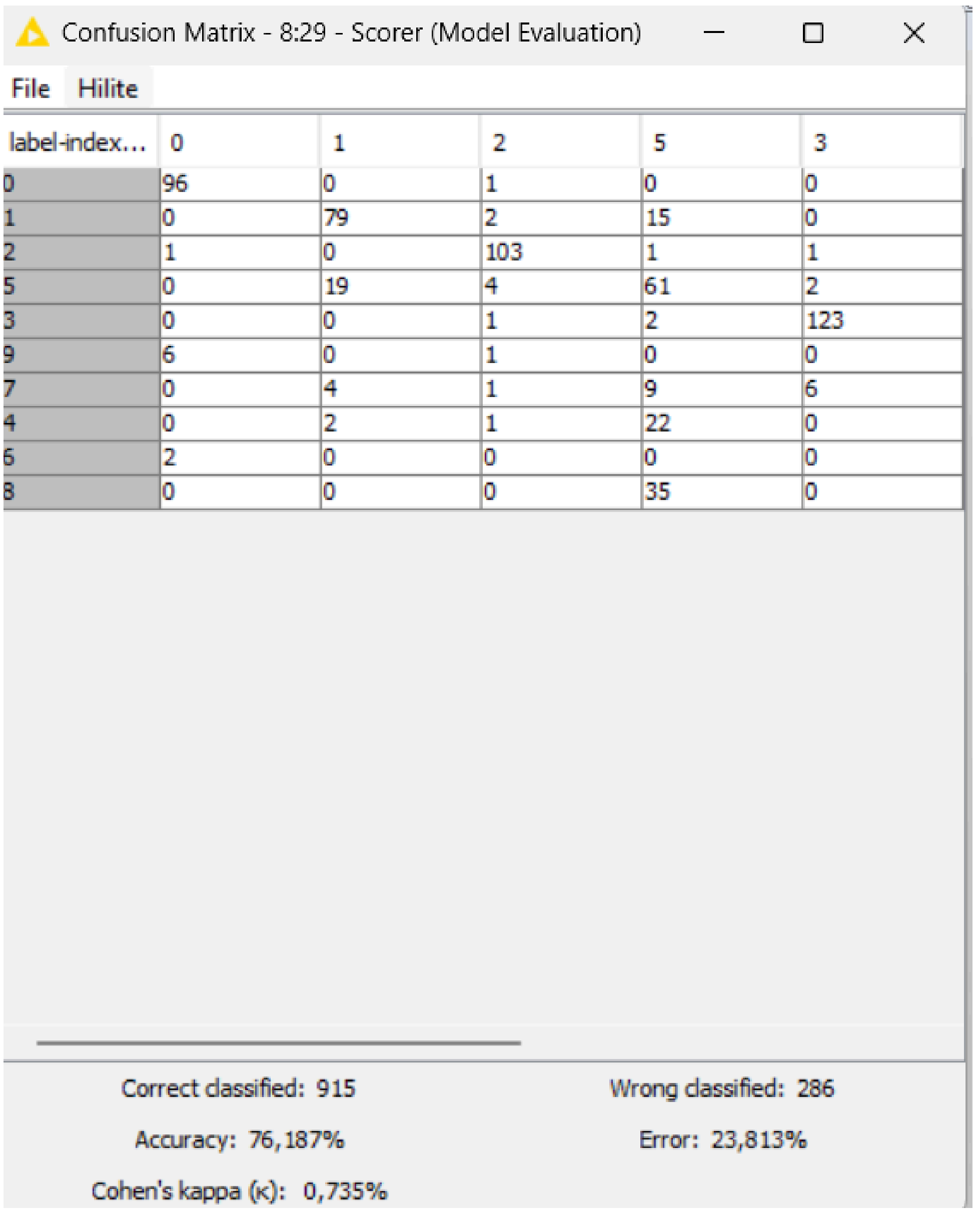
1.7. Model Evaluation

1.7.1. Test Accuracy

After training, the CNN model is evaluated on the test dataset using the Scorer node in KNIME. The model achieves an overall test accuracy of 76.187%, indicating that the network is able to learn relevant visual features and generalize reasonably well to unseen data.

1.7.2. Confusion Matrix

The confusion matrix is used to analyze the classification performance for each individual class.



label-index...	0	1	2	5	3
0	96	0	1	0	0
1	0	79	2	15	0
2	1	0	103	1	1
5	0	19	4	61	2
3	0	0	1	2	123
9	6	0	1	0	0
7	0	4	1	9	6
4	0	2	1	22	0
6	2	0	0	0	0
8	0	0	0	35	0

Correct classified: 915 Wrong classified: 286

Accuracy: 76,187% Error: 23,813%

Cohen's kappa (κ): 0,735%

Figure 6: Confusion matrix for Fashion-MNIST classification

Most correct predictions are located along the diagonal of the matrix, indicating accurate classification for several classes. However, misclassifications mainly occur between visually similar clothing items such as T-shirt, Pullover, and Shirt. This behavior is expected in the Fashion-MNIST dataset due to the similar shapes and textures of these categories.

1.8. Model Improvements and Experiments

1.8.1. Regularization Techniques

To reduce overfitting and improve generalization, regularization techniques are applied:

- Dropout layers randomly deactivate neurons during training
- Batch normalization stabilizes and accelerates training

The following nodes are used:
- Keras Dropout Layer
- Keras Batch Normalization Layer

1.9. Discussion

The CNN performs well on most classes and successfully extracts meaningful visual features. However, confusion remains between visually similar clothing items. Regularization techniques improve generalization but may slightly reduce training accuracy.

1.10. Conclusion

In this lab, a Convolutional Neural Network was successfully designed and trained using the KNIME Analytics Platform to classify Fashion-MNIST images. The model achieved good classification accuracy and demonstrated the effectiveness of convolutional architectures for image recognition tasks.