Batik Image Classification Using a Convolutional Neural Network

1. Project Overview

Indonesian batik is a well-known traditional textile created using a resist-dye technique that has been practiced for centuries. This technique produces complex patterns on cotton or silk fabrics, which are commonly used for clothing, tableware, and even baby carriers, highlighting batik's deep cultural significance and widespread use in everyday life.

Originally from Java, batik has been adopted across different regions of Indonesia, with each province developing its own distinctive styles characterized by unique patterns and color palettes. This regional diversity makes batik an interesting and challenging subject for image classification tasks.

Given this context, this project aims to apply convolutional neural networks (CNNs) to classify batik patterns based on their region of origin. The goal is to build a model capable of identifying and predicting the originating region of a batik image.

The dataset used in this project was sourced from Kaggle and is organized into training and testing folders, with each class representing a specific regional or provincial batik motif. In total, the dataset contains 20 distinct batik classes.

This project demonstrates an end-to-end CNN image classification workflow, including data preprocessing, model design, training, evaluation, and performance analysis on a real-world dataset. It also explores hyperparameter choices and highlights both the strengths and limitations of training a CNN from scratch..

2. Dataset Description

The dataset was taken from the following kaggle link:
https://www.kaggle.com/datasets/hendryhb/batik-nusantara-batik-indonesia-dataset

There are 20 classes, aka batik motifs. Per class consists of 32 raw images with image dimensions of 224 x 224 for training. Whereas test data consists of 8 images per class. This reveals that there are around 32x20 = 640 images to train with, which is relatively small for a CNN model. In the following paragraphs, we will see how this translates in the model evaluation and what can be improved in the future.

3. Data Preprocessing

Image Resizing

All images were resized to a fixed resolution of 224 × 224 pixels. This ensures a consistent input shape for the convolutional neural network and allows efficient batch processing during training.

Pixel Normalization

Original image pixel values ranged from 0 to 255, where 0 represents black and 255 represents maximum brightness. These values were rescaled to the range [0, 1] by dividing each pixel by 255.

Normalization improves training stability by keeping input values within a small, consistent range and helps the optimizer converge more efficiently. After normalization, a pixel value of 0 remains 0, and a pixel value of 255 becomes 1.

Data Augmentation

The dataset used in this project is relatively small for deep learning, which increases the risk of overfitting.

To minimise this, data augmentation was applied only to the training set. Data augmentation increases variability in the training data without altering the class labels, encouraging the model to learn robust visual features rather than memorizing specific images.

The following augmentation techniques were applied:

- Random horizontal flipping
- Random rotation (up to 10%)
- Random zooming in and out

By exposing the model to different variations of the same image, data augmentation improves generalization and reduces overfitting on small datasets.

4. Model Architecture

The final model consists of three convolutional blocks, followed by global average pooling and two fully connected (dense) layers, with a total of approximately 29,000 trainable parameters.
The convolutional layers are responsible for feature extraction, while the dense layers perform classification.

Convolutional Layers

The model includes three convolutional layers with an increasing number of filters:

- 16 filters in the first layer to learn basic visual patterns such as edges and lines
- 32 filters in the second layer
- 64 filters in the third layer to capture more complex and abstract patterns

All convolutional layers use a kernel size of 3 × 3, meaning each filter scans small local regions of the image. This allows the model to learn spatial patterns efficiently.

The ReLU (Rectified Linear Unit) activation function is used after each convolution. ReLU introduces non-linearity into the model and determines which neurons are activated and passed to the next layer, enabling the network to learn complex relationships.

Pooling Layers

Each convolutional layer is followed by a MaxPooling2D layer with a 2 × 2 window.
The pooling operation scans the feature maps (outputs of the convolutional layers) and retains only the maximum value in each 2 × 2 region.

This process:

- Reduces the spatial dimensions of the feature maps
- Decreases computational complexity
- Retains the most important features
- Helps reduce overfitting

Global Average Pooling

After the final convolutional block, Global Average Pooling is applied. This layer converts each feature map into a single numerical value by averaging all its spatial values.

At this stage, the data still resembles a grid or image-like structure, whereas dense layers require a one-dimensional feature vector. Global average pooling acts as a bridge between feature extraction and classification by transforming feature maps into a compact 1D vector.

Additionally, this approach significantly reduces the number of trainable parameters, which helps improve generalization.

Dropout

A Dropout layer with a dropout rate of 0.3 is applied during training. This regularization technique randomly sets 30% of the input values to zero in each training step, preventing the model from relying too heavily on specific features.

Dropout helps reduce overfitting, especially when working with limited data.

Dense Layers

The first dense layer contains 64 neurons and receives 64 input values from the global average pooling layer. Each neuron:

- Is fully connected to all input features
- Learns different combinations of extracted image features

This layer enables the model to combine learned visual patterns into higher-level representations.

The final dense layer serves as the classification layer and uses a softmax activation function. Softmax:

- Converts raw output scores into probabilities between 0 and 1
- Ensures that all class probabilities sum to 1

Each neuron in this layer corresponds to one batik class, and the class with the highest probability is selected as the model's prediction.

5. Model Training

Loss Function

Categorical crossentropy was used as the loss function, as this is the standard choice for multi-class classification problems with one-hot encoded labels and a softmax output layer. This loss function penalizes confident incorrect predictions more heavily, encouraging the model to assign high probability to the correct class.

Optimizer

The Adam optimizer was selected for training due to its efficiency and stability. Adam adapts the learning rate automatically during training, making it well-suited for convolutional neural networks and reducing the need for manual tuning.

Training Configuration

The model was trained for a maximum of 50 epochs, where one epoch corresponds to a full pass through the training dataset. However, training did not necessarily run for all 50 epochs due to the use of early stopping.

Early Stopping

An early stopping strategy was applied to prevent overfitting and unnecessary computation. Training was stopped if the validation loss did not improve for 10 consecutive epochs, and the model weights were restored to the best-performing state observed during training. This ensures that the final model represents the best generalization performance rather than the final training iteration.

6. Evaluation Metrics

Test Performance

- **Test Accuracy:** approximately 32.5%
- **Test Loss:** approximately 2.32

Why These Metrics Are Appropriate for Multi-Class Classification

The task involves classifying images into 20 batik motif classes, with exactly one correct label per image. The model outputs a probability distribution over all classes using a softmax activation, where the class with the highest probability is selected as the prediction.

Accuracy

Accuracy measures the proportion of correctly classified images and provides a clear and intuitive indicator of overall model performance. It is particularly useful for high-level comparison between models, especially when class distributions are relatively balanced, as is the case in this dataset (approximately 32 images per class).

However, accuracy alone does not reveal *which* classes are being misclassified or the nature of those errors. To address this limitation, additional analysis using a confusion matrix is required.

Loss

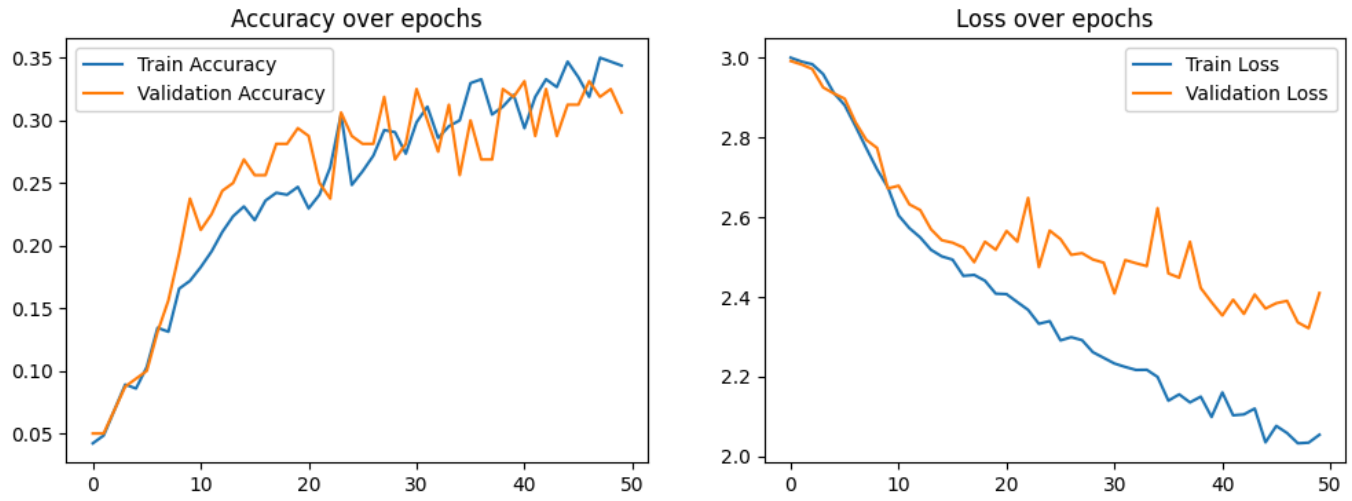
Loss measures how far the predicted probability distribution deviates from the true labels. Unlike accuracy, loss accounts for prediction confidence, penalizing confident incorrect predictions more severely than uncertain ones. As a result, loss provides deeper insight into model behavior by capturing not only how many mistakes are made, but also how severe those mistakes are.

7. Results

Test Performance

- **Test Accuracy:** approximately **32.5%**
- **Test Loss:** approximately **2.32**

Training vs Validation Curves



The training curve shows how well the model fits the training data while the validation curve shows how well the model generalizes to unseen data. What we aim for is both improve together to show a healthy learning pattern.

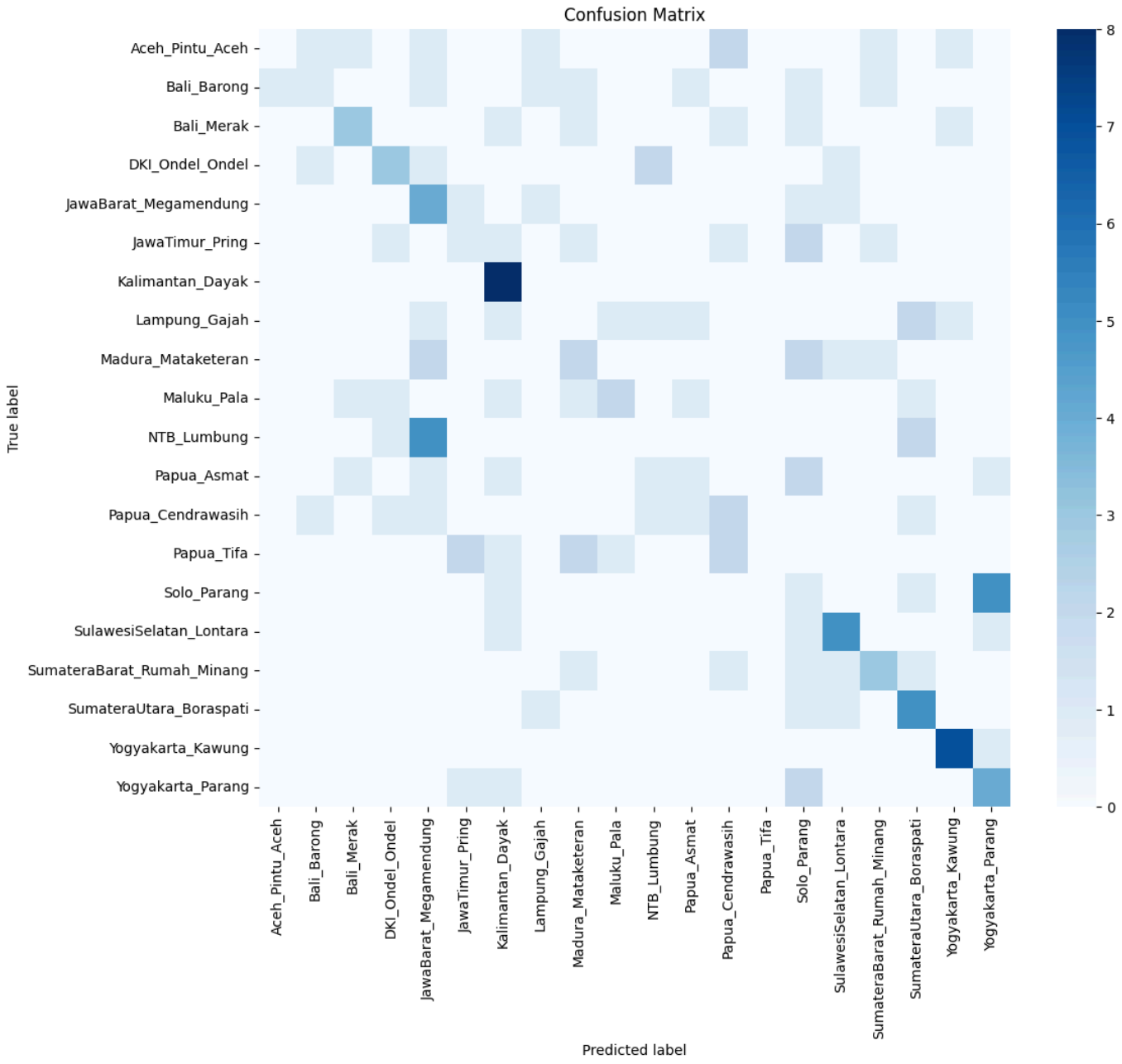
Accuracy Over Epochs

We see that training and validation accuracy increase slowly (ends between 0.30 - 0.35 which is a low value), the curves stay relatively close and no large divergence. As both models plateau at low performance (0.30 - 0.35 range), the plot shows that the model is underfitting the data: in other words, the model cannot fully capture the complexity of batik patterns. This may be due to limited data and training from scratch.

Loss Over Epochs

Train Loss is computed on training data and is expected to decrease as the model learns. Whereas validation loss is computed on unseen validation data and tells us how well the model generalises. In our model, we see that the train loss continues to decrease over epochs as expected. However, we see that the validation loss starts to plateau at epoch 15. This illustrates that the model is learning but its improvements don't transfer fully to unseen data. This is a sign of limited model capacity or limited data, otherwise known as underfitting.

Confusion Matrix



Understanding The Confusion Matrix

Rows show true class, columns show predicted class. Diagonal cells illustrate correct predictions, so the darker the cell, the better performance for that class (i.e. we see Kalimantan Dayak has the darkest cell illustrating that our model does well classifying for those classes). Off-diagonal cells show misclassifications, in which classes are confused with each other. Darker off-diagonal cells mean the model frequently misclassifies one class as another (e.g. NTB Lumbung and Jawa Barat Megamendung: around 4-5 images of NTB Lumbung were predicted as Batik Jawa Barat Megamendung). This is likely due to high visual similarity and shared pattern structures. This indicates overlapping feature representations learned by the CNN, which is expected when training from scratch on a limited dataset.

In summary, the confusion matrix is crucial for understanding model weaknesses and identifies visually similar batik motifs. Our confusion matrix shows that our model only predicts very few correct batik patterns, and confuses some patterns with the other.

9. Limitations

Three limitations can be seen from the results and analysis section: 1) Dataset size constraints: From our Loss over Epochs plot we see that the model's validation loss starts to plateau at early epochs, illustrating that although it is still learning it is not able to generalise to unseen data, indicating a limited model or data capacity. 2) This then leads to the second limitation, training CNN from scratch: This model was built with only 3 convolutional layers and two dense layers, which, given the limited data and yet the complexity of batik patterns, this model may be too simplistic for the dataset. 3) Lastly, this model is limited due to high visual similarity between classes. This is shown from the confusion matrix, whereby we see many darker shades of cells outside the intended dark diagonal line - which illustrates correct classification. This means that the model tends to misclassify batik patterns, with NTB Lumbung and Jawa Barat Megamendung pairing being the worst (highest number of NTB Lumbung patterns that were misclassified as Jawa Barat Megamendung). In short, this model is limited due to data and model capacity, and visual similarities between batik patterns. The following section will address how best we can better these limitations and therefore improve the overall model.

10. Future Improvements

The model can be improved in several ways.

1) Addressing model simplicity through transfer learning

To address the issue of model simplicity, pretrained convolutional neural networks (CNNs) can be explored and fine-tuned to better suit batik pattern classification. Pretrained CNNs are models that have already been trained on millions of images using deep convolutional architectures, and as a result, they generally perform better than models trained from scratch on small datasets.

An example is Google's MobileNetV2, a lightweight 53-layer convolutional neural network designed for efficient image classification, detection, and segmentation tasks. By leveraging such pretrained models and fine-tuning them on batik images, the model can benefit from robust feature representations learned from large-scale datasets while adapting to batik-specific patterns.

2) Addressing data constraints with a larger dataset

The analysis also shows that model performance is limited by dataset size. The current dataset is relatively small for deep learning, which restricts the model's ability to generalize. To address this limitation, a larger dataset of labeled batik patterns is required.

A suitable alternative dataset was identified on Kaggle
(https://www.kaggle.com/datasets/hydiexe/dataset-fix),
which contains approximately 14,000 batik images split into 70% training, 20% validation, and 10% testing. Using this larger dataset would provide more diverse examples and allow more complex models to be trained effectively.

11. Conclusion

The baseline CNN trained from scratch achieves ~32% accuracy, outperforming random guessing but struggling with generalization due to limited data and high inter-class similarity. To address this, transfer learning with a pretrained CNN (e.g., MobileNet) is a natural next step, as it leverages robust low-level feature representations learned from large-scale image datasets.