Task 1: Classification and outlier detection

Ameya Rathod
International Institute of Information Technology, Hyderabad
ameya.rathod@research.iiit.ac.in

March 24, 2025

Abstract

This report details the methodology, experiments, and evaluation results for two approaches to classify paintings based on attributes such as artist, genre, and style using convolutional-recurrent architectures. In addition, an outlier detection module was implemented based on embeddings, prediction confidence, and Mahalanobis distance. Two model variants were explored: a model trained from scratch and a model utilizing an EfficientNet-B0 backbone. The report includes a discussion of the network architectures, training procedures, evaluation metrics, and a comparative analysis of the experimental results.

1 Introduction

The aim of this work is to develop deep learning models for art classification and outlier detection using convolutional-recurrent architectures. Specifically, we focus on:

- **Classification** of paintings by artist, genre, and style.
- **Outlier Detection:** Identifying paintings that do not conform to their assigned categories.

For classification, two variants were explored:

1. Training a convolutional-recurrent network from scratch.
2. Leveraging a pre-trained EfficientNet-B0 model combined with LSTM layers.

2 Task 1: Convolutional-Recurrent Architectures for Painting Classification

2.1 Methodology

Two distinct approaches were implemented.

2.1.1 Variant 1: Training from Scratch

In the first approach, a convolutional-recurrent network was built from scratch. The architecture consists of:

- **Convolutional Layers:** Two convolutional layers with ReLU activation followed by max pooling.
- **Adaptive Average Pooling:** Used to standardize the output spatial dimensions.
- **LSTM Layer:** A bidirectional LSTM that processes the reshaped feature map.
- **Dropout and Fully Connected Layer:** To regularize and produce the final class predictions.

The Python code for the model is given below:

```
class RecurrentCNN(nn.Module):
    def __init__(self, num_classes, lstm_hidden_size=256, dropout_prob
       =0.5):
        super(RecurrentCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.adaptive_pool = nn.AdaptiveAvgPool2d((14, 56))
        self.lstm_input_size = 64 * 56
        self.lstm_hidden_size = lstm_hidden_size
        self.lstm = nn.LSTM(input_size=self.lstm_input_size, hidden_size=
           lstm_hidden_size,
                            batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc = nn.Linear(2 * lstm_hidden_size, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = self.adaptive_pool(x)
        x = x.permute(0, 2, 1, 3).contiguous()
        batch_size, seq_len, channels, width = x.shape
        x = x.view(batch_size, seq_len, channels * width)
        lstm_out, _ = self.lstm(x)
        x = lstm_out.mean(dim=1)
        x = self.dropout(x)
        x = self.fc(x)
        return x
```

2.1.2 Variant 2: Using EfficientNet-B0

The second variant employs a pre-trained EfficientNet-B0 backbone for feature extraction. The steps include:

- **EfficientNet-B0 Backbone:** Pre-trained on ImageNet, outputs 1280 feature channels.
- **Channel Reduction:** A convolutional block reduces the number of channels from 1280 to 512.
- **LSTM Layers:** Two layers of bidirectional LSTM process the spatial features.
- **Classifier:** A fully connected classifier maps the final LSTM representation to the desired output classes.

The corresponding code snippet is shown below:

```
class EffNetLSTM(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        # EfficientNet-B0 backbone (outputs 1280 channels)
        effnet = models.efficientnet_b0(weights=models.
           EfficientNet_B0_Weights.IMAGENET1K_V1)
        self.cnn = effnet.features

        self.channel_reducer = nn.Sequential(
            nn.Conv2d(1280, 512, kernel_size=1),
            nn.BatchNorm2d(512),
            nn.ReLU()
        )

        self.lstm = nn.LSTM(
            input_size=512,
            hidden_size=256,
            num_layers=2,
            bidirectional=True,
            batch_first=True
        )

        self.classifier = nn.Sequential(
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        features = self.cnn(x)
        x = self.channel_reducer(features)
        bs, c, h, w = x.size()
        x = x.permute(0, 2, 3, 1).reshape(bs, h*w, c)
        lstm_out, (h_n, c_n) = self.lstm(x)
        last_hidden = torch.cat((h_n[-2], h_n[-1]), dim=1)
        return self.classifier(last_hidden)
```

2.2 Evaluation Metrics and Results

The models were evaluated on three classification tasks: artist, genre, and style. The evaluation was performed using Top-1 and Top-5 accuracy metrics.

2.2.1 Results

Table 1: Classification Performance (EfficientNet-B0 Variant)

Category	Top-1 Accuracy	Top-5 Accuracy
Artist	0.75	0.94
Genre	0.75	0.98
Style	0.56	0.92

Table 2: Classification Performance (Trained from scratch)

Category	Top-1 Accuracy	Top-5 Accuracy
Artist	0.51	0.82
Genre	0.56	0.94
Style	0.36	0.79

It is evident that the EfficientNet-B0 based model outperforms the model trained from scratch in most cases.

3 Outlier Detection

Outlier detection was performed using three approaches:

1. **Embeddings:** Analyzing the learned feature embeddings to identify paintings that lie far from typical clusters.
2. **Confidence of Prediction:** Setting a threshold on prediction confidence (with a threshold of 0.5) to flag uncertain classifications.
3. **Mahalanobis Distance:** Measuring the statistical distance between a painting's feature vector and the class mean.

The effectiveness of the outlier detection was measured by the accuracy of correctly flagging mis-assigned paintings. The obtained values are summarized in Table 3.

It was observed that the confidence-based method yielded the highest accuracy for outlier detection when using a threshold of 0.5, particularly for the scratch-based model in artist and style categories.

Table 3: Outlier Detection Accuracy using confidence

Model Variant	**Artist**	**Genre**	**Style**
EfficientNet-B0	0.70	0.66	0.67
Scratch	0.77	0.66	0.72

4 Discussion and Conclusion

The experiments demonstrate that leveraging a pre-trained EfficientNet-B0 backbone combined with LSTM layers results in improved classification performance over a model trained from scratch. For example, the EfficientNet-B0 variant achieved Top-1 accuracies of 0.75 for artist and genre compared to 0.51 and 0.56 from the scratch variant, respectively. However, the scratch-based model showed a slight edge in outlier detection accuracy for artist (0.77 vs. 0.70) and style (0.72 vs. 0.67) categories.

Outlier detection is most successful when we consider confidence of prediction to classify.