SMOTE-Based Synthetic Image Generation for MNIST Digits

Syed Muhammad Mehran 24L-8014

Introduction

The MNIST dataset is a benchmark in the field of machine learning, commonly used for image classification tasks involving handwritten digits. This report presents an experiment involving the Synthetic Minority Over-sampling Technique (SMOTE) to generate new examples for digits 3 and 5 from the MNIST dataset. While SMOTE is traditionally used for tabular data, we explore its effects when applied directly to pixel vectors of image data.

Methodology

We implemented SMOTE from scratch in Python. The technique generates synthetic samples by interpolating between a data point and one of its k-nearest neighbors. The following steps were carried out:

1. Extract 50 samples each of digits 3 and 5 from the MNIST training dataset.
2. For each class, use the custom SMOTE class to generate 15 new samples.
3. Display the synthetic images to visually inspect the generated samples.

The following Python code outlines the implementation:

```
def compute_knn(X, k=5):
    n_samples = X.shape[0]
    knn_indices = np.zeros((n_samples, k), dtype=int)

    for i in range(n_samples):
        distances = np.linalg.norm(X - X[i], axis=1)
        distances[i] = np.inf
        neighbors = np.argsort(distances)[:k]
        knn_indices[i] = neighbors

    return knn_indices

class Smote():
    def __init__(self, data, k=5, samples_to_generate=15):
        self.k = k
        self.samples_to_generate = samples_to_generate
        self.X = data

    def smote_implementation(self):
        self.new_samples = []
        knn_indices = compute_knn(self.X, self.k)

        for i in range(self.samples_to_generate):
            j = np.random.randint(0, len(self.X))
            nn = np.random.choice(knn_indices[j])
            gap = np.random.rand()
            diff = self.X[nn] - self.X[j]
            new_sample = self.X[i] + gap * diff
            self.new_samples.append(new_sample)

        return np.array(self.new_samples)

    def generate_images(self, title):
        rows = 5
        cols = 3
        plt.figure(figsize=(8, 10))
        for i, img in enumerate(self.new_samples):
            plt.subplot(rows, cols, i+1)
            plt.imshow(img.reshape(28, 28), cmap='gray')
            plt.axis('off')
        plt.suptitle(title)
        plt.tight_layout()
        plt.show()
```

Listing 1: Python implementation of SMOTE with KNN from scratch

Results

The following synthetic images were generated using the SMOTE technique applied to the MNIST digits:

- **Digit 3:** 15 synthetic samples generated using 50 original samples.
- **Digit 5:** 15 synthetic samples generated similarly.

Visual inspection revealed that some samples preserved the digit structure well, while others appeared blurry or ghost-like. This is expected since SMOTE blends pixel values linearly, which does not always preserve the stroke continuity in handwritten digits.

Discussion

While SMOTE is effective for low-dimensional tabular datasets, its direct application to image data has limitations:

- Interpolating pixel values does not guarantee perceptual coherence.
- Some generated samples were fuzzy or appeared like a combination of multiple digits.
- Alternatives such as affine transformations, GANs, or autoencoders may yield better synthetic image quality.

Despite these limitations, the experiment demonstrates a creative adaptation of SMOTE for image data and highlights the importance of using domain-specific augmentations for vision tasks.

Conclusion

This experiment implemented SMOTE from scratch and explored its application to MNIST digits. While results were mixed, it served as an insightful exploration into generative data augmentation techniques. For practical applications in computer vision, more specialized methods may be preferred.

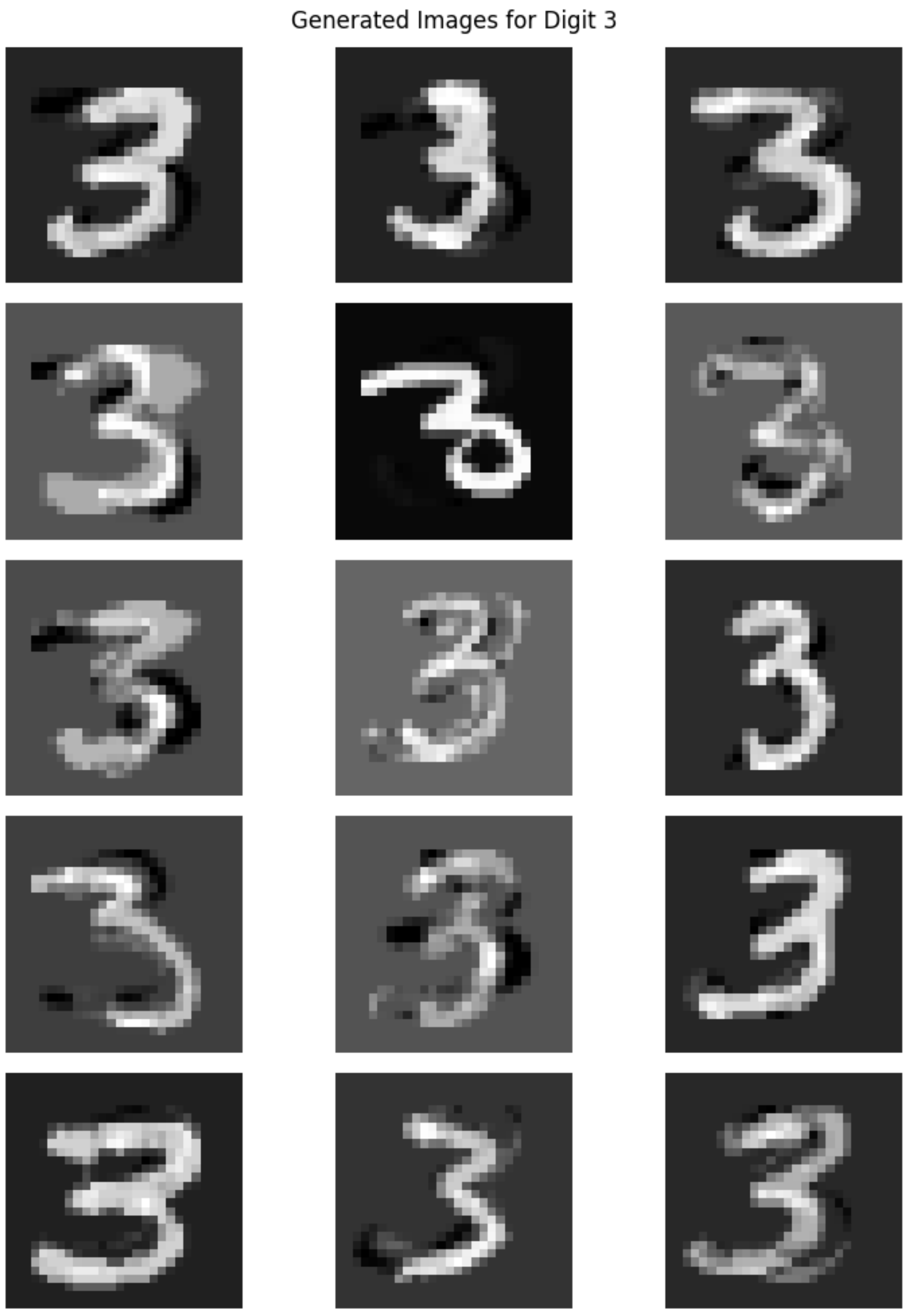

Figure 1: SMOTE-generated images for digit 3

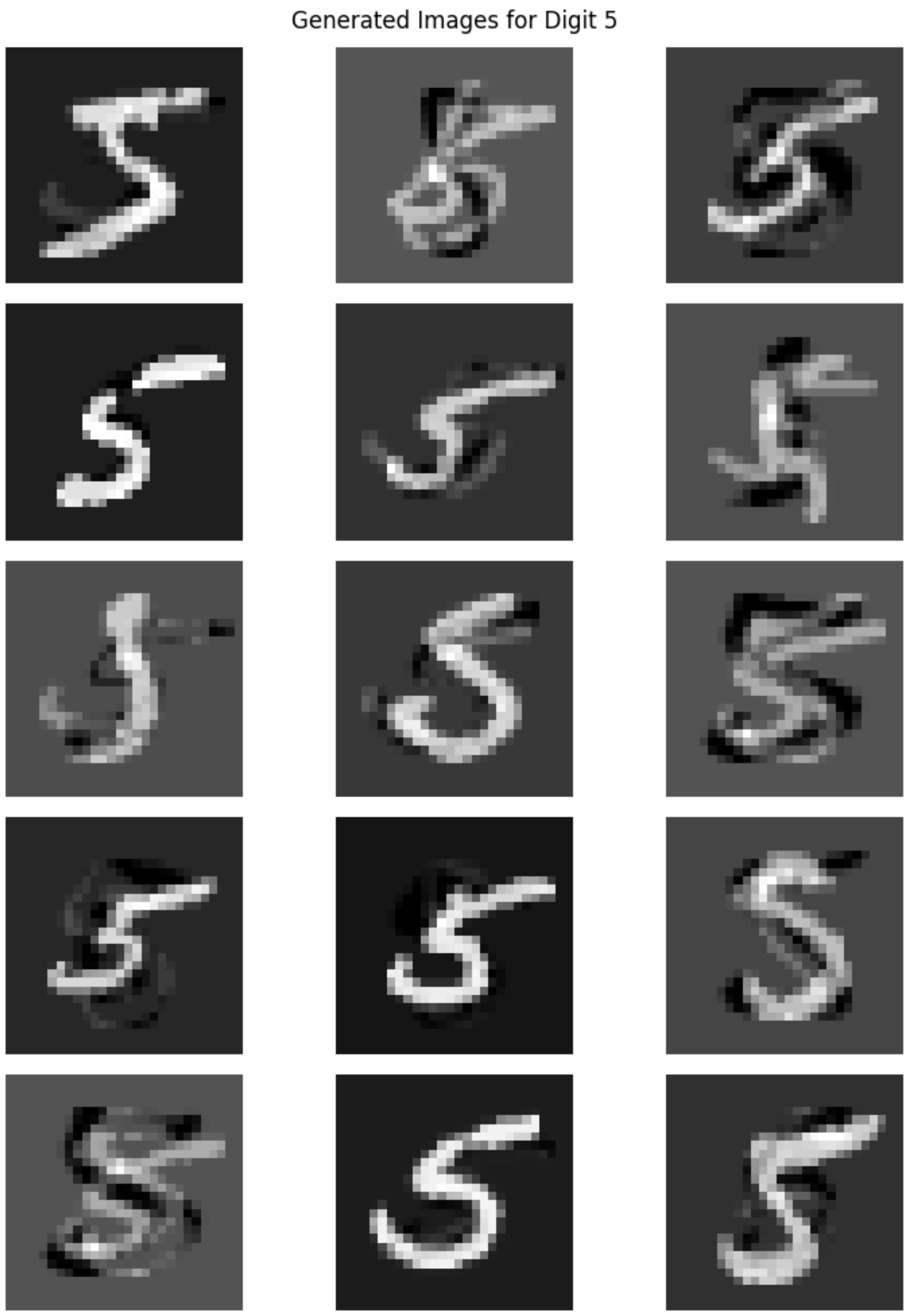

Figure 2: SMOTE-generated images for digit 5