The Synthetic Dilemma: Assessing the Impact of AI-Generated Images on the Accuracy of Convolutional Neural Networks, Illustrated through the Example of Emotion Classification

L. Schürmann

March 12, 2024

Abstract

This study investigates the efficacy of incorporating AI-generated images as supplementary data in emotion-recognizing Convolutional Neural Networks. Emotion recognition is a critical task in various applications such as human-computer interaction and affective computing. However, limited labeled data availability often hinders the performance of deep learning models. To address this challenge, we explore the use of stable diffusion, a generative model capable of synthesizing high-quality images, to augment existing datasets.

1 Introduction

In the realm of artificial intelligence and machine learning, the quest for enhancing model performance through dataset augmentation is an ongoing endeavor. This study delves into the intriguing realm of supplementing an existing emotion classification dataset with controlled random stable diffusion images. By incorporating synthetic data generated through stable diffusion techniques, this research explores not only the generation and training processes but also delves into the performance evaluation of the resulting models. Through meticulous examination of the interplay between synthetic and authentic data, this investigation aims to shed light on the efficacy and implications of leveraging controlled synthetic imagery to augment real-world datasets. Beyond merely expanding the dataset, this study seeks to unravel the intricate dynamics of how synthetic data integration influences the accuracy and robustness of emotion classification models, thus offering valuable insights for advancing the frontier of AI-driven emotion recognition systems.

Disclaimer: The following study[1] is a small side project conducted independently and should not be construed as part of or associated with my current studies at the university. This research endeavor is undertaken solely for personal interest and exploration, and its findings are not intended to be applied or interpreted within an academic or professional context.

2 Dataset

2.1 Base Dataset

The base dataset utilized in this research comprises 28,821 training images and 7,066 validation images labeled with following emotions: "angry", "disgust", "fear", "happy", "neutral", "sad", "surprise". Each image is represented in grayscale and exclusively contains the facial region, optimizing computational efficiency for real-time applications, particularly on lower-end hardware configurations. With a standardized dimension of 48x48 pixels, this dataset offers a focused and manageable scope for training CNNs tailored to emotion recognition tasks. The deliberate selection of grayscale and facial region exclusivity ensures a streamlined computational pipeline, aligning to achieve efficient real-time performance while maintaining sufficient data granularity for accurate emotion classification.

[1] GitHub Repository & Source Code

Figure 1: 128 Example Images from our Base Dataset

2.2 Synthetic Dataset

The synthetic dataset, generated using stable diffusion techniques, comprises 5,096 training images labeled accordingly. Each image is presented in grayscale and is limited to the facial region, maintaining consistency with the base dataset's specifications. Despite originating from a higher resolution of 256x256 pixels, the images have been downscaled to 128x128 pixels to enhance generation quality and conserve disk space.

Figure 2: 128 Example Images from our Synthetic Dataset

2.3 Concatenation of Datasets

The concatenation of the base and synthetic datasets involves resizing the synthetic images, originally 128x128 pixels, to match the dimensions of the base dataset at 48x48 pixels. This process ensures uniformity in dataset dimensions while incorporating the synthesized data seamlessly into the training and validation sets.

3 Generation Pipeline

The generation process leveraged the Stable Diffusion 1.5[1] checkpoint, specifically the epiCRealism pure Evolution V5[2], renowned for its ability to produce realistic face images exhibiting diverse expressions. Employing a combination of random prompts for expression, sex, age, and ethnicity, the generation process aimed to ensure the synthesis of a varied and representative dataset. By incorporating these dynamic prompts, the synthetic images captured a spectrum of facial expressions and demographic characteristics, essential for training robust emotion recognition models.
Generated locally on a single Nvidia GTX 1070, the process yielded approximately 0.25 images per second, reflecting a computationally efficient approach to synthetic data creation within constrained hardware environments.
Generation Attributes:

- Batch Size: 8
- CFG Scale: 7
- Inference Steps: 25
- Width, Height: 256x256
- Negative Prompt: "text, watermark, cutout, sketch, worst quality, glitches, mutations, extra limbs, missing limbs, missing fingers, body, cgi, 3d render, unreal"
- Example Prompt: "happy teenage hispanic male, portrait of the whole face, round realistic eyes, photo, ultra-realistic, 8k, amazing details, amazing skin texture, natural mouth, natural hair, smiling, laughing, joyful look"

In each generation iteration, OpenCV was employed to verify the presence of a visible and detectable face within the synthesized images. This quality control step ensured that only images containing identifiable facial features were retained while discarding instances of failed or incomplete generations. By incorporating this validation process, the dataset maintained a high standard of image quality and consistency.

Figure 3: Example Failure Generations

Despite the utilization of OpenCV for face detection, occasional instances of false positives were identified, comprising approximately 2-4% of the dataset. These false detections represented non-facial elements mistakenly recognized as faces by the algorithm and required manual removal to maintain dataset integrity.

[1] Huggingface Model Card
[2] CivitAI Checkpoint

4 Model

To accommodate the real-time requirements and limited hardware resources, a lightweight custom Convolutional Neural Network architecture was devised for the emotion recognition task. The design of this custom CNN prioritized computational efficiency and faster training times while ensuring adequate performance for real-time applications. Although ResNet18 and similar pre-trained models offer enhanced capabilities, their deployment necessitates higher computational resources and training times, which were not aligned with the objectives of this research endeavor.

4.1 Structure

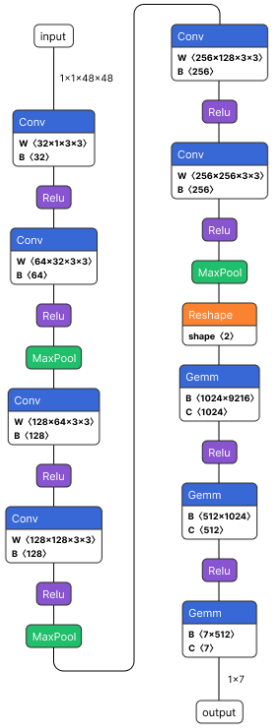


Figure 4: CNN Model Structure

4.2 Training

Four models were sequentially trained on a single Nvidia GTX 1070, with each training session lasting approximately two hours.

Hyper-Parameter:

- Batch Size: 128
- Epochs: 250
- Learning Rate: 0.001
- Optimizer: Adam

5 Results

5.1 Evaluation: Base Dataset

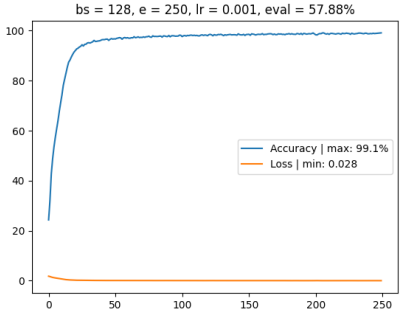
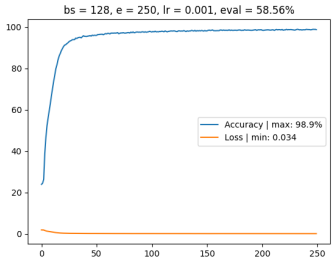


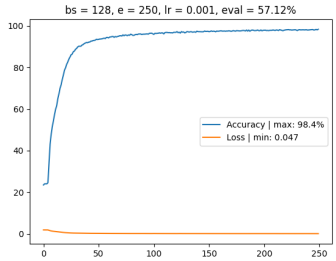
Figure 5: Training with Base Dataset

The base model exhibited the highest training accuracy and lowest loss among the four models evaluated. Despite these favorable training metrics, its evaluation accuracy ranked as the second best, albeit with a marginal difference of 0.68% from the top-performing model. This variance in evaluation accuracy could potentially be attributed to the random initialization of weights at the beginning of training.

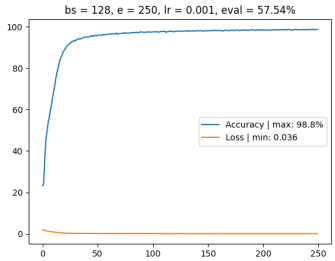
5.2 Evaluation: Base + Synthetic Dataset



(a) 7.12% Synthetic



(b) 11.78% Synthetic



(c) 15.02% Synthetic

Figure 6: Training with Base + Synthetic Dataset

The three additional models, while exhibiting slightly lower performance compared to the base model, demonstrated no significant deviation in evaluation accuracy, with differences ranging from 0.34-0.76%. Despite these minor disparities, the overall evaluation metrics remained consistent, suggesting

that the additional models did not incur a significant reduction in evaluation performance compared to the base model.

5.3 Application Evaluation

The evaluation of the four models revealed a discernible performance discrepancy in real-world application scenarios. Despite comparable training accuracies, variations in model behavior became apparent during practical evaluation, indicating nuanced differences in their ability to generalize to unseen data.

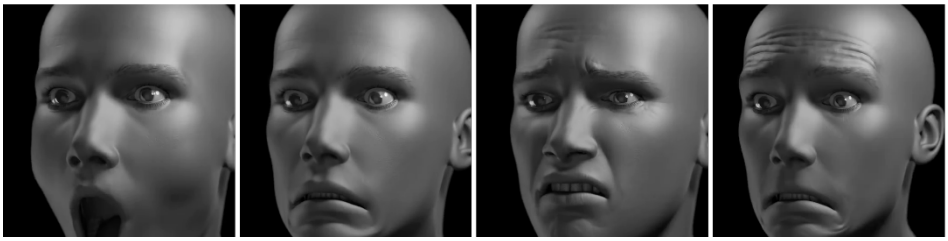


Figure 7: Difficult Situation Examples

Image	Correct Label	Base Prediction	Base + Synthetic Prediction
1	Surprise	Surprise	Sad
2	Fear	Fear	Neutral
3	Disgust	Neutral	Neutral
4	Fear	Angry	Sad

Table 1: Associated Labels

In challenging scenarios involving footage of a 3D-rendered human exhibiting facial expressions, both CNNs - one trained with synthetic supplement data and the other without - demonstrated comparable performance[1]. However, under particularly demanding conditions, the base CNN exhibited marginally superior results. Notably, both CNN architectures encountered difficulty in recognizing expressions of disgust consistently.

5.4 Conclusion

In conclusion, the tests conducted indicate that while the synthetic data augmented the dataset with higher-quality images, the model's performance suffered in challenging scenarios. Despite efforts to ensure diversity in facial expressions and human appearance, the similarity among the AI-generated faces may have contributed to a reduction in dataset diversity, thereby dulling the model's ability to generalize effectively in difficult cases. However, in non challenging scenarios, where facial expressions are less varied, the performance of the model remained largely consistent.
Drawing parallels to the observable decline in linguistic diversity[2] resulting from the proliferation of AI-generated content on the internet, the findings of this study underscore broader concerns regarding the impact of synthetic data on dataset diversity. Much like the homogenization of language and expression in digital communication platforms, the inundation of AI-generated images, while visually appealing and high in quality, risks diminishing the diversity essential for robust model performance across varied real-world scenarios. As AI technologies continue to evolve and permeate various domains, including image synthesis and natural language generation, it becomes imperative to address the potential consequences of reduced diversity in datasets.

[1]The individual frames are accessible for viewing in the "application_eval" directory of the GitHub Repository.
[2]The Curious Decline of Linguistic Diversity: Training Language Models on Synthetic Text

6 Code

In the subsequent sections, we provide a concise overview of our source code implementation process, spanning from the creation of custom datasets to the development of the PyTorch model and real-time evaluation using OpenCV. This walkthrough is intended for individuals with a keen interest in coding. The complete source code is openly accessible on GitHub for further examination and utilization.

Disclaimer: The following code snippets are presented as pseudo-Python code and are intended solely to illustrate the underlying process and steps involved. They are not executable code and should be interpreted as conceptual representations rather than functional implementations.

6.1 Diffusion Process

```
def main():
    # Load SD Checkpoint & CascadeClassifier
    pipeline = StableDiffusionPipeline.from_single_file(model_path)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + '
    haarcascade_frontalface_default.xml')

    # Generation Process
    while True:
        emotion_index, prompt = generate_prompt()

        # Generate Images
        images = pipeline(prompt=prompt, negative_prompt=negative_prompt, width=width,
                height=height, num_inference_steps=inference_steps,
                guidance_scale=scale, num_images_per_prompt=batch_size,
                generator=generator).images

        # Save images if a face gets detected
        for image in images:
            # PIL -> OpenCV
            gray_frame = cv2.cvtColor(np.array(image), cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray_frame, scaleFactor=1.1,
                    minNeighbors=5, minSize=(30, 30))

            if len(list(faces)) > 0:
                x, y, w, h = list(faces)[0]

                # Crop Face
                ROI = gray_frame[y:y+h, x:x+w]
                ROI = cv2.resize(ROI, (64, 64))

                #cv2.imwrite("dataset_synthetic/test_images/" +
                    str(j) + "_" + str(emotion_index) + ".png", ROI)
                save_img(ROI, emotion_index, j, True)
```

6.2 Dataset Creation

```
class SyntheticExpressionDataset(torch.Dataset):
    def __init__(self, train : bool):
        print("Initializing Synthetic Dataset...")

        # Go through each sub-dir and gather images
        self.imgs_path = "dataset_synthetic/" + folder_path + "/"
        sub_folders = [name for name in os.listdir(self.imgs_path) if os.path.isdir(os
    .path.join(self.imgs_path, name))]
        self.data = []

        for i in range(len(sub_folders)):
            path = self.imgs_path + sub_folders[i] + "/"
            file_list = glob.glob(path + "*")

            for img_path in file_list:
                self.data.append([img_path, sub_folders[i]])

        # Label images accordingly
        self.class_map = {"angry" : 0,
                          "disgust": 1,
                          "fear": 2,
                          "happy": 3,
                          "neutral": 4,
                          "sad": 5,
                          "surprise": 6}
        # Set image dimensions
        self.img_dim = (48, 48)
```

6.3 Model

```
class FacialExpressionNet(torch.nn.Module):
    def __init__(self):
        super(FacialExpressionNet, self).__init__()
        # Define layers
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc_layers = nn.Sequential(
            # adj. based on the out-shape after conv layers
            nn.Linear(256 * 6 * 6, 1024),
            nn.ReLU(),
            nn.Dropout(0.5),  # Dropout for regularization
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 7)  # Output layer for 7 classes
        )
```

6.4 Realtime Inference

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + '
    haarcascade_frontalface_default.xml')
ort_session = rt.InferenceSession("trained_models/expr_net_s_synth_final.onnx")

classes = ("angry", "disgust", "fear", "happy", "neutral", "sad", "surprise")

cap = cv2.VideoCapture(0) # Used for realtime webcam capture

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(gray_frame, scaleFactor=1.1, minNeighbors=5,
     minSize=(30, 30))

    for i in range(len(list(faces))):
        if i == 0:
            x, y, w, h = list(faces)[i]

            # Get Region Of Interest (Face)
            ROI = gray_frame[y:y+h, x:x+w]
            ROI_SAFE = ROI
            ROI = cv2.resize(ROI, (48, 48))

            to_tensor = transforms.ToTensor()
            ROI = to_tensor(ROI)
            ROI.unsqueeze_(0)

            # Inference
            ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(ROI)}
            ort_outs = ort_session.run(None, ort_inputs)
            prediction = classes[np.argmax(ort_outs)]

            # UI
            cv2.putText(frame, text=prediction, org=(x, y + 20), fontFace=cv2.
    FONT_HERSHEY_SIMPLEX, fontScale=0.9, color=(0, 0, 255), thickness=2)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

# Release the capture and close all windows
cap.release()
cv2.destroyAllWindows()
```