Data Augmentation

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

The Data Hunger Problem

Deep learning models need data:

- ResNet-50 trained on 1.2M ImageNet images
- GPT-3 trained on 45TB of text
- AlphaGo trained on 30M game positions

Your reality:

- 500 labeled images
- 1,000 text samples
- 100 audio clips

Solution: Create more data from existing data through augmentation

What is Data Augmentation?

Data Augmentation: Apply transformations to existing data to create new training examples

Key Idea: Generate variations that preserve the label but increase diversity

Example (Image):

- Original: Cat image
- Rotated 10°: Still a cat
- Flipped horizontally: Still a cat
- Slightly darker: Still a cat

Benefits:

- More training data without labeling
- Better generalization
- Reduced overfitting

Why Data Augmentation Works

1. Implicit Regularization

- Model sees slightly different versions
- Learns robust features
- Reduces overfitting

2. Invariance Learning

- Model learns that rotations don't change identity
- Small color shifts don't matter
- Position in frame doesn't change class

3. Coverage of Data Distribution

- Fills gaps in training data
- Simulates real-world variations
- Tests robustness

Data Augmentation vs Data Collection

Data Collection:

- Time: Weeks to months
- Cost: High (labeling, storage)
- Effort: Manual collection and annotation
- Diversity: Limited by budget

Data Augmentation:

- Time: Minutes to hours
- Cost: Low (just compute)
- Effort: Automated transformations
- Diversity: Programmatically generated

Best Practice: Do both! Augmentation complements collection.

Image Augmentation: Geometric Transforms

Basic transformations:

1. **Rotation**: Rotate ±15-30 degrees
2. **Horizontal Flip**: Mirror image left-right
3. **Vertical Flip**: Mirror image top-bottom (use carefully)
4. **Translation**: Shift image by pixels
5. **Scaling**: Zoom in/out
6. **Shearing**: Skew image
7. **Cropping**: Random crops

Implementation with PIL:

```
from PIL import Image

img = Image.open('cat.jpg')
rotated = img.rotate(15)
flipped = img.transpose(Image.FLIP_LEFT_RIGHT)
```

Image Augmentation: Color Transforms

Color space adjustments:

1. **Brightness**: Make lighter/darker
2. **Contrast**: Increase/decrease contrast
3. **Saturation**: Make more/less colorful
4. **Hue**: Shift color spectrum
5. **Grayscale**: Convert to black and white
6. **Color Jittering**: Random color variations

```
from PIL import ImageEnhance

enhancer = ImageEnhance.Brightness(img)
brighter = enhancer.enhance(1.5)  # 50% brighter

enhancer = ImageEnhance.Contrast(img)
higher_contrast = enhancer.enhance(1.3)
```

Image Augmentation: Advanced Techniques

1. **Cutout**: Remove random patches

```
# Remove 16×16 patch
x, y = random.randint(0, w-16), random.randint(0, h-16)
img[y:y+16, x:x+16] = 0
```

2. **Mixup**: Blend two images

```
lambda_val = np.random.beta(alpha, alpha)
mixed = lambda_val * img1 + (1 - lambda_val) * img2
label = lambda_val * label1 + (1 - lambda_val) * label2
```

3. **CutMix**: Replace patch with another image
4. **AugMix**: Apply multiple augmentations and mix

Albumentations Library

Fast and flexible image augmentation library

```
import albumentations as A
from albumentations.pytorch import ToTensorV2

transform = A.Compose([
    A.RandomRotate90(),
    A.Flip(),
    A.Transpose(),
    A.GaussNoise(),
    A.OneOf([
        A.MotionBlur(p=0.2),
        A.MedianBlur(blur_limit=3, p=0.1),
        A.Blur(blur_limit=3, p=0.1),
    ], p=0.2),
    A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.2, rotate_limit=45, p=0.2),
    A.OneOf([
        A.OpticalDistortion(p=0.3),
        A.GridDistortion(p=0.1),
    ], p=0.2),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2(),
])

augmented = transform(image=image)['image']
```

Albumentations - Key Features

Why Albumentations?

1. **Fast**: Optimized with NumPy/OpenCV
2. **Flexible**: Easy to compose transformations
3. **Framework-agnostic**: Works with PyTorch, TensorFlow, etc.
4. **Preserves Bounding Boxes**: For object detection
5. **Keypoint Support**: For pose estimation

Common Augmentations:

- Geometric: Rotate, Flip, Shift, Scale
- Blur: Motion, Gaussian, Median
- Noise: Gaussian, ISO, Salt & Pepper
- Weather: Rain, Fog, Snow, Sun Flare
- Advanced: Cutout, CoarseDropout

Image Augmentation Best Practices

1. Choose Appropriate Augmentations

- Natural images: Rotation, flip, color jitter
- Medical images: Be careful with flips (anatomy matters)
- Text/OCR: No rotation, no flip (orientation matters)

2. Augmentation Strength

- Start mild, increase gradually
- Too strong: Model learns wrong patterns
- Too weak: No benefit

3. Validation Set

- Don't augment validation/test data
- Measure performance on real distribution

4. Domain-Specific

When NOT to Augment

Be careful with:

1. **Medical imaging**: Artifacts can mislead diagnosis
2. **OCR/Text**: Rotation can make text unreadable
3. **Fine-grained classification**: Too much blur loses details
4. **Small objects**: Heavy cropping loses object
5. **Asymmetric objects**: Flips change meaning (e.g., left/right lung)

Rule: Only augment if transformation preserves label

Text Augmentation: Overview

Challenges:

- Discrete tokens (can't interpolate like pixels)
- Semantic meaning matters
- Grammar and syntax constraints

Approaches:

1. **Rule-based**: Synonym replacement, random operations
2. **Back-translation**: Translate to another language and back
3. **Paraphrasing**: LLMs generate paraphrases
4. **Contextual**: BERT-based word replacement

Text Augmentation: EDA

Easy Data Augmentation (EDA) - Simple but effective

4 Operations:

1. **Synonym Replacement**: Replace words with synonyms

```
"The movie was great" → "The film was excellent"
```

2. **Random Insertion**: Insert random synonyms

```
"I love this" → "I really love this"
```

3. **Random Swap**: Swap word positions

```
"She likes pizza" → "She pizza likes"
```

4. **Random Deletion**: Delete words randomly

Text Augmentation with nlpaug

nlpaug: Comprehensive text augmentation library

```
import nlpaug.augmenter.word as naw
import nlpaug.augmenter.sentence as nas

# Synonym replacement using WordNet
aug_syn = naw.SynonymAug(aug_src='wordnet')
text = "The quick brown fox jumps over the lazy dog"
augmented = aug_syn.augment(text)
print(augmented)
# Output: "The fast brown fox jump over the lazy dog"

# Contextual word embeddings (BERT)
aug_bert = naw.ContextualWordEmbsAug(
    model_path='bert-base-uncased',
    action="substitute"
)
augmented = aug_bert.augment(text)

# Back-translation
aug_back = naw.BackTranslationAug(
    from_model_name='facebook/wmt19-en-de',
    to_model_name='facebook/wmt19-de-en'
)
augmented = aug_back.augment(text)
```

Text Augmentation: Back-Translation

Idea: Translate to another language and back

```
from transformers import pipeline

# English → German → English
en_de = pipeline("translation", model="Helsinki-NLP/opus-mt-en-de")
de_en = pipeline("translation", model="Helsinki-NLP/opus-mt-de-en")

text = "I love machine learning"
german = en_de(text)[0]['translation_text']
back = de_en(german)[0]['translation_text']

print(f"Original: {text}")
print(f"German: {german}")
print(f"Back: {back}")
# Output: "I love machine learning" → "Ich liebe maschinelles Lernen" → "I love machine learning"
```

Pros: Maintains meaning, natural variations

Cons: Expensive (requires translation models)

Text Augmentation: Paraphrasing with LLMs

Use LLMs to generate paraphrases

```
from google import genai
import os

client = genai.Client(api_key=os.environ['GEMINI_API_KEY'])

def paraphrase(text, n=3):
    prompt = f"""
    Generate {n} paraphrases of the following text.
    Keep the same meaning but use different words.
    Return one paraphrase per line.

    Text: {text}
    """

    response = client.models.generate_content(
        model="models/gemini-2.0-flash-exp",
        contents=prompt
    )

    paraphrases = response.text.strip().split('\n')
    return paraphrases

text = "The model achieved 95% accuracy"
paraphrases = paraphrase(text, n=3)
for p in paraphrases:
    print(p)
```

Text Augmentation Best Practices

1. Preserve Label

- Sentiment: Don't change positive to negative
- NER: Keep entity boundaries
- Classification: Maintain class meaning

2. Maintain Coherence

- Avoid random operations that break grammar
- Check that output is readable

3. Domain-Specific

- Legal text: Minimal changes (meaning critical)
- Social media: More aggressive OK (informal)
- Code: Very careful (syntax matters)

4. Quality Control

Audio Augmentation: Overview

Audio = Waveform + Spectrogram

Time Domain Augmentations:

- Time stretching
- Pitch shifting
- Adding noise
- Volume changes
- Time shifting

Frequency Domain Augmentations:

- SpecAugment
- Frequency masking
- Time masking

Audio Augmentation with audiomentations

```
from audiomentations import Compose, AddGaussianNoise, TimeStretch, PitchShift

augment = Compose([
    AddGaussianNoise(min_amplitude=0.001, max_amplitude=0.015, p=0.5),
    TimeStretch(min_rate=0.8, max_rate=1.25, p=0.5),
    PitchShift(min_semitones=-4, max_semitones=4, p=0.5),
])

import librosa

# Load audio
audio, sr = librosa.load('audio.wav', sr=16000)

# Augment
augmented_audio = augment(samples=audio, sample_rate=sr)

# Save
import soundfile as sf
sf.write('augmented_audio.wav', augmented_audio, sr)
```

SpecAugment for Speech Recognition

SpecAugment: Augment spectrograms directly

Operations:

1. **Time Masking**: Mask consecutive time steps
2. **Frequency Masking**: Mask frequency channels
3. **Time Warping**: Warp time axis

```
import torch
from torchaudio.transforms import FrequencyMasking, TimeMasking

# Convert to spectrogram
spectrogram = torchaudio.transforms.MelSpectrogram()(audio)

# Apply augmentations
freq_mask = FrequencyMasking(freq_mask_param=30)
time_mask = TimeMasking(time_mask_param=100)

augmented_spec = time_mask(freq_mask(spectrogram))
```

Audio Augmentation: Common Techniques

1. Background Noise

```
from audiomentations import AddBackgroundNoise

augment = AddBackgroundNoise(
    sounds_path="/path/to/noise/files",
    min_snr_db=3,
    max_snr_db=30,
    p=1.0
)
```

2. Room Impulse Response

```
from audiomentations import ApplyImpulseResponse

augment = ApplyImpulseResponse(
    ir_path="/path/to/impulse/responses",
    p=0.5
)
```

Augly: Facebook's Augmentation Library

Unified API for images, audio, video, and text

```
import augly.image as imaugs
import augly.audio as audaugs
import augly.text as textaugs

# Image
img_augmented = imaugs.augment_image(
    img,
    [
        imaugs.Blur(),
        imaugs.RandomNoise(),
        imaugs.Rotate(degrees=15),
    ]
)

# Audio
audio_augmented = audaugs.apply_lambda(
    audio,
    aug_function=audaugs.add_background_noise,
    snr_level_db=10
)

# Text
text_augmented = textaugs.simulate_typos(
    text,
    aug_char_p=0.05,
    aug_word_p=0.05
)
```

Augly Features

Cross-Modal Augmentations:

Images:

- Blur, brightness, contrast, noise
- Overlay emoji, text, shapes
- Meme generation
- Pixel distortions

Audio:

- Background noise, reverb, pitch shift
- Clipping, speed, volume
- Time stretch

Text:

- Typos, contractions, slang

Designing an Augmentation Pipeline

Step 1: Understand Your Task

- Classification: Aggressive augmentation OK
- Detection: Preserve bounding boxes
- Segmentation: Transform masks too

Step 2: Start Simple

```
# Baseline: No augmentation
# Then add one at a time
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
])
```

Step 3: Gradually Increase

```
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
```

Augmentation Hyperparameters

Key parameters to tune:

1. **Probability (p)**: How often to apply
    - Start: p=0.5
    - Increase if underfitting
    - Decrease if validation worse
2. **Magnitude**: Strength of transformation
    - Rotation: ±10° → ±30°
    - Brightness: ±10% → ±30%
3. **Combination**: How many augmentations together
    - Start: 1-2 at a time
    - Advanced: 3-5 at a time

Strategy: Grid search or random search

AutoAugment & RandAugment

AutoAugment: Learn augmentation policy with RL

Problem: Manual tuning is tedious

Solution: Use RL to find best augmentation sequence

RandAugment: Simplified version

```
from torchvision.transforms import RandAugment

transform = RandAugment(
    num_ops=2,      # Number of augmentations to apply
    magnitude=9     # Strength (0-30)
)

augmented = transform(image)
```

Policies learned on ImageNet work well on other datasets!

Test-Time Augmentation (TTA)

Idea: Augment at inference time and average predictions

```
import albumentations as A

def tta_predict(model, image, n_augments=10):
    transform = A.Compose([
        A.HorizontalFlip(p=0.5),
        A.Rotate(limit=15, p=0.5),
    ])

    predictions = []

    # Original prediction
    predictions.append(model.predict(image))

    # Augmented predictions
    for _ in range(n_augments - 1):
        aug_image = transform(image=image)['image']
        pred = model.predict(aug_image)
        predictions.append(pred)

    # Average predictions
    avg_pred = np.mean(predictions, axis=0)
    return avg_pred
```

Measuring Augmentation Effectiveness

Experiment Design:

1. **Baseline**: Train without augmentation
2. **With Aug**: Train with augmentation
3. **Compare**:
    - Training loss curves
    - Validation accuracy
    - Test accuracy
    - Overfitting gap

```
# No augmentation
model1 = train_model(train_data, augment=False)
acc_no_aug = model1.evaluate(test_data)

# With augmentation
model2 = train_model(train_data, augment=True)
acc_with_aug = model2.evaluate(test_data)
```

Data Augmentation + Active Learning

Combine both techniques:

1. **Active Learning**: Select most informative samples
2. **Data Augmentation**: Generate variations of selected samples

```
# Active learning loop with augmentation
for iteration in range(n_iterations):
    # Query uncertain samples
    query_idx = uncertainty_sampling(model, X_pool, n_samples=10)

    # Augment queried samples
    X_aug, y_aug = augment_samples(X_pool[query_idx], y_pool[query_idx])

    # Add original + augmented to training set
    X_train = np.vstack([X_train, X_pool[query_idx], X_aug])
    y_train = np.hstack([y_train, y_pool[query_idx], y_aug])

    # Retrain
    model.fit(X_train, y_train)
```

Domain-Specific Augmentation

Medical Imaging:

- Mild rotations, flips (check anatomy)
- Brightness/contrast (simulate different machines)
- Elastic deformations
- Avoid: Heavy blurs, unrealistic colors

Satellite Imagery:

- Any rotation (no canonical orientation)
- Color shifts (atmospheric conditions)
- Cloud overlays

Document OCR:

- Perspective transforms
- Shadows, creases

Synthetic Data Generation

Beyond augmentation: Generate completely new data

Techniques:

1. **GANs**: Generate realistic images
2. **Style Transfer**: Change image style
3. **3D Rendering**: Render synthetic scenes
4. **Text-to-Image**: Stable Diffusion, DALL-E
5. **Simulation**: Physics engines for robotics

Example: Car detection

- Render 3D car models in various poses
- Add backgrounds
- Train detector

Benefits: Unlimited data, perfect labels

GANs for Data Augmentation

Use trained GAN to generate new samples

```
from torchvision.models import inception_v3
import torch

# Train GAN on your dataset
# Then generate new samples

generator = load_trained_gan()

# Generate 1000 new images
z = torch.randn(1000, latent_dim)
fake_images = generator(z)

# Add to training set
X_train_augmented = torch.cat([X_train, fake_images])
```

Challenges:

- Training GANs is hard

Augmentation for Object Detection

Challenge: Must transform bounding boxes too

```
import albumentations as A

transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=10, p=0.5),
    A.RandomBrightnessContrast(p=0.3),
], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['labels']))

# Apply transformation
augmented = transform(
    image=image,
    bboxes=[[23, 45, 120, 150], [50, 80, 200, 250]],
    labels=[0, 1]
)

aug_image = augmented['image']
aug_bboxes = augmented['bboxes']
aug_labels = augmented['labels']
```

Augmentation for Semantic Segmentation

Challenge: Transform masks along with images

```
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=30, p=0.5),
    A.ElasticTransform(p=0.3),
    A.GridDistortion(p=0.3),
])

# Apply to both image and mask
augmented = transform(image=image, mask=mask)

aug_image = augmented['image']
aug_mask = augmented['mask']

# Mask is transformed identically to image
assert aug_image.shape[:2] == aug_mask.shape
```

Common Mistakes

1. Augmenting Test Data

- Only augment training data!
- Test on original distribution

2. Too Strong Augmentation

- Model learns wrong patterns
- Check augmented samples visually

3. Not Preserving Labels

- Digit '6' flipped → '9' (different label!)
- Medical: Left vs right matters

4. Inconsistent Preprocessing

- Normalize before or after augmentation?
- Be consistent in training and inference

Tools & Libraries Summary

Images:

- **Albumentations**: Fast, flexible, comprehensive
- **imgaug**: Similar to Albumentations
- **torchvision.transforms**: PyTorch native
- **Augly**: Facebook's unified library

Text:

- **nlpaug**: Comprehensive text augmentation
- **TextAugment**: EDA implementation
- **Augly.text**: Facebook's text augs

Audio:

- **audiomentations**: Time-domain augmentations
- **torchaudio.transforms**: Frequency-domain

Augmentation in Production

Considerations:

1. **Performance**: Augment on-the-fly vs pre-computed
    - On-the-fly: Saves storage, more variety
    - Pre-computed: Faster training
2. **Reproducibility**: Set random seeds

```
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
```

3. **Validation**: Don't augment val/test sets
4. **Monitoring**: Track which augmentations used
5. **A/B Testing**: Compare models with different augmentations

Research Directions

Current Trends:

1. **Learned Augmentation**: AutoML for augmentation policies
2. **Adversarial Augmentation**: Generate hard examples
3. **Curriculum Augmentation**: Start easy, increase difficulty
4. **Cross-Modal Augmentation**: Transfer between modalities
5. **Foundation Model Augmentation**: Use DALL-E, ChatGPT

Open Problems:

- Optimal augmentation for small datasets
- Task-specific augmentation design
- Augmentation for few-shot learning
- Augmentation quality metrics

Case Study: Image Classification

Dataset: CIFAR-10 (10 classes, 50k train images)

Baseline (No Augmentation):

- Train accuracy: 99%
- Test accuracy: 70%
- Clear overfitting!

With Standard Augmentation:

```
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.Rotate(limit=15, p=0.5),
])
```

- Train accuracy: 85%
- Test accuracy: 82%

What We've Learned

Core Concepts:

- Data augmentation creates training data variations
- Preserves labels while increasing diversity
- Reduces overfitting and improves generalization

Techniques:

- Image: Geometric + color transforms
- Text: Synonym replacement, back-translation, paraphrasing
- Audio: Time stretching, pitch shifting, noise

Libraries:

- Albumentations (images)
- nlpaug (text)
- audiomentations (audio)

Practical Recommendations

Getting Started:

1. Use Albumentations for images
2. Start with flip + rotate + brightness
3. Measure baseline vs augmented
4. Gradually add more augmentations

Hyperparameter Tuning:

- Probability: 0.3-0.7
- Magnitude: Start low, increase if underfitting
- Number of augs: 2-4 simultaneously

Production:

- Augment on-the-fly during training
- No augmentation at inference (unless TTA)

Resources

Papers:

- "AutoAugment: Learning Augmentation Policies from Data" (2019)
- "RandAugment: Practical automated data augmentation" (2020)
- "SpecAugment: A Simple Data Augmentation Method for ASR" (2019)
- "mixup: Beyond Empirical Risk Minimization" (2018)

Libraries:

- Albumentations: https://albumentations.ai/
- nlpaug: https://github.com/makcedward/nlpaug
- audiomentations: https://github.com/iver56/audiomentations
- Augly: https://github.com/facebookresearch/AugLy

Tutorials:

- Albumentations documentation

Questions?

Lab: Implement and compare augmentation strategies

Measure impact on model performance