

# 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

# 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

# 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

# Image Augmentation: Geometric Transforms

## Basic transformations:

1. **Rotation:** Rotate  $\pm 15$ -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
```

# 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)
```

# Image Augmentation: Advanced Techniques

## 1. Cutout: Remove random patches

```
# Remove 16x16 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(),
])
```

# 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

# 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

# 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'
```

# 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)
```

# 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)

# 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

# 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)
```

# 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, and text

```
import augly.image as imaugs
import augly.audio as audaug
import augly.text as textaug

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

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

# Text
text_augmented = textaug.simulate_typos(
    text,
    aug_prob=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



# 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

# 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:  $\pm 10^\circ \rightarrow \pm 30^\circ$
- Brightness:  $\pm 10\% \rightarrow \pm 30\%$

3. **Combination**: How many augmentations together

- Start: 1-2 at a time

# 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
```

# 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
```

# 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
```

# 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:

# 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



# 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:**

# 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']
```

# 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

# 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:

# 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

# 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%



# 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)

# 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:

# 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>

# Mathematical Foundations: Invariance and Equivariance

**Invariance:** Output doesn't change under transformation

$$f(T(x)) = f(x)$$

**Example:** Image classifier should be invariant to rotation

- $f(\text{rotate}(\text{cat})) = \text{"cat"}$

**Equivariance:** Output transforms consistently with input

$$f(T(x)) = T'(f(x))$$

**Example:** Segmentation should be equivariant to rotation

- $\text{segment}(\text{rotate}(\text{image})) = \text{rotate}(\text{segment}(\text{image}))$

**Data augmentation teaches invariance:**

# Manifold Hypothesis and Augmentation

**Manifold Hypothesis:** High-dimensional data lies on low-dimensional manifold

**Augmentation explores the manifold:**

- Original data: Sparse samples on manifold
- Augmented data: Fill gaps along manifold

**Interpolation on manifold:**

$$x_{aug} = x + \epsilon \cdot \nabla_{\theta} T(x)$$

where  $T$  is transformation,  $\epsilon$  is small

**Theoretical benefit:**

- Smoother decision boundaries
- Better generalization

# Mixup: Theory and Implementation

**Mixup:** Linear interpolation of examples and labels

**Formula:**

$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j$$

$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j$$

where  $\lambda \sim \text{Beta}(\alpha, \alpha)$

**Theoretical motivation:**

- Vicinal Risk Minimization (VRM)
- Encourages linear behavior between training examples
- Regularizes network to output convex combinations

**Implementation:**

# Mixup Variants: CutMix and MoEx

CutMix: Replace patches instead of blending

Advantages over Mixup:

- Preserves localization ability
- More efficient for CNNs (no blend artifacts)

```
def cutmix(x, y, alpha=1.0):  
    """CutMix augmentation."""  
    lam = np.random.beta(alpha, alpha)  
    batch_size, _, H, W = x.shape  
    index = torch.randperm(batch_size)  
  
    # Random box  
    cut_rat = np.sqrt(1. - lam)  
    cut_w = int(W * cut_rat)  
    cut_h = int(H * cut_rat)  
  
    cx = np.random.randint(W)  
    cy = np.random.randint(H)  
  
    bbx1 = np.clip(cx - cut_w // 2, 0, W)
```

# Diffusion Models for Data Augmentation

**Modern approach:** Use diffusion models to generate variations

**Workflow:**

1. Add small noise to image
2. Denoise with pretrained diffusion model
3. Use denoised version as augmentation

```
from diffusers import StableDiffusionImg2ImgPipeline

pipe = StableDiffusionImg2ImgPipeline.from_pretrained("runwayml/stable-diffusion-v1-5")

def diffusion_augment(image, prompt, strength=0.3):
    """Augment image using diffusion model."""
    # Strength: How much to change (0=no change, 1=complete re-generation)
    augmented = pipe(
        prompt=prompt,
        image=image,
        strength=strength,
```



# Contrastive Learning Augmentation Strategies

**SimCLR:** Self-supervised learning via contrastive loss

**Key idea:** Different augmentations of same image should have similar representations

**Augmentation composition:**

```
import torchvision.transforms as transforms

# SimCLR augmentation pipeline
simclr_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomApply([
        transforms.ColorJitter(0.8, 0.8, 0.8, 0.2)
    ], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    transforms.GaussianBlur(kernel_size=23),
    transforms.ToTensor(),
])
```

# MoCo (Momentum Contrast) Augmentation

## MoCo v2 augmentation:

```
moco_transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.2, 1.0)),
    transforms.RandomApply([
        transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)
    ], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    transforms.RandomApply([transforms.GaussianBlur(kernel_size=23)], p=0.5),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

## Queue-based approach:

- Maintain queue of negatives
- Momentum encoder for consistency

# Invariant Risk Minimization (IRM)

**Goal:** Learn invariant features across environments

**Formulation:**

$$\min_{\Phi} \sum_{e \in \mathcal{E}} R^e(\Phi) + \lambda \|\nabla_{w|w=1.0} R^e(w \cdot \Phi)\|^2$$

where:

- $\mathcal{E}$ : Set of environments (different augmentations)
- $R^e$ : Risk in environment  $e$
- $\Phi$ : Feature extractor

**Augmentation as environments:**

```
def irm_loss(model, x, y, augmentations):  
    """IRM loss across augmentation environments."""
```

# Consistency Regularization: UDA and FixMatch

## Unsupervised Data Augmentation (UDA):

**Idea:** Model predictions should be consistent under augmentation

$$L_{consistency} = \mathbb{E}_{x,aug}[KL(p(y|x)||p(y|aug(x)))]$$

```
def uda_loss(model, x_unlabeled, strong_aug, weak_aug):  
    """UDA consistency loss."""  
    # Weak augmentation prediction (pseudo-label)  
    with torch.no_grad():  
        weak_pred = model(weak_aug(x_unlabeled))  
        pseudo_label = torch.softmax(weak_pred, dim=1)  
  
    # Strong augmentation prediction  
    strong_pred = model(strong_aug(x_unlabeled))  
  
    # Consistency loss  
    loss = F.kl_div(  
        F.log_softmax(strong_pred, dim=1),
```

# Learnable Augmentation Policies

Neural Augmentation: Learn transformation parameters

Approach:

```
class LearnableAugmentation(nn.Module):
    def __init__(self):
        super().__init__()
        # Learnable parameters for augmentation
        self.rotation_range = nn.Parameter(torch.tensor(15.0))
        self.brightness_factor = nn.Parameter(torch.tensor(0.2))

    def forward(self, x):
        # Apply augmentation with learned parameters
        angle = torch.rand(1) * self.rotation_range
        brightness = 1 + torch.rand(1) * self.brightness_factor

        x_aug = rotate(x, angle)
        x_aug = adjust_brightness(x_aug, brightness)

        return x_aug

# Training: Backprop through augmentation
learnable_aug = LearnableAugmentation()
optimizer = torch.optim.Adam(learnable_aug.parameters())
```

# Adversarial Training as Augmentation

**Adversarial examples:** Inputs with small perturbations that fool model

**PGD (Projected Gradient Descent):**

$$x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x L(f(x), y))$$

**Adversarial training:**

```
def pgd_attack(model, x, y, epsilon=0.3, alpha=0.01, num_iter=10):  
    """Generate adversarial example."""  
    x_adv = x.clone().detach()  
  
    for _ in range(num_iter):  
        x_adv.requires_grad = True  
  
        # Compute loss  
        pred = model(x_adv)  
        loss = F.cross_entropy(pred, y)  
  
        # Gradient ascent  
        loss.backward()  
        grad = x_adv.grad  
  
        # Update adversarial example  
        x_adv = x_adv + alpha * grad.sign()  
  
        # Project back to epsilon ball  
        perturbation = torch.clamp(x_adv - x, -epsilon, epsilon)  
        x_adv = torch.clamp(x + perturbation, 0, 1).detach()
```

# Meta-Learning for Augmentation

**Goal:** Learn which augmentations help for specific tasks

**Meta-augmentation:**

```
class MetaAugmentation:
    def __init__(self, augmentations):
        self.augmentations = augmentations
        # Learnable weights for each augmentation
        self.weights = nn.Parameter(torch.ones(len(augmentations)))

    def sample_augmentation(self):
        """Sample augmentation based on learned weights."""
        probs = F.softmax(self.weights, dim=0)
        idx = torch.multinomial(probs, 1).item()
        return self.augmentations[idx]

    def meta_train(self, meta_train_tasks, meta_val_tasks):
        """Meta-training loop."""
        for epoch in range(num_epochs):
            for task in meta_train_tasks:
                # Sample augmentation
                aug = self.sample_augmentation()

                # Train on augmented data
                x_aug, y_aug = aug(task.x_train), task.y_train
                model.train_step(x_aug, y_aug)
```

# Augmentation Budget and Efficiency

Computational cost:

Augmentation	Cost (ms/image)	Speedup Strategy
Horizontal flip	0.1	Already fast
Rotation	2.0	Use smaller angles
Color jitter	1.5	GPU acceleration
Cutout	0.5	Already fast
Mixup	0.2	In-place operations
Back-translation	500.0	Cache results
Diffusion models	5000.0	Offline generation

Optimization strategies:



# Data Mixing Beyond Mixup

**SaliencyMix:** Mix based on saliency maps

```
def saliencymix(x, y, saliency_fn):  
    """Mix based on saliency."""  
    batch_size = x.size(0)  
    index = torch.randperm(batch_size)  
  
    # Get saliency maps  
    sal_a = saliency_fn(x)  
    sal_b = saliency_fn(x[index])  
  
    # Mix based on saliency  
    mask = (sal_a > sal_b).float()  
    mixed_x = mask * x + (1 - mask) * x[index]  
  
    # Label proportional to saliency  
    lam = mask.sum() / mask.numel()  
  
    return mixed x, y, y[index], lam
```

# Policy Search for Optimal Augmentation

## Population Based Augmentation (PBA):

### Algorithm:

1. Initialize population of augmentation policies
2. Train models with different policies
3. Select best performers
4. Mutate and combine policies
5. Repeat

```
class AugmentationPolicy:
    def __init__(self):
        self.ops = random.sample(ALL_OPS, k=5)
        self.probs = np.random.uniform(0, 1, size=5)
        self.magnitudes = np.random.uniform(0, 1, size=5)

    def mutate(self):
        """Mutate policy."""
        idx = random.randint(0, 4)
        if random.random() < 0.5:
            self.probs[idx] += np.random.normal(0, 0.1)
        else:
            self.magnitudes[idx] += np.random.normal(0, 0.1)

    def crossover(self, other):
```

# Augmentation for Long-Tail Distribution

**Problem:** Rare classes benefit more from augmentation

**Class-balanced augmentation:**

```
class ClassBalancedAugmentation:
    def __init__(self, class_counts):
        # Compute augmentation probability per class
        # More augmentation for rare classes
        total = sum(class_counts)
        self.aug_probs = {
            cls: 1.0 - (count / total)
            for cls, count in enumerate(class_counts)
        }

    def __call__(self, x, y):
        """Apply augmentation based on class."""
        aug_prob = self.aug_probs[y]

        if random.random() < aug_prob:
            # Strong augmentation for rare classes
            x = strong_augment(x)
        else:
            # Weak augmentation for common classes
            x = weak_augment(x)
```

# Temporal Augmentation for Videos

## Video-specific challenges:

- Temporal consistency
- Motion patterns
- Longer sequences

## Temporal augmentation:

```
def temporal_augment(video, fps=30):  
    """Augment video data."""  
    # 1. Temporal crop  
    start = random.randint(0, len(video) - 64)  
    video = video[start:start+64]  
  
    # 2. Temporal sub-sampling  
    stride = random.choice([1, 2])  
    video = video[::stride]
```

# 3D Augmentation for Point Clouds

## Point cloud augmentation:

```
def pointcloud_augment(points):  
    """Augment 3D point cloud."""  
    # 1. Random rotation  
    angle = np.random.uniform(0, 2*np.pi)  
    rotation_matrix = np.array([  
        [np.cos(angle), -np.sin(angle), 0],  
        [np.sin(angle), np.cos(angle), 0],  
        [0, 0, 1]  
    ])  
    points = points @ rotation_matrix.T  
  
    # 2. Random scaling  
    scale = np.random.uniform(0.8, 1.2)  
    points = points * scale  
  
    # 3. Random jitter  
    noise = np.random.normal(0, 0.02, size=points.shape)  
    points = points + noise  
  
    # 4. Random point dropout  
    keep_mask = np.random.random(len(points)) > 0.1  
    points = points[keep_mask]
```

# Graph Augmentation for GNNs

## Graph-specific augmentation:

```
def graph_augment(graph):  
    """Augment graph structure."""  
    # 1. Edge dropping  
    edge_mask = torch.rand(graph.num_edges) > 0.1  
    graph.edge_index = graph.edge_index[:, edge_mask]  
  
    # 2. Node dropping  
    node_mask = torch.rand(graph.num_nodes) > 0.1  
    graph = graph.subgraph(node_mask)  
  
    # 3. Feature masking  
    feat_mask = torch.rand(graph.x.size(1)) > 0.2  
    graph.x[:, ~feat_mask] = 0  
  
    # 4. Edge perturbation (add random edges)  
    n_new_edges = int(0.1 * graph.num_edges)  
    src = torch.randint(0, graph.num_nodes, (n_new_edges,))  
    dst = torch.randint(0, graph.num_nodes, (n_new_edges,))  
    new_edges = torch.stack([src, dst])  
    graph.edge_index = torch.cat([graph.edge_index, new_edges], dim=1)
```

# Augmentation Evaluation Metrics

How to measure augmentation quality?

## 1. Downstream Performance:

```
def evaluate_augmentation(aug_fn, model, data):  
    """Evaluate by downstream task performance."""  
    # Train with augmentation  
    model_aug = train_model(data, augmentation=aug_fn)  
    acc_aug = evaluate(model_aug, test_data)  
  
    # Train without augmentation  
    model_no_aug = train_model(data, augmentation=None)  
    acc_no_aug = evaluate(model_no_aug, test_data)  
  
    improvement = acc_aug - acc_no_aug  
    return improvement
```

## 2. Diversity Score:

# Curriculum Augmentation

Idea: Start with weak augmentation, gradually increase strength

Progressive augmentation:

```
class CurriculumAugmentation:
    def __init__(self, max_epochs):
        self.max_epochs = max_epochs
        self.current_epoch = 0

    def get_augmentation(self):
        """Return augmentation based on training progress."""
        # Linearly increase augmentation strength
        progress = self.current_epoch / self.max_epochs

        if progress < 0.3:
            # Early: weak augmentation
            return A.Compose([
                A.HorizontalFlip(p=0.5),
            ])
        elif progress < 0.7:
            # Mid: medium augmentation
            return A.Compose([
                A.HorizontalFlip(p=0.5),
                A.Rotate(limit=15, p=0.5),
                A.RandomBrightnessContrast(p=0.3),
            ])
        else:
            # Late: strong augmentation
            return A.Compose([
                A.HorizontalFlip(p=0.5),
                A.Rotate(limit=30, p=0.5),
                A.RandomBrightnessContrast(p=0.5),
                A.GaussNoise(p=0.3),
                A.Cutout(num_holes=8, max_h_size=16, max_w_size=16, p=0.5),
            ])

    def update_epoch(self, epoch):
        self.current_epoch = epoch
```

```
# Training loop
cur_aug = CurriculumAugmentation(max_epochs=100)
```



# Multi-Modal Augmentation

**Cross-modal augmentation:** Augment multiple modalities consistently

**Example:** Image + Text

```
def multimodal_augment(image, caption):  
    """Augment image and caption together."""  
    # Image augmentation  
    if random.random() < 0.5:  
        image = horizontal_flip(image)  
        # Update caption if needed  
        # "person on left" → "person on right"  
        caption = flip_spatial_words(caption)  
  
    # Color augmentation  
    if random.random() < 0.3:  
        image = grayscale(image)  
        # Update caption: remove color words  
        caption = remove_color_adjectives(caption)
```

# Foundation Model-Based Augmentation

## Stable Diffusion for augmentation:

```
from diffusers import StableDiffusionPipeline

pipe = StableDiffusionPipeline.from_pretrained("stabilityai/stable-diffusion-2-1")

def foundation_augment(original_image, class_name, n_augments=5):
    """Generate augmented images using Stable Diffusion."""
    # Create prompt from class name
    prompts = [
        f"a photo of a {class_name}",
        f"a {class_name} in different lighting",
        f"a {class_name} from different angle",
        f"a {class_name} in different background",
        f"a high quality photo of a {class_name}",
    ]

    augmented_images = []
    for prompt in prompts[:n_augments]:
        # Generate with guidance from original image
        image = pipe(
            prompt=prompt,
            image=original_image,
            strength=0.5, # How much to change
        ).images[0]

        augmented_images.append(image)

    return augmented_images
```

# Augmentation Transferability

**Question:** Do augmentations learned on one dataset transfer to others?

**Empirical findings:**

**ImageNet → Other Vision Tasks:**

- AutoAugment policies from ImageNet work well on CIFAR, SVHN
- **Transferability:** ~80-90% of performance

**Natural Images → Medical Images:**

- Standard augmentations (rotation, flip) transfer well
- Advanced (CutMix, MixUp) less effective
- **Transferability:** ~60-70%

**Practical implications:**

# Advanced Augmentation Summary

## Theoretical Foundations:

- Invariance and equivariance
- Manifold hypothesis
- Vicinal risk minimization (Mixup)
- Consistency regularization

## Advanced Mixing Strategies:

- Mixup, CutMix, MoEx
- SaliencyMix, PuzzleMix
- Class-balanced mixing

## Modern Approaches:

# Questions?

Lab: Implement and compare augmentation strategies

Measure impact on model performance