Data Augmentation Lab

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Overview

By the end of this lab, you will:

- Master image augmentation with Albumentations
- Implement text augmentation with nlpaug
- Apply audio augmentation techniques
- Build augmentation pipelines
- Measure impact on model performance
- Compare different augmentation strategies

Structure:

- Part 1: Image Augmentation (60 min)
- Part 2: Text Augmentation (45 min)
- Part 3: Audio Augmentation (45 min)

Setup (10 minutes)

Install packages:

```
pip install albumentations opencv-python matplotlib pillow
pip install nlpaug torch torchvision
pip install audiomentations librosa soundfile
pip install scikit-learn pandas
```

Verify installation:

```
import albumentations as A
import nlpaug.augmenter.word as naw
from audiomentations import Compose, AddGaussianNoise
import cv2
import matplotlib.pyplot as plt

print("All packages installed successfully!")
```

Exercise 1.1: Load and Visualize Image (10 min)

Task: Load an image and display it

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Load image (BGR format)
image = cv2.imread('sample_image.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB

def show_image(image, title="Image"):
    plt.figure(figsize=(8, 6))
    plt.imshow(image)
    plt.title(title)
    plt.axis('off')
    plt.show()

show_image(image, "Original Image")

# If you don't have an image, download one
from PIL import Image
import requests
from io import BytesIO

url = "https://upload.wikimedia.org/wikipedia/commons/thumb/3/3a/Cat03.jpg/1200px-Cat03.jpg"
response = requests.get(url)
image = Image.open(BytesIO(response.content))
image = np.array(image)

show_image(image, "Downloaded Image")
```

Exercise 1.2: Basic Geometric Transforms (15 min)

Task: Apply rotation, flip, and crop

```
import albumentations as A

# Define transformations
transform_rotate = A.Rotate(limit=45, p=1.0)
transform_flip = A.HorizontalFlip(p=1.0)
transform_crop = A.RandomCrop(height=256, width=256, p=1.0)

# Apply transformations
rotated = transform_rotate(image=image)['image']
flipped = transform_flip(image=image)['image']
cropped = transform_crop(image=image)['image']

# Visualize
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes[0, 0].imshow(image)
axes[0, 0].set_title('Original')
axes[0, 0].axis('off')

axes[0, 1].imshow(rotated)
axes[0, 1].set_title('Rotated')
axes[0, 1].axis('off')

axes[1, 0].imshow(flipped)
axes[1, 0].set_title('Flipped')
axes[1, 0].axis('off')

axes[1, 1].imshow(cropped)
axes[1, 1].set_title('Cropped')
axes[1, 1].axis('off')

plt.tight_layout()
plt.show()
```

Exercise 1.3: Color Transforms (15 min)

Task: Adjust brightness, contrast, and saturation

```
# Define color transformations
transform_bright = A.RandomBrightness(limit=0.3, p=1.0)
transform_contrast = A.RandomContrast(limit=0.3, p=1.0)
transform_hue = A.HueSaturationValue(
    hue_shift_limit=20,
    sat_shift_limit=30,
    val_shift_limit=20,
    p=1.0
)

# Apply
brighter = transform_bright(image=image)['image']
higher_contrast = transform_contrast(image=image)['image']
color_shifted = transform_hue(image=image)['image']

# Visualize
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes[0, 0].imshow(image)
axes[0, 0].set_title('Original')
axes[0, 0].axis('off')

axes[0, 1].imshow(brighter)
axes[0, 1].set_title('Brighter')
axes[0, 1].axis('off')

axes[1, 0].imshow(higher_contrast)
axes[1, 0].set_title('Higher Contrast')
axes[1, 0].axis('off')

axes[1, 1].imshow(color_shifted)
axes[1, 1].set_title('Color Shifted')
axes[1, 1].axis('off')

plt.tight_layout()
plt.show()
```

Exercise 1.4: Compose Multiple Transforms (15 min)

Task: Build augmentation pipeline

```
# Compose multiple transformations
transform = A.Compose([
    A.RandomRotate90(p=0.5),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.HueSaturationValue(hue_shift_limit=10, sat_shift_limit=20, val_shift_limit=10, p=0.5),
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),
])

# Generate 6 augmented versions
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

for i, ax in enumerate(axes.flat):
    augmented = transform(image=image)['image']
    ax.imshow(augmented)
    ax.set_title(f'Augmented {i+1}')
    ax.axis('off')

plt.tight_layout()
plt.savefig('augmented_examples.png', dpi=150)
plt.show()

print("Notice how each augmentation is different!")
```

Exercise 1.5: Advanced Augmentations (15 min)

Task: Apply blur, noise, and cutout

```
# Advanced transformations
transform_advanced = A.Compose([
    A.OneOf([
        A.MotionBlur(blur_limit=5, p=1.0),
        A.MedianBlur(blur_limit=5, p=1.0),
        A.GaussianBlur(blur_limit=5, p=1.0),
    ], p=0.5),

    A.OneOf([
        A.GaussNoise(var_limit=(10, 50), p=1.0),
        A.ISONoise(p=1.0),
    ], p=0.3),

    A.CoarseDropout(
        max_holes=8,
        max_height=32,
        max_width=32,
        fill_value=0,
        p=0.5
    ),
])

# Generate examples
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

for i, ax in enumerate(axes.flat):
    augmented = transform_advanced(image=image)['image']
    ax.imshow(augmented)
    ax.set_title(f'Advanced Aug {i+1}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```

Exercise 1.6: Train with Augmentation (15 min)

Task: Compare model performance with/without augmentation

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import cv2

# Load digits dataset
digits = load_digits()
X, y = digits.data, digits.target

# Use only 2 classes
mask = (y == 3) | (y == 8)
X, y = X[mask], y[mask]

# Reshape to 8x8 images
X_images = X.reshape(-1, 8, 8)

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X_images, y, test_size=0.3, random_state=42
)

def augment_digit(image, n_augments=5):
    """Generate augmented versions of digit image."""
    # Resize for augmentation
    img_large = cv2.resize(image.astype(np.uint8), (32, 32))

    transform = A.Compose([
        A.Rotate(limit=15, p=0.7),
        A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.1, rotate_limit=15, p=0.7),
        A.GaussNoise(var_limit=(1, 5), p=0.3),
    ])

    augmented = []
    for _ in range(n_augments):
        aug = transform(image=img_large)['image']
        # Resize back to 8x8
        aug_small = cv2.resize(aug, (8, 8))
        augmented.append(aug_small.flatten())

    return np.array(augmented)
```

Exercise 1.7: Evaluate Augmentation Impact (15 min)

```
# Train WITHOUT augmentation
model_no_aug = LogisticRegression(max_iter=1000, random_state=42)
model_no_aug.fit(X_train.reshape(len(X_train), -1), y_train)
acc_no_aug = model_no_aug.score(X_test.reshape(len(X_test), -1), y_test)

print(f"Accuracy without augmentation: {acc_no_aug:.3f}")

# Train WITH augmentation
X_train_aug = []
y_train_aug = []

for img, label in zip(X_train, y_train):
    # Add original
    X_train_aug.append(img.flatten())
    y_train_aug.append(label)

    # Add augmented versions
    aug_images = augment_digit(img, n_augments=3)
    for aug_img in aug_images:
        X_train_aug.append(aug_img)
        y_train_aug.append(label)

X_train_aug = np.array(X_train_aug)
y_train_aug = np.array(y_train_aug)

print(f"Training set size: {len(X_train)} → {len(X_train_aug)}")

model_with_aug = LogisticRegression(max_iter=1000, random_state=42)
model_with_aug.fit(X_train_aug, y_train_aug)
acc_with_aug = model_with_aug.score(X_test.reshape(len(X_test), -1), y_test)

print(f"Accuracy with augmentation: {acc_with_aug:.3f}")
print(f"Improvement: {acc_with_aug - acc_no_aug:.3f} ({(acc_with_aug - acc_no_aug)/acc_no_aug*100:.1f}%)")
```

Exercise 2.1: Text Augmentation - EDA (15 min)

Task: Implement Easy Data Augmentation

```
import nlpaug.augmenter.word as naw
import nlpaug.augmenter.char as nac

texts = [
    "The movie was absolutely fantastic!",
    "I really enjoyed the performance.",
    "This product is amazing and high quality.",
]

# Synonym replacement
aug_syn = naw.SynonymAug(aug_src='wordnet')

print("=== Synonym Replacement ===")
for text in texts:
    augmented = aug_syn.augment(text, n=2)  # Generate 2 versions
    print(f"Original: {text}")
    for i, aug_text in enumerate(augmented, 1):
        print(f"  Aug {i}: {aug_text}")
    print()

# Random insertion
aug_insert = naw.SynonymAug(aug_src='wordnet', aug_min=1)

print("=== Random Insertion ===")
for text in texts:
    augmented = aug_insert.augment(text)
    print(f"Original: {text}")
    print(f"Augmented: {augmented}")
    print()
```

Exercise 2.2: Character-Level Augmentation (10 min)

Task: Simulate typos and keyboard errors

```
# Character augmentation (typos)
aug_char_sub = nac.KeyboardAug(aug_char_p=0.1)
aug_char_del = nac.RandomCharAug(action="delete", aug_char_p=0.1)

texts = [
    "Please review this document carefully.",
    "The meeting is scheduled for tomorrow.",
]

print("=== Character Substitution (Typos) ===")
for text in texts:
    augmented = aug_char_sub.augment(text, n=3)
    print(f"Original: {text}")
    for i, aug_text in enumerate(augmented, 1):
        print(f"  Aug {i}: {aug_text}")
    print()

print("=== Random Character Deletion ===")
for text in texts:
    augmented = aug_char_del.augment(text, n=2)
    print(f"Original: {text}")
    for i, aug_text in enumerate(augmented, 1):
        print(f"  Aug {i}: {aug_text}")
    print()
```

Exercise 2.3: Back-Translation (15 min)

Task: Augment text via translation

```
# Back-translation (slower, requires internet)
# Install: pip install transformers sentencepiece

try:
    aug_back_trans = naw.BackTranslationAug(
        from_model_name='facebook/wmt19-en-de',
        to_model_name='facebook/wmt19-de-en',
        device='cpu'
    )

    texts = [
        "Machine learning is transforming technology.",
        "Data augmentation improves model performance.",
    ]

    print("=== Back-Translation ===")
    for text in texts:
        print(f"Original: {text}")
        augmented = aug_back_trans.augment(text)
        print(f"Augmented: {augmented}")
        print()

except Exception as e:
    print(f"Back-translation failed (requires models download): {e}")
    print("Skipping this augmentation...")
```

Exercise 2.4: Contextual Word Embeddings (15 min)

Task: Use BERT for context-aware augmentation

```
# BERT-based substitution (context-aware)
try:
    aug_bert = naw.ContextualWordEmbsAug(
        model_path='bert-base-uncased',
        action="substitute",
        device='cpu'
    )

    texts = [
        "The cat sat on the mat.",
        "Python is a programming language.",
    ]

    print("=== BERT Contextual Substitution ===")
    for text in texts:
        print(f"Original: {text}")
        augmented = aug_bert.augment(text, n=3)
        for i, aug_text in enumerate(augmented, 1):
            print(f"  Aug {i}: {aug_text}")
        print()

except Exception as e:
    print(f"BERT augmentation failed (requires model download): {e}")
    print("Skipping this augmentation...")
```

Exercise 2.5: Measure Text Augmentation Impact (15 min)

Task: Train sentiment classifier with augmented data

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Sample sentiment data
texts = [
    "I love this product",
    "This is amazing",
    "Great quality",
    "Excellent service",
    "I hate this",
    "Terrible experience",
    "Very disappointed",
    "Waste of money",
] * 10  # Repeat to have more samples

labels = [1, 1, 1, 1, 0, 0, 0, 0] * 10

X_train, X_test, y_train, y_test = train_test_split(
    texts, labels, test_size=0.3, random_state=42
)

# Baseline: No augmentation
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_vec, y_train)
acc_baseline = model.score(X_test_vec, y_test)

print(f"Baseline accuracy: {acc_baseline:.3f}")

# With augmentation
aug = naw.SynonymAug(aug_src='wordnet')
X_train_aug = list(X_train)
y_train_aug = list(y_train)

for text, label in zip(X_train, y_train):
    try:
        aug_texts = aug.augment(text, n=2)
        for aug_text in aug_texts:
            X_train_aug.append(aug_text)
            y_train_aug.append(label)
    except:
        pass

print(f"Training size: {len(X_train)} → {len(X_train_aug)}")

vectorizer_aug = TfidfVectorizer()
X_train_aug_vec = vectorizer_aug.fit_transform(X_train_aug)
X_test_aug_vec = vectorizer_aug.transform(X_test)
```

Exercise 3.1: Audio Augmentation - Add Noise (15 min)

Task: Add Gaussian noise to audio

```
from audiomentations import Compose, AddGaussianNoise, TimeStretch, PitchShift
import librosa
import librosa.display
import soundfile as sf
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic audio (sine wave)
def generate_test_audio(duration=2, sr=16000):
    t = np.linspace(0, duration, int(sr * duration))
    audio = np.sin(2 * np.pi * 440 * t)  # 440 Hz (A4 note)
    return audio, sr

audio, sr = generate_test_audio()

# Add Gaussian noise
augment_noise = AddGaussianNoise(min_amplitude=0.001, max_amplitude=0.015, p=1.0)
audio_noisy = augment_noise(samples=audio, sample_rate=sr)

# Visualize
fig, axes = plt.subplots(2, 1, figsize=(12, 6))

axes[0].plot(audio[:1000])
axes[0].set_title('Original Audio')
axes[0].set_xlabel('Sample')
axes[0].set_ylabel('Amplitude')

axes[1].plot(audio_noisy[:1000])
axes[1].set_title('Audio with Gaussian Noise')
axes[1].set_xlabel('Sample')
axes[1].set_ylabel('Amplitude')

plt.tight_layout()
plt.show()
```

Exercise 3.2: Time Stretch and Pitch Shift (15 min)

Task: Change speed and pitch of audio

```
# Time stretching (change speed without changing pitch)
augment_time = TimeStretch(min_rate=0.8, max_rate=1.2, p=1.0)
audio_stretched = augment_time(samples=audio, sample_rate=sr)

# Pitch shifting (change pitch without changing speed)
augment_pitch = PitchShift(min_semitones=-4, max_semitones=4, p=1.0)
audio_pitched = augment_pitch(samples=audio, sample_rate=sr)

print(f"Original length: {len(audio)} samples")
print(f"Stretched length: {len(audio_stretched)} samples")
print(f"Pitched length: {len(audio_pitched)} samples")

# Visualize spectrograms
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

# Original
D = librosa.amplitude_to_db(np.abs(librosa.stft(audio)), ref=np.max)
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz', ax=axes[0])
axes[0].set_title('Original')

# Time stretched
D_stretch = librosa.amplitude_to_db(np.abs(librosa.stft(audio_stretched)), ref=np.max)
librosa.display.specshow(D_stretch, sr=sr, x_axis='time', y_axis='hz', ax=axes[1])
axes[1].set_title('Time Stretched')

# Pitch shifted
D_pitch = librosa.amplitude_to_db(np.abs(librosa.stft(audio_pitched)), ref=np.max)
librosa.display.specshow(D_pitch, sr=sr, x_axis='time', y_axis='hz', ax=axes[2])
axes[2].set_title('Pitch Shifted')

plt.tight_layout()
plt.show()

# Save
sf.write('audio_stretched.wav', audio_stretched, sr)
sf.write('audio_pitched.wav', audio_pitched, sr)
```

Exercise 3.3: Compose Audio Augmentations (15 min)

Task: Build audio augmentation pipeline

```
# Compose multiple augmentations
audio_transform = 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),
])

# Generate multiple augmented versions
fig, axes = plt.subplots(3, 2, figsize=(15, 10))

for i, ax in enumerate(axes.flat):
    if i == 0:
        # Show original
        ax.plot(audio[:1000])
        ax.set_title('Original')
    else:
        # Show augmented
        audio_aug = audio_transform(samples=audio, sample_rate=sr)
        ax.plot(audio_aug[:1000])
        ax.set_title(f'Augmented {i}')

    ax.set_xlabel('Sample')
    ax.set_ylabel('Amplitude')

plt.tight_layout()
plt.show()

print("Each augmentation is a random combination of transforms!")
```

Exercise 4.1: Augmentation Comparison (15 min)

Task: Compare different augmentation strategies

```
import pandas as pd

def evaluate_augmentation_strategy(transform, dataset, model_class, n_augments=3):
    """
    Evaluate model with specific augmentation strategy.
    """
    X_train, X_test, y_train, y_test = dataset

    # Train without augmentation
    model_baseline = model_class()
    model_baseline.fit(X_train.reshape(len(X_train), -1), y_train)
    acc_baseline = model_baseline.score(X_test.reshape(len(X_test), -1), y_test)

    # Train with augmentation
    X_aug = [X_train.reshape(len(X_train), -1)]
    y_aug = [y_train]

    for img in X_train:
        img_large = cv2.resize(img.astype(np.uint8), (32, 32))
        for _ in range(n_augments):
            augmented = transform(image=img_large)['image']
            augmented_small = cv2.resize(augmented, (8, 8))
            X_aug.append(augmented_small.flatten().reshape(1, -1))
            y_aug.append(y_train[0])  # Duplicate label

    X_aug = np.vstack(X_aug)
    y_aug = np.hstack(y_aug)

    model_aug = model_class()
    model_aug.fit(X_aug, y_aug)
    acc_aug = model_aug.score(X_test.reshape(len(X_test), -1), y_test)

    return acc_baseline, acc_aug

# Load digits dataset
digits = load_digits()
X, y = digits.data, digits.target
mask = (y == 3) | (y == 8)
X, y = X[mask], y[mask]
X_images = X.reshape(-1, 8, 8)

dataset = train_test_split(X_images, y, test_size=0.3, random_state=42)

# Test different strategies
strategies = {
    "Flip + Rotate": A.Compose([
        A.HorizontalFlip(p=0.5),
        A.Rotate(limit=15, p=0.7),
    ]),
    "Color + Noise": A.Compose([
        A.RandomBrightnessContrast(p=0.5),
        A.GaussNoise(var_limit=(1, 5), p=0.3),
    ]),
    "Geometric": A.Compose([
        A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.1, rotate_limit=15, p=0.7),
    ]),
}

results = []
for name, transform in strategies.items():
    baseline, augmented = evaluate_augmentation_strategy(
        transform, dataset, LogisticRegression
    )
    results.append({
        "Strategy": name,
        "Baseline": baseline,
        "Augmented": augmented,
        "Improvement": augmented - baseline
    })
    print(f"{name}: {baseline:.3f} → {augmented:.3f} (+{augmented - baseline:.3f})")

results_df = pd.DataFrame(results)
print("\n", results_df)
```

Exercise 4.2: Visualize Augmentation Effect (15 min)

Task: Create before/after comparison

```
def visualize_augmentation_progression(image, transform, n_steps=5):
    """
    Show effect of augmentation at different intensities.
    """
    fig, axes = plt.subplots(1, n_steps + 1, figsize=(20, 4))

    # Original
    axes[0].imshow(image)
    axes[0].set_title('Original')
    axes[0].axis('off')

    # Apply transform with increasing intensity
    for i in range(1, n_steps + 1):
        augmented = transform(image=image)['image']
        axes[i].imshow(augmented)
        axes[i].set_title(f'Augmented {i}')
        axes[i].axis('off')

    plt.tight_layout()
    plt.savefig('augmentation_progression.png', dpi=150)
    plt.show()

# Test with different transforms
transform_strong = A.Compose([
    A.Rotate(limit=30, p=1.0),
    A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=1.0),
    A.GaussNoise(var_limit=(20, 50), p=0.5),
])

visualize_augmentation_progression(image, transform_strong, n_steps=5)
```

Bonus Exercise 1: Object Detection Augmentation (20 min)

Task: Augment images with bounding boxes

```
# Create sample image with bounding boxes
img_height, img_width = image.shape[:2]

# Define bounding boxes (pascal_voc format: [x_min, y_min, x_max, y_max])
bboxes = [
    [50, 50, 200, 200],
    [250, 100, 400, 300],
]
labels = [0, 1]  # Class labels

# Transform for object detection
transform_bbox = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=15, p=0.5),
    A.RandomBrightnessContrast(p=0.3),
], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['labels']))

# Apply transformation
augmented_result = transform_bbox(
    image=image,
    bboxes=bboxes,
    labels=labels
)

aug_image = augmented_result['image']
aug_bboxes = augmented_result['bboxes']
aug_labels = augmented_result['labels']

# Visualize
def draw_bboxes(img, boxes, labels_list):
    img_copy = img.copy()
    for bbox, label in zip(boxes, labels_list):
        x_min, y_min, x_max, y_max = map(int, bbox)
        cv2.rectangle(img_copy, (x_min, y_min), (x_max, y_max), (255, 0, 0), 2)
        cv2.putText(img_copy, f'Class {label}', (x_min, y_min - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
    return img_copy

fig, axes = plt.subplots(1, 2, figsize=(15, 7))

axes[0].imshow(draw_bboxes(image, bboxes, labels))
axes[0].set_title('Original with Bboxes')
axes[0].axis('off')

axes[1].imshow(draw_bboxes(aug_image, aug_bboxes, aug_labels))
axes[1].set_title('Augmented with Transformed Bboxes')
axes[1].axis('off')
```

Bonus Exercise 2: Test-Time Augmentation (20 min)

Task: Implement TTA for better predictions

```
def tta_predict(model, image, transform, n_augments=10):
    """
    Test-Time Augmentation: Average predictions over augmented versions.
    """
    predictions = []

    # Original prediction
    img_flat = image.flatten().reshape(1, -1)
    pred = model.predict_proba(img_flat)[0]
    predictions.append(pred)

    # Augmented predictions
    for _ in range(n_augments - 1):
        img_large = cv2.resize(image.astype(np.uint8), (32, 32))
        augmented = transform(image=img_large)['image']
        augmented_small = cv2.resize(augmented, (8, 8))
        aug_flat = augmented_small.flatten().reshape(1, -1)
        pred = model.predict_proba(aug_flat)[0]
        predictions.append(pred)

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

# Train a model
model = LogisticRegression(random_state=42, max_iter=1000)
model.fit(X_train.reshape(len(X_train), -1), y_train)

# Test TTA
transform_tta = A.Compose([
    A.Rotate(limit=10, p=0.7),
    A.GaussNoise(var_limit=(1, 3), p=0.3),
])

# Test on a few samples
for i in range(3):
    test_img = X_test[i]
    true_label = y_test[i]

    # Standard prediction
    standard_pred = model.predict_proba(test_img.reshape(1, -1))[0]

    # TTA prediction
    tta_pred, all_preds = tta_predict(model, test_img, transform_tta, n_augments=10)

    print(f"\nSample {i+1} (True label: {true_label})")
    print(f"Standard prediction: {standard_pred}")
    print(f"TTA prediction: {tta_pred}")
    print(f"TTA std: {np.std(all_preds, axis=0)}")
    print(f"Confidence boost: {np.max(tta_pred) - np.max(standard_pred):.3f}")
```

Bonus Exercise 3: AutoAugment with RandAugment (20 min)

Task: Use learned augmentation policies

```
# Note: RandAugment is typically used with PyTorch/TensorFlow
# Here's a simplified version using Albumentations

def rand_augment(image, n_ops=2, magnitude=9):
    """
    Simplified RandAugment implementation.

    Args:
        n_ops: Number of augmentation operations to apply
        magnitude: Strength of augmentations (0-10)
    """
    # Scale magnitude to augmentation parameters
    scale = magnitude / 10.0

    # Available operations
    operations = [
        A.Rotate(limit=int(30 * scale), p=1.0),
        A.ShiftScaleRotate(shift_limit=scale * 0.1, scale_limit=scale * 0.1,
                           rotate_limit=0, p=1.0),
        A.HueSaturationValue(hue_shift_limit=int(20 * scale),
                             sat_shift_limit=int(30 * scale),
                             val_shift_limit=int(20 * scale), p=1.0),
        A.RandomBrightnessContrast(brightness_limit=scale * 0.3,
                                   contrast_limit=scale * 0.3, p=1.0),
        A.GaussNoise(var_limit=(10 * scale, 50 * scale), p=1.0),
    ]

    # Randomly select n_ops operations
    selected_ops = np.random.choice(len(operations), size=n_ops, replace=False)

    # Apply selected operations
    result = image.copy()
    for op_idx in selected_ops:
        op = operations[op_idx]
        result = op(image=result)['image']

    return result

# Test RandAugment with different magnitudes
fig, axes = plt.subplots(2, 4, figsize=(16, 8))

for i, magnitude in enumerate([3, 5, 7, 9]):
    # n_ops = 2
    aug1 = rand_augment(image, n_ops=2, magnitude=magnitude)
    aug2 = rand_augment(image, n_ops=2, magnitude=magnitude)

    axes[0, i].imshow(aug1)
    axes[0, i].set_title(f'Magnitude {magnitude} (1)')
    axes[0, i].axis('off')

    axes[1, i].imshow(aug2)
```

Deliverables

Submit the following:

1. Jupyter notebook with all exercises
2. Visualizations showing:
    - Image augmentation examples (6+ transforms)
    - Before/after comparisons
    - Model performance with/without augmentation
3. Analysis report including:
    - Which augmentation strategy worked best
    - Performance improvements measured
    - Recommendations for different data types
4. Saved augmented samples (images, audio)

Bonus:

Testing Checklist

Before submission:

- [ ] Image augmentations work correctly
- [ ] Text augmentations generate valid sentences
- [ ] Audio augmentations preserve recognizability
- [ ] Model performance improves with augmentation
- [ ] Visualizations are clear and saved
- [ ] Code is well-commented
- [ ] Augmentation pipelines are reproducible
- [ ] Baseline comparison completed
- [ ] Analysis report written

Common Issues and Solutions

Issue: Augmentations too strong

- Reduce probability (p)
- Decrease magnitude/limits
- Visually inspect augmented samples

Issue: No performance improvement

- Dataset might be too easy
- Try different augmentations
- Increase training epochs
- Check if model is too simple

Issue: Memory errors

- Don't store all augmented data

Resources

Libraries:

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

Documentation:

- Albumentations docs: https://albumentations.ai/docs/
- torchvision transforms: https://pytorch.org/vision/stable/transforms.html

Tutorials:

- Albumentations examples
- Kaggle augmentation notebooks

Excellent Work!

You've mastered data augmentation across multiple modalities!

Course Complete!

Questions? Office hours: Tomorrow 3-5 PM