Basketball Training Dataset Preparation Guide

Overview

This guide provides step-by-step instructions for preparing the basketball training dataset for model training, including annotation, augmentation, and upload to training platforms.

Dataset Location: `/home/ubuntu/basketball_app/training_data/`
Total Images: 7,280
Status: ✅ Collection Complete | ⏳ Annotation Pending

Table of Contents

Quick Start

Prerequisites

```
# Install required Python packages
pip install opencv-python pillow imagehash roboflow labelimg

# Install annotation tools
# Option 1: CVAT (Docker-based)
docker pull cvat/server

# Option 2: Label Studio
pip install label-studio

# Option 3: RoboFlow (Web-based - no installation)
# Visit: https://roboflow.com
```

Directory Structure

```
training_data/
├── shooting_form_keypoints/
│   ├── professional/           # 773 images
│   ├── amateur/                # 28 images
│   ├── front_view/             # 480 images
│   ├── side_view/              # 252 images
│   └── 45_degree/              # 198 images
├── form_quality_classifier/
│   ├── excellent_form/         # 300 images
│   ├── good_form/              # 28 images
│   ├── needs_work/             # 15 images
│   └── poor_form/              # 10 images
├── ball_trajectory/
│   ├── jump_shots/             # 300 images
│   ├── free_throws/            # 200 images
│   └── various_angles/         # 4,696 images
├── raw_downloads/              # Original downloaded data
├── scripts/                    # Utility scripts
├── DATASET_SOURCES.md
├── DATASET_SUMMARY.md
└── DATASET_PREPARATION_GUIDE.md (this file)
```

Data Quality Control

Step 1: Remove Duplicates

```
cd /home/ubuntu/basketball_app/training_data

# Run duplicate detection script
python3 scripts/remove_duplicates.py
```

Script: `scripts/remove_duplicates.py`

```
#!/usr/bin/env python3
import os
from pathlib import Path
import imagehash
from PIL import Image
from collections import defaultdict

def find_duplicates(directory):
    \"\"\"Find duplicate images using perceptual hashing\"\"\"
    hashes = defaultdict(list)

    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                filepath = Path(root) / file
                try:
                    img_hash = imagehash.average_hash(Image.open(filepath))
                    hashes[str(img_hash)].append(filepath)
                except Exception as e:
                    print(f\"Error processing {filepath}: {e}\")

    duplicates = {h: files for h, files in hashes.items() if len(files) > 1}
    return duplicates

if __name__ == \"__main__\":
    base_dir = Path(\"/home/ubuntu/basketball_app/training_data\")

    # Check each category
    for category in [\"shooting_form_keypoints\", \"form_quality_classifier\", \"ball_
trajectory\"]:
        category_path = base_dir / category
        print(f\"\\nChecking {category}...\")

        duplicates = find_duplicates(category_path)

        if duplicates:
            print(f\"Found {len(duplicates)} duplicate groups:\")
            for hash_val, files in duplicates.items():
                print(f\"  Hash {hash_val}: {len(files)} duplicates\")
                # Keep first file, remove others
                for file in files[1:]:
                    print(f\"    Removing: {file}\")
                    # file.unlink()  # Uncomment to actually remove
        else:
            print(\"  No duplicates found\")
```

Step 2: Verify Image Quality

```
# Run quality check script
python3 scripts/check_quality.py
```

Quality Criteria:

- ✅ Minimum resolution: 640x640
- ✅ Aspect ratio: 0.5 to 2.0 (not too stretched)
- ✅ File integrity: No corrupted images
- ✅ Brightness: Not too dark/overexposed

Script: `scripts/check_quality.py`

```
#!/usr/bin/env python3
import os
from pathlib import Path
from PIL import Image
import numpy as np

def check_image_quality(image_path):
    \"\"\"Check if image meets quality standards\"\"\"
    try:
        img = Image.open(image_path)
        width, height = img.size

        # Check resolution
        min_dimension = min(width, height)
        is_high_res = min_dimension >= 640

        # Check aspect ratio
        aspect_ratio = width / height
        is_good_aspect = 0.5 <= aspect_ratio <= 2.0

        # Check brightness
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img_array = np.array(img)
        avg_brightness = img_array.mean()
        is_good_brightness = 30 <= avg_brightness <= 225

        return {
            \"path\": image_path,
            \"width\": width,
            \"height\": height,
            \"aspect_ratio\": aspect_ratio,
            \"brightness\": avg_brightness,
            \"passes\": is_high_res and is_good_aspect and is_good_brightness,
            \"issues\": []
        }
    except Exception as e:
        return {\"path\": image_path, \"passes\": False, \"issues\": [str(e)]}

if __name__ == \"__main__\":
    base_dir = Path(\"/home/ubuntu/basketball_app/training_data\")

    low_quality = []
    total_checked = 0

    for root, dirs, files in os.walk(base_dir):
        if \"raw_downloads\" in root or \"scripts\" in root:
            continue

        for file in files:
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                filepath = Path(root) / file
                result = check_image_quality(filepath)
                total_checked += 1

                if not result[\"passes\"]:
                    low_quality.append(result)

    print(f\"\\nQuality Check Results:\")
    print(f\"  Total images checked: {total_checked}\")
    print(f\"  Low quality images: {len(low_quality)}\")
```

```
    print(f\"  Pass rate: {((total_checked - len(low_quality)) / total_checked *
100):.1f}%\")
```

Step 3: Create Clean Dataset

```
# Create clean dataset directory
mkdir -p /home/ubuntu/basketball_app/training_data/clean_dataset

# Copy only high-quality, unique images
python3 scripts/create_clean_dataset.py
```

Annotation Process

Option 1: RoboFlow (Recommended)

Pros:
- Web-based, no installation
- Built-in augmentation
- Direct model training
- Collaboration features

Steps:

1. **Create RoboFlow Account**
   `Visit: https://roboflow.com`
   `Sign up for free account`

2. **Create New Project**
   `Project Name: Basketball Shot Analysis`
   `Project Type: Object Detection (for keypoints)`
   `License: Private`

3. **Upload Images**
   `bash`
   `# Use RoboFlow API`
   `python3 scripts/upload_to_roboflow.py`

4. **Annotate Images**
   - Use RoboFlow annotation interface
   - Add keypoint labels for body parts
   - Add bounding boxes for ball detection

Option 2: CVAT (Open Source)

Pros:
- Free and open source
- Powerful annotation tools
- Local data control

Steps:

1. **Start CVAT Server**
   `bash`
   `docker-compose up -d`

2. **Access Web Interface**
   `Open: http://localhost:8080`

3. **Create Annotation Task**
   - Upload images
   - Define labels (keypoints, bounding boxes)
   - Assign annotators

Option 3: Label Studio

Pros:
- Flexible labeling interface
- ML-assisted labeling
- Python SDK

Steps:

1. **Start Label Studio**
   `bash`
   `label-studio start`

2. **Create Project**
   `Open: http://localhost:8080`
   `Import images`
   `Configure labeling interface`

Annotation Schema

For Shooting Form Keypoints

Keypoint Labels (17 points):

```
{
  \"keypoints\": [
    \"nose\",
    \"left_eye\", \"right_eye\",
    \"left_ear\", \"right_ear\",
    \"left_shoulder\", \"right_shoulder\",
    \"left_elbow\", \"right_elbow\",
    \"left_wrist\", \"right_wrist\",
    \"left_hip\", \"right_hip\",
    \"left_knee\", \"right_knee\",
    \"left_ankle\", \"right_ankle\"
  ]
}
```

For Ball Detection

Bounding Box Labels:

```
{
  \"classes\": [\"basketball\"],
  \"format\": \"COCO\",
  \"coordinates\": \"[x_min, y_min, width, height]\"
}
```

For Form Quality

Classification Labels:

```
{
  \"classes\": [
    \"excellent_form\",
    \"good_form\",
    \"needs_work\",
    \"poor_form\"
  ]
}
```

Data Augmentation

Using Albumentations

Install:

```
pip install albumentations
```

Augmentation Script:

```
#!/usr/bin/env python3
import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import os
from pathlib import Path

# Define augmentation pipeline
transform = A.Compose([
    A.RandomRotate90(p=0.5),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.3),
    A.GaussianBlur(blur_limit=(3, 7), p=0.2),
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1, p=0.3),
    A.RandomResizedCrop(height=640, width=640, scale=(0.8, 1.0), p=0.5),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
], keypoint_params=A.KeypointParams(format='xy', remove_invisible=False))

def augment_dataset(input_dir, output_dir, num_augmentations=5):
    \"\"\"Apply augmentations to dataset\"\"\"
    input_path = Path(input_dir)
    output_path = Path(output_dir)
    output_path.mkdir(parents=True, exist_ok=True)

    for img_file in input_path.glob('*.jpg'):
        image = cv2.imread(str(img_file))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Save original
        cv2.imwrite(str(output_path / img_file.name), image)

        # Generate augmentations
        for i in range(num_augmentations):
            augmented = transform(image=image)
            aug_image = augmented['image']

            output_file = output_path / f\"{img_file.stem}_aug{i}{img_file.suffix}\"
            cv2.imwrite(str(output_file), aug_image)

        print(f\"Processed {img_file.name}: {num_augmentations} augmentations created\
")

if __name__ == \"__main__\":
    base_dir = Path(\"/home/ubuntu/basketball_app/training_data\")

    # Augment each category
    categories = [
        \"shooting_form_keypoints/professional\",
        \"form_quality_classifier/excellent_form\",
        \"ball_trajectory/jump_shots\"
    ]

    for category in categories:
        input_dir = base_dir / category
        output_dir = base_dir / \"augmented\" / category
        augment_dataset(input_dir, output_dir, num_augmentations=3)
```

Run Augmentation:

```
python3 scripts/augment_dataset.py
```

Expected Output:
- Original: 7,280 images
- After 3x augmentation: ~29,000 images

Upload to RoboFlow

Step 1: Get API Keys

```
1. Go to: https://roboflow.com
2. Navigate to: Settings → Roboflow API
3. Copy your API key
```

Available Keys:
- Private API Key: `rDWynPrytSysASUlyGvK`
- Publishable Key: `rf_qisv7ZQd27SzKITWRc2blZZo5F83`

Step 2: Upload Script

Script: `scripts/upload_to_roboflow.py`

```
#!/usr/bin/env python3
from roboflow import Roboflow
from pathlib import Path
import os

# Initialize RoboFlow
ROBOFLOW_API_KEY = \"rDWynPrytSysASUlyGvK\"  # Private key for uploads
rf = Roboflow(api_key=ROBOFLOW_API_KEY)

# Create workspace
workspace = rf.workspace()

# Create projects
projects = {
    \"basketball-shooting-form\": \"shooting_form_keypoints\",
    \"basketball-form-quality\": \"form_quality_classifier\",
    \"basketball-ball-tracking\": \"ball_trajectory\"
}

def upload_dataset(project_name, dataset_path):
    \"\"\"Upload dataset to RoboFlow project\"\"\"
    try:
        # Create or get project
        project = workspace.project(project_name)

        # Upload images
        dataset_dir = Path(dataset_path)
        image_count = 0

        for img_file in dataset_dir.rglob('*.jpg'):
            project.upload(image_path=str(img_file))
            image_count += 1
            if image_count % 100 == 0:
                print(f\"  Uploaded {image_count} images...\")

        print(f\"✓ Uploaded {image_count} images to {project_name}\")
        return True
    except Exception as e:
        print(f\"✗ Error uploading to {project_name}: {str(e)}\")
        return False

if __name__ == \"__main__\":
    base_dir = Path(\"/home/ubuntu/basketball_app/training_data\")

    print(\"RoboFlow Upload\")
    print(\"=\"*60)

    for project_name, dataset_path in projects.items():
        print(f\"\\nUploading {dataset_path}...\")
        full_path = base_dir / dataset_path
        upload_dataset(project_name, full_path)

    print(\"\\n\" + \"=\"*60)
    print(\"Upload complete! Visit RoboFlow to annotate and train.\")
```

Run Upload:

```
python3 scripts/upload_to_roboflow.py
```

Step 3: Generate Dataset Version

In RoboFlow:
1. Complete annotations
2. Go to: Generate → New Version
3. Configure:
- Preprocessing: Auto-Orient, Resize (640x640)
- Augmentation: Flip, Rotate, Brightness
- Split: 70% train, 20% val, 10% test
4. Click: Generate

Model Training Setup

YOLOv8 Pose Estimation

Install Ultralytics:

```
pip install ultralytics
```

Training Script:

```
from ultralytics import YOLO

# Load pretrained model
model = YOLO('yolov8n-pose.pt')

# Train on custom dataset
results = model.train(
    data='/path/to/roboflow/dataset.yaml',
    epochs=100,
    imgsz=640,
    batch=16,
    device=0  # GPU
)

# Validate
metrics = model.val()

# Export
model.export(format='onnx')
```

Custom Keypoint Detection

Using PyTorch:

```
import torch
import torchvision
from torchvision.models.detection import keypointrcnn_resnet50_fpn

# Load pretrained model
model = keypointrcnn_resnet50_fpn(pretrained=True)

# Fine-tune on basketball dataset
# (Add custom training loop here)
```

Troubleshooting

Issue: Duplicate Images

Solution:

```
python3 scripts/remove_duplicates.py
```

Issue: Low Quality Images

Solution:

```
# Filter by quality score
python3 scripts/check_quality.py
# Move low-quality images to separate folder
```

Issue: RoboFlow Upload Fails

Solution:

```
# Check API key
echo $ROBOFLOW_API_KEY

# Verify internet connection
ping -c 3 roboflow.com

# Check file formats
find . -type f ! -name \"*.jpg\" ! -name \"*.png\"
```

Issue: Insufficient Data for Category

Solution:

1. Download additional images from Kaggle
2. Use data augmentation (3-5x multiplier)
3. Generate synthetic data with Stable Diffusion

Next Steps

Immediate (Week 1)

- [ ] Run quality control scripts
- [ ] Upload dataset to RoboFlow
- [ ] Begin manual annotation

Short-term (Week 2-4)

- [ ] Complete keypoint annotations
- [ ] Generate augmented dataset
- [ ] Train baseline YOLOv8 model
- [ ] Evaluate on test set

Long-term (Month 2+)

- [ ] Collect additional WNBA/youth basketball images
- [ ] Train custom models (ResNet, EfficientNet)
- [ ] Deploy to production API
- [ ] Set up continuous learning pipeline

Resources

Documentation

- **RoboFlow Docs:** https://docs.roboflow.com
- **Ultralytics YOLOv8:** https://docs.ultralytics.com
- **CVAT Guide:** https://opencv.github.io/cvat/docs/
- **Albumentations:** https://albumentations.ai/docs/

Tutorials

- **Basketball Pose Estimation:** [YouTube Tutorial]
- **YOLOv8 Custom Training:** [Ultralytics Blog]
- **Data Augmentation Best Practices:** [Papers with Code]

Community

- **RoboFlow Community:** https://community.roboflow.com
- **r/computervision:** Reddit community
- **Computer Vision Discord:** Active community for CV

Last Updated: December 13, 2025
Version: 1.0
Maintainer: Basketball App Development Team
Location: `/home/ubuntu/basketball_app/training_data/`