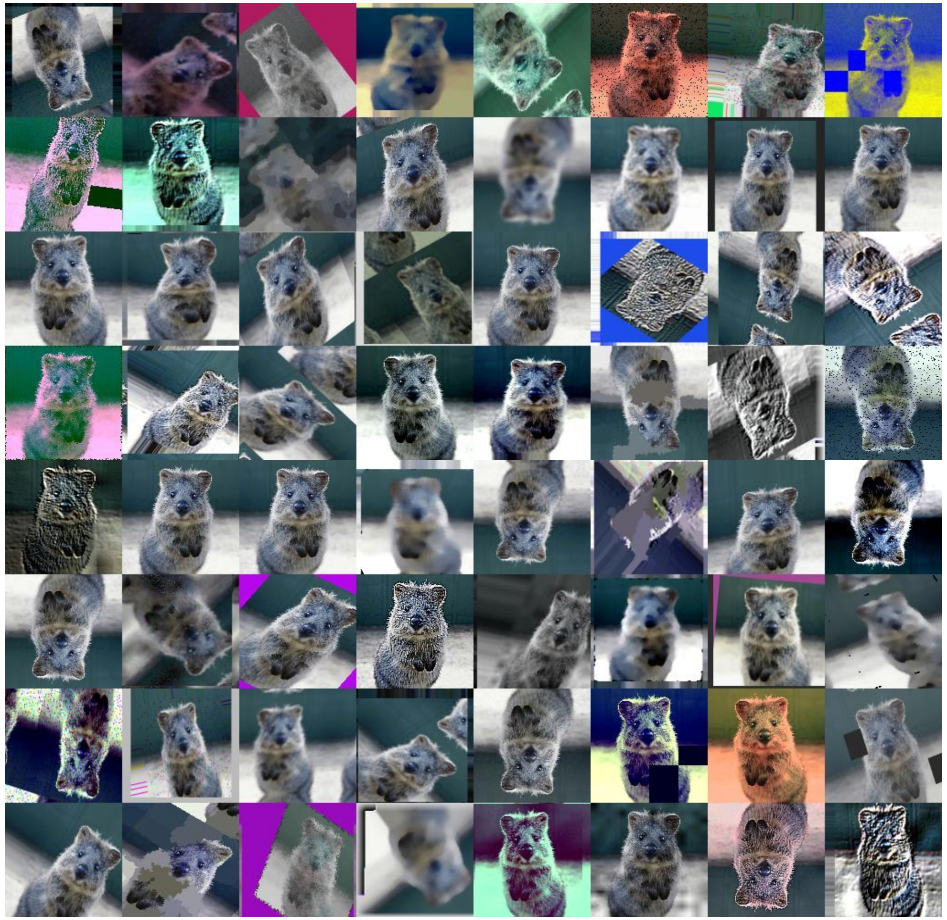
Image Pre-Processing & Augmentation

What is Image Augmentation?



What do we need Image Processing?

- Need for large datasets
- Prevent Overfitting
- Feature Extraction
- Class Imbalances
- Stability of Network

Types of Augmentation

Offline Augmentation

- Used on smaller datasets
- Used to create more data from original data
- Done prior to training
- Transformations done in the beginning

Online Augmentation

- Used when dataset is large
- Used to randomly modify existing data
- Done during training
- Transformations done on minibatches in dataloader

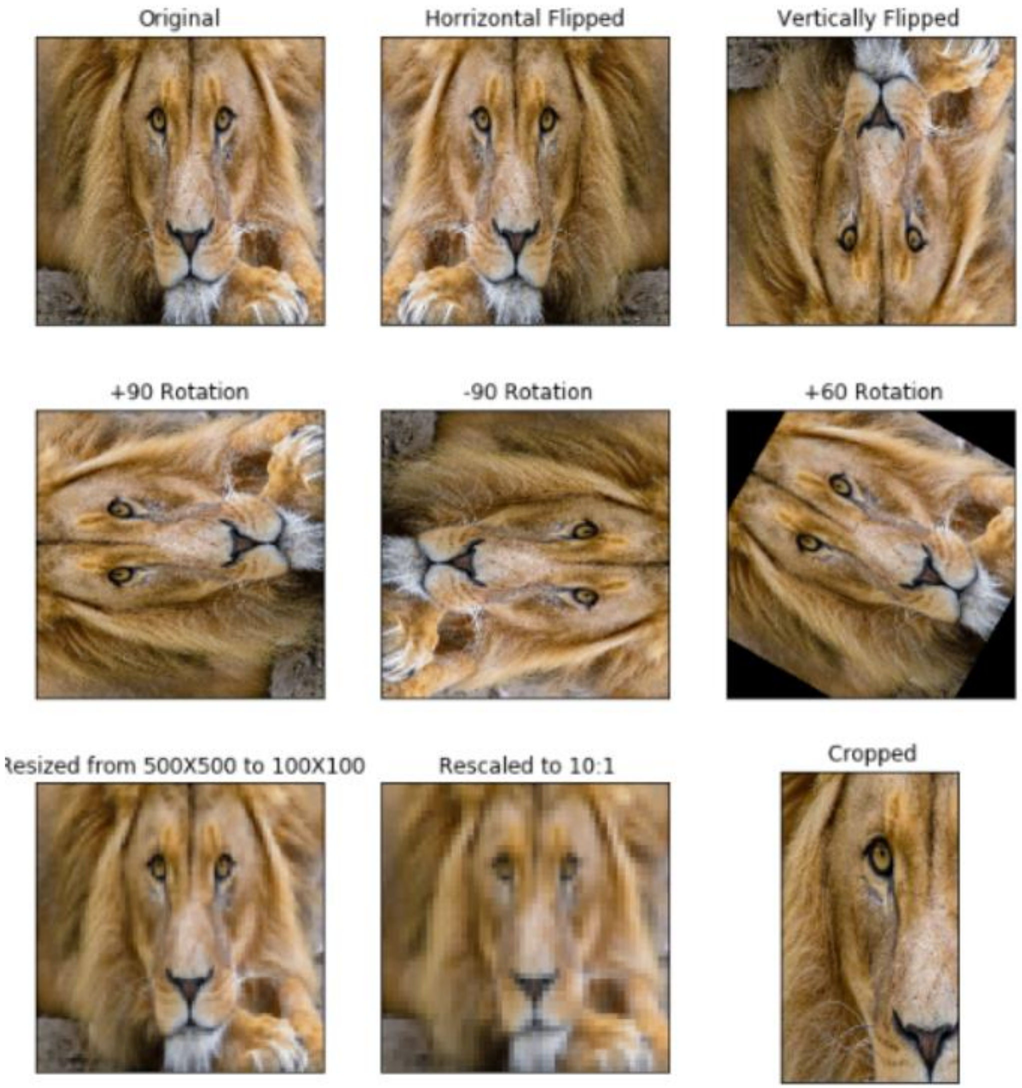
```python
class ImageData(Dataset):
    def __init__(self,df,data_dir,transform=None):
        self.df = df
        self.data_dir = data_dir
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self,index):
        img_path = os.path.join(self.data_dir,self.df.iloc[index,0])
        labels = torch.tensor(self.df.iloc[index,1:],dtype=torch.long)
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            image = self.transform(img)
        return (image,labels)
```

Online Augmentation code example

Types of Transformation

Geometric transformations

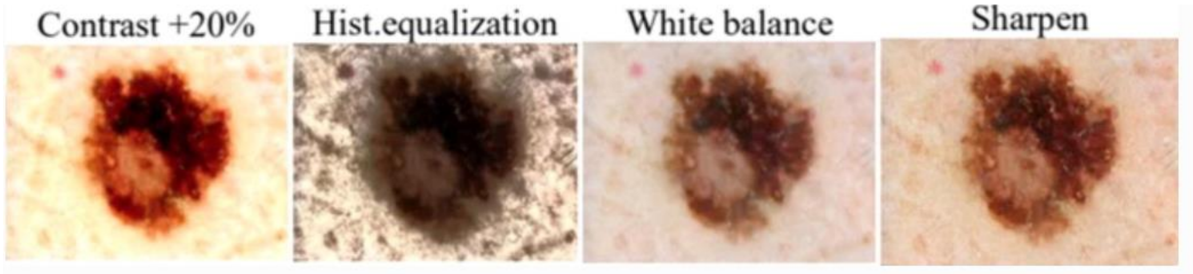
1. Flipping
2. Cropping
3. Scaling
4. Translating
5. Rotating
6. Adding Noise
7. Warping/Distorting



Types of Transformations

Colour Space Transformations

1. Colour shifting
2. Contrast
3. Sharpness
4. Blurring
5. Brightness

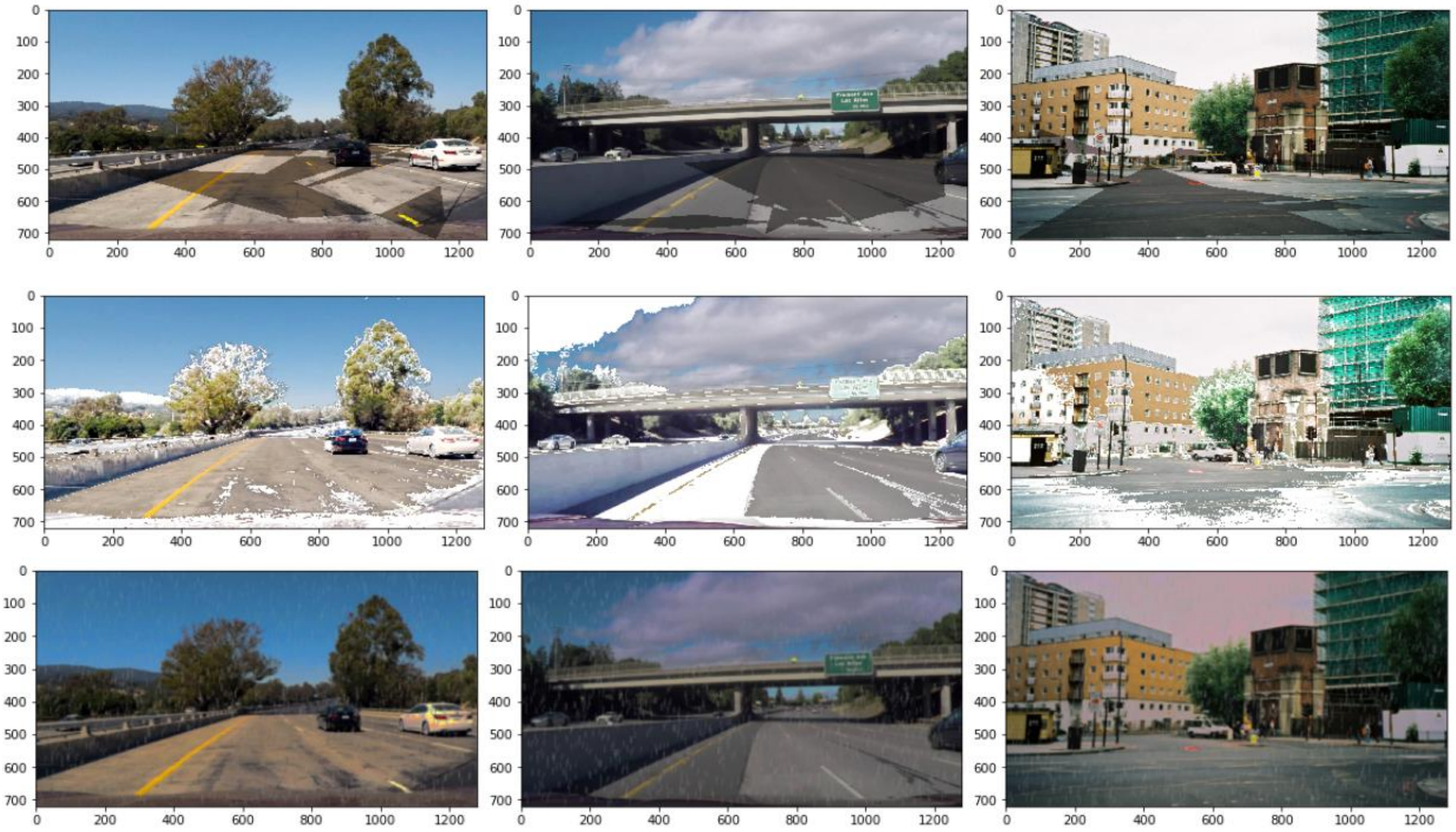


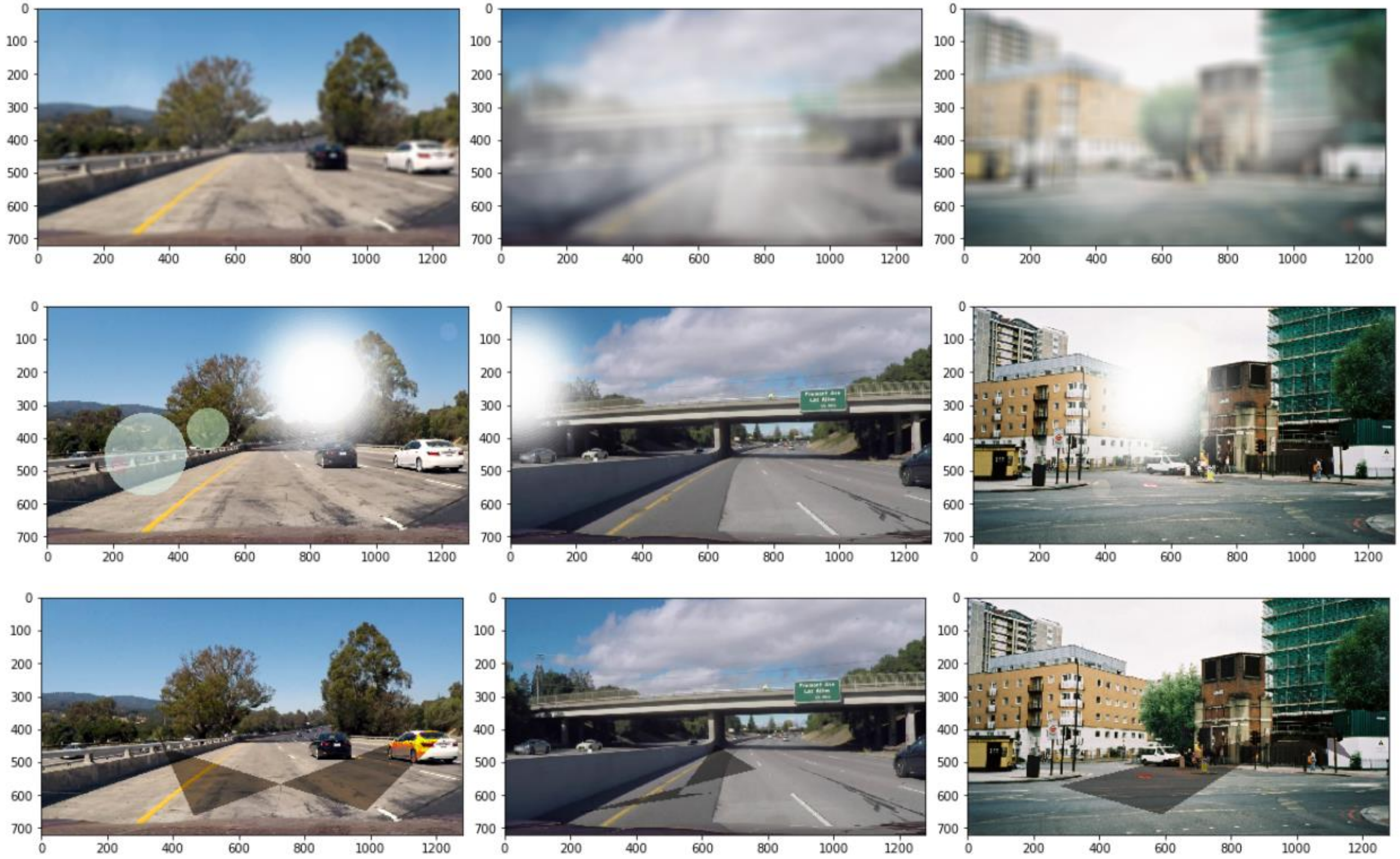
Types of Transformations

Other Techniques

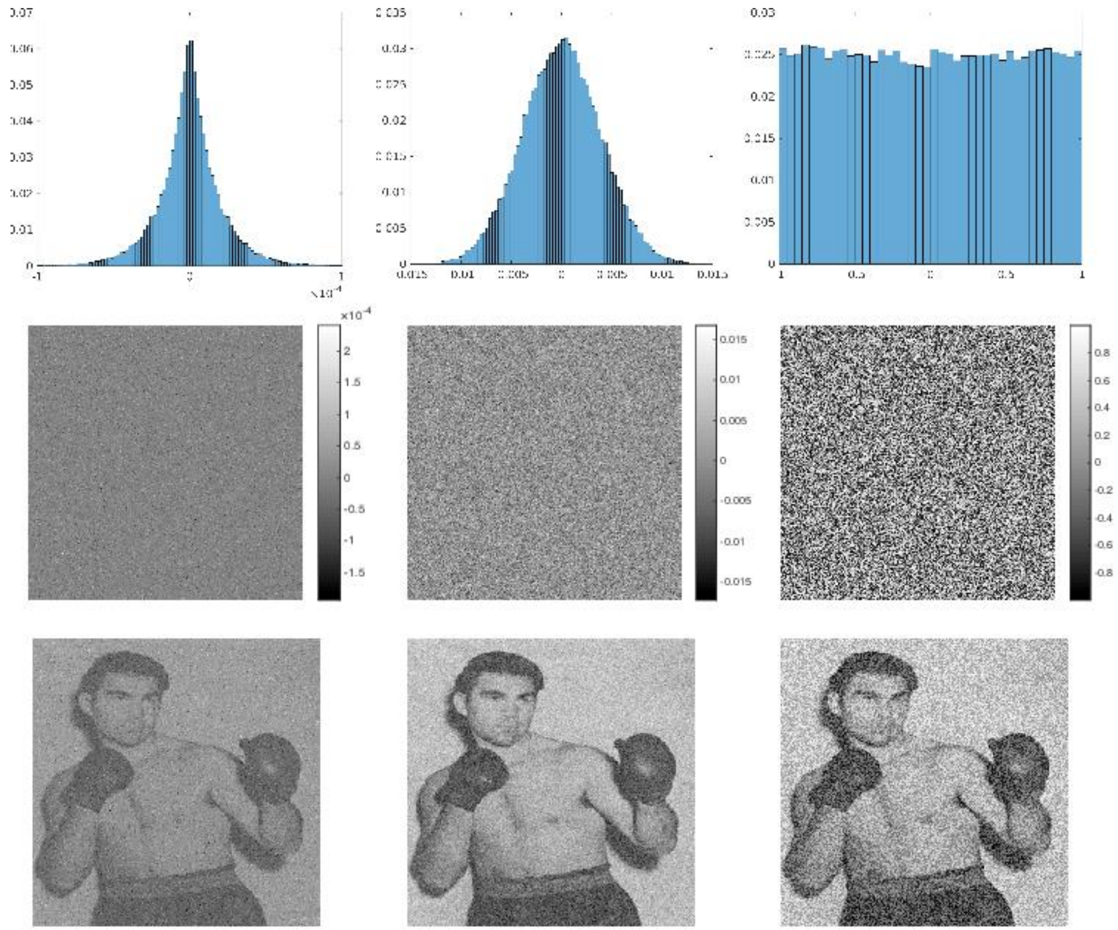
1. Pixel Dropout
2. Jitter
3. Adding
    1. Rain
    2. Snow
    3. Sunflare
    4. Shadow

Albumentation (Automold)





Gaussian Noise



$p = 1$ $\qquad$ $p = 2$ $\qquad$ $p = \infty$

Feature Invariances

- Translation
- Scaling
- Rotation

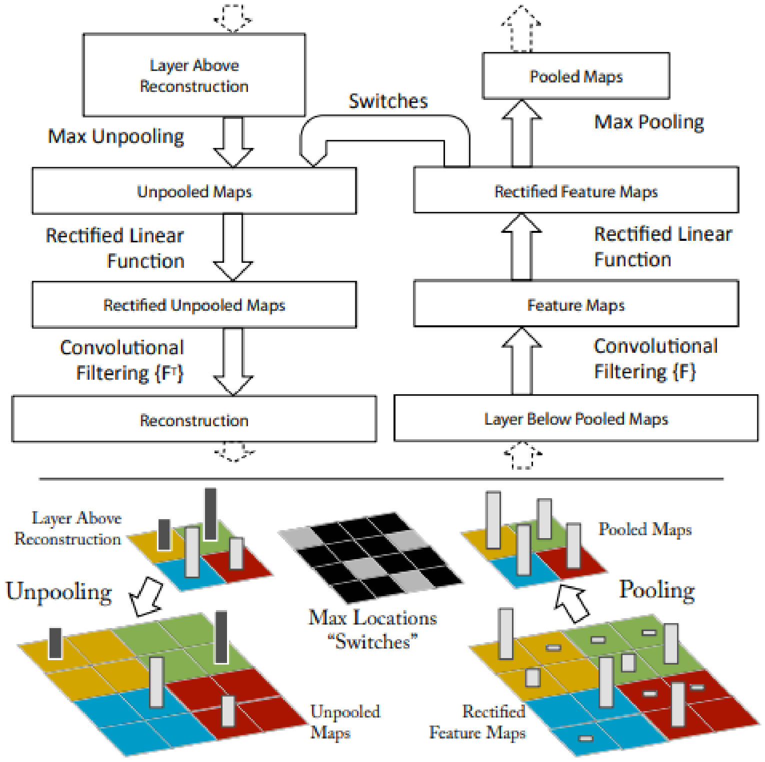
Understanding effect of some fundamental transformations

Visualizing and Understanding Convolutional Networks

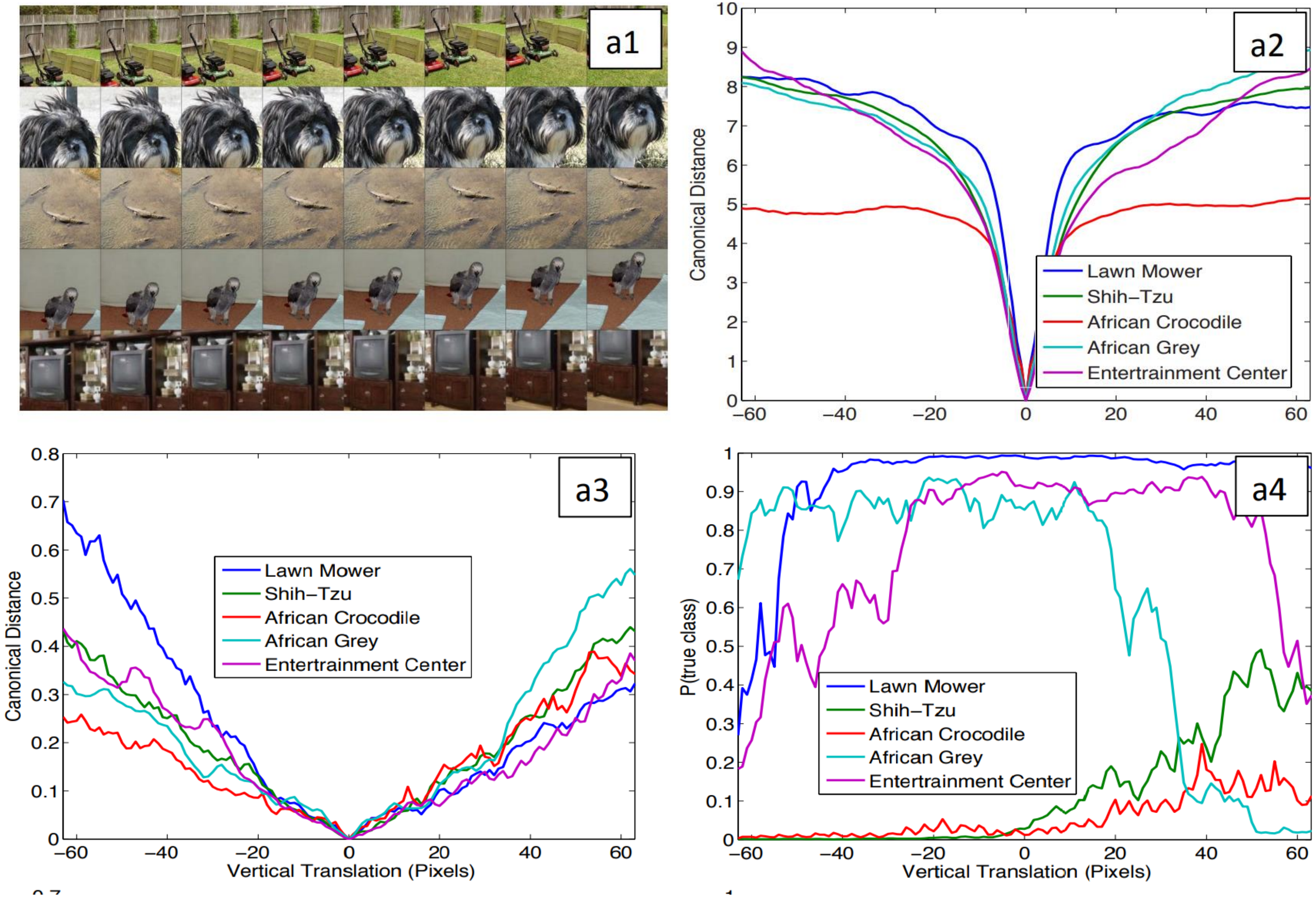
Matthew D. Zeiler ZEILER@CS.NYU.EDU
Dept. of Computer Science, Courant Institute, New York University

Rob Fergus FERGUS@CS.NYU.EDU
Dept. of Computer Science, Courant Institute, New York University

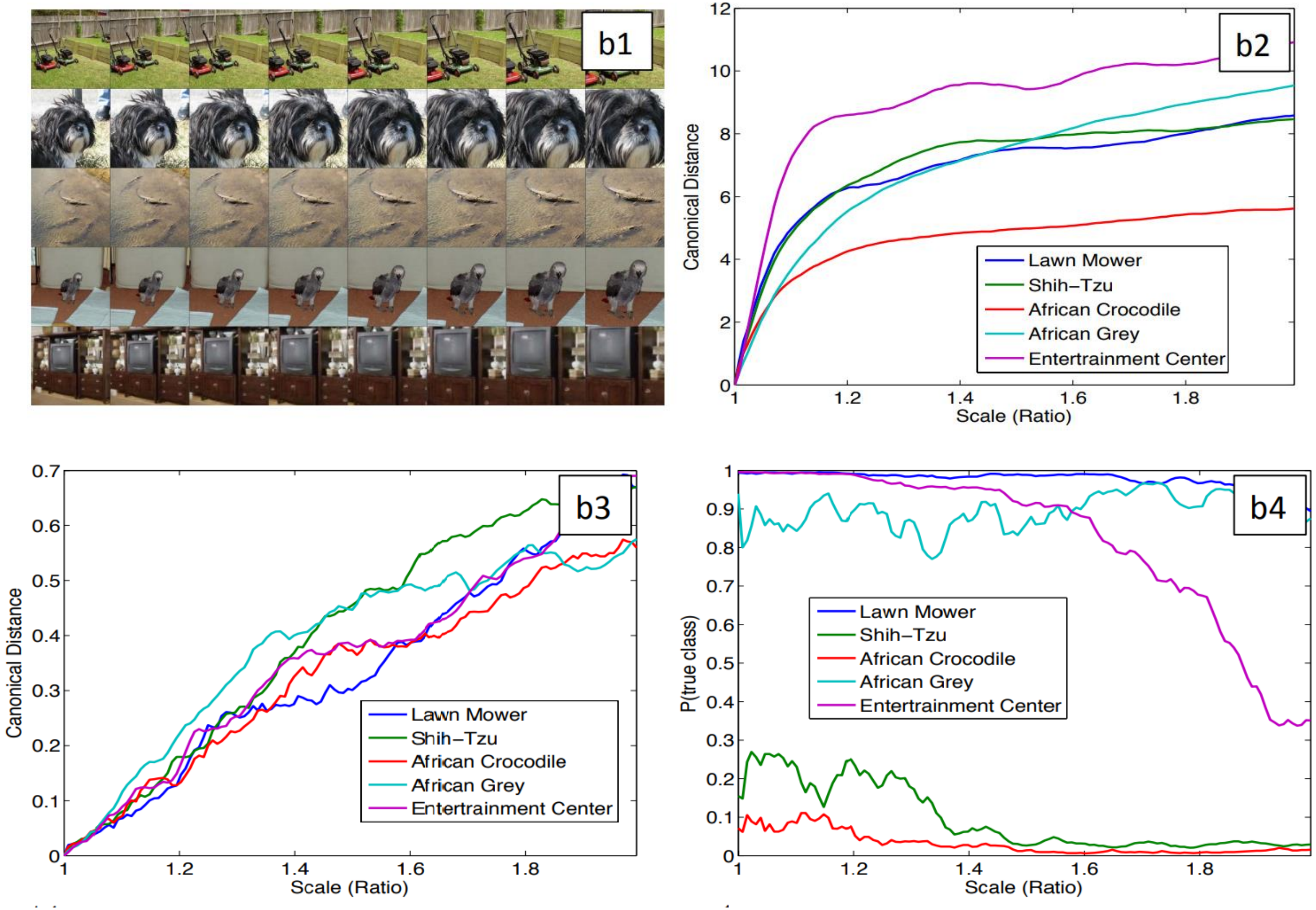
Link: https://arxiv.org/abs/1311.2901



TRANSLATION

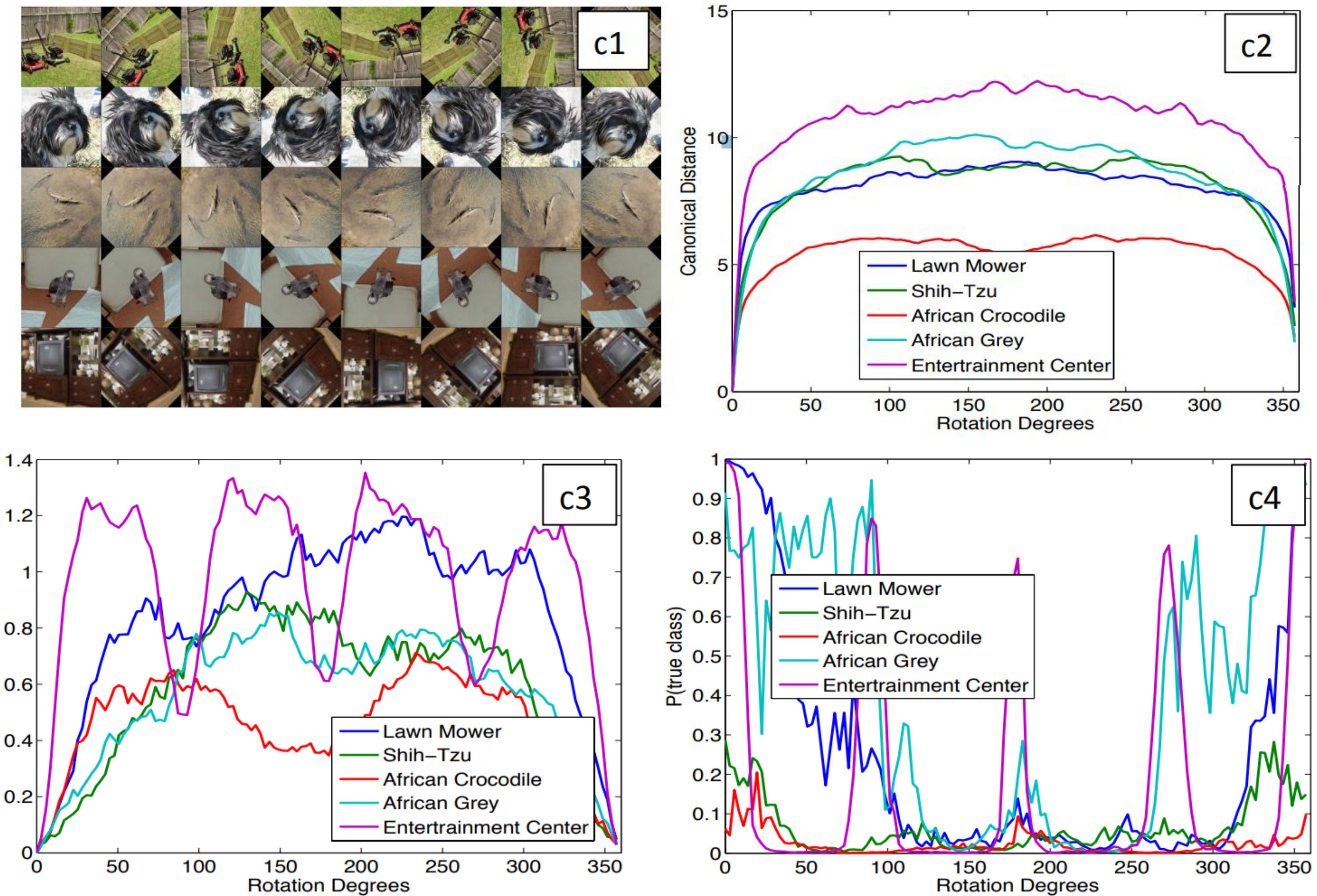


SCALING



b1
b2
b3
b4
Canonical Distance
Scale (Ratio)
P(true class)
Lawn Mower
Shih–Tzu
African Crocodile
African Grey
Entertrainment Center

ROTATION



c1
c2
Canonical Distance
Rotation Degrees
Lawn Mower
Shih–Tzu
African Crocodile
African Grey
Entertrainment Center
c3
Rotation Degrees
Lawn Mower
Shih–Tzu
African Crocodile
African Grey
Entertrainment Center
c4
P(true class)
Rotation Degrees
Lawn Mower
Shih–Tzu
African Crocodile
African Grey
Entertrainment Center

OBSERVATIONS

- 1st layer shows dramatic difference in output for any transformation
- 7th layer has lesser impact and is quasi-linear for translation and scaling
- Rotations are not corrected in the last layer unless the object is rotationally symmetric

CONCLUSION

Network output is stable to translation and scaling but is not invariant to rotation

Normalizing Image Data

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

Albumentation

Albumentations is a fast image augmentation library and easy to use wrapper around other libraries.

Albumentations package is written based on numpy, OpenCV, and imgaug. It is a very popular package written by Kaggle masters and used widely in Kaggle competitions.

Features

Albumentations package is capable of:

- Over 60 pixel-level and spatial-level transformations;
- Transforming images with masks, bounding boxes, and keypoints;
- The library is faster than other libraries on most of the transformations;
- Organizing augmentations into pipelines;
- PyTorch integration;
- Was used to get top results in many DL competitions at Kaggle, topcoder, CVPR, MICCAI;
- Written by Kaggle Masters.

Pixel-Level Transforms

Pixel-level transforms will change just an input image and will leave any additional targets such as masks, bounding boxes, and keypoints unchanged. The list of pixel-level transforms:

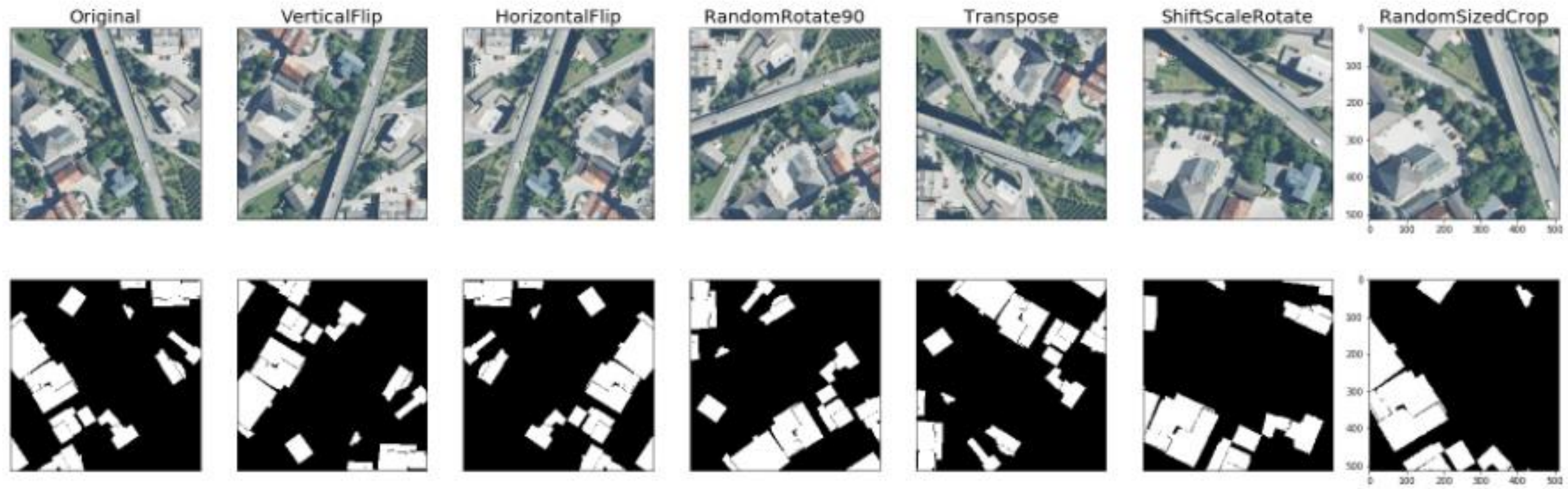
- Blur
- CLAHE
- ChannelShuffle
- HueSaturationValue
- RGBShift
- RandomBrightnessContrast
- ToGray
- GaussNoise

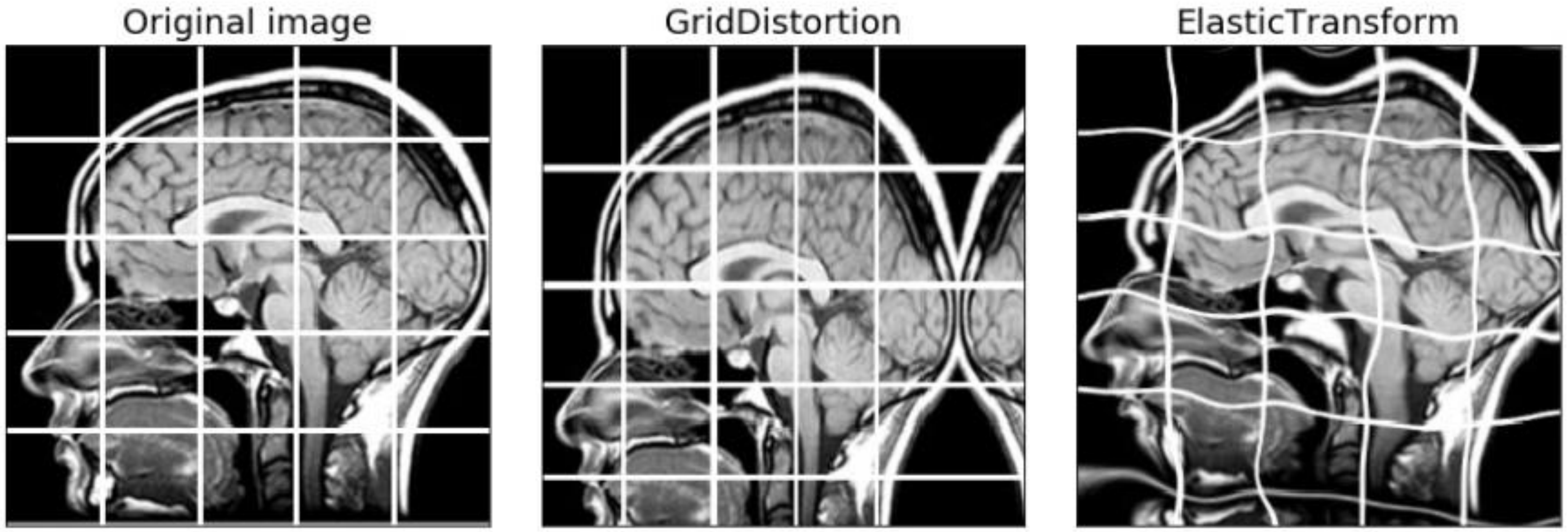

Original image
RGBShift
HueSaturationValue
ChannelShuffle
CLAHE
RandomContrast
RandomGamma
RandomBrightness
Blur
MedianBlur
ToGray
JpegCompression

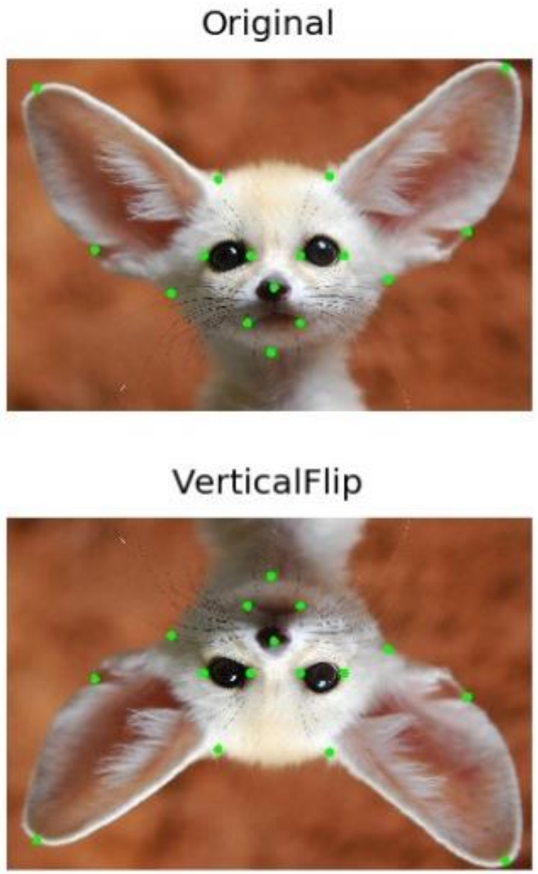
Spatial-level Transforms

Spatial-level transforms will simultaneously change both an input image as well as additional targets such as masks, bounding boxes, and keypoints.

- VerticalFlip
- HorizontalFlip
- RandomRotate90
- Transpose
- RandomResizedCrop
- GridDistortion
- ElasticTransform


Original
VerticalFlip
HorizontalFlip
RandomRotate90
Transpose
ShiftScaleRotate
RandomSizedCrop


Original image
GridDistortion
ElasticTransform


Original
HorizontalFlip
VerticalFlip
ShiftScaleRotate


Original image
Augmented image
Street Light
Street Light
Traffic Sign (Front)
Car
Car
Street Light
Traffic Sign
Car
Car
Original mask
Augmented mask

Pipelines

We want to stack many transformations together into a single pipeline.
Depending on the framework we are using there are different methods.

```
>>> transforms.Compose([
>>>     transforms.CenterCrop(10),
>>>     transforms.ToTensor(),
>>> ])
```

```
# Compose a complex augmentation pipeline
augmentation_pipeline = A.Compose(
    [
        A.HorizontalFlip(p = 0.5), # apply horizontal flip to 50% of images
        A.OneOf(
            [
                # apply one of transforms to 50% of images
                A.RandomContrast(), # apply random contrast
                A.RandomGamma(), # apply random gamma
                A.RandomBrightness(), # apply random brightness
            ],
            p = 0.5
        ),
        A.OneOf(
            [
                # apply one of transforms to 50% images
                A.ElasticTransform(
                    alpha = 120,
                    sigma = 120 * 0.05,
                    alpha_affine = 120 * 0.03
                ),
                A.GridDistortion(),
                A.OpticalDistortion(
                    distort_limit = 2,
                    shift_limit = 0.5
                ),
            ],
            p = 0.5
        )
    ],
    p = 1
)
```

```python
# Import pytorch utilities from albumentations
from albumentations.pytorch import ToTensor

# Define the augmentation pipeline
augmentation_pipeline = A.Compose(
    [
        A.HorizontalFlip(p = 0.5), # apply horizontal flip to 50% of images
        A.OneOf(
            [
                # apply one of transforms to 50% of images
                A.RandomContrast(), # apply random contrast
                A.RandomGamma(), # apply random gamma
                A.RandomBrightness(), # apply random brightness
            ],
            p = 0.5
        ),

        A.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]),

        ToTensor() # convert the image to PyTorch tensor
    ],
    p = 1
)

# Load the augmented data

# Define the demo dataset
class DogDataset2(Dataset):
    '''
    Sample dataset for Albumentations demonstration.
    The dataset will consist of just one sample image.
    '''

    def __init__(self, image, augmentations = None):
        self.image = image
        self.augmentations = augmentations # save the augmentations

    def __len__(self):
        return 1 # return 1 as we have only one image

    def __getitem__(self, idx):
        # return the augmented image
        # no need to convert to tensor, because image is converted to tensor already by the p
        augmented = self.augmentations(image = self.image)
        return augmented['image']

# Initialize the dataset, pass the augmentation pipeline as an argument to init function
train_ds = DogDataset2(image, augmentations = augmentation_pipeline)

# Initilize the dataloader
trainloader = DataLoader(train_ds, batch_size=1, shuffle=True, num_workers=0)
```

OPEN CV & PIL

```python
class TorchvisionDataset(Dataset):
    def __init__(self, file_paths, labels, transform=None):
        self.file_paths = file_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        label = self.labels[idx]
        file_path = self.file_paths[idx]

        # Read an image with PIL
        image = Image.open(file_path)
        if self.transform:
            image = self.transform(image)
        return image, label


torchvision_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225],
    )
])
```

```python
class AlbumentationsDataset(Dataset):
    """__init__ and __len__ functions are the same as in TorchvisionDataset"""
    def __init__(self, file_paths, labels, transform=None):
        self.file_paths = file_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        label = self.labels[idx]
        file_path = self.file_paths[idx]

        # Read an image with OpenCV
        image = cv2.imread(file_path)

        # By default OpenCV uses BGR color space for color images,
        # so we need to convert the image to RGB color space.
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        if self.transform:
            augmented = self.transform(image=image)
            image = augmented['image']
        return image, label


albumentations_transform = A.Compose([
    A.Resize(256, 256),
    A.RandomCrop(224, 224),
    A.HorizontalFlip(),
    A.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225],
    ),
    ToTensorV2()
])
```

Data Augmentation as a subfield

Fig. 2