Preprocessing and Data Augmentation

Ramin Zarebidoky (LiterallyTheOne)

02 Dec 2025

Preprocessing and Augmentation

Introduction

In the previous tutorial, we have learned about the basic layers used in CNNs. In this tutorial, we are going to learn about **preprocessing** and **augmentation** layers in **Keras**.

Preprocessing layers in Keras

Preprocessing

Data preprocessing is a set of steps that we take before feeding the data to our model. These steps help us to have clean, consistent, and meaningful inputs. Also, they help the model to have a better accuracy, convergence speed, and generalization. When we were loading our dataset, we used two transformations: `Resize` and `ToTensor`. These two functions were related to the **PyTorch** and we were using them on the `ImageFolder`. Now, we are going to learn about some preprocessing layers in **Keras**.

Resizing

`Resizing` layer is a layer that resizes its input to match the given size. Here is an example that we resized an image with the size of 1920×1080 to 224×224.

```python
from keras.layers import Resizing

resizing_layer = Resizing(224, 224)

input_image = np.random.randint(0, 256, (1, 1920, 1080, 3))

result_image = resizing_layer(input_image)

print(f"Input's shape: {input_image.shape}")
print(f"Result's shape: {result_image.shape}")
```

```
"""
--------
output:

Input's shape: (1, 1920, 1080, 3)
Result's shape: torch.Size([1, 224, 224, 3])
"""
```

Rescaling

`Rescaling` layer is a layer that rescales its input to the given scale. In the example below, we have made a `Rescaling` layer with the scale of $\frac{1}{255}$.

```
from keras.layers import Rescaling

rescaling_layer = Rescaling(1 / 255)

input_image = np.random.randint(0, 256, (1, 224, 224, 3))

result_image = rescaling_layer(input_image)

print(f"Input's max: {input_image.max()}")
print(f"Input's min: {input_image.min()}")
print(f"Result's max: {result_image.max()}")
print(f"Result's min: {result_image.min()}")

"""
--------
output:

Input's max: 255
Input's min: 0
Result's max: 1.0
Result's min: 0.0
"""
```

Specific model preprocessing

Each model has its own preprocessing procedure. In **Keras** we can load and use them. Here is an example of the preprocessing for `MobileNetV2`.

```
from keras.applications.mobilenet_v2 import preprocess_input

input_image = np.random.randint(0, 256, (1, 224, 224, 3))
```

```
result_image = preprocess_input(input_image)

print(f"Input's max: {input_image.max()}")
print(f"Input's min: {input_image.min()}")
print(f"Result's max: {result_image.max()}")
print(f"Result's min: {result_image.min()}")


"""
--------
output:

Input's max: 255
Input's min: 0
Result's max: 1.0
Result's min: -1.0
"""
```

As you can see, in the example above, the input is mapped to the range of $[-1.0, 1.0]$. This is the way **MobileNetV2** expects its input to be. If we want to use this preprocessing procedure in our model layers, we can use a layer called `Lambda`. `Lambda` takes a function, like `preprocess_input`, and turns it to a layer. Here is an example of how we can achieve that.

```
from keras.layers import Lambda

input_image = np.random.randint(0, 256, (1, 224, 224, 3))
input_image = np.array(input_image, dtype=float)

preprocessing_layer = Lambda(preprocess_input)

result_image = preprocessing_layer(input_image)

print(f"Input's max: {input_image.max()}")
print(f"Input's min: {input_image.min()}")
print(f"Result's max: {result_image.max()}")
print(f"Result's min: {result_image.min()}")


"""
--------
output:

Input's max: 255.0
Input's min: 0.0
Result's max: 1.0
```

```
Result's min: -1.0
"""
```

Augmentation

Data Augmentation is a technique in machine learning that artificially expands our training dataset by applying different transformations. **Data Augmentation** is extremely useful when we don't have enough data or our data is not balanced. It helps us with the generalization and prevents the model from over-fitting. We have so many different **augmentation** techniques for different use-cases. Let's get to know how to use some of them in **Keras**. You can see the output of all the examples in this notebook

RandomFlip

`RandomFlip`, technically, has a 50 chance to flip its input in the given mode. Modes can be:

- `horizontal`
- `vertical`
- `horizontal_and_vertical`

Here is an example that only flips horizontally:

```
from keras.layers import RandomFlip

random_flip_layer = RandomFlip("horizontal")
```

- The most common rotation is horizontal
- Use it when left and right rotation doesn't matter

RandomRotation

`RandomRotation` rotates its input with the given factor. The range of the rotation would be: $[-factor * \pi, +factor * \pi]$
For example, if we put the factor to 0.2, it would rotate the input in the range of

$$[-0.2 * \pi, +0.2 * \pi] = [-0.2 * 180^\circ, 0.2 * 180^\circ] = \boxed{[-36^\circ, 36^\circ]}$$

Here is an example of this layer:

```
from keras.layers import RandomRotation

random_rotation_layer = RandomRotation(0.2)
```

- Make your model robust to the rotation

RandomZoom

RandomZoom zooms in or out respect to the height_factor and width_factor. Here is an example of this layer:

```python
from keras.layers import RandomZoom

random_zoom_layer = RandomZoom(0.4, 0.2)
```

- Helps the model to handle scale changes
- Super effective in classification problems

RandomTranslation

RandomZoom moves the image respect to the height_factor and width_factor. Here is an example of this layer:

```python
from keras.layers import RandomTranslation

random_translation_layer = RandomTranslation(0.2, 0.2)
```

- Simulates small camera movements
- It is super important for the tasks that position of the object doesn't matter

RandomContrast

RandomContrast changes the contrast respect to the given factor. Here is an example of this layer:

```python
from keras.layers import RandomContrast

random_contrast_layer = RandomContrast(0.4)
```

- Helps us with the different lightning setups
- Useful in outdoor scenes and natural environments

RandomBrightness

RandomBrightness changes the brightness respect to the given factor. Here is an example of this layer:

```python
from keras.layers import RandomBrightness

random_brightness_layer = RandomBrightness(0.1)
```

- Helps us with the different lightning environments
- Specially data's taken in the different times of the day in the nature

RandomCrop

`RandomCrop` crops to the given `height` and `width` randomly. Here is an example of this layer:

```
from keras.layers import RandomCrop

random_crop_layer = RandomCrop(224, 224)
```

- Simulates random object placements
- Extremely useful in large-scale training

Add preprocessing and augmentation layers to our model

We should add our preprocessing and augmentation layers before feeding our data to the model. Here is an example:

```
"""
augmentation_layers = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomFlip("vertical"),
        layers.RandomZoom(0.1, 0.1),
        layers.RandomTranslation(0.05, 0.05),
        layers.RandomRotation(0.05),
    ]
)

model = keras.Sequential(
    [
        layers.Input(shape=(3, 224, 224)),
        layers.Permute((2, 3, 1)),
        layers.Rescaling(1.0 / 255),
        augmentation_layers,
        layers.Lambda(preprocess_input),
        base_model,
        layers.Flatten(),
        layers.Dense(4, activation="softmax"),
    ]
)


--------
output:

Model: "sequential_5"

 Layer (type)                      Output Shape
↪  Param #
```

```
 permute_2 (Permute)              (None, 224, 224, 3)
↪  0

 rescaling (Rescaling)            (None, 224, 224, 3)
↪  0

 sequential_4 (Sequential)        (None, 224, 224, 3)
↪  0

 lambda (Lambda)                  (None, 224, 224, 3)
↪  0

 mobilenetv2_1.00_224             (None, 7, 7, 1280)
↪  2,257,984
 (Functional)
↪

 flatten_2 (Flatten)              (None, 62720)
↪  0

 dense_2 (Dense)                  (None, 4)
↪  250,884

Total params: 2,508,868 (9.57 MB)
Trainable params: 250,884 (980.02 KB)
Non-trainable params: 2,257,984 (8.61 MB)

"""
```

In the example above, we have defined a `Sequential` to add our augmentation layers. Our augmentation layers consists of filliping, zooming, translation, and rotation. We also added the preprocess unit and rescaling.

We should always consider not over stack these layers. In this example, we only wanted to show you how we can add multiple augmentation layers. It might be too much for our model, which right now doesn't have so many parameters to learn.

Your turn

Now, choose the correct preprocessing and augmentation for your model and dataset and see the outputs.

Conclusion

In this tutorial, we have learned about preprocessing and augmentation. First, we explained about preprocessing and how to use them in **Keras**. Then, we explored three different preprocessors. After that, we explained about the data augmentations and their use-cases. We introduced some of the most important augmentation layers. Finally, we learned how to add these layers in our model.