BST 261: Data Science II
Lecture 5
Convolutional Neural Networks (CNNs): Data Augmentation and Pretrained Bases

Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2021

Recipe of the Day!

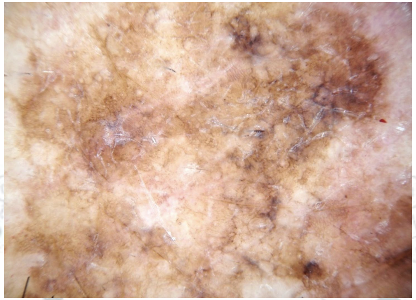
Iced vanilla and caramel profiteroles

CNNs in Python

Classifying Skin Lesions

- We'll be using data from the International Skin Imaging Collaboration: Melanoma Project
- We'll be classifying images as malignant or benign
- The overarching goal of the ISIC Melanoma Project is to support efforts to reduce melanoma-related deaths and unnecessary biopsies by improving the accuracy and efficiency of melanoma early detection

Malignant



Malignant

Benign

Benign

Classifying Skin Lesions

- This archive contains 23k images of classified skin lesions. It contains both malignant and benign examples
  - We'll be using a fraction of this
- Each example contains the image of the lesion, meta data regarding the lesion (including classification and segmentation) and meta data regarding the patient
- The data can be viewed in this link (in the gallery section)
- It can be downloaded through the site or by using this repository

Classification Skin Lesions

- The subsample of the data is available in a Google Drive folder
- You can access it with this code and notebook
- You can also download the images to your machine if you would like
  - There are zip files available on canvas
- We’ll start with creating a simple CNN

```python
# Define model
model = keras.Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Flatten(),

  layers.Dense(512, activation='relu'),

  layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

```python
# Define model
model = keras.Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Flatten(),

  layers.Dense(512, activation='relu'),

  layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Convolution and pooling layers - notice how the output size decreases with each layer. Remember what applying a filter, padding, and/or strides does to an input.

```python
# Define model
model = keras.Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Flatten(),

  layers.Dense(512, activation='relu'),

  layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

We finish the network with 1 hidden dense layer and 1 output layer. Note that most of the parameters in the model come from the hidden dense layer and not the convolution or pooling layers.

```python
# Define model
model = keras.Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),

  layers.Flatten(),

  layers.Dense(512, activation='relu'),

  layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Total # of parameters that need to be learned by the network

```python
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size = (150, 150),
        batch_size = 20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode = 'binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size = (150, 150),
        batch_size = 20,
        class_mode = 'binary')
```

We first scale the data to get values between 0 and 1.

Then we transform the images to be 150x150 pixels in size (this is arbitrary), declare a batch size of 20 (this is also arbitrary), and declare the class mode (i.e. the type of classification we want to do)

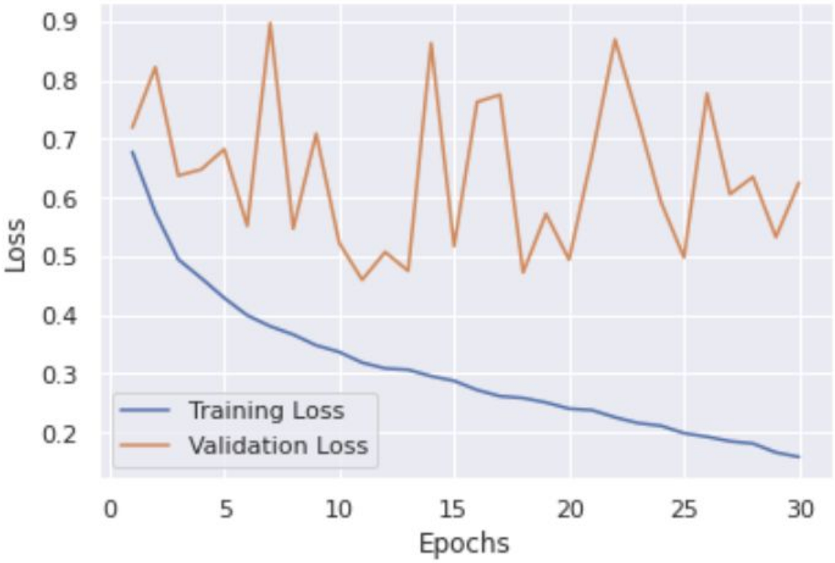
```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

We can see the shape of the images: they are in batches of 20, with each image being represented by 3, 150x150 tensors: one for R, one for G and one for B color "channels".

```
history = model.fit(
        train_generator,
        steps_per_epoch = 81, # ceil(1609/20)
        epochs = 30,
        validation_data = validation_generator,
        validation_steps = 22) # ceil(426/20)
```

The number of training examples divided by the batch size, i.e. how many batches we need to go through until the model sees all training data. For both the training and validation sets.


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Loss
Training Loss
Validation Loss
0
5
10
15
20
25
30
Epochs


0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
Accuracy
Training Accuracy
Validation Accuracy
0
5
10
15
20
25
30
Epochs

Data Augmentation

- As we have seen, overfitting is caused by having too few training examples to learn from

- Data augmentation generates more training data from existing training examples by **augmenting** the samples via a number of random transformations

- These transformations should yield believable images

Data Augmentation

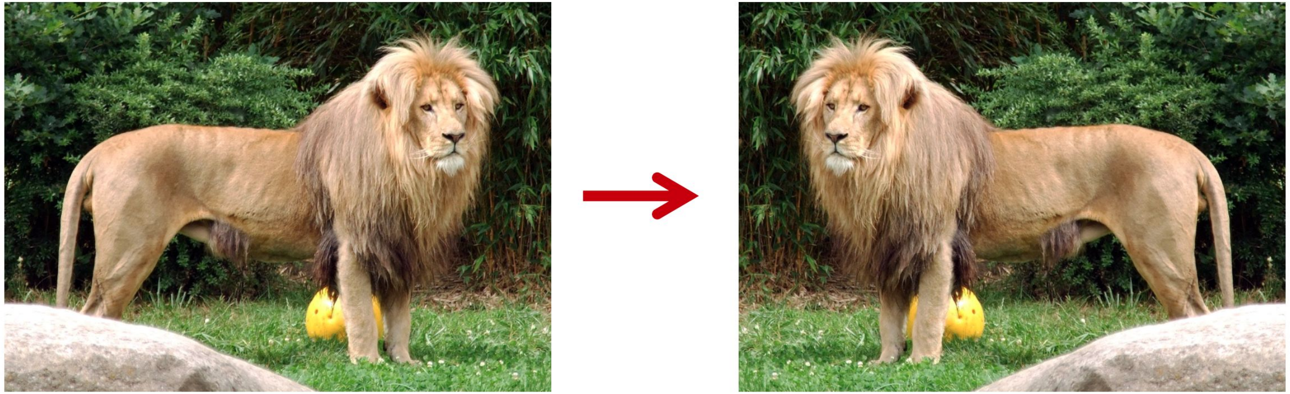
- Types of augmentation:
  - Rotation
  - Horizontal/vertical flip
  - Random crops/scales
    - Zoom
    - Width or height shifts
  - Shearing
  - Brightness, contrast, saturation
  - Lens distortions

Types of data augmentation

1. Rotations

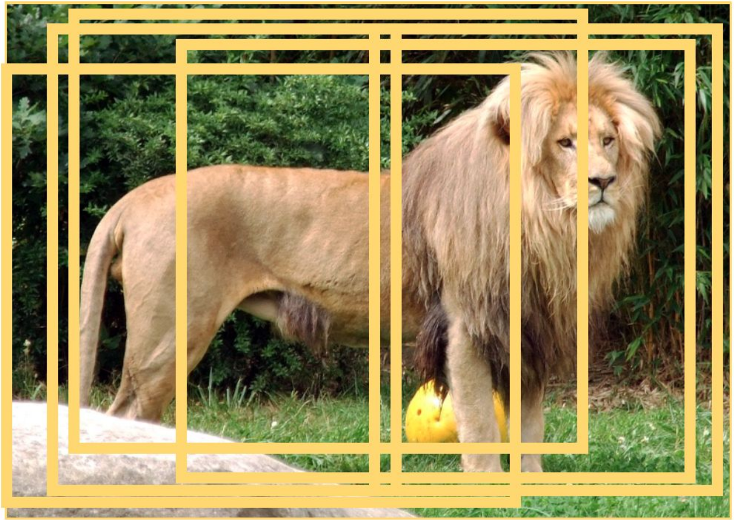
Types of data augmentation

2. Horizontal/Vertical Flips



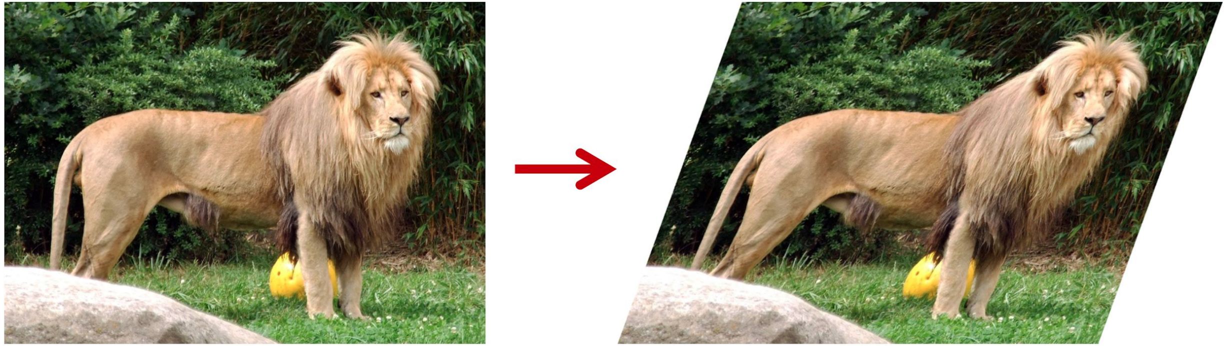
Types of data augmentation

3. Random crops/scales



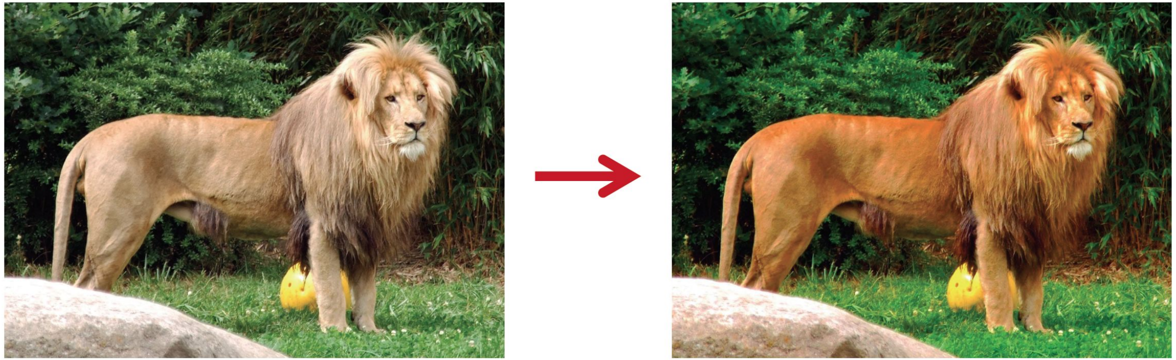
Types of data augmentation

4. Shearing



Types of data augmentation

5. Brightness, contrast, saturation

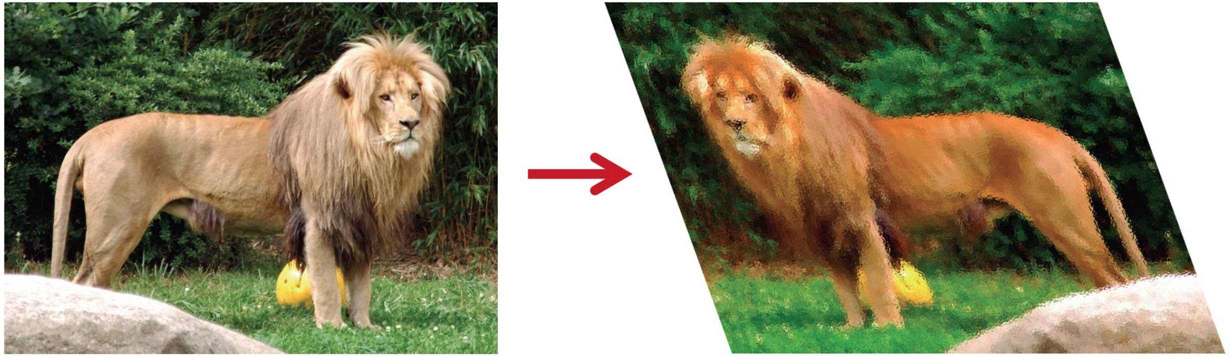


Types of data augmentation

6. Lens distortions

Types of data augmentation

7. Combinations of the above



Data Augmentation

- If you train a network using data augmentation, it will never see the same input twice, but the inputs will still be heavily correlated
    - You're remixing known information, not producing new information

- May not completely escape overfitting due to this correlation

- Adding dropout can also help

Data Augmentation in Keras

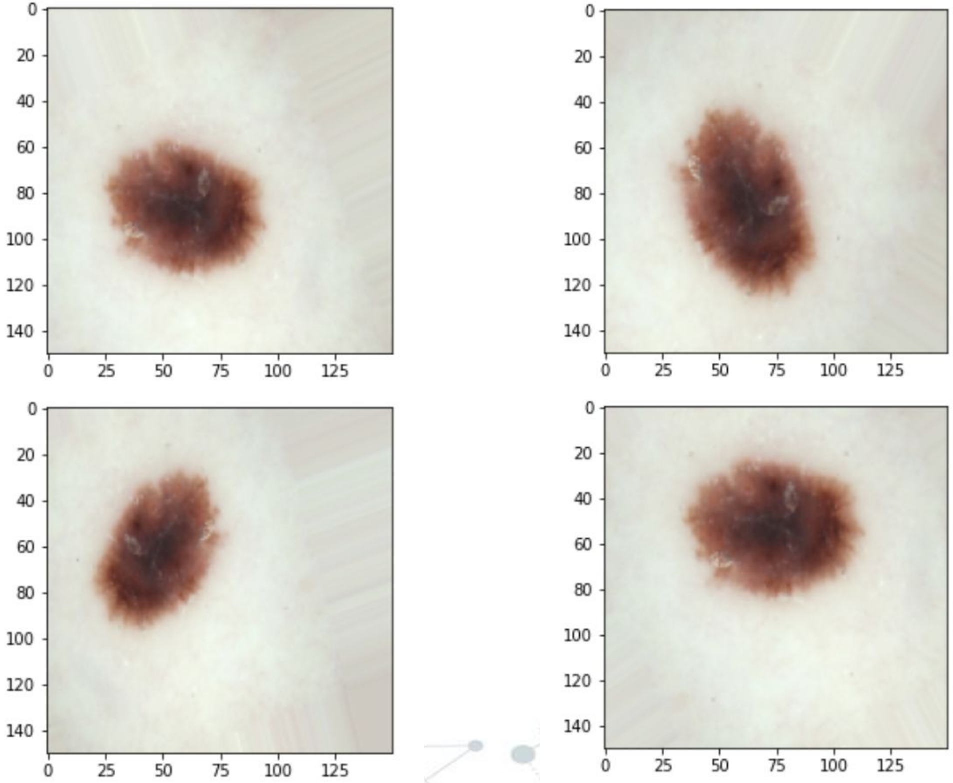
```python
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        rotation_range = 40,
        width_shift_range = 0.2,
        height_shift_range = 0.2,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True,
        fill_mode = 'nearest')
```

You can create your own data generator with any specifications you'd like. The values chosen here are arbitrary.

You can check out the Keras documentation to see all of the available options and values each type of augmentation type can take.

Note that only your training data should be augmented - not the test or validation sets. The point of augmentation is to "increase" your training set size.

Data Augmentation in Keras

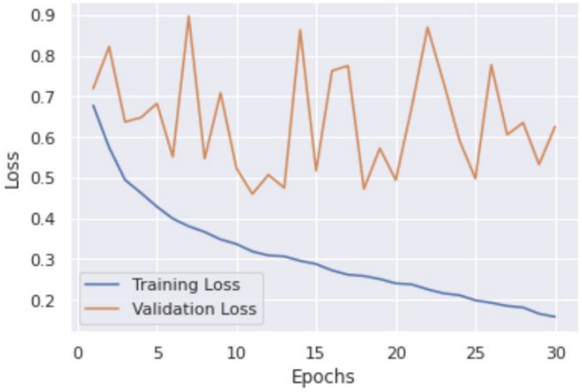


Back to the notebook

[Skin lesions with data augmentation]()

Data Augmentation in Keras

Without augmentation



With augmentation