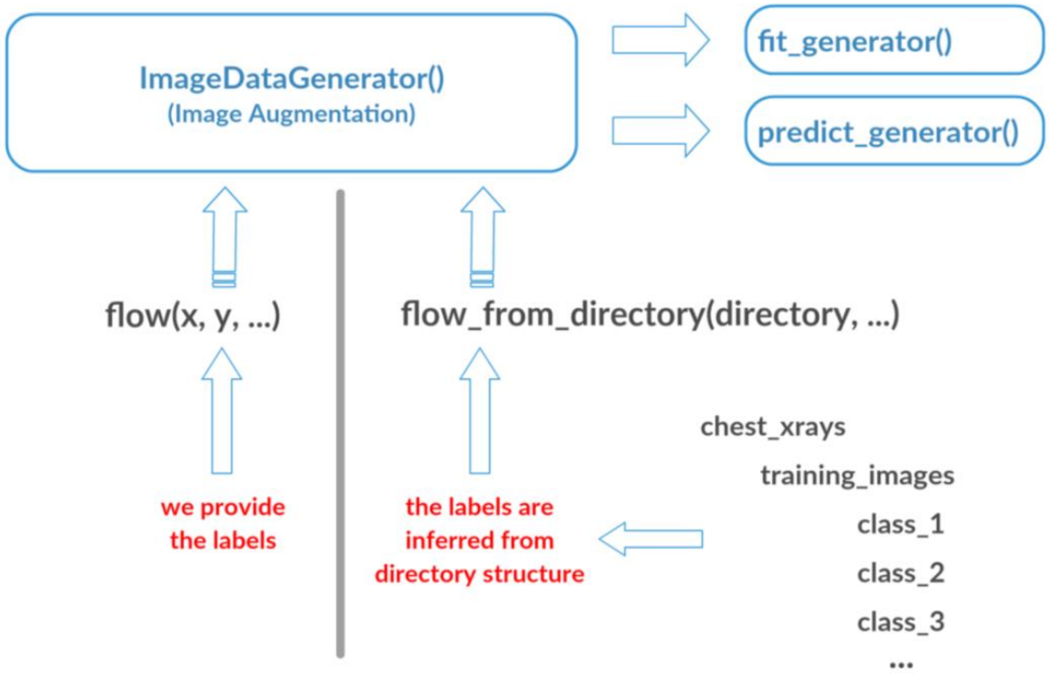
- Organizing dataset files for training
- ConvNet example
  - training binary classifier with dogs-cats dataset
- Data augmentation

Training on large datasets

- Small datasets like *mnist* and *imdb* can be loaded into memory.
- For most cases, datasets are larger than computer memory
- They are generally brought in parts rather than bringing all data into memory.



flow_from_directory() infers the labels from directory structure

Kaynak: https://medium.com/difference-engine-ai/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2

Organizing dataset files for training

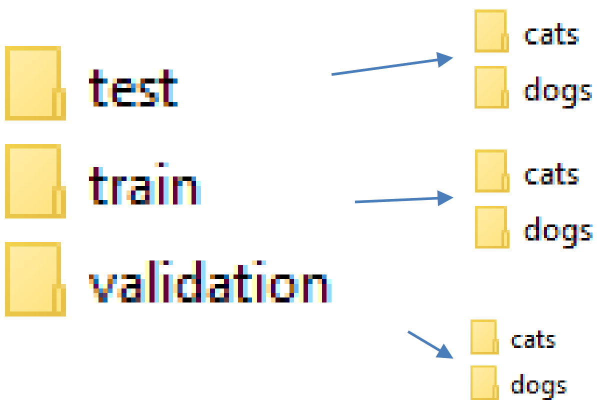
- To *fit()*, or *fit_generator()* using *flow()* via *ImageDataGenerator()*, we supply the labels ourselves.
- *flow_from_directory()* automatically infers the labels from the directory structure of the folders containing images. Every subfolder inside the training-folder(or validation-folder) will be considered a target class.

Template	Example
Data Folder	chest_xrays
Training Image Folder	training_images
Folder 1	class_1
Folder 2	class_2
Folder 3	class_3
•••	•••

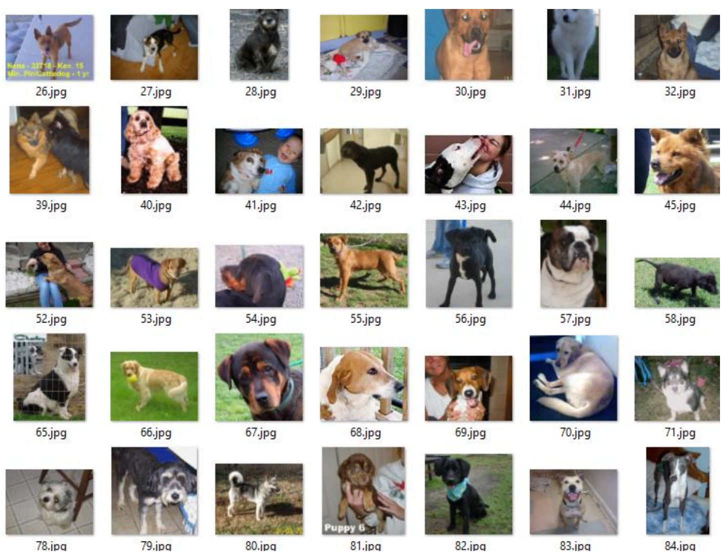
flow_from_directory() automatically infers the labels from the directory structure of the folders

Kaynak: https://medium.com/difference-engine-ai/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2

Organizing the dataset

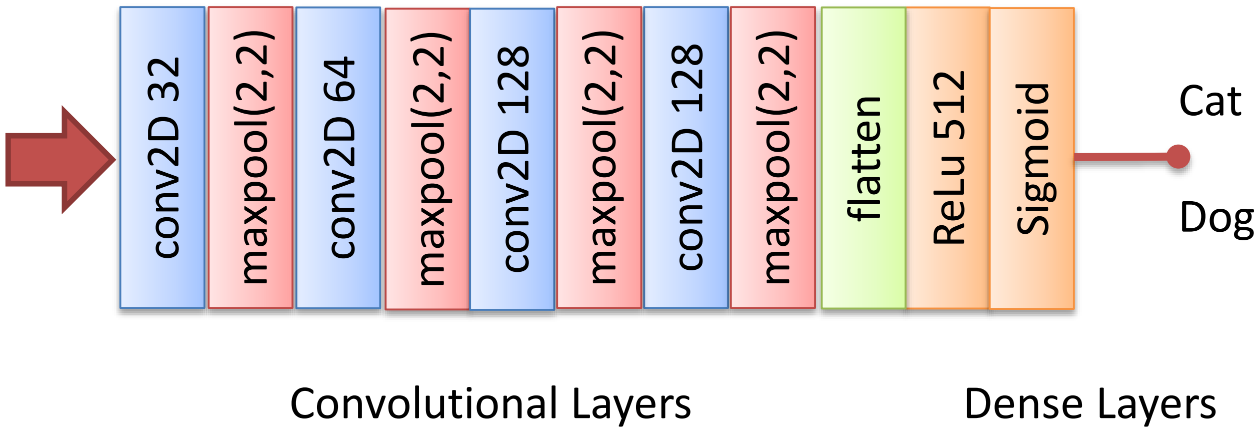


- A subset of images from the dataset will be used:
  - Training : 2000 images
  - Validation: 1000 images
  - Test: 1000 images
  - https://www.kaggle.com/c/dogs-vs-cats/data



https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.2-using-convnets-with-small-datasets.ipynb

Determining the model




```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-4),metrics=['acc'])
```

Exercise: compute the number of parameters for each layer of the network without using summary() function.

Determining the model

```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
________________________________________________________________
```

Input image is filtered 32 time with differerent 3*3 kernels. Also input image is in RGB format which mean there different kernels for each channels. (32*3*3)*3+32=896

Second layer has 32 input and 64 kernels. 32*64*3*3+32= 18496

64*128*3*3+128= 73856

128*128*3*3+128= 147584

6272*512+512 = 3211776

512*1+1 =513

Kaynak: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.2-using-convnets-with-small-datasets.ipynb

Örnek: ImageDataGenerator

```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),
        #No. of images to be yielded from the generator per batch.
        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')
```

Min-max normalization

All images are scaled to a fixed size

Each batch contains 20 training images.

Binary classification mode

Similar operations for validation

Örnek: Veri zenginleştirme

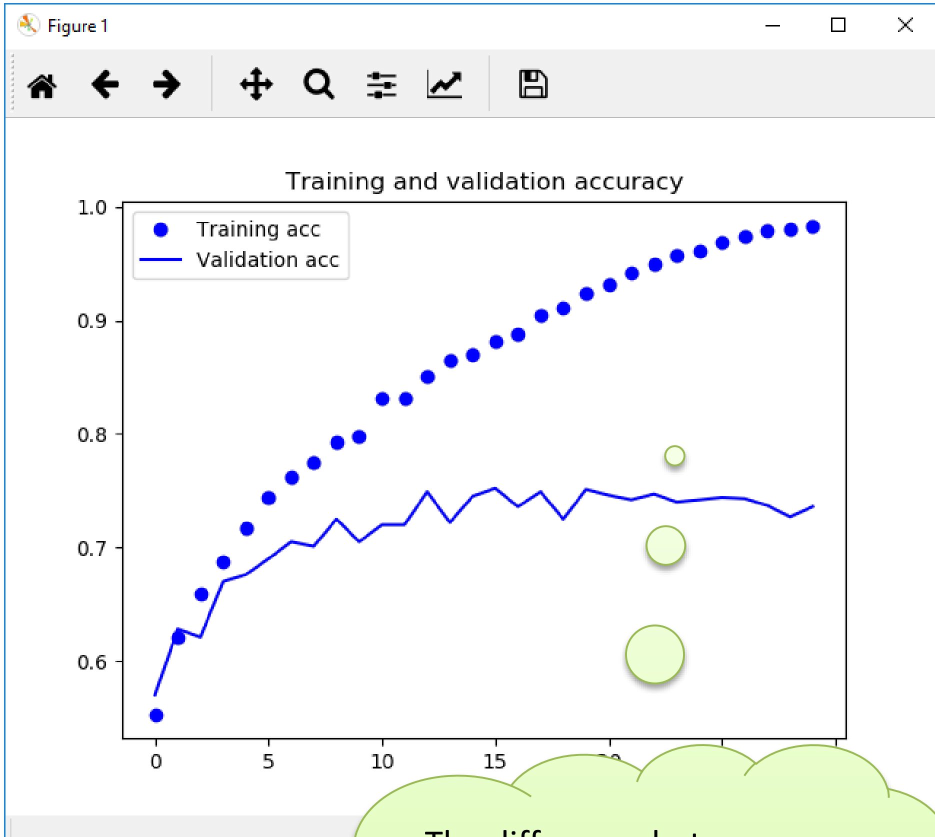
fit_generator is used for previous versions of Tensorflow

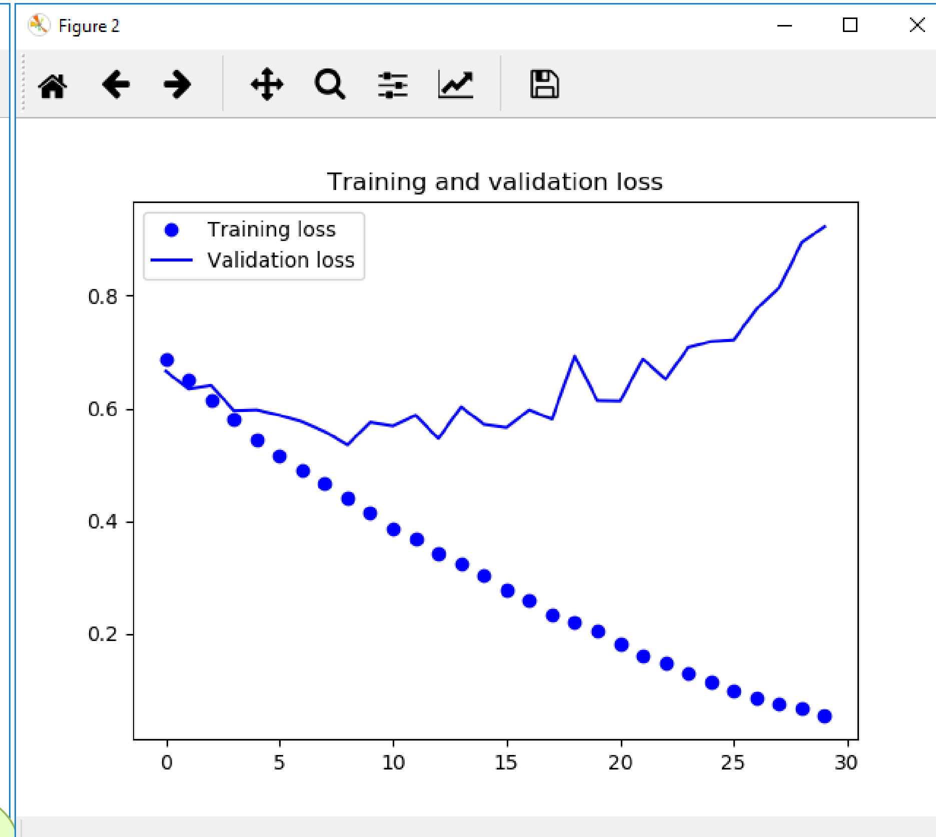
steps_per_epoch : It specifies how many times the weights updated in an epoch.
steps_per_epoch = TotalTrainingSamples / TrainingBatchSize

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)


model.save('cats_and_dogs_small_1.h5')
```

validation_steps = TotalvalidationSamples / ValidationBatchSize

Training results





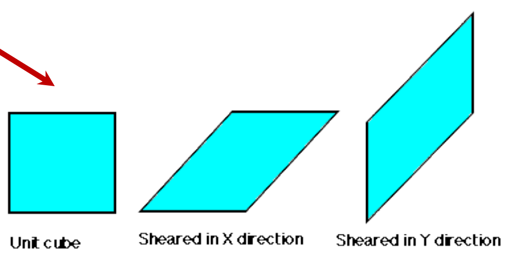
The difference between training and validation accuracies can be reduced by data enrichment.

ImageDataGenerator class

- With data augmentation, the ability of the model to generalize is increased and overfitting is reduced.

- **ImageDataGenerator class:** Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches).
- **rotation_range:** is a value in degrees (0-180), a range within which to randomly rotate pictures.
- **width_shift and height_shift:** are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- **shear_range:** is for randomly applying shearing transformations.
- **zoom_range:** is for randomly zooming inside pictures.
- **horizontal_flip:** is for randomly flipping half of the images horizontally -- relevant when there are no assumptions of horizontal asymmetry (e.g. real-world pictures).
- **fill_mode:** is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

```
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')
```



Additional source to read: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

Example

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
```

Data augmentation

Veri zenginleştirme (data augmentation)

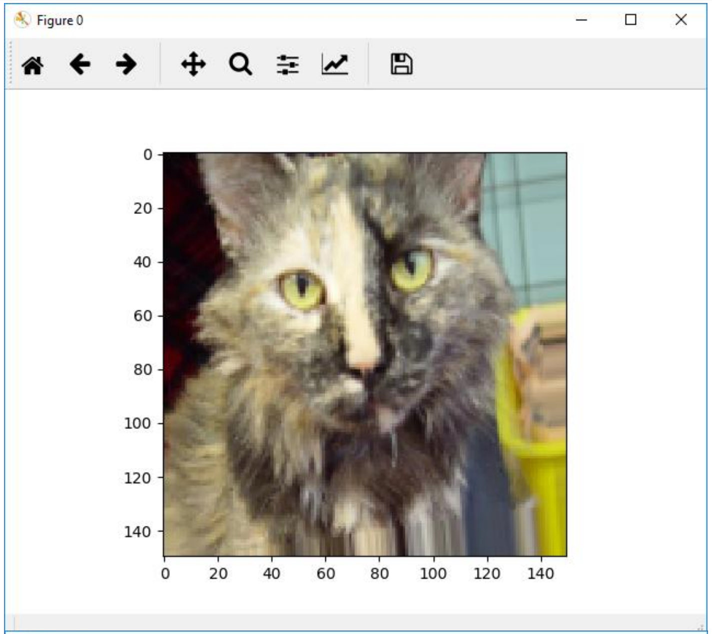
```python
fnames = [os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)]
# We pick one image to "augment"
img_path = fnames[10]

# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)
# The .flow() command below generates batches of randomly transformed images.
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```

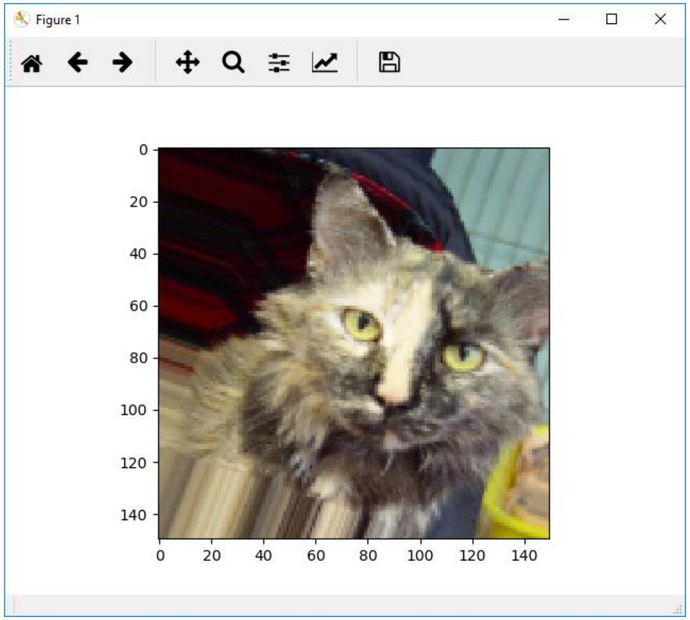
Data augmentation on an example image

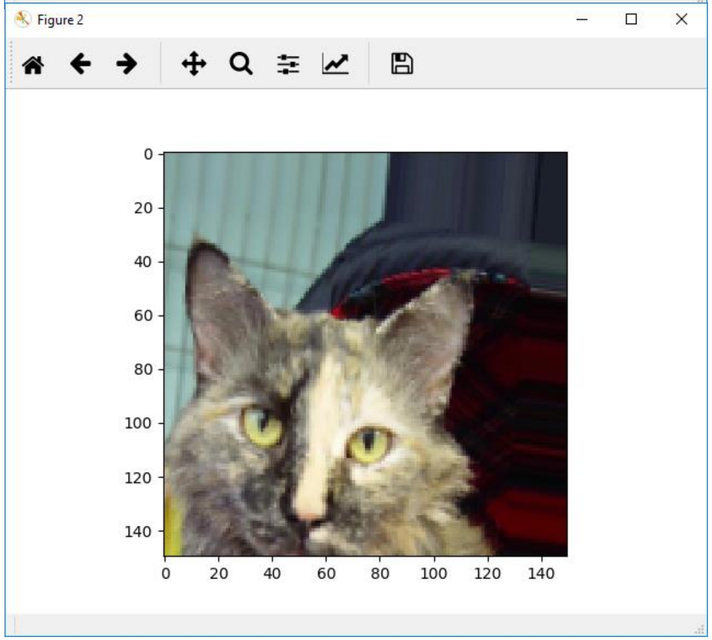
fill_mode: One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode:

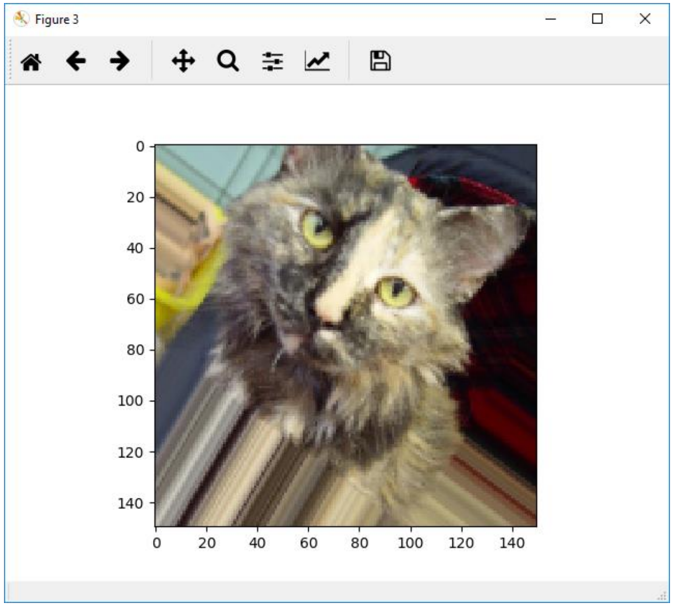
- 'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k)
- 'nearest': aaaaaaaa|abcd|dddddddd
- 'reflect': abcddcba|abcd|dcbaabcd
- 'wrap': abcdabcd|abcd|abcdabcd

Data augmentation



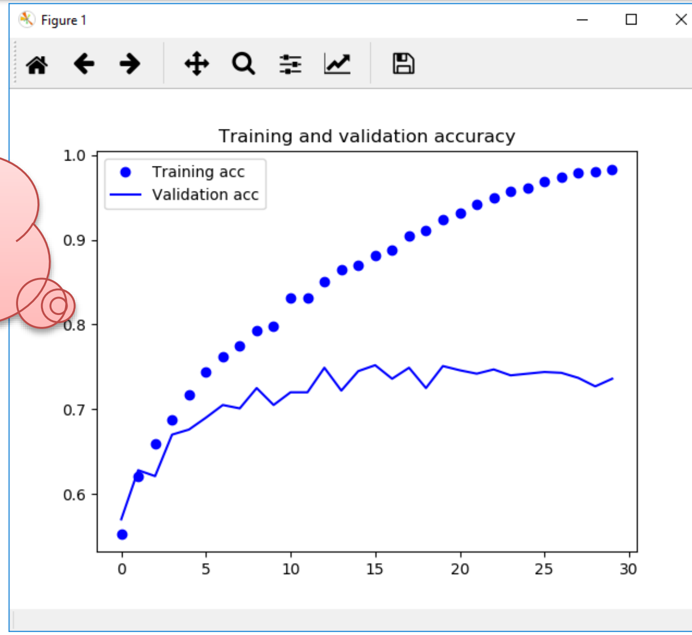


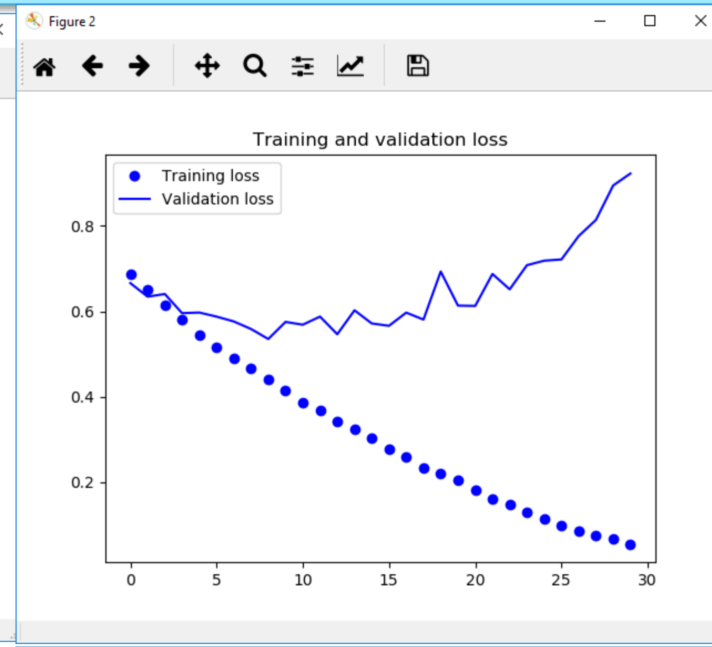




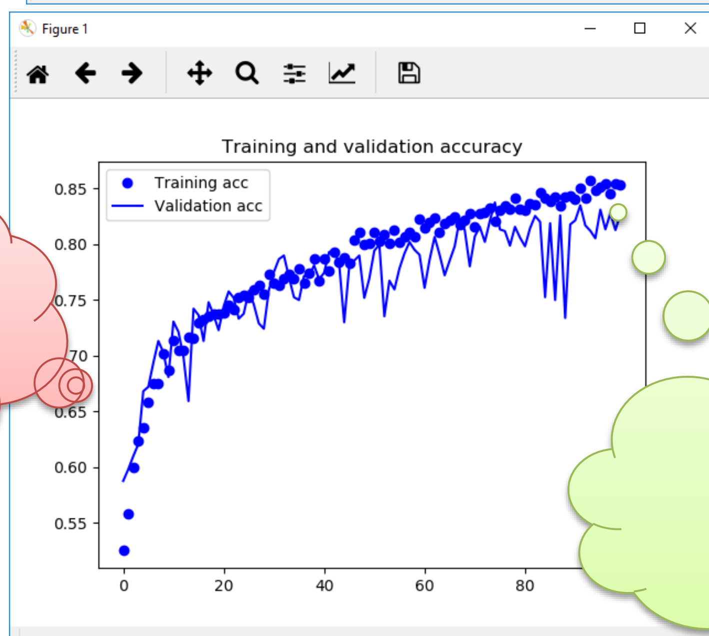
Training results after data augmentation

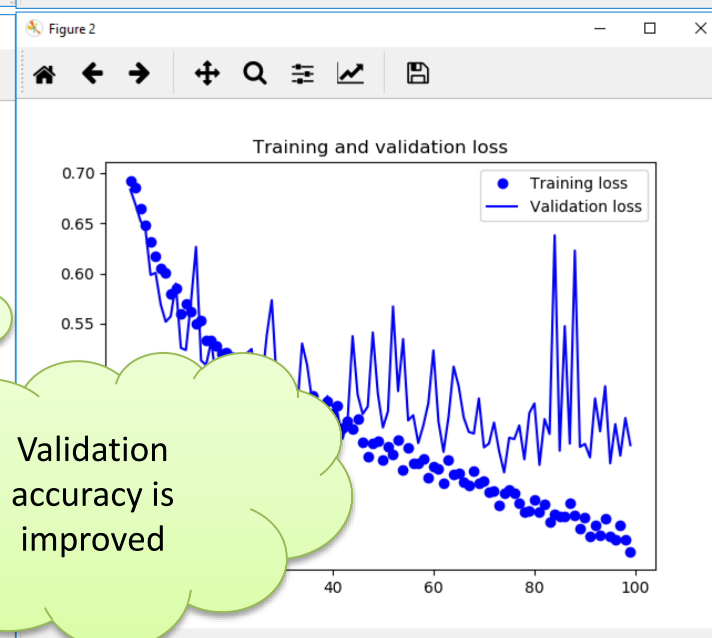
Before data augmentation





After data augmentation





Validation accuracy is improved