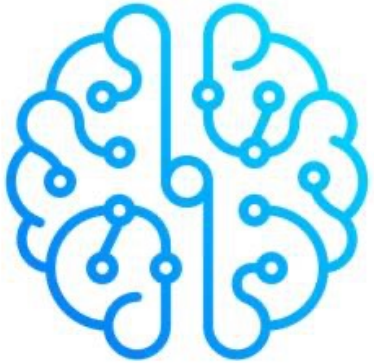


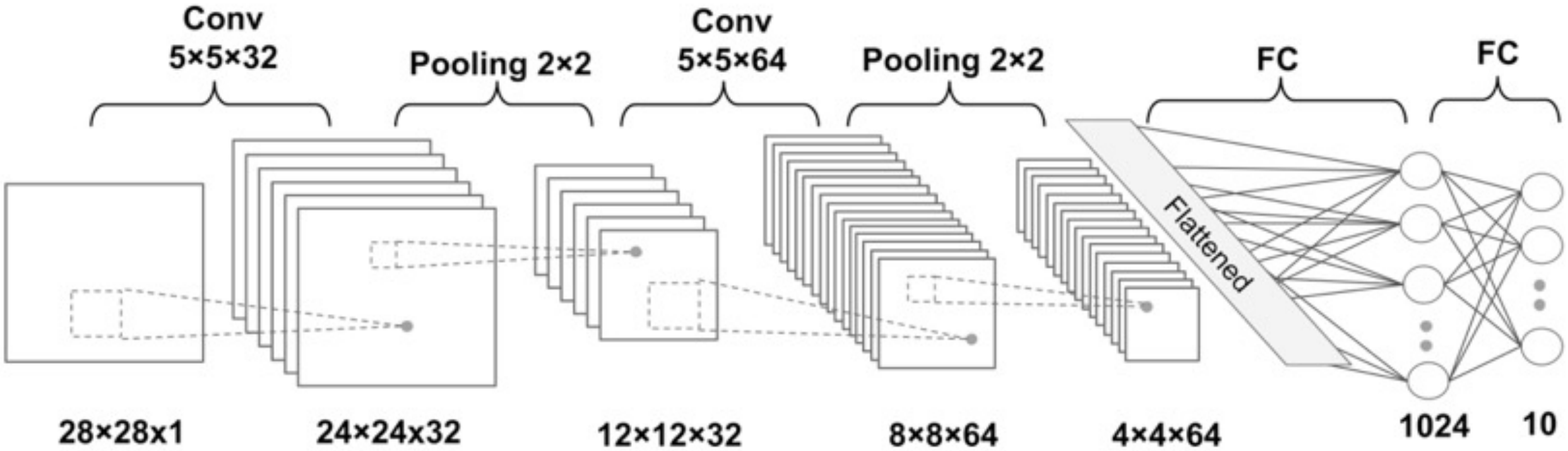
Deep Convolutional Neural Networks – Part 2

Machine Learning for Engineering Applications

Fall 2023

DCNN + TensorFlow

- Input layer: 28x28 pixel resolution images (MNIST – digits)
- Batch size: 28 x 28 x 1 (one channel – gray images)
- Kernel size: 5x5
- 1st conv: 32 output feature maps
- 2nd conv: 64 output feature maps
- max-pooling after each conv-layer
- Two dense-layers (1st-1024, 2nd 10 [softmax])


Conv
5×5×32
Pooling 2×2
Conv
5×5×64
Pooling 2×2
FC
FC
Flattened
28×28x1
24×24x32
12×12×32
8×8×64
4×4×64
1024
10

- **Tensor dimensions**:

  - Input: [batchsize× 28× 28×1]
  - Conv_1: [batchsize× 24× 24×32]
  - Pooling_1: [batchsize×12×12×32]

  - Conv_2: [batchsize×8×8× 64]
  - Pooling_2: [batchsize× 4× 4× 64]

  - FC_1: [batchsize×1024]

  - FC_2 and softmax layer: [batchsize×10]

- Loading the MNIST dataset:

```
X_data, y_data = load_mnist('./mnist/', kind='train')
print('Rows: {}, Columns: {}'.format(X_data.shape[0], X_data.shape[1]))
      Rows: 60000, Columns: 784
X_test, y_test = load_mnist('./mnist/', kind='t10k')
print('Rows: {}, Columns: {}'.format( X_test.shape[0], X_test.shape[1]))
      Rows: 10000, Columns: 784
X_train, y_train = X_data[:50000,:], y_data[:50000]
X_valid, y_valid = X_data[50000:,:], y_data[50000:]

print('Training: ', X_train.shape, y_train.shape)
      Training: (50000, 784) (50000,)
print('Validation: ', X_valid.shape, y_valid.shape)
      Validation: (10000, 784) (10000,)
print('Test Set: ', X_test.shape, y_test.shape)
      Test Set: (10000, 784) (10000,)
```

- Data preprocessing

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) =
                                                    ...mnist.load_data()

# training tensor and normalization
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

# test tensor and normalization
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

# string to numerical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

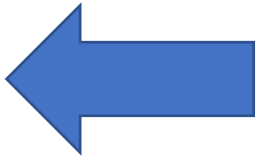
- Straight forward CNN Layers (*think of the MNIST dataset*)

```
from keras import layers
from keras import models

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
                                    ...input_shape=(28, 28, 1)))
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(64, (3, 3), activation='relu'))
```

- Print out the summary of your CNN model

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

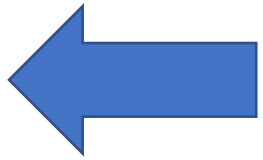


- Adding the dense layers

```
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        ...input_shape=(28, 28, 1)))
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(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

- Adding the dense layers

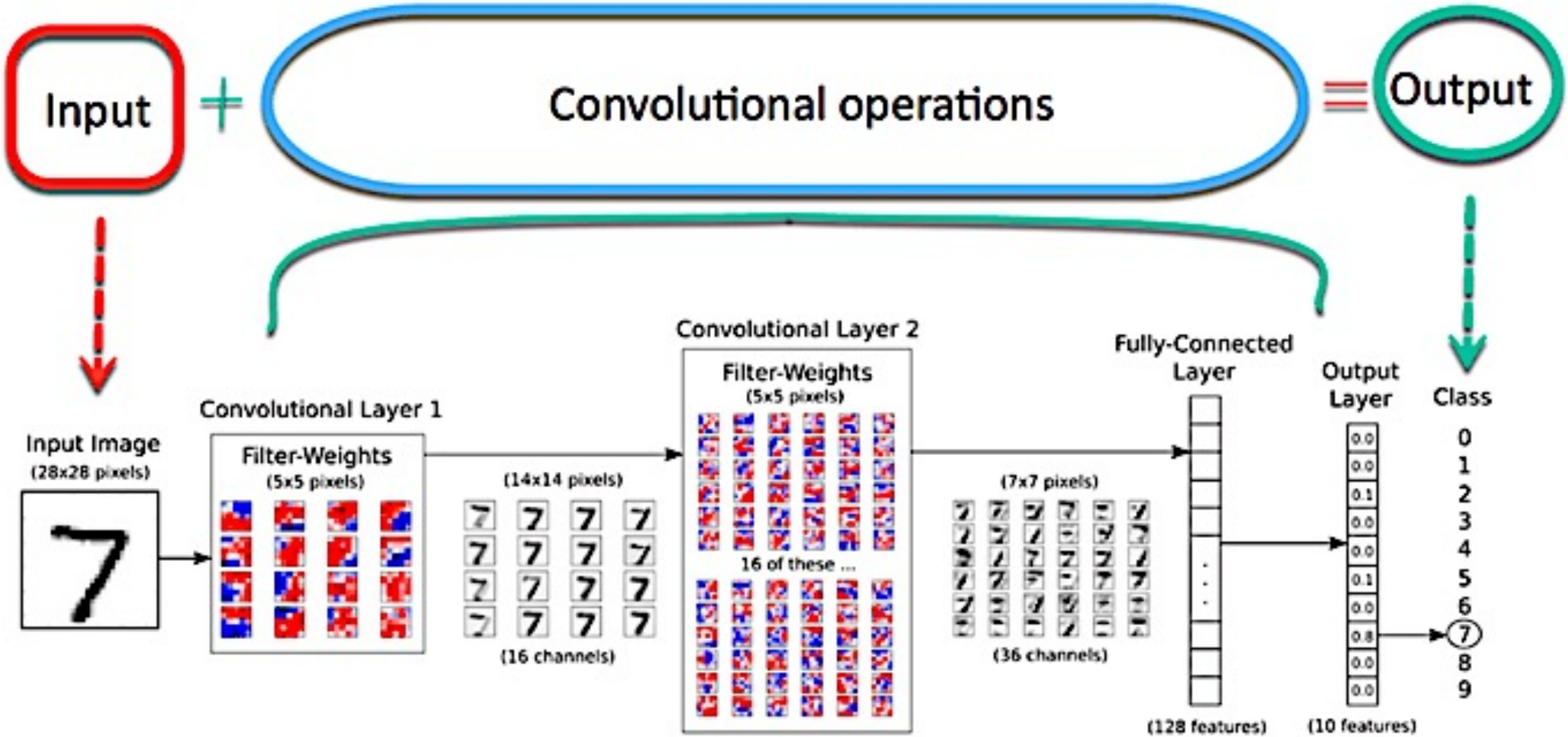
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```



- Optimize / Train / Test

```
model.compile(optimizer='rmsprop’,
     ...loss='categorical_crossentropy’, metrics=['accuracy’])

model.fit(train_images, train_labels, epochs=5, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
Epoch 1/5
938/938 [==============================] - 54s 57ms/step -
loss: 0.1606 - accuracy: 0.9492
Epoch 2/5
938/938 [==============================] - 55s 59ms/step -
loss: 0.0457 - accuracy: 0.9857
Epoch 3/5
938/938 [==============================] - 53s 57ms/step -
loss: 0.0319 - accuracy: 0.9902
Epoch 4/5
938/938 [==============================] - 52s 55ms/step -
loss: 0.0227 - accuracy: 0.9930
Epoch 5/5
938/938 [==============================] - 46s 49ms/step -
loss: 0.0172 - accuracy: 0.9945
313/313 [==============================] - 2s 8ms/step - loss:
0.0330 - accuracy: 0.9911
```


Input
Convolutional operations
Output
Input Image
(28x28 pixels)
Convolutional Layer 1
Filter-Weights
(5x5 pixels)
(14x14 pixels)
(16 channels)
Convolutional Layer 2
Filter-Weights
(5x5 pixels)
16 of these ...
(7x7 pixels)
(36 channels)
Fully-Connected
Layer
(128 features)
Output
Layer
(10 features)
Class
0
1
2
3
4
5
6
7
8
9

CATS vs. DOGS

- Dataset: https://www.kaggle.com/c/dogs-vs-cats/data







- How to deal with images from other sources?
- How do you organize and format the images to your needs?
- How do you split training / validation / testing?
- Sometimes:
  - The longer and organized the code is, the less likely to make a mistake….just comment a lot.

```
import os, shutil

original_dataset_dir = '/home/dvalles/Downloads/kaggle_original_data'

base_dir = '/home/dvalles/Downloads/cats_and_dogs_small'
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)
```

Directories for the training / validation / test splits

Own training folders for cats and dogs

```
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)
```

Own validation folders for cats and dogs

```
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```

Own test folders for cats and dogs

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
      src = os.path.join(original_dataset_dir, fname)
      dst = os.path.join(train_cats_dir, fname)
      shutil.copyfile(src, dst)
```

1st 1000 cat copies to the training folder for cat

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
      src = os.path.join(original_dataset_dir, fname)
      dst = os.path.join(validation_cats_dir, fname)
      shutil.copyfile(src, dst)
```

The next 500 cat copies to the validation folder for cat

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
      src = os.path.join(original_dataset_dir, fname)
      dst = os.path.join(test_cats_dir, fname)
      shutil.copyfile(src, dst)
```

The next 500 cat copies to the test folder for cat

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
      src = os.path.join(original_dataset_dir, fname)
      dst = os.path.join(train_dogs_dir, fname)
      shutil.copyfile(src, dst)
```

1st 1000 dog copies to the training folder for cat

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
      src = os.path.join(original_dataset_dir, fname)
      dst = os.path.join(validation_dogs_dir, fname)
      shutil.copyfile(src, dst)
```

The next 500 dog copies to the validation folder for cat

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
      src = os.path.join(original_dataset_dir, fname)
      dst = os.path.join(test_dogs_dir, fname)
      shutil.copyfile(src, dst)
```

The next 500 dog copies to the test folder for cat

```
>>> print('total training cat images:', len(os.listdir(train_cats_dir)))
total training cat images: 1000

>>> print('total training dog images:', len(os.listdir(train_dogs_dir)))
total training dog images: 1000

>>> print('total validation cat images:', len(os.listdir(validation_cats_dir)))
total validation cat images: 500

>>> print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
total validation dog images: 500

>>> print('total test cat images:', len(os.listdir(test_cats_dir)))
total test cat images: 500

>>> print('total test dog images:', len(os.listdir(test_dogs_dir)))
total test dog images: 500
```

Preprocess the images

- Read the picture files.
- Decode the JPEG content to RGB grids of pixels.
- Convert these into floating-point tensors.
- Rescale the pixel values (between 0 and 255) to the [0, 1] interval

- Preprocess the images

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

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
      train_dir,
      target_size=(150, 150)
      batch_size=32,
      class_mode='binary’)

validation_generator = test_datagen.flow_from_directory(
      validation_dir,
      target_size=(150, 150),
      batch_size=32,
      class_mode='binary')
```

- The CNN:

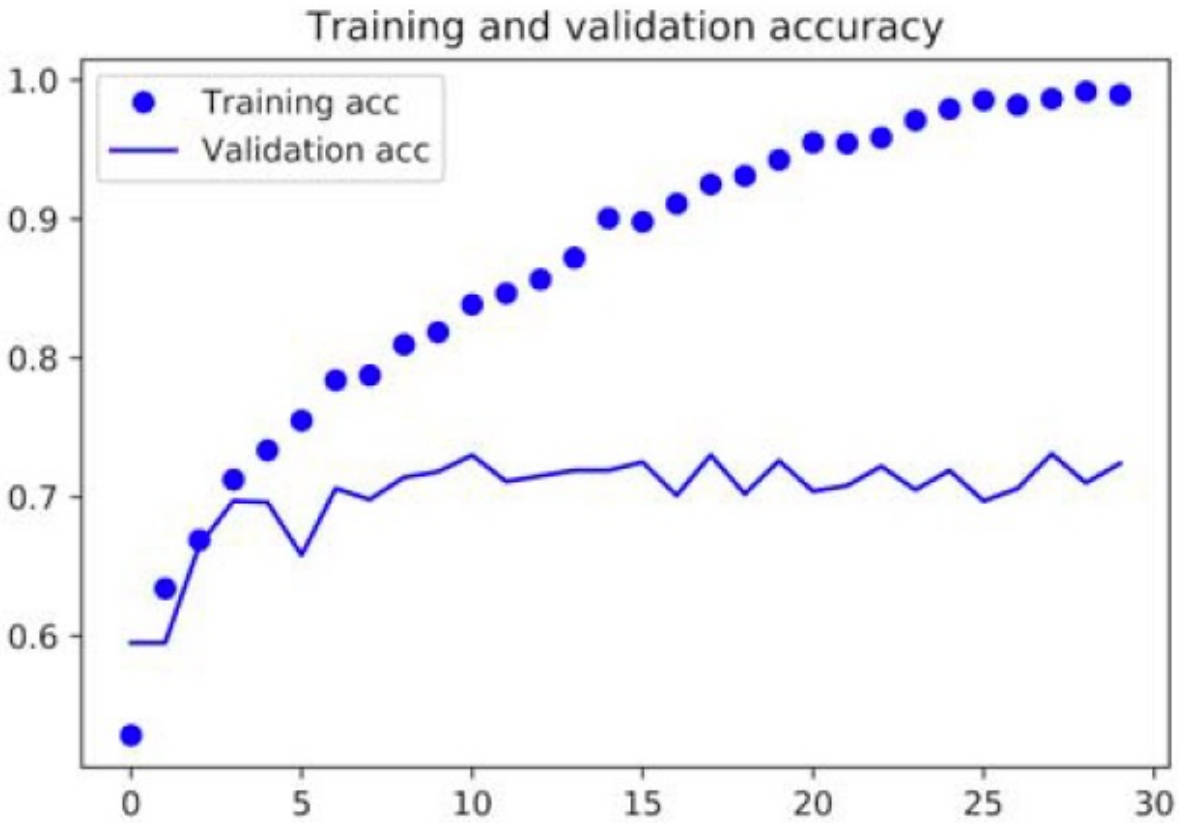
```
from tf.keras import layers
from tf.keras import models

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'))
```

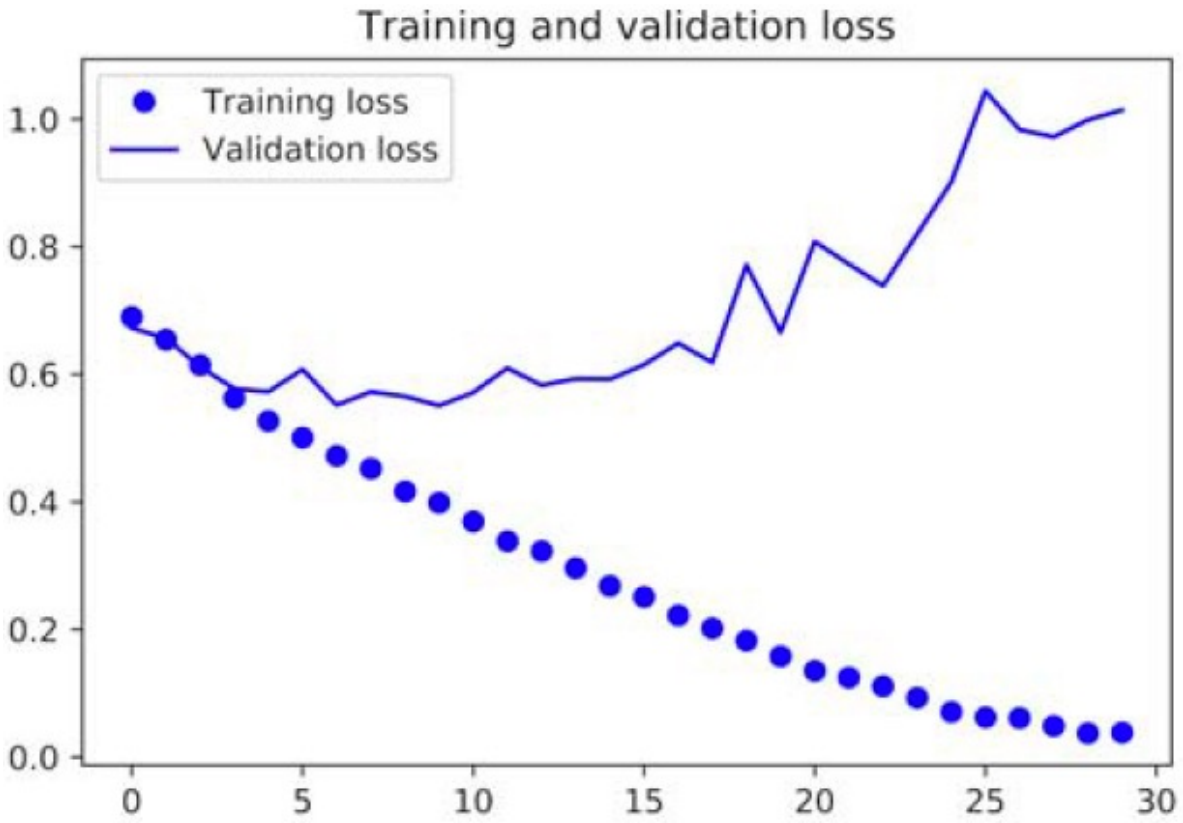
- Optimize / Train & Save

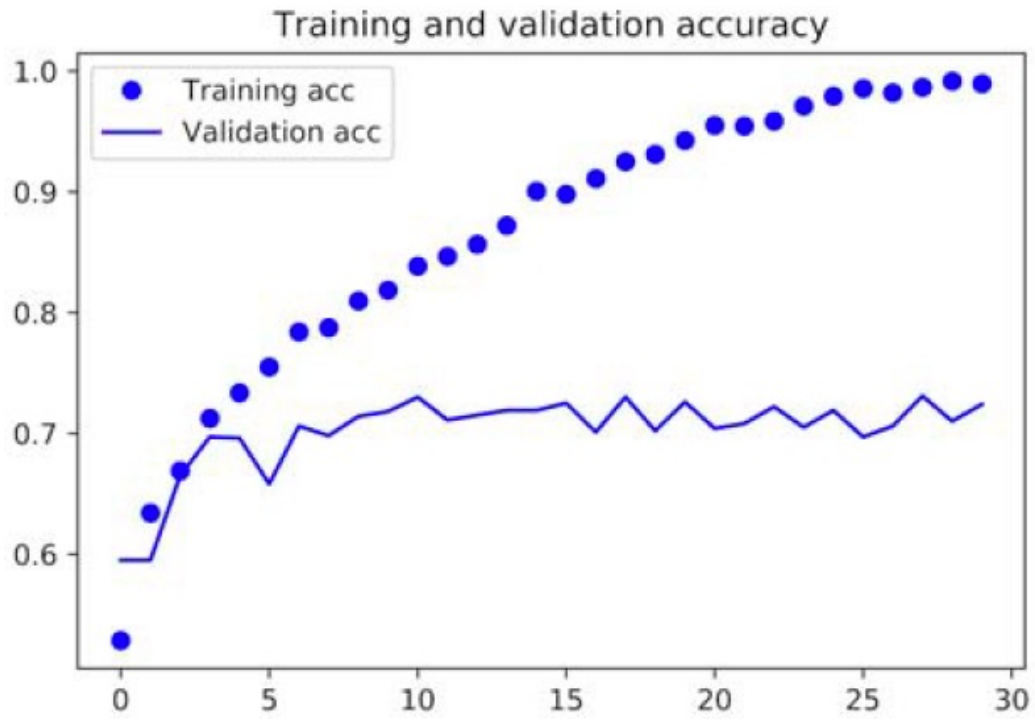
```
from tf.keras import optimizers

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

history = model.fit_generator(
              train_generator,
              steps_per_epoch=100,  #number of gradient step before next epoch
              epochs=30,
              validation_data=validation_generator,
              validation_steps=50)
        # how many batches to draw from the validation generator for evaluation

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

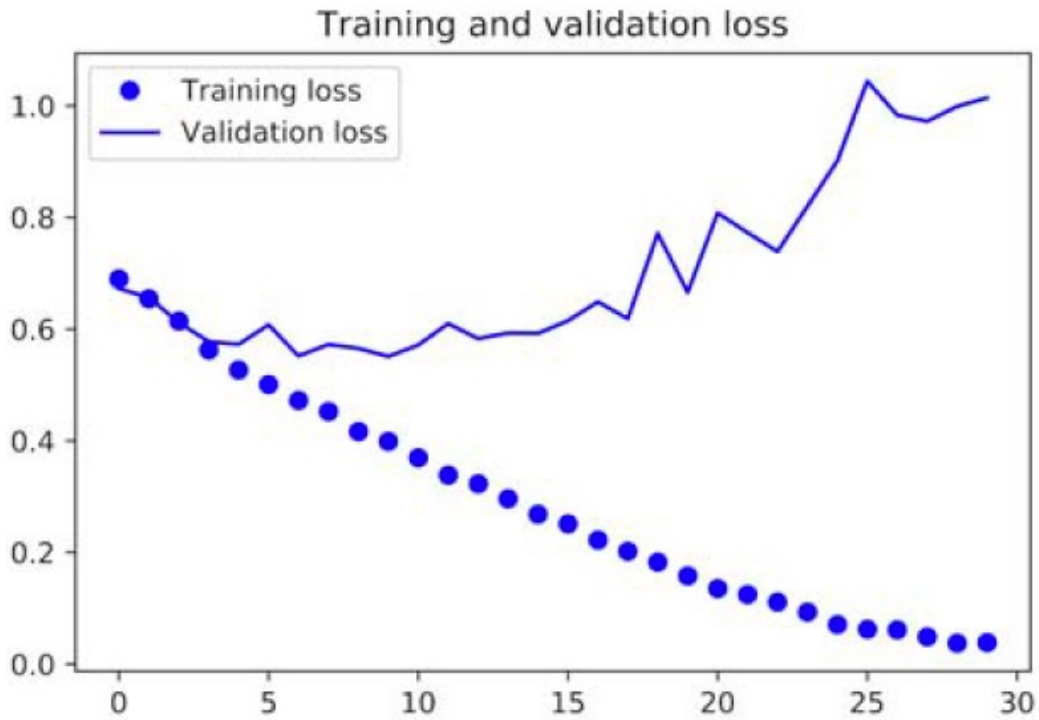
- Plot the accuracy and losses

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30


Training and validation loss
Training loss
Validation loss
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

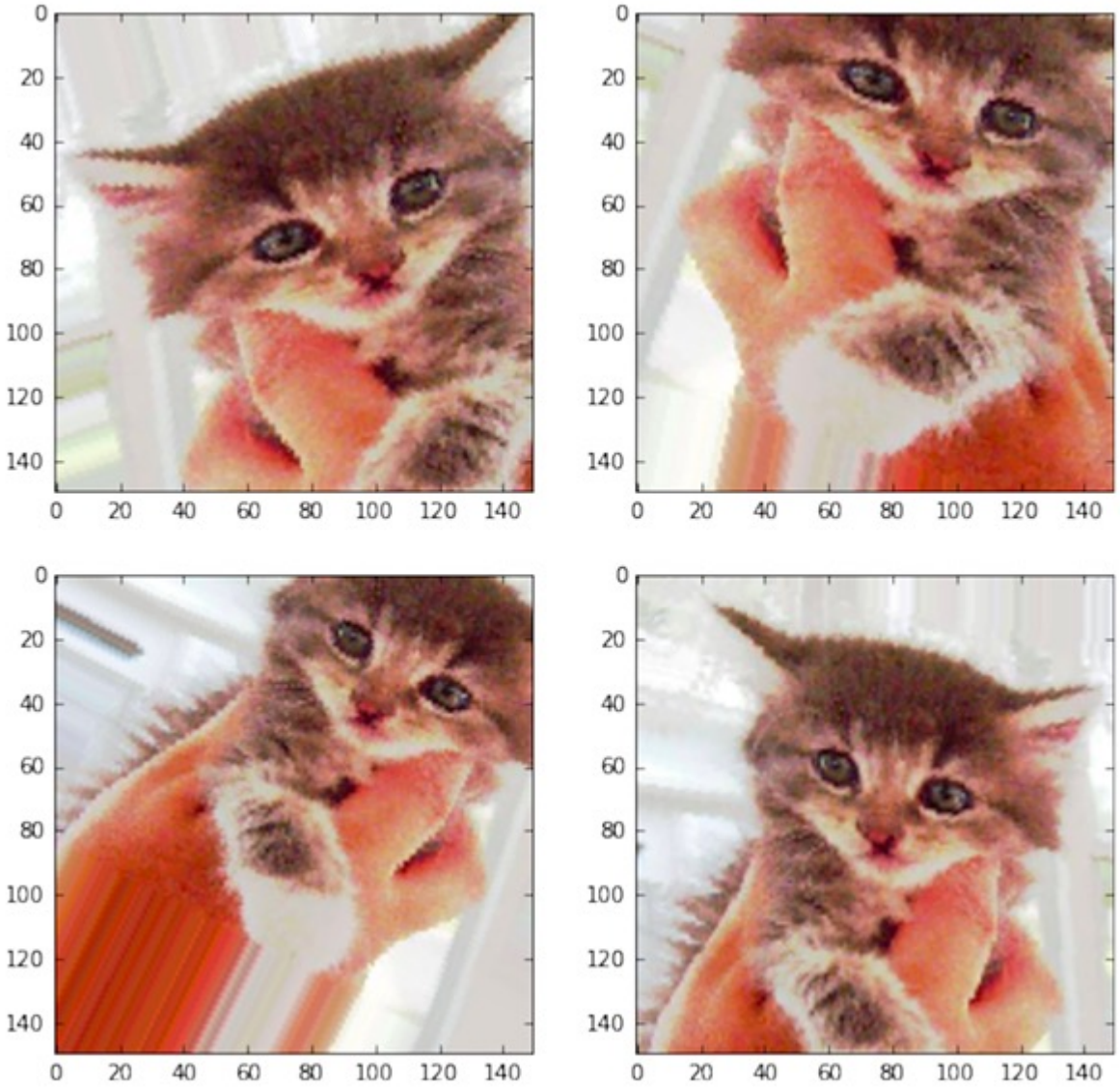




- How to address the problem?
- **Data Augmentation** is a way to compensate the dataset
- Dropout layer were not implemented

- Data Augmentation

```
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’)
```



- Data Augmentation

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.

- Data Augmentation

zoom_range is for randomly zooming inside pictures.

horizontal_flip is for randomly flipping half the images horizontally—relevant when there are no assumptions of horizontal asymmetry (for example, real-world pictures).

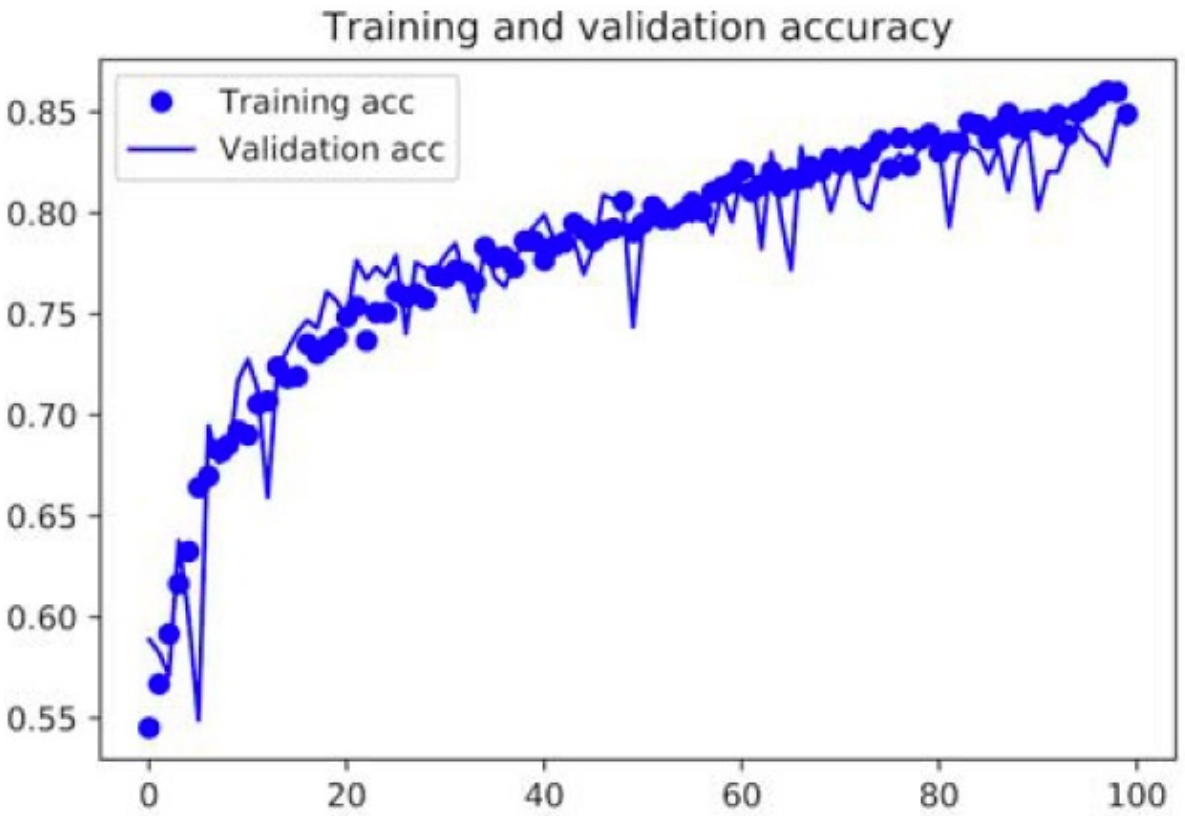
fill_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

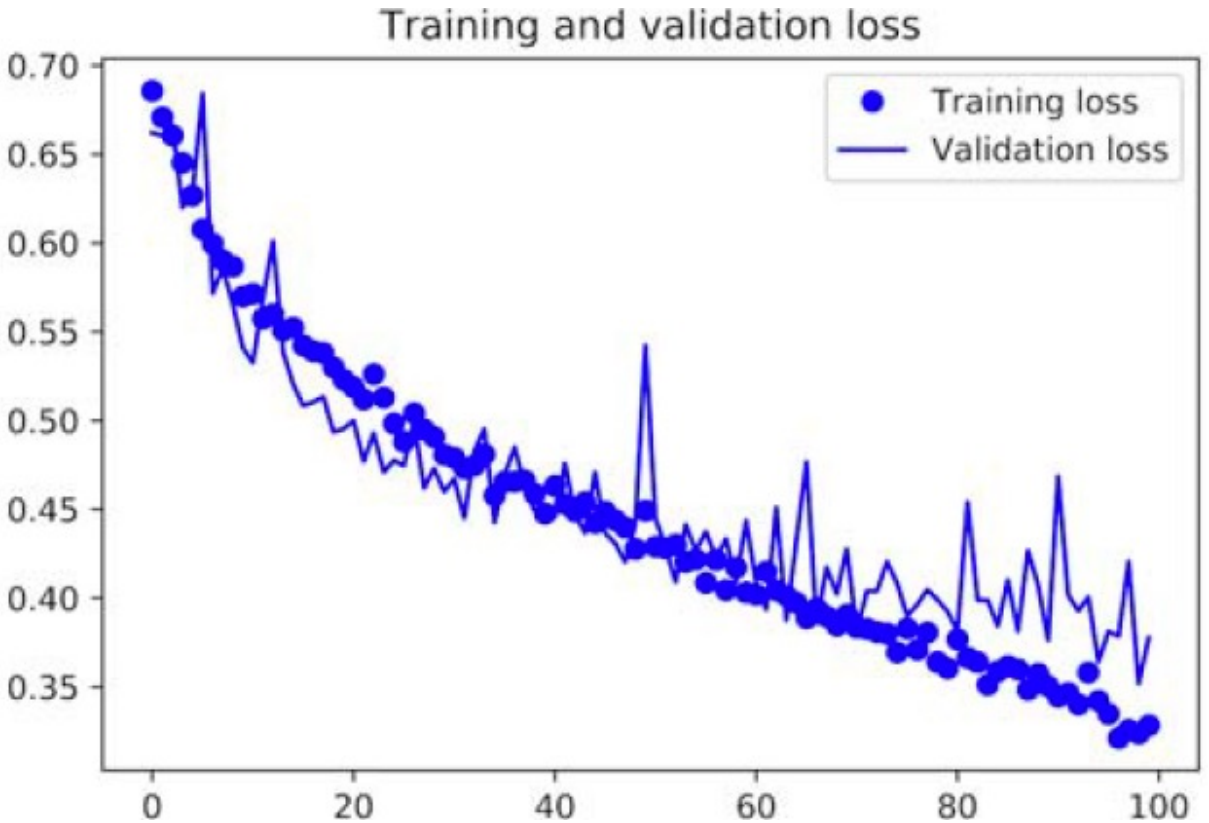
- Data Preprocessing

```
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)

test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(
      train_dir, target_size=(150, 150),
      batch_size=32, class_mode='binary’)


validation_generator = test_datagen.flow_from_directory(
      validation_dir, target_size=(150, 150),
      batch_size=32, class_mode='binary')
```

- Dropout Layers

```
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)) #Try one position, add more if need it
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```


Training and validation accuracy
Training acc
Validation acc
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
20
40
60
80
100


Training and validation loss
Training loss
Validation loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
20
40
60
80
100

MNIST CNN