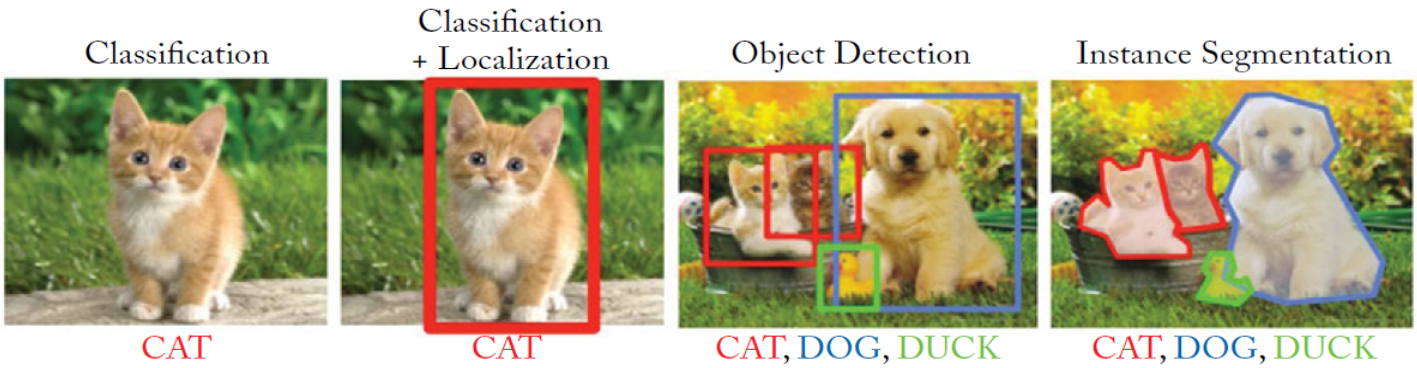
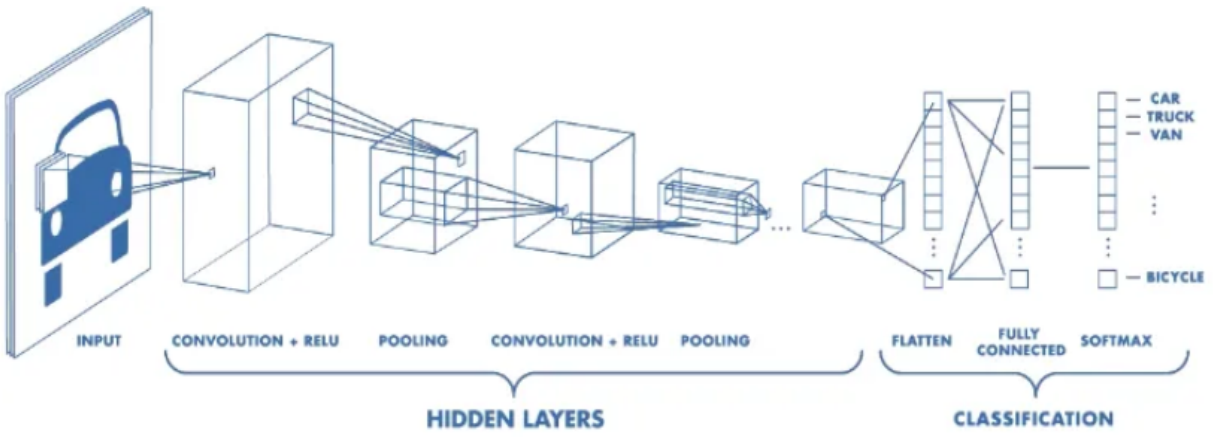
CONVOLUTIONAL NEURAL NETWORK

A **convolutional neural network** (CNN) is one of the most popular types of deep learning algorithms. It works well on images.



A CNN is an excellent tool for (i) image classification (categorization of image: cat/dog/horse), (ii) classification and localization of an object in the image (drawing a **bounding box** around an object and naming the class); (iii) object detection (localization and labeling of all objects present in the image); and (iv) instance segmentation (segmenting individual objects present in the picture).

Historical Note. The earliest form of CNN was the Neocognitron model proposed by Kunihiko Fukushima (Fukushima and Miyake, 1982). The Neocognitron was motivated by the seminal work by David Hubel and Torsten Wiesel (1959) which demonstrated that the neurons in the brain are organized in the form of layers. These layers learn to recognize visual patterns by first extracting local features and subsequently combining them to obtain a higher-level representations.



The architecture of CNN comprises several layers: a **convolution** layer, a **pooling** layer, and a **fully connected** layer.

Convolution Layer

An image of size $r \times c$ pixels is represented by three matrices of size $r \times c$ each, containing the intensity values of red, green, and blue primary colors (numbers between 0 and 255) on the RGB scale. The convolution layer performs a **dot product** between two matrices, where one matrix is the set of **learnable parameters** (otherwise known as a **kernel** or **filter** or **feature detector**), and the other matrix is the restricted portion of the image.

Example. Mathematically, the kernel is a matrix of weights. For example, the following 3×3 kernel detects vertical lines.

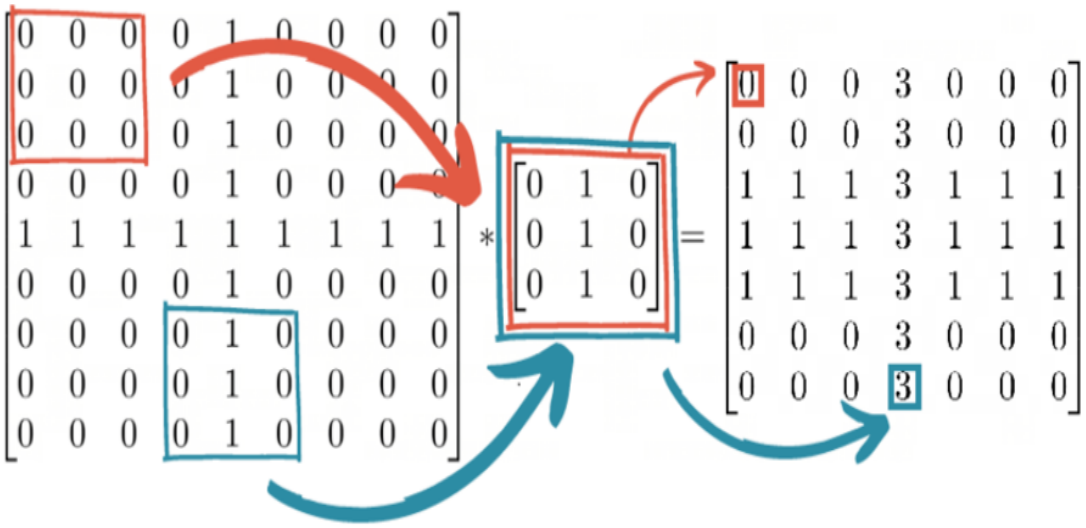
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Suppose we have an 9×9 input image of a plus sign. This has two kinds of lines, horizontal and vertical, and a crossover. In matrix format, the image would look as follows:

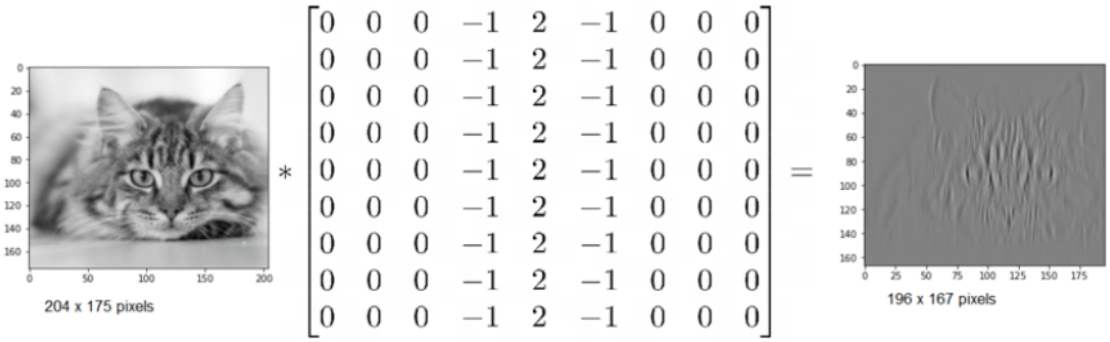
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Suppose we want to test the vertical line detector kernel on the plus sign image. To perform the convolution, we slide the convolution kernel over the image. At each position, we multiply each element of the convolution kernel by the element of the image that it covers and sum the results.

Since the kernel has width 3, it can only be positioned at 7 different positions horizontally in an image of width 9. So the end result of the convolution operation on an image of size 9×9 with a 3×3 convolution kernel is a new image of size 7×7.



To see how it works on an image, we consider a grayscaled image of a tabby cat with dimensions 204×175 pixels, which we can be represented by a matrix with values in the range between 0 and 1, where 1 is white and 0 is black.



Applying the convolution with a sophisticated vertical line detector, a 9×9 convolution kernel, we see that the filter has performed a kind of vertical line detection. The vertical stripes on the tabby cat's head are highlighted in the output. The output image is 8 pixels smaller in both dimensions

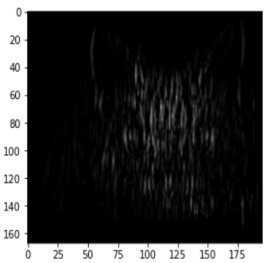
due to the size of the kernel (9×9).

A convolution layer is made up of a series of convolution kernels: a vertical line detector, a horizontal line detector, and various diagonal, border, curve, and corner detectors. These feature detector kernels serve as the first stage of the image recognition process.

Later layers in the neural network can build on the features detected by earlier layers and identify ever more complex shapes.

Activation Function

After passing an image through a convolutional layer, the output is normally passed through an activation function. A common activation function is called **rectified linear unit** (ReLU) defined as $f(x) = \max(0, x)$. The activation function has the effect of adding non-linearity into the convolutional neural network. If the activation function was not present, all the layers of the neural network could be condensed down to a single matrix multiplication. In the case of the cat image above, applying a ReLU function to the first layer output results in a stronger contrast highlighting the vertical lines, and removing the noise originating from other non-vertical features.



Repeating Structure of a CNN

A CNN can be viewed as a series of convolutional layers, followed by an activation function, followed by a **pooling** (**downscaling**) layer, repeated many times.

With the repeated combination of these operations, the first layer detects simple features such as edges in an image, and the second layer begins to detect higher-level features. By the tenth layer, a convolutional neural network can detect more complex shapes such as eyes. By the twentieth layer, it is often able to differentiate faces from one another.

This power comes from the repeated layering of operations, each of which can detect slightly higher-order features than its predecessor.

Once the image is processed through the convolution and pooling layers, it goes into the classification stage consisting of a flatten layer, a connected layer, and a softmax layer. The **flatten** layer collapses the spatial dimensions of the input into a single channel dimension. **Fully connected** layers connect every neuron in one layer to every neuron in another layer. The flattened matrix goes through a fully connected layer to classify the images.

Softmax Function

The **softmax function** is a normalized exponential function that takes as input a vector $(z_1, \ldots, z_K)$ of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, before applying softmax, some vector components could be negative, or greater than one, and might not sum to 1, but after applying softmax, each component will be in the interval $(0, 1)$, and the components will add up to 1, so that they can be interpreted as probabilities. We have

$$softmax(z_1, \ldots, z_K) = \Big(\frac{e^{z_1}}{\sum_{j=1}^{K} e^{z_j}}, \ldots, \frac{e^{z_K}}{\sum_{j=1}^{K} e^{z_j}}\Big).$$

Applications of Convolutional Neural Networks

Several companies, such as Tesla and Uber, are using convolutional neural networks as the computer vision component of a self-driving car. A self-driving car's computer vision system must be capable of localization, obstacle avoidance, and path planning.

Say, consider the case of pedestrian detection. A pedestrian is a kind of obstacle that moves. A convolutional neural network must be able to identify the location of the pedestrian and extrapolate their current motion to calculate if a collision is imminent.

A convolutional neural network for object detection is slightly more complex than a classification model, in that it must not only classify an object but also return the four coordinates of its bounding box.

Furthermore, the convolutional neural network designer must avoid unnecessary false alarms for irrelevant objects, such as litter, but also take into account the high cost of miscategorizing a true pedestrian and causing a fatal accident.

A major challenge for this kind of use is collecting labeled training data. Google's Captcha system is used for authenticating websites, where a user is asked to categorize images as fire hydrants, traffic lights, cars, etc. This is actually a useful way to collect labeled training images for purposes such as self-driving cars and Google StreetView.

Another famous application of CNN is that of drug discovery. The first stage of a drug development program is drug discovery, where a pharmaceutical company identifies candidate compounds that

are more likely to interact with the body in a certain way. Testing candidate molecules in pre-clinical or clinical trials is expensive, so it is advantageous to be able to screen molecules as early as possible.

Proteins that play an important role in disease are known as 'targets'. Some targets can cause inflammation or help tumors grow. The goal of drug discovery is to identify molecules that will interact with the target for a particular disease. The drug molecule must have the appropriate shape to interact with the target and bind to it, like a key fitting in a lock.

The San Francisco-based startup Atomwise developed an algorithm called AtomNet, based on a convolutional neural network, which was able to analyze and predict interactions between molecules.

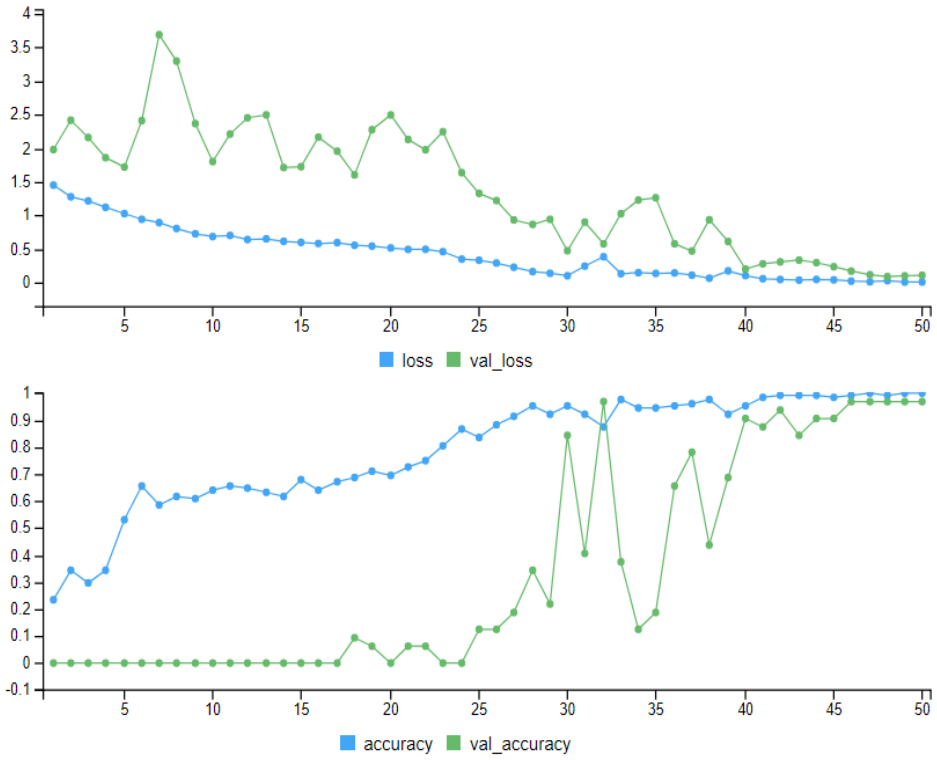
Atomwise was able to use AtomNet to identify lead candidates for drug research programs. AtomNet successfully identified a candidate treatment for the Ebola virus, which had previously not been known to have any antiviral activity. The molecule later went on to pre-clinical trials.

Example. The folder "PlayingCardsImages" contains .jpg images of 43 cards of each suit (clubs, diamonds, hearts, and spades). The following R code splits the data into a training set (40 cards of each suit) and a testing set (3 cards of each suit), then a CNN model is trained on the training set and used to predict the suit for the images in the testing set.

```
library(keras)
library(EBImage)

setwd("./PlayingCardImages/Club")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
club<- cards

setwd("./PlayingCardImages/Heart")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
heart<- cards

setwd("./PlayingCardImages/Spade")
```

```
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
spade<- cards

setwd("./PlayingCardImages/Diamond")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
diamond<- cards

#splitting into training and testing sets and permuting dimensions
train.pool<- c(club[1:40], heart[1:40], spade[1:40], diamond[1:40])
train<- aperm(combine(train.pool), c(4,1,2,3))
test.pool<- c(club[41:43], heart[41:43], spade[41:43], diamond[41:43])
test<- aperm(combine(test.pool), c(4,1,2,3))

#creating image labels
train.y<- c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test.y<- c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train.lab<- to_categorical(train.y)
test.lab<- to_categorical(test.y)

#building model
model.card<- keras_model_sequential()
model.card %>% layer_conv_2d(filters=40, kernel_size=c(4,4),
activation='relu', input_shape=c(100,100,4)) %>%
layer_conv_2d(filters=40, kernel_size=c(4,4), activation='relu') %>%
layer_max_pooling_2d(pool_size=c(4,4))%>% layer_dropout(rate=0.25) %>%
layer_conv_2d(filters=80, kernel_size=c(4,4), activation='relu') %>%
layer_conv_2d(filters=80, kernel_size=c(4,4), activation='relu') %>%
layer_max_pooling_2d(pool_size=c(4,4)) %>% layer_dropout(rate=0.35) %>%
layer_flatten() %>% layer_dense(units=256, activation='relu') %>%
layer_dropout(rate=0.25) %>% layer_dense(units=4, activation="softmax") %>%

compile(loss='categorical_crossentropy', optimizer=optimizer_adam(), metrics=c("accuracy"))
```

```
history<- model.card %>% fit(train, train.lab, epochs=50, batch_size=40, validation_split=0.2)
```




```
#computing prediction accuracy for testing set
model.card %>% evaluate(test, test.lab)
pred.class<- as.array(model.card %>% predict(test) %>% k_argmax())
print(pred.class)
```

```
0 0 0 1 1 1 2 2 2 3 3 3
```

```
print(test.y)
```

```
0 0 0 1 1 1 2 2 2 3 3 3
```

```
print(paste("accuracy=", round(1-mean(test.y!=pred.class),digits=4)))
```

```
"accuracy= 1"
```

□

Example. The built-in data set "MNIST" (short for "Modified National Institute of Standards and Technology") contains a collection of 70,000, 28×28 images of handwritten digits from 0 to 9. These data are already split into a training set (60,000 images) and a testing set (10,000 images). The R and Python codes below train a CNN model and classify digits in the testing set. The prediction accuracy is computed.

In R:

```
library(keras)

mnist<- dataset_mnist()
train.x<- mnist$train$x #x=images
train.y<- mnist$train$y #y=labels
test.x<- mnist$test$x
test.y<- mnist$test$y

str(train.x)
```

```
int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(train.y)
```

```
int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...
```

```
str(test.x)
```

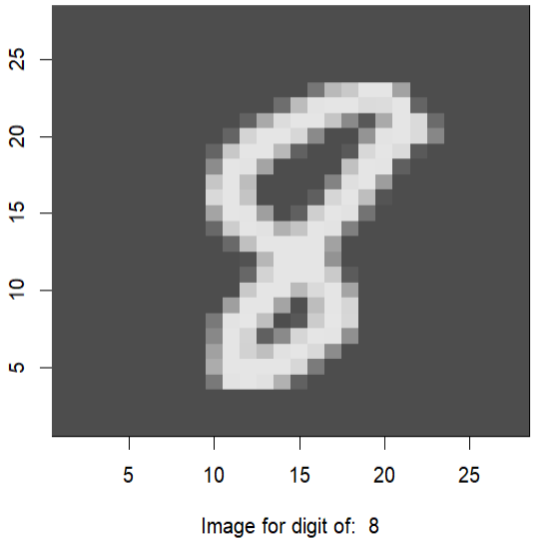
```
int [1:10000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(test.y)
```

```
int [1:10000(1d)] 7 2 1 0 4 1 4 9 5 9 ...
```

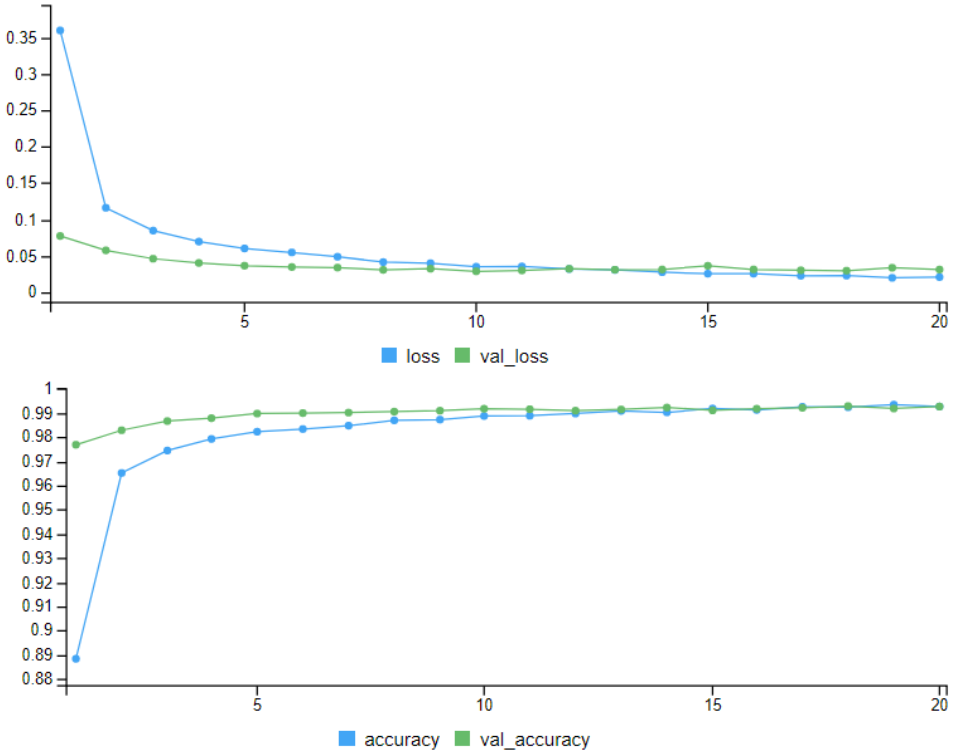
```
index.image<- 2700 #picked randomly
input.matrix<- train.x[index.image,1:28,1:28]
output.matrix<- apply(input.matrix, 2, rev)
output.matrix<- t(output.matrix)
image(1:28, 1:28, output.matrix, col=gray.colors(256), xlab=paste('Image for digit of: ', train.y[index.image]),
```

```
ylab="")
```




```
#specifying parameters
batch.size<- 128
num.classes<- 10
epochs<- 20
img.rows<- 28
img.cols<- 28

train.x<- array_reshape(train.x, c(nrow(train.x), img.rows, img.cols, 1))
test.x<- array_reshape(test.x, c(nrow(test.x), img.rows, img.cols, 1))
input.shape<- c(img.rows, img.cols, 1)

#rescaling images
train.x<- train.x/255
test.x<- test.x/255

#converting class vectors to binary class matrices
train.y<- to_categorical(train.y, num.classes)
test.lab<- to_categorical(test.y, num.classes)

#defining model architecture
cnn_model<- keras_model_sequential() %>%
layer_conv_2d(filters=32, kernel_size=c(3,3), activation='relu', input_shape=input.shape) %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
```

```
layer_conv_2d(filters=64, kernel_size=c(3,3), activation='relu') %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_dropout(rate=0.25) %>% layer_flatten() %>%
layer_dense(units=128, activation='relu') %>% layer_dropout(rate=0.5) %>%
layer_dense(units=num.classes, activation='softmax')

#compiling model
cnn_model %>% compile(loss=loss_categorical_crossentropy, optimizer=optimizer_adam(), metrics=c('accuracy'))

#training model
cnn_model %>% fit(train.x, train.y, batch_size=batch.size, epochs=epochs, validation_split=0.2)
```




```
#computing prediction accuracy
cnn_model %>% evaluate(test.x, test.lab)
pred.class<- as.array(cnn_model %>% predict(test.x) %>% k_argmax())
head(pred.class, n=50)
```

```
 [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0
```

```
[30] 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3 5 1 2 4 4
```

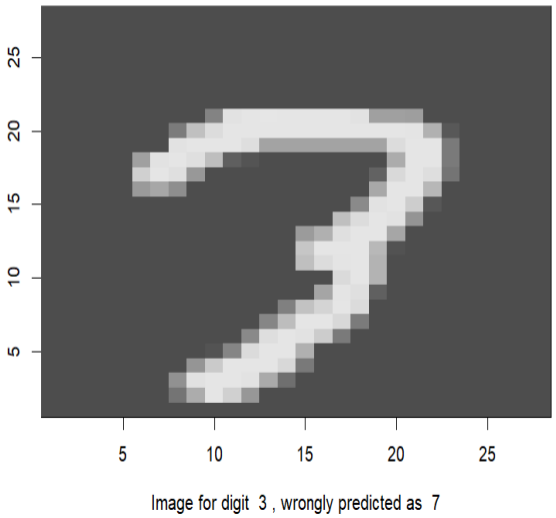
```
head(test.y, n=50)
```

```
 [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0
[30] 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3 5 1 2 4 4
```

```
print(paste("accuracy=", round(1-mean(test.y!=pred.class), digits=4)))
```

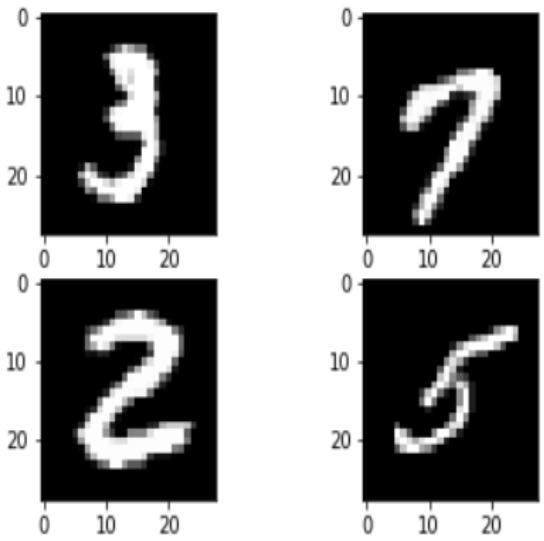
```
"accuracy= 0.9924"
```

```
#displaying misclassified images
missed.image<- mnist$test$x[pred.class != mnist$test$y,]
missed.digit<- mnist$test$y[pred.class != mnist$test$y]
missed.pred<- pred.class[pred.class != mnist$test$y]

index.image<- 270
input.matrix<- missed.image[index.image,1:28,1:28]
output.matrix<- apply(input.matrix, 2, rev)
output.matrix <- t(output.matrix)
image(1:28, 1:28, output.matrix, col=gray.colors(256), xlab=paste('Image for digit ', missed.digit[index.image],',
wrongly predicted as ', missed.pred[index.image]), ylab="")
```



In Python:

```
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
#loading the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#plotting some four images on gray scale
plt.subplot(221)
plt.imshow(X_train[10], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[15], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[25], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[35], cmap=plt.get_cmap('gray'))
plt.show()
```



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
```

```
#loading training and testing sets (X=images, y=labels)
(train_X, train_y),(test_X, test_y)=mnist.load_data()
```

```
#flattening 28x28 images to a 784 vector for each image
num_pixels=train_X.shape[1]*train_X.shape[2]
train_X=train_X.reshape((train_X.shape[0],num_pixels)).astype('float32')
test_X=test_X.reshape((test_X.shape[0],num_pixels)).astype('float32')

#preparing labels
train_y=to_categorical(train_y)
test_y=to_categorical(test_y)
num_classes=test_y.shape[1]
```

```
#building CNN model
def baseline_model():
    model = Sequential()
    model.add(Dense(num_pixels, input_shape=(num_pixels,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
    model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
    return model

model = baseline_model()
model.fit(train_X, train_y, validation_data=(test_X, test_y), epochs=10, batch_size=200)

#computing prediction accuracy
accuracy=model.evaluate(test_X,test_y)
print("Prediction Accuracy:", round(accuracy[1]*100,2),"%")
```

```
Prediction Accuracy: 97.25 %
```

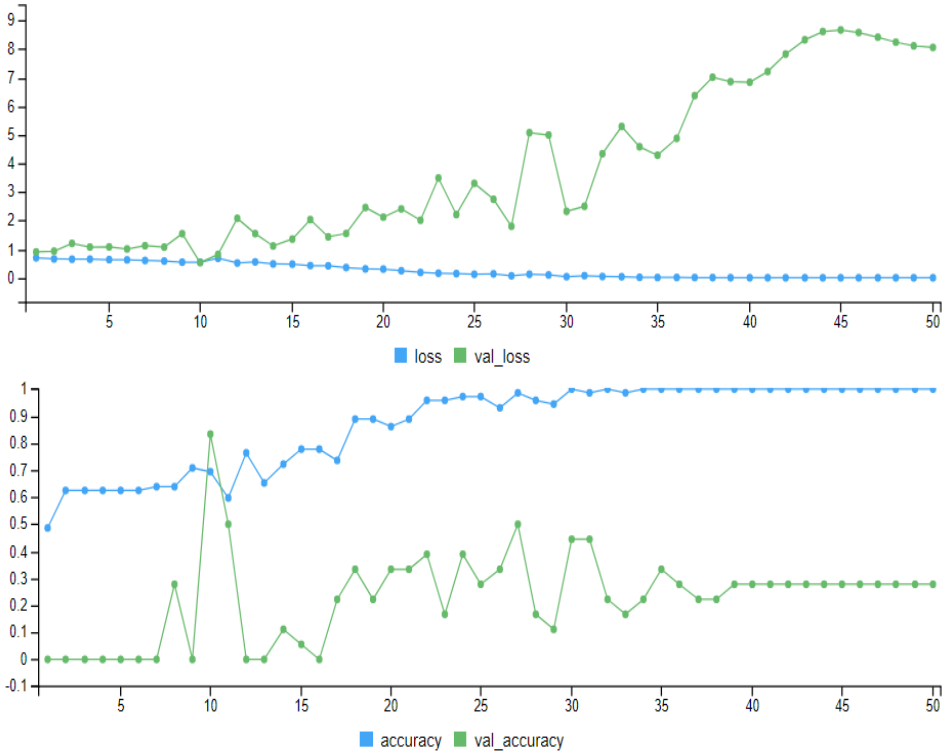
□

Example. The folder "PetsImages" contains the training set of 45 images of cats and 45 images of dogs, and a testing set of 34 images of our own pets (12 images of cats and 22 images of dogs). The R and Python codes below train a CNN model and classify the images in the testing set.

In R:

```
library(keras)
library(EBImage)

train.files<- Sys.glob(file.path("./PetsImages/train/*.jpg"))
train.labels<- substring(basename(train.files), 1,3) #extracting label: 'cat' or 'dog'
train.lab<- as.numeric(ifelse(train.labels=='cat',1,0))
```

```
setwd("./PetsImages/train")
img.pets<- sample(dir());
train.pets<- list(NULL);
for(i in 1:length(img.pets)) {
    train.pets[[i]]<- readImage(img.pets[i])
    train.pets[[i]]<- resize(train.pets[[i]], 100, 100)
}

train<- aperm(combine(train.pets), c(4,1,2,3)) #permuting dimensions

#building model
cnn.model<- keras_model_sequential() %>%
layer_conv_2d(filters=32, kernel_size=c(3, 3), activation="relu",
input_shape=c(100, 100, 3)) %>% layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=64, kernel_size=c(3, 3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=128, kernel_size=c(3, 3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=128, kernel_size=c(3, 3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>% layer_flatten() %>%
layer_dense(units=512, activation="relu") %>% layer_dense(units=1, activation="sigmoid")

cnn.model %>% compile(loss="binary_crossentropy", optimizer=optimizer_adam(), metrics="accuracy")

history<- cnn.model %>% fit(train, train.lab, epochs=50, batch_size=40, validation_split=0.2)
```




```
#computing prediction accuracy for testing set
setwd("./PetsImages/ourpets")
img.pets<- sample(dir());
test.pets<- list(NULL);
for(i in 1:length(img.pets)) {
    test.pets[[i]]<- readImage(img.pets[i])
    test.pets[[i]]<- resize(test.pets[[i]], 100, 100)
}

test<- aperm(combine(test.pets), c(4,1,2,3))
test.lab<- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1)

cnn.model %>% evaluate(test, test.lab)
pred.prob<- cnn.model %>% predict(test)

true.class<- ifelse(test.lab==1,'cat','dog')
pred.class<- c()
match<- c()
for (i in 1:length(pred.prob)) {
    pred.class[i]<- ifelse(pred.prob[i]>0.5,'cat','dog')
```

```
    match[i]<- ifelse(true.class[i]==pred.class[i],1,0)
}

print(true.class)
```

```
[1] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog"
[15] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "cat" "cat" "cat" "cat" "cat" "cat"
[29] "cat" "cat" "cat" "cat" "cat" "cat"
```

```
print(pred.class)
```

```
 [1] "dog" "cat" "cat" "cat" "dog" "cat" "cat" "dog" "dog" "dog" "cat" "dog" "dog" "cat"
[15] "cat" "dog" "cat" "cat" "dog" "dog" "cat" "cat" "dog" "dog" "dog" "dog" "cat" "cat"
[29] "cat" "dog" "cat" "cat" "dog" "dog"
```

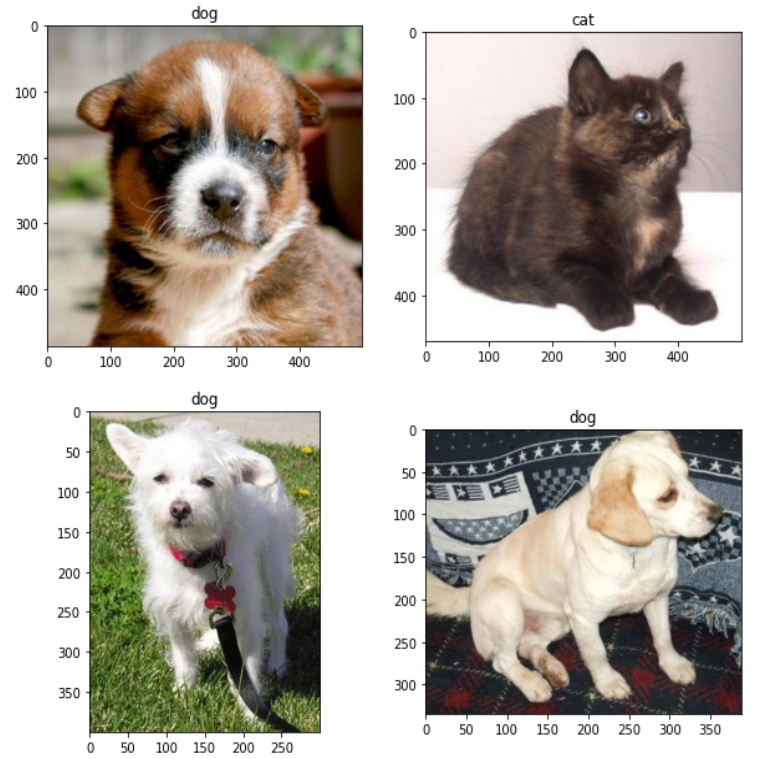
```
print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.4412"
```

In Python:

```
import numpy
import pandas
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import os
from PIL import Image
import glob
import zipfile
```

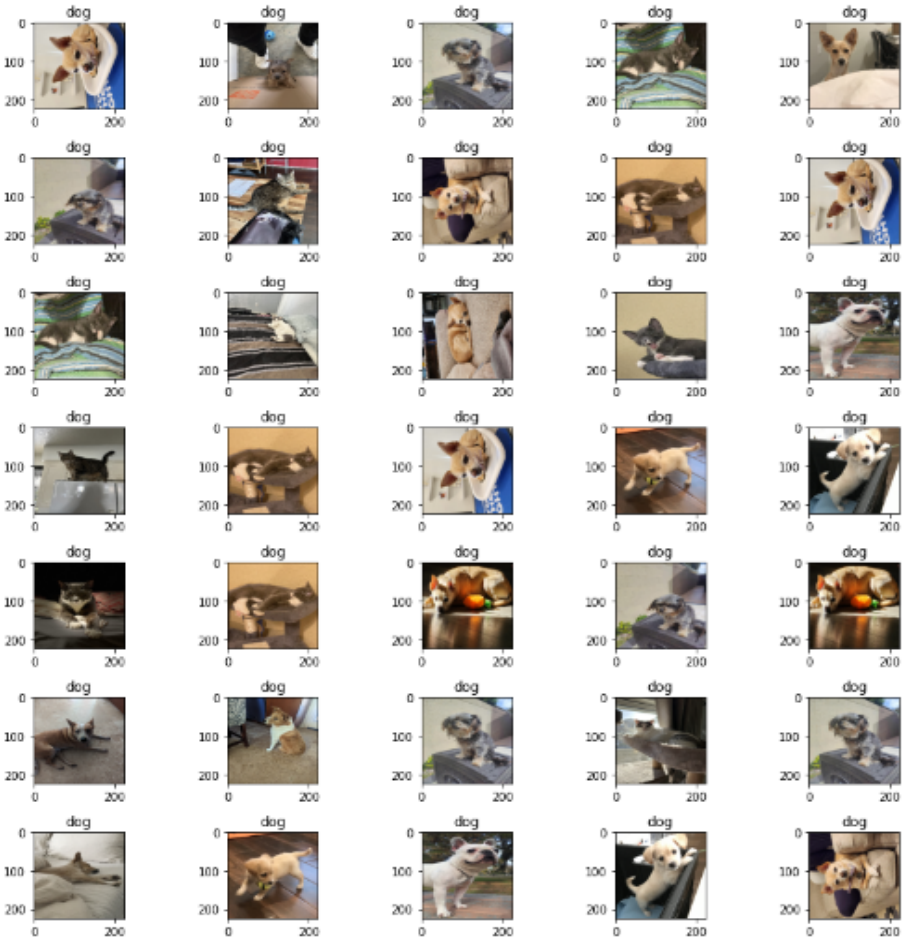
```
train_files=glob.glob('./PetsImages/train/*.jpg')
train_labels=[i.strip('./PetsImages/train/')[1:4] for i in train_files]
train_df=pandas.DataFrame({'filename': train_files, 'class': train_labels})
fig,axs=plt.subplots(2, 2, figsize=(10,10))
axs=axs.ravel()
for i in range(0,4):
    idx = random.choice(train_df.index)
    axs[i].imshow(Image.open(train_df['filename'][idx]))
    axs[i].set_title(train_df['class'][idx])
```




```
train_datagen=ImageDataGenerator(rotation_range=5,rescale=1./255, horizontal_flip=True, shear_range=0.2,
zoom_range=0.2, validation_split=0.2)

img_height,img_width=224, 224
batch_size=64

train_generator=train_datagen.flow_from_dataframe(train_df, target_size=(img_height, img_width),
batch_size=batch_size, class_mode='categorical', subset='training')

validation_generator=train_datagen.flow_from_dataframe(train_df, target_size=(img_height, img_width),
batch_size=batch_size, class_mode='categorical', subset='validation')
```

```
#training CNN model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model=Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2 ,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model.compile(loss='binary_crossentropy', metrics=['accuracy'])

history=model.fit(train_generator, epochs=50, validation_data=validation_generator)
```

```
test_files=glob.glob('./PetsImages/ourpets/*.jpg')
test_df=pandas.DataFrame({'filename': test_files})
test_gen=ImageDataGenerator(rescale=1./255)
test_generator=test_gen.flow_from_dataframe(test_df, x_col='filename', y_col=None,
class_mode=None, target_size=(img_height,img_width), batch_size=batch_size)

def visualize_predictions(test_generator, model):
    plt.figure(figsize=(12,12))
    for i in range(0, 35):
        plt.subplot(7, 5, i+1)
        for X_batch in test_generator:
            prediction=model.predict(X_batch)[0]
            image=X_batch[0]
            plt.imshow(image)
            plt.title('cat' if numpy.argmax(prediction)==0 else 'dog')
            break
    plt.tight_layout()
    plt.show()

visualize_predictions(test_generator, model)
```



Note that every image is classified as a dog.
□