Advanced Computer Science Course Lecture 1

Tishreen University

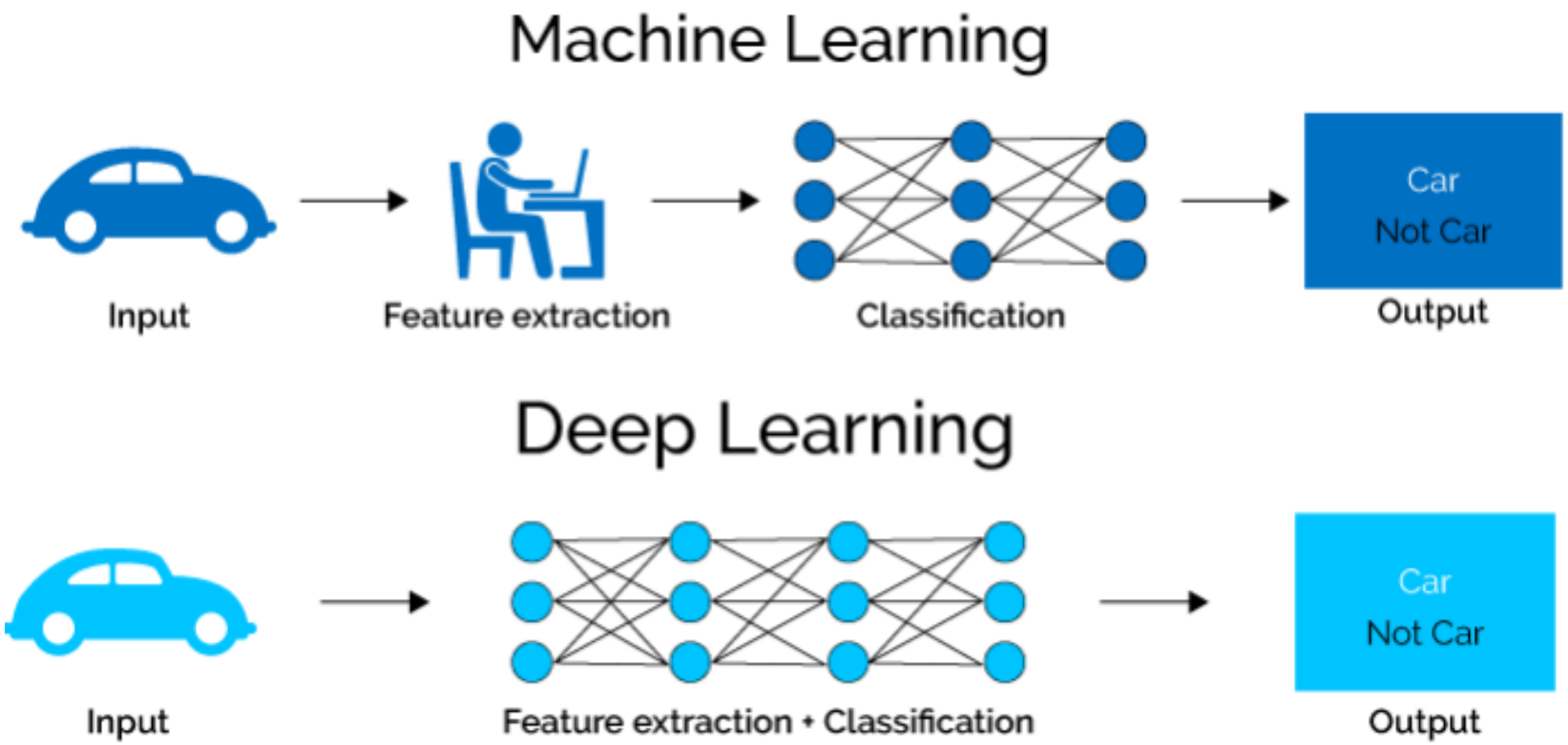
Computer and automatic control engineering dept.

Master Program- 2024

1st year

Dr. Ali Mahmoud Mayya

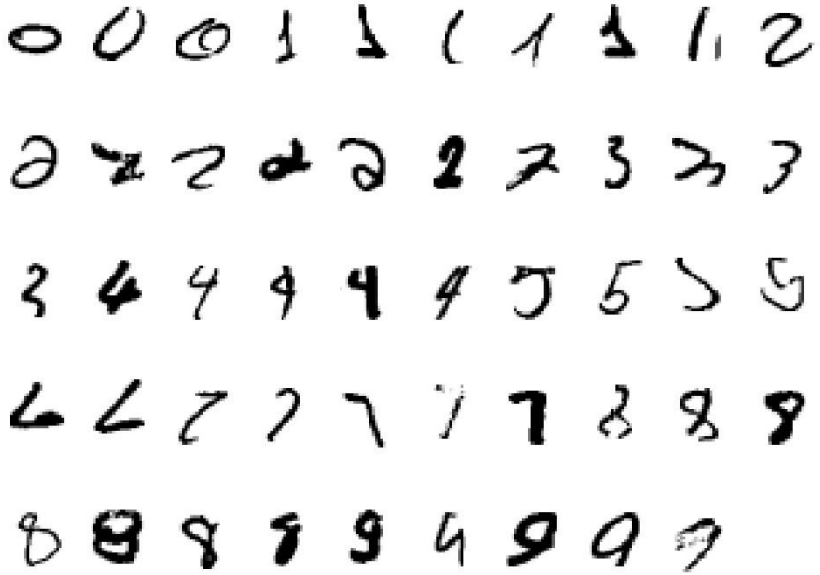
Deep Learning Vs. Machine Learning

- 

Machine Learning

- **Supervised learning** (training with labeled data), **un-supervised learning** (clustering un-labeled data), and **semi-supervised learning** (use both labeled and unlabeled data)
- Supervised learning: **classification** and **regression**
- **Classification**: output is discrete value
- **Regression**: output is real value

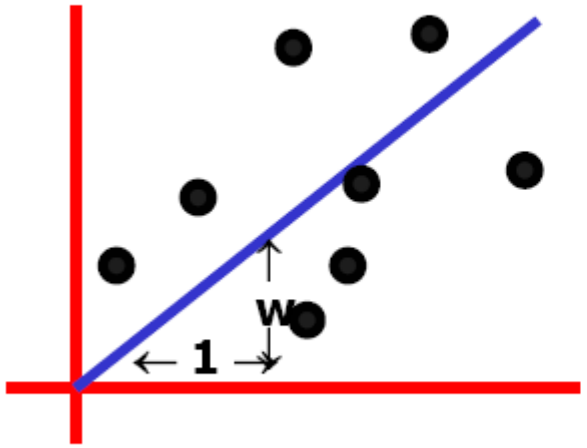
Learning Example: Recognize Handwriting



Classification: recognize each number
Clustering: cluster the same numbers together
Regression: predict the index of Dow-Jones

Machine Learning- Linear regression

Statistics Root: Linear Regression Example



DATASET

inputs	outputs
$x_1 = 1$	$y_1 = 1$
$x_2 = 3$	$y_2 = 2.2$
$x_3 = 2$	$y_3 = 2$
$x_4 = 1.5$	$y_4 = 1.9$
$x_5 = 4$	$y_5 = 3.1$

Fish length vs. weight?

X: input or predictor
Y: output or response
Goal: learn a linear function E[y|x] = wx + b.

Machine Learning- Linear regression

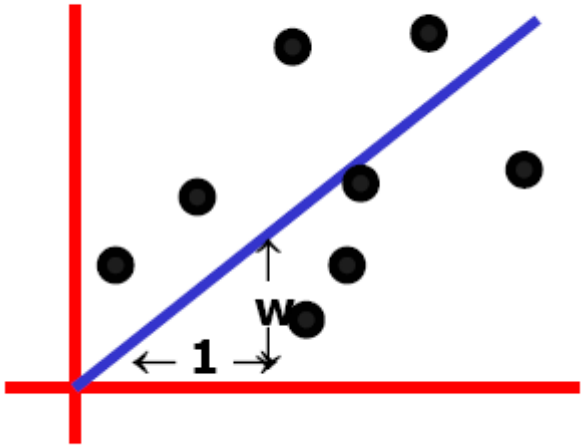
- **Definition of a linear model:**

  $y = wx + b + noise$.

  noise ~ N (0, σ^2)
- Assume σ is a **constant**.

  $y \sim N(wx + b, \sigma^2)$
- Estimate expected value of y given x ($E[y/x] = wx + b$) .
- **Given** a set of data:

  $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- **Goal**: to find the optimal parameters w and b.



Fish length vs. weight?

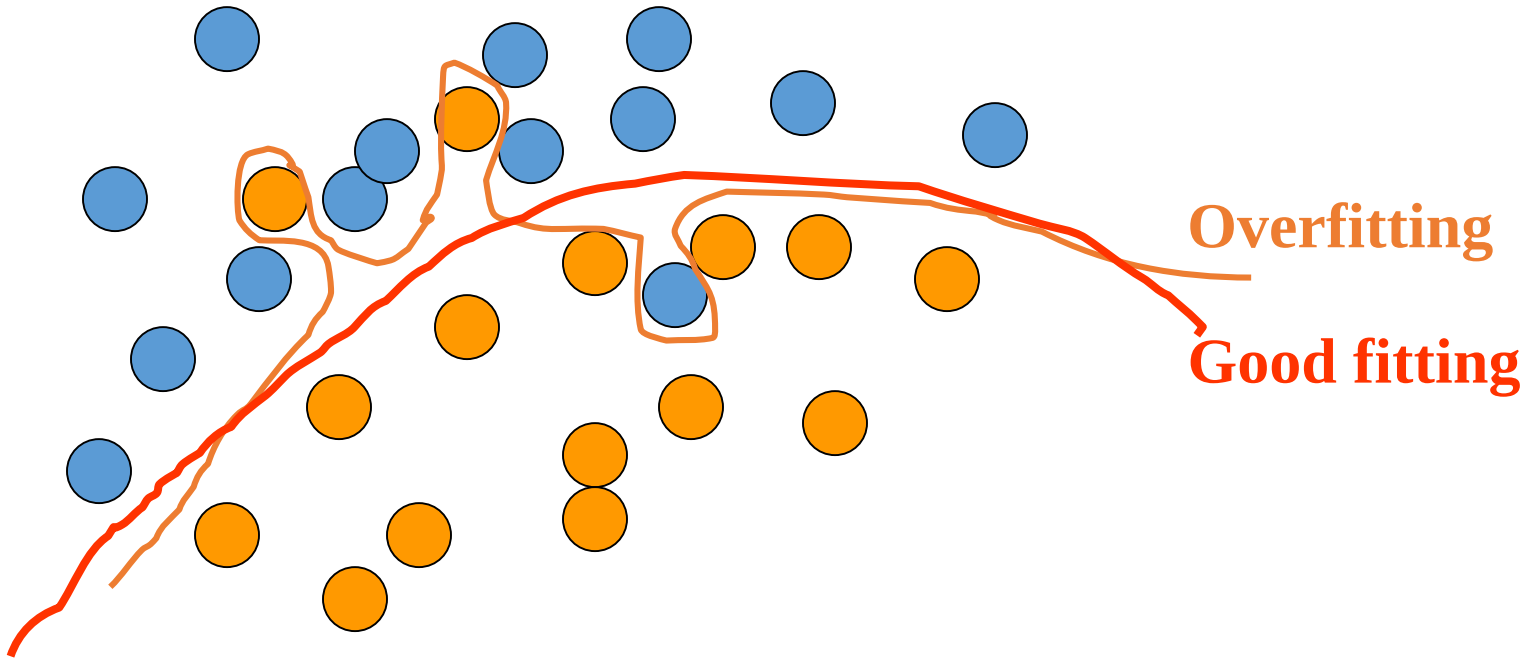
Objective Function
(Least squared error)

$$\sum_{i=1}^{N} (y_i - wx_i - b)^2$$

Machine Learning- Overfitting

The training data contains information about the regularities in the mapping from input to output. But it also contains noise
When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.

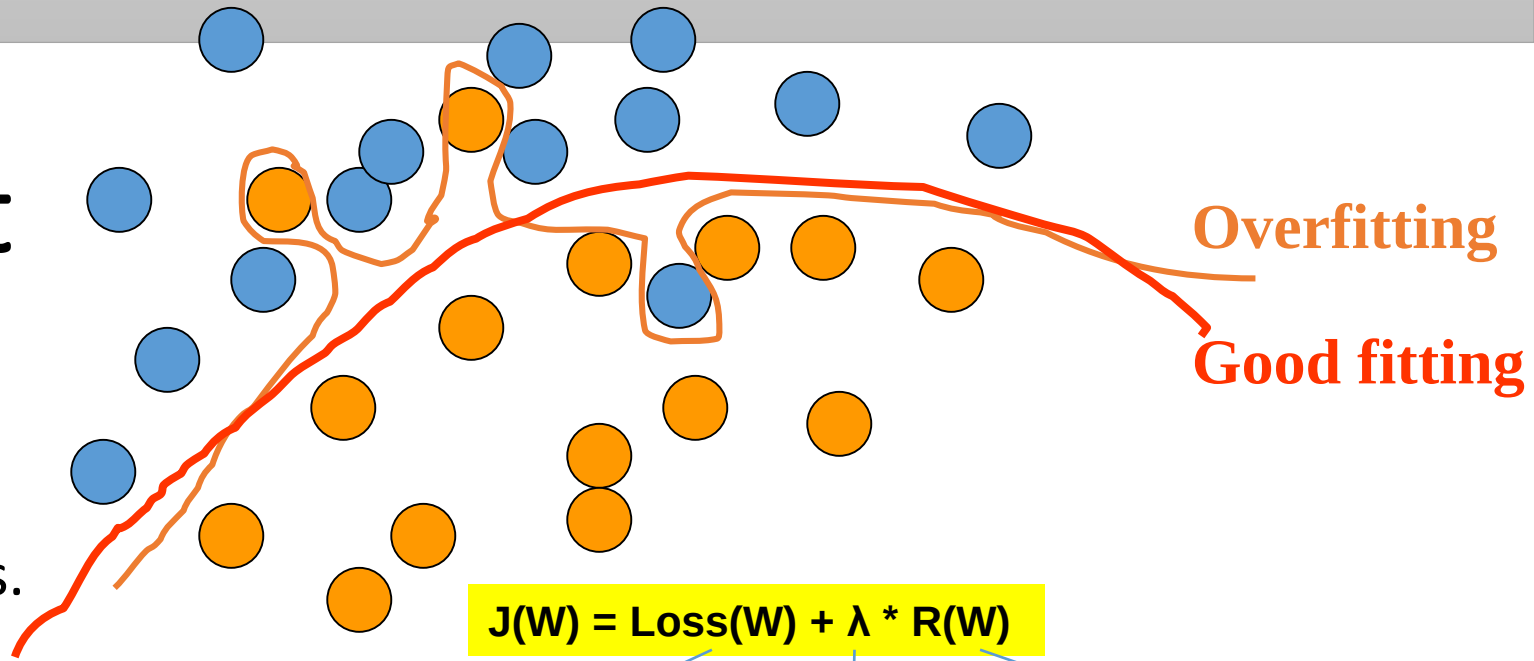
So it fits both kinds of regularity.



Machine Learning- Prevent Overfitting

Standard ways to limit the capacity of a neural net:

1. Limit the number of hidden units.
Limit the size of the weights (Weight-decay).
Adding a ***penalty*** term to the cost function during training. This penalty term encourages the model to have smaller weights.
2. Stop the learning before it has time to overfit (early stop training).



$$J(W) = Loss(W) + \lambda * R(W)$$

- loss function
- regularization parameter
- regularization term

$$R(W) = ||W||^2 = W_1^2 + W_2^2 + \ldots + W_n^2$$

The regularization parameter λ determines the relative importance of the weight decay penalty compared to the loss function. Higher values of λ increase the penalty for large weights, leading to more weight shrinkage and stronger regularization.

Machine Learning- Prevent Overfitting **(training loop)**

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(input_size,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
# Define loss function
loss_fn = tf.keras.losses.CategoricalCrossentropy()
# Define optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
# Define the regularization parameter (lambda)
l2_lambda = 0.01
# Define the training loop
for epoch in range(num_epochs):
    for x_batch, y_batch in train_dataset:
        with tf.GradientTape() as tape:
            # Forward pass
            logits = model(x_batch)
            # Calculate the loss
            loss_value = loss_fn(y_batch, logits)
# Calculate the weight decay penalty
            l2_regularization = tf.reduce_sum([
                tf.nn.l2_loss(var) for var in model.trainable_variables])
            weight_decay_penalty = l2_lambda * l2_regularization
# Add the weight decay penalty to the loss
            total_loss = loss_value + weight_decay_penalty
        # Compute the gradients
        grads = tape.gradient(total_loss,
model.trainable_variables)
        # Update the model's weights
        optimizer.apply_gradients(zip(grads,
model.trainable_variables))
```

$$J(W) = Loss(W) + \lambda * R(W)$$

- **loss function**
- **regularization parameter**
- **regularization term**

$$R(W) = ||W||^2 = W_1^2 + W_2^2 + \ldots + W_n^2$$

Machine Learning- Prevent Overfitting **(Fit function)**

```
import tensorflow as tf
# Define model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(input_size,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(num_classes,
activation='softmax')
])
# Define loss function
loss_fn =
tf.keras.losses.CategoricalCrossentropy()
# Define optimizer with weight decay
optimizer = tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    decay=1e-5,  # Weight decay parameter
)
```

```
# Compile the model
model.compile(optimizer=optimizer, loss=loss_fn,
metrics=['accuracy'])

# Define early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
)

# Train the model using model.fit
history = model.fit(
    x=train_data,
    y=train_labels,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_data=(val_data, val_labels),
    callbacks=[early_stopping],
)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x=test_data, y=test_labels)
```

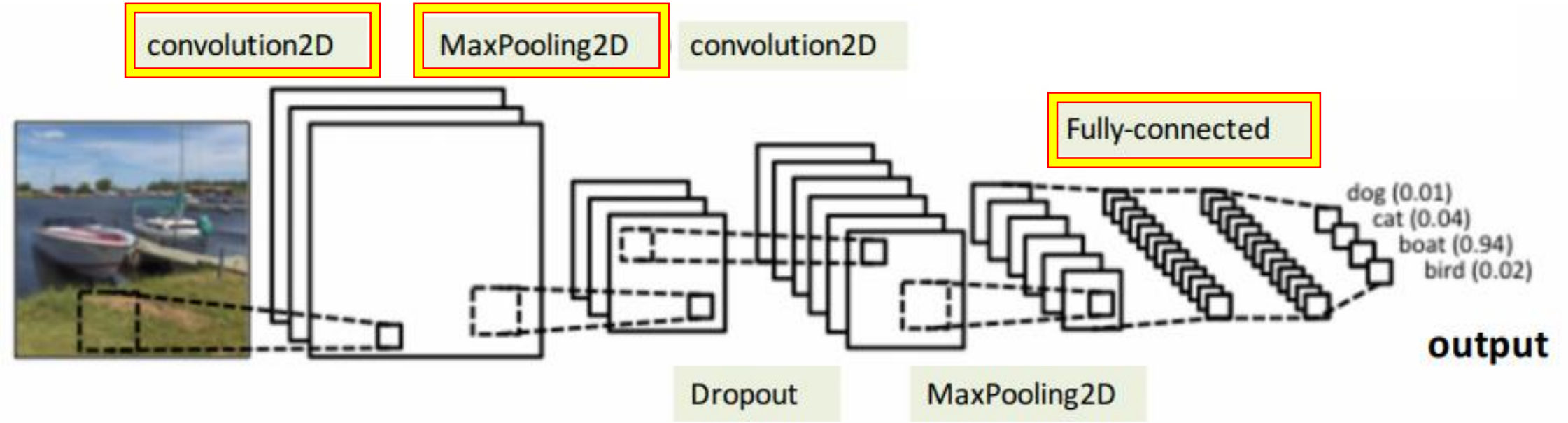
Machine Learning- Prevent Overfitting- Use validation set

Divide the total dataset into three subsets:

Training data is used for learning the parameters of the model.
Validation data is not used of learning but is used for deciding what type of model and what amount of regularization works best.
Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.

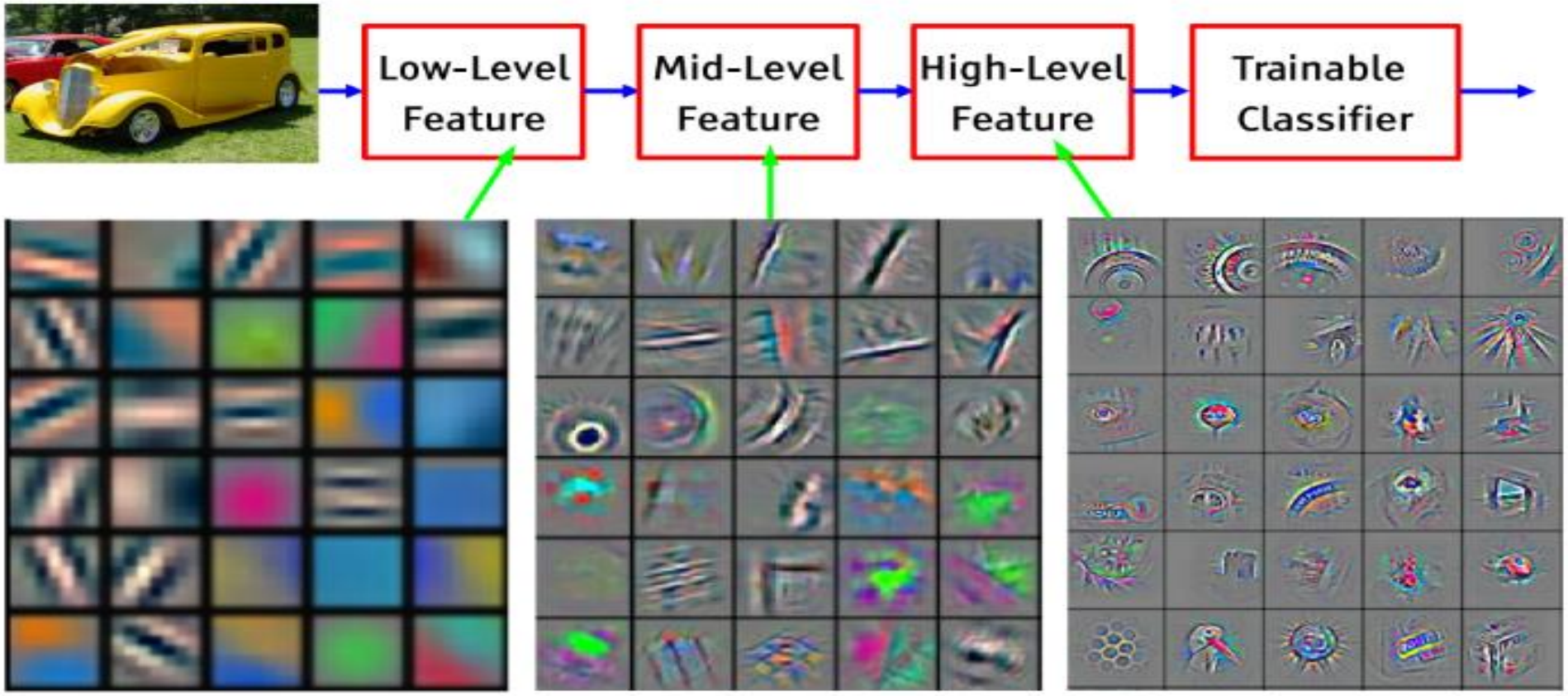
We could then re-divide the total dataset to get another unbiased estimate of the true error rate.

Deep Learning- CNN Convolutional Neural Networks

- 

Deep Learning- CNN Convolutional Neural Networks

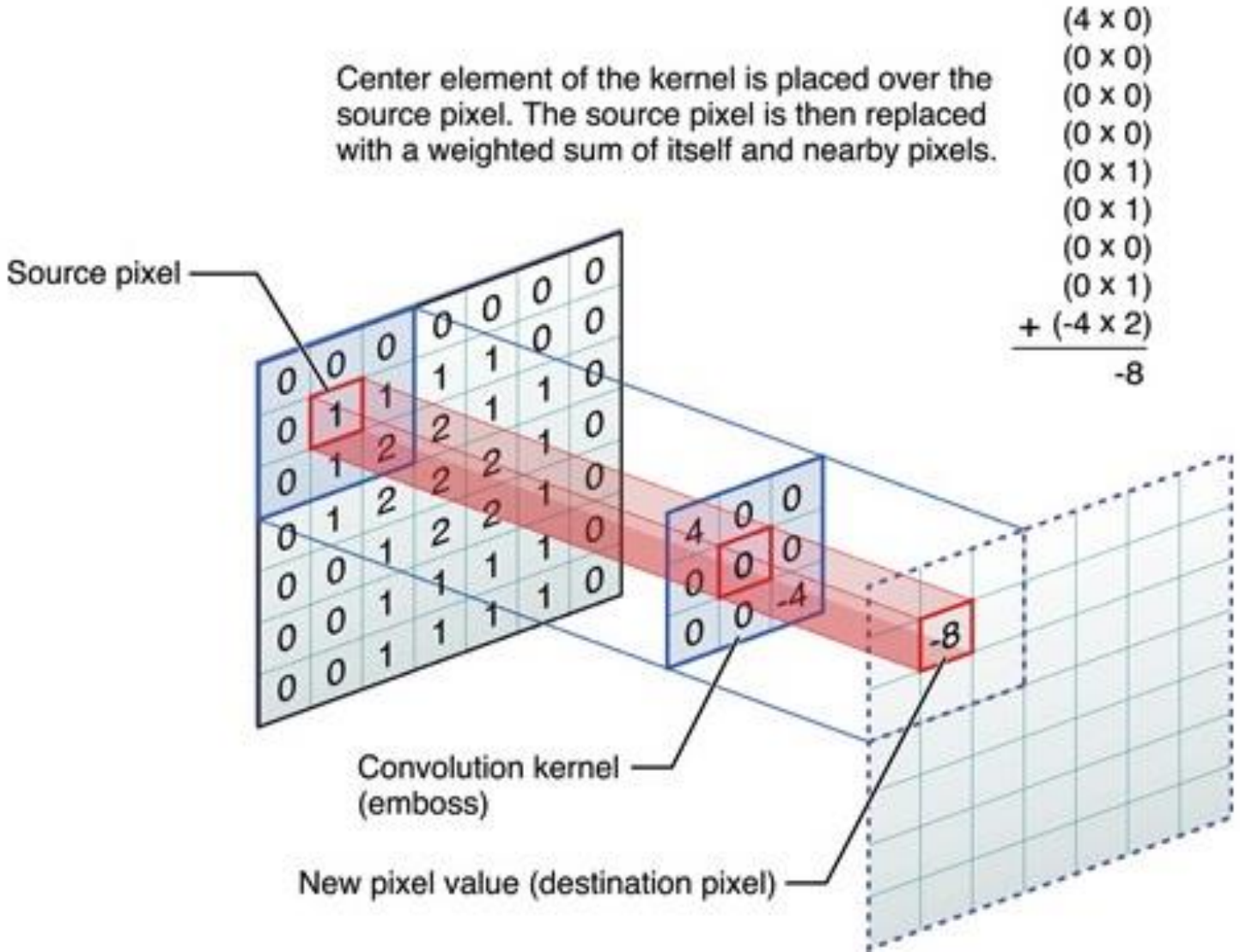
Hierarchical representation

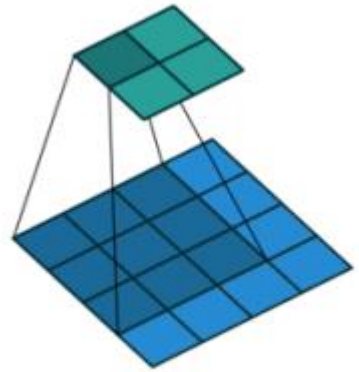


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

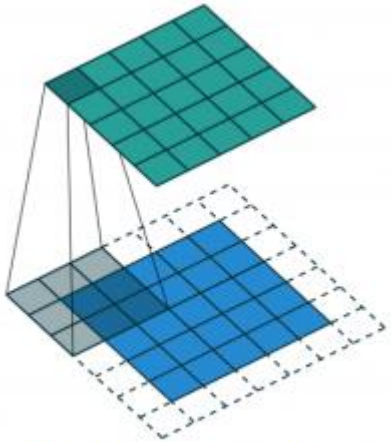
Deep Learning- CNN Convolutional Neural Networks

Convolution

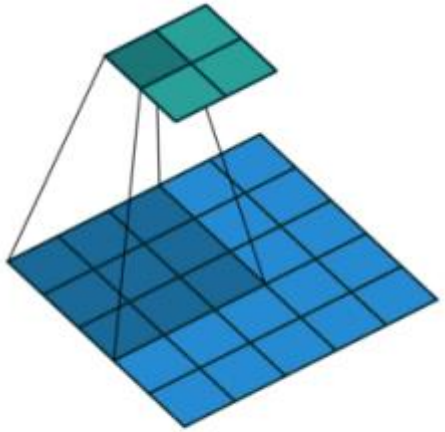




No padding, stride 1



Padding 1, stride 1



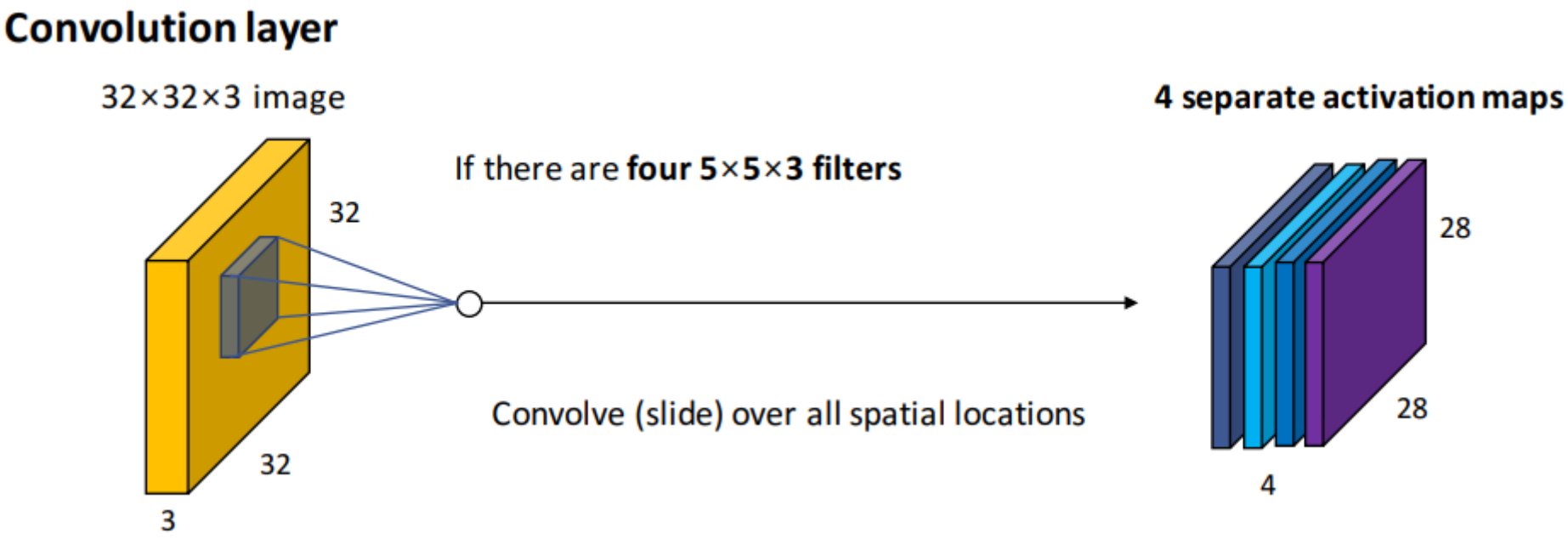
No padding, stride 2

Deep Learning- CNN Convolutional Neural Networks

Convolution

-

Output volume size= N*((W−K+2P)/S+1)



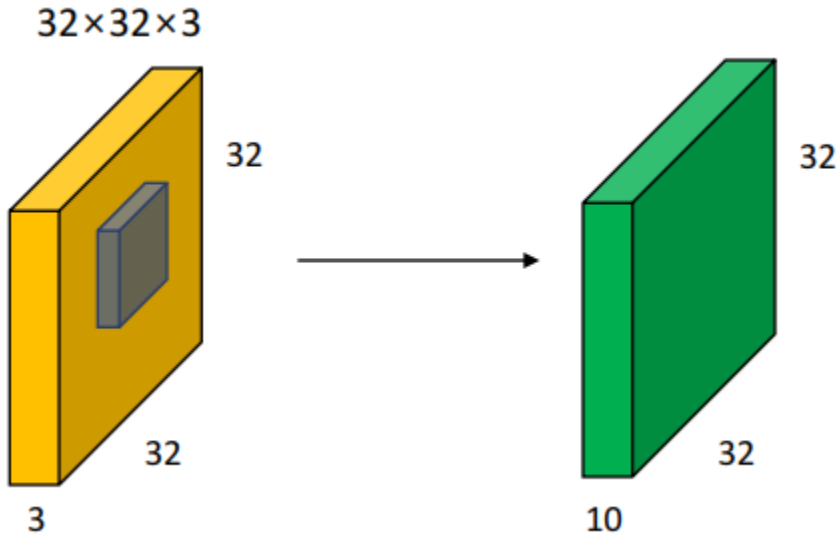
Deep Learning- CNN Convolutional Neural Networks

Convolution

- **Output volume size= N*((W−K+2P)/S+1)**

Input size: 32*32, Kernel size: 5*5, padding=2, stride = 1

Output volume size=(32 + 2×2 - 5)/1 + 1 = 32 spatially
Each filter has 5*5*3 + 1 (for bias) = 76 parameters
For all 10 filters → 760 parameters

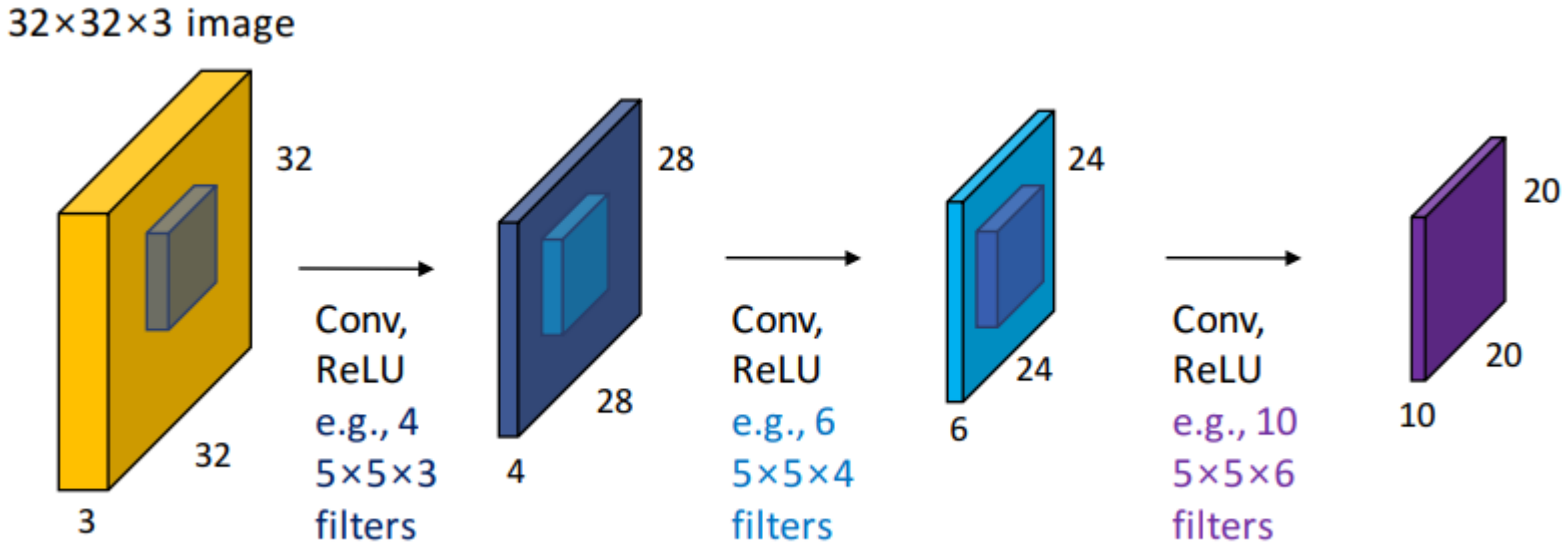


Deep Learning- CNN Convolutional Neural Networks

Convolution

-

Output volume size= N*((W−K+2P)/S+1)

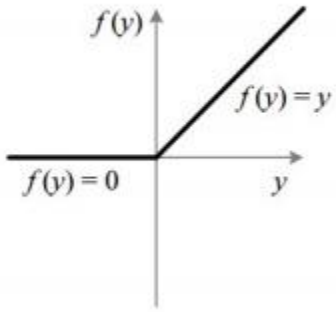


(32-4,32-4,4)

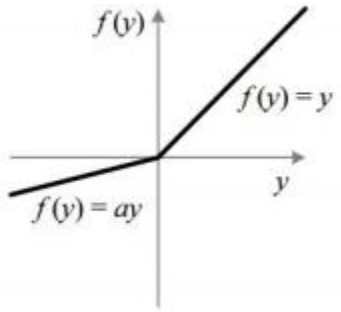
(28-4,28-4,6)

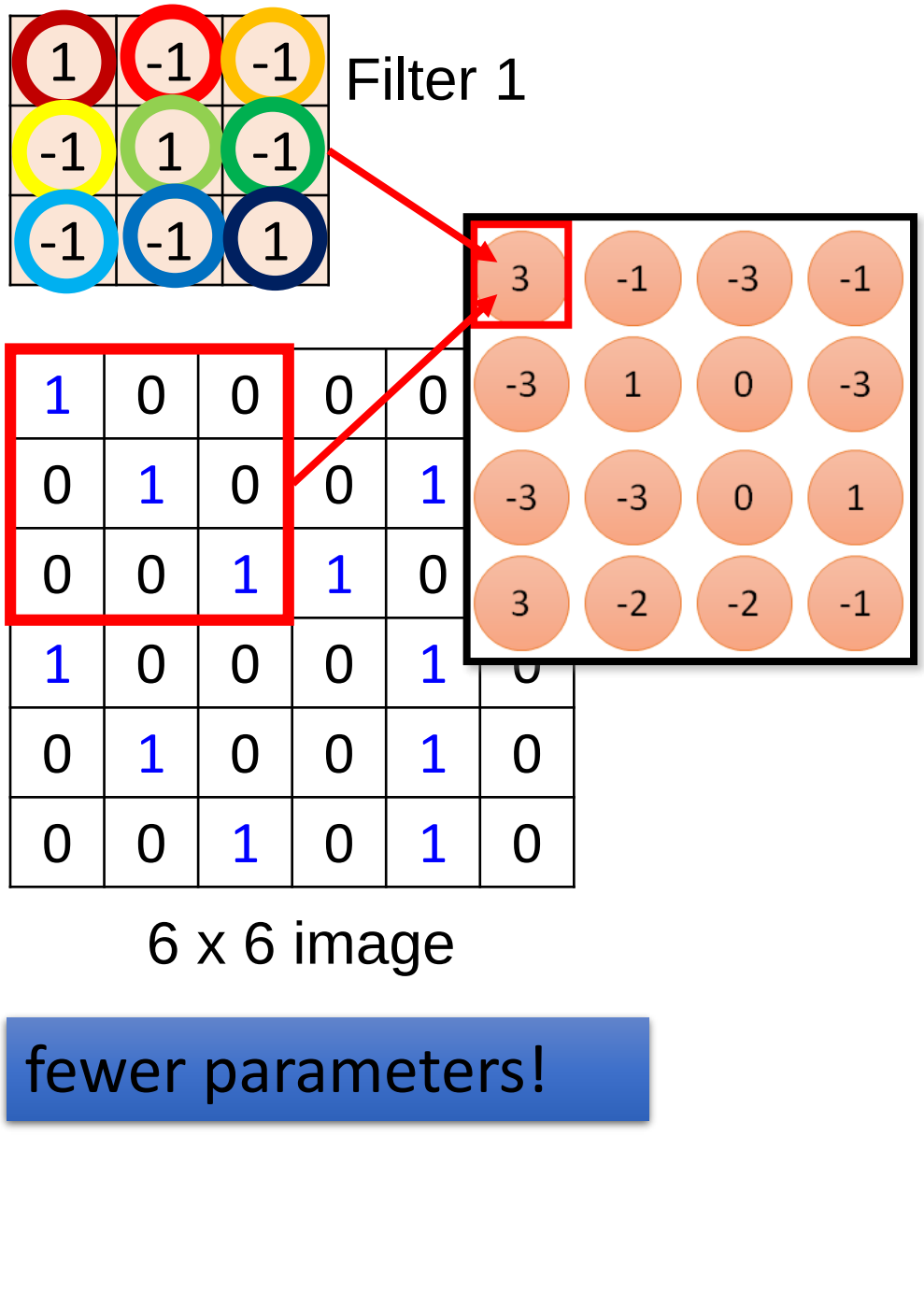
(24-4,24-4,10)

ReLU

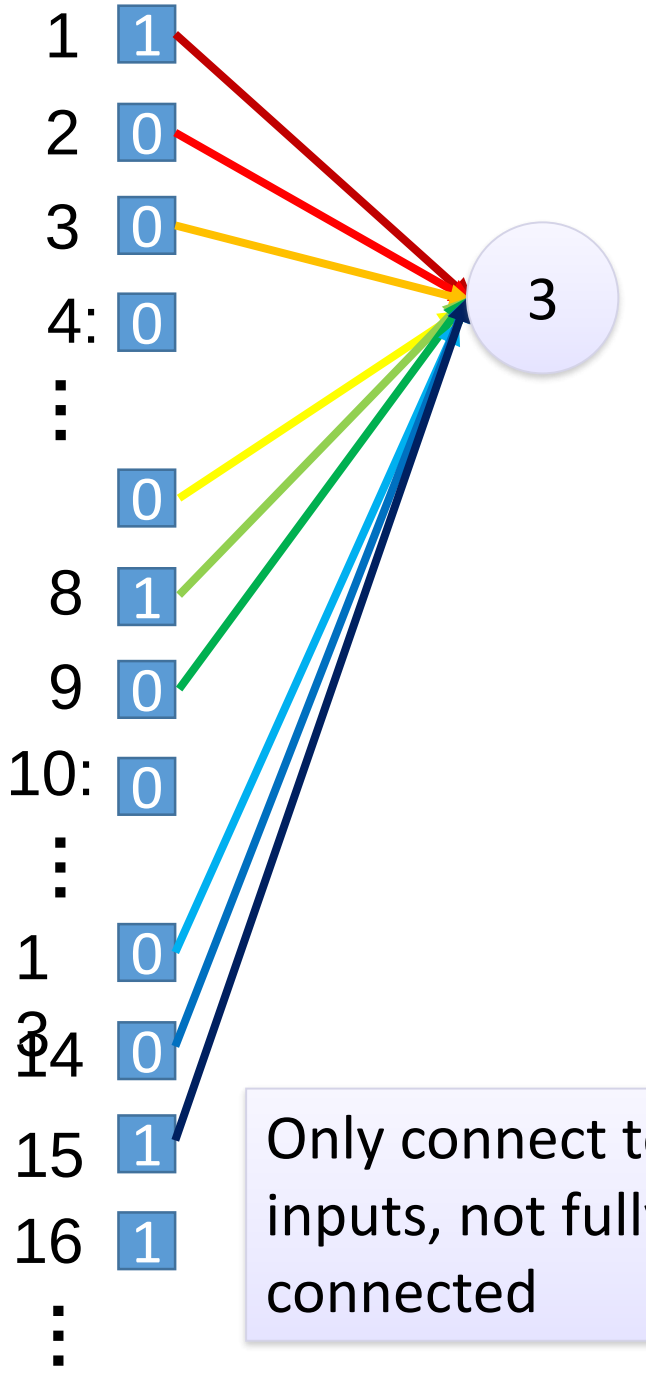


LeakyReLU



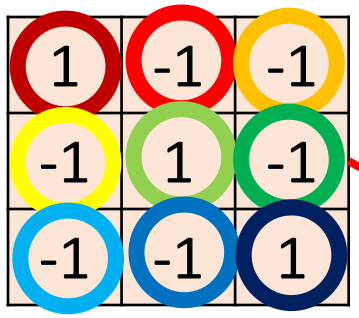


fewer parameters!

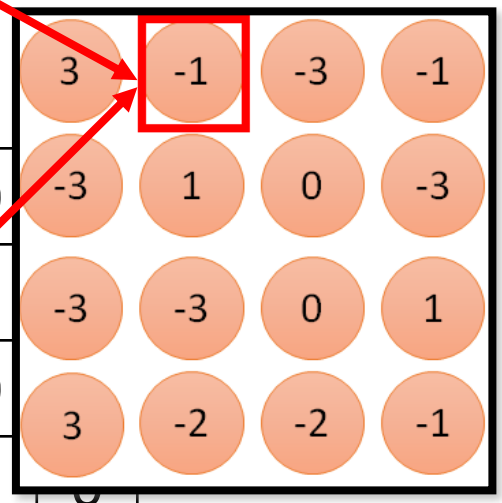


Convolution

Only connect to 9 inputs, not fully connected



Filter 1

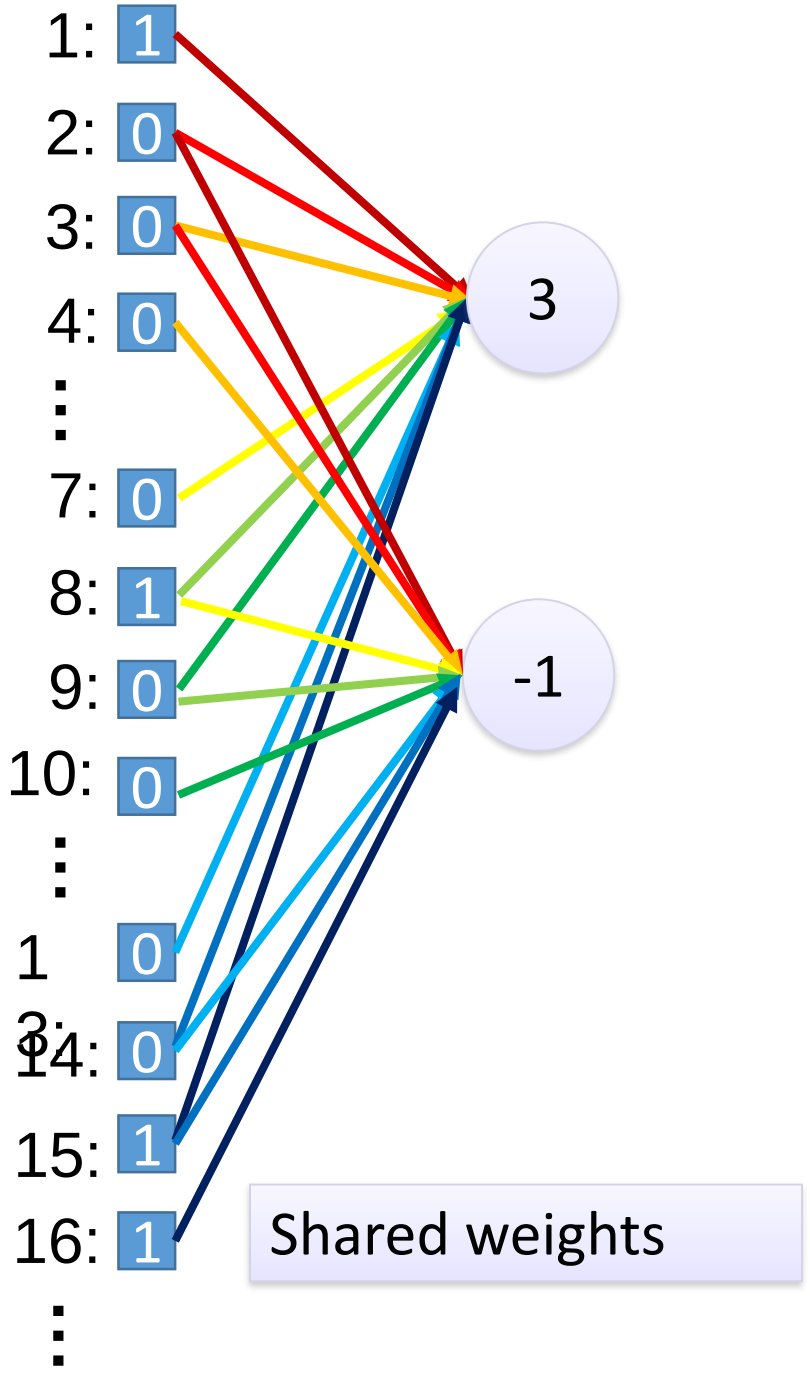


1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Fewer parameters

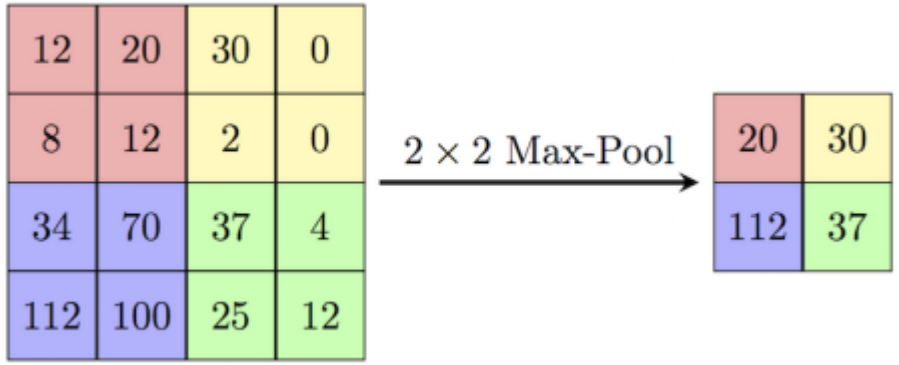
Even fewer parameters

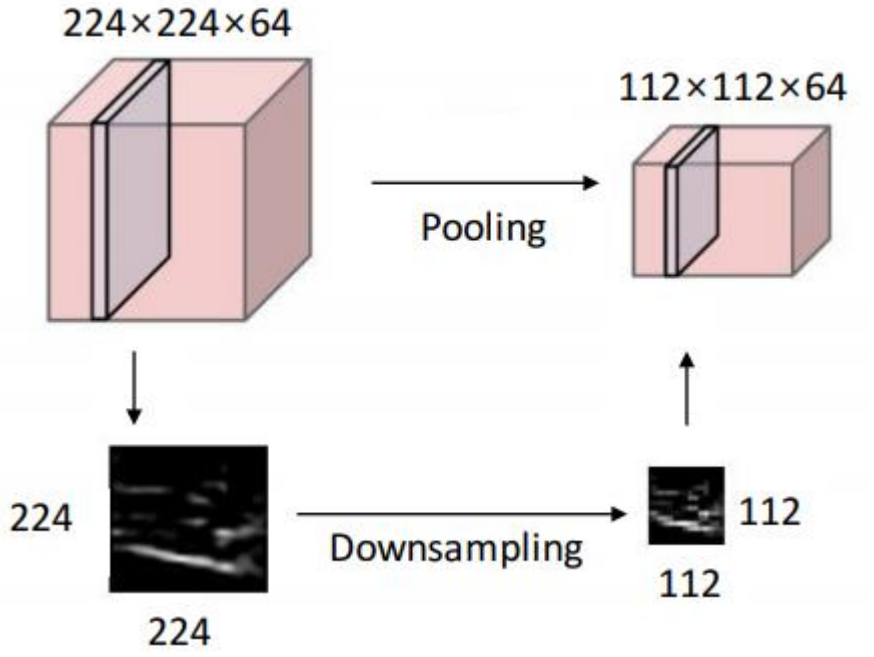


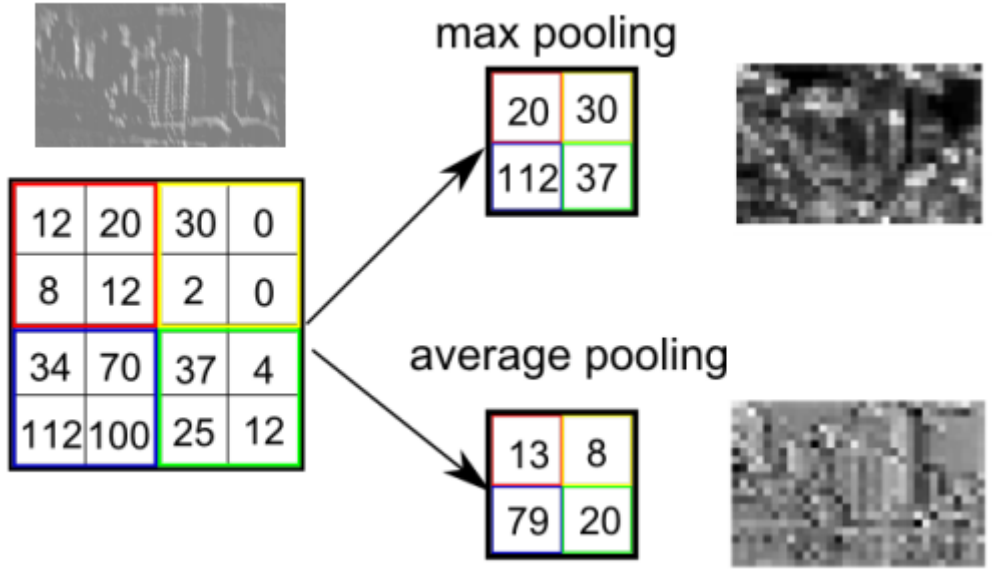
Shared weights

Deep Learning- CNN Convolutional Neural Networks

Pooling

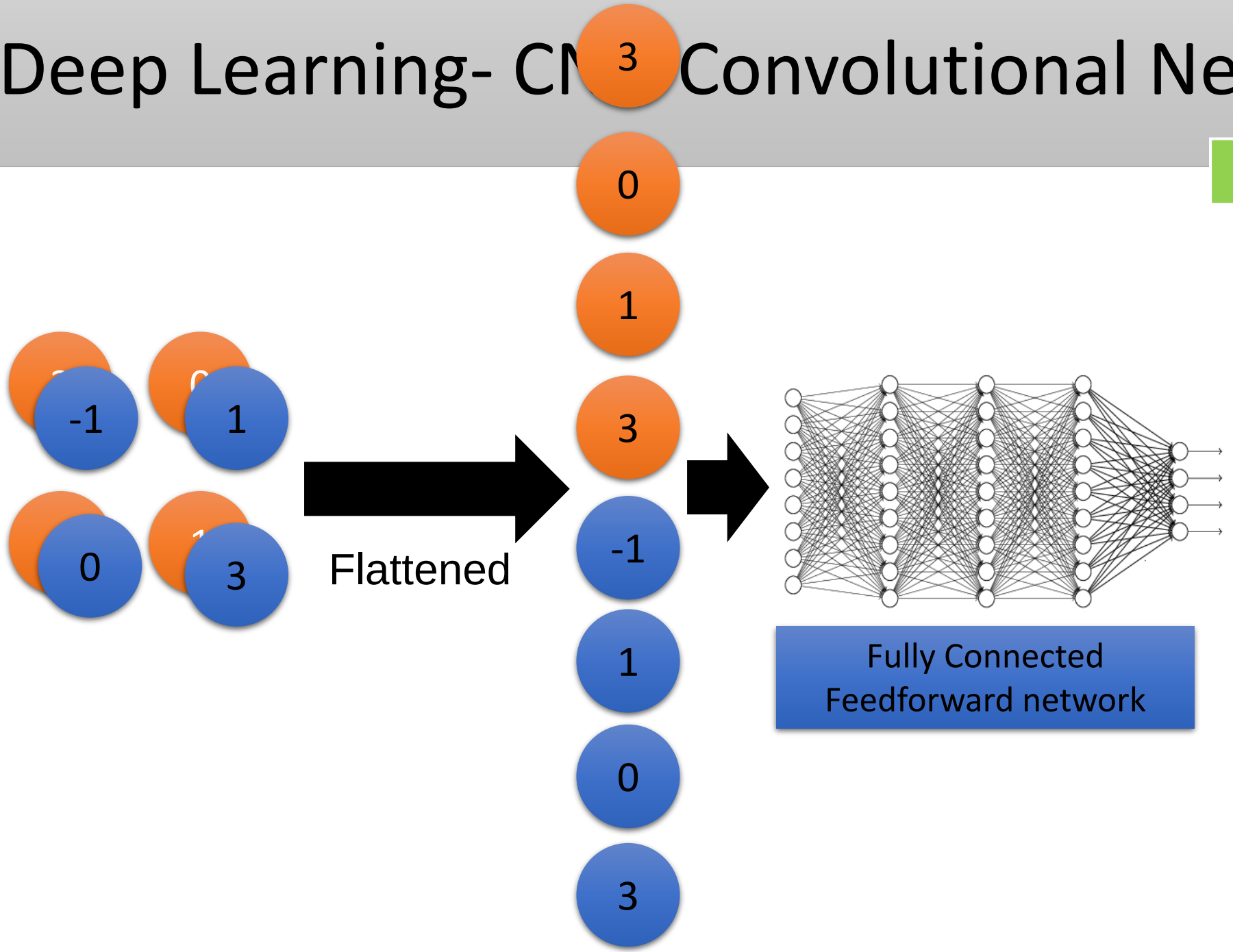
- 





Deep Learning- CNN Convolutional Neural Networks

Flattening and FC



The whole CNN- Cat dog classification example

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flattened

Fully Connected Feedforward network

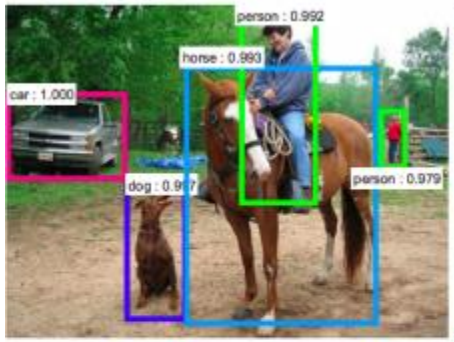
cat dog ……

Deep Learning- CNN applications

-

Self-driving cars

Detection [Ren et al., 2015]



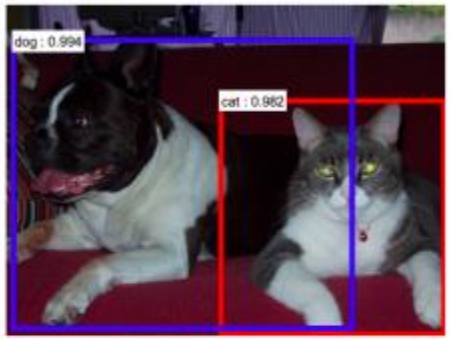


Image Captioning

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

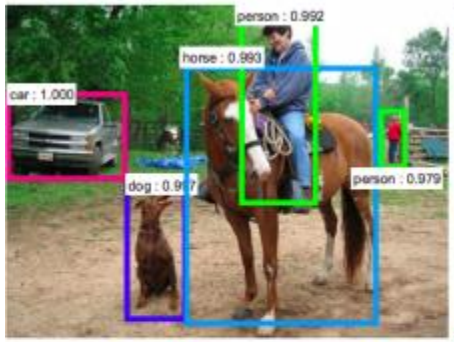
Image Recognition

Deep Learning- CNN applications

-

Self-driving cars

Detection [Ren et al., 2015]



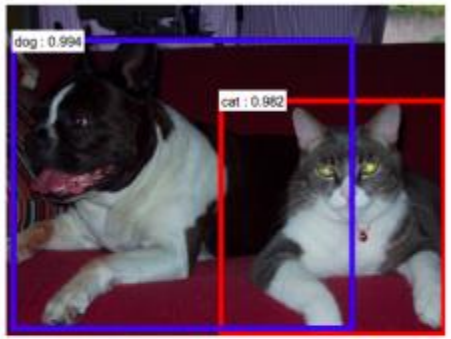


Image Captioning

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

Image Recognition

CNN-Keras | Python Implementation

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# read train
train = pd.read_csv("../input/train.csv")
print(train.shape)
train.head()
# put labels into y_train variable
Y_train = train["label"]
# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)
```

Build & Train CNN

Simple Character Recognition Example

استيراد المكاتب اللازمة
Numpy للتعامل مع المصفوفات
Matplotlib للرسم والعرض
Pandas للتعامل مع ملفات مجموعات البيانات النصية مثل CSV data
Train_test_split لتقسيم مجموعة البيانات

قراءة مجموعة بيانات التدريب من الملف train.csv

تحديد الخرج Y_train

تحديد الدخل X_train

خطوة 1: تحميل مجموعة بيانات الصور

ثم تقسيم مجموعة البيانات عند الحاجة

CNN Python Implementation

Build & Train CNN

Simple Character Recognition Example

خطوة 1:
تحميل مجموعة بيانات الصور

ثم تقسيم مجموعة البيانات عند الحاجة

```
X_train = X_train / 255.0
print("x_train shape: ",X_train.shape)
```

تطبيع البيانات Normalize Data بتقسيم قيم البيانات على 255

```
x_train shape:  (42000, 784)
```

ناتج طباعة حجم مصفوفة X_train

```
X_train = X_train.values.reshape(-1,28,28,1)
print("x_train shape: ",X_train.shape)
```

تحويل مصفوفة X_train لحجم مناسب للشبكة 28*28*1

```
x_train shape:  (42000, 28, 28, 1)
```

ناتج طباعة حجم مصفوفة الدخل

```
from keras.utils.np_utils import to_categorical
Y_train = to_categorical(Y_train, num_classes = 10)
```

ترميز شعاع الخرج

2 ⇒ [0,0,1,0,0,0,0,0,0,0]

4 ⇒ [0,0,0,0,1,0,0,0,0,0]

```
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=2)
```

تقسيم مجموعة بيانات التدريب الأصلية إلى تدريب بنسبة 90% وتحقق بنسبة 10%

CNN

Python Implementation

```
print("x_train shape",X_train.shape)
print("x_val shape",X_val.shape)
print("y_train shape",Y_train.shape)
print("y_val shape",Y_val.shape)
```

طباعة حجم كل من مجموعتي بيانات التدريب والتحقق

```
x_train shape (37800, 28, 28, 1)
x_test shape (4200, 28, 28, 1)
y_train shape (37800, 10)
y_test shape (4200, 10)
```

Build & Train CNN

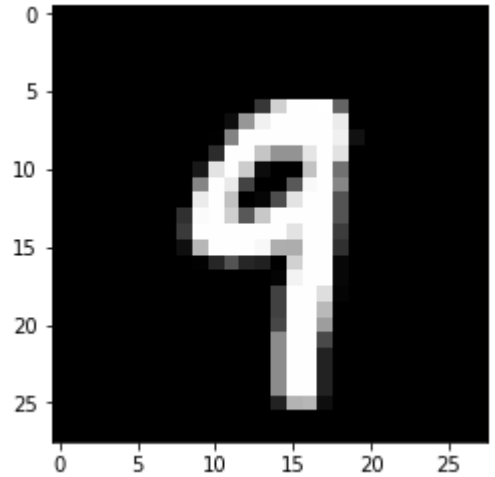
Simple Character Recognition Example

خطوة 1:
تحميل مجموعة بيانات الصور

ثم تقسيم مجموعة البيانات عند الحاجة

```
# Some examples
plt.imshow(X_train[2][:,:,0],cmap='gray')
plt.show()
```

رسم (عرض) بعض عينات مجموعة البيانات



CNN Python Implementation

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop,Adam
from keras.preprocessing.image import ImageDataGenerator
```

Build & Train CNN

Simple Character Recognition Example

خطوة 2: استيراد المكاتب اللازمة

مكتبة Keras للتعلم العميق:

Keras.sequential

لبناء طبقات شبكة التعلم العميق

Keras.optimizers

لاختيار نوع خوارزمية التحسين المستخدمة في عملية التدريب

Keras.preprocessing

.image لتطبيق عمليات المعالجة المسبقة على الصورة

Optimizer هو خوارزمية تستخدم لتقليل تابع الخطأ Loss Function لتعديل أوزان وانحيازات الشبكة

في حين تقوم خوارزمية التدريب Back Propagation باشتقاق ونشر الخطأ Gradient وحساب تعديلات الأوزان، يقوم Optimizer بتعديل الأوزان وفقاً لقاعدة محددة يستخدم فيها معدل التعلم لتعديل الأوزان مما يساعد الشبكة على التقارب بشكل أسرع

Optimizer الأكثر استخداماً لشبكات CNN هو ADAM

CNN

Python Implementation

Build & Train CNN

خطوة 3: بناء طبقات الشبكة

يتم إضافة طبقات للمودل model باستخدام الأمر

model.add

```
model = Sequential()
model.add(Conv2D(filters = 8, kernel_size = (5,5),padding = 'Same',
          activation ='relu', input_shape = (28,28,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters = 16, kernel_size = (3,3),padding = 'Same',
          activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
# fully connected
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
```

طبقة Conv2D عدد المرشحات 8 وحجم نافذة المرشح 5 ويتم استخدام حشو وتابع التفعيل Relu، حجم صورة الدخل 28*28

طبقة Maxpooling بحجم 2*2

طبقة dropout بنسبة %25

طبقة Conv2D عدد المرشحات 16 وحجم نافذة المرشح 3 ويتم استخدام حشو وتابع التفعيل Relu

طبقة Maxpooling بحجم 2*2

طبقة dropout بنسبة %25

الطبقة كاملة الاتصال:
طبقة Flatten وطبقة تخفيض وسيطية بحجم 256 عصبون وتابع Relu
طبقة Dropout بنسبة %50
طبقة تخفيض تمثل طبقة التصنيف ب 10 عصبونات وتابع تفعيل Softmax

CNN

Python Implementation

Build & Train CNN

Model.compile

خطوة 4:
تحديد Optimizer

خطوة 5:

Compile Model

خطوة 6:

تحديد بعض بارامترات التدريب

```
# Define the optimizer
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

تحديد Optimizer وهو هنا خوارزمية Adam بمعدل تعلم 0.001 ومعاملات تدرج beta1, beta2

```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
```

مرحلة Compile Model
نحدد 3 أمور هي:
1- optimizer وهو هنا Adam
2- تابع Loss المستخدم في عملية التدريب وهو هنا Categorical_Crossentropy لأن المسألة هنا تصنيف متعدد الأصناف وليست Binary Classification وفي حال كانت مسألة Binary Classification يستخدم تابع Binary_Crossentropy
3-والمقياس المستخدم لتقييم الأداء وهو الدقة Accuracy ويمكن اختيار Loss

```
epochs = 10  # for better result increase the epochs
batch_size = 250
```

عدد التكرارات
حجم Batch Size
وهو يمثل عدد عينات التدريب التي ستقدم للنموذج (المودل) كل دفعة تدريب وهنا مثلا 250.
بفرض عدد العينات 1000 سنحتاج 1000/250= 4 سيستغرق 4 GPU دفعات ليتم عملية تدريب واحدة.

CNN

Python Implementation

```
datagen = ImageDataGenerator(
rotation_range=5,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
       horizontal_flip=False,
        vertical_flip=False)
datagen.fit(X_train)
```

مجال التدوير العشوائي المستخدم 5 درجات.

مجال ال Zoom هو 0.1

مجال الإزاحة الأفقية والعمودية 0.1

إمكانية القلب الأفقي والعمودي False (يمكن وضعها True) لكن في مثال الأرقام قد تؤثر على الرقم.

يمكن استخدام عمليات أخرى مثل تغيير الإضاءة.

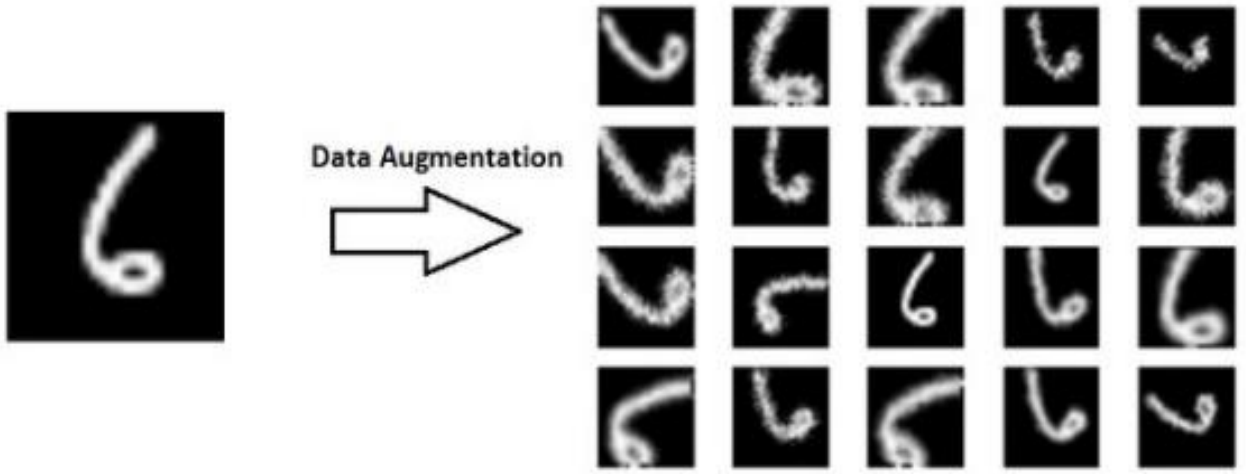
نستخدم ImageDataGenerator لتوليد عينات جديدة (عشوائياً) من عينات مجموعة البيانات

يمكن استخدام عملية واحدة أو أكثر.

Build & Train CNN

خطوة 7: Data Augmentation

تستخدم هذه الخطوة لتجنب Overfitting ولزيادة عدد عينات مجموعة التدريب وتوليد عينات جديدة بخصائص جديدة مما يزيد قوة التدريب



CNN

Python Implementation

```
# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val), steps_per_epoch=X_train.shape[0] // batch_size)
```

نمرر لتابع التدريب

1- عينات التدريب X_train وخرج كل عينة Y_train بعد تمريرها على تابع datagen

2- حجم الدفعة Batch Size

3- عدد التكرارات

4- مجموعة بيانات التحقق وهي X_val, Y_val

5- عدد العينات في كل تكرار وهو ناتج قسمة عدد العينات على حجم الدفعة.

Build & Train CNN

خطوة 8:
تدريب النموذج

Model Fit

تستخدم هذه الخطوة لتجنب Overfitting ولزيادة عدد عينات مجموعة التدريب وتوليد عينات جديدة بخصائص جديدة مما يزيد قوة التدريب

Build & Train CNN

نتيجة التدريب

37800/250 = 151.2≈151

```
Epoch 1/10
151/151 [==============================] - 26s 170ms/step - loss: 1.1310 - acc: 0.6186 - val_loss: 0.2363 - val_acc: 0.9386
Epoch 2/10
151/151 [==============================] - 24s 160ms/step - loss: 0.4386 - acc: 0.8604 - val_loss: 0.1476 - val_acc: 0.9590
Epoch 3/10
151/151 [==============================] - 24s 162ms/step - loss: 0.3276 - acc: 0.8960 - val_loss: 0.1146 - val_acc: 0.9674
Epoch 4/10
151/151 [==============================] - 24s 160ms/step - loss: 0.2716 - acc: 0.9144 - val_loss: 0.1002 - val_acc: 0.9705
Epoch 5/10
151/151 [==============================] - 24s 162ms/step - loss: 0.2458 - acc: 0.9228 - val_loss: 0.0858 - val_acc: 0.9750
Epoch 6/10
151/151 [==============================] - 25s 163ms/step - loss: 0.2242 - acc: 0.9315 - val_loss: 0.0804 - val_acc: 0.9762
Epoch 7/10
151/151 [==============================] - 24s 158ms/step - loss: 0.2076 - acc: 0.9361 - val_loss: 0.0733 - val_acc: 0.9781
Epoch 8/10
151/151 [==============================] - 25s 164ms/step - loss: 0.1981 - acc: 0.9387 - val_loss: 0.0705 - val_acc: 0.9781
Epoch 9/10
151/151 [==============================] - 24s 161ms/step - loss: 0.1886 - acc: 0.9423 - val_loss: 0.0647 - val_acc: 0.9817
Epoch 10/10
151/151 [==============================] - 25s 163ms/step - loss: 0.1713 - acc: 0.9462 - val_loss: 0.0594 - val_acc: 0.9836
```

CNN

Python Implementation

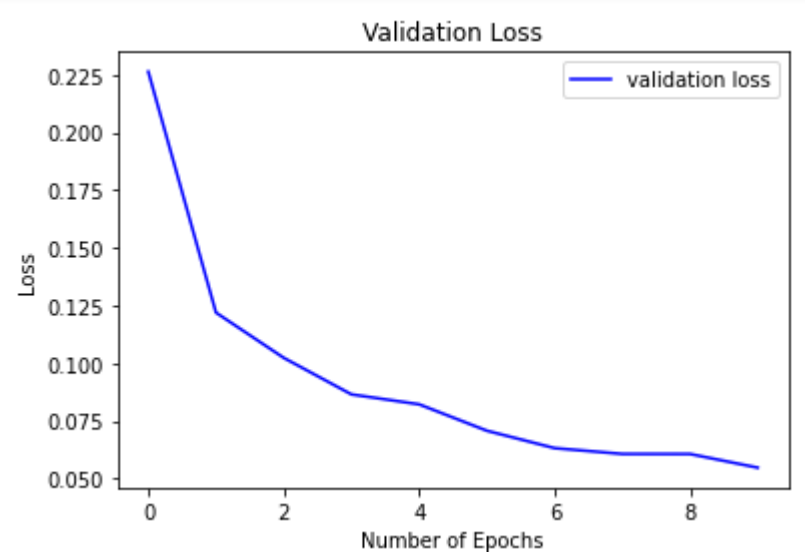
```
# Plot the loss and accuracy curves for training and validation

plt.plot(history.history['val_loss'],  color='b',  label="validation loss")

plt.title("Test Loss")

plt.xlabel("Number of Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()
```

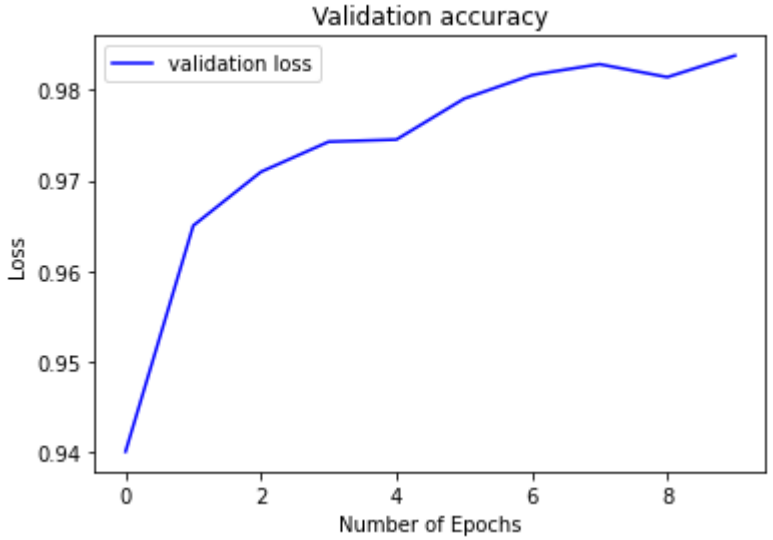
نستخدم تعليمة الرسم history
التدريب للشبكة لعرض نتيجة
Val Loss
Val Accuracy

Build & Train CNN

لتقييم أداء النموذج يتم رسم منحني الأداء ويمكن استخدام مصفوفة الـ Confusion Matrix ومعاملات قياس الأداء أيضاً.

خطوة 9:
رسم منحني الأداء (الدقة والـLOSS)



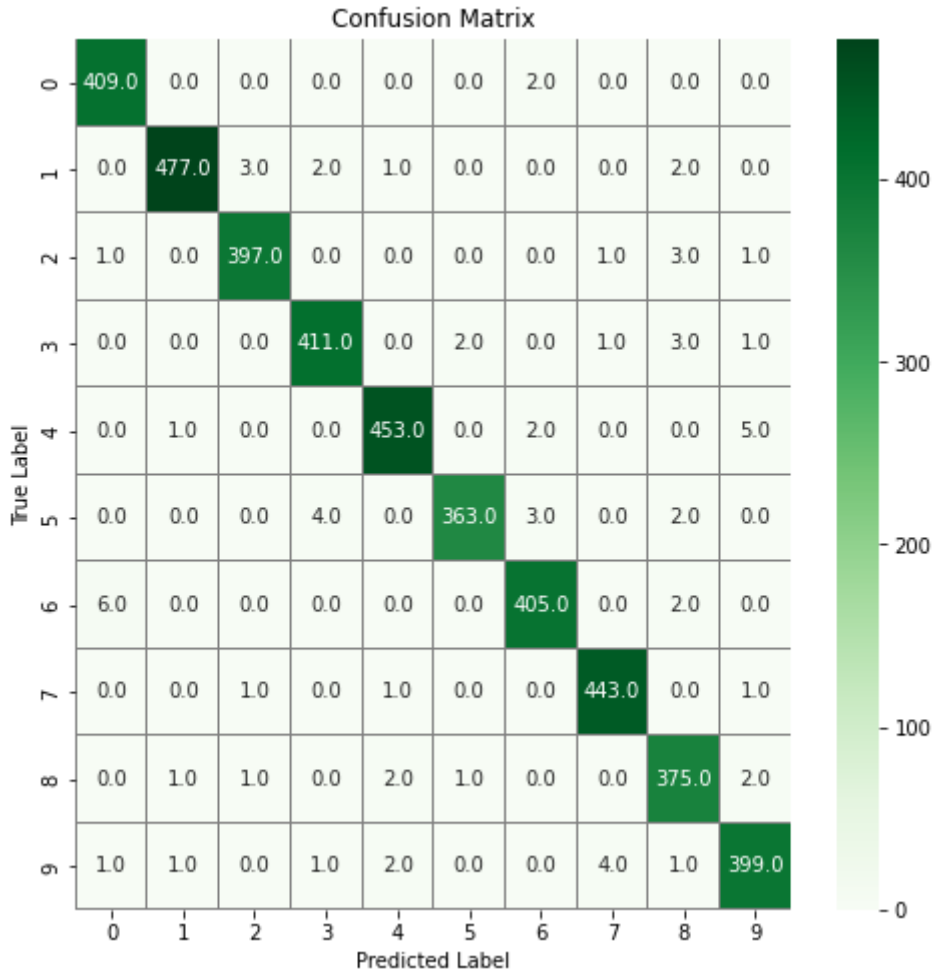


Build & Train CNN

خطوة 9: رسم منحني الأداء (الدقة والـ LOSS)

ترسم مصفوفة ال CM العلاقة بين Predicted classes وTrue Classes من أجل معرفة تفاصيل نتائج كل صنف لوحده ومعرفة الخلل في أي صنف في حال وجوده

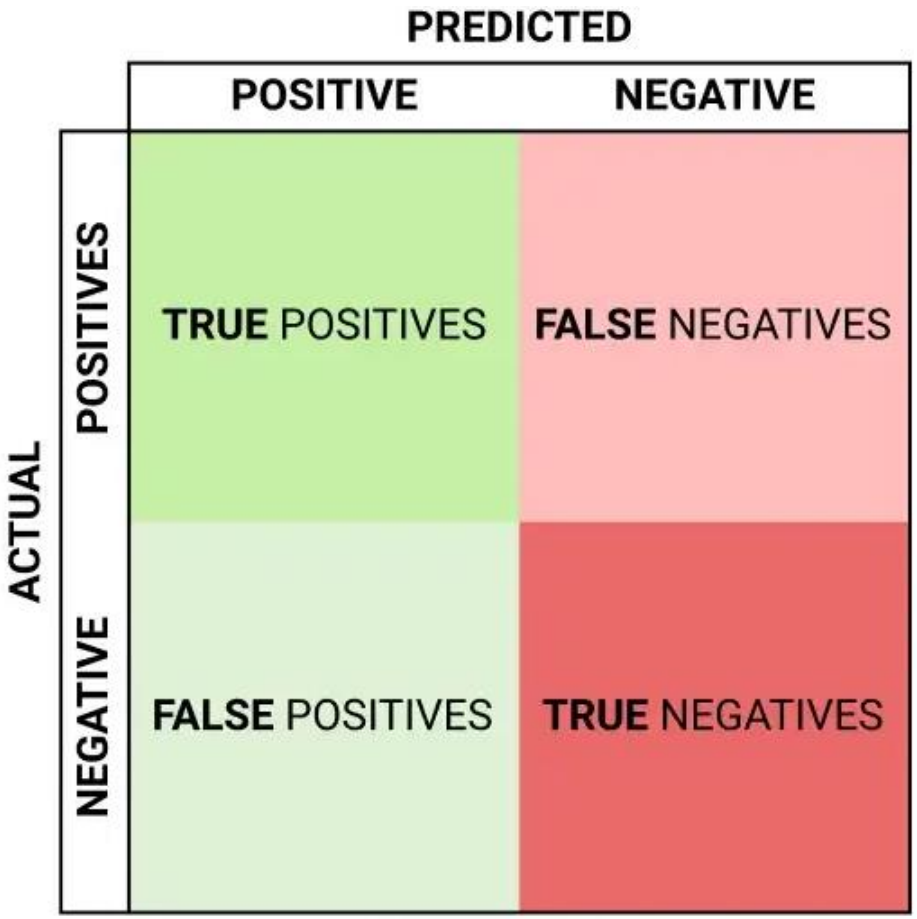
```
# confusion matrix
import seaborn as sns
# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx,annot=True,
linewidths=0.01,cmap="Greens",linecolor="gray",fmt=
'.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

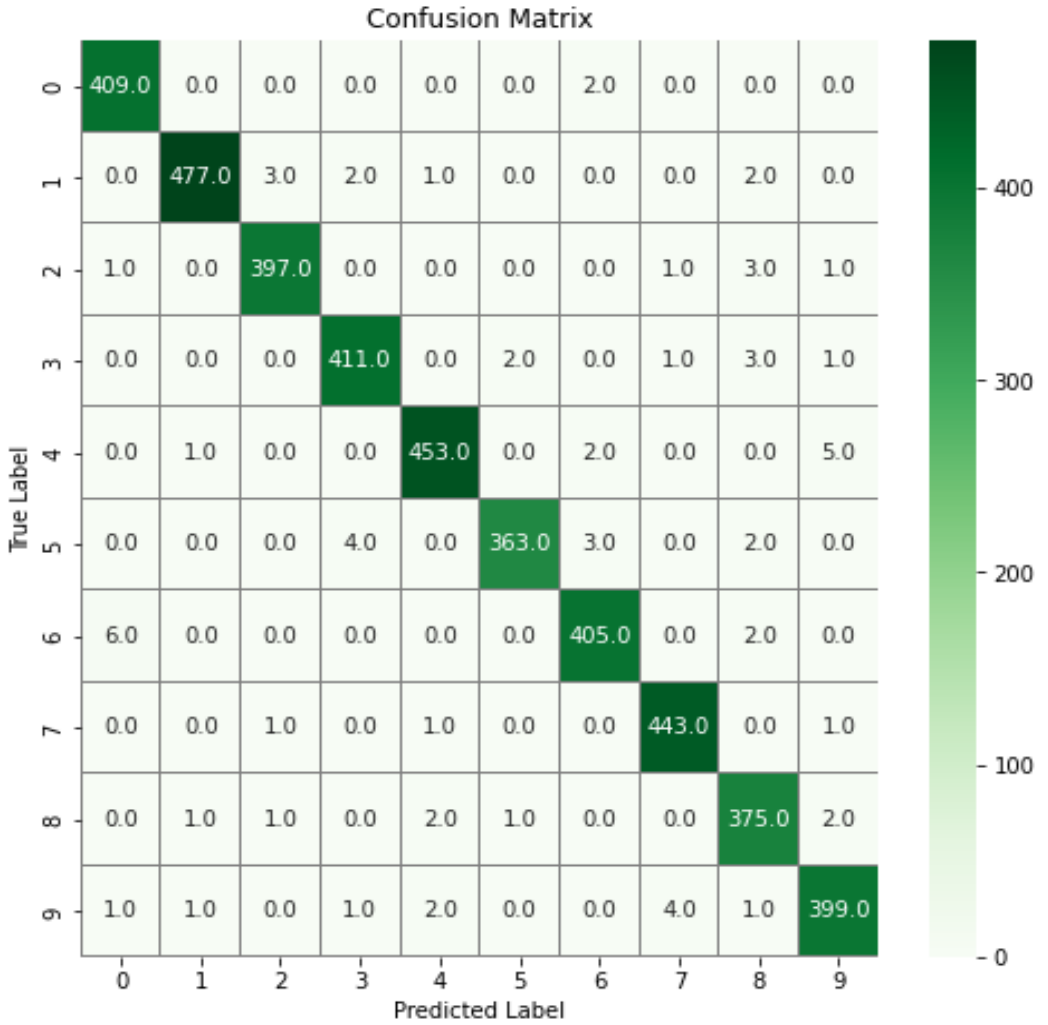


لتقييم أداء النموذج يتم رسم منحنى الأداء ويمكن استخدام مصفوفة الـ Confusion Matrix ومعاملات قياس الأداء أيضاً.

Build & Train CNN

خطوة 9: رسم منحني الأداء (الدقة والـ LOSS)





من خلال مصفوفة CM نستطيع حساب بارامترات

True Positives TP

True Negatives TN

False Positives FP

False Negatives FN

CNN | Python Implementation

Build & Train CNN

```
from sklearn.metrics import classification_report
classification_report(Y_true, Y_pred_classes)
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	411
1	1.00	0.98	0.99	485
2	0.98	0.99	0.98	403
3	0.99	0.98	0.98	418
4	0.99	0.97	0.98	461
5	0.99	0.98	0.98	372
6	0.98	0.99	0.99	413
7	0.98	0.99	0.99	446
8	0.97	0.99	0.98	382
9	0.97	0.97	0.97	409
accuracy			**0.98**	4200
macro avg	0.98	0.98	0.98	4200
weighted avg	0.98	0.98	0.98	4200

الدقة أو مقياس الجودة

Precision= TP/ (TP+FP)

قابلية الإرجاع

Recall = TP/(TP+FN)

الدقة الوسطية **F1-score**

F1-Score = 2*Precision*Recall/(Precision+Recall)

مفهوم الدقة الشامل **Accuracy**

ACC= (TP+TN)/ (TP+TN+FP+FN)

يستخدم لتقييم أداء النموذج حيث يتم حساب قيم Precision, Recall, F1-score لكل صنف من أصناف المسألة

ثم يتم حساب الدقة الوسطية