NUS National University of Singapore

NUS AI SUMMER EXPERIENCE | 2019

Tensorflow with Applications

LI Qiaohong

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

01 Introduction to Tensorflow

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

Key factors driving the success of AI

Big Data

- Caltech-101 (10,000 images 2003)
- ImageNet (14+ million images, 2009)
- Youtube-8M (8+ million videos, 2016)

Hardware Acceleration

- Graphics processing unit (GPU)
- parallelizing heavy matrix operations.

Software Toolboxes

TensorFlow

scikit learn contrib

Deep Learning framework zoo

TensorFlow

Keras

PyTorch

infer.net

Caffe2

Microsoft CNTK

Chainer

DL4J

GLUON

Sonnet

ONNX

theano

mxnet

Deep Learning framework zoo

- Keras
  Quick starter to build NN models
- PyTorch
  Research purposes
- TensorFlow
  Industry production with Google Cloud Service
- MXNet
  Industry production with AWS
- D4LJ
  Android developers should pay attention to D4LJ, for iOS, a similar range of tasks is compromised by Core ML.
- ONNX
  Interaction between different frameworks.

Deep Learning framework zoo

- Industry demand

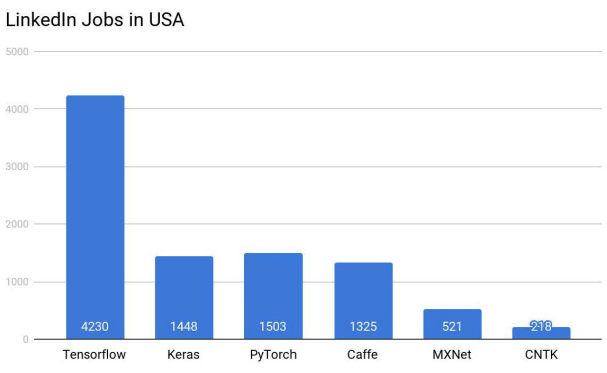


Figure data is from 2019 March 29 Survey

Deep Learning framework zoo

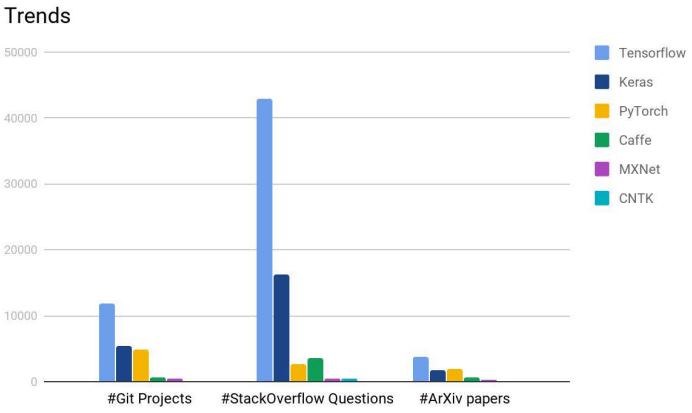


Figure data is from 2019 March 29 Survey

Why Tensorflow?

- Distributed training capability
- Availability for web, mobile, edge, embedded, and more
- Visual debugging with TensorBoard
- Very good documentation and community support
- Great for both research and production

The TensorFlow Software Library

TensorFlow™ is is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

- Source code available at:
  https://github.com/tensorflow/tensorflow
- TensorFlow Python API:
  https://www.tensorflow.org/versions/r2.0/api_docs/python/
- Tensorflow Youtube Channel:
  https://www.youtube.com/channel/UC0rqucBdTuFTjJiefW5t-IQ

Tensorflow: Versions

- Pre-2.0 (r1.14 latest)

  Default static graph
  Eager mode with dynamic graph

  Advantages of graph-based execution:
  Performance optimizations, remote execution and the ability to serialize, export and deploy easily

- **2.0 Beta** (Released June 2019)

  Default dynamic graph

  Advantages of eager execution:
  Ease of use, smooth debugging

Tensorflow 1.x vs. Tensorflow 2

- **Lazy execution**
  Operations are not run by the framework until asked specifically.

```
import tensorflow as tf

a = tf.constant([1, 2, 3])
b = tf.constant([0, 0, 1])
c = tf.add(a, b)

print(c)
print(tf.__version__)
```

```
Tensor("Add_1:0", shape=(3,), dtype=int32)
1.14.0
```

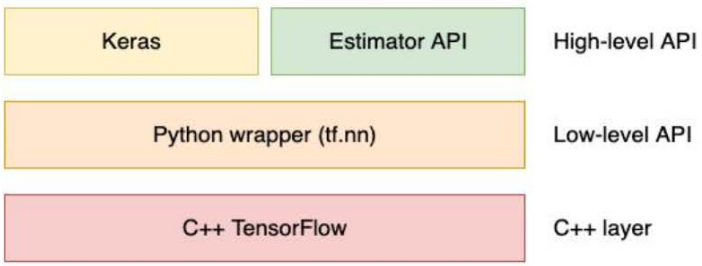
- **Eager execution**
  Evaluates operations immediately, without building graphs: operations return concrete values.

```
import tensorflow as tf

a = tf.constant([1, 2, 3])
b = tf.constant([0, 0, 1])
c = tf.add(a, b)

print(c)
print(tf.__version__)
```

```
tf.Tensor([1 2 4], shape=(3,), dtype=int32)
2.0.0-beta1
```

TensorFlow main architecture

- In Tensorflow 2 it is recommended to use the Keras API.



Introducing Keras

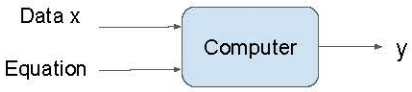
- Stand-alone Keras
  - Has backend support for TensorFlow, CNTK, Theano, and PlaidML
- tf.keras
  - The Sequential API
  - The Functional API
  - The Model Subclassing API

- We rely on **tf.keras** not the stand-alone Keras.
  - import tf.keras and not keras.
  - use the tf.keras documentation on TensorFlow's website and not the keras.io documentation.
  - When using external Keras libraries, make sure they are compatible with tf.keras.
  - Some saved models might not be compatible between versions of Keras.

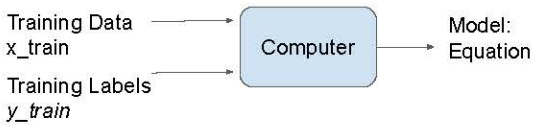
Training your first model

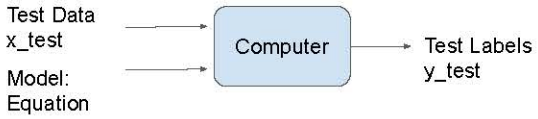
Convert from Celsius to Fahrenheit $f = c \times 1.8 + 32$

- Traditional Programming



- Machine Learning Programming





Training your first model

Convert from Celsius to Fahrenheit $f = c \times 1.8 + 32$

- Traditional Programming

```
def c_to_f (x):
  y = x*1.8+32
  return y
```

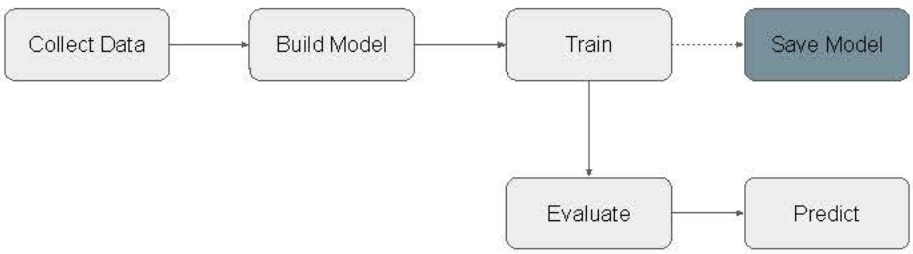
- Machine Learning Programming

```
def train(x_train, y_train):
  # Machine Learning
  return model

def predict(model, x_test):
  # Use model to predict labels
  return y_test
```

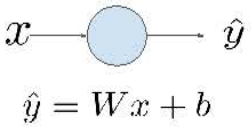
Training your first model

- Convert from Celsius to Fahrenheit

$$f = c \times 1.8 + 32$$



Training your first model



$$\hat{y} = Wx + b$$

- Collect data

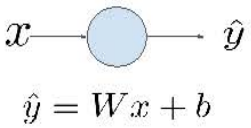
- Build model

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=1, input_shape=[1]))

model.compile(loss='mean_squared_error',
              optimizer=tf.keras.optimizers.Adam(0.1),
              metrics=['mae'])
```

- Train

```
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=500)
```

Training your first model



$$\hat{y} = Wx + b$$

- Evaluate

```
score = model.evaluate(x_test, y_test)
print("Test mse loss: ", score[0], "Test mae: ", score[1])
```

- Predict

```
print(model.predict([100.0]))
```

- Save

```
# Save the model
model.save('my_model.h5')
```

Tensors

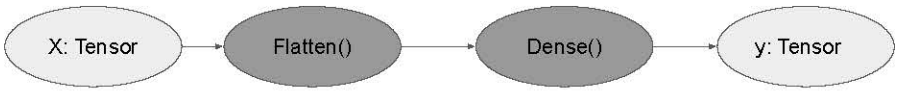
- Tensors
  A tensor is a multi-dimensional array. A tensor could be a scalar (0-d), a vector (1-d), a 2D matrix (2-d), or an N-dimensional matrix (N-d).
- tf.Tensor
  tf.Tensor objects have a data type and a shape.
  Type: string, float32, float16 or int8 and others.
  Shape: The dimensions of the data. e.g., the shape would be () for a scalar, (n) for a vector of size **n**, and (n, m) for a 2D matrix of size **n × m**.

  tf.constant contains fixed values.
  tf.Variable contains changing values.
- tf.Tensor vs. np.array
  - TensorFlow operations convert numpy arrays to Tensors automatically.
  - NumPy operations convert Tensors to numpy arrays automatically.
  - The .numpy() method explicitly converts a Tensor to a numpy array

Operations

- **Operation**

  TensorFlow uses **Tensors** as inputs as well as outputs. A component that transforms input into output is called an **operation.**



Keras Models: Sequential vs. Functional

- Sequential API
  - The **Sequential** API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))
```

- Functional API
  - The **Functional** API is much more versatile and expressive than the **Sequential** API. It allows for multiple input or output models as well as models that share layers.

```
model_input = tf.keras.layers.Input(shape=input_shape)
output = tf.keras.layers.Flatten()(model_input)
output = tf.keras.layers.Dense(128, activation='relu')(output)
output = tf.keras.layers.Dense(num_classes, activation='softmax')(output)
model = tf.keras.Model(model_input, output)
```

Keras models and layers

- **Model** class (tf.keras.Model)
  - .inputs and .outputs: Provide access to the inputs and outputs of the model.
  - .layers: Lists the model's layers as well as their shape.
  - .summary(): Prints the architecture of the model.
  - .save(): Saves the model, its architecture, and the current state of training. It is very useful to resume training later on. Models can be instantiated from a file using tf.keras.models.load_model().
  - .save_weights(): Only saves the weights of the model.

Keras models and layers

- **Layer** class (tf.keras.layers.Layer)
  - .get_weights(): Returns the weights of the layer.
  - .set_weights(weights): Sets the weights of the layer.
- Keras provides pre-made layers for the most common deep learning operations.
  - tf.keras.layers.InputLayer
  - tf.keras.layers.Flatten
  - tf.layers.Dense
  - tf.keras.layers.Dropout
  - tf.layers.MaxPooling2D
  - tf.layers.Conv2D

tf.keras.layers API

Keras model.compile()

After we create the model architecture, and just before we start training the model, we use .compile() to tell how we want the model to be trained.

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- **Optimizer**: This is the component that will operate the gradient descent.
- **Loss**: This is the metric we will optimize.
- **Metrics**: These are additional metric functions evaluated during training to provide further visibility over the model's performance (they are not used in the optimization process).

Keras loss functions

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- Loss is the distance between what we have and what we want.
- Loss is mostly a distance measure.

Keras loss functions

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- Regression

$$\hat{y} = Wx + b$$

e.g. what's your score for course A?

The output $\hat{y}$ is a continuous scalar, in the range [0, 100].

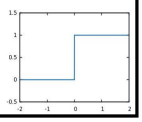
- Classification

$$s = Wx + b$$

$$\hat{y} = \begin{cases} 1, & s \geq 0 \\ 0. & s < 0 \end{cases}$$

e.g. Will you pass course B?

The output $\hat{y}$ is a discrete class label, 1 for pass, 0 for fail.



Keras loss functions

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- Regression
  - Mean Squared Error Loss

    model.compile(loss = 'mse')
    model.compile(loss='mean_squared_error'
)
  - Mean Squared Logarithmic Error Loss
  - Mean Absolute Error Loss
- Classification
  - Binary Classification Loss Functions
    - Binary Cross-Entropy
    - Hinge Loss
    - Squared Hinge Loss
  - Multi-Class Classification Loss Functions
    - Multi-Class Cross-Entropy Loss
    - Sparse Multiclass Cross-Entropy Loss
    - Kullback Leibler Divergence Loss

sparse_categorical_crossentropy vs. categorical_crossentropy
scc: labels in the form of [0,1,2,3]
cc: labels in the form of [[1 0 0 0], [0 1 0 0], [0 0 1 0], [0 0 0 1]]

Keras metrics

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- Regression
  - Mean Squared Error Loss

    model.compile(metrics =['mse'])
  - Mean Squared Logarithmic Error Loss
  - Mean Absolute Error Loss
- Classification
  - accuracy

    Accuracy is a special metric. The "acc" metric can be specified to report on accuracy regardless of the type of problem in question.
  - loss

Keras optimizers

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- Stochastic Gradient Descent (SGD)
  - Combined with weight decay, momentum, Nesterov Momentum
  - ops = tf.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
- Adaptive Moments (Adam)
  - 2015 ICLR Adam: A method for stochastic optimization
  - Preferred by CV practitioners
- Root Mean Squared Propagation (RMSprop)
  - Introduced in Geoffrey Hinton's Coursera course
  - Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude
  - A good choice for RNN

Keras callbacks

- **Callbacks** class (tf.keras.callbacks.Callback)
  - Keras callbacks are utility functions that you can pass to a Keras model's .fit() method to add functionality.
- Pre-defined Keras callbacks include the following:
  - CSVLogger: Logs training information in a CSV file.
  - EarlyStopping: Stops training if the loss or a metric stops improving. It can be useful to avoid overfitting.
  - LearningRateScheduler: Changes learning rate on each epoch according to a schedule.
  - ReduceLROnPlateau: Automatically reduces learning rate when the loss or a metric stops improving.

TensorBoard: TensorFlow's visualization toolkit

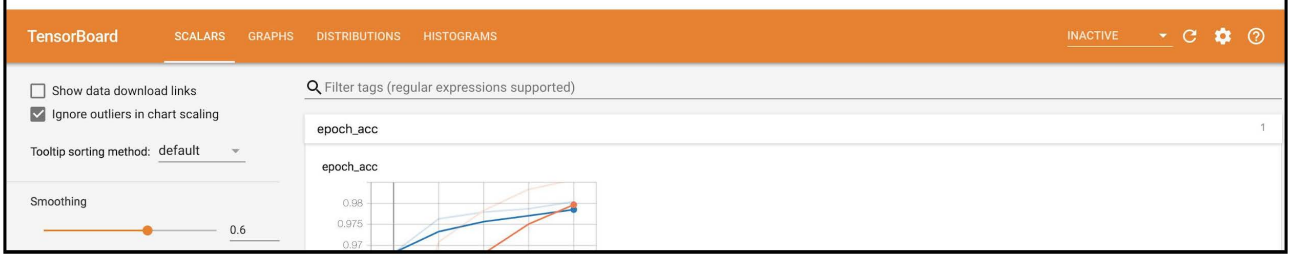
- Using TensorBoard with Keras Model.fit()
  - Tracking experiment metrics like loss and accuracy
  - Display input and output images
  - Visualize the model graph
  - Project embeddings to a lower dimensional space

```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])
```

```
%tensorboard --logdir logs/fit
```

TensorBoard: TensorFlow's visualization toolkit

- The **Scalars** dashboard shows how the loss and metrics change with every epoch.
- The **Graphs** dashboard helps you visualize your model.
- The **Distributions and Histograms** dashboards show the distribution of a Tensor over time. This can be useful to visualize weights and biases and verify that they are changing in an expected way.

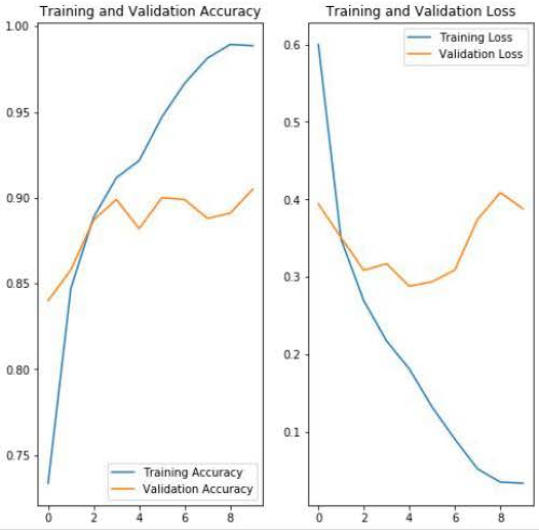


Visualization with the history object

```
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=EPOCHS)
```

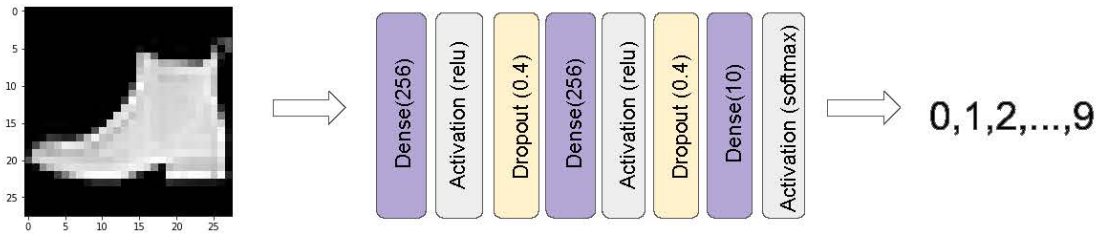
```
acc = history.history['acc']
val_acc = history.history['val_acc']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(8,8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.savefig('./foo.png')
plt.show()
```

Visualization with the history object



Saving and serializing models with tf.keras

- We build the model with three Dense layers. We denote the model after **5 epochs** as model-5, and the model after 10 epochs as **model-10**. Are **model-5** and **model-10** the same model?



Saving and serializing models with tf.keras

- Whole model saving
  - The model's architecture
  - The model's weight values (which were learned during training)
  - The model's training config (what you passed to compile), if any
  - The optimizer and its state, if any (this enables you to restart training where you left off)

```
# Save the model
model.save('path_to_my_model.h5')

# Recreate the exact same model purely from the file
new_model = keras.models.load_model('path_to_my_model.h5')
# Note that the optimizer state is preserved as well:
# you can resume training where you left off.
```

Saving and serializing models with tf.keras

- Export to SavedModel

TensorFlow SavedModel format is a standalone serialization format for Tensorflow objects, supported by TensorFlow serving as well as TensorFlow implementations other than Python.

- A TensorFlow checkpoint containing the model weights.
- A SavedModel proto containing the underlying Tensorflow graph.
- The model's architecture config.

```
# Export the model to a SavedModel
keras.experimental.export_saved_model(model, 'path_to_saved_model')

# Recreate the exact same model
new_model = keras.experimental.load_from_saved_model('path_to_saved_model')
```

Saving and serializing models with tf.keras

- Architecture-only saving

The **config** is a Python dict that enables you to recreate the same model architecture, without any of the information learned previously during training.

```
config = model.get_config()
reinitialized_model = keras.Model.from_config(config)
```

- Weights-only saving

```
weights = model.get_weights()  # Retrieves the state of the model.
model.set_weights(weights)  # Sets the state of the model.

# Save weights to disk
model.save_weights('path_to_my_weights.h5')
# Load weights from disk
new_model.load_weights('path_to_my_weights.h5')
```

Others

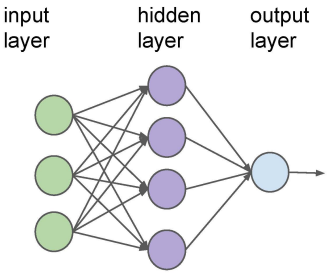
- Tensorflow extra modules
  - **TensorFlow Addons** is a collection of extra functionalities with newest and unstable algorithms.
  - **TensorFlow Extended (TFX)** is an end-to-end machine learning platform for TensorFlow. It offers several useful tools: TensorFlow Data Validation, TensorFlow Transform, TensorFlow Model Analysis, TensorFlow Serving.
- Versatile devices
  - **TensorFlow Lite** is designed to run model predictions on mobile phones and embedded devices.
  - **TensorFlow.js** is designed to run deep learning models in almost any web browser.

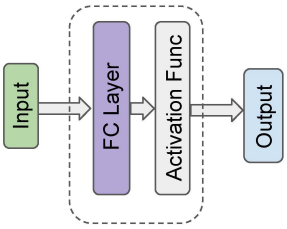
02 Three Core Deep Learning Models

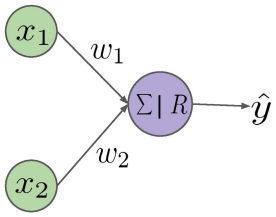
Three Core Deep Learning Models

- MLPs: Multilayer perceptrons
- CNNs: Convolutional neural networks
- RNNs: Recurrent neural networks

MLPs: Multilayer perceptrons

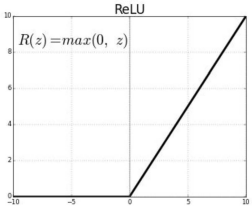




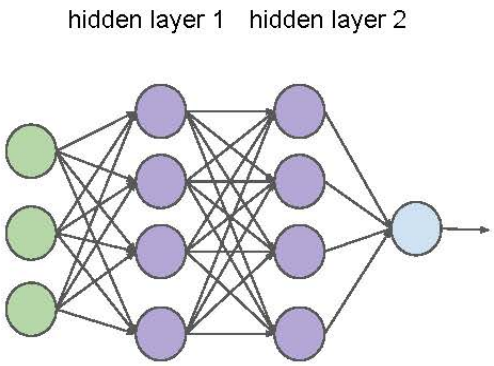


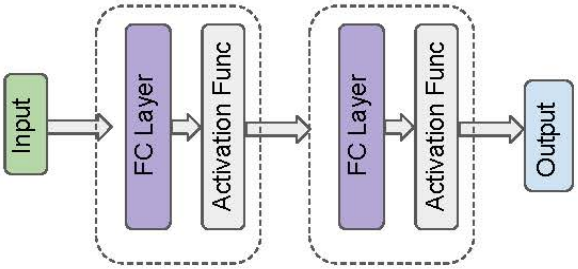
$$s = w_1 x_1 + w_2 x_2 + b$$

$$\hat{y} = R(s) = \begin{cases} 1, & s \geq 0 \\ 0. & s < 0 \end{cases}$$



MLPs: Multilayer perceptrons





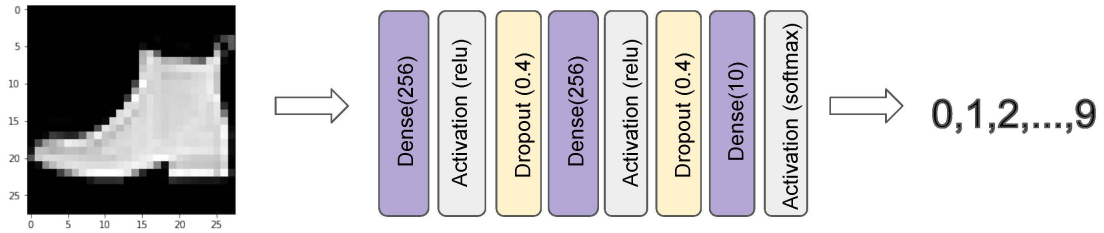
Explore MLPs

- Fashion-MNIST
  - 60,000 training images
  - 10,000 test images
  - 28X28 grayscale image
  - 10 classes

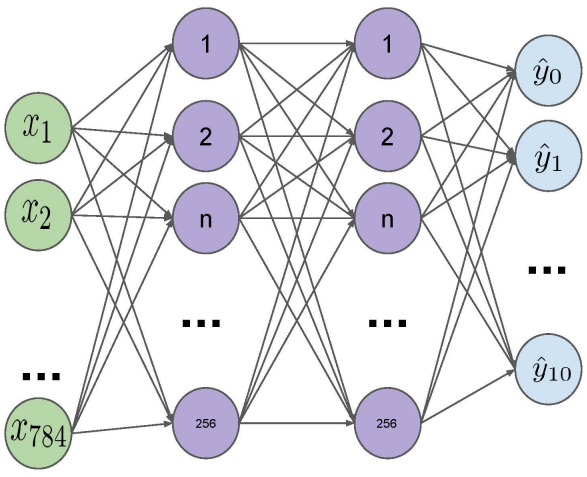
Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

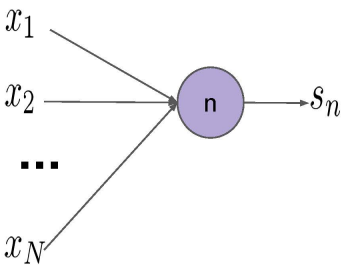
Explore MLPs

- Train the MLP models on Fashion-MNIST dataset.



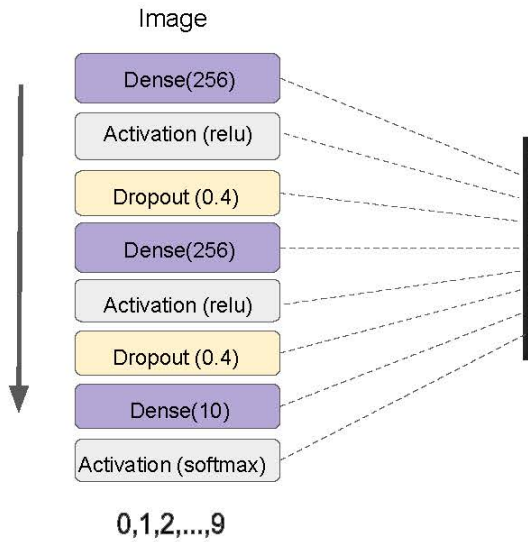
Explore MLPs





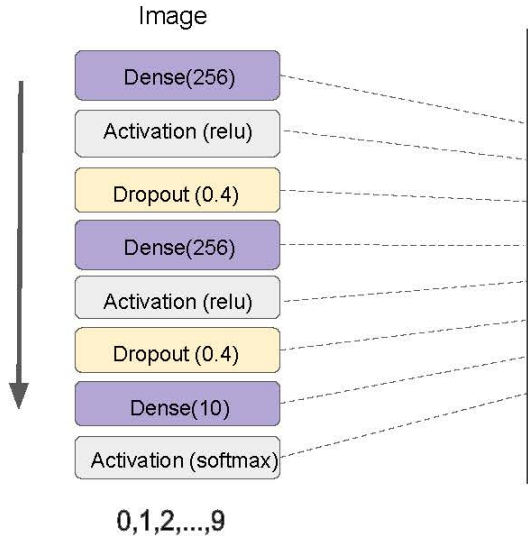
$$s_n = \sum_{i=1}^{N} x_i w_i + b_n$$

Explore MLPs




```
model = Sequential()
model.add(Dense(256, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Explore MLPs



model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               200960
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

Explore MLPs

```
model = Sequential()
model.add(Dense(256, activation='relu',input_shape=(784,)))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))
```

=

```
model = Sequential()
model.add(Dense(256, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(10))
model.add(Activation('softmax'))
```

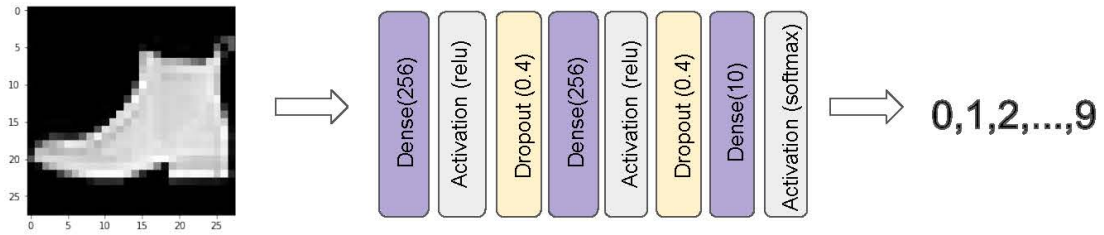
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               200960
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

=

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               200960
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

Explore MLPs

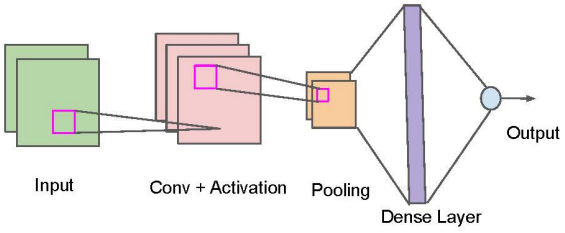
- How many layers in this MLP model?

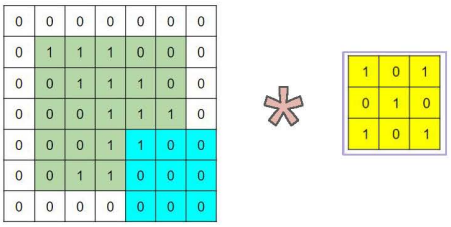


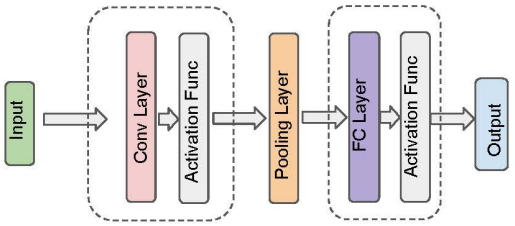
Explore MLPs

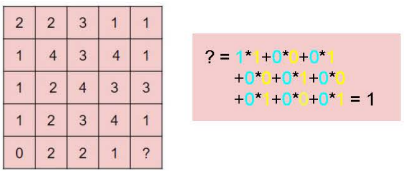
- MLPs properties
  - Input to dense layers is 1D tensor. e.g., images are flattened as 1D vector as input
  - For classification problem, dense layers with softmax activation function are usually adopted as the last layer.
  - MLPs are not optimal for processing sequential and multi-dimensional data patterns.
  - MLPs are not parameter efficient.
  - MLPs, RNNs, and CNNs are combined to make the most out of each network.
- Exercise 1

CNNs: Convolutional neural networks



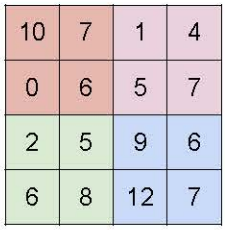




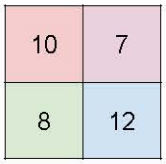


CNNs: Convolutional neural networks

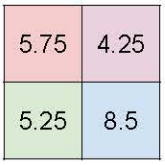
- Pooling Layer



input feature map



max-pooling

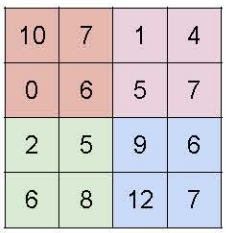


average-pooling

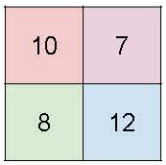
output feature map

CNNs: Convolutional neural networks

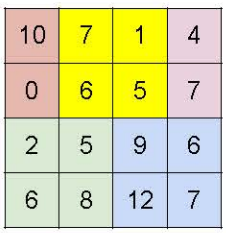
- Stride

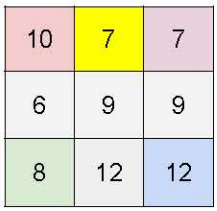


input feature map



max-pooling,
Stride = 2

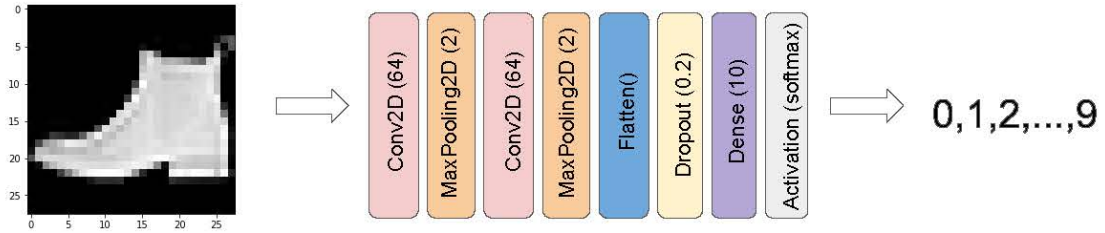




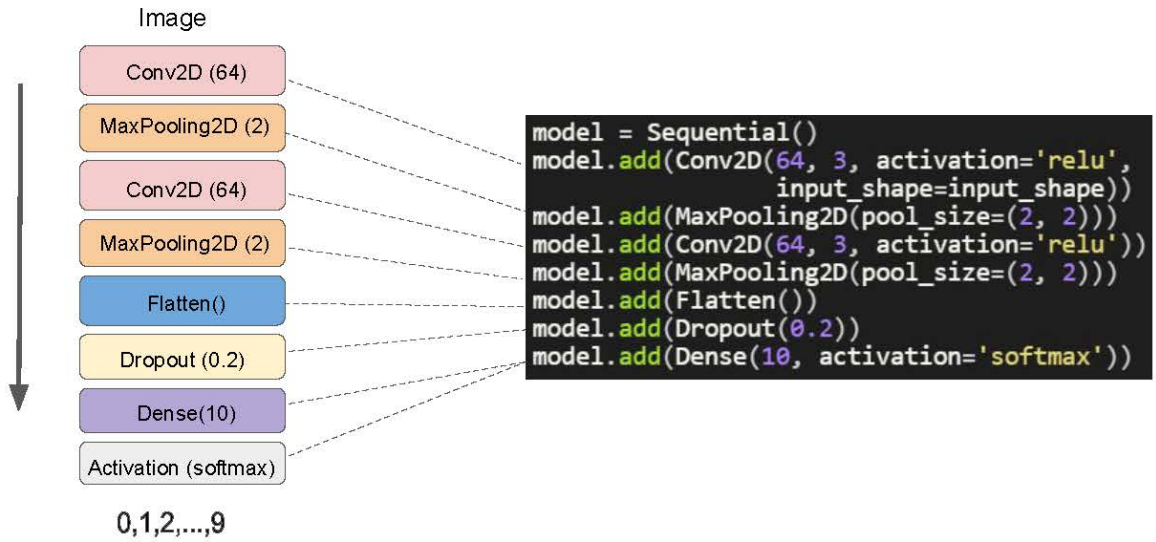
max-pooling,
Stride = 1

Explore CNNs

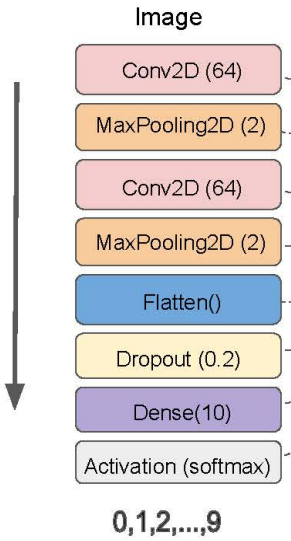
- Train the CNN models on Fashion-MNIST dataset.



Explore CNNs



Explore CNNs

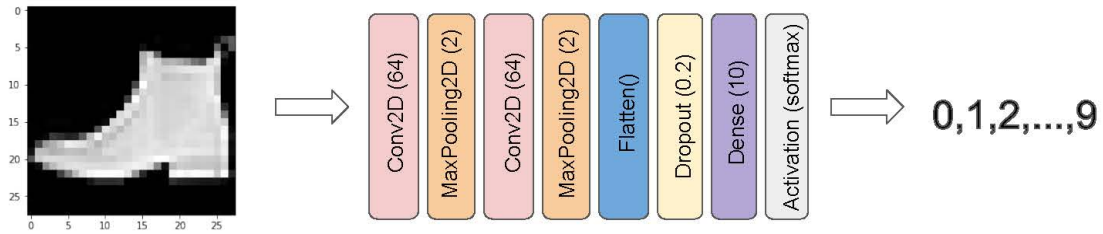


model.summary()

```
Model: "cnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1600)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                16010
=================================================================
Total params: 53,578
Trainable params: 53,578
Non-trainable params: 0
```

Explore CNNs

- How many layers in this CNN model?



Explore CNNs

- CNNs properties
  - Input to CNN (height, width, channels)
  - CNN excels in extracting feature maps for multi-dimensional data like images and videos.
  - CNN in the form of a 1D convolution can also used for sequential input data.
  - CNNs are more parameter efficient and have a higher accuracy than MLPs.
  - MLPs, RNNs, and CNNs are combined to make the most out of each network.

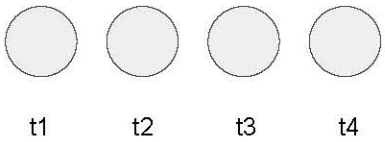
- Exercise 2

RNNs: Recurrent neural networks

- What is the next location of the ball?

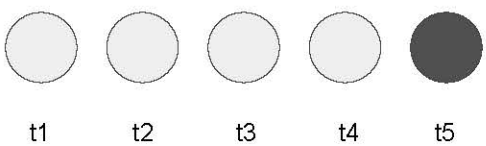
RNNs: Recurrent neural networks

- What is the next location of the ball?



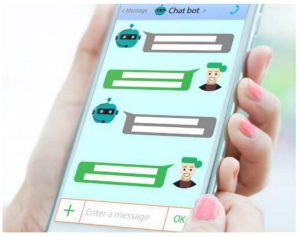
RNNs: Recurrent neural networks

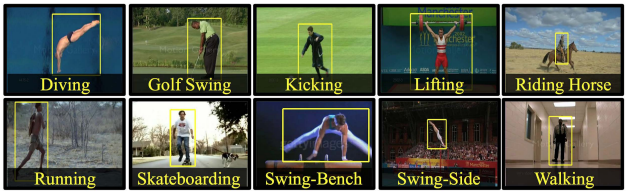
- What is the next location of the ball?



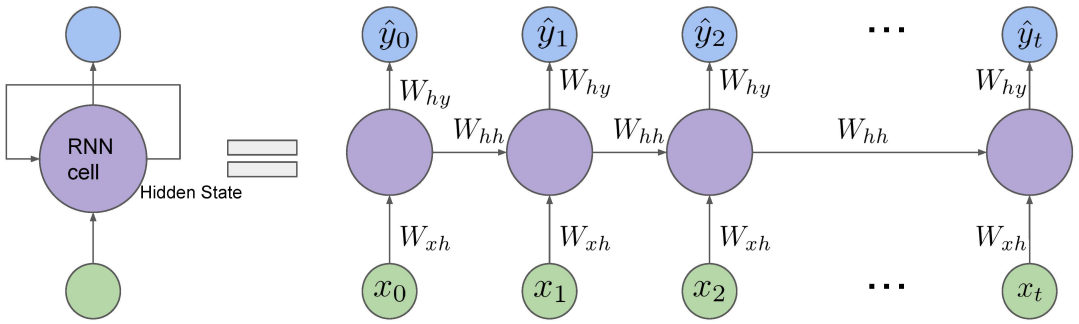
RNNs: Recurrent neural networks

- RNN's are neural networks that are good at modeling sequence data.

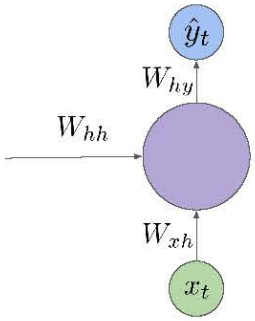




RNNs: Recurrent neural networks



Recurrent neural network (RNN)



Input vector x_t

Update hidden state

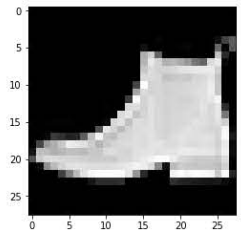
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

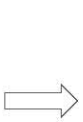
Output vector

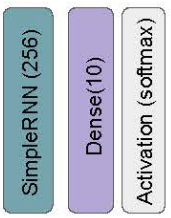
$$\hat{y}_t = \mathrm{softmax}(W_{hy}h_t + c)$$

Explore RNNs

- Train the RNN models on Fashion-MNIST dataset.

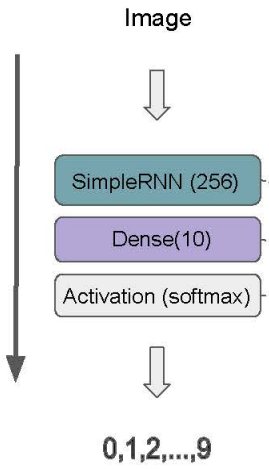






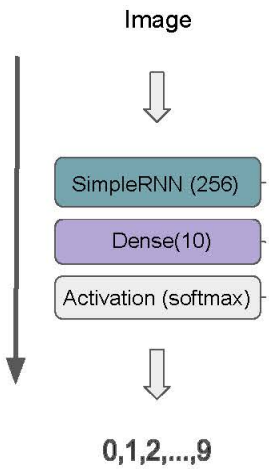
0,1,2,...,9

Explore RNNs




```
model = Sequential(name = 'rnn')
model.add(SimpleRNN(256, dropout = 0.2,
                    input_shape=input_shape))
model.add(Dense(10, activation='softmax'))
```

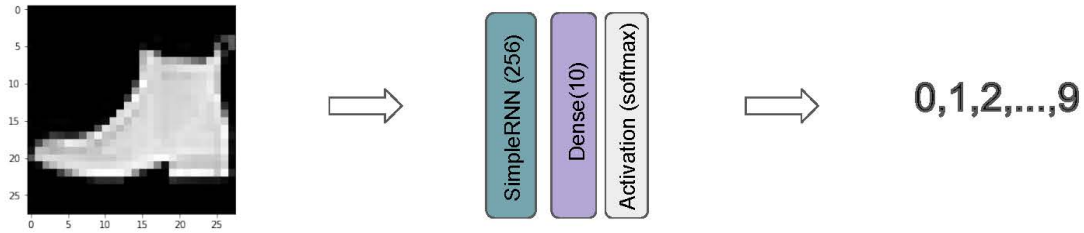
Explore RNNs




```
Model: "rnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 256)               72960
_________________________________________________________________
dense_9 (Dense)              (None, 10)                2570
=================================================================
Total params: 75,530
Trainable params: 75,530
Non-trainable params: 0
```

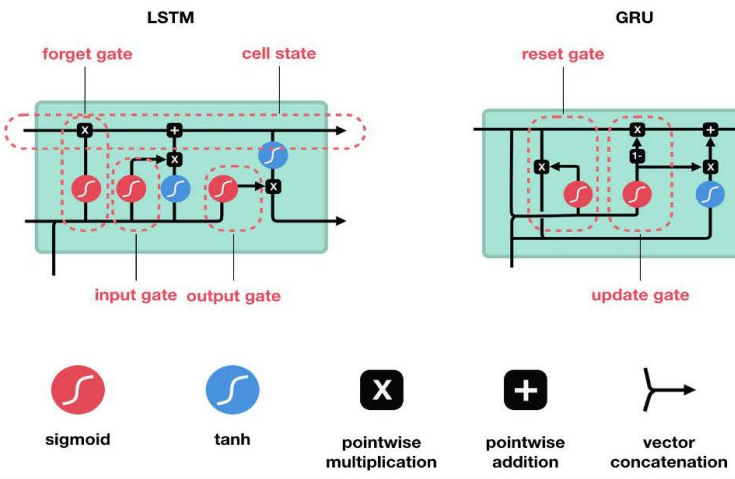
Explore RNNs

- How many layers in this RNN model?



Recurrent neural network (RNN)

- Long Short-Term Memory (LSTM)
- Gated Recurrent Units (GRU)



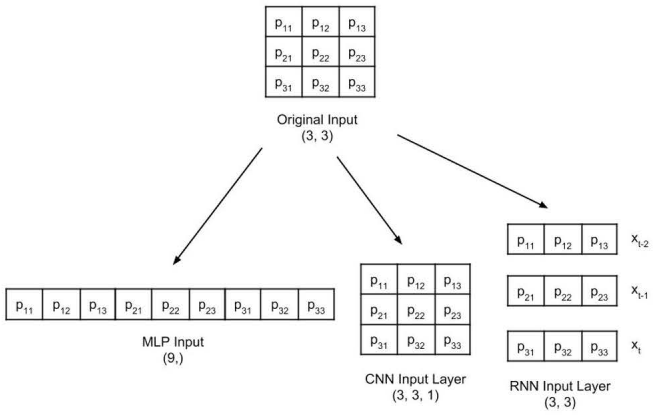
Explore RNNs

- RNNs properties
  - RNNs are popular for sequential data input.
  - RNN suffers from short-memory problems, its variants LSTM and GRU better at learning the long-range dependencies across time steps.
  - MLPs, RNNs, and CNNs are combined to make the most out of each network.

- Exercise 3

Compare MLPs, CNNs, RNNs.

- Input



Compare MLPs, CNNs, RNNs.

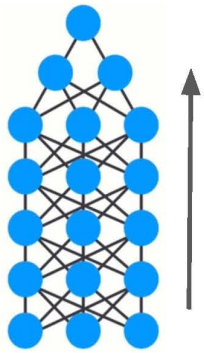
- MLPs are common in simple logistic and linear regression problems. MLP requires a substantial number of parameters to process multi-dimensional data.
- For sequential data input, RNNs are popular because the internal design allows the network to discover dependency in the history of data that is useful for prediction.
- For multi-dimensional data like images and videos, CNN excels in extracting feature maps for classification, segmentation, generation, and other purposes.
- In most deep learning models, MLPs, RNNs, and CNNs are combined to make the most out of each network.

03 Training Neural Networks

Training Neural Networks

- Forward pass to make prediction

  Pred = NN (inputs)

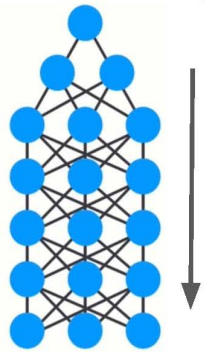


- Compare the prediction to the ground truth using a loss function
- The loss function outputs an error value of how poorly the network is performing

  E = loss(Pred, Truth)

- It use the error value to do backpropagation

  NN.weights=NN.weights-lr* $\frac{\mathrm{d}E}{\mathrm{d}NN_{.weights}}$



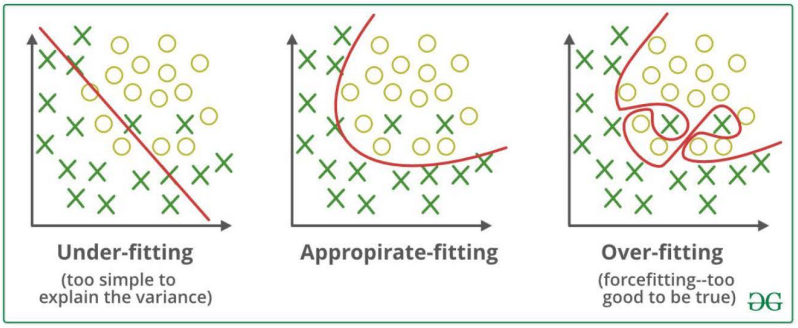
Overfitting and Underfitting

- Overfitting

  Our model doesn't generalize well from our training data to unseen data.

- Underfitting

  Our model is too simple, that it can not work for both training data and unseen data.



Overfitting and Underfitting

Both **bias** and **variance** are forms of prediction error in machine learning.

Simple learners tend to have less variance in their predictions but more bias towards wrong outcomes.

Complex learners tend to have more variance in their predictions.

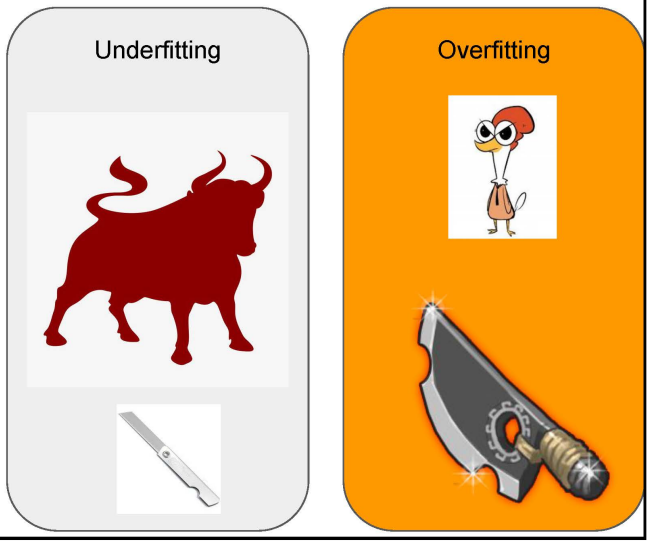
Overfitting and Underfitting

You task is the problem that you need to solve

Kill the bull
Or
Kill the chick

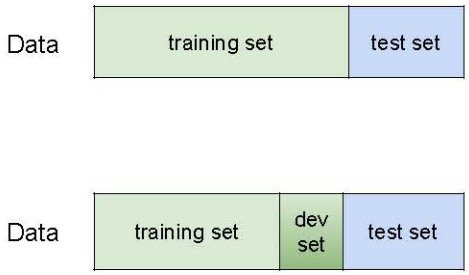
Your model is your toolbox.

Use the small knife
Or
Use the Broadsword



Building a Machine Learning System

- Data driven principle



- Dev Set is used for **hyperparameter** tuning
  - learning rate [0.0001 1]
  - # hidden units for each layer [50 100]
  - momentum
  - mini-batch size
  - # hidden layers
  - learning rate decay
  - # epochs

Overfitting and Underfitting

- How to Detect Overfitting and Underfitting

train set error %	1	15	15	0.5
dev set error %	11	16	30	1
result	High variance	High bias	High bias and high variance	Low bias and low variance

overfitting | underfitting | too bad model | just right

Training Neural Networks

- Reasons for underfitting
  - The model is not powerful enough
  - Over-regularized
  - Not been trained long enough
- Strategies to prevent underfitting
  - Increase model capacity
  - Remove regularization terms
  - Train more epochs

Training Neural Networks

- Reason for overfitting
  - Too much reliance on the training data
  - High model complexity
  - Data scarce
- Strategies to prevent overfitting
  - More training data
  - Data augmentation
  - Reduce the size of the model
  - Add weight regularization
  - Add dropout
  - Early stopping
  - Batch normalization

Training Neural Networks

- Strategies to prevent overfitting
  - More training data
  - Data augmentation

- Collecting more data would be expensive.
- Augmenting dataset is more preferable.

  It means applying **random transformations** to their content, in order to obtain different-looking versions for each, such as scale jittering, random flipping, rotation, color shift, and more.
- Generating synthetic images from 3D models
- Leveraging domain adaptation and generative models (VAEs and GANs)

Training Neural Networks

- Strategies to prevent overfitting
  - More training data
  - Data augmentation
  - Reduce the size of the model
    - Remove some layers
    - Use less nodes at each layer
    - Prune the model by removing the least important neurons.

Training Neural Networks

- Strategies to prevent overfitting
  - More training data
  - Data augmentation
  - Reduce the size of the model
  - Add weight regularization

$$L(y, \hat{y}) + \lambda R(P)$$

L1 regularization

L1 regularization introduces sparsity to make some of your weight parameters zero.

$$R_{L1}(P) = \|P\|_1 = \sum_k |P_k|$$

L2 regularization (weight decay)

L2 regularization penalizes the weights parameters without making them sparse—one reason why L2 is more common.

$$R_{L2}(P) = \frac{1}{2}\|P\|_2^2 = \frac{1}{2}\sum_k P_k^2$$

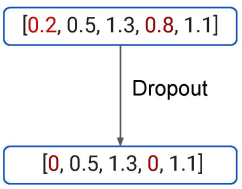
```
# We instantiate a regularizer (L1 for example):
l1_reg = tf.keras.regularizers.l1(0.01)
# We can then pass it as a parameter to the target model's layers:
model = Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), padding='same',
                 activation='relu', input_shape=input_shape,
                 kernel_regularizer=l1_reg))
```

Training Neural Networks

- Strategies to prevent overfitting
  - More training data
  - Data augmentation
  - Reduce the size of the model
  - Add weight regularization
  - Add dropout

    Randomly "dropping out" (i.e. set to zero) a number of output features of the layer during training

    - dropout rate 0.2~0.5
    - no dropout at test time

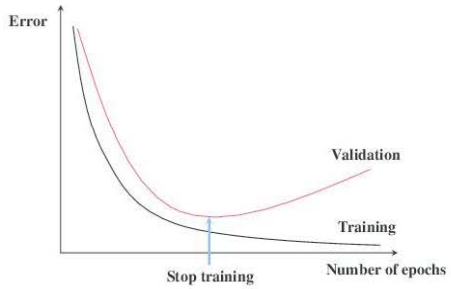


- tf.keras introduces dropout via the Dropout layer, which gets applied to the output of layer right before.

```
tf.keras.layers.Dropout(0.5)
```

Training Neural Networks

- Strategies to prevent overfitting
  - More training data
  - Data augmentation
  - Reduce the size of the model
  - Add weight regularization
  - Add dropout
  - Early stopping



- Figure out the proper number of training epochs a model needs.
- Monitor and plot the validation loss and metrics as a function of the training iterations, and we restore the saved weights at the optima.
- tf.keras.callbacks.EarlyStopping can do early stopping automatically.

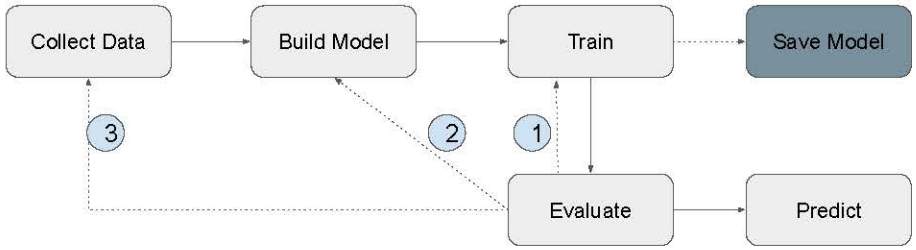
Training Neural Networks

- Strategies to prevent overfitting
  - More training data
  - Data augmentation
  - Reduce the size of the model
  - Add weight regularization
  - Add dropout
  - Early stopping
  - Batch normalization

- It subtracts the batch mean and divides by the batch standard deviation.
  - 2015 JMLR Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

tf.keras.layers.BatchNormalization()

Train Neural Networks



04 Advanced CNN Architectures

Advanced CNN Architectures

	Architecture	# layers	# parameters	Top-1 Accuracy	Top-5 Accuracy
VGG16	serial CNN layers	16	138M	70.5	91.2
Inception V1	inception module	22	7M	69.8	89.3
ResNet-50	skip connection	50	25.5M	75.2	93

- ImageNet (millions of images in 22k categories)
  - ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)
    - 1000 images in 1000 categories
    - 1.2 million training, 50k validation, 150k testing images

VGG-16 architecture

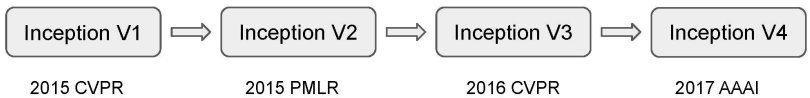
- Standardizing CNN architectures
  - Replacing large convolutions with multiple smaller ones
  - Increasing the depth of the feature maps
  - Augmenting data with scale jittering

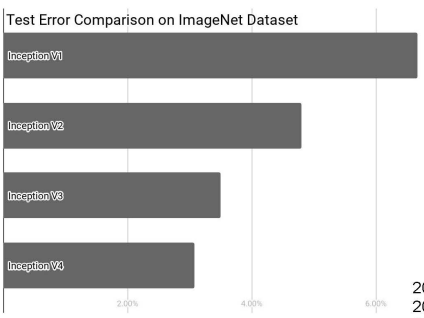
VGG-16 architecture

- Contributions – standardizing CNN architectures
  - Replacing large convolutions with multiple smaller ones
  - Increasing the depth of the feature maps
  - Augmenting data with scale jittering
  - Replacing fully-connected layers with convolutions

Popular CNN architectures

- GoogleNet

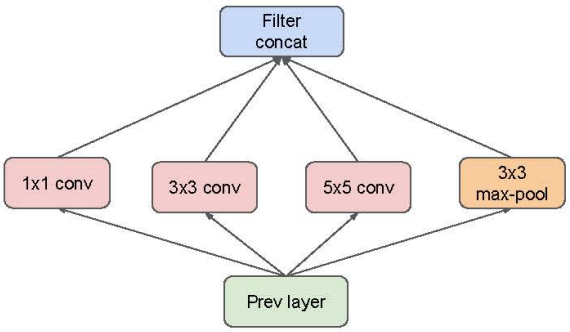


- 

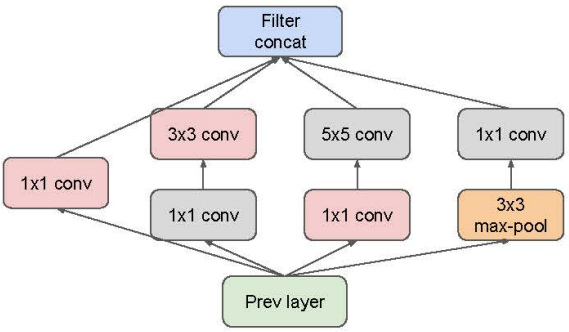
2015 CVPR Going Deeper With Convolutions
2015 PMLR Batch Normalization Accelerating Deep Network Training by Reducing Internal Covariate Shift
2016 CVPR Rethinking the Inception Architecture for Computer Vision
2017 AAAI Inception-v4, inception-resnet and the impact of residual connections on learning

Popular CNN architectures

- GoogleNet
  - Inception Module



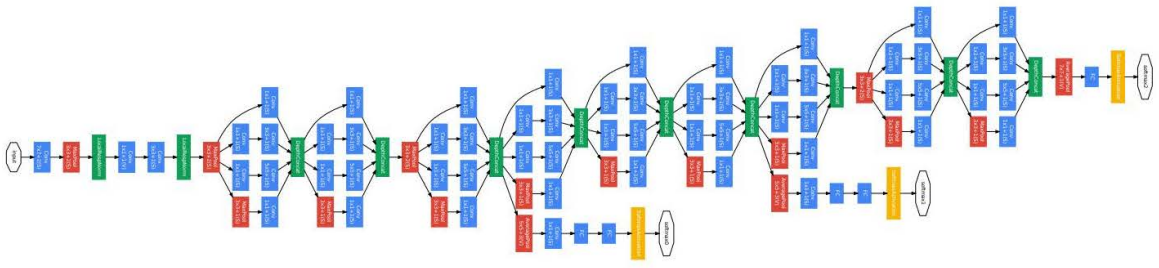
naive inception module



inception module v1

Popular CNN architectures

- GoogleNet (Inception V1)

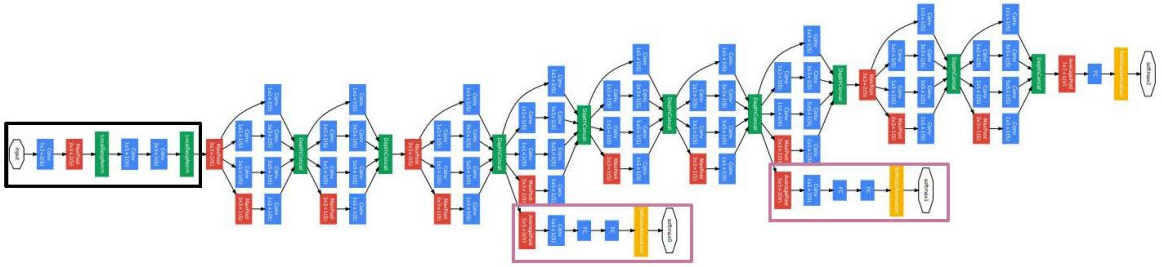


GoogLeNet has 9 such inception modules stacked linearly.

Source: Inception V1 paper

Popular CNN architectures

- GoogleNet (Inception V1)



Fighting vanishing gradient with intermediary losses

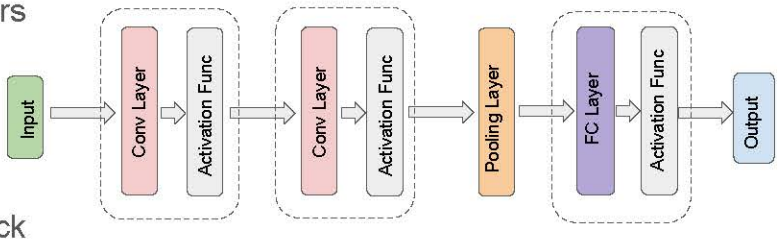
Source: Inception V1 paper

GoogLeNet

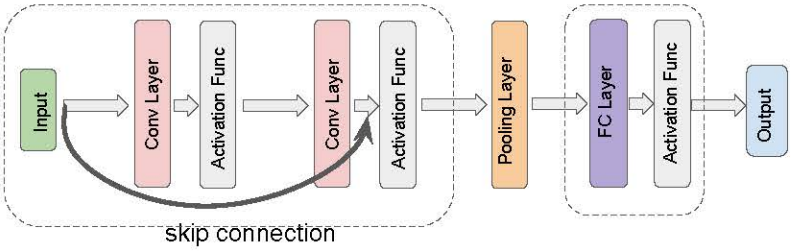
- Contributions – popularizing larger blocks and bottlenecks
  - Capturing various details with Inception modules
  - Using 1 x 1 convolutions as bottlenecks
  - Pooling instead of fully-connecting
  - Fighting vanishing gradient with intermediary losses

Popular CNN architectures

- 2 CNN layers

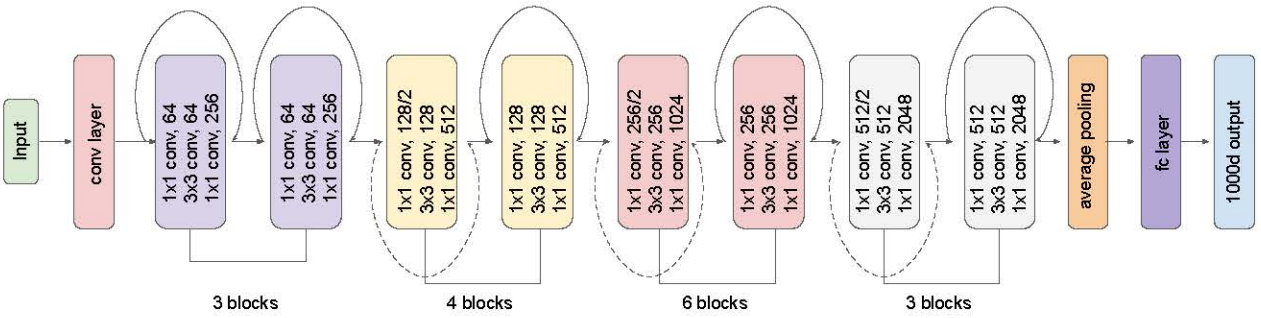


- ResNet block



Popular CNN architectures

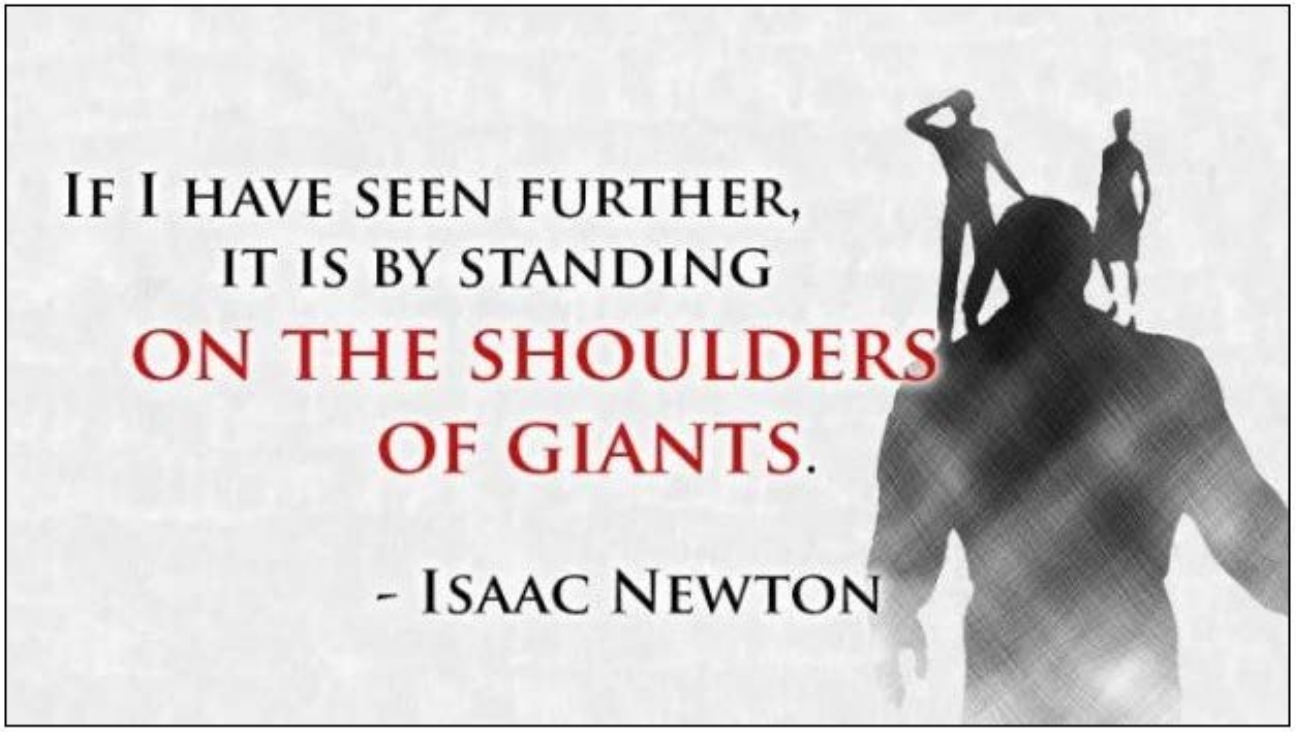
- ResNet-50



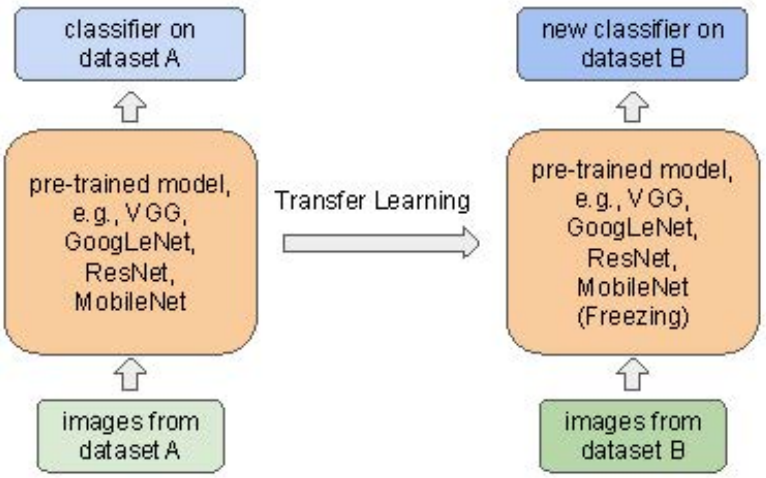
ResNet

- Contributions – forwarding the information deeper
  - Estimating a residual function instead of a mapping
  - Going "ultra-deep"

05 Transfer Learning with Tensorflow Hub

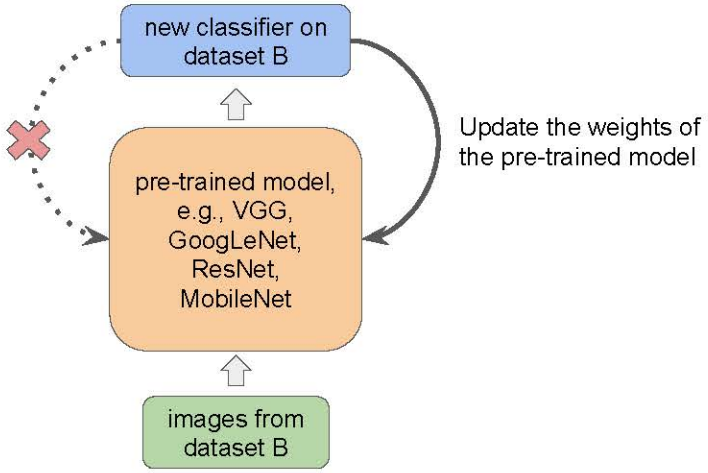


Transfer Learning



Transfer Learning

- Feature Extraction
- Fine-Tuning



TensorFlow Hub

TensorFlow Hub is a repository of pre-trained models.

1. Select a mode from tfhub.dev.
2. The model is stored in model_url.
3. Read the model details to know how to interface with it. e.g. what input (image size) the model expects.
4. Fetch the model and instantiate it as a Keras layer.
5. Add layers on top of it for our task.

TensorFlow Hub

TensorFlow Hub is a repository of pre-trained models.

```
import tensorflow as tf
import tensorflow_hub as hub

url = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/2"
hub_feature_extractor = hub.KerasLayer(
    url, trainable=False,
    input_shape=(299, 299, 3),
    output_shape=(2048,),
    dtype=tf.float32)

inception_model = Sequential(
    [hub_feature_extractor, Dense(num_classes, activation='softmax')],
name="inception_tf_hub")
```

Computer Vision Example of Transfer Learning

- General machine learning workflow
  a. Examine and understand the data
  b. Build an input pipeline (Keras ImageDataGenerator)
  c. Compose our model
  d. Load in our pretrained base model (and pretrained weights)
  e. Stack our classification layers on top
  f. Train our model
  g. Evaluate model

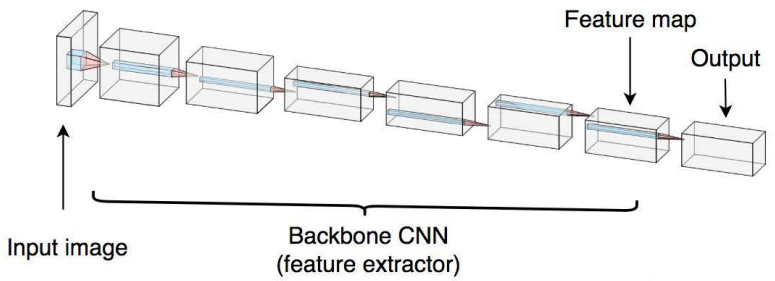
- Exercise 6

06 More CV Applications

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

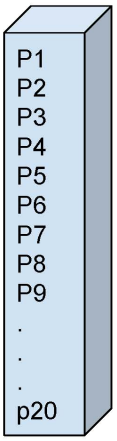
CV applications

- **Recognition** vs. **Detection** vs. **Segmentation**
  - The input are the same. (image_height, image_width, n_channels)
  - Feature extraction part are the same. (CNN backbone model)
  - They differ in their outputs.



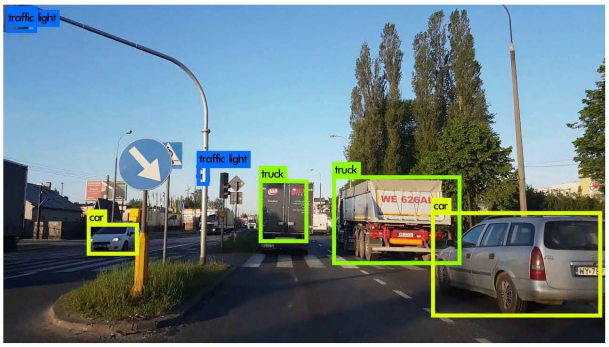
Recognition

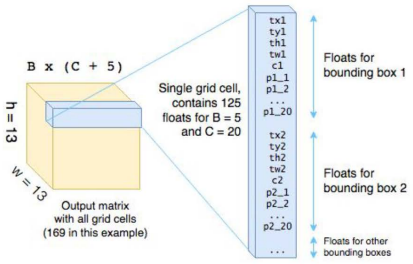
- **Recognition** outputs 1-vector indicating the probability of images belongs to each class



Detection

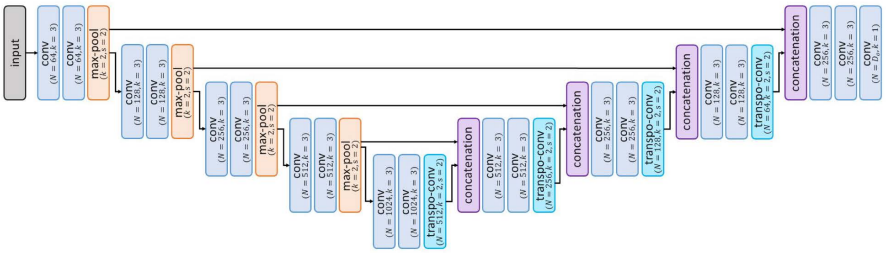
- Detection outputs the coordinates and bounding box sizes as well as the class probabilities for each candidate box. (**Classification+Localization**)





Semantic Segmentation

- Segmentation outputs N maps with input image size, where pixel-level label is required.



07 Exercises

Google Colab

- A free cloud service based on Jupyter Notebooks that supports free GPU
  - Installation and configuration is easy
  - Documents that contain live code, equations, visualizations and narrative text.
  - Easy to share and co-operation with others
  - Free GPU for experiments
- A short tutorial
  - Set up free GPU
  - Install python packages
  - Code Cell (write yr codes) & Text Cell (write related text in markdown)
  - Mount Google Drive for saving data
  - Integration with GitHub

LI Qiaohong

qhli@nus.edu.sg

96995747

NUS National University of Singapore

NUS AI SUMMER EXPERIENCE 2019

Thank You

Tensorflow with Applications

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM