Deep Learning 02: *CNN*

Lecture 11

Computer Vision for Geosciences

2021-05-28

Universidad Nacional Autónoma de México

Table of Contents

Last week: MLP for MNIST-fashion dataset classification task

```
import tensorflow as tf

# Load data
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

# Preprocess data
X_train, X_test, X_valid = X_train/255.0, X_test/255.0, X_valid/255.0

# Build model (using the Sequential API)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
model.summary()

# Compile model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])

# Train model
history = model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
                    epochs=30, # nb of times X_train is seen seen
                    batch_size=32) # nb of images per training instance
print('training instances per epoch = {}'.format(X_train.shape[0] / 32))

# Plot training history
import pandas as pd
pd.DataFrame(history.history).plot()

# Evaluate model
test_loss, test_acc = model.evaluate(X_test,  y_test)
print('Test accuracy:', test_acc)

# Predict
img = X_test[0,:,:]
img = (np.expand_dims(img,0)) # add image to a batch
y_proba = model.predict(img).round(2)
y_pred = np.argmax(model.predict(img), axis=-1)

plt.bar(range(10), y_proba[0])
plt.imshow(img[0,:,:], cmap='binary')
plt.title('class {} = {}'.format(y_pred, class_names[np.argmax(y_proba)]))
```

1.1 Load data
- training dataset
- validation dataset
- test dataset

1.2 Preprocess data
- scale pixel intensities to 0–1

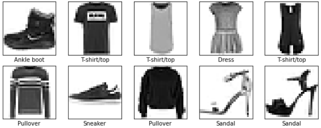
2.1 Build model
- set layer type/order

2.2 Compile model
- set loss function
- set optimizer
- set metrics

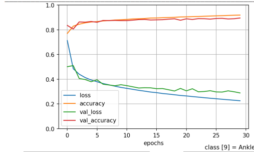
3. Train model
- learn layer parameters (weights/biases)
- plot training history (check for overfitting)

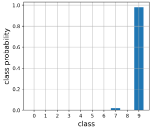
4. Evaluate model
- evaluate accuracy on test dataset

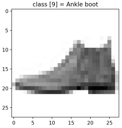
5. Predict from model
- predict image class using learned model




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```







Last week: MLP for MNIST-fashion dataset classification task

```
import tensorflow as tf

# Load data
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

# Preprocess data
X_train, X_test, X_valid = X_train/255.0, X_test/255.0, X_valid/255.0

# Build model (using the Sequential API)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
model.summary()

# Compile model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])

# Train model
history = model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
                    epochs=30, # nb of times X_train is seen seen
                    batch_size=32) # nb of images per training instance
print('training instances per epoch = {}'.format(X_train.shape[0] / 32))

# Plot training history
import pandas as pd
pd.DataFrame(history.history).plot()

# Evaluate model
test_loss, test_acc = model.evaluate(X_test,  y_test)
print('Test accuracy:', test_acc)

# Predict
img = X_test[0,:,:]
img = (np.expand_dims(img,0)) # add image to a batch
y_proba = model.predict(img).round(2)
y_pred = np.argmax(model.predict(img), axis=-1)

plt.bar(range(10), y_proba[0])
plt.imshow(img[0,:,:], cmap='binary')
plt.title('class {} = {}'.format(y_pred, class_names[np.argmax(y_proba)]))
```

1.1 Load data
- training dataset
- validation dataset
- test dataset

1.2 Preprocess data
- scale pixel intensities to 0–1

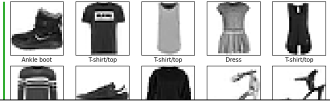
2.1 Build model
- set layer type/order

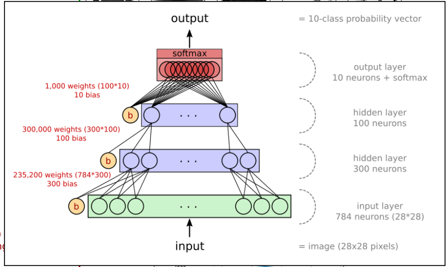
2.2 Compile model
- set loss function
- set optimizer
- set metrics

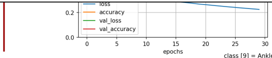
3. Train model
- learn layer parameters (weights/biases)
- plot training history (check for overfitting

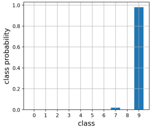
4. Evaluate model
- evaluate accuracy on test dataset

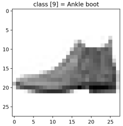
5. Predict from model
- predict image class using learned model











Last week: MLP for MNIST-fashion dataset classification task

```
import tensorflow as tf

# Load data
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

# Preprocess data
X_train, X_test, X_valid = X_train/255.0, X_test/255.0, X_valid/255.0

# Build model (using the Sequential API)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
model.summary()

# Compile model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])

# Train model
history = model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
                    epochs=30, # nb of times X_train is seen seen
                    batch_size=32) # nb of images per training instance
print('training instances per epoch = {}'.format(X_train.shape[0] / 32))

# Plot training history
import pandas as pd
pd.DataFrame(history.history).plot()

# Evaluate model
test_loss, test_acc = model.evaluate(X_test, y_test)

# Predict
img = X_test[0,:,:]
y_proba = model.predict(img).round(2)

plt.bar(range(10), y_proba[0])
plt.imshow(img[0,:,:], cmap='binary')
plt.title('class {} = {}'.format(y_pred, class_names[np.argmax(y_proba)]))
```

1.1 Load data
- training dataset
- validation dataset
- test dataset

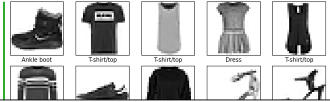
1.2 Preprocess data
- scale pixel intensities to 0–1

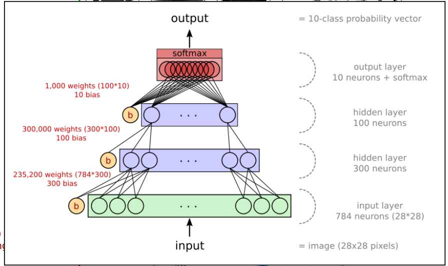
2.1 Build model
- set layer type/order

2.2 Compile model
- set loss function
- set optimizer
- set metrics

3. Train model
- learn layer parameters (weights/biases)
- plot training history (check for overfitting

4. Evaluate model





MLP are powerful, but break for large images due to the huge amount of parameters to optimize

EX1: *simple* model above on the *simple* MNIST-fashion dataset (28x28 pix) $\Rightarrow$ 266,610 parameters

EX2: 100x100 image = 10,000 pixels, with first hidden layer having 1,000 neurons (which is already very restrictive)
$\Rightarrow$ 10,000 x 1,000 = 10 million connections, only for the first layer!

This week: CNN for MNIST-fashion dataset classification task

```python
# Build model (MLP)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

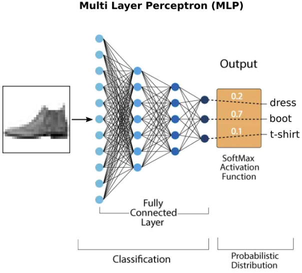
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_5 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_5 (Dense)              (None, 300)               235500
_________________________________________________________________
dense_6 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

```python
# Build model (CNN)
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, 7, activation="relu", padding="same", input_shape=[28, 28, 1]),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    tf.keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    tf.keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

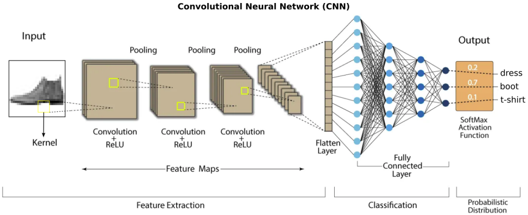
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        3200
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 128)               295040
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 1,413,834
Trainable params: 1,413,834
Non-trainable params: 0
_________________________________________________________________
```

This week: CNN for MNIST-fashion dataset classification task

```
# Build model (MLP)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```




```
# Build model (CNN)
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, 7, activation="relu", padding="same", input_shape=[28, 28, 1]),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    tf.keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    tf.keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation="softmax")
])
```



Desining and training your own network can be difficult (or impossible if you do not have enough data)

$\Rightarrow$ **Transfer Learning** allows to fine-tune a pretrained network

$\Rightarrow$ Most famous CNN networks achieving very good performances on the ImageNet dataset:

(ImageNet = several millions of images, large size (256 pixels), with >1000 classes)

- LeNet-5 (1998)
- AlexNet (2012)
- GoogLeNet (2014)
- ResNet (2015)
- Xception (2016)
- SENet (2017)

Desining and training your own network can be difficult (or impossible if you do not have enough data)

$\Rightarrow$ **Transfer Learning** allows to fine-tune a pretrained network

$\Rightarrow$ Most famous CNN networks achieving very good performances on the ImageNet dataset:

(ImageNet = several millions of images, large size (256 pixels), with >1000 classes)

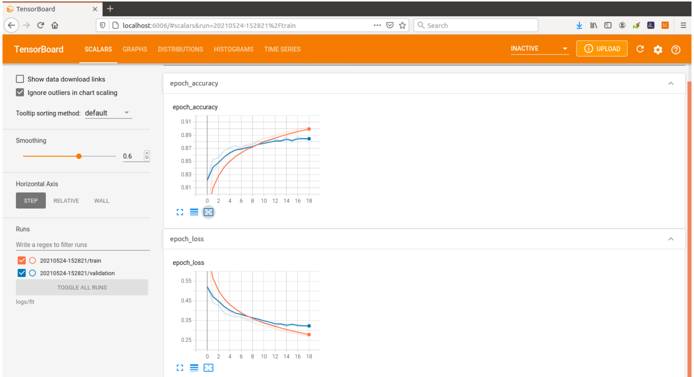
- LeNet-5 (1998)
- AlexNet (2012)
- GoogLeNet (2014)
- ResNet (2015)
- Xception (2016)
- SENet (2017)

In the today's exercice we will use the **Xception** model with weigths learned from the **ImageNet** dataset

TensorBoard: TensorFlow's visualization toolkit

⇒ TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases
- etc.



TensorBoard: TensorFlow's visualization toolkit

⇒ TensorBoard is installed during the TensorFlow conda installation

⇒ To use it, you should:

1. Add the **tf.keras.callbacks.TensorBoard** callback to the Keras Model.fit() method *(ensures that logs are created and stored)*

```
# Create callback
import datetime
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)

# Add callback to model.fit()
history = model.fit(X_train, y_train, callbacks=[tensorboard_callback])
```

2. Run TensorBoard from command line

```
$ conda activate tf
$ cd <working dir>
$ tensorboard --logdir logs/fit    # set directory used to store logs
```

3. Open a web-browser to the address

```
http://localhost:6006/
```

Nota Bene: you can open it directly from a Jupyter cell (after training has finished however) as follows:

```
%load_ext tensorboard          # Load the TensorBoard notebook extension
%tensorboard --logdir logs     # Open TensorBoard in cell
```

TensorBoard: TensorFlow's visualization toolkit

⇒ TensorBoard is installed during the TensorFlow conda installation

⇒ To use it, you should:

1. Add the **tf.keras.callbacks.TensorBoard** callback to the Keras Model.fit() method *(ensures that logs are created and stored)*

```
# Create callback
import datetime
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)

# Add callback to model.fit()
history = model.fit(X_train, y_train, callbacks=[tensorboard_callback])
```

2. Run TensorBoard from command line

```
$ conda activate tf
$ cd <working dir>
$ tensorboard --logdir logs/fit     # set directory used to store logs
```

3. Open a web-browser to the address

```
http://localhost:6006/
```

Nota Bene: you can open it directly from a Jupyter cell (after training has finished however) as follows:

```
%load_ext tensorboard          # Load the TensorBoard notebook extension
%tensorboard --logdir logs     # Open TensorBoard in cell
```

TensorBoard: TensorFlow's visualization toolkit

⇒ TensorBoard is installed during the TensorFlow conda installation

⇒ To use it, you should:

1. Add the **tf.keras.callbacks.TensorBoard** callback to the Keras Model.fit() method *(ensures that logs are created and stored)*

```
# Create callback
import datetime
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)

# Add callback to model.fit()
history = model.fit(X_train, y_train, callbacks=[tensorboard_callback])
```

2. Run TensorBoard from command line

```
$ conda activate tf
$ cd <working dir>
$ tensorboard --logdir logs/fit    # set directory used to store logs
```

3. Open a web-browser to the address

```
http://localhost:6006/
```

Nota Bene: you can open it directly from a Jupyter cell (after training has finished however) as follows:

```
%load_ext tensorboard          # Load the TensorBoard notebook extension
%tensorboard --logdir logs     # Open TensorBoard in cell
```

TensorBoard: TensorFlow's visualization toolkit

⇒ TensorBoard is installed during the TensorFlow conda installation

⇒ To use it, you should:

1. Add the **tf.keras.callbacks.TensorBoard** callback to the Keras Model.fit() method *(ensures that logs are created and stored)*

```
# Create callback
import datetime
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)

# Add callback to model.fit()
history = model.fit(X_train, y_train, callbacks=[tensorboard_callback])
```

2. Run TensorBoard from command line

```
$ conda activate tf
$ cd <working dir>
$ tensorboard --logdir logs/fit     # set directory used to store logs
```

3. Open a web-browser to the address

```
http://localhost:6006/
```

Nota Bene: you can open it directly from a Jupyter cell (after training has finished however) as follows:

```
%load_ext tensorboard          # Load the TensorBoard notebook extension
%tensorboard --logdir logs     # Open TensorBoard in cell
```

TensorBoard: TensorFlow's visualization toolkit

⇒ TensorBoard is installed during the TensorFlow conda installation

⇒ To use it, you should:

1. Add the **tf.keras.callbacks.TensorBoard** callback to the Keras Model.fit() method *(ensures that logs are created and stored)*

```
# Create callback
import datetime
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)

# Add callback to model.fit()
history = model.fit(X_train, y_train, callbacks=[tensorboard_callback])
```

2. Run TensorBoard from command line

```
$ conda activate tf
$ cd <working dir>
$ tensorboard --logdir logs/fit     # set directory used to store logs
```

3. Open a web-browser to the address

```
http://localhost:6006/
```

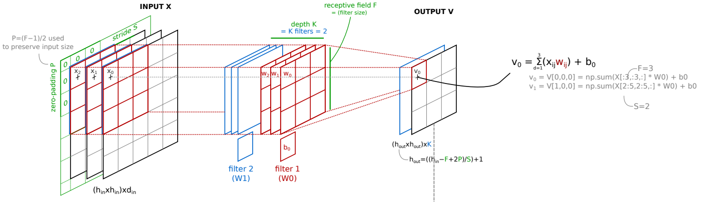
Nota Bene: you can open it directly from a Jupyter cell (after training has finished however) as follows:

```
%load_ext tensorboard           # Load the TensorBoard notebook extension
%tensorboard --logdir logs      # Open TensorBoard in cell
```

CNN cheat sheet: layers types and hyperparameters

CONV **= convolutional layer**

=> 1 filter = "boxcar" filter (where each pixel is multiplied by a weight, products summed across bands, and bias added to the result), with sliding step S
=> convolution layer = N filters



EXAMPLE:

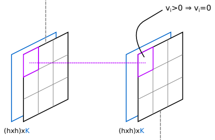
Input volume:
size h_{in}=5, d_{in}=3, zero-padding P=0
=> X.shape = (h_{in}+2*P,h_{in}+2*P,d_{in}) = (5,5,3)

Filter kernel:
depth K=2 (W0 & W1), size F=3, stride S=1
=> W0.shape = (F,F,d_{in}) = (3,3,3)
=> W0 weights = F*F*d_{in} = 3^3 = 27, W0 bias = 1

Output volume:
h_{out} = ((h_{in}−F+2P)/S)+1 = (5-3+0)/1+1 = 3
=> V.shape = (3x3)x2

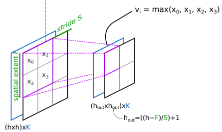
ReLu **= activation layer**

=> activation function
=> applies threshold



POOL **= max pool layer**

=> reduces dimentionality (for memory issue)
=> enables network to see image at large scale



FC **= fully connected**

Hyper parameters

- **receptive field** (**F**)
  = filter size
  NB: usually an odd number, so that it is centered on a central pixel
- **depth** (**K**)
  = number of filters
  NB: depth column = set of neurons that are all looking at the same region of the input
- **stride** (**S**)
  = number of pixels the filter slides across the image at each step
  EX: stride 2 => filter moves 2 pix at a time => produces smaller outputs
- **zero-padding** (**P**)
  = pad the input volume with zeros around the border

NB: hyper-parameters control the output volume size:
width & height = ((W−F+2P)/S) +1   where W = input width/height
depth = K