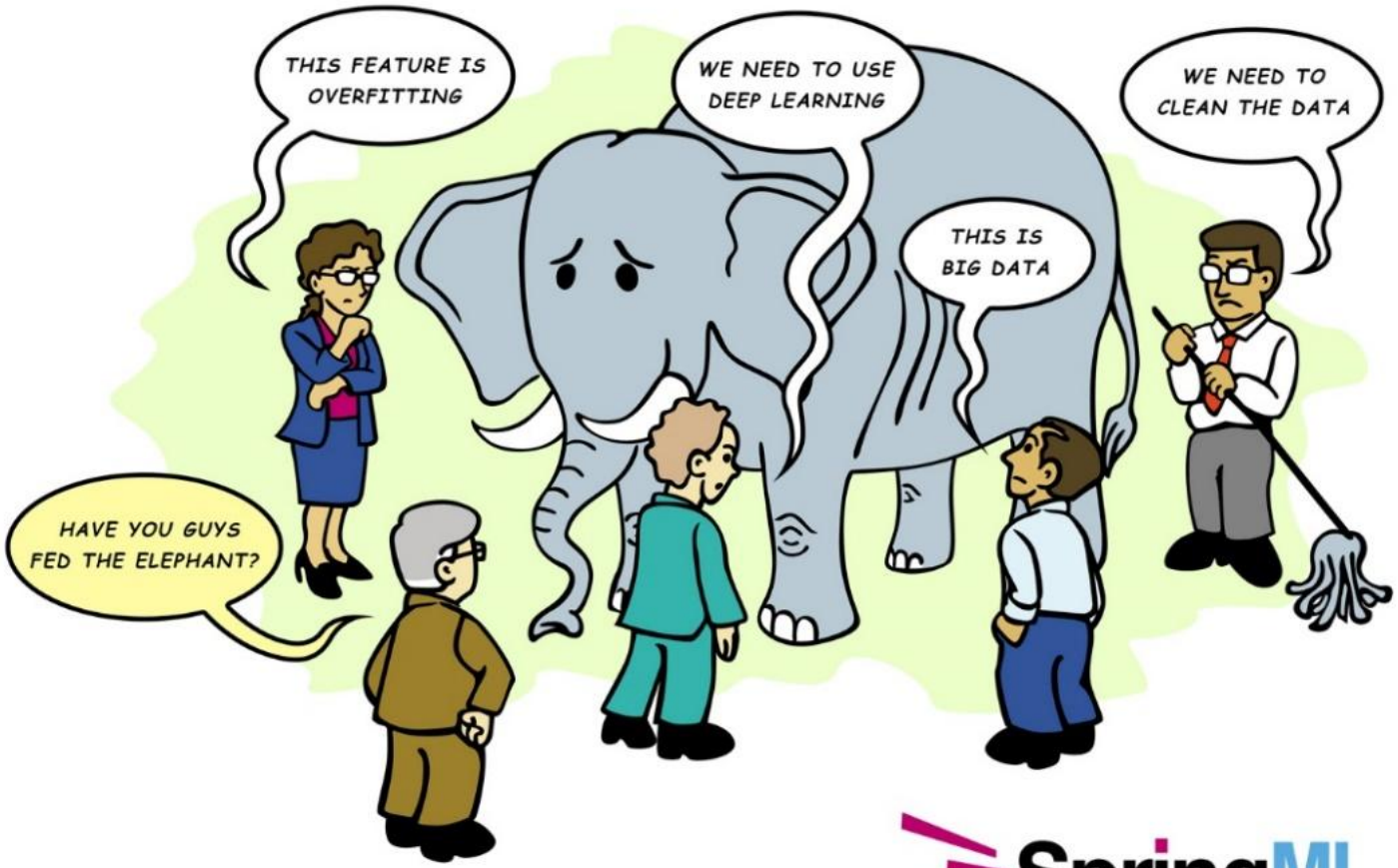
Deep learning for image processing

Ruxandra Stoean

Further bibliography

- Ian Goodfellow et al, Deep Learning (Adaptive Computation and Machine Learning series), MIT Press, 2016 http://www.deeplearningbook.org/
- John Kelleher, Deep Learning (MIT Press Essential Knowledge series), 2019
- Charu Aggarwal, Neural Networks and Deep Learning: A Textbook 2nd ed, Springer, 2023
- Aston Zhang et al, Dive into Deep Learning, Cambridge University Press, 2023
- Simon Prince, Understanding Deep Learning, MIT Press, 2023
- Jonah Gamba, Deep Learning Models, A Practical Approach for Hands-On Professionals, 2024


THIS FEATURE IS OVERFITTING
WE NEED TO USE DEEP LEARNING
WE NEED TO CLEAN THE DATA
THIS IS BIG DATA
HAVE YOU GUYS FED THE ELEPHANT?

SpringML

Deep Learning

What society thinks I do

What my friends think I do

What other computer scientists think I do

What mathematicians think I do

What I think I do

```
In [1]:
import keras
Using TensorFlow backend.
```

What I actually do

https://www.kdnuggets.com/2017/08/first-steps-learning-deep-learning-image-classification-keras.html

Definitions

- "*For most flavors of the old generations of learning algorithms … performance will plateau. … deep learning … is the first class of algorithms … that is scalable. … performance just keeps getting better as you feed them more data.*" - Andrew Ng

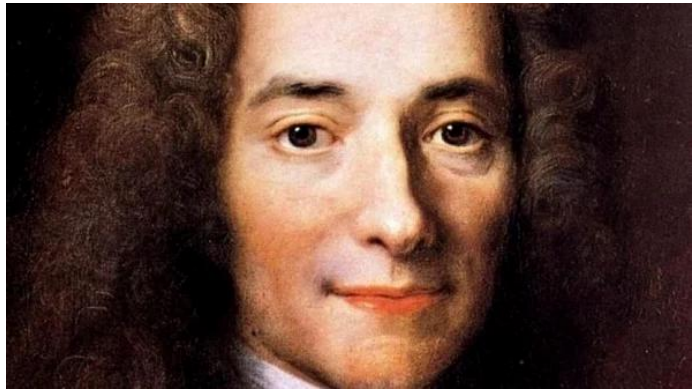
- "*The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning.*" – Ian Goodfellow

- "*Deep learning [is] … a pipeline of modules all of which are trainable. … deep because [has] multiple stages in the process of recognizing an object and all of those stages are part of the training*" - Yann LeCun

- "*At which problem depth does Shallow Learning end, and Deep Learning begin? Discussions with DL experts have not yet yielded a conclusive response to this question. […], let me just define for the purposes of this overview: problems of depth > 10 require Very Deep Learning.*"- Jurgen Schmidhuber

https://machinelearningmastery.com/what-is-deep-learning/

"I never guess. It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."



“The more I read, the more I acquire,

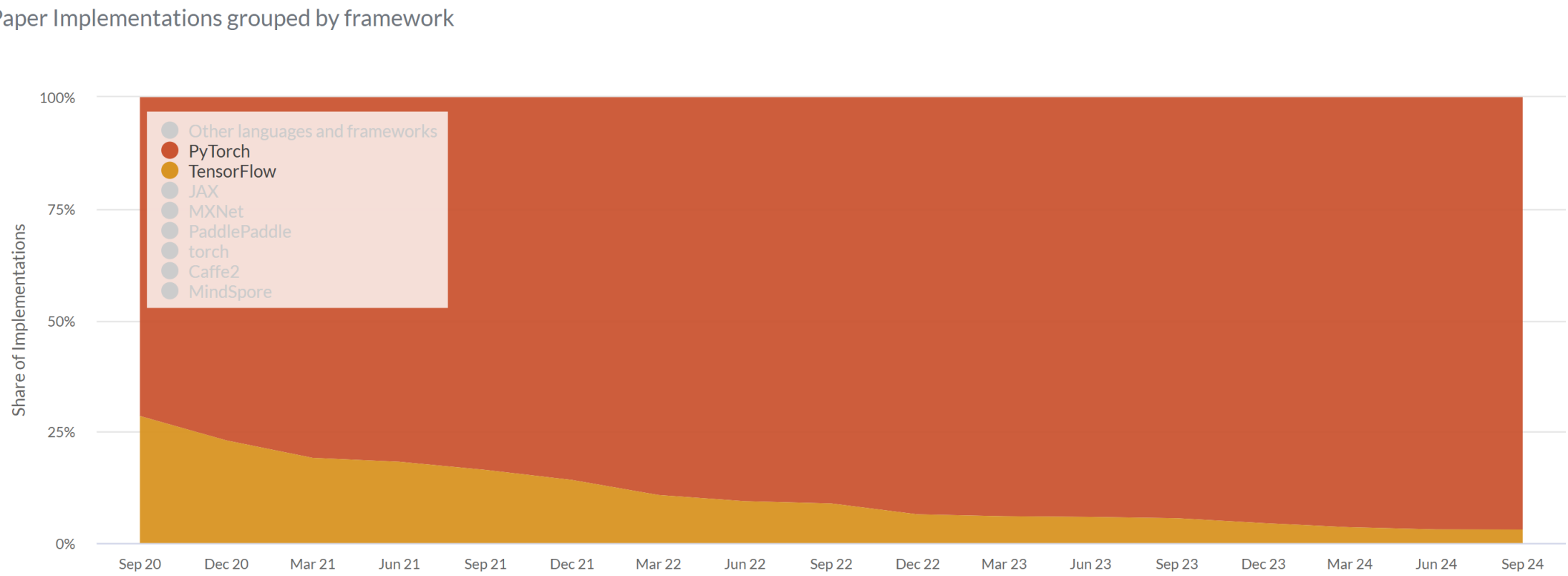
the more certain I am that I know nothing.”

— Voltaire

Frameworks for implementing deep learning

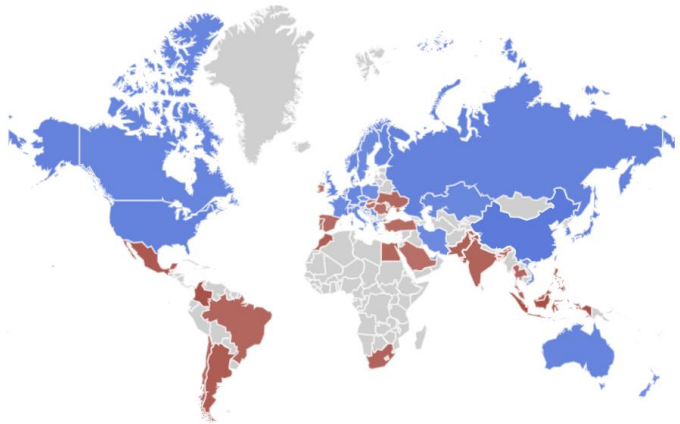
Tensorflow	Pytorch
Developed by Google	Developed by Meta
Visualization with Tensorboard	Efficient memory usage
Simple high-level API	Python low-level coding
Mature library	Younger library, but increasing in popularity

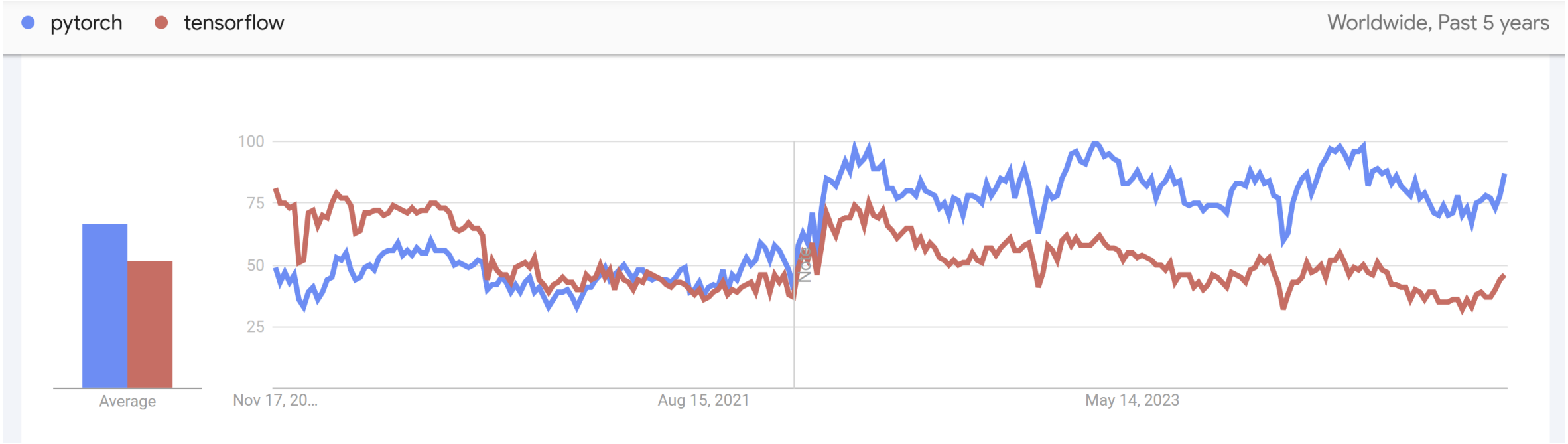
Tensorflow vs. Pytorch



https://paperswithcode.com/trends

Tensorflow vs. Pytorch





https://trends.google.com/trends/explore?date=today%205-y&q=pytorch,tensorflow&hl=en

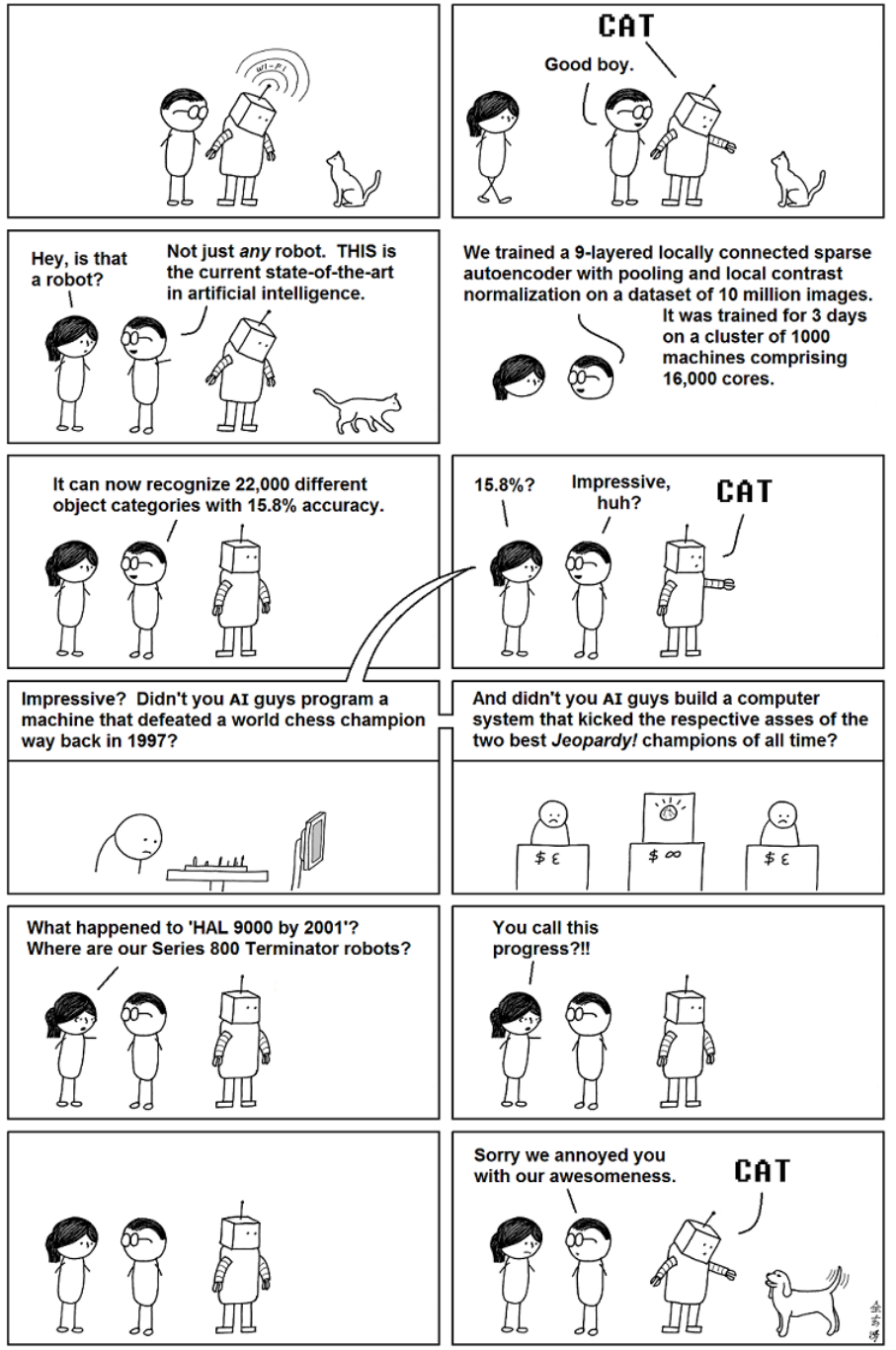
Working with Tensorflow/Keras under R

- Installation:
  - Install Rtools last version from https://cran.r-project.org/bin/windows/Rtools/
  - Install.packages("dplyr")
  - Library(devtools)
  - Install_github("rstudio/reticulate")
  - Install.packages("keras") for Tensorflow and Keras

Image processing. Convolutional neural networks

Convolutional neural networks (CNN)

https://www.quora.com/What-are-some-good-data-science-statistics-machine-learning-jokes

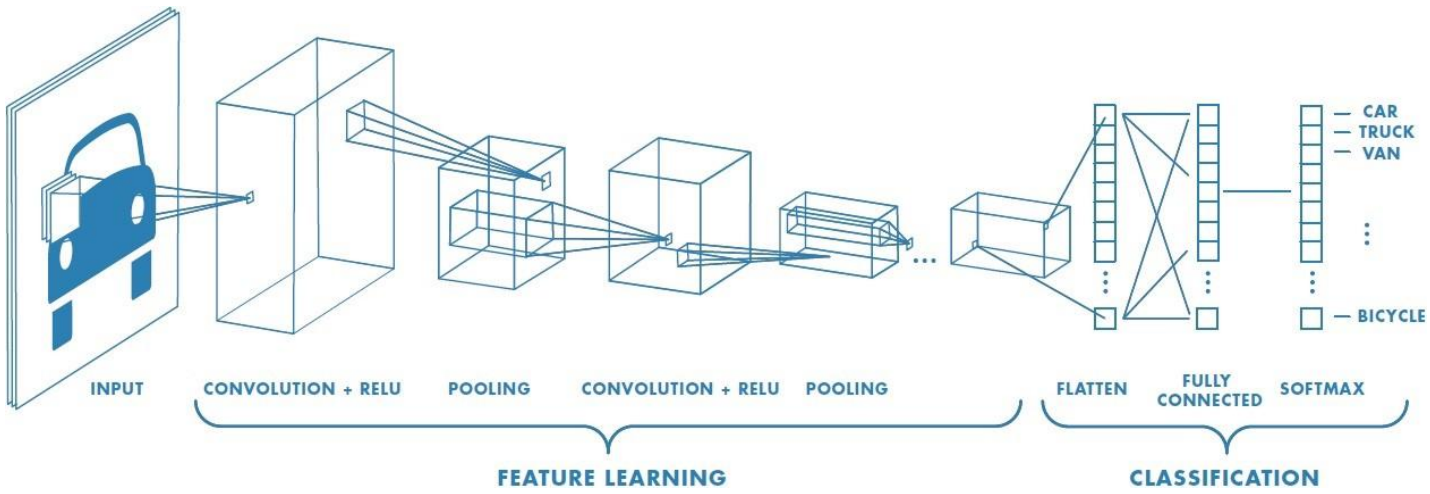


CNN

- Automatic feature learning: from low-level to high-level
- Important applications in computer vision
  - Classification – There is a building in this image
  - Semantic Segmentation – These are the pixels of the building
  - Object Detection – There are buildings in this image
  - Instance Segmentation – There are several instances of buildings in this image and these are the pixels of each

Architecture

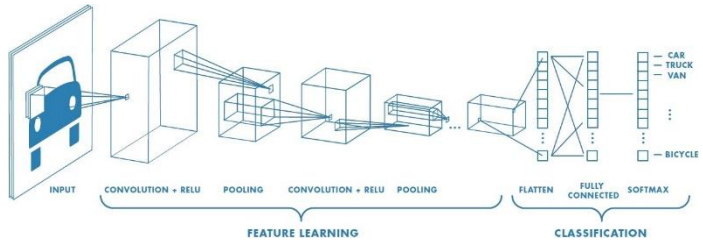
- Layers
  - Feature learning
    - Convolution
    - ReLU
    - Pooling
  - Classification
    - Fully connected

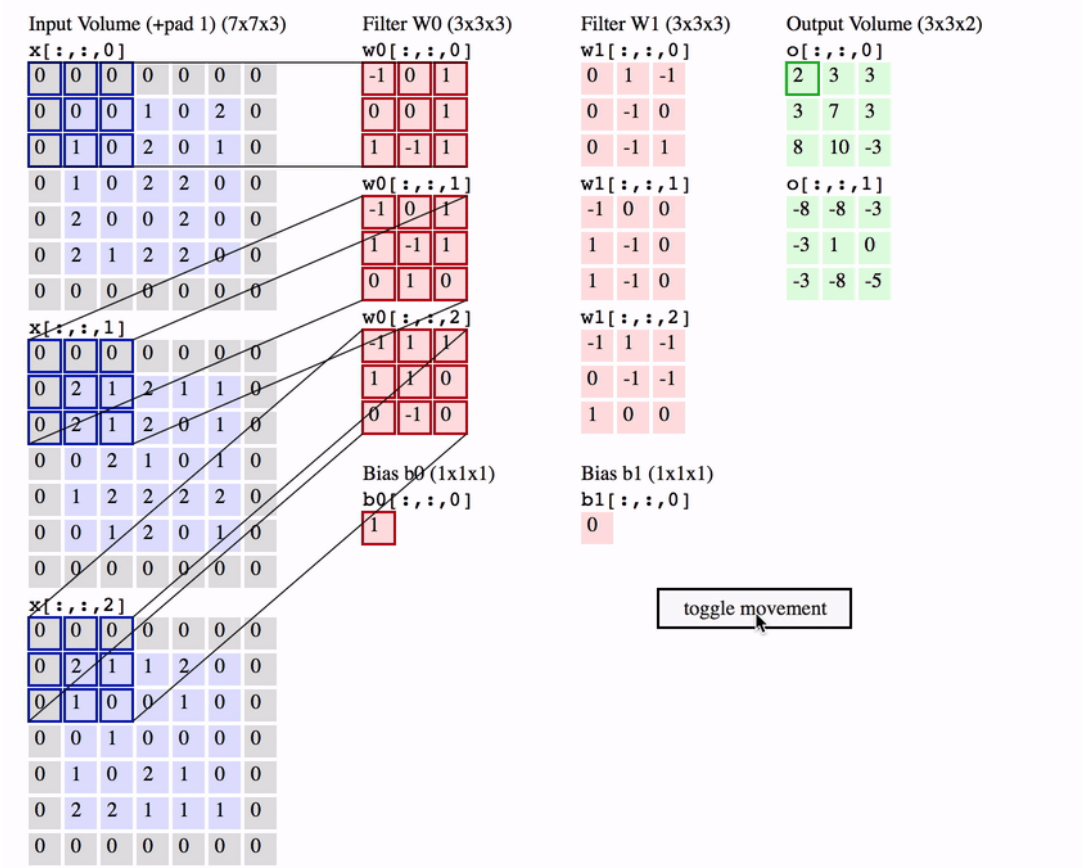


https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Convolution

- Input
  - Volume of Width x Height x Depth
- Kernel (or filter) - a shared set of weights
- Forward pass – convolution between filter and input volume
- Result: KD activation maps of size KSxKS stacked in a new volume
- Four hyperparameters:
  - Kernel size (KS)
  - Kernel depth (number of filters) (KD)
  - Stride
  - Zero-padding

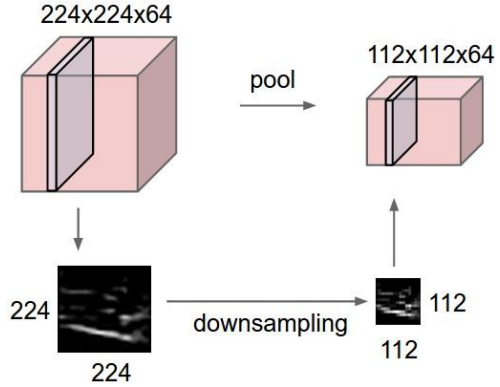


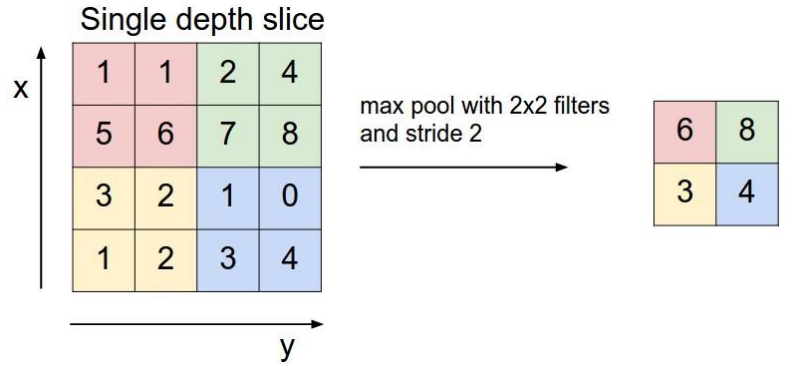


http://cs231n.github.io/convolutional-networks/
https://www.youtube.com/watch?v=AQirPKrAyDg

ReLU & Pooling

- Rectified Linear Unit – transfer layer for nonlinearity
- (Max) Pooling – downsamples the volume by taking the (maximum) value
  - Window size
  - Stride

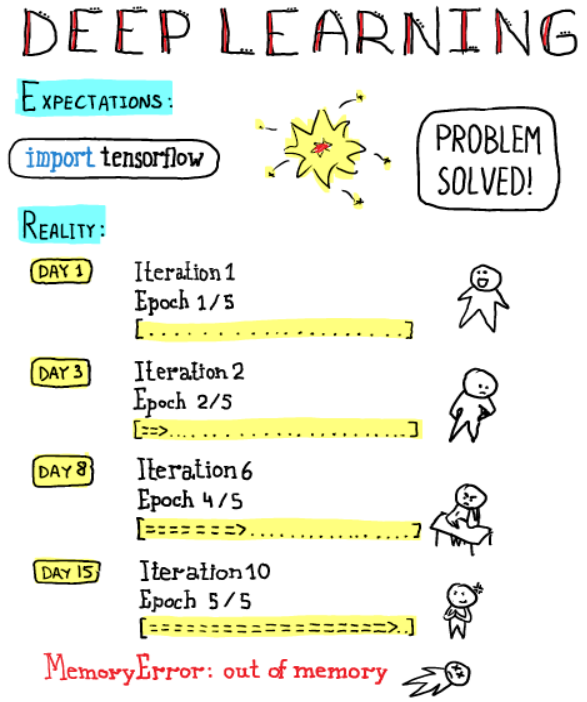




http://cs231n.github.io/convolutional-networks/
https://www.youtube.com/watch?v=AQirPKrAyDg

The practice

- Choice of the proper architecture
- Parameter dependence on the problem
- Large runtime
- Great computing power required
- Small sample size in real-world
- Overfitting
- Difficult model interpretation
- Not plug & play!



https://medium.com/@elmira/deep-learning-frustration-4c26d69609f2

Machine Learning Memes for Convolutional Teens

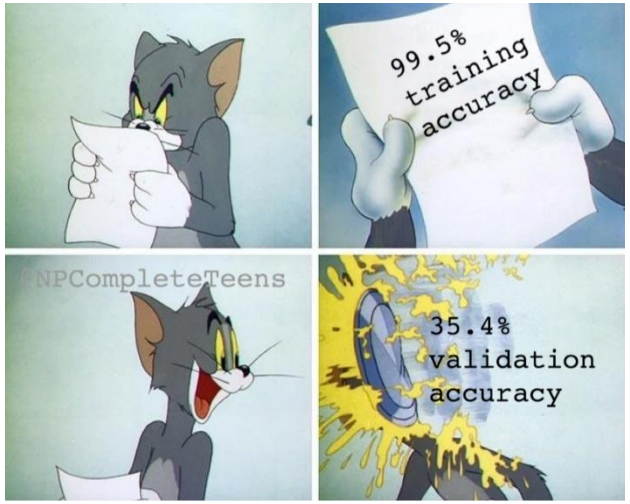
Parametrization

- Convolutional kernels
  - Sizes, depths, strides
- Pooling layers
  - Window sizes, strides
- Dropout rates
- Batch size
- Number of epochs
- Initial weights
- Optimizers
- Learning rate
- Activation functions
- Number of units in the fully connected layers
- Topology

- Tuning
  - Manual
  - Automatic

Overfitting

- When the model is too complex for the data
- Working with real-world problems
  - Small sample size
- Ways to combat:
  - Data augmentation
    - Flip, rotate, scale, crop, translate, Gaussian noise
    - Generative adversarial networks (GAN): one network generates, the other evaluates
  - Dropout layer
  - Regularization
    - Weight penalty (decay) L1 and L2
  - Early stopping
    - Use of checkpoints to save the model each epoch
    - Pick the best candidate from validation result after last epoch



Machine Learning Memes for Convolutional Teens

Image classification

MNIST digits classification in Keras/TF under R

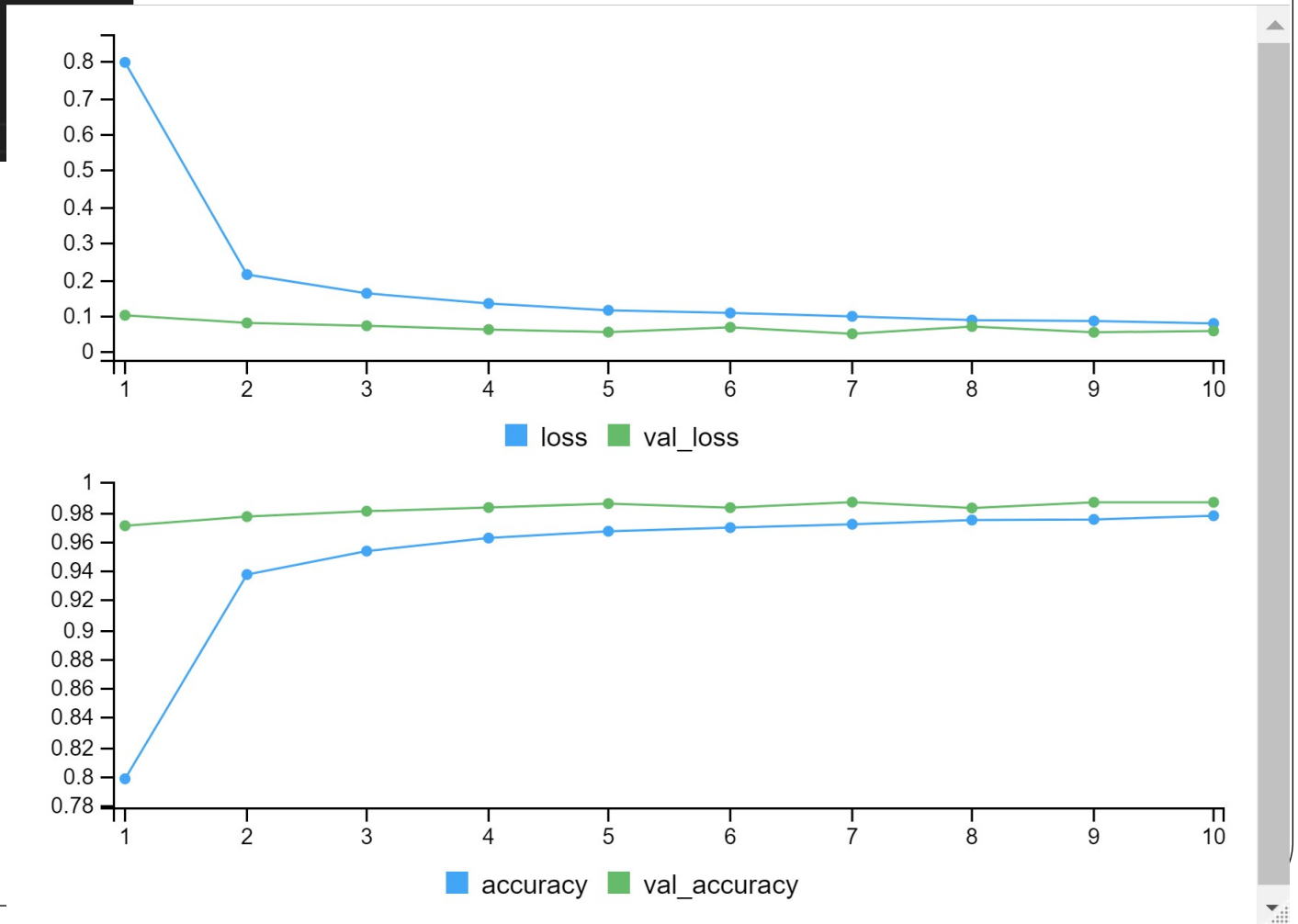
- A 2-layer CNN for MNIST digit recognition

```r
library("keras")

# load MNIST data set
mnist   <- dataset_mnist()
c(x_train, y_train) %<-% mnist$train
c(x_test, y_test)   %<-% mnist$test
rbind(dim(x_train), dim(x_test))

# transform to format for CNN (length, width, number of color channels)
x_train_original <- x_train
x_test_original  <- x_test
x_train          <- array_reshape(x_train, c(nrow(x_train), 28, 28, 1))
x_test           <- array_reshape(x_test, c(nrow(x_test), 28, 28, 1))
input_shape      <- c(28, 28, 1)

# transform classes codified by 0-9 digits to one-hot binary encoding
y_train_original <- y_train
y_test_original  <- y_test
y_train          <- to_categorical(y_train, 10)
y_test           <- to_categorical(y_test, 10)
```

```r
# construct CNN architecture
# 1. Feature Learning
# two convolutionary layers, one with 8 filters of 3 x 3, the second with 16 filter of size 5 x 5
# RELU nonlinearity
# Max Pooling of size 2 x 2 and stride 2
# Dropout of 0.25
# 2. Classification
# one fully connected layer with 100 units and ReLU activation
# Dropout of 0.5
# Last fully connected layer to the 10 target classes with softmax activation

model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 8, kernel_size = c(3,3), activation = 'relu', padding="same",
                input_shape = input_shape) %>%
  layer_conv_2d(filters = 16, kernel_size = c(5,5), activation = 'relu', padding="same") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10, activation = 'softmax')

summary(model)

# compile model with categorical cross-entropy loss, Adam optimizer and accuracy as metric
model %>% compile(
  loss = 'categorical_crossentropy',  # for multi-class classification
  optimizer = 'adam',                 # optimizer
  metrics = c('accuracy')             # accuracy as model performance
)
```

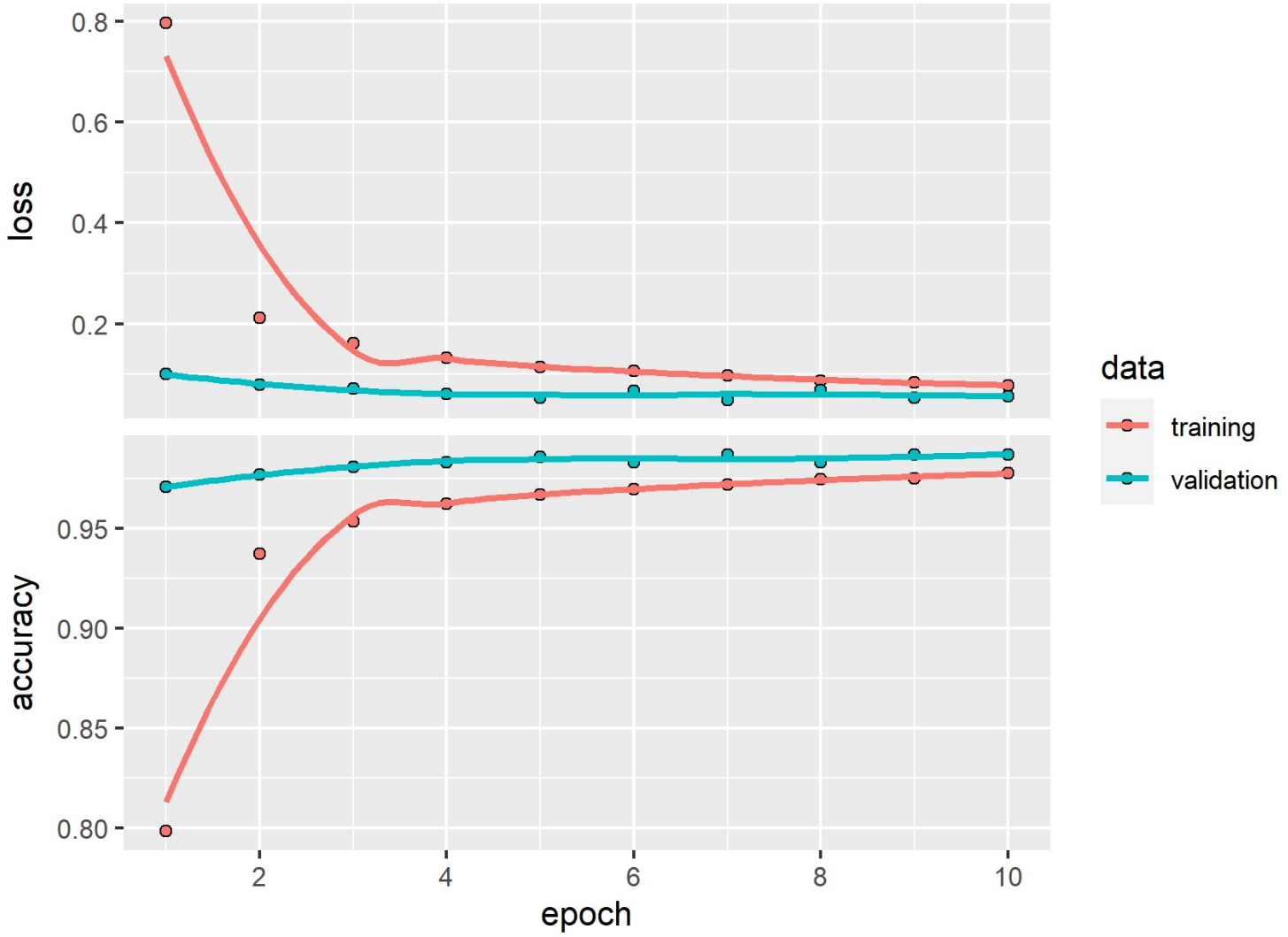
Real-time training history

```
54  # train model
55  history <- model %>% fit(
56    x_train, y_train,
57    epochs = 10,           # number of epochs (10 complete passes of the training set)
58    batch_size = 25,       # size of batch in one data pass
59    validation_split = 0.25  # training 75% - validation 25%
60  )
61
62  # plot loss and accuracy along training
63  plot(history)
```



Plot training history

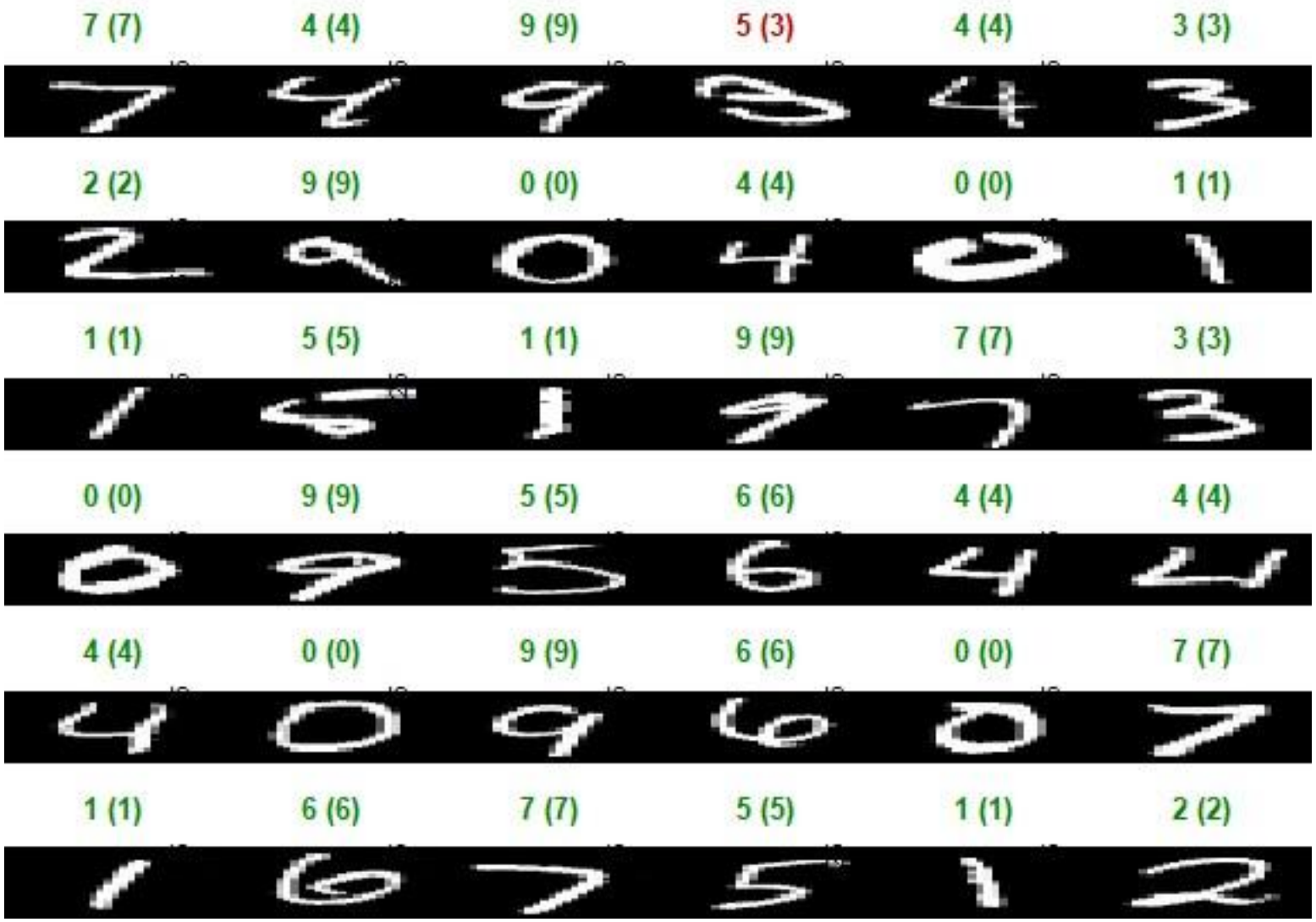
```
62  # plot loss and accuracy along training
63  plot(history)
64
65  # evaluate model on test
66  model %>% evaluate(x_test, y_test)
67
68  # CNN model predictions as class probabilities
69  # label with class of highest probability
70
71  y_test_predicted_val <- model %>% predict(x_test)
72  y_test_predicted_ten <- y_test_predicted_val %>% k_argmax()
73  y_test_predicted <- as.numeric(y_test_predicted_ten)
```



Visualize test predictions

```
77  par(mfcol=c(6,6))
78  par(mar=c(0, 0, 3, 0), xaxs='i', yaxs='i')
79  for (idx in 1:36) {
80    im <- x_test_original[idx,,]
81    im <- t(apply(im, 2, rev))
82    if (y_test_predicted[idx] == y_test_original[idx]) {
83      color <- '#008800'
84    } else {
85      color <- '#bb0000'
86    }
87    image(1:28, 1:28, im, col=gray((0:255)/255),
88          xaxt='n', main=paste0(y_test_predicted[idx], " (", y_test_original[idx], ")"),
89          col.main=color)
90  }
```

Accuracy:
98.79%



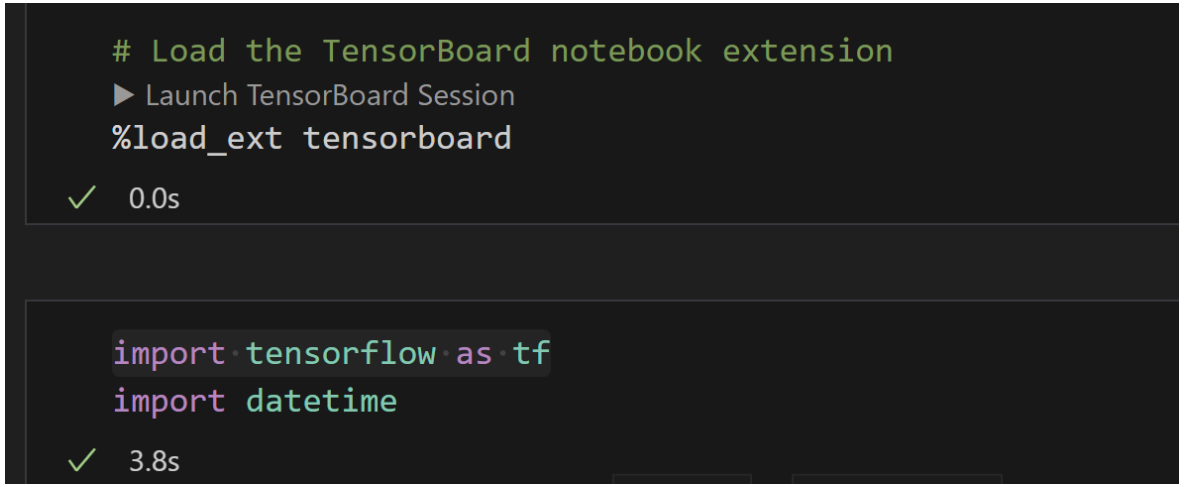
CNN for CIFAR-10 in Tensorflow/Keras

- https://www.cs.toronto.edu/~kriz/cifar.html
- A data set of 60000 images, size 32x32
- Real-world pictures
- 10 classes, 6000 images per class
- 50000 training data, 10000 test data

TensorBoard

- Real time and storable measurements/visualizations during the machine learning flow

- !Delete contents of "\AppData\Local\Temp\.tensorboard-info" to be able to launch new TB session!
- See TB directly in notebook



```
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

# load CIFAR10 data set

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# normalize pixel values
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Plot some samples from the training data

plt.rcParams.update({'text.color': "black",
                     'axes.labelcolor': "white"})

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

```python
# The CNN model

# The convolutional layers for feature learning
# input_shape for pictures of the form [length, width, number of channels]]
# CNN architecture: 3 convolutional layers, depths 32, 64, 64 and sizes 3x3, max-pooling with size 2x2
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Dropout(0.5))

# The flatten layer collapses the spatial dimensions of the input into 1D
model.add(layers.Flatten())

# The dense layers for classification: one fully connected layer with 64 neurons and the final dense layer with 10 outputs for the classes
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation = 'softmax'))

# Summary of the CNN architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

```
▶ Launch TensorBoard Session
%tensorboard --logdir logs
```

[6] ✓ 3.6s

```
# view training/validation history in real time
from tensorboard import notebook
notebook.list()
```

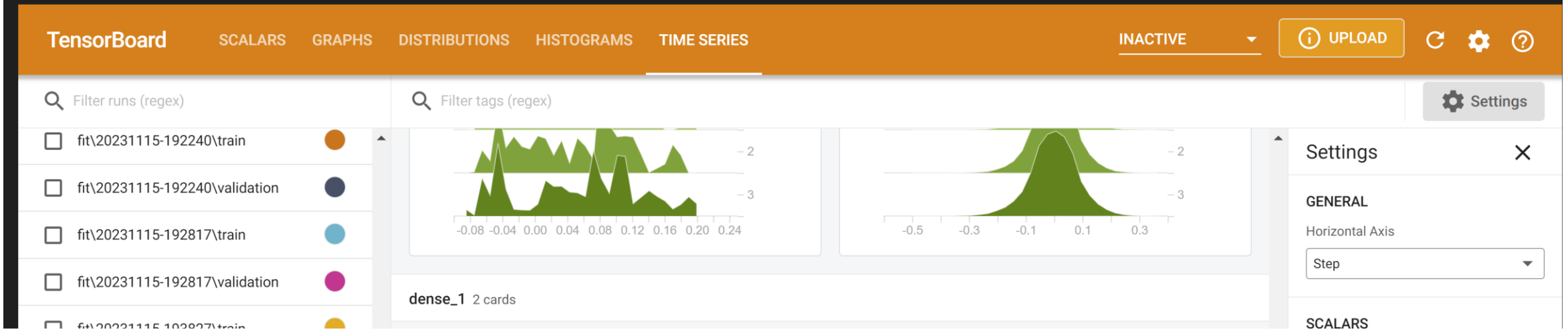
✓ 0.0s Pythor

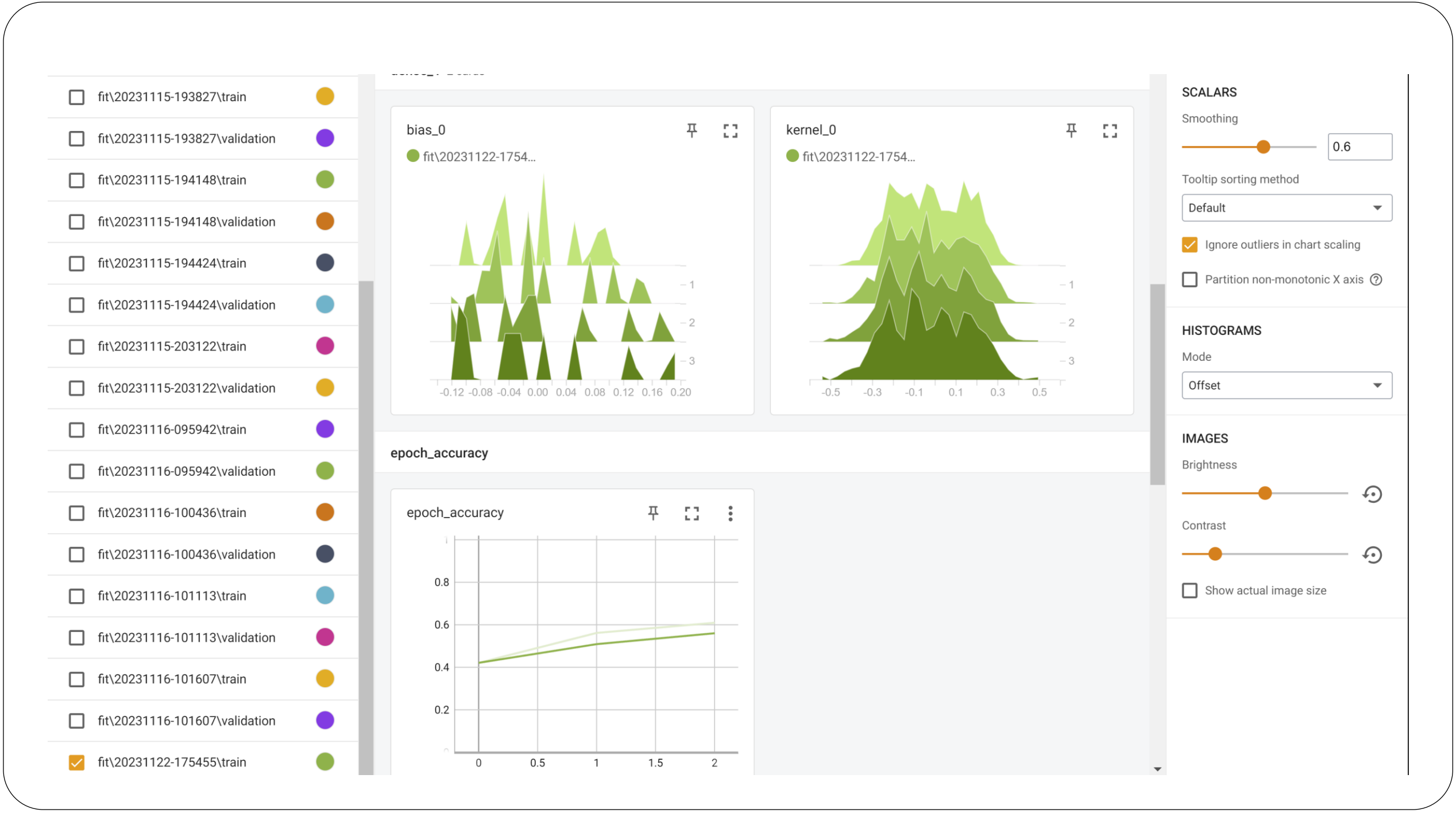
```
Known TensorBoard instances:
  - port 6006: logdir logs (started 0:00:06 ago; pid 31792)
```

```
notebook.display(port=6006, height=1000)
```

✓ 0.0s Pythor

```
Selecting TensorBoard with logdir logs (started 0:00:12 ago; port 6006, pid 31792).
```




fit\20231115-193827\train
fit\20231115-193827\validation
fit\20231115-194148\train
fit\20231115-194148\validation
fit\20231115-194424\train
fit\20231115-194424\validation
fit\20231115-203122\train
fit\20231115-203122\validation
fit\20231116-095942\train
fit\20231116-095942\validation
fit\20231116-100436\train
fit\20231116-100436\validation
fit\20231116-101113\train
fit\20231116-101113\validation
fit\20231116-101607\train
fit\20231116-101607\validation
fit\20231122-175455\train
bias_0
fit\20231122-1754...
-0.12 -0.08 -0.04 0.00 0.04 0.08 0.12 0.16 0.20
kernel_0
fit\20231122-1754...
-0.5 -0.3 -0.1 0.1 0.3 0.5
epoch_accuracy
epoch_accuracy
0.8
0.6
0.4
0.2
0
0.5
1
1.5
2
SCALARS
Smoothing
0.6
Tooltip sorting method
Default
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS
Mode
Offset
IMAGES
Brightness
Contrast
Show actual image size

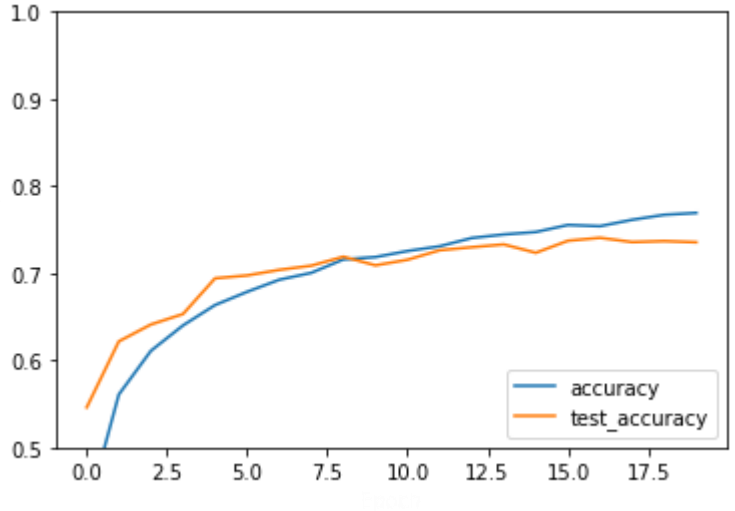
```
# compile model
# optimizer: Adam
# loss: SparseCategoricalCrossentropy - when there are two or more label classes and labels are provided as integers
# metric - accuracy
# default batch_size = 32
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

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)

# Train model with 20 epochs
# Get loss/accuracy also on the test set
history = model.fit(train_images, train_labels, epochs=20,
                    validation_data=(test_images, test_labels),
                    callbacks=[tensorboard_callback])
```

```
Epoch 1/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.5614 - accuracy: 0.4249 - val_loss: 1.2551 - val_accuracy: 0.5460
Epoch 2/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2293 - accuracy: 0.5608 - val_loss: 1.0759 - val_accuracy: 0.6217
Epoch 3/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.0935 - accuracy: 0.6109 - val_loss: 1.0063 - val_accuracy: 0.6411
Epoch 4/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.0132 - accuracy: 0.6400 - val_loss: 0.9667 - val_accuracy: 0.6531
Epoch 5/20
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9518 - accuracy: 0.6634 - val_loss: 0.8780 - val_accuracy: 0.6941
Epoch 6/20
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9098 - accuracy: 0.6786 - val_loss: 0.8656 - val_accuracy: 0.6974
Epoch 7/20
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8693 - accuracy: 0.6924 - val_loss: 0.8335 - val_accuracy: 0.7038
Epoch 8/20
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8440 - accuracy: 0.7004 - val_loss: 0.8259 - val_accuracy: 0.7086
Epoch 9/20
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8105 - accuracy: 0.7158 - val_loss: 0.8046 - val_accuracy: 0.7189
Epoch 10/20
1563/1563 [==============================] - 6s 4ms/step - loss: 0.7932 - accuracy: 0.7185 - val_loss: 0.8389 - val_accuracy: 0.7090
Epoch 11/20
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7759 - accuracy: 0.7254 - val_loss: 0.8048 - val_accuracy: 0.7154
Epoch 12/20
1563/1563 [==============================] - 10s 7ms/step - loss: 0.7575 - accuracy: 0.7309 - val_loss: 0.7812 - val_accuracy: 0.7263
Epoch 13/20
...
Epoch 19/20
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6615 - accuracy: 0.7670 - val_loss: 0.7557 - val_accuracy: 0.7368
Epoch 20/20
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6493 - accuracy: 0.7691 - val_loss: 0.7725 - val_accuracy: 0.7355
```

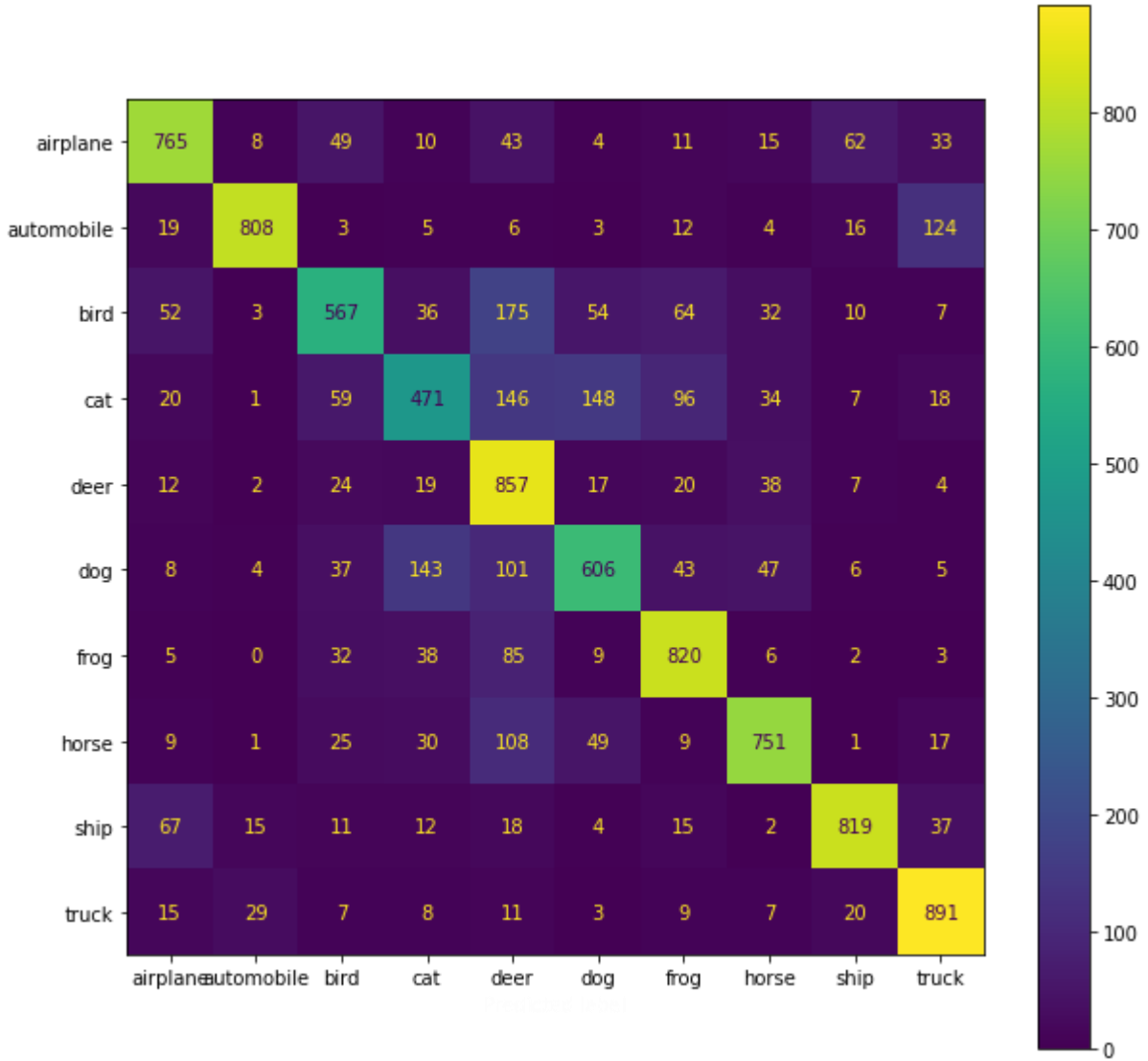
```
# Model accuracy

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'test_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

y_pred = model.predict(test_images)
predicted_categories = tf.argmax(y_pred, axis=1)

cm = confusion_matrix(test_labels, predicted_categories)

cmp = ConfusionMatrixDisplay(cm, display_labels = class_names)
fig, ax = plt.subplots(figsize=(10,10))
cmp.plot(ax = ax)
```

```
from sklearn.metrics import classification_report

print(classification_report(test_labels, predicted_categories, labels = list(range(10)), target_names= class_names))
```

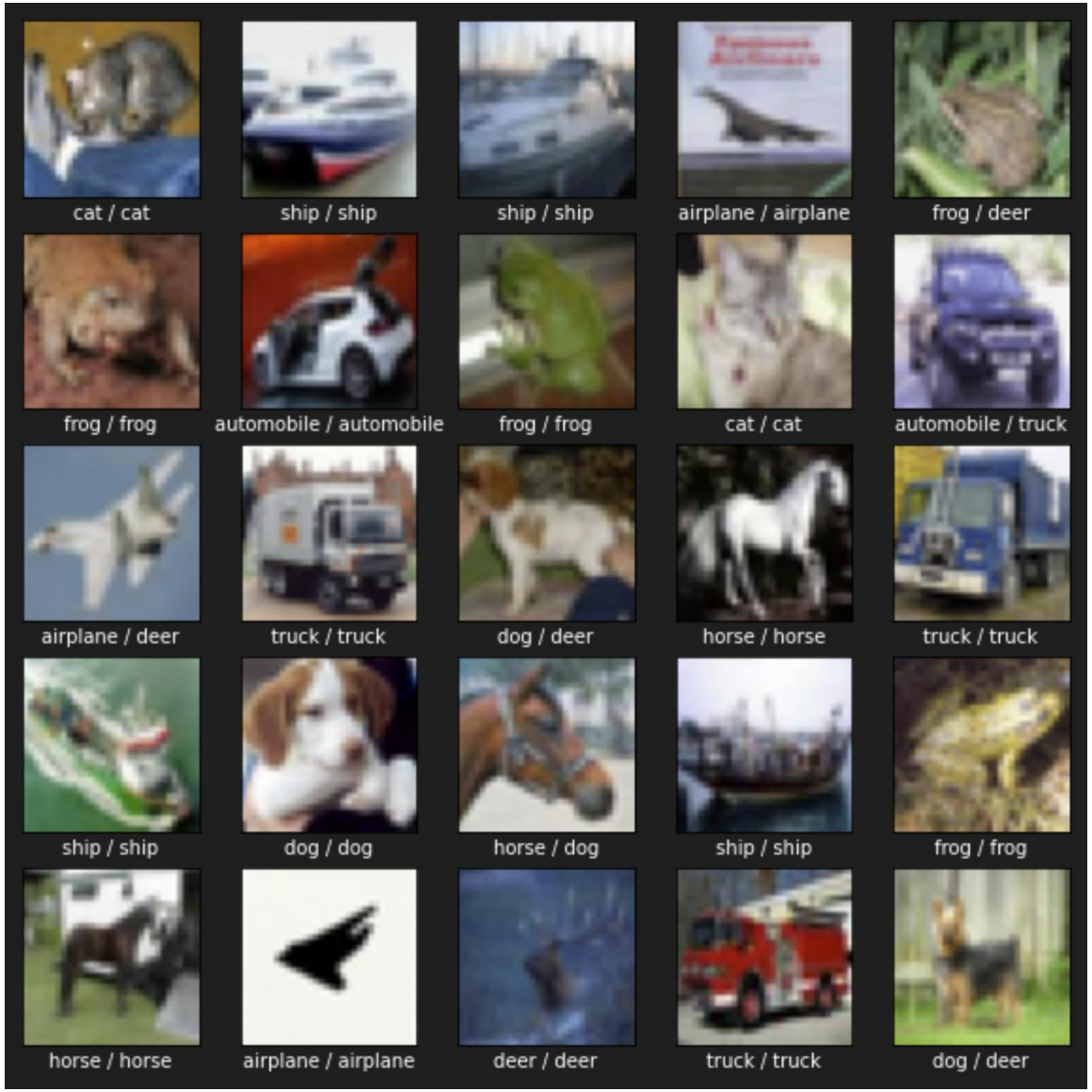


	precision	recall	f1-score	support
airplane	0.79	0.77	0.78	1000
automobile	0.93	0.81	0.86	1000
bird	0.70	0.57	0.63	1000
cat	0.61	0.47	0.53	1000
deer	0.55	0.86	0.67	1000
dog	0.68	0.61	0.64	1000
frog	0.75	0.82	0.78	1000
horse	0.80	0.75	0.78	1000
ship	0.86	0.82	0.84	1000
truck	0.78	0.89	0.83	1000
accuracy			0.74	10000
macro avg	0.74	0.74	0.73	10000
weighted avg	0.74	0.74	0.73	10000



Prediction on test

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i])

    # Model was trained on batches,
    # hence the format expected for prediction is a tensor of the form
    # [batch_size, length, width, number of channels]
    # A picture means a batch_size = 1 => add a dimension to x
    x = np.expand_dims(test_images[i], axis=0)

    predict_x = model.predict(x)

    # The most probable class is taken from the resulting class probabilities
    classes_x=np.argmax(predict_x,axis=-1)

    # label showing real class/predicted class
    plt.xlabel(class_names[test_labels[i][0]] + ' / ' + class_names[classes_x[0]])
plt.show()
```


cat / cat
ship / ship
ship / ship
airplane / airplane
frog / deer
frog / frog
automobile / automobile
frog / frog
cat / cat
automobile / truck
airplane / deer
truck / truck
dog / deer
horse / horse
truck / truck
ship / ship
dog / dog
horse / dog
ship / ship
frog / frog
horse / horse
airplane / airplane
deer / deer
truck / truck
dog / deer

CNN visualization

- Visual tutorial on CNN functioning
  - https://poloclub.github.io/cnn-explainer/
- Tool to visualize personal models
  - https://github.com/zetane/viewer

DeepExplainer

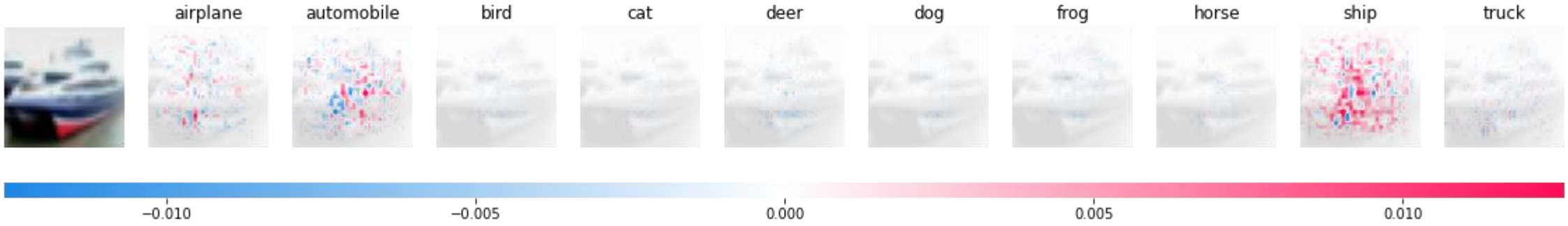
- Determines SHAP values by an improved version of the DeepLIFT algorithm
  - DeepLIFT backpropagates the neuron contributions to the input features
  - Contribution (positive or negative) = difference between neuron activation and reference activation

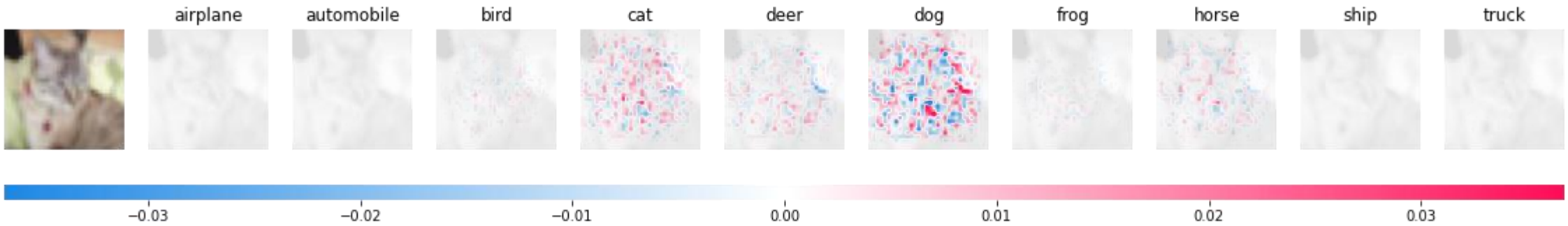
- The expectations are estimated on the base of selected background examples
  - 100 samples represent a good estimate; 1000 a very good one, but costly

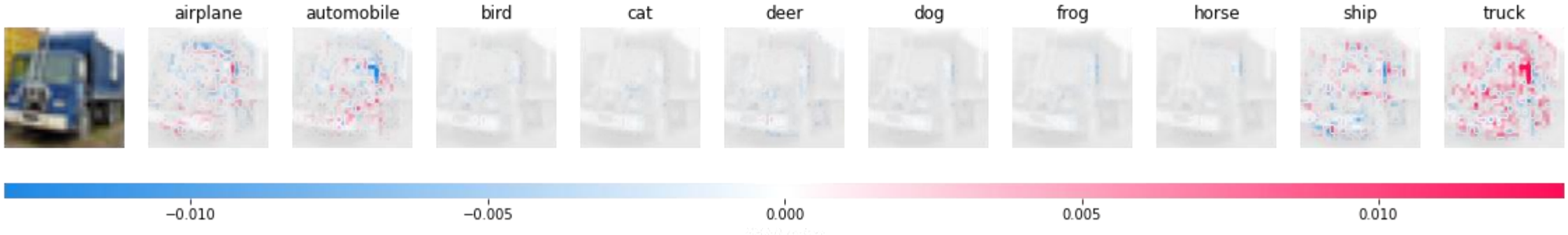
- Interpretation:
  - Red pixels (positive) denote the features whose presence is significant for the class
  - Blue pixels (negative) show the features whose absence is significant for the class

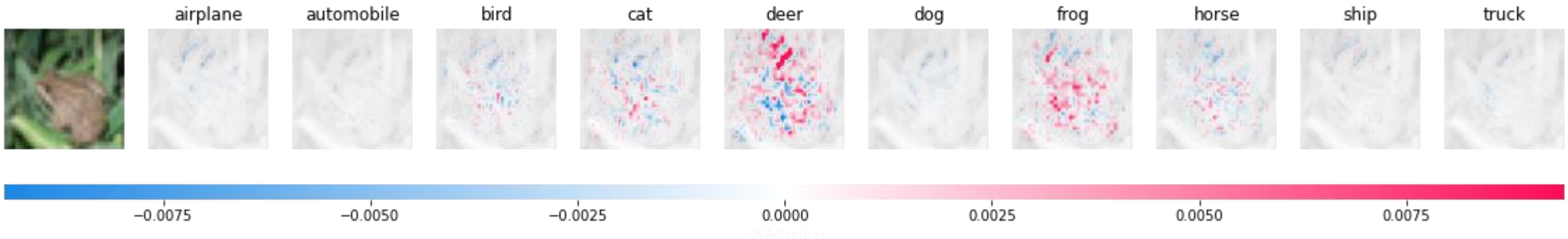
```python
# DeepExplainer SHAP

import shap
import numpy as np

# select a background set of 100 examples to get the expectation
examples = train_images[np.random.choice(train_images.shape[0], 100, replace=False)]
explainer = shap.DeepExplainer(model, examples, class_names)
```

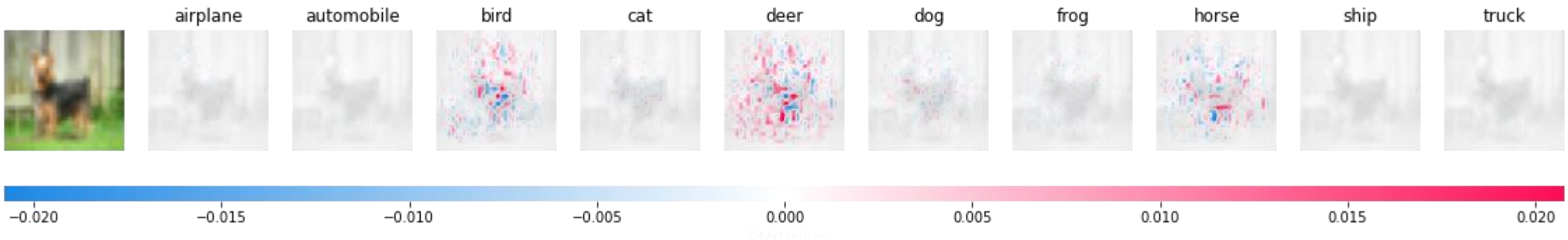
```python
# explain model predictions on several test images
# red pixels increase the model output (presence of that area favors output)
# blue pixels decrease the output (absence of that area favors output)
for i in range(25):
    print(class_names[test_labels[i][0]])
    shap_values = explainer.shap_values(test_images[i:i+1])
    shap.image_plot(shap_values, test_images[i:i+1], labels=class_names)
```


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
−0.010
−0.005
0.000
0.005
0.010


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
−0.010
−0.005
0.000
0.005
0.010


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
−0.0075
−0.0050
−0.0025
0.0000
0.0025
0.0050
0.0075


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
−0.020
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020

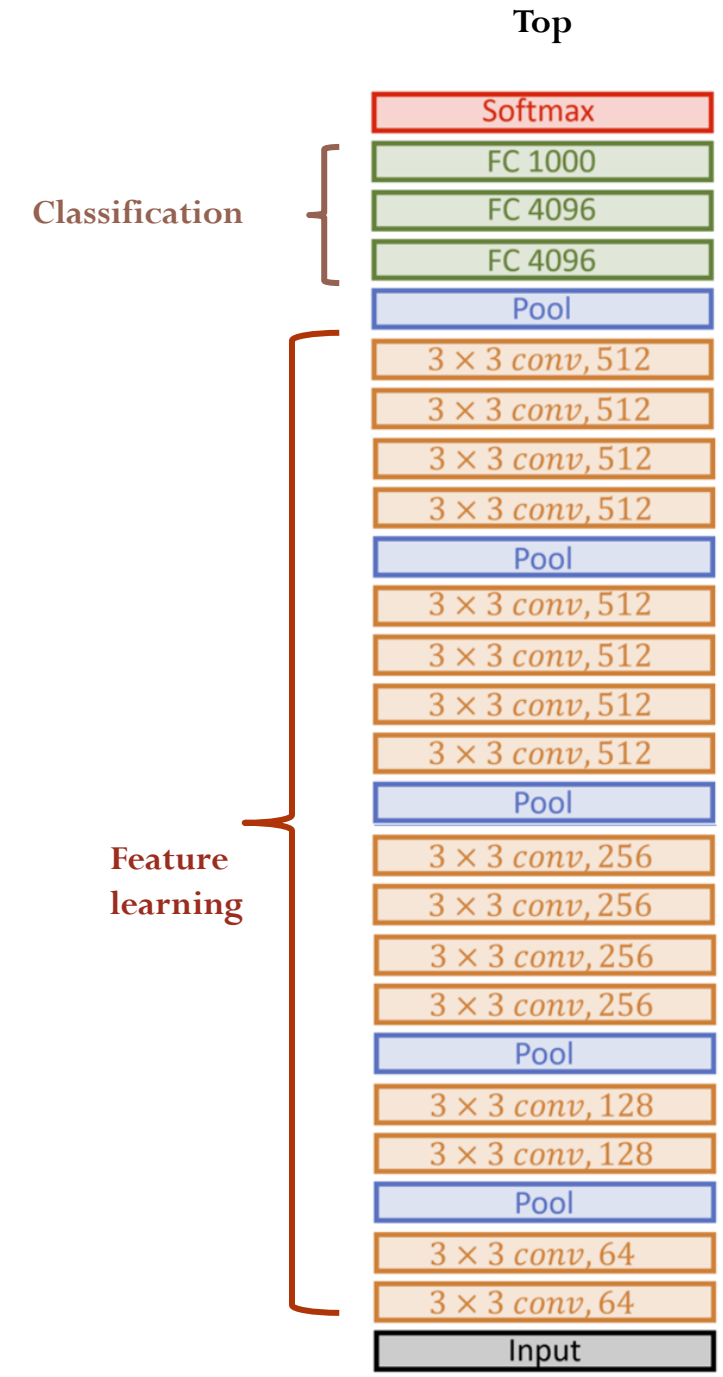
Transfer learning

- Deep learning needs
  - Big data
  - Big computing resources
- Take the parameters from an already trained network on a large data set
  - Data: ImageNet, CIFAR
  - Pre-trained models: VGG, Inception, AlexNet, ResNet
  - Initial layers have learnt general features
- Train the final layers for the problem at hand
  - Learn the specific features of the current data
- Hence
  - Resolve problem with scarce available data
  - Less parameters to train

VGG-19

- Common choice for transfer learning
- 19 layers (16 convolutional + 3 feed-forward)
- Top layers are the last layers added to the network

```
# Transfer Learning with VGG19

import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from keras.applications.vgg19 import VGG19
import matplotlib.pyplot as plt
import numpy as np

# load CIFAR10 data set

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# normalize pixel values
train_images, test_images = train_images / 255.0, test_images / 255.0
```

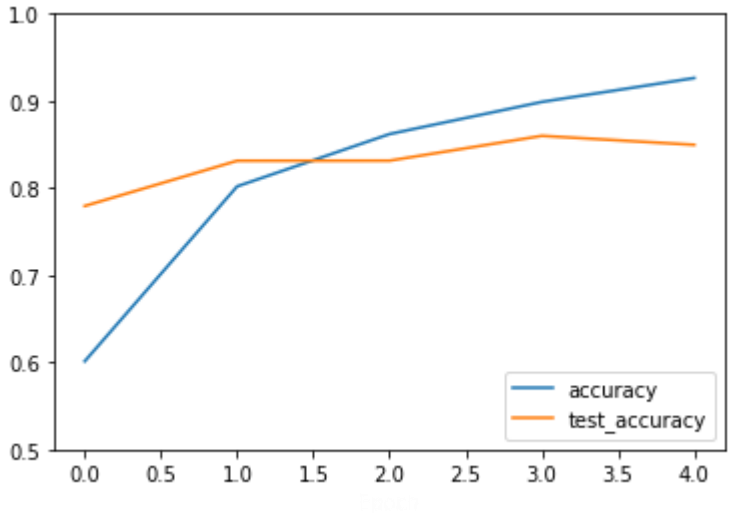


```python
# Base: VGG19 pretrained on the ImageNet data set -> set of weights of the pretraining
# include_top: (not) include the 3 fully-connected layers (those for classification) at the top of the networ
# i.e., take just the feature extraction convolutional layers

TLmodel = VGG19(include_top = False, weights = 'imagenet', classes= train_labels)

# keep all layers as trained on ImageNet (i.e. frozen), except the last given number
# i.e., the first layer have learned the basic features
# the last ones will learn the characteristics specific to the current CIFAR problem

for layer in TLmodel.layers[:-6]: # just the last given number of layers are trainable
    layer.trainable = False

# see all trainable layers
for layer in TLmodel.layers:
    print(layer, layer.trainable)

# create a sequential model that begins with the pretrained VGG + the last trainable layers
model = models.Sequential()

# increase image size to be better fitted with the VGG19
model.add(layers.UpSampling2D())
model.add(layers.UpSampling2D())

# add the VGG19
model.add(TLmodel)

# The flatten layer collapses the spatial dimensions of the input into 1D, and then fully connected follows
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.7))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.3))

# Add the final dense layer with the output for the 10 classes
model.add(layers.Dense(10, activation='softmax'))
```

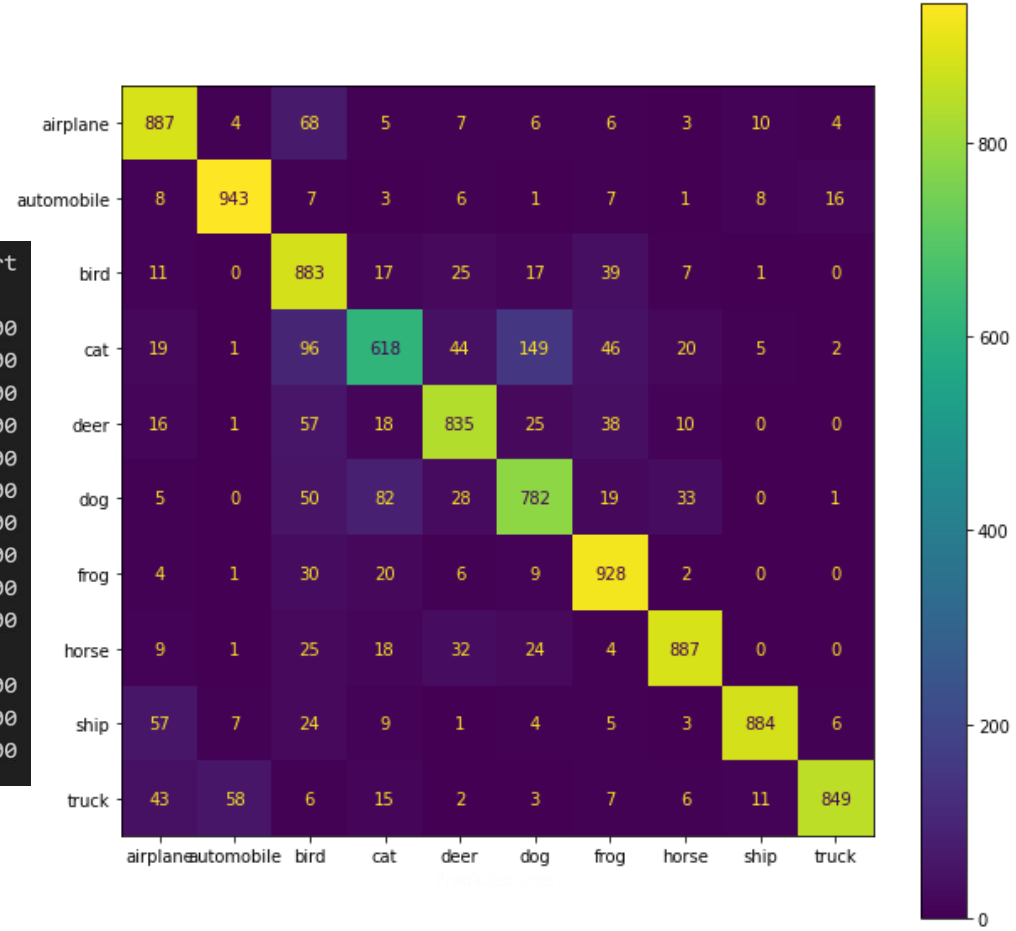
```
<keras.engine.input_layer.InputLayer object at 0x0000023496ADD730> False
<keras.layers.convolutional.Conv2D object at 0x0000023496ADD3D0> False
<keras.layers.convolutional.Conv2D object at 0x0000023496ADD880> False
<keras.layers.pooling.MaxPooling2D object at 0x000002348E35AB80> False
<keras.layers.convolutional.Conv2D object at 0x000002348E321FD0> False
<keras.layers.convolutional.Conv2D object at 0x0000023496AE1640> False
<keras.layers.pooling.MaxPooling2D object at 0x0000023496AB01C0> False
<keras.layers.convolutional.Conv2D object at 0x0000023496AF0340> False
<keras.layers.convolutional.Conv2D object at 0x000002348E5529A0> False
<keras.layers.convolutional.Conv2D object at 0x0000023496AF45B0> False
<keras.layers.convolutional.Conv2D object at 0x0000023497AD2BB0> False
<keras.layers.pooling.MaxPooling2D object at 0x0000023497AD2310> False
<keras.layers.convolutional.Conv2D object at 0x0000023497AD2520> False
<keras.layers.convolutional.Conv2D object at 0x0000023496AFDD90> False
<keras.layers.convolutional.Conv2D object at 0x0000023497ADED00> False
<keras.layers.convolutional.Conv2D object at 0x0000023497ADE0D0> False
<keras.layers.pooling.MaxPooling2D object at 0x0000023497AE60D0> True
<keras.layers.convolutional.Conv2D object at 0x0000023496AF4F70> True
<keras.layers.convolutional.Conv2D object at 0x0000023497ADEBB0> True
<keras.layers.convolutional.Conv2D object at 0x0000023497ADEF70> True
<keras.layers.convolutional.Conv2D object at 0x0000023497AF2AC0> True
<keras.layers.pooling.MaxPooling2D object at 0x0000023497AFE9D0> True
```

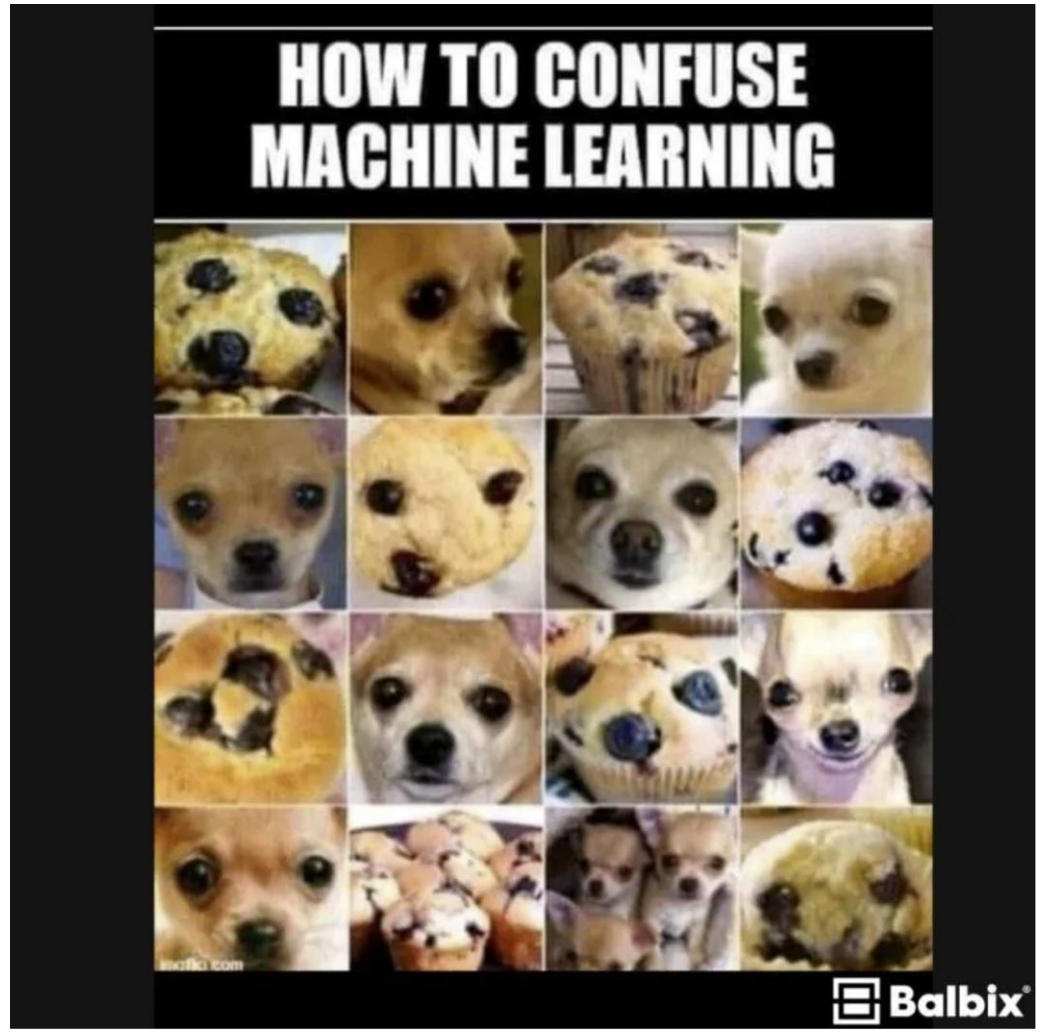
```python
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/5
1563/1563 [==============================] - 115s 73ms/step - loss: 1.1800 - accuracy: 0.6012 - val_loss: 0.6623 - val_accuracy: 0.7794
Epoch 2/5
1563/1563 [==============================] - 147s 94ms/step - loss: 0.6321 - accuracy: 0.8019 - val_loss: 0.5281 - val_accuracy: 0.8313
Epoch 3/5
1563/1563 [==============================] - 176s 113ms/step - loss: 0.4448 - accuracy: 0.8619 - val_loss: 0.5440 - val_accuracy: 0.8314
Epoch 4/5
1563/1563 [==============================] - 146s 94ms/step - loss: 0.3236 - accuracy: 0.8988 - val_loss: 0.4772 - val_accuracy: 0.8599
Epoch 5/5
1563/1563 [==============================] - 158s 101ms/step - loss: 0.2378 - accuracy: 0.9262 - val_loss: 0.5607 - val_accuracy: 0.8496
```



	precision	recall	f1-score	support
airplane	0.91	0.81	0.86	1000
automobile	0.87	0.94	0.90	1000
bird	0.88	0.73	0.80	1000
cat	0.67	0.70	0.68	1000
deer	0.77	0.84	0.80	1000
dog	0.77	0.72	0.75	1000
frog	0.80	0.92	0.86	1000
horse	0.89	0.85	0.87	1000
ship	0.89	0.93	0.91	1000
truck	0.90	0.89	0.90	1000
accuracy			0.83	10000
macro avg	0.84	0.83	0.83	10000
weighted avg	0.84	0.83	0.83	10000

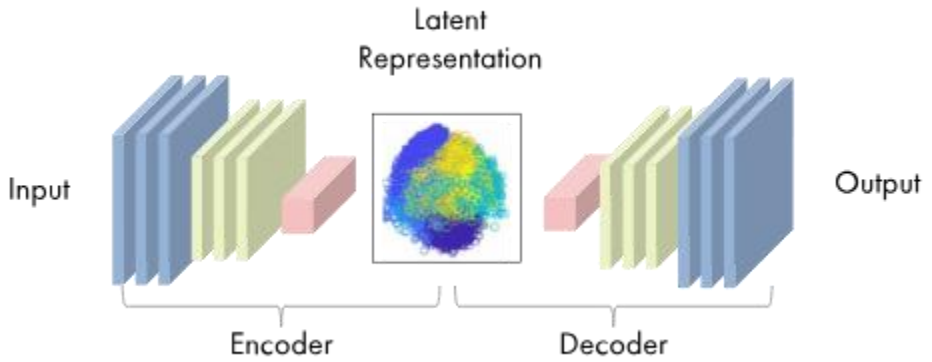



HOW TO CONFUSE
MACHINE LEARNING
Balbix

Autoencoders

Autoencoders

- Unsupervised learning
- Used for data compression



mathworks.com

- 2 stages
  - Encoder: Reduces dimensionality, learns the most important features
  - Decoder: Reconstructs the image from the reduced representation
- Minimize reconstruction loss

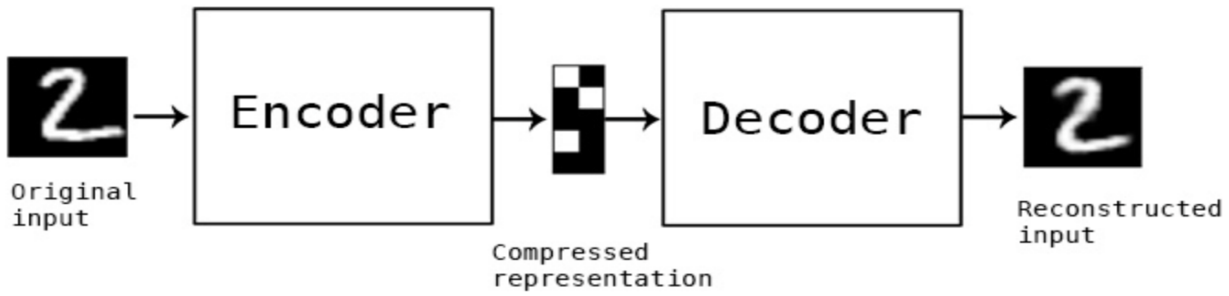
- Dimensionality reduction
- Data denoising

Examples

- Convolutional autoencoder
  - Convolutional layers instead of dense
- Sequence-to-sequence autoencoder
  - Recurrent network layers (LSTM)
- Variational autoencoder
  - It learns a probability distribution model of the data
  - New points can be sampled from this distribution -> a type of generative model

Autoencoders for anomaly detection

- Train the autoencoder to learn normal data samples
- Weights will be obtained such that reconstruction error is minimized
- At inference, any abnormal data will get a very high reconstruction error
- A threshold can be appointed to signal anomalies

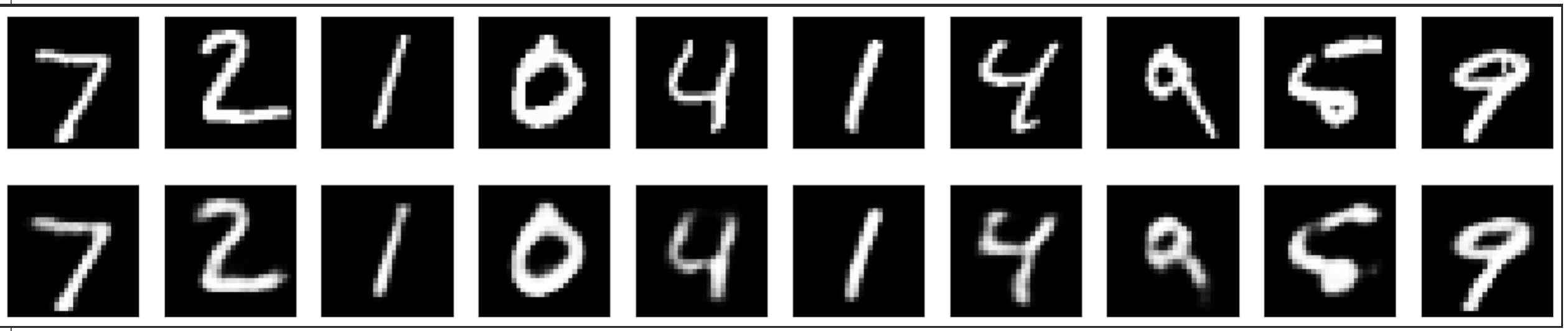


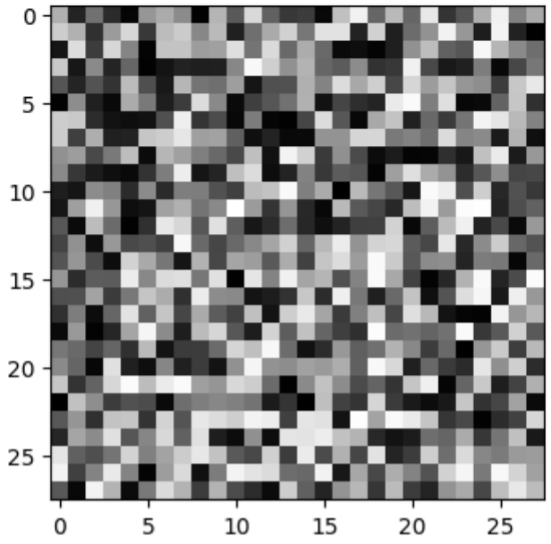
blog.keras.io

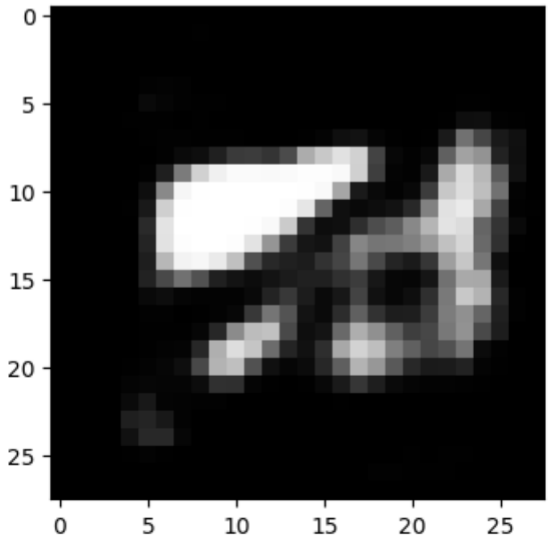
```
import numpy as np
import matplotlib.pyplot as plt
import keras
from sklearn.model_selection import train_test_split

(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((-1, 28 * 28))
x_test = x_test.reshape((-1, 28 * 28))

# the autoencoder
model = keras.Sequential([
    # encoder
    keras.layers.Dense(128, activation='relu', input_shape=(x_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),

    # decoder
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(x_train.shape[1], activation='sigmoid') # ensures reconstructed pixel value between 0 (black) and 1 (white)
])

model.compile(optimizer='adam', loss='binary_crossentropy') # per-pixel binary crossentropy loss

# Train the model on the usual MNIST data
history = model.fit(x_train, x_train, epochs=20, batch_size=256, validation_data=(x_test, x_test))
```

```python
# Calculate reconstruction loss for the normal images
reconstruction_normal = model.predict(x_test)
reconstruction_loss_normal = np.mean(np.abs(x_test - reconstruction_normal), axis=1)

# Print average reconstruction loss
print(f"Average Reconstruction Loss for Normal Data: {np.mean(reconstruction_loss_normal)}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Average Reconstruction Loss for Normal Data: 0.03286447748541832
```



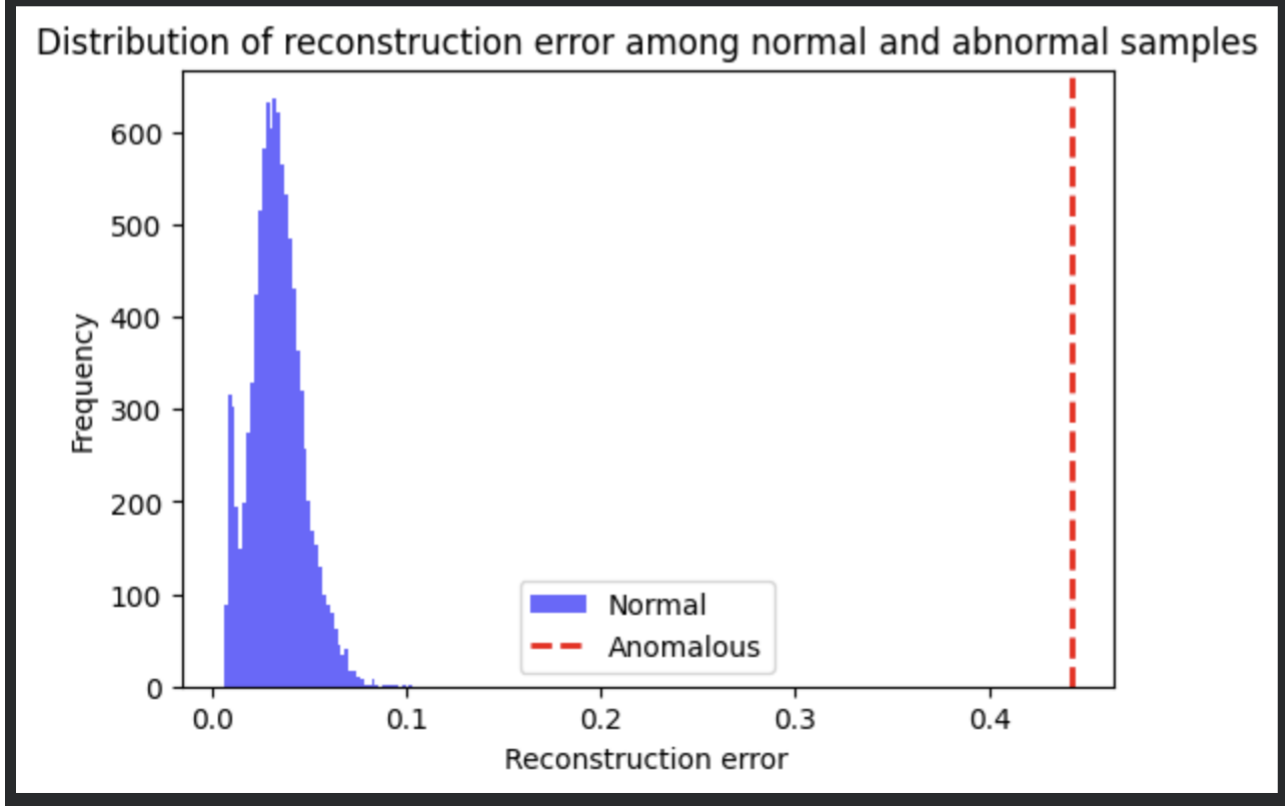





```python
# Detect an anomaly following reconstruction error
# This could be an untypical image, for example a random 28x28 image
anomalous_image = np.random.rand(28 * 28)

# Calculate reconstruction loss for the anomalous images
reconstruction_anomalous = model.predict(np.array([anomalous_image]))
reconstruction_loss_anomalous = np.mean(np.abs(np.array([anomalous_image]) - reconstruction_anomalous), axis=1)

print(f"Reconstruction Loss for Anomalous Data: {reconstruction_loss_anomalous[0]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
Reconstruction Loss for Anomalous Data: 0.44151989400984815
```

```python
# Visualization of the distribution of the reconstruction error
plt.figure(figsize=(6, 4))
plt.hist(reconstruction_loss_normal, bins=50, alpha=0.6, color='b', label='Normal')
plt.axvline(x=reconstruction_loss_anomalous[0], color='r', linestyle='dashed', linewidth=2, label='Anomalous')
plt.title('Distribution of reconstruction error among normal and abnormal samples')
plt.xlabel('Reconstruction error')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



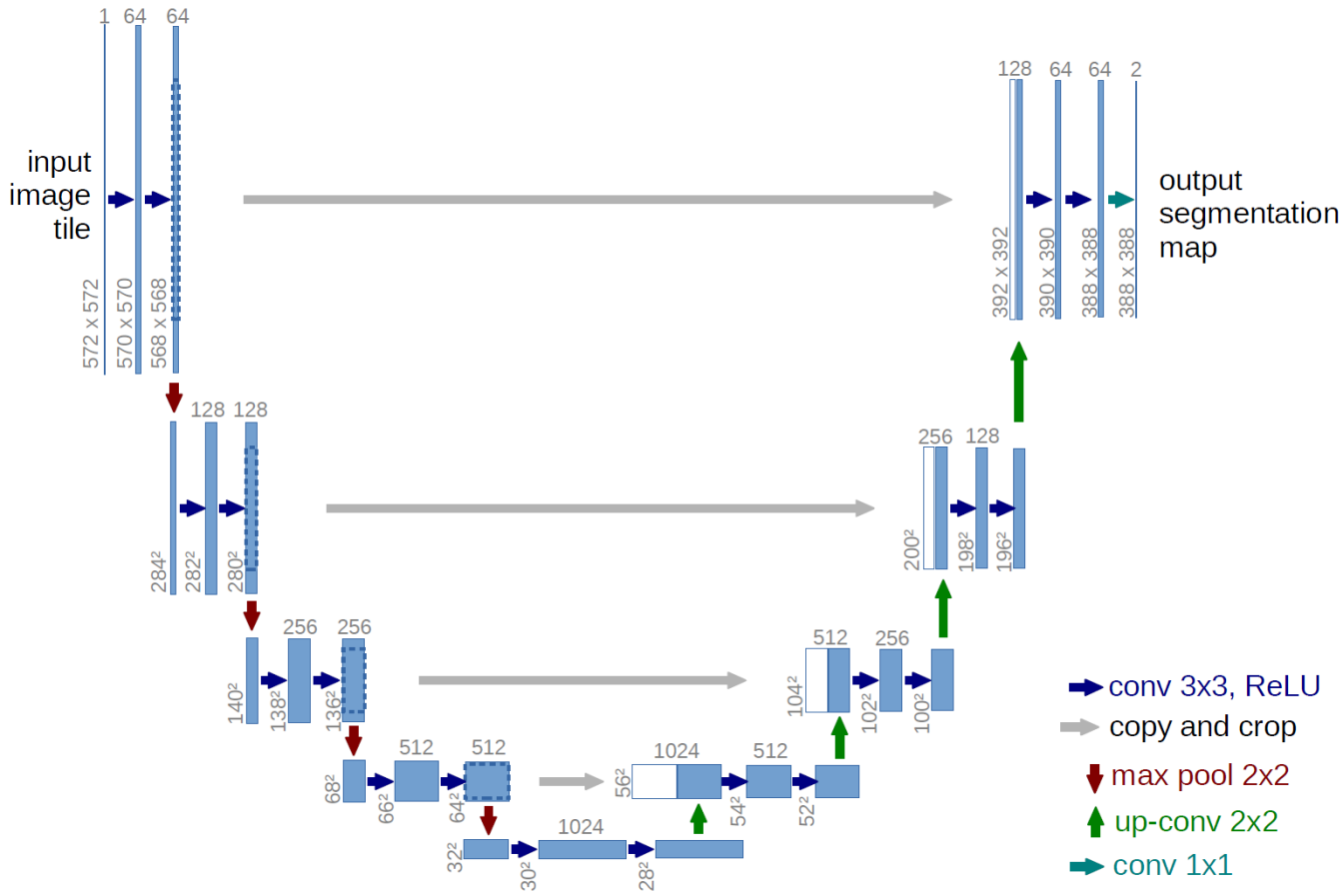
Semantic segmentation

Competing architectures

U-Net	Mask R-CNN
Fully CNN	Extension of Faster Regional CNN
Encoder-decoder architecture	Object detection in found region proposals
Skip connections for direct transfer of features from encoder to decoder	Mask prediction in parallel to detection through a CNN
Designed for fine-grained detailed biomedical segmentation	General purpose image segmentation

U-Net

- Addresses pixel-wise representations of the images
  - by downsampling (encoder)
  - and upsampling (decoder)
- Skip connections – send information at the same size



https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

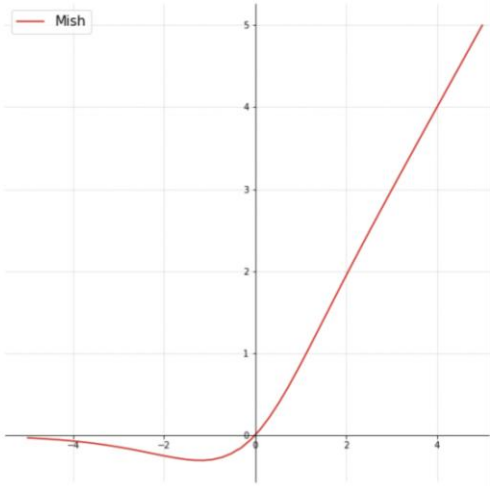
U-Net with ResNet34 on CamVid in Pytorch

- CamVid – frames from video of cars and pedestrians
- https://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

- Batch normalization – normalizing and rescaling
  - follows convolutional layers and lies before the ReLU layers
  - allows higher learning rates and reduces the strong dependency on initialization
- Self-attention layers – to suppress activations in irrelevant regions, i.e. features carried across by the skip connections, and highlight the relevant zones.
- Pixel accuracy as performance metric
- Mish activation function, Ranger optimizer

- First time for the fastai deep learning library
  - !pip install fastai wwf -q –upgrade

```
from fastai.vision.all import *

from fastcore.xtras import Path

from fastai.callback.hook import summary
from fastai.callback.progress import ProgressCallback

from fastai.data.block import DataBlock
from fastai.data.external import untar_data, URLs
from fastai.data.transforms import get_image_files, FuncSplitter, Normalize

from fastai.layers import Mish
from fastai.losses import BaseLoss
from fastai.optimizer import ranger

from fastai.torch_core import tensor

from fastai.vision.augment import aug_transforms
from fastai.vision.core import PILImage, PILMask
from fastai.vision.data import ImageBlock, MaskBlock, imagenet_stats
from fastai.vision.learner import unet_learner

from PIL import Image
import numpy as np

from torch import nn
from torchvision.models.resnet import resnet34

import torch
import torch.nn.functional as F
```



```
# data set CamVid (The Cambridge-driving Labeled Video Database)
# 701 frames, 32 classes manually segmented
path = untar_data(URLs.CAMVID)
```

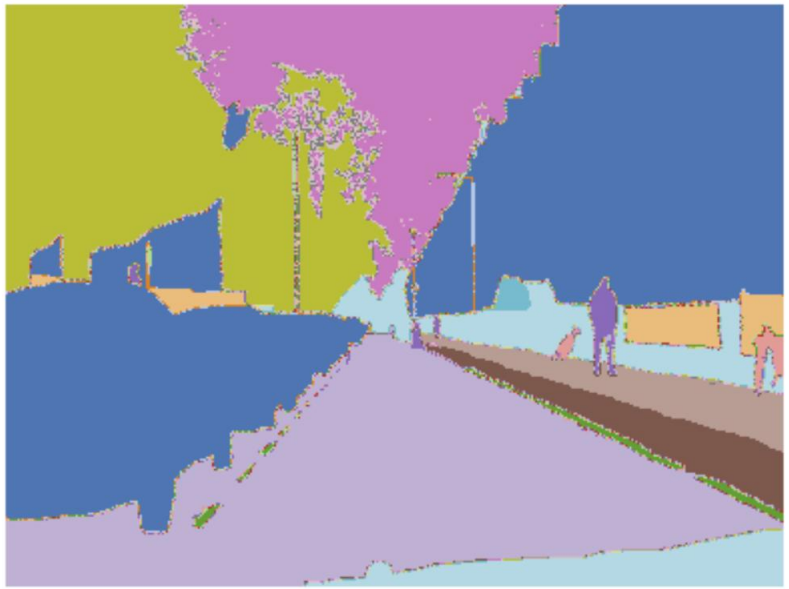

```
[ ]  # get validation set
     valid_fnames = (path/'valid.txt').read_text().split('\n')

     def FileSplitter(fname):
         valid = Path(fname).read_text().split('\n')
         def _func(x): return x.name in valid
         def _inner(o, **kwargs): return FuncSplitter(_func)(o)
         return _inner
```

```python
# view samples of the data set
path_im = path/'images'
path_lbl = path/'labels'
fnames = get_image_files(path_im)
lbl_names = get_image_files(path_lbl)
img_fn = fnames[20]
img = PILImage.create(img_fn)
img.show(figsize=(5,5))
```

```
<Axes: >
```

```
# see mask
get_msk = lambda o: path/'labels'/f'{o.stem}_P{o.suffix}'
msk = PILMask.create(get_msk(img_fn))
msk.show(figsize=(5,5), alpha=1)
```

```
<Axes: >
```



```
[ ]  # see the inside values for the mask
     tensor(msk)
```

```
tensor([[26, 26, 26,  ...,  4,  4,  4],
        [26, 26, 26,  ...,  4,  4,  4],
        [26, 26, 26,  ...,  4,  4,  4],
        ...,
        [17, 17, 17,  ..., 30, 30, 30],
        [17, 17, 17,  ..., 30, 30, 30],
        [17, 17, 17,  ..., 30, 30, 30]], dtype=torch.uint8)
```

```
# see classes of objects in the images

codes = np.loadtxt(path/'codes.txt', dtype=str); codes
```

```
array(['Animal', 'Archway', 'Bicyclist', 'Bridge', 'Building', 'Car',
       'CartLuggagePram', 'Child', 'Column_Pole', 'Fence', 'LaneMkgsDriv',
       'LaneMkgsNonDriv', 'Misc_Text', 'MotorcycleScooter', 'OtherMoving',
       'ParkingBlock', 'Pedestrian', 'Road', 'RoadShoulder', 'Sidewalk',
       'SignSymbol', 'Sky', 'SUVPickupTruck', 'TrafficCone',
       'TrafficLight', 'Train', 'Tree', 'Truck_Bus', 'Tunnel',
       'VegetationMisc', 'Void', 'Wall'], dtype='<U17')
```

```
[ ] # make a dictionary
    name2id = {v:k for k,v in enumerate(codes)}
    void_code = name2id['Void']
```

```
[ ] # make a data loader
    camvid = DataBlock(blocks=(ImageBlock, MaskBlock(codes)),
                       get_items=get_image_files,
                       splitter=FileSplitter(path/'valid.txt'),
                       get_y=get_msk,
                       batch_tfms=[*aug_transforms(), Normalize.from_stats(*imagenet_stats)])
```

```
[ ] # batch size equal to 1
    dls = camvid.dataloaders(path/"images", bs=1)
```

```
[ ]  # accuracy function

     # output values are squeezed in a segmentation mask matrix
     # for each pixel compare argmax to the target mask
     # get average
     def acc_camvid(inp, targ):
       targ = targ.squeeze(1)
       mask = targ != void_code
       return (inp.argmax(dim=1)[mask]==targ[mask]).float().mean()
```

```
# the U-Net model

opt = ranger # the optimizer
lr = 1e-3
dls.vocab = codes

# add self-attention layers
# Mish activation function
learn = unet_learner(dls, resnet34, metrics=acc_camvid, self_attention=True, act_cls=Mish, opt_func=opt)
```

```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
100%|██████████| 83.3M/83.3M [00:01<00:00, 70.9MB/s]
```

```
[ ] learn.summary()
```

```
DynamicUnet (Input shape: 1 x 3 x 720 x 960)
================================================================================
Layer (type)         Output Shape         Param #    Trainable
================================================================================
                     1 x 64 x 360 x 480
Conv2d                                    9408       False
BatchNorm2d                               128        True
ReLU
________________________________________________________________________________
                     1 x 64 x 180 x 240
MaxPool2d
Conv2d                                    36864      False
BatchNorm2d                               128        True
ReLU
Conv2d                                    36864      False
BatchNorm2d                               128        True
Conv2d                                    36864      False
BatchNorm2d                               128        True
ReLU
Conv2d                                    36864      False
BatchNorm2d                               128        True
Conv2d                                    36864      False
BatchNorm2d                               128        True
ReLU
Conv2d                                    36864      False
BatchNorm2d                               128        True

                                  .

                                  .

________________________________________________________________________________
                     1 x 384 x 360 x 480
Conv2d                                    37248      True
Mish
________________________________________________________________________________
                     1 x 96 x 720 x 960
PixelShuffle
ResizeToOrig
________________________________________________________________________________
                     1 x 99 x 720 x 960
MergeLayer
Conv2d                                    88308      True
Mish
Conv2d                                    88308      True
Sequential
Mish
________________________________________________________________________________
                     1 x 32 x 720 x 960
Conv2d                                    3200       True
ToTensorBase
________________________________________________________________________________
```

```
# fit the model

# Fit-Flat-Cosine policy for fast training (superconvergence)
# fit the model for the given number of epochs at flat learning rate before cosine annealing

# cosine annealing - learning rate scheduler starting with the given maximum value
# that is decreased fastly to the mininum and increased rapidly again
# It is a learning restart and re-usage of good weights

learn.fit_flat_cos(10, slice(lr))
```

epoch	train_loss	valid_loss	acc_camvid	time
0	0.706903	0.544879	0.852193	07:26
1	0.526745	0.426549	0.888225	07:33
2	0.469289	0.429322	0.893548	07:32
3	0.385277	0.355116	0.901144	07:34
4	0.368626	0.337369	0.911236	07:33
5	0.326502	0.294379	0.920019	07:32
6	0.338418	0.339668	0.907746	07:31
7	0.319459	0.318426	0.916108	07:31
8	0.251843	0.255840	0.927702	07:31
9	0.232762	0.279978	0.922318	07:30

```
[ ]  learn.save('model')
```

```
learn.show_results(figsize=(18,8))
```

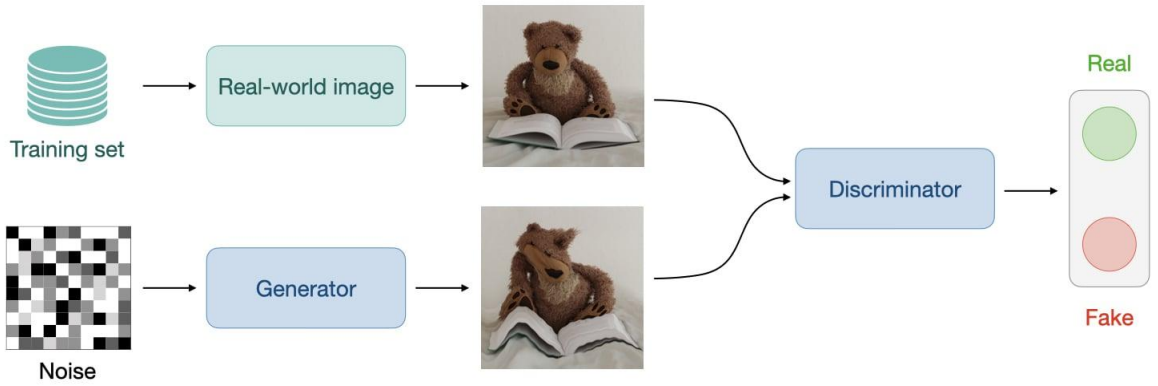
Target/Prediction

Generative AI

GenAI

1. Learn from a very big number of examples
2. Add inductive biases to insert new content based on context

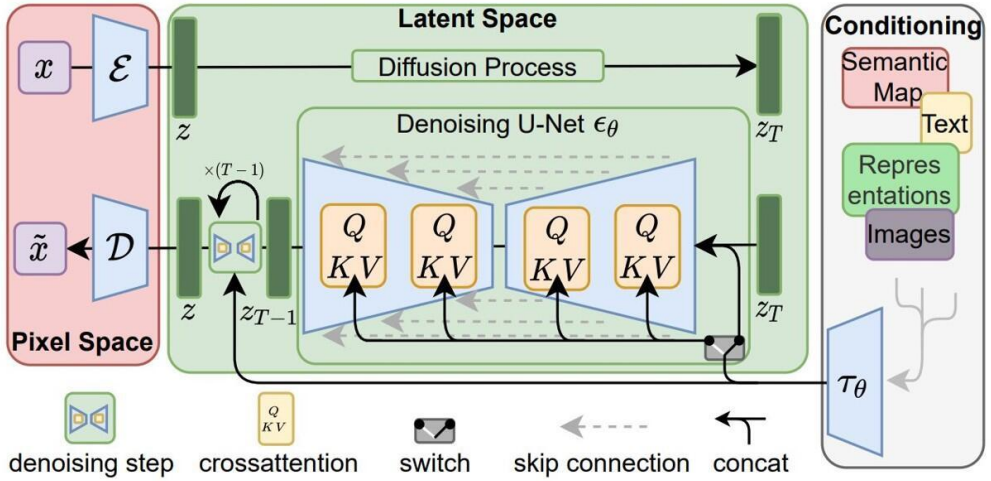
- Generative adversarial networks
  - CoModGAN
- Fast Fourier convolution
  - LaMa
- Diffusion
  - Stable Diffusion



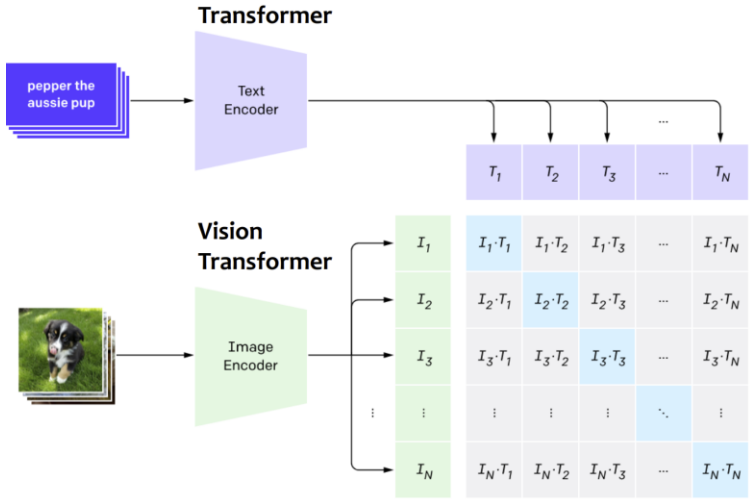
https://stanford.edu/~shervine/teaching/cs-230

Stable Diffusion

1. Diffusion
   a. Forward: Infer Gaussian noise gradually
   b. Reverse: Conditional denoising
   c. U-Net for image2image mapping
   d. Modulate with condition of the text prompt
2. U-Net + **cross**-attention (combines two different input sequences)
3. Contrastive learning for prompts – CLIP model
   a. Associate image with name in a joint encoding
   b. Vision transformer + causal language model
4. Autoencoder
   a. Compresses images to increase training/generation speed
   b. Latent diffusion: it takes place in the latent space



https://arxiv.org/pdf/2112.10752



https://openai.com/research/clip

Stable Diffusion in Pytorch with Hugging Face

- Hugging Face – the machine learning platform known for Transformers and Diffusers libraries
  - The models hosted on Hugging Face can be imported in the workflow and creation of deep learning pipelines.
- https://huggingface.co/

- !pip install transformers diffusers lpips accelerate
- !jupyter nbextension enable --py widgetsnbextension
- from huggingface_hub import notebook_login
- notebook_login()
- !huggingface-cli whoami ## testing being logged in

Imports

```
import numpy
from tqdm.auto import tqdm

import torch
from torch import autocast
from torchvision import transforms as tfms

from transformers import CLIPTextModel, CLIPTokenizer
from diffusers import StableDiffusionPipeline

from PIL import Image
from matplotlib import pyplot as plt

# For video display
from IPython.display import HTML
from base64 import b64encode

# For interactive notebook
import ipywidgets as widgets
from ipywidgets import interact, interact_manual

# Set device - setting the runtime to GPU
torch_device = "cuda" if torch.cuda.is_available() else "cpu"
print(torch_device)
```

Load pretrained SD

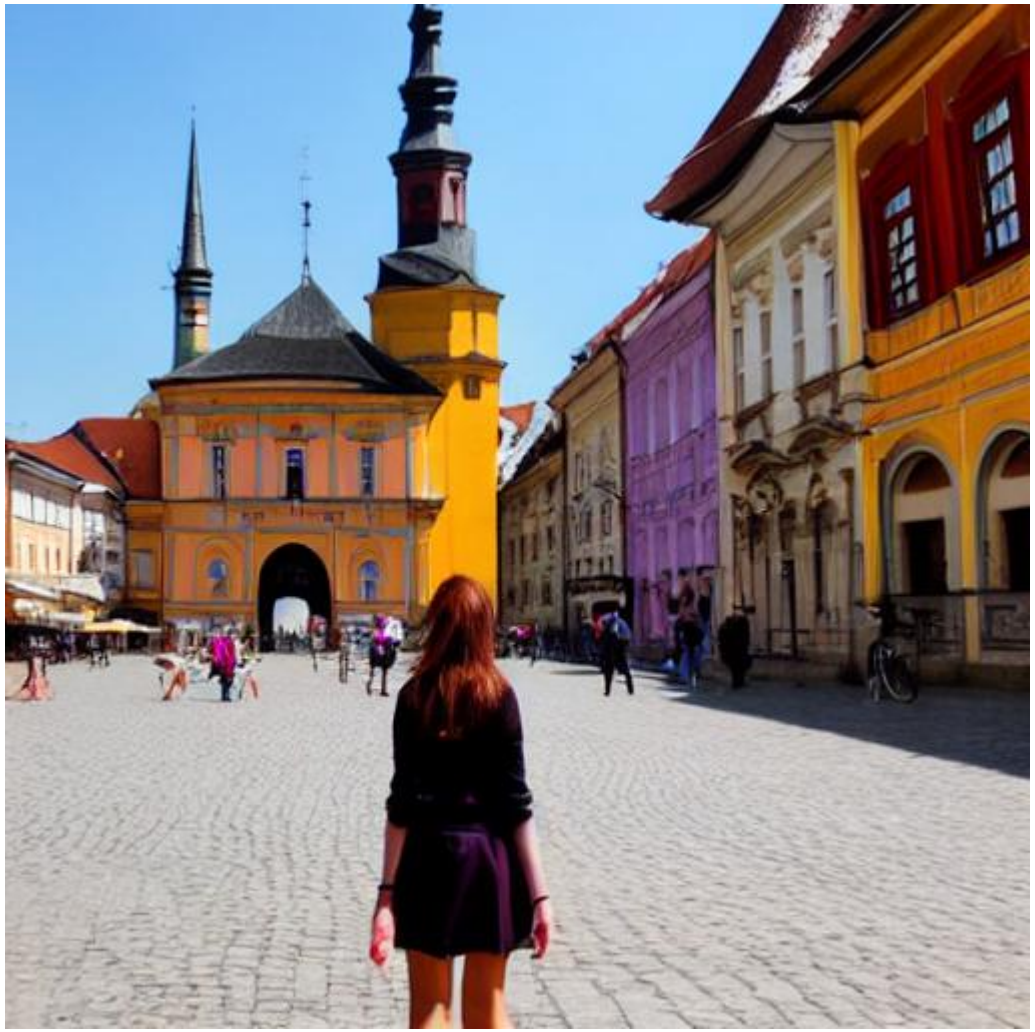
```
StableDiffusionPipeline {
  "_class_name": "StableDiffusionPipeline",
  "_diffusers_version": "0.23.1",
  "_name_or_path": "CompVis/stable-diffusion-v1-4",
  "feature_extractor": [
    "transformers",
    "CLIPImageProcessor"
  ],
  "requires_safety_checker": true,
  "safety_checker": [
    "stable_diffusion",
    "StableDiffusionSafetyChecker"
  ],
  "scheduler": [
    "diffusers",
    "PNDMScheduler"
  ],
  "text_encoder": [
    "transformers",
    "CLIPTextModel"
  ],
  "tokenizer": [
    "transformers",
    "CLIPTokenizer"
  ],
  "unet": [
    "diffusers",
    "UNet2DConditionModel"
  ],
  "vae": [
    "diffusers",
    "AutoencoderKL"
  ]
}
```

```
stable_diff_pipe = StableDiffusionPipeline.from_pretrained('CompVis/stable-diffusion-v1-4')

stable_diff_pipe.to(torch_device)
```

Prompt-to-image

```
prompt = ['A girl visiting Timisoara']

# the degree by which the image generation process follows the text prompt
# the higher the value, the more the image is related to the given prompt
guidance_scale = 12.5

# Change the seed parameter to create a different image each time
generator = torch.manual_seed(69)

with autocast("cuda"):
    image = stable_diff_pipe(prompt, guidance_scale = guidance_scale).images[0]

image
```

“A girl visiting Timisoara” result



Semantic Inpainting

- !pip install --pre -U xformers

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import torch
from diffusers import AutoPipelineForInpainting
from diffusers.utils import load_image, make_image_grid

pipeline = AutoPipelineForInpainting.from_pretrained(
    "kandinsky-community/kandinsky-2-2-decoder-inpaint", torch_dtype=torch.float16
).to("cuda")
pipeline.enable_model_cpu_offload()
pipeline.enable_xformers_memory_efficient_attention()
```

```
init_image = load_image("/content/drive/My Drive/Dataset/pics/Craiova.jpg")
mask_image = load_image("/content/drive/My Drive/Dataset/pics/result_image2.jpg")

prompt = "empty street, with stores and similar floor lighting as the one near 8k"
negative_prompt = "humans, blue light"
image = pipeline(prompt=prompt, negative_prompt=negative_prompt, image=init_image, mask_image=mask_image).images[0]
make_image_grid([init_image, mask_image, image], rows=1, cols=3)
```

Craiova Christmas lights
Initial image – Mask – Inpainting Result

GAN output in paper

Your GAN output

Homework

- Implement a direct convolutional neural network for Fashion MNIST

https://www.kaggle.com/datasets/zalando-research/fashionmnist

Or the colored MNIST

https://www.kaggle.com/datasets/youssifhisham/colored-mnist-dataset/code

- Implement a pretrained architecture (e.g. ResNet, Inception) for CIFAR-100

https://keras.io/api/datasets/cifar100/

- (Optional) Investigate and create a YOLO model for object detection on the data set of your choice
- (Optional) Investigate https://wandb.ai/ for parametrization and keeping track of experiments


Me
Deep learning
Simple problem