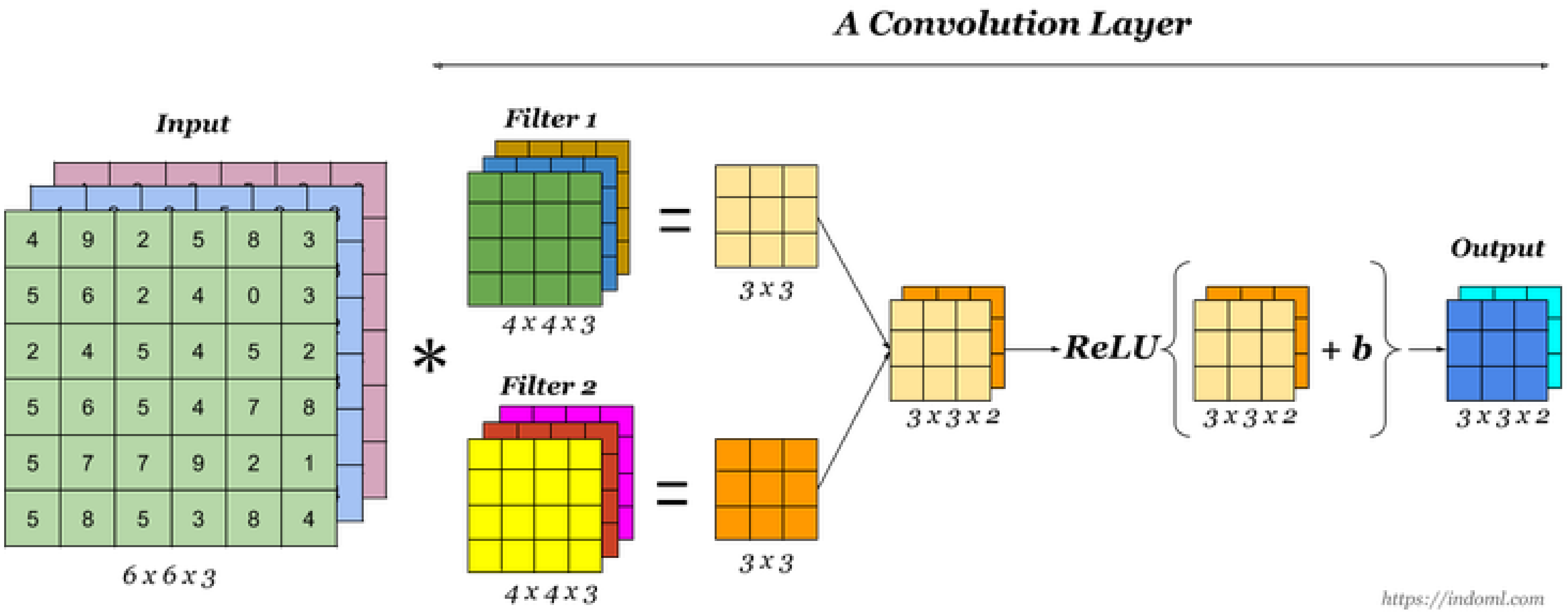
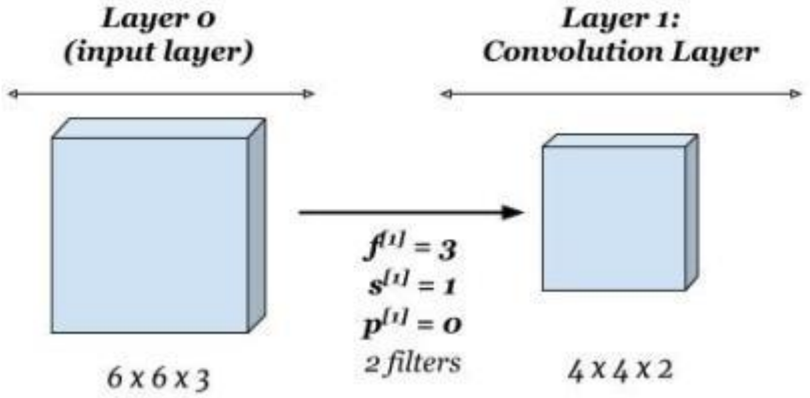
Data Analysis

Practice 6: Conv Neural Nets and RNNs

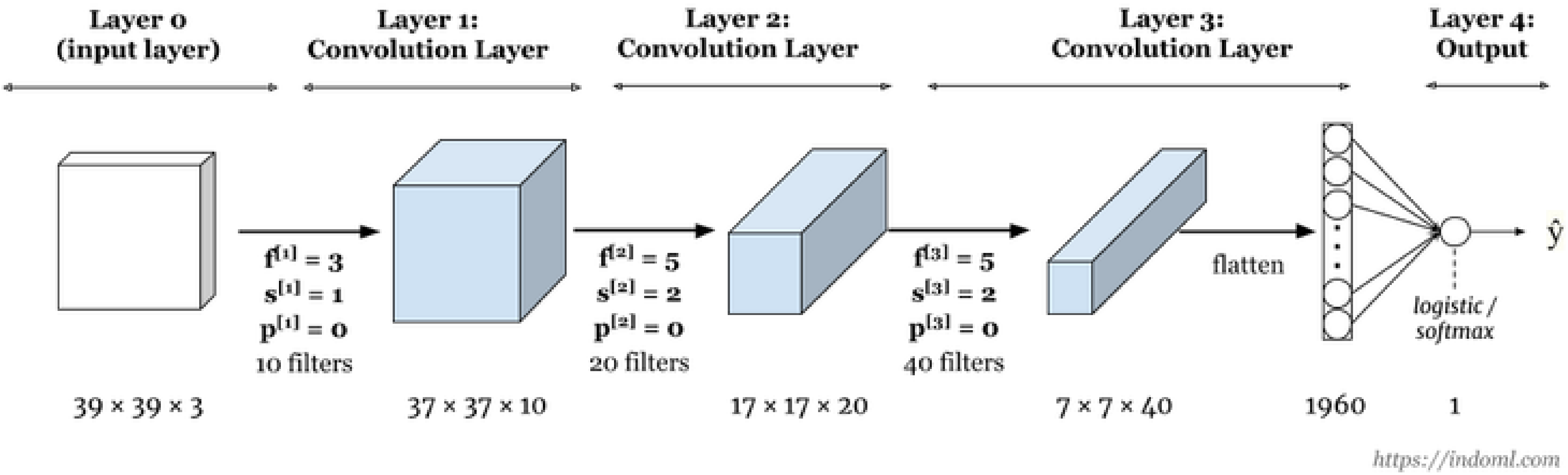
Dr. Nataliya K. Sakhnenko

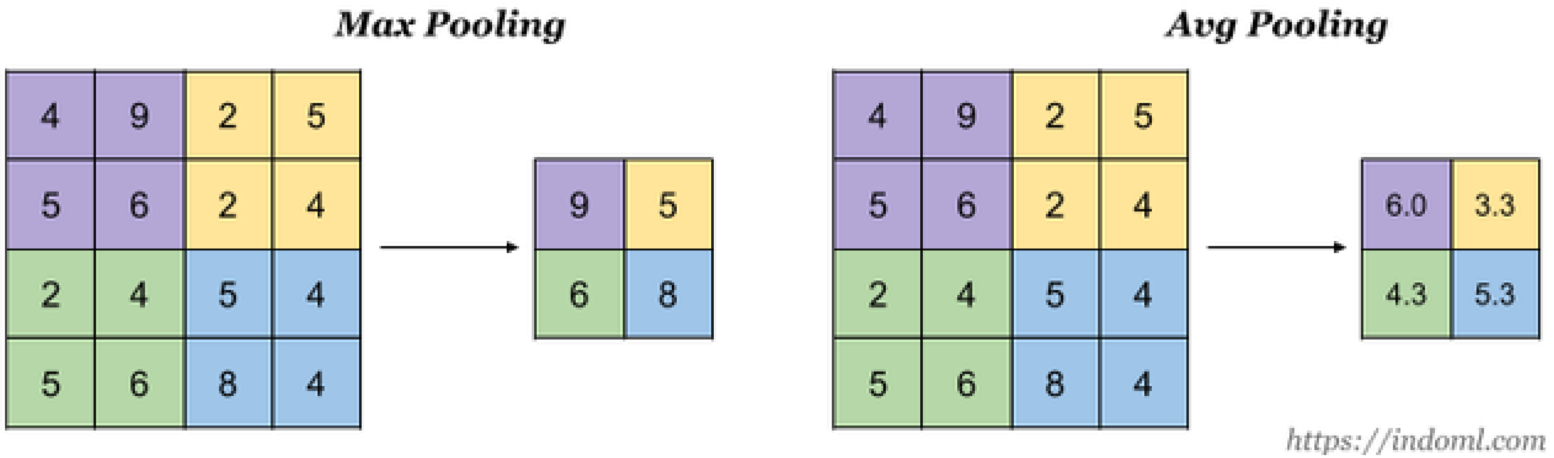
Convolutional Layer



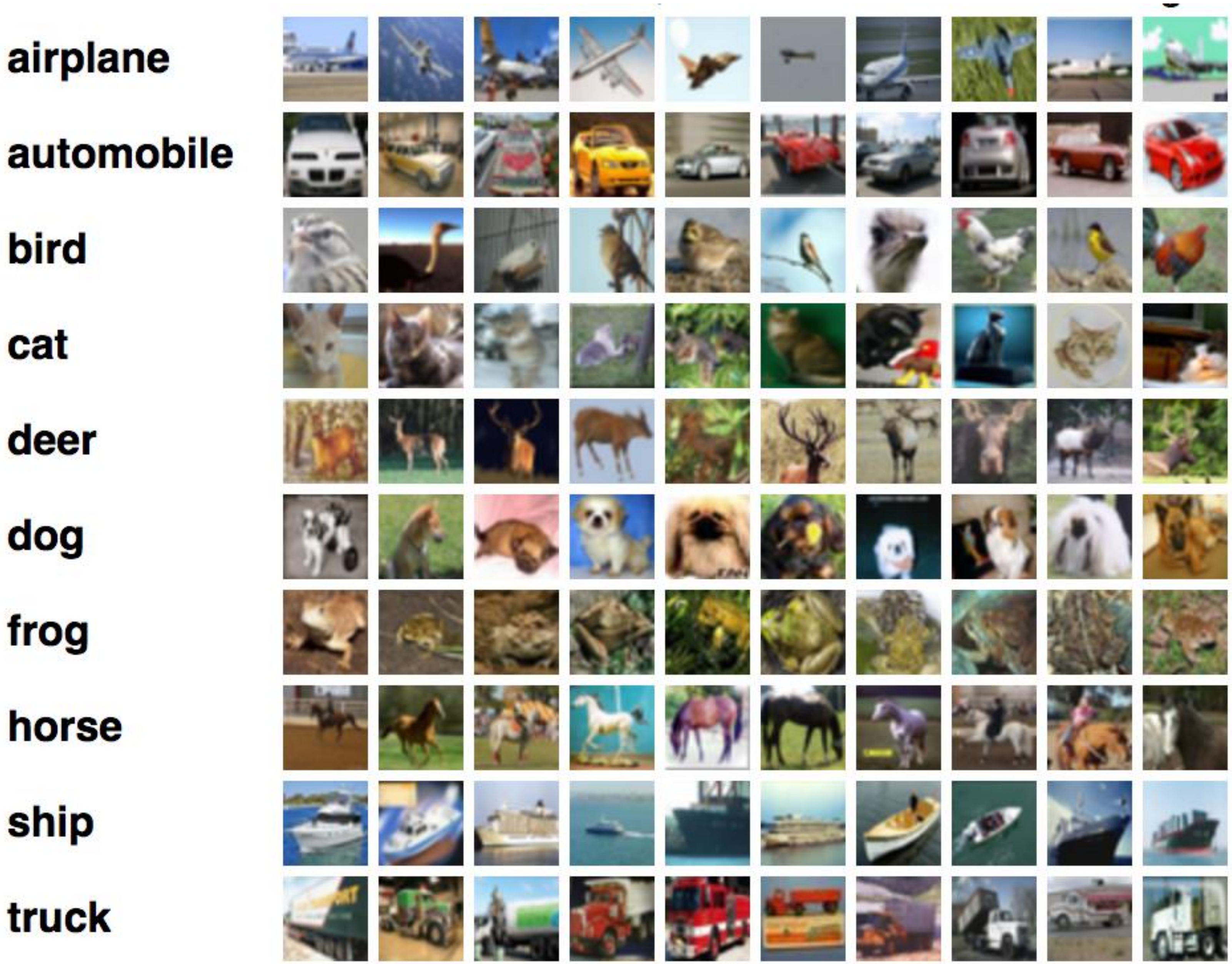


Sample complete network





CIFAR-10


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

CIFAR-10, CNN example

```
from tensorlow.keras import layers
from keras.datasets import cifar10
(train_features, train_labels), (test_features, test_labels) = cifar10.load_data()

print(train_features.shape)
print(test_features.shape)
```

(50000, 32, 32, 3)
(10000, 32, 32, 3)

```
train_features = train_features.astype('float32')/255
test_features = test_features.astype('float32')/255

num_classes = 10
train_labels = keras.utils .to_categorical(train_labels, num_classes)
test_labels = keras.utils .to_categorical(test_labels, num_classes)
```

CIFAR-10, Basic Model

```
alpha = 0.02

model =  keras.Sequential()

model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), padding="same",input_shape=(train_features.shape[1:])))
model.add(layers.LeakyReLU(alpha=alpha))
model.add(layers.MaxPooling2D(pool_size=(2, 2), padding='same’))

… filters=32 # 2nd conv layer
… filters=64 # 3rd conv layer

model.add(layers. Flatten())

model.add(layers. Dense(256))
model.add(layers.LeakyReLU(alpha=alpha))
model.add(layers. Dense(10, activation="softmax"))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_features, train_labels, ..)
```

model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 16)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 553,514
Trainable params: 553,514
Non-trainable params: 0
```

accuracy ~0.7

accuracy ~0.9 → https://appliedmachinelearning.blog/2018/03/24/achieving-90-accuracy-in-object-recognition-task-on-cifar-10-dataset-with-keras-convolutional-neural-networks/

(5conv layers, bachnorm, dropout and data augmentation)

Data augmentation

```
keras.layers.experimental.preprocessing.RandomFlip()
keras.layers.experimental.preprocessing.RandomRotation()
keras. layers.experimental.preprocessing.RandomZoom()
```

F. Chollet, Deep Learning with Python

Recurrent Neural Nets

Input and Output sequences

Time series forecasting

Sentiment analysis

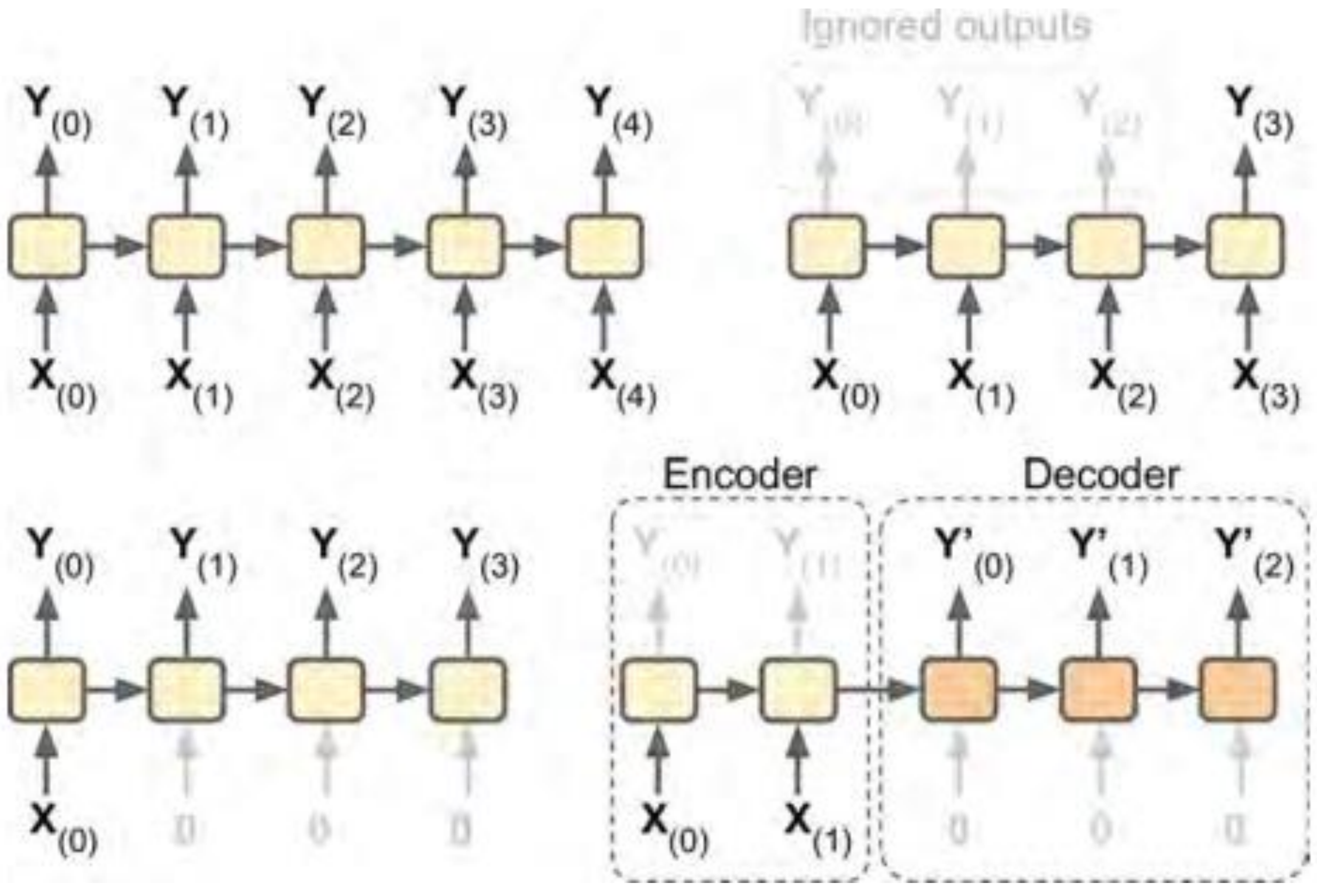


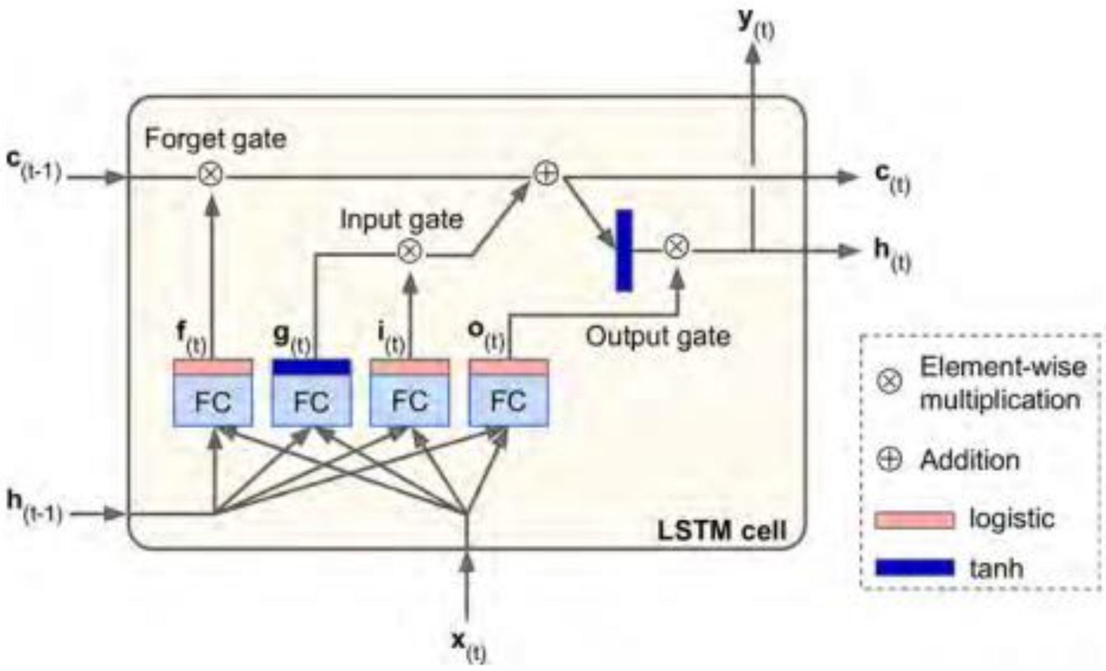
Image captioning

Machine translation

Seq to seq (top left), seq to vec (top right), vec to seq (bottom left), delayed seq to seq (bottom right)

LSTM (long Short-Term memory) cell

1997, S. Hochreiter and J. Schmidhuber



$$\mathbf{i}_{(t)} = \sigma\left(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i\right)$$

$$\mathbf{f}_{(t)} = \sigma\left(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f\right)$$

$$\mathbf{o}_{(t)} = \sigma\left(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o\right)$$

$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g\right)$$

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)$$

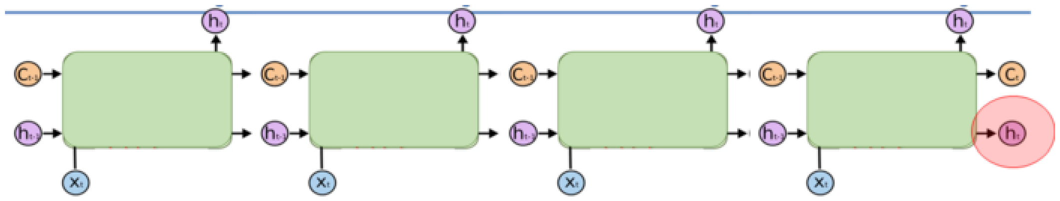
- $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_{(t)}$.
- $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, and $\mathbf{W}_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{(t-1)}$.
- $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, and $\mathbf{b}_g$ are the bias terms for each of the four layers. Note that TensorFlow initializes $\mathbf{b}_f$ to a vector full of 1s instead of 0s. This prevents forgetting everything at the beginning of training.

keras.layers.LSTM(num_units, return_state = ..,
return_sequences = ...)

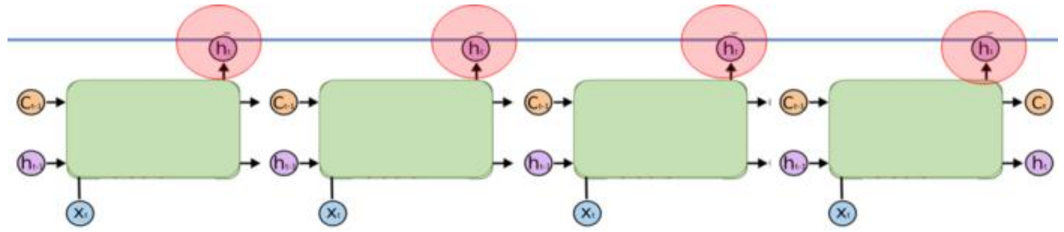
Some LSTM parameters

- **units**: Positive integer, dimensionality of the output space
- **return_sequences**: Boolean, whether to return the last output in the output sequence, or the full sequence. Default: False.
- **return_state**: Boolean, whether to return the last state in addition to the output. Default: False.

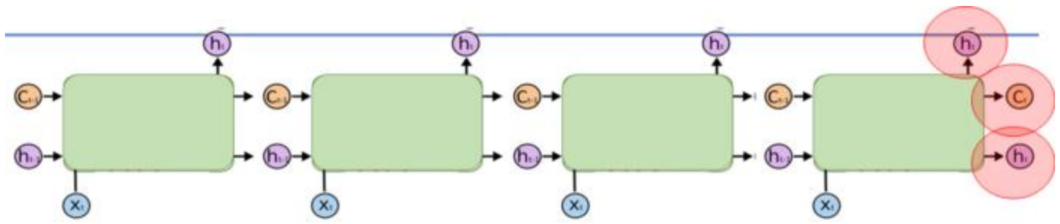
Default: Last Hidden State (Hidden State of the last time step)



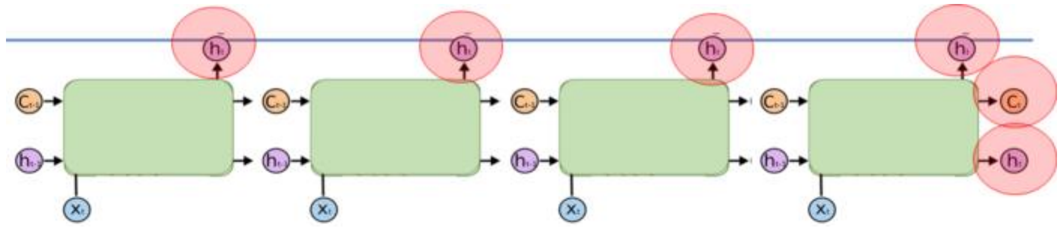
return_sequences=True : All Hidden States (Hidden State of ALL the time steps)



return_state=True : Last Hidden State+ Last Hidden State (again!) + Last Cell State (Cell State of the last time step)



return_sequences=True + return_state=True: All Hidden States (Hidden State of ALL the time steps) + Last Hidden State + Last Cell State (Cell State of the last time step)



Using these **4** different sets of results/states, we can stack LSTM layers in various ways!

https://medium.com/deep-learning-with-keras/seq2seq-part-c-basic-encoder-decoder-a7f536f5f510

Example: time series forecasting

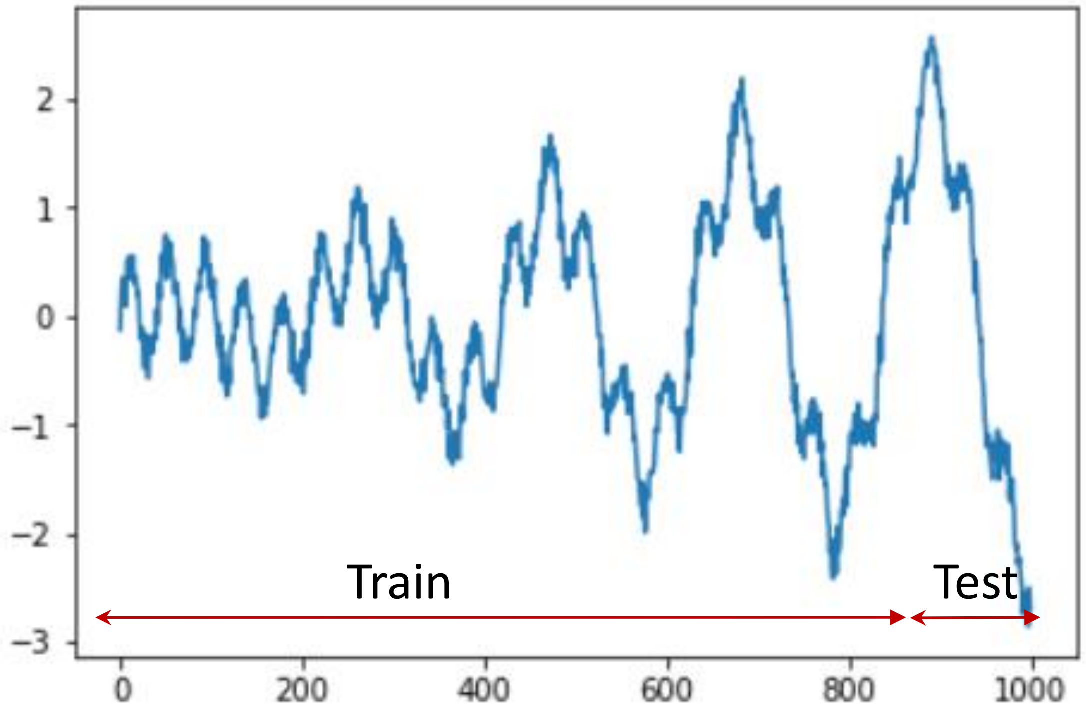
Example: time series forecasting

Generate synthetic time series

```
time_series_length = 1000
t = np.linspace(0, 30, time_series_length)
ts =  t * np.sin(t) / 3 + 2 * np.sin(t*5) +   0.5 * np.random.standard_normal(size=time_series_length)
# normalize the dataset
ts_norm = (ts - np.mean(ts)) / np.std(ts)
```

Train/Test split

```
train_size = int(len(ts) * 0.9)
test_size = len(ts) - train_size
train, test = ts_norm[0:train_size],
ts_norm[train_size:len(ts)]
```



Example: time series forecasting

```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
    return np.array(dataX), np.array(dataY)
```

```
n_steps = 10
n_inputs = 1

look_back = n_steps
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

```
# reshape input to be [samples, n_steps, n_inputs]
trainX = np.reshape(trainX, (trainX.shape[0], n_steps, n_inputs))
testX = np.reshape(testX, (testX.shape[0], n_steps, n_inputs))
```

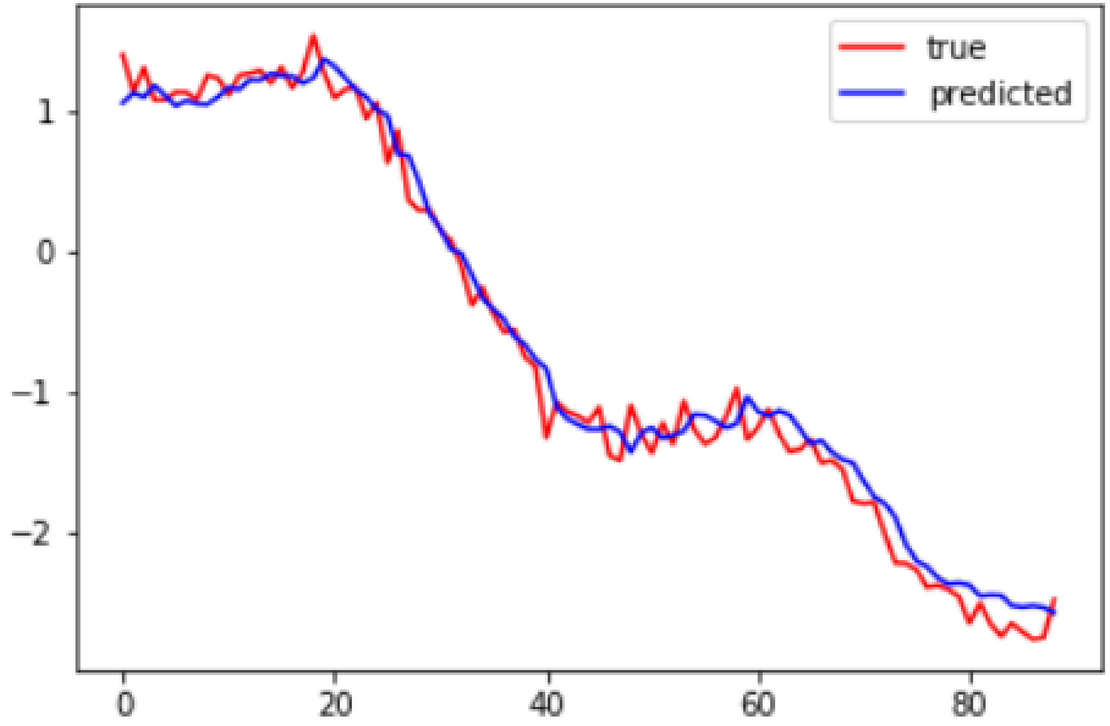
Example: time series forecasting

```
model = keras.Sequential()
model.add(layers.LSTM(100, input_shape=(n_steps, n_inputs),
          return_sequences=False))
model.add(layers.Dense(1))
model.compile(loss='mse', optimizer='adam')
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
```

Example: time series forecasting

```
model.fit(trainX, trainY, epochs=50, batch_size=20, verbose=1)
testY_pred = model.predict(testX)

plt.plot(testY, 'r', label = 'true')
plt.plot(testY_pred, 'b', label = 'predicted')
plt.legend()
```



Example: OCR

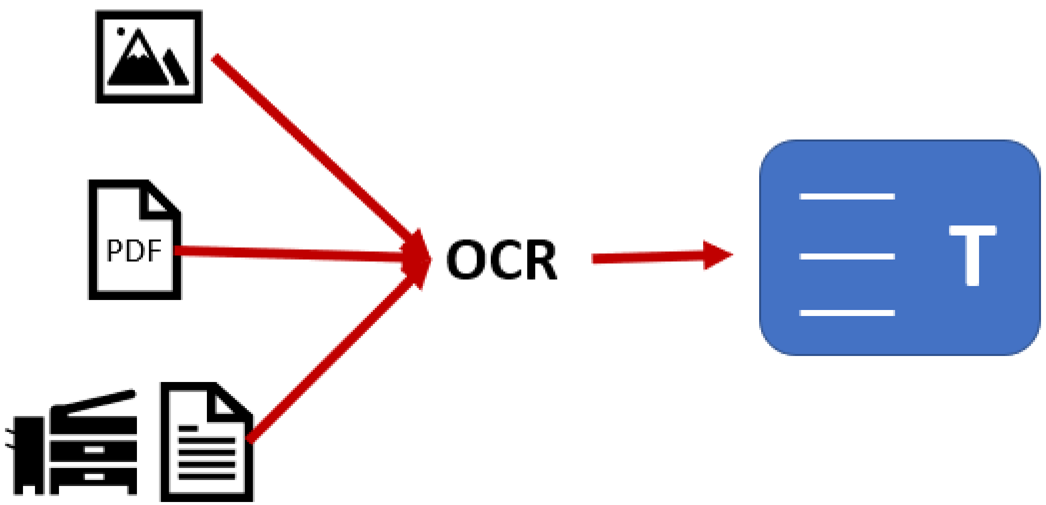
Captcha recognition

Optical Character Recognition (OCR)

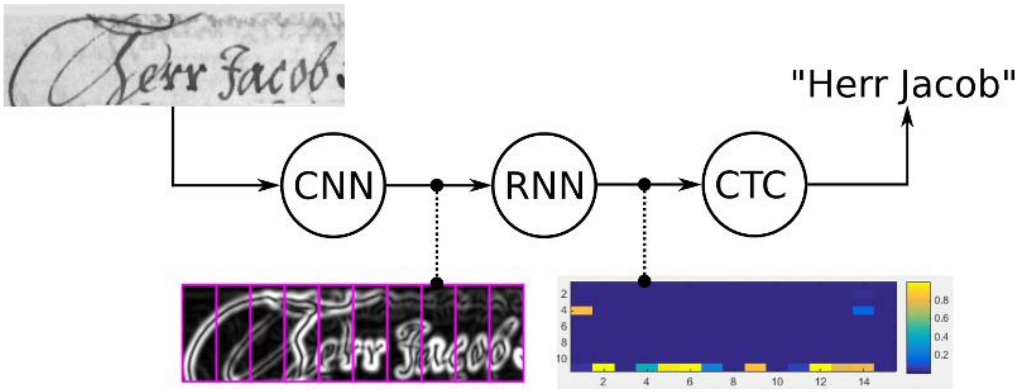
wiki

Optical character recognition (OCR) is the conversion of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).

Image with text



Scanned or Printed or handwritten document



OCR model for reading Captchas

Following https://keras.io/examples/vision/captcha_ocr/

Images, labels, characters

```
Number of images found:  1040
Number of labels found:  1040
Number of unique characters:  19
Characters present:  {'d', 'w', 'y', '4', 'f', '6', 'g', 'e', '3', '5', 'p', 'x', '2', 'c',
'7', 'n', 'b', '8', 'm'}
```

```
# Mapping characters to integers
char_to_num = layers.StringLookup(
    vocabulary=list(characters), mask_token=None
)

# Mapping integers back to original characters
num_to_char = layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), mask_token=None, invert=True
)
```

Keras Dataset object

- TensorFlow makes available the tf.data API to create efficient input pipelines for machine learning models. Its core class is tf.data.Dataset.

- A Dataset object is an iterator: you can use it in a for loop. It will typically return batches of input data and labels. You can pass a Dataset directly to the fit() method of a Keras model.

- The Dataset class handles many key features that would otherwise be cumbersome to implement yourself, a in particular asynchronous data prefetching (preprocessing the next batch of data while the previous one is being handled by the model, which keeps execution flowing without interruptions).

- The class also exposes a functional-style API for modifying datasets.

Keras Dataset object

```
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = (
    train_dataset.map(
        encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
    )
    .batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)

validation_dataset = tf.data.Dataset.from_tensor_slices((x_valid, y_valid))
validation_dataset = (
    validation_dataset.map(
        encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
    )
    .batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)
```

https://keras.io/examples/vision/captcha_ocr/

CTC Loss: Keras Custom Layers

```python
class CTCLayer(layers.Layer):
    def __init__(self, name=None):
        super().__init__(name=name)
        self.loss_fn = keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred):
        # Compute the training-time loss value and add it
        # to the layer using `self.add_loss()`.
        batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
        input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
        label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

        input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
        label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

        loss = self.loss_fn(y_true, y_pred, input_length, label_length)
        self.add_loss(loss)

        # At test time, just return the computed predictions
        return y_pred
```

One of the central abstraction in Keras is the Layer class. A layer encapsulates both a state (the layer's "weights") and a transformation from inputs to outputs (a "call", the layer's forward pass).

https://keras.io/examples/vision/captcha_ocr/

Model

```python
input_img = layers.Input(
    shape=(img_width, img_height, 1), name="image", dtype="float32"
)
labels = layers.Input(name="label", shape=(None,), dtype="float32")

# First conv block
x = layers.Conv2D(
    32,
    (3, 3),
    activation="relu",
    kernel_initializer="he_normal",
    padding="same",
    name="Conv1",
)(input_img)
x = layers.MaxPooling2D((2, 2), name="pool1")(x)

# Second conv block
x = layers.Conv2D(
    64,
    (3, 3),
    activation="relu",
    kernel_initializer="he_normal",
    padding="same",
    name="Conv2",
)(x)
x = layers.MaxPooling2D((2, 2), name="pool2")(x)
```

```python
# We have used two max pool with pool size and strides 2.
# Hence, downsampled feature maps are 4x smaller. The number of
# filters in the last layer is 64. Reshape accordingly before
# passing the output to the RNN part of the model
new_shape = ((img_width // 4), (img_height // 4) * 64)
x = layers.Reshape(target_shape=new_shape, name="reshape")(x)
x = layers.Dense(64, activation="relu", name="dense1")(x)
x = layers.Dropout(0.2)(x)

# RNNs
x = layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.25))(x)
x = layers.Bidirectional(layers.LSTM(64, return_sequences=True, dropout=0.25))(x)

# Output layer
x = layers.Dense(
    len(char_to_num.get_vocabulary()) + 1, activation="softmax", name="dense2"
)(x)

# Add CTC layer for calculating CTC loss at each step
output = CTCLayer(name="ctc_loss")(labels, x)

# Define the model
model = keras.models.Model(
    inputs=[input_img, labels], outputs=output, name="ocr_model_v1"
)
# Optimizer
opt = keras.optimizers.Adam()
# Compile the model and return
model.compile(optimizer=opt)
```

model.fit(…)

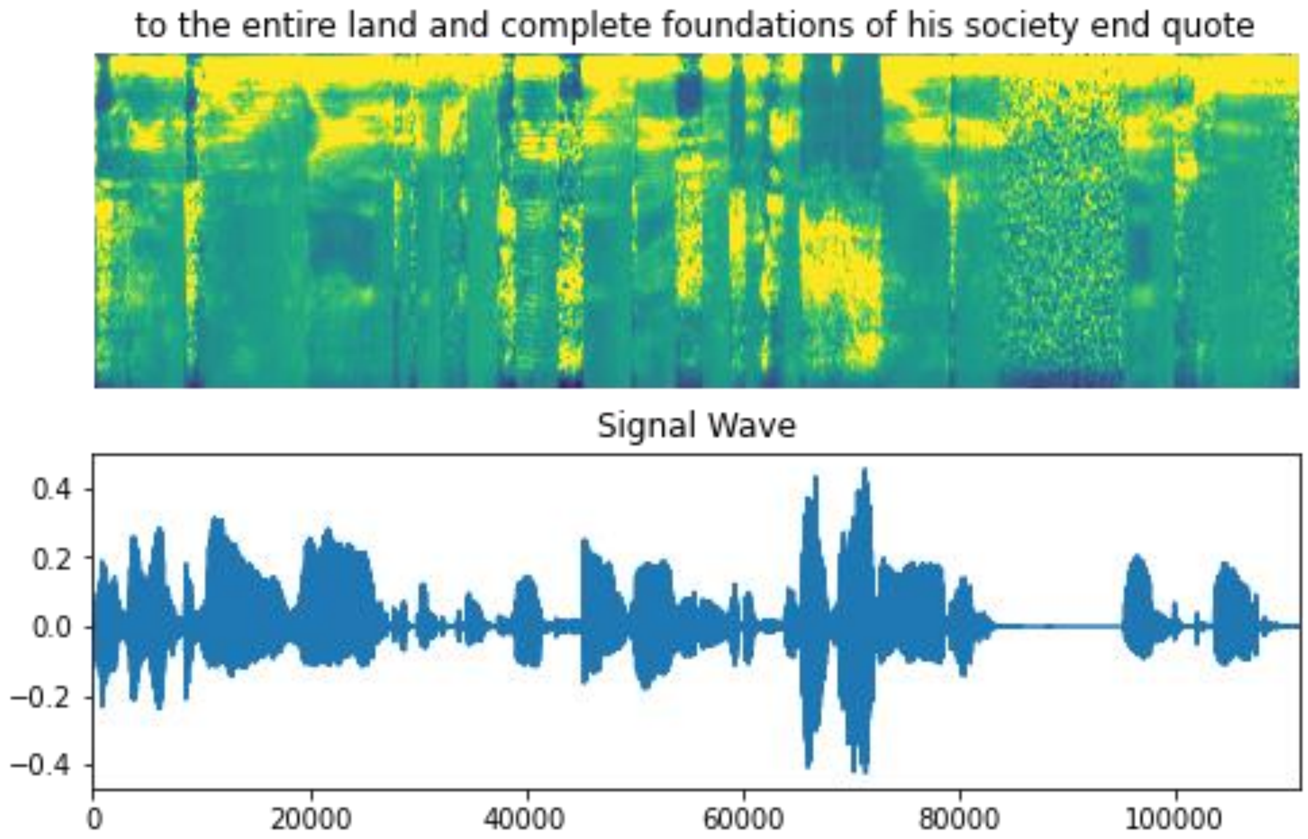
```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
image (InputLayer)              [(None, 200, 50, 1)] 0
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 200, 50, 32)  320         image[0][0]
__________________________________________________________________________________________________
pool1 (MaxPooling2D)            (None, 100, 25, 32)  0           Conv1[0][0]
__________________________________________________________________________________________________
Conv2 (Conv2D)                  (None, 100, 25, 64)  18496       pool1[0][0]
__________________________________________________________________________________________________
pool2 (MaxPooling2D)            (None, 50, 12, 64)   0           Conv2[0][0]
__________________________________________________________________________________________________
reshape (Reshape)               (None, 50, 768)      0           pool2[0][0]
__________________________________________________________________________________________________
dense1 (Dense)                  (None, 50, 64)       49216       reshape[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 50, 64)       0           dense1[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 50, 256)      197632      dropout[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 50, 128)      164352      bidirectional[0][0]
__________________________________________________________________________________________________
label (InputLayer)              [(None, None)]       0
__________________________________________________________________________________________________
dense2 (Dense)                  (None, 50, 20)       2580        bidirectional_1[0][0]
__________________________________________________________________________________________________
ctc_loss (CTCLayer)             (None, 50, 20)       0           label[0][0]
                                                                 dense2[0][0]
==================================================================================================
Total params: 432,596
Trainable params: 432,596
Non-trainable params: 0
```

Inference

```python
prediction_model = keras.models.Model(
    model.get_layer(name="image").input, model.get_layer(name="dense2").output
)
prediction_model.summary()

# A utility function to decode the output of the network
def decode_batch_predictions(pred):
    input_len = np.ones(pred.shape[0]) * pred.shape[1]
    # Use greedy search. For complex tasks, you can use beam search
    results = keras.backend.ctc_decode(pred, input_length=input_len, greedy=True)[0][0][
        :, :max_length
    ]
    # Iterate over the results and get back the text
    output_text = []
    for res in results:
        res = tf.strings.reduce_join(num_to_char(res)).numpy().decode("utf-8")
        output_text.append(res)
    return output_text


#  Let's check results on some validation samples
for batch in validation_dataset.take(1):
    batch_images = batch["image"]
    batch_labels = batch["label"]

    preds = prediction_model.predict(batch_images)
    pred_texts = decode_batch_predictions(preds)
```

Handwriting recognition problem: https://keras.io/examples/vision/handwriting_recognition/

Automatic Speech Recognition using CTC

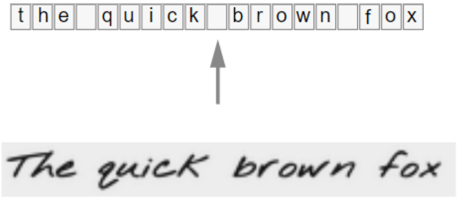


For details: https://keras.io/examples/audio/ctc_asr/

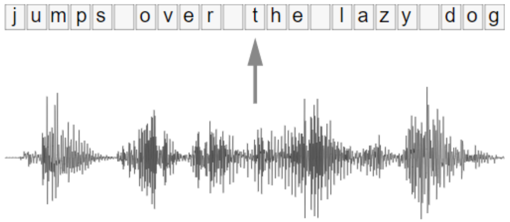
Appendix

Connectionist Temporal Classification

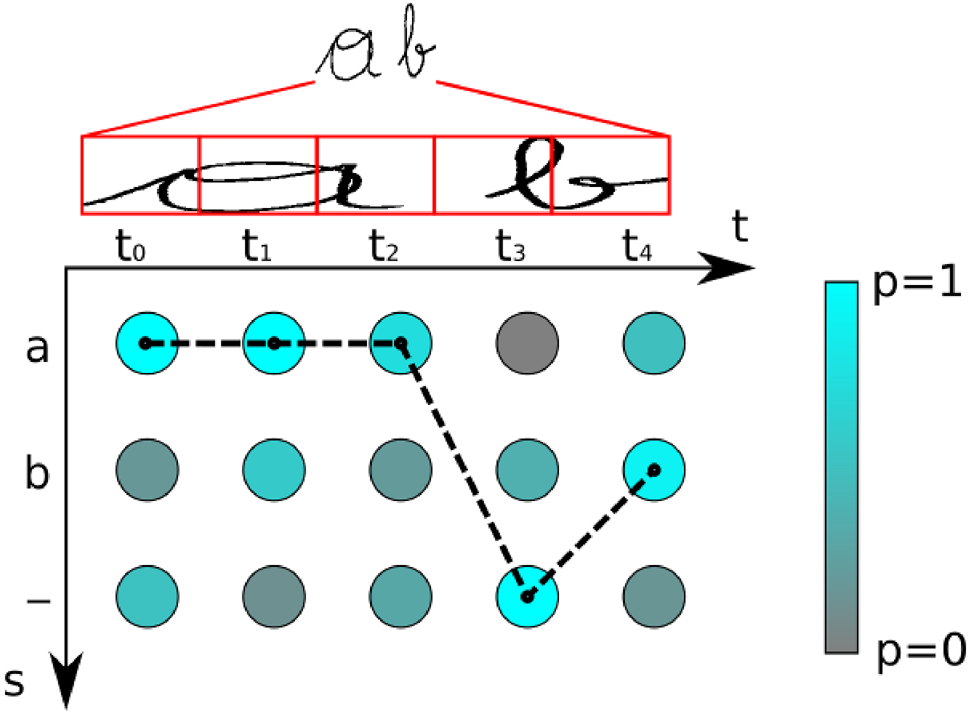
Connectionist Temporal Classification (CTC) is a way to get around not knowing the alignment between the input and the output. As we'll see, it's especially well suited to applications like speech and handwriting recognition.

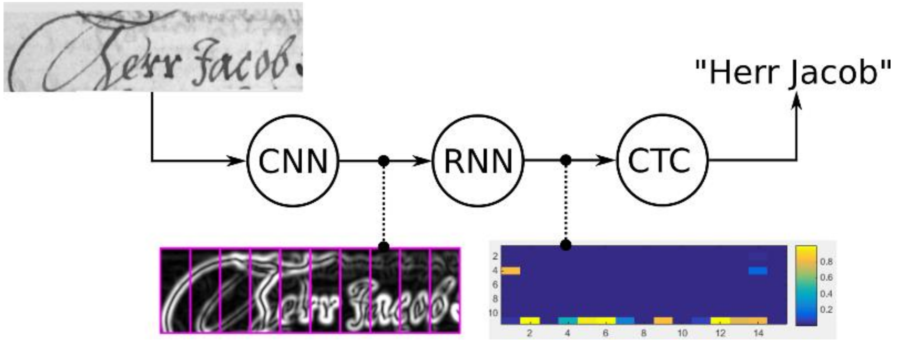


Handwriting recognition: The input can be (x, y) coordinates of a pen stroke or pixels in an image.



Speech recognition: The input can be a spectrogram or some other frequency based feature extractor.





https://distill.pub/2017/ctc/
https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c