# Keras

Lecture 6
Subir Varma

# Keras https://keras.io

- High Level API for TensorFLow
- Can run on CPU/GPU/TPU
- Built in support for ConvNets, RNNs and Transformers
- Supports arbitrary network architectures
- Can be freely used in commercial projects
- Over a million users

# A Keras Program (Model Setup)

```
import keras
keras.__version__
```

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Import Dataset
(already in Tensor form)

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Data Reshaping
+
Data Normalization

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Label Conversion from Sparse to Categorical (1-Hot Encoded)

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Define the Network

```
network.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Compile the Model

Learning Rate specified here

# Today's Class → How can I import Raw Data (Image, Text, Tabular)?

```
import keras
keras.__version__
```

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

```
network.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
history = network.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)
```

# In Future Classes

```
import keras
keras.__version__
```

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

How can I feed in 2D/3D Tensor Data directly without Reshaping?

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

How Can I define Other Types of Networks (ConvNets, RNNs, Transformers)?

```
network.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

- What are some Faster Optimizers?
- Specifying User Defined Metrics

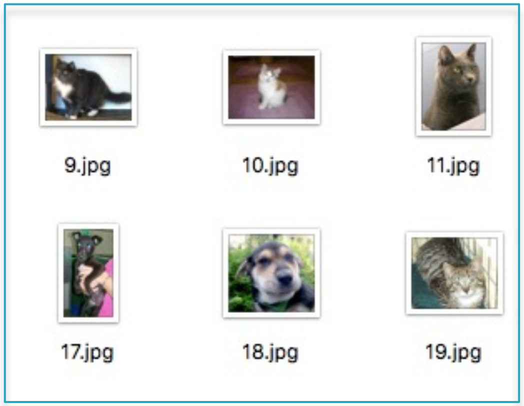
```
history = network.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)
```

How long should I run the Model?
How to interrupt the Execution?

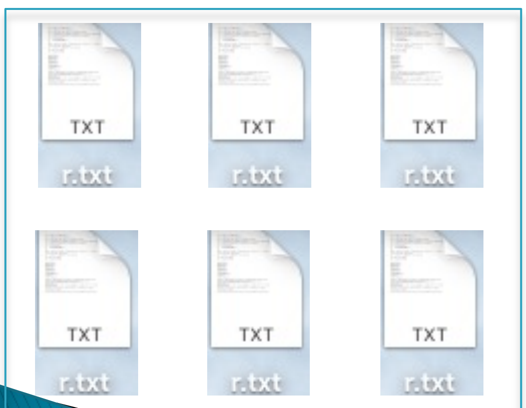
# Ingesting Data using Keras

# Data Ingestion

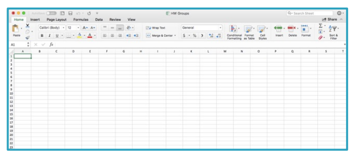
Raw Image Data




Raw Text Data




Raw Tabular Data



How to convert all of these into Tensors so that they can be fed into a DL Network?

# Image Data

(Section 8.2 in Chollet)

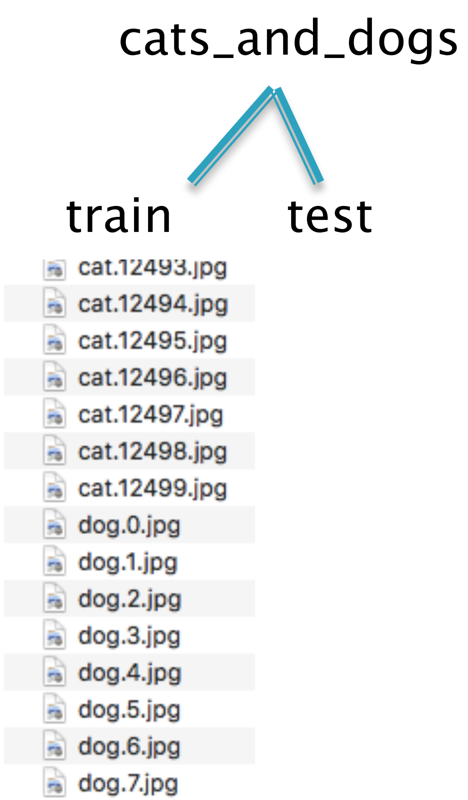
# Processing Raw Image Data

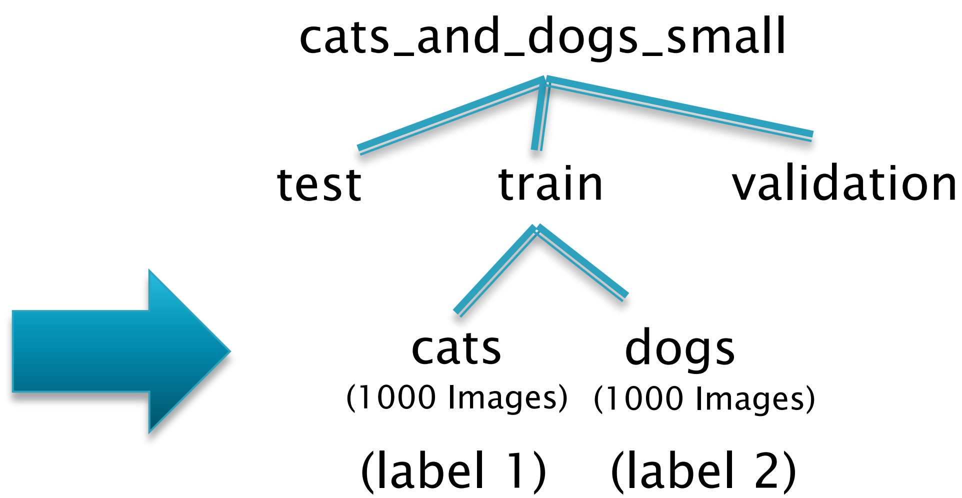
- Read the picture files.
- Decode the JPEG content to RBG grids of pixels.
- Convert these into floating point tensors.

- Keras has utilities to take care of these steps automatically.
  - Keras has a module with image processing helper tools, located at keras.preprocessing.image.
- In particular, it contains the utility image_dataset_from_directory that can turn image files on disk into batches of pre-processed tensors.

# Cats/Dogs Classification:

50,000 images evenly divided between cats and dogs




Create a hierarchical directory with
each image category in a separate directory

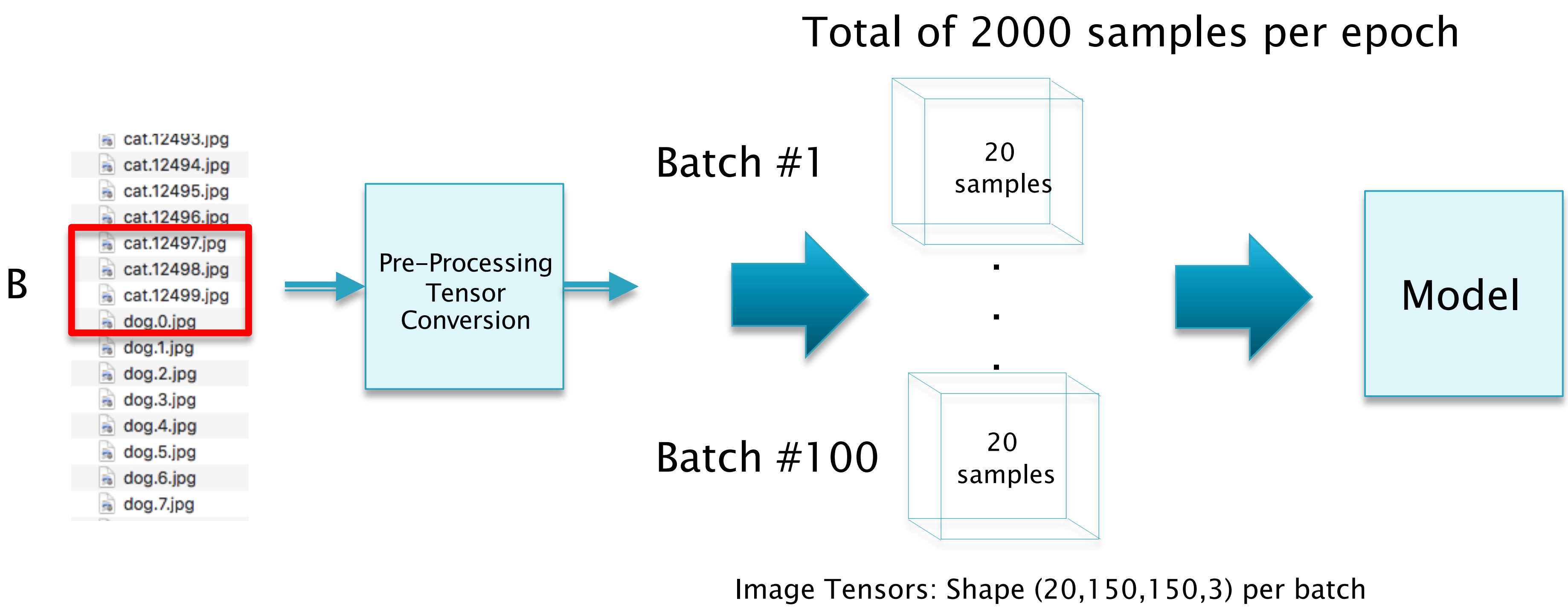
# Creating Training and Validation Datasets

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(150, 150),
    batch_size=20)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(150, 150),
    batch_size=20)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

- Converts the jpg images to the RGB format, and stores them as tensors of shape (150,150,3)
- Converts these into floating point tensors
- Crops the images so that they all have a size of (150,150) pixels
- Creates batches of image data (of size 20 in this case) and stores them in tensors of shape (20,150,150,3), which are then fed into the model during execution.
- Creates Labels for each image. This is done by assigning a different one-hot label to images belonging to different directories.

# Training


Total of 2000 samples per epoch
B
cat.12493.jpg
cat.12494.jpg
cat.12495.jpg
cat.12496.jpg
cat.12497.jpg
cat.12498.jpg
cat.12499.jpg
dog.0.jpg
dog.1.jpg
dog.2.jpg
dog.3.jpg
dog.4.jpg
dog.5.jpg
dog.6.jpg
dog.7.jpg
Pre-Processing Tensor Conversion
Batch #1
20 samples
Batch #100
20 samples
Model
Image Tensors: Shape (20,150,150,3) per batch

# Model

```
from keras.layers import Embedding, Flatten, Dense
from tensorflow.keras import optimizers

from keras import Model
from keras import layers
from keras import Input

input_tensor = Input(shape=(150,150,3,))
a = layers.Flatten()(input_tensor)
a = layers.Rescaling(1./255)(a)
layer_1 = layers.Dense(64, activation='relu')(a)
layer_2 = layers.Dense(64, activation='relu')(layer_1)
layer_3 = layers.Dense(64, activation='relu')(layer_2)
layer_4 = layers.Dense(64, activation='relu')(layer_3)
output_tensor = layers.Dense(1, activation='sigmoid')(layer_4)

model = Model(input_tensor, output_tensor)

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=1e-4),
              metrics=['acc'])

model.summary()
```

# Text Data

(Section 11.3.3 in Chollet)

# Feeding Text into the Neural Network

- Map 10,000 most frequently occurring words to integers → Each review becomes a vector (of variable length)
- Pad out the vectors with 0s so that they are of the same length
- Create Tensors of shape (samples, word_indices)
- Map each word_index to an embedding vector, so now the input Tensor has shape (samples, word_indices, embedding)

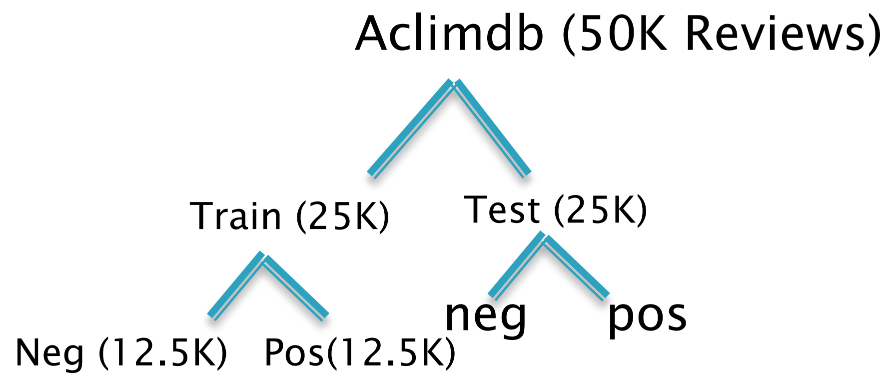
Keras contains the utility text_dataset_from_directory that can turn image files on disk into batches of pre-processed tensors.

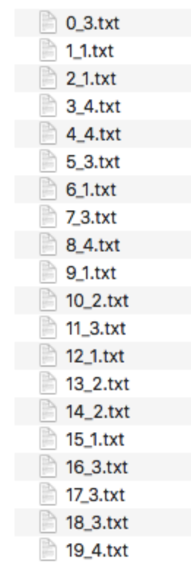
# IMDB: Move Review Database

- 50,000 Reviews
- 25,000 Reviews for Training, 25,000 for Test
- 50% negative, 50% positive reviews

# Raw Text Data

```python
import os

#imdb_dir = '/home/ubuntu/data/aclImdb'
imdb_dir = '/Users/subirvarma/handson-ml/datasets/aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```




12,500 Positive Reviews
12,500 Negative Reviews

# Create a Batched Dataset

```
from tensorflow import keras
batch_size = 32

train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=batch_size
)
val_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/val", batch_size=batch_size
)
test_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=batch_size
)
```

Result ➔

(Review 1, Label 1)
.
.
.
(Review 32, Label 32)

But Data still in Text strings

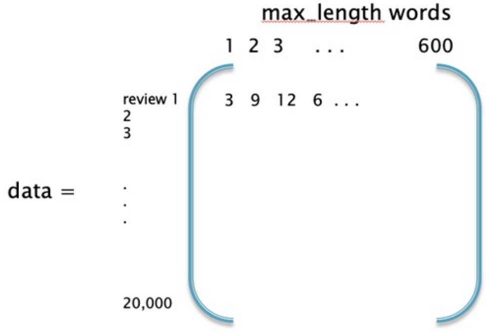
Each Review is paired up with the corresponding Label

# Text Vectorization: Text to Integers

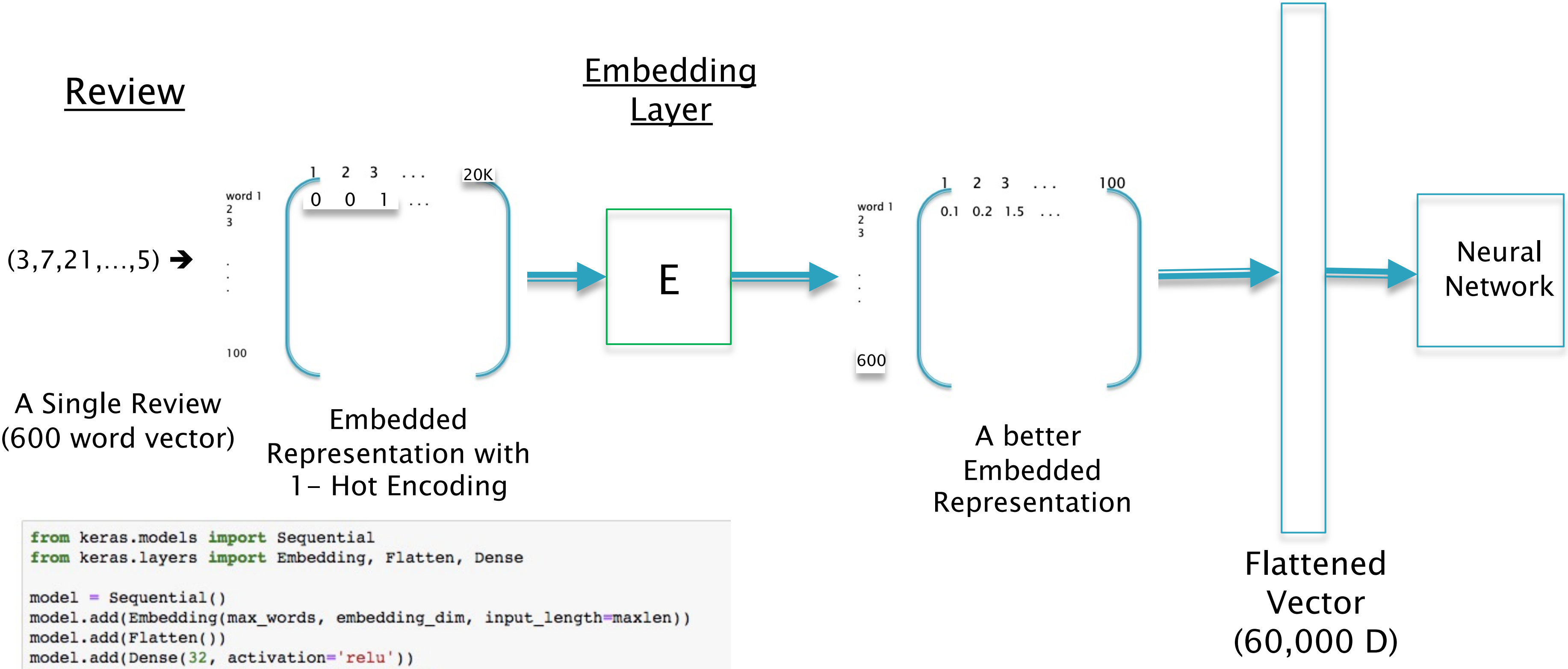
```
from tensorflow.keras import layers

max_length = 600
max_tokens = 20000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds = train_ds.map(lambda x, y: (text_vectorization(x), y))
int_val_ds = val_ds.map(lambda x, y: (text_vectorization(x), y))
int_test_ds = test_ds.map(lambda x, y: (text_vectorization(x), y))
```

Takes each review and converts it from text to integers.

- It does so by cutting of the number of words in the reviews to the top 20,000 most frequently occurring words (specified by the parameter *max_tokens*), and then mapping each word to an unique integer in the range 0 to 20,000 (after removing all punctuation).
- It furthermore truncates each review to a maximum of max_length = 600 words, and pads the reviews with less than 600 words with zeroes.

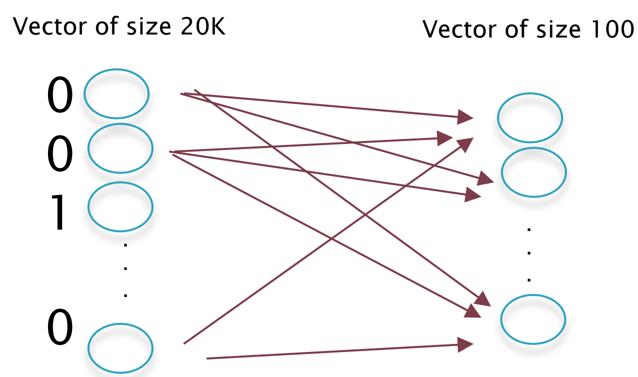


# Feeding Data into the Model (Trained Embedding)




```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

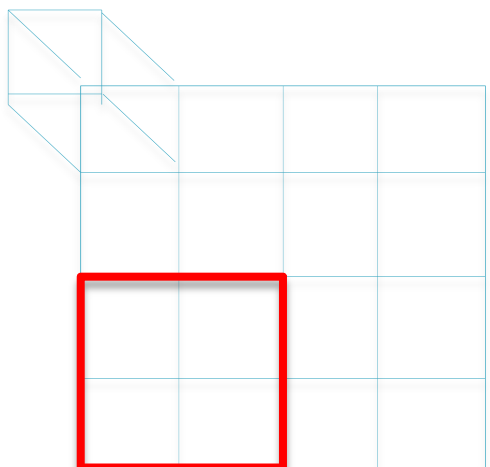
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_2 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```




E = Embedding Matrix

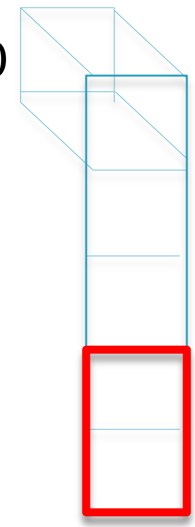
# Comparing Image and Text Representations

Depth = 3
(RGB)



3D Tensor

Depth = 100
(Embedding)



Word 1

Word 2

.
.
.

2D Tensor

| | | |
|---|---|---|
| Top Level: | Image | Sentence |
| 2nd Level : | Pixels | Words |
| 3rd Level: | RGB | Embedding |

# Tabular Data

# Keras Utilities for Tabular Data

- Structured Data Example:

https://keras.io/examples/structured_data/structured_data_classification_from_scratch/

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 1 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 3 | 0 | fixed | 0 |
| **1** | 67 | 1 | 4 | 160 | 286 | 0 | 2 | 108 | 1 | 1.5 | 2 | 3 | normal | 1 |
| **2** | 67 | 1 | 4 | 120 | 229 | 0 | 2 | 129 | 1 | 2.6 | 2 | 2 | reversible | 0 |
| **3** | 37 | 1 | 3 | 130 | 250 | 0 | 0 | 187 | 0 | 3.5 | 3 | 0 | normal | 0 |
| **4** | 41 | 0 | 2 | 130 | 204 | 0 | 2 | 172 | 0 | 1.4 | 1 | 0 | normal | 0 |

Number

Category

Cleveland Clinic Foundation Heart Disease Data

Predict this column

# Tabular Data

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 1 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 3 | 0 | fixed | 0 |
| **1** | 67 | 1 | 4 | 160 | 286 | 0 | 2 | 108 | 1 | 1.5 | 2 | 3 | normal | 1 |
| **2** | 67 | 1 | 4 | 120 | 229 | 0 | 2 | 129 | 1 | 2.6 | 2 | 2 | reversible | 0 |
| **3** | 37 | 1 | 3 | 130 | 250 | 0 | 0 | 187 | 0 | 3.5 | 3 | 0 | normal | 0 |
| **4** | 41 | 0 | 2 | 130 | 204 | 0 | 2 | 172 | 0 | 1.4 | 1 | 0 | normal | 0 |

| Column | Description | Feature Type |
|---|---|---|
| Age | Age in years | Numerical |
| Sex | (1 = male; 0 = female) | Categorical |
| CP | Chest pain type (0, 1, 2, 3, 4) | Categorical |
| Trestbpd | Resting blood pressure (in mm Hg on admission) | Numerical |
| Chol | Serum cholesterol in mg/dl | Numerical |
| FBS | fasting blood sugar in 120 mg/dl (1 = true; 0 = false) | Categorical |
| RestECG | Resting electrocardiogram results (0, 1, 2) | Categorical |
| Thalach | Maximum heart rate achieved | Numerical |
| Exang | Exercise induced angina (1 = yes; 0 = no) | Categorical |
| Oldpeak | ST depression induced by exercise relative to rest | Numerical |
| Slope | Slope of the peak exercise ST segment | Numerical |
| CA | Number of major vessels (0-3) colored by fluoroscopy | Both numerical & categorical |
| Thal | 3 = normal; 6 = fixed defect; 7 = reversible defect | Categorical |
| Target | Diagnosis of heart disease (1 = true; 0 = false) | Target |

# Creating the Dataset

We start by downloading the data and storing it in a Pandas dataframe.

```
file_url = "http://storage.googleapis.com/download.tensorflow.org/data/heart.csv"
dataframe = pd.read_csv(file_url)
dataframe.shape
```

```
(303, 14)
```

The data is randomly split in validation and training sets

```
val_dataframe = dataframe.sample(frac=0.2, random_state=1337)
train_dataframe = dataframe.drop(val_dataframe.index)

print(
    "Using %d samples for training and %d for validation"
    % (len(train_dataframe), len(val_dataframe))
)
```

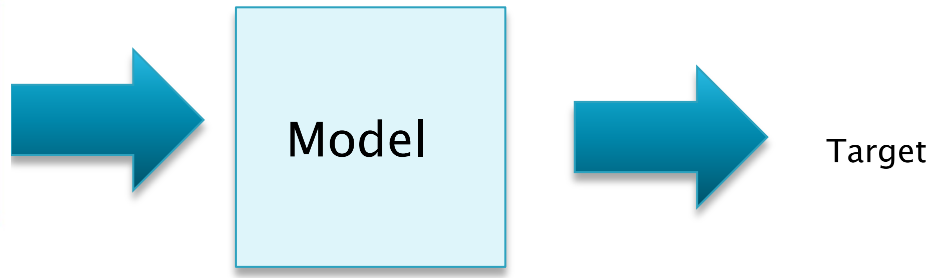
```
Using 242 samples for training and 61 for validation
```

The following procedure invokes the *Dataset.from_tensor_slices* procedure in order to create labels for each input and pair it with the rest of the data in each row.
This results in the formation of the training and validation datasets.

```
def dataframe_to_dataset(dataframe):
    dataframe = dataframe.copy()
    labels = dataframe.pop("target")
    ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
    ds = ds.shuffle(buffer_size=len(dataframe))
    return ds

train_ds = dataframe_to_dataset(train_dataframe)
val_ds = dataframe_to_dataset(val_dataframe)
```

# Training

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 1 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 3 | 0 | fixed | 0 |
| **1** | 67 | 1 | 4 | 160 | 286 | 0 | 2 | 108 | 1 | 1.5 | 2 | 3 | normal | 1 |
| **2** | 67 | 1 | 4 | 120 | 229 | 0 | 2 | 129 | 1 | 2.6 | 2 | 2 | reversible | 0 |
| **3** | 37 | 1 | 3 | 130 | 250 | 0 | 0 | 187 | 0 | 3.5 | 3 | 0 | normal | 0 |
| **4** | 41 | 0 | 2 | 130 | 204 | 0 | 2 | 172 | 0 | 1.4 | 1 | 0 | normal | 0 |



# Time Series Analysis

# Time Series Analysis

Input 2D Tensor of shape (9,14)

| 1 | Date Time | p (mbar) | T (degC) | Tpot (K) | Tdew (degC) | rh (%) | VPmax (mba | VPact (mbar | VPdef (mbar | sh (g/kg) | H2OC (mmol | rho (g/m**3 | wv (m/s) | max. wv (m/ | wd (deg) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 01.01.2009 00:10:00 | 996.52 | -8.02 | 265.4 | -8.9 | 93.3 | 3.33 | 3.11 | 0.22 | 1.94 | 3.12 | 1307.75 | 1.03 | 1.75 | 152.3 |
| 3 | 01.01.2009 00:20:00 | 996.57 | -8.41 | 265.01 | -9.28 | 93.4 | 3.23 | 3.02 | 0.21 | 1.89 | 3.03 | 1309.8 | 0.72 | 1.5 | 136.1 |
| 4 | 01.01.2009 00:30:00 | 996.53 | -8.51 | 264.91 | -9.31 | 93.9 | 3.21 | 3.01 | 0.2 | 1.88 | 3.02 | 1310.24 | 0.19 | 0.63 | 171.6 |
| 5 | 01.01.2009 00:40:00 | 996.51 | -8.31 | 265.12 | -9.07 | 94.2 | 3.26 | 3.07 | 0.19 | 1.92 | 3.08 | 1309.19 | 0.34 | 0.5 | 198 |
| 6 | 01.01.2009 00:50:00 | 996.51 | -8.27 | 265.15 | -9.04 | 94.1 | 3.27 | 3.08 | 0.19 | 1.92 | 3.09 | 1309 | 0.32 | 0.63 | 214.3 |
| 7 | 01.01.2009 01:00:00 | 996.5 | -8.05 | 265.38 | -8.78 | 94.4 | 3.33 | 3.14 | 0.19 | 1.96 | 3.15 | 1307.86 | 0.21 | 0.63 | 192.7 |
| 8 | 01.01.2009 01:10:00 | 996.5 | -7.62 | 265.81 | -8.3 | 94.8 | 3.44 | 3.26 | 0.18 | 2.04 | 3.27 | 1305.68 | 0.18 | 0.63 | 166.5 |
| 9 | 01.01.2009 01:20:00 | 996.5 | -7.62 | 265.81 | -8.36 | 94.4 | 3.44 | 3.25 | 0.19 | 2.03 | 3.26 | 1305.69 | 0.19 | 0.5 | 118.6 |
| 10 | 01.01.2009 01:30:00 | 996.5 | -7.91 | 265.52 | -8.73 | 93.8 | 3.36 | 3.15 | 0.21 | 1.97 | 3.16 | 1307.17 | 0.28 | 0.75 | 188.5 |
| 11 | 01.01.2009 01:40:00 | 996.53 | -8.43 | 264.99 | -9.34 | 93.1 | 3.23 | 3 | 0.22 | 1.88 | 3.02 | 1309.85 | 0.59 | 0.88 | 185 |
| 12 | 01.01.2009 01:50:00 | 996.62 | -8.76 | 264.66 | -9.66 | 93.1 | 3.14 | 2.93 | 0.22 | 1.83 | 2.94 | 1311.64 | 0.45 | 0.88 | 183.2 |
| 13 | 01.01.2009 02:00:00 | 996.62 | -8.88 | 264.54 | -9.77 | 93.2 | 3.12 | 2.9 | 0.21 | 1.81 | 2.91 | 1312.25 | 0.25 | 0.63 | 190.3 |
| 14 | [illegible] | [illegible] | -8.85 | 264.57 | -9.7 | 93.5 | 3.12 | 2.92 | 0.2 | 1.82 | 2.93 | 1312.11 | 0.16 | 0.5 | 158.3 |
| 15 | [illegible] | [illegible] | -8.83 | 264.58 | -9.68 | 93.5 | 3.13 | 2.92 | 0.2 | 1.83 | 2.93 | 1312.15 | 0.36 | 0.63 | 184.8 |
| 16 | [illegible] | [illegible] | -8.66 | 264.74 | -9.46 | 93.9 | 3.17 | 2.98 | 0.19 | 1.86 | 2.99 | 1311.37 | 0.33 | 0.75 | 155.9 |
| 17 | 01.01.2009 02:40:00 | 996.81 | -8.66 | 264.74 | -9.5 | 93.6 | 3.17 | 2.97 | 0.2 | 1.85 | 2.98 | 1311.38 | 0.07 | 0.5 | 272.4 |
| 18 | 01.01.2009 02:50:00 | 996.86 | -8.7 | 264.7 | -9.55 | 93.5 | 3.16 | 2.95 | 0.21 | 1.85 | 2.96 | 1311.64 | 0.32 | 0.63 | 219.2 |
| 19 | 01.01.2009 03:00:00 | 996.84 | -8.81 | 264.59 | -9.66 | 93.5 | 3.13 | 2.93 | 0.2 | 1.83 | 2.94 | 1312.18 | 0.18 | 0.63 | 167.2 |
| 20 | 01.01.2009 03:10:00 | 996.87 | -8.84 | 264.56 | -9.69 | 93.5 | 3.13 | 2.92 | 0.2 | 1.83 | 2.93 | 1312.37 | 0.07 | 0.25 | 129.3 |
| 21 | 01.01.2009 03:20:00 | 996.97 | -8.94 | 264.45 | -9.82 | 93.3 | 3.1 | 2.89 | 0.21 | 1.81 | 2.9 | 1313.01 | 0.1 | 0.63 | 115.3 |
| 22 | 01.01.2009 03:30:00 | 997.08 | -8.94 | 264.44 | -9.8 | 93.4 | 3.1 | 2.9 | 0.2 | 1.81 | 2.9 | 1313.15 | 0.3 | 0.75 | 149.3 |
| 23 | 01.01.2009 03:40:00 | 997.1 | -8.86 | 264.52 | -9.76 | 93.1 | 3.12 | 2.9 | 0.22 | 1.81 | 2.91 | 1312.78 | 0.29 | 0.75 | 149.7 |
| 24 | 01.01.2009 03:50:00 | 997.06 | -8.99 | 264.39 | -9.99 | 92.4 | 3.09 | 2.85 | 0.23 | 1.78 | 2.86 | 1313.39 | 0.12 | 0.63 | 231.7 |
| 25 | 01.01.2009 04:00:00 | 996.99 | -9.05 | 264.34 | -10.02 | 92.6 | 3.07 | 2.85 | 0.23 | 1.78 | 2.85 | 1313.61 | 0.1 | 0.38 | 240 |
| 26 | 01.01.2009 04:10:00 | 997.05 | -9.23 | 264.15 | -10.25 | 92.2 | 3.03 | 2.79 | 0.24 | 1.74 | 2.8 | 1314.62 | 0.1 | 0.38 | 203.9 |
| 27 | 01.01.2009 04:20:00 | 997.11 | -9.49 | 263.89 | -10.54 | 92 | 2.97 | 2.73 | 0.24 | 1.71 | 2.74 | 1316.02 | 0.34 | 0.75 | 159.7 |
| 28 | 01.01.2009 04:30:00 | 997.19 | -9.5 | 263.87 | -10.51 | 92.3 | 2.97 | 2.74 | 0.23 | 1.71 | 2.75 | 1316.16 | 0.43 | 0.88 | 66.16 |
| 29 | 01.01.2009 04:40:00 | 997.24 | -9.35 | 264.02 | -10.29 | 92.8 | 3 | 2.79 | 0.22 | 1.74 | 2.79 | 1315.47 | 0.4 | 0.88 | 105 |
| 30 | 01.01.2009 04:50:00 | 997.37 | -9.47 | 263.89 | -10.46 | 92.4 | 2.97 | 2.75 | 0.23 | 1.72 | 2.75 | 1316.25 | 0.37 | 0.75 | 125.8 |
| 31 | 01.01.2009 05:00:00 | 997.46 | -9.63 | 263.72 | -10.65 | 92.2 | 2.94 | 2.71 | 0.23 | 1.69 | 2.71 | 1317.19 | 0.4 | 0.88 | 157 |
| 32 | 01.01.2009 05:10:00 | 997.43 | -9.67 | 263.68 | -10.63 | 92.6 | 2.93 | 2.71 | 0.22 | 1.69 | 2.72 | 1317.35 | 0.36 | 0.75 | 132.5 |
| 33 | 01.01.2009 05:20:00 | 997.42 | -9.68 | 263.67 | -10.73 | 92 | 2.92 | 2.69 | 0.23 | 1.68 | 2.7 | 1317.4 | 0.09 | 0.5 | 143.2 |
| 34 | 01.01.2009 05:30:00 | 997.53 | -9.9 | 263.45 | -10.98 | 91.7 | 2.87 | 2.64 | 0.24 | 1.64 | 2.64 | 1318.68 | 0.29 | 1 | 72.5 |
| 35 | 01.01.2009 05:40:00 | 997.6 | -9.91 | 263.43 | -10.9 | 92.4 | 2.87 | 2.65 | 0.22 | 1.66 | 2.66 | 1318.81 | 0.5 | 1 | 60.72 |

jena_climate_2009_2016

Output Label

14 quantities recorded every 10 minutes from 2009-2016

# Time Series Analysis

Input Batch: 3D Tensor of shape (2,9,14)

| 1 | Date Time | p (mbar) | T (degC) | Tpot (K) | Tdew (degC) | rh (%) | VPmax (mba | VPact (mbar | VPdef (mbar | sh (g/kg) | H2OC (mmol | rho (g/m**3 | wv (m/s) | max. wv (m/ | wd (deg) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 01.01.2009 00:10:00 | 996.52 | -8.02 | 265.4 | -8.9 | 93.3 | 3.33 | 3.11 | 0.22 | 1.94 | 3.12 | 1307.75 | 1.03 | 1.75 | 152.3 |
| 3 | 01.01.2009 00:20:00 | 996.57 | -8.41 | 265.01 | -9.28 | 93.4 | 3.23 | 3.02 | 0.21 | 1.89 | 3.03 | 1309.8 | 0.72 | 1.5 | 136.1 |
| 4 | 01.01.2009 00:30:00 | 996.53 | -8.51 | 264.91 | -9.31 | 93.9 | 3.21 | 3.01 | 0.2 | 1.88 | 3.02 | 1310.24 | 0.19 | 0.63 | 171.6 |
| 5 | 01.01.2009 00:40:00 | 996.51 | -8.31 | 265.12 | -9.07 | 94.2 | 3.26 | 3.07 | 0.19 | 1.92 | 3.08 | 1309.19 | 0.34 | 0.5 | 198 |
| 6 | 01.01.2009 00:50:00 | 996.51 | -8.27 | 265.15 | -9.04 | 94.1 | 3.27 | 3.08 | 0.19 | 1.92 | 3.09 | 1309 | 0.32 | 0.63 | 214.3 |
| 7 | 01.01.2009 01:00:00 | 996.5 | -8.05 | 265.38 | -8.78 | 94.4 | 3.33 | 3.14 | 0.19 | 1.96 | 3.15 | 1307.86 | 0.21 | 0.63 | 192.7 |
| 8 | 01.01.2009 01:10:00 | 996.5 | -7.62 | 265.81 | -8.3 | 94.8 | 3.44 | 3.26 | 0.18 | 2.04 | 3.27 | 1305.68 | 0.18 | 0.63 | 166.5 |
| 9 | 01.01.2009 01:20:00 | 996.5 | -7.62 | 265.81 | -8.36 | 94.4 | 3.44 | 3.25 | 0.19 | 2.03 | 3.26 | 1305.69 | 0.19 | 0.5 | 118.6 |
| 10 | 01.01.2009 01:30:00 | 996.5 | -7.91 | 265.52 | -8.73 | 93.8 | 3.36 | 3.15 | 0.21 | 1.97 | 3.16 | 1307.17 | 0.28 | 0.75 | 188.5 |
| 11 | 01.01.2009 01:40:00 | 996.53 | -8.43 | 264.99 | -9.34 | 93.1 | 3.23 | 3 | 0.22 | 1.88 | 3.02 | 1309.85 | 0.59 | 0.88 | 185 |
| 12 | 01.01.2009 01:50:00 | 996.62 | -8.76 | 264.66 | -9.66 | 93.1 | 3.14 | 2.93 | 0.22 | 1.83 | 2.94 | 1311.64 | 0.45 | 0.88 | 183.2 |
| 13 | 01.01.2009 02:00:00 | 996.62 | -8.88 | 264.54 | -9.77 | 93.2 | 3.12 | 2.9 | 0.21 | 1.81 | 2.91 | 1312.25 | 0.25 | 0.63 | 190.3 |
| 14 | 01.01.2009 02:10:00 | 996.63 | -8.85 | 264.57 | -9.7 | 93.5 | 3.12 | 2.92 | 0.2 | 1.82 | 2.93 | 1312.11 | 0.16 | 0.5 | 158.3 |
| 15 | 01.01.2009 02:20:00 | 996.74 | -8.83 | 264.58 | -9.68 | 93.5 | 3.13 | 2.92 | 0.2 | 1.83 | 2.93 | 1312.15 | 0.36 | 0.63 | 184.8 |
| 16 | 01.01.2009 02:30:00 | 996.81 | -8.66 | 264.74 | -9.46 | 93.9 | 3.17 | 2.98 | 0.19 | 1.86 | 2.99 | 1311.37 | 0.33 | 0.75 | 155.9 |
| 17 | 01.01.2009 02:40:00 | 996.81 | -8.66 | 264.74 | -9.5 | 93.6 | 3.17 | 2.97 | 0.2 | 1.85 | 2.98 | 1311.38 | 0.07 | 0.5 | 272.4 |
| 18 | 01.01.2009 02:50:00 | 996.86 | -8.7 | 264.7 | -9.55 | 93.5 | 3.16 | 2.95 | 0.21 | 1.85 | 2.96 | 1311.64 | 0.32 | 0.63 | 219.2 |
| 19 | 01.01.2009 03:00:00 | 996.84 | -8.81 | 264.59 | -9.66 | 93.5 | 3.13 | 2.93 | 0.2 | 1.83 | 2.94 | 1312.18 | 0.18 | 0.63 | 167.2 |
| 20 | 01.01.2009 03:10:00 | 996.87 | -8.84 | 264.56 | -9.69 | 93.5 | 3.13 | 2.92 | 0.2 | 1.83 | 2.93 | 1312.37 | 0.07 | 0.25 | 129.3 |
| 21 | 01.01.2009 03:20:00 | 996.97 | -8.94 | 264.45 | -9.82 | 93.3 | 3.1 | 2.89 | 0.21 | 1.81 | 2.9 | 1313.01 | 0.1 | 0.63 | 115.3 |
| 22 | 01.01.2009 03:30:00 | 997.08 | -8.94 | 264.44 | -9.8 | 93.4 | 3.1 | 2.9 | 0.2 | 1.81 | 2.9 | 1313.15 | 0.3 | 0.75 | 149.3 |
| 23 | 01.01.2009 03:40:00 | 997.1 | -8.86 | 264.52 | -9.76 | 93.1 | 3.12 | 2.9 | 0.22 | 1.81 | 2.91 | 1312.78 | 0.29 | 0.75 | 149.7 |
| 24 | 01.01.2009 03:50:00 | 997.06 | -8.99 | 264.39 | -9.99 | 92.4 | 3.09 | 2.85 | 0.23 | 1.78 | 2.86 | 1313.39 | 0.12 | 0.63 | 231.7 |
| 25 | 01.01.2009 04:00:00 | 996.99 | -9.05 | 264.34 | -10.02 | 92.6 | 3.07 | 2.85 | 0.23 | 1.78 | 2.85 | 1313.61 | 0.1 | 0.38 | 240 |
| 26 | 01.01.2009 04:10:00 | 997.05 | -9.23 | 264.15 | -10.25 | 92.2 | 3.03 | 2.79 | 0.24 | 1.74 | 2.8 | 1314.62 | 0.1 | 0.38 | 203.9 |
| 27 | 01.01.2009 04:20:00 | 997.11 | -9.49 | 263.89 | -10.54 | 92 | 2.97 | 2.73 | 0.24 | 1.71 | 2.74 | 1316.02 | 0.34 | 0.75 | 159.7 |
| 28 | 01.01.2009 04:30:00 | 997.19 | -9.5 | 263.87 | -10.51 | 92.3 | 2.97 | 2.74 | 0.23 | 1.71 | 2.75 | 1316.16 | 0.43 | 0.88 | 66.16 |
| 29 | 01.01.2009 04:40:00 | 997.24 | -9.35 | 264.02 | -10.29 | 92.8 | 3 | 2.79 | 0.22 | 1.74 | 2.79 | 1315.47 | 0.4 | 0.88 | 105 |
| 30 | 01.01.2009 04:50:00 | 997.37 | -9.47 | 263.89 | -10.46 | 92.4 | 2.97 | 2.75 | 0.23 | 1.72 | 2.75 | 1316.25 | 0.37 | 0.75 | 125.8 |
| 31 | 01.01.2009 05:00:00 | 997.46 | -9.63 | 263.72 | -10.65 | 92.2 | 2.94 | 2.71 | 0.23 | 1.69 | 2.71 | 1317.19 | 0.4 | 0.88 | 157 |
| 32 | 01.01.2009 05:10:00 | 997.43 | -9.67 | 263.68 | -10.63 | 92.6 | 2.93 | 2.71 | 0.22 | 1.69 | 2.72 | 1317.35 | 0.36 | 0.75 | 132.5 |
| 33 | 01.01.2009 05:20:00 | 997.42 | -9.68 | 263.67 | -10.73 | 92 | 2.92 | 2.69 | 0.23 | 1.68 | 2.7 | 1317.4 | 0.09 | 0.5 | 143.2 |
| 34 | 01.01.2009 05:30:00 | 997.53 | -9.9 | 263.45 | -10.98 | 91.7 | 2.87 | 2.64 | 0.24 | 1.64 | 2.64 | 1318.68 | 0.29 | 1 | 72.5 |
| 35 | 01.01.2009 05:40:00 | 997.6 | -9.91 | 263.43 | -10.9 | 92.4 | 2.87 | 2.65 | 0.22 | 1.66 | 2.66 | 1318.81 | 0.5 | 1 | 60.72 |

jena_climate_2009_2016

14 quantities recorded every 10 minutes from 2009-2016

# Raw Tabular Data

Inspecting the data

```
import os

#data_dir = '/home/ubuntu/data/'
data_dir = '/Users/subirvarma/handson-ml/datasets/'
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')

f = open(fname)
data = f.read()
f.close()

lines = data.split('\n')
header = lines[0].split(',')
lines = lines[1:]

print(header)
print(len(lines))
```

Outputs a count of 420,551 lines of data

```
['Date Time', 'p (mbar)', 'T (degC)', 'Tpot (K)', 'Tdew (degC)', 'rh (%)', 'VPmax (mbar)', 'VPact (mbar)', 'VPdef (mb
ar)', 'sh (g/kg)', 'H2OC (mmol/mol)', 'rho (g/m**3)', 'wv (m/s)', 'max. wv (m/s)', 'wd (deg)']
420551
```

# Raw Tabular Data

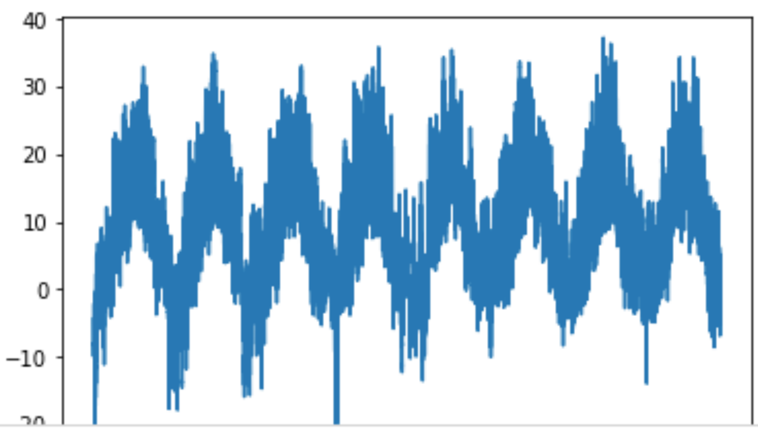
Let's convert all of these 420,551 lines of data into a Numpy array:

```
import numpy as np

float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[1:]]
    float_data[i, :] = values
```

Leave out the first row and the first column

For instance, here is the plot of temperature (in degrees Celsius) over time

```
from matplotlib import pyplot as plt

temp = float_data[:, 1]  # temperature (in degrees Cel
plt.plot(range(len(temp)), temp)
plt.show()
```



# Raw Tabular Data: Normalize the Data

Split samples into training validation and test

```
num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

```
num_train_samples: 210275
num_val_samples: 105137
num_test_samples: 105139
```

Data Normalization

```
mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std
```

# Dataset for Time Series Analysis

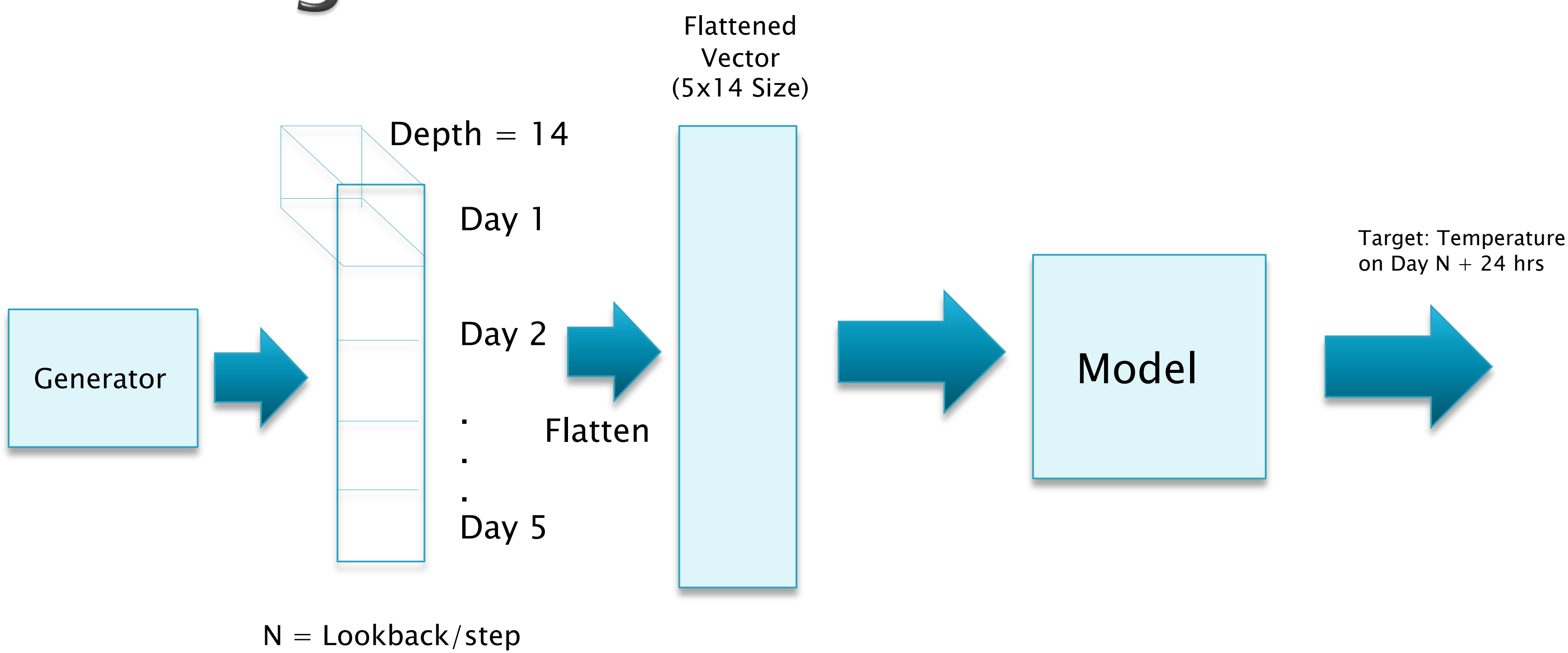
Sample once every hour

Equivalent to 5 days of measurements

Try to predict temperature 1 day into the future

```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256

train_dataset = tensorflow.keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = tensorflow.keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = tensorflow.keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

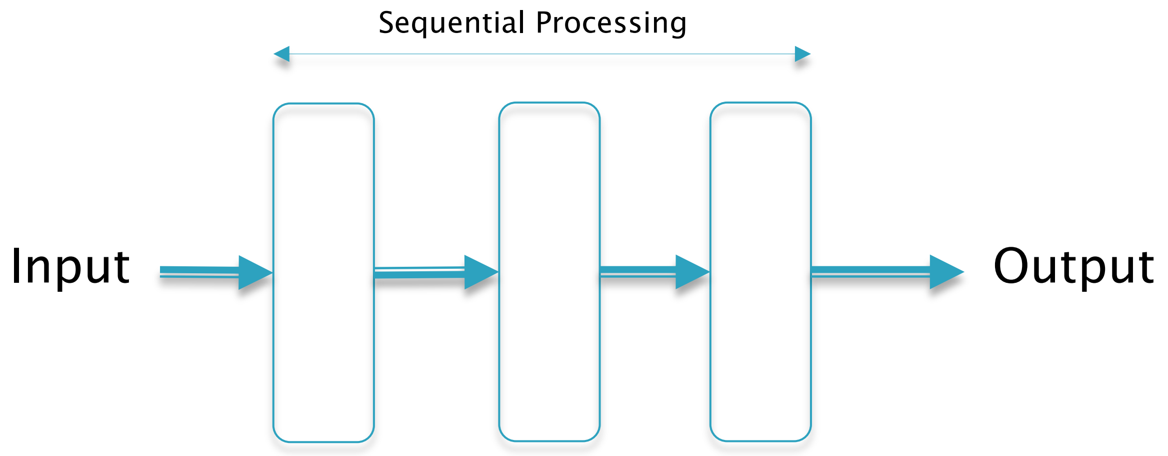
# Training




```python
from keras.models import Sequential
from keras import layers
from tensorflow.keras.optimizers import RMSprop

from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Flatten()(inputs)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset)
```

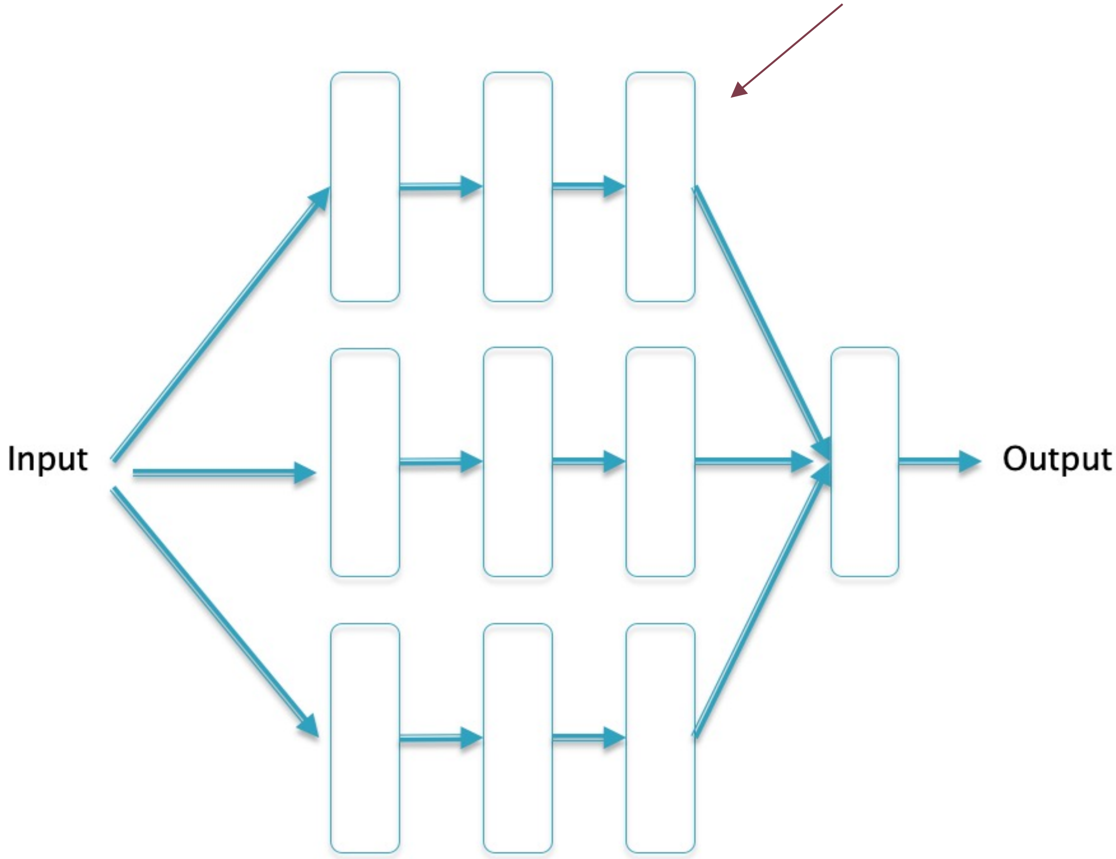
# Network Topologies for Deep Networks

# Sequential Processing



# Parallel Processing: Model Ensembles

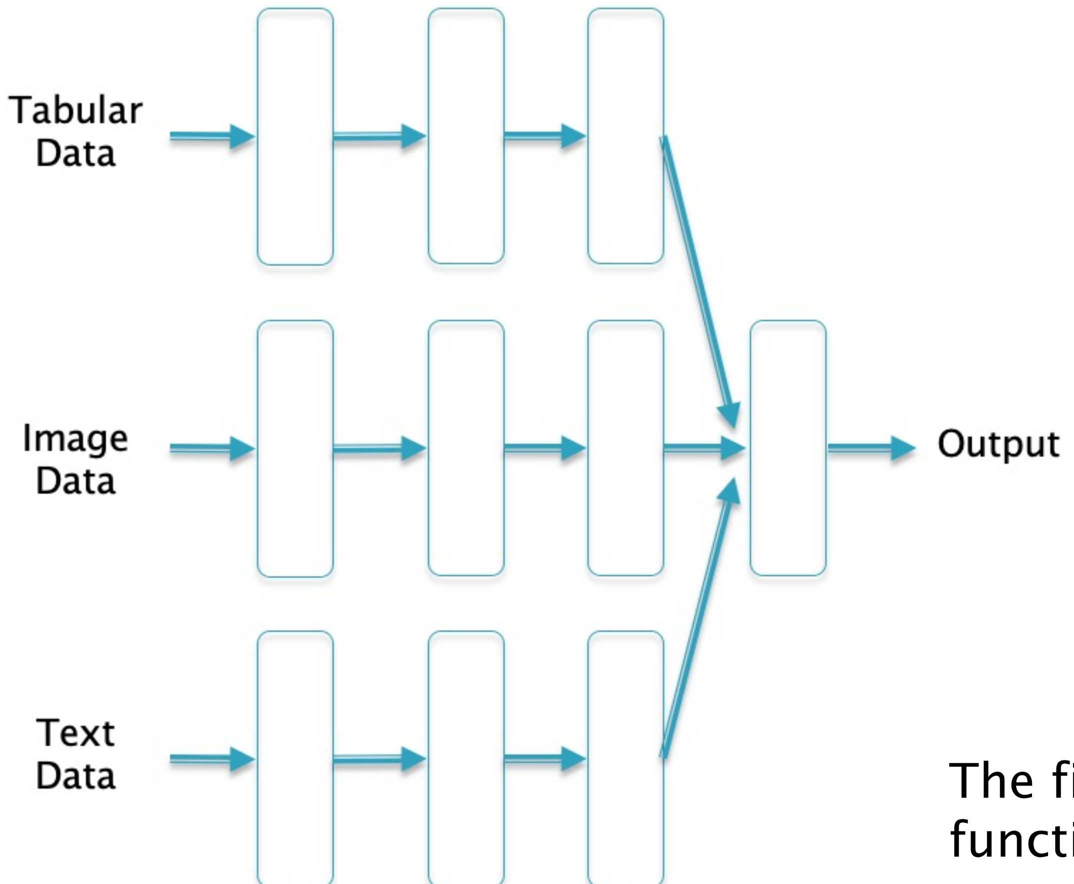
Multiple Predictions




Example: Majority Vote models

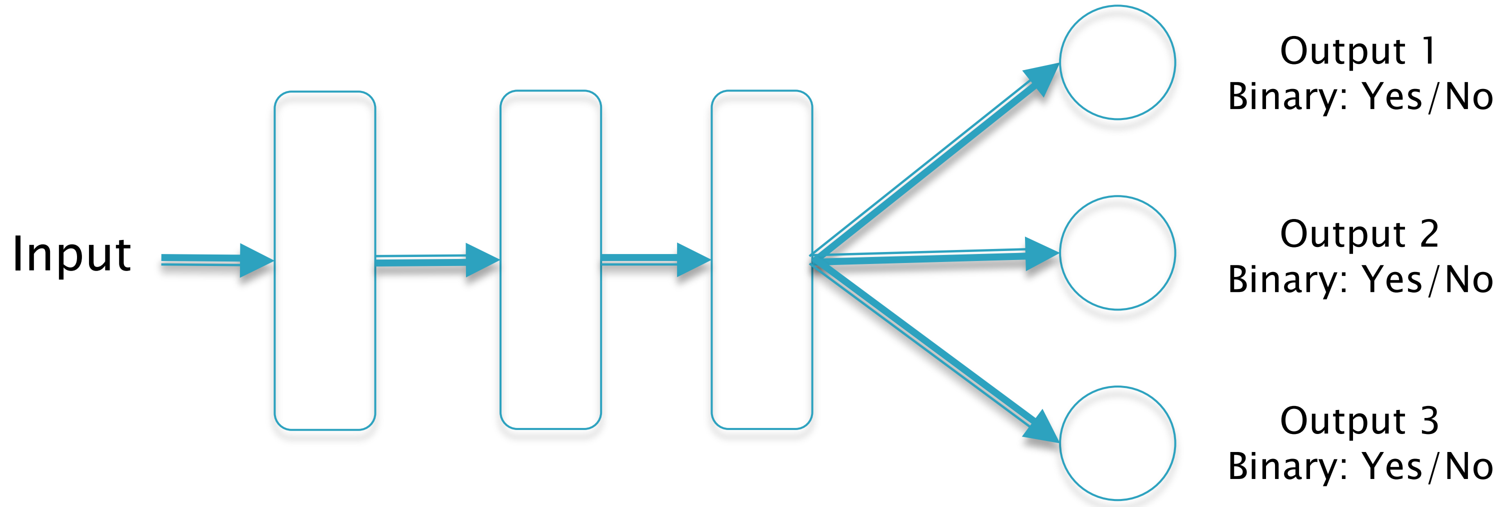
Increases prediction accuracy

# Multi-Input Models



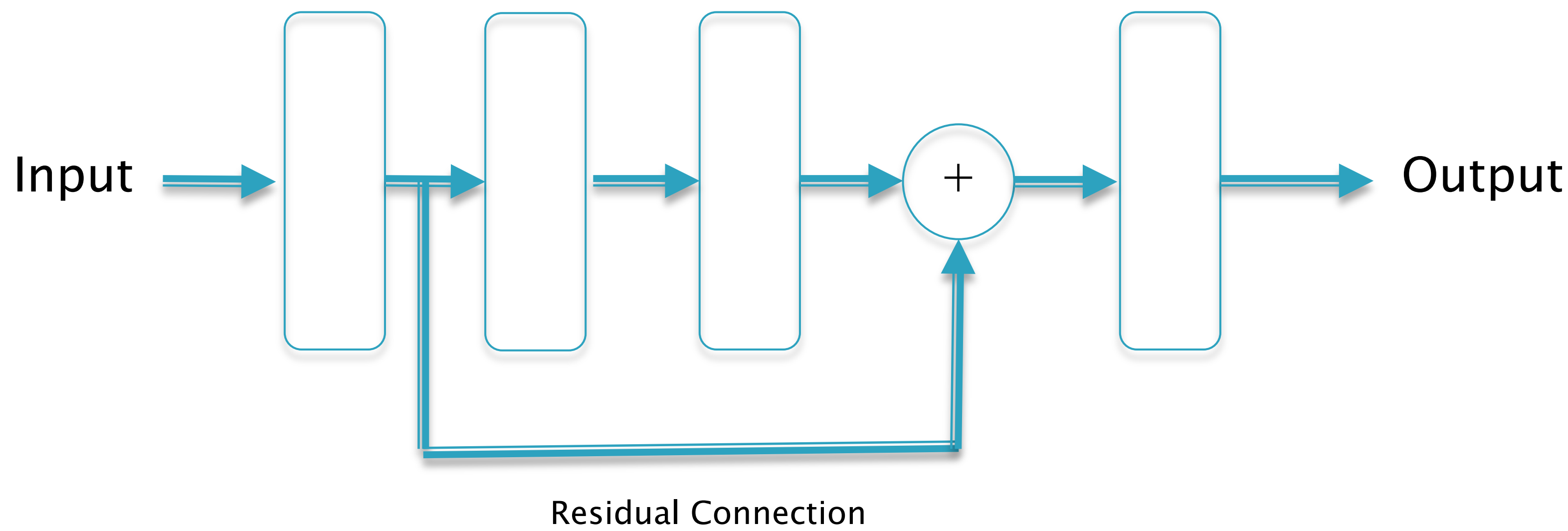

The final decision is a function of more than one type of input data

# Multi-Label Classification




For classifying more than one object per input

# Residual Connections




Enables the training of models with hundreds of hidden layers

# Keras Sequential vs Functional API

All these different topologies can be easily coded using the Keras Functional API

```
import keras
keras.__version__
from keras import Sequential, Model
from keras import layers
from keras import Input

from keras.datasets import cifar10

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

train_images = train_images.reshape((50000, 32 * 32 * 3))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 32 * 32 * 3))
test_images = test_images.astype('float32') / 255

from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

input_tensor = Input(shape=(32 * 32 * 3,))
x = layers.Dense(20, activation='relu')(input_tensor)
y = layers.Dense(15, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(y)

model = Model(input_tensor, output_tensor)

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

history = model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)
```

# Keras Callbacks

- Model Checkpointing: Saving the current state of the model at different points during training
- Early Stopping: Interrupting Training when the Validation Loss is no longer improving (and saving the best model)
- Dynamically adjusting hyper parameter values: Example Learning Rate
- Logging Training and Validation Metrics

# Further Reading

- Das and Varma: Chapter 6 – NNDeepLearning
- Keras Code Examples: https://keras.io/examples/

- Neural Network Playground
  https://playground.tensorflow.org