# Transfer Learning

# What is Transfer Learning

**The reuse of a previously learned model on a new problem is known as transfer learning.**

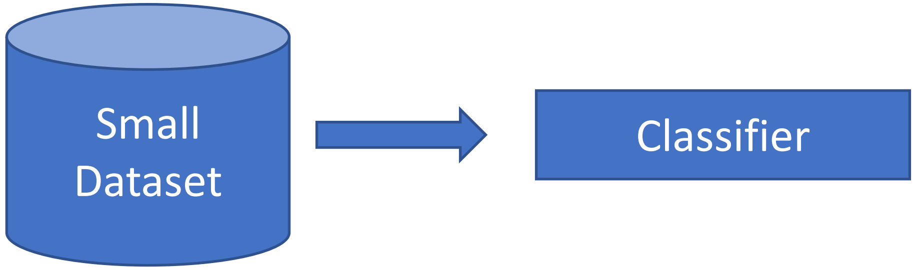
It uses knowledge learn from one model to perform a new task.

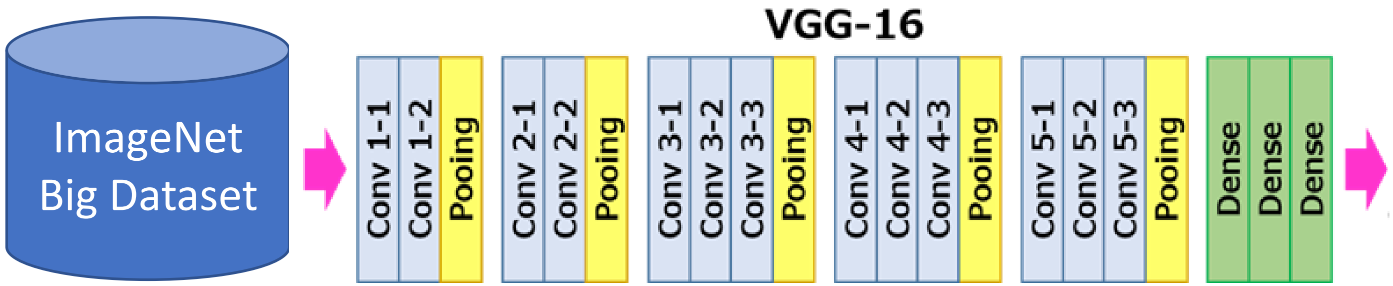
## Motivation

Deep learning methods are data-hungry

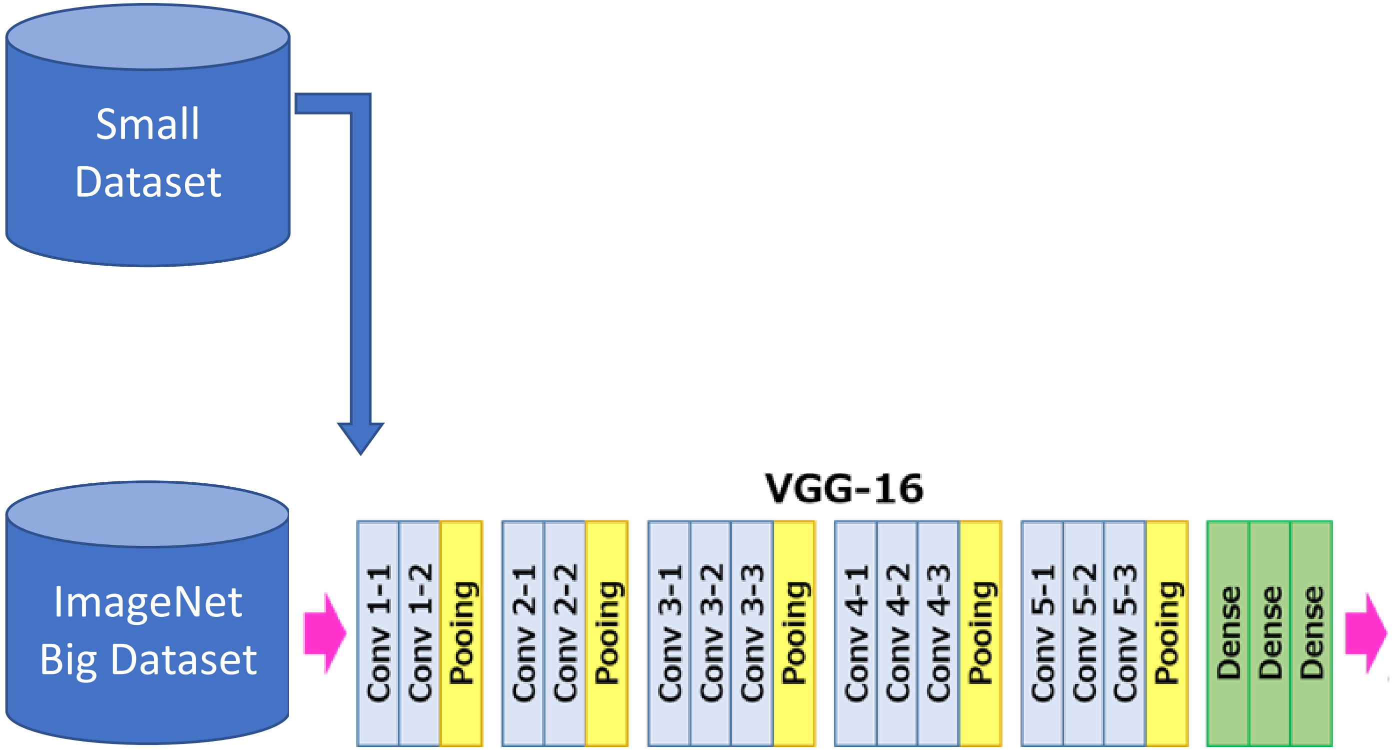
Lots of data, time, resources needed to train and tune a neural network from scratch.

# Example – Image Classification

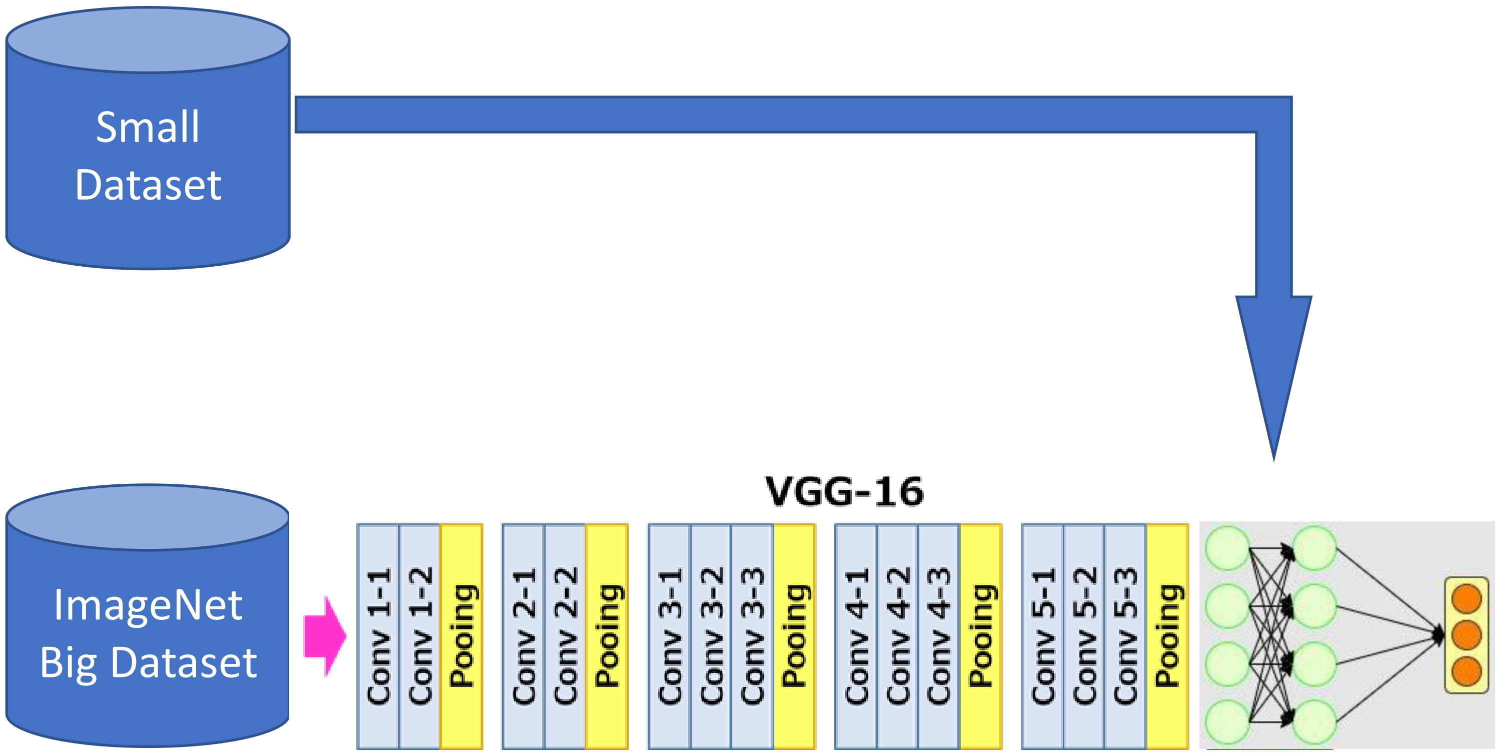


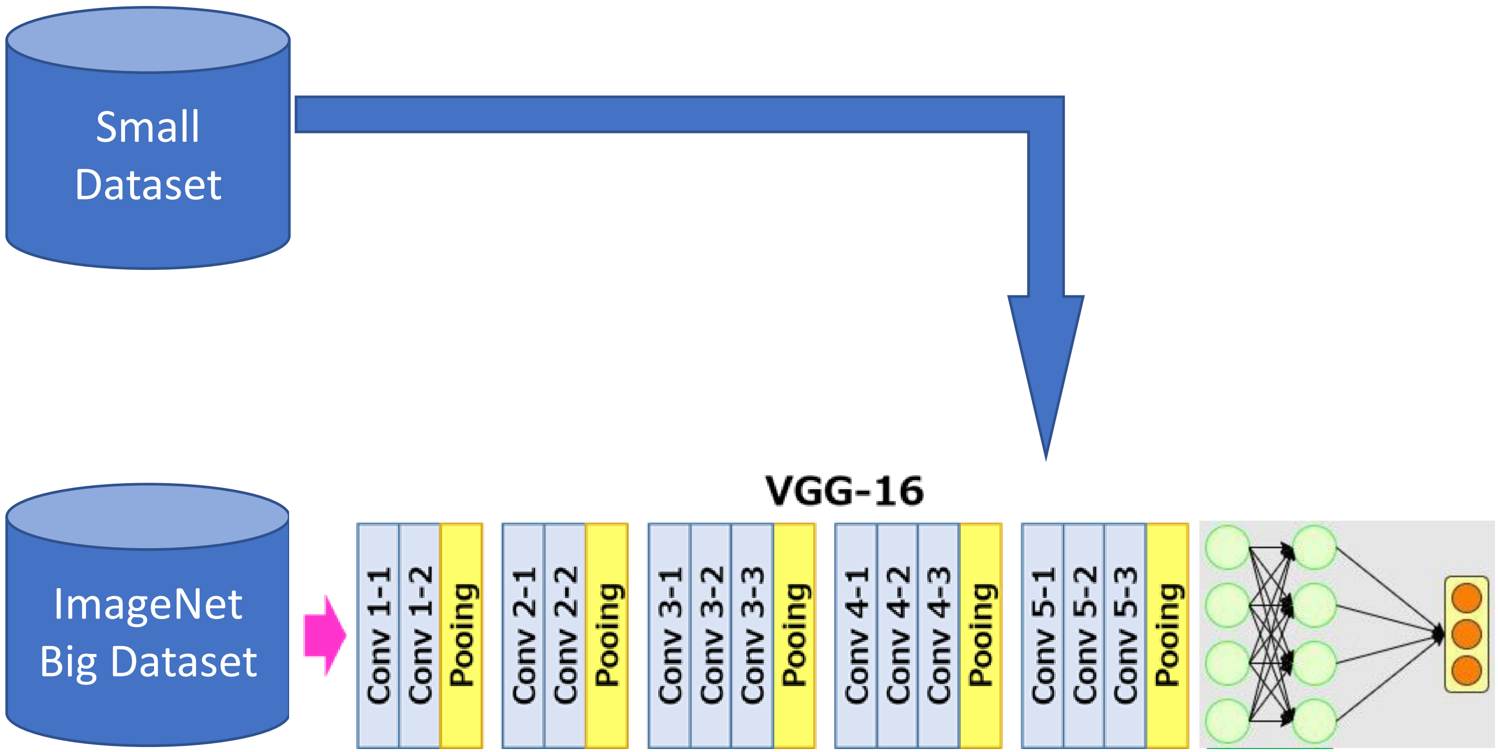
# Example – Image Classification



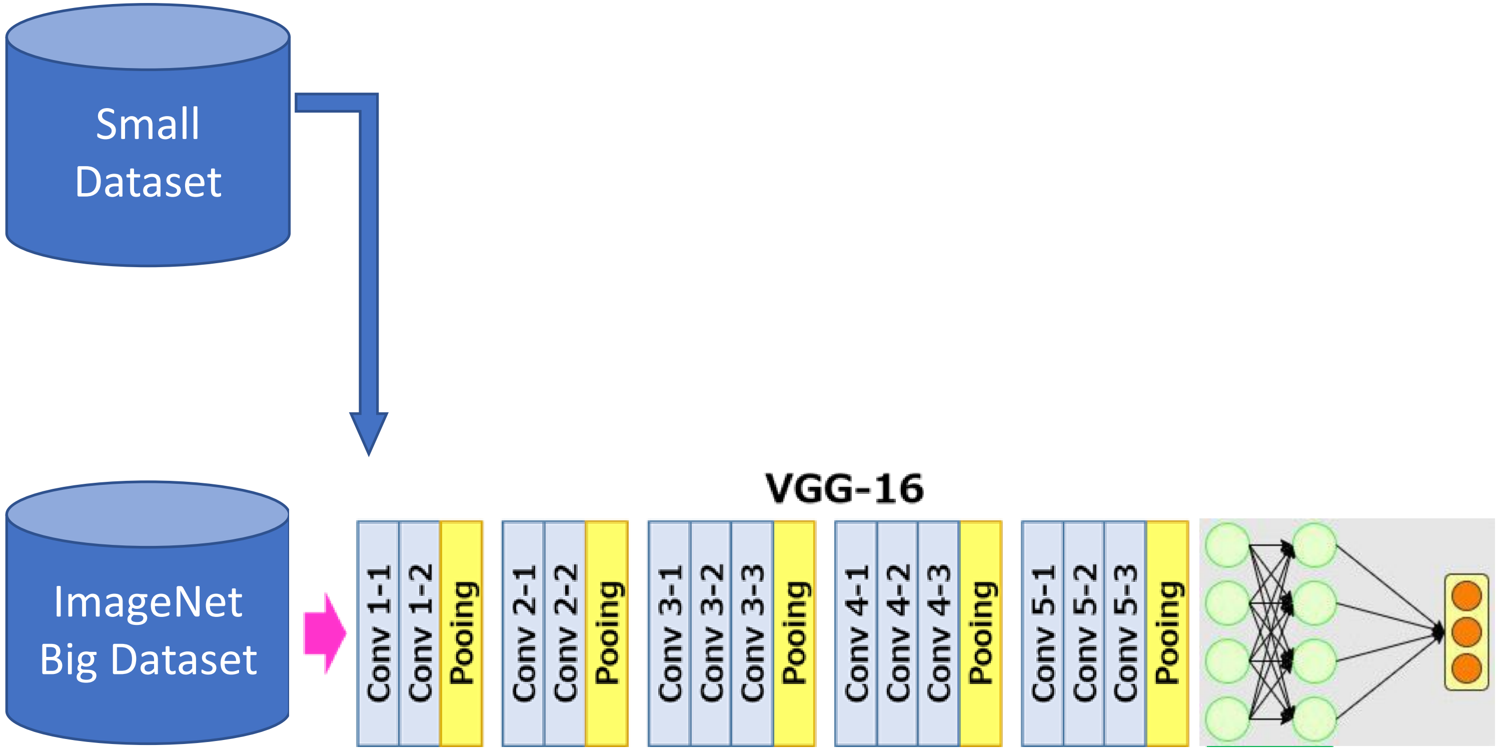
# Example – Image Classification



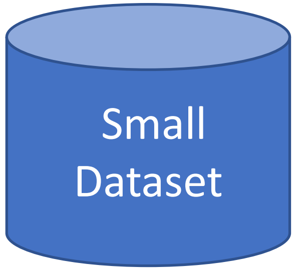
# Example – Image Classification

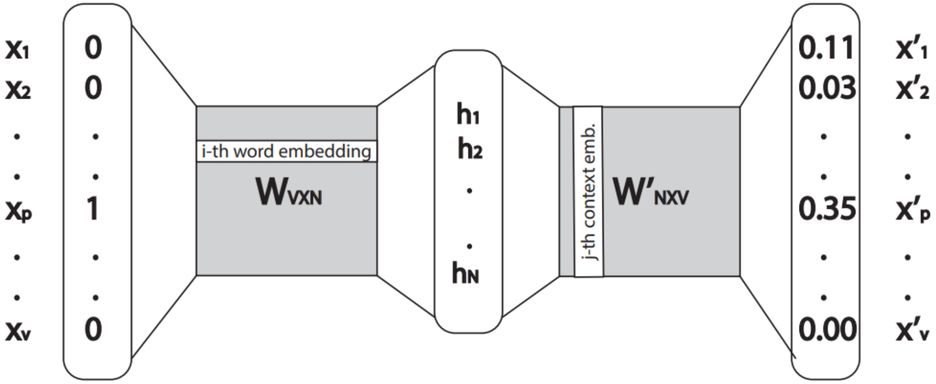


# Example – Image Classification

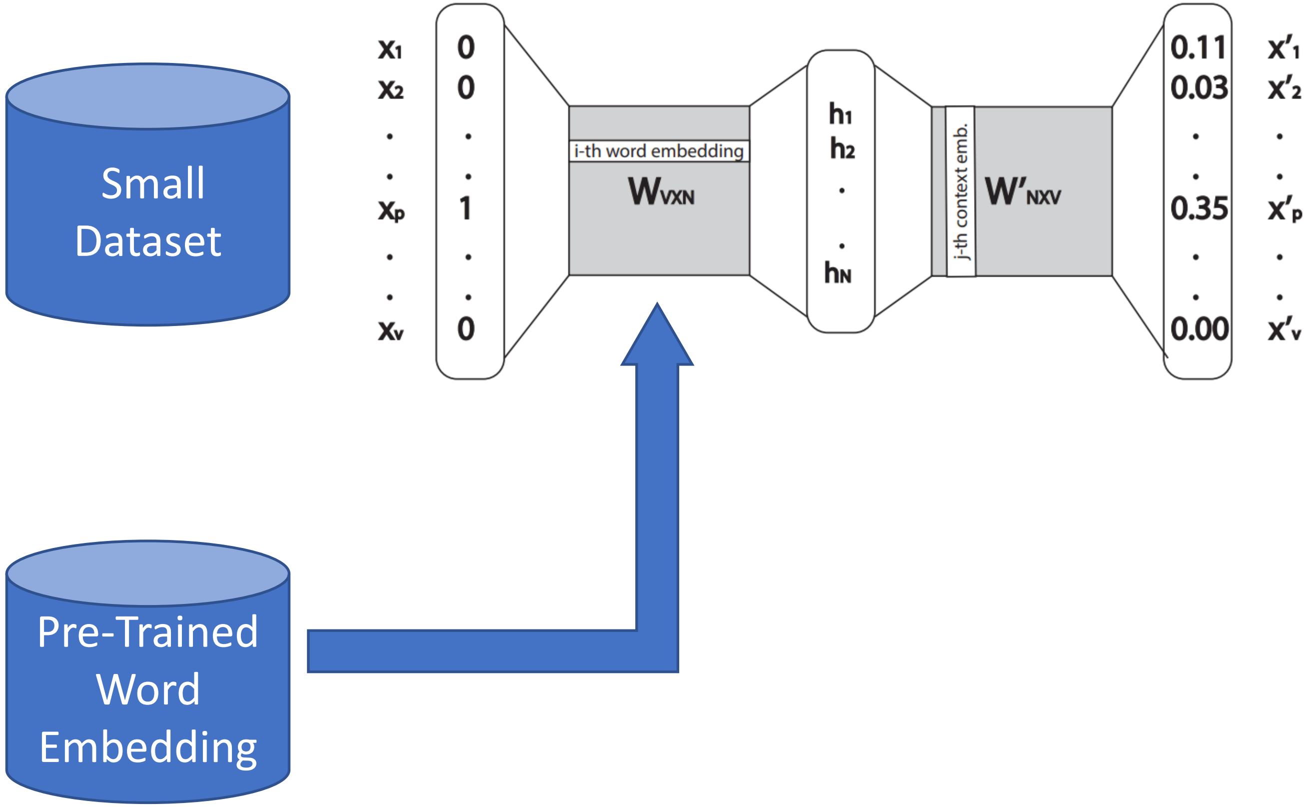


# Example – Task Specific Text Representation

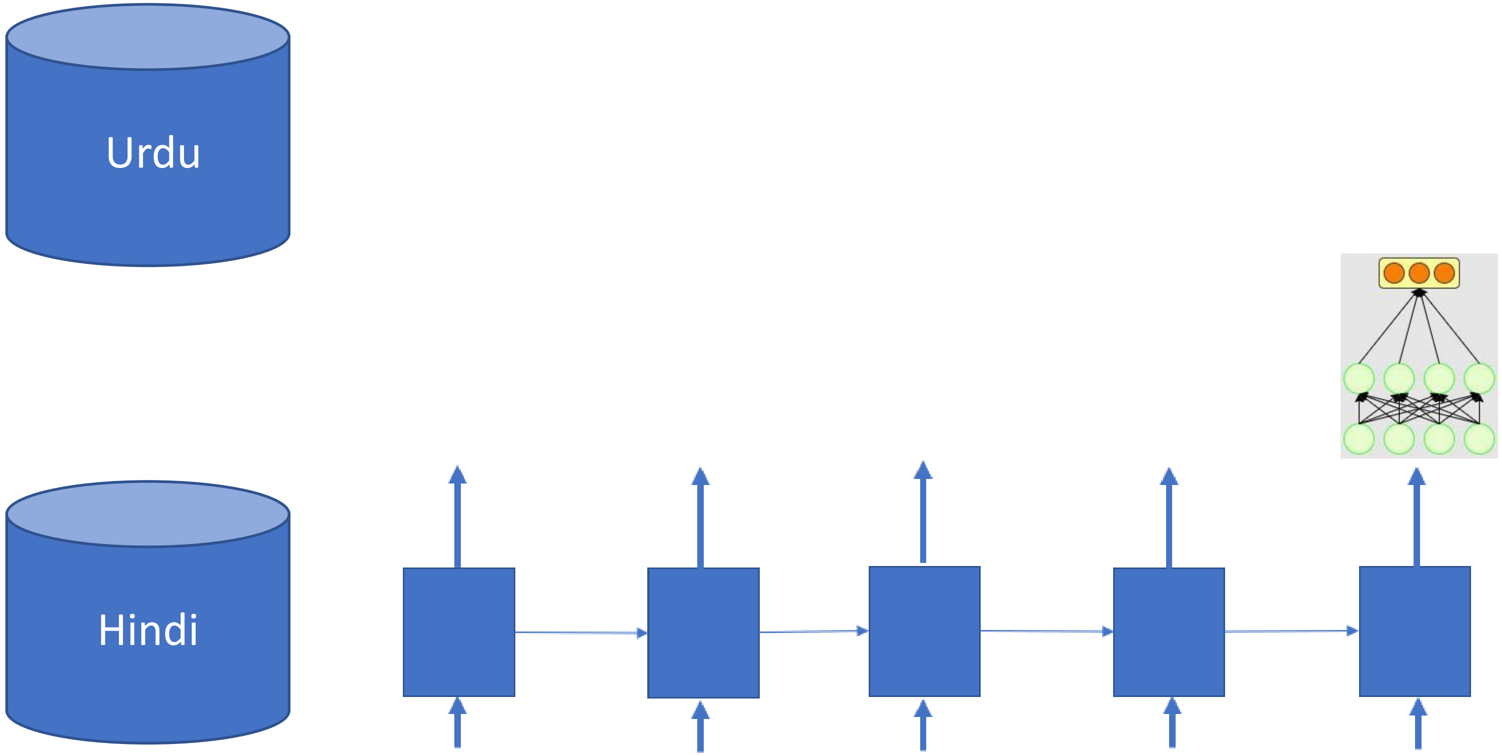


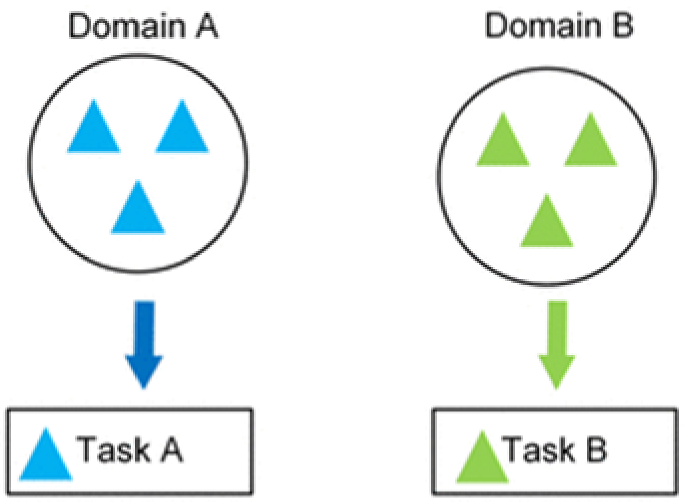
# Example – Task Specific Text Representation

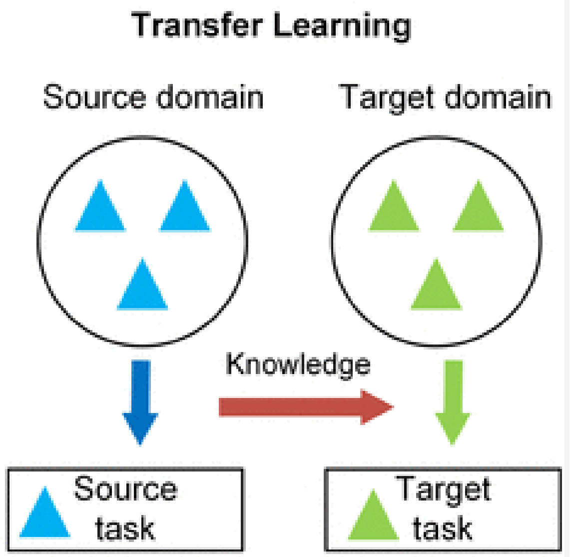


# Example – Sentiment Analysis for Urdu using Hindi

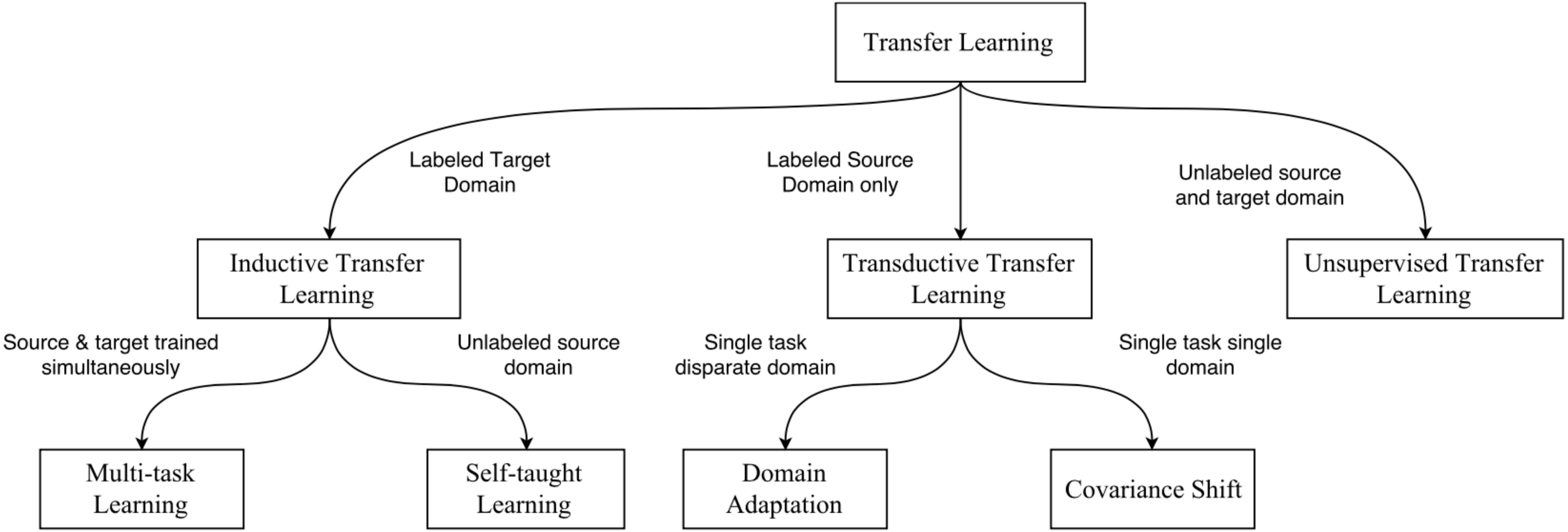


# Traditional Vs Transfer Learning



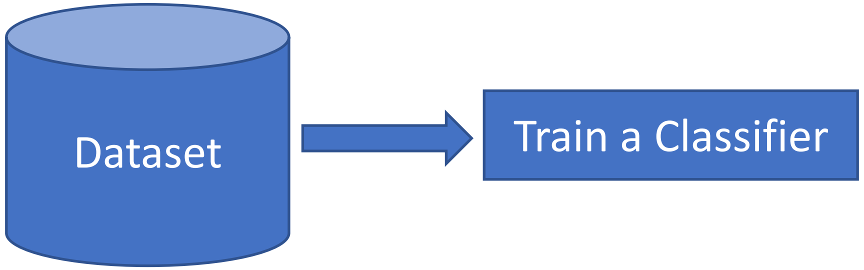



# Types of Transfer Learning

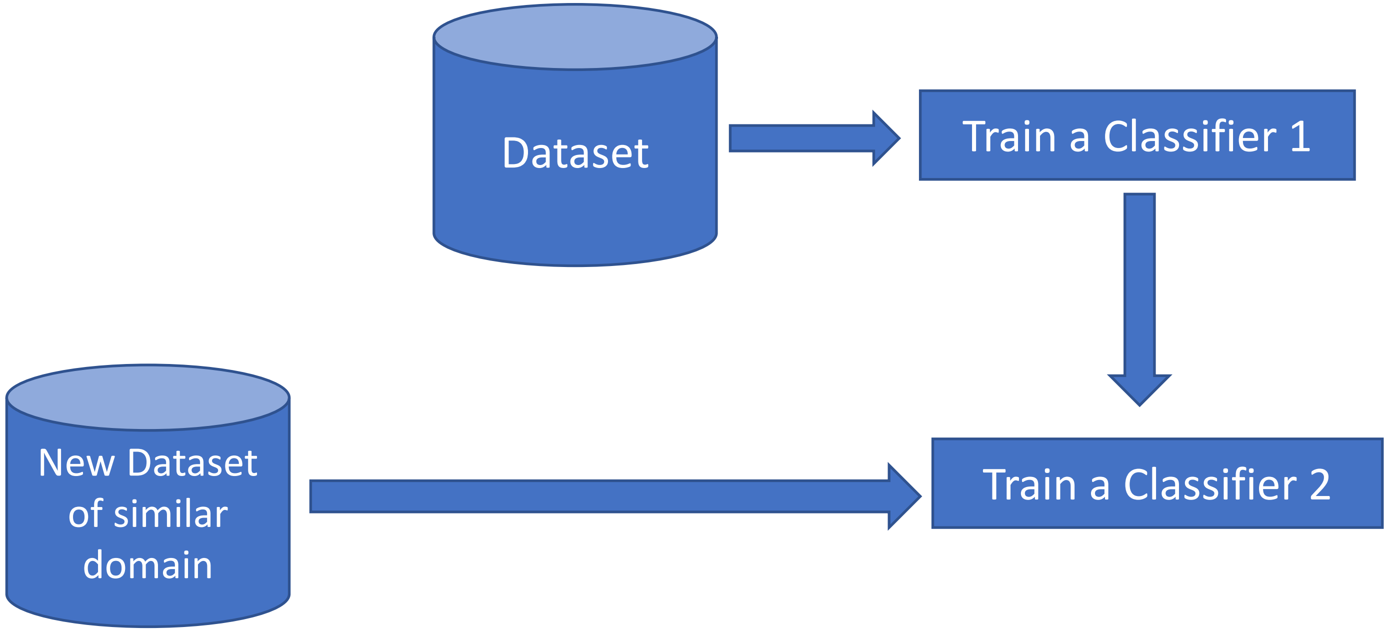



Source: https://arxiv.org/pdf/1802.05934.pdf

# Transductive Transfer Learning

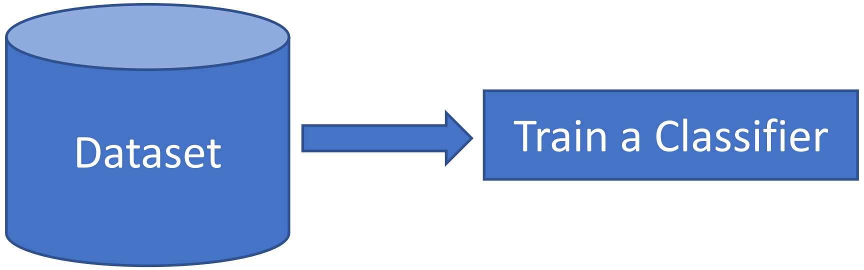


# Transductive Transfer Learning

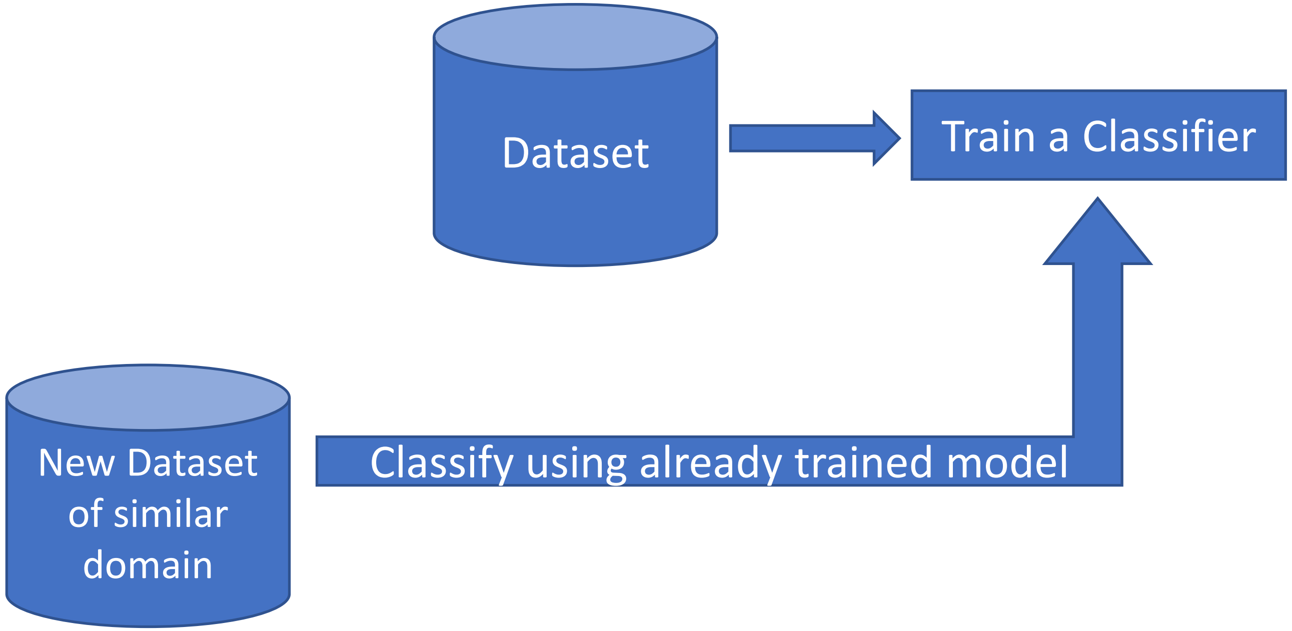



It is also known as **domain adaptation.**

# Inductive Transfer Learning

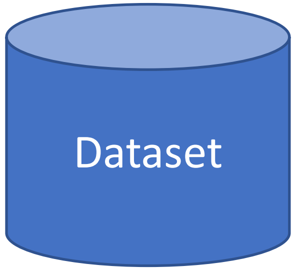


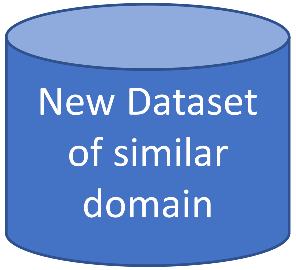
# Inductive Transfer Learning




It is the same as ***traditional supervised learning***. It is also known as **domain generalization.**

# Covariate Shift



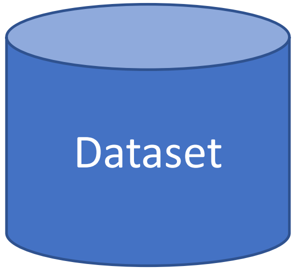

Assumption: $P_{dataset}(\mathrm{x}, \mathrm{y}) \neq P_{new}(x, y)$

Goal: Model a predictor $C_{new}(y|x)$ using $P_{dataset}(\mathrm{x}, \mathrm{y})$

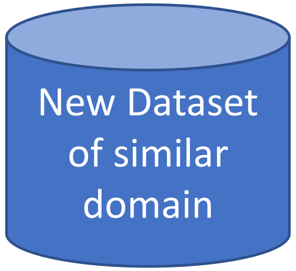
Covariate Shift Assumption:

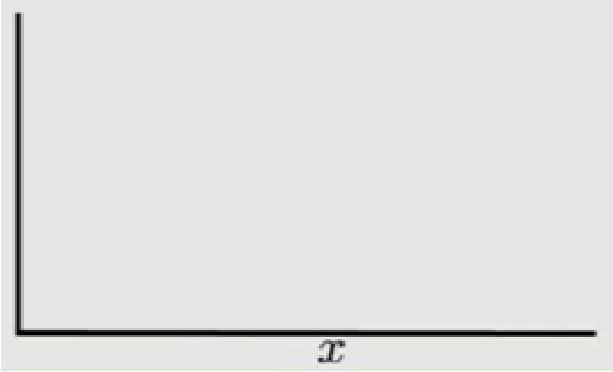
- $P_{dataset}(y|x) = P_{new}(y|x)$
- $P_{dataset}(x) \neq P_{new}(x)$
- $Support_{dataset}(x) = Support_{new}(x)$

# Covariate Shift







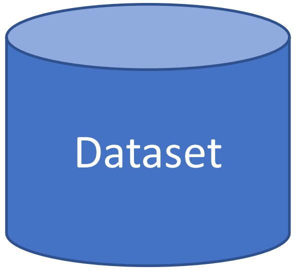

Assumption: $P_{dataset}(\mathrm{x}, \mathrm{y}) \neq P_{new}(x, y)$

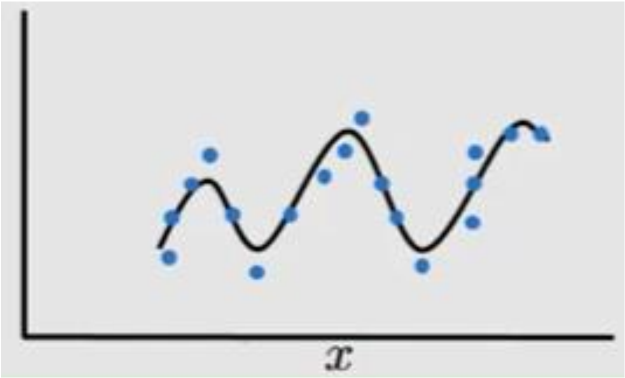
Goal: Model a predictor $C_{new}(y|x)$ using $P_{dataset}(\mathrm{x}, \mathrm{y})$

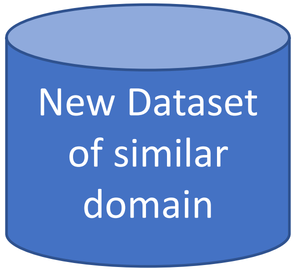
Covariate Shift Assumption:

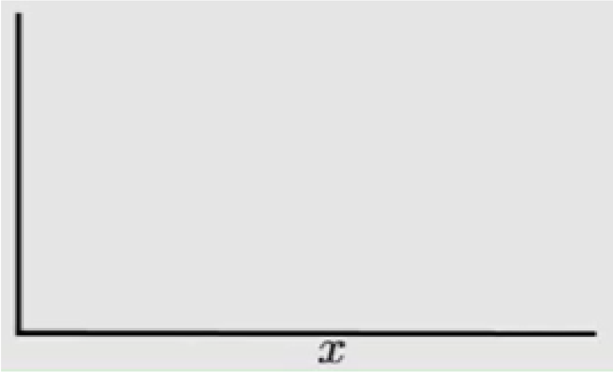
- $P_{dataset}(y|x) = P_{new}(y|x)$
- $P_{dataset}(x) \neq P_{new}(x)$
- $Support_{dataset}(x) = Support_{new}(x)$

# Covariate Shift



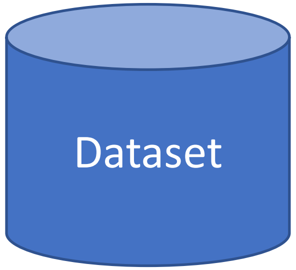

Assumption: $P_{dataset}(\mathrm{x},\mathrm{y}) \neq P_{new}(x,y)$

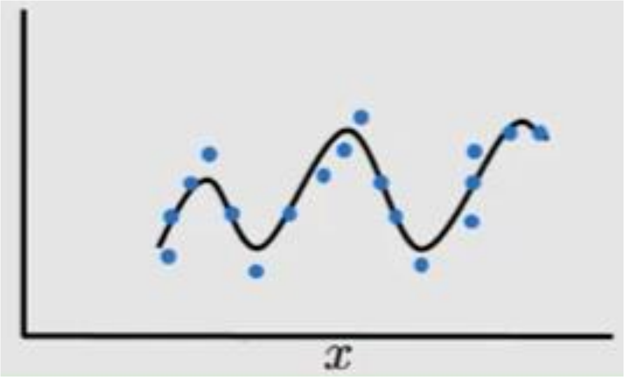
Goal: Model a predictor $C_{new}(y|x)$ using $P_{dataset}(\mathrm{x},\mathrm{y})$

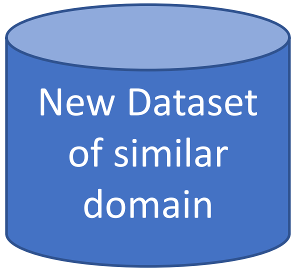
Covariate Shift Assumption:

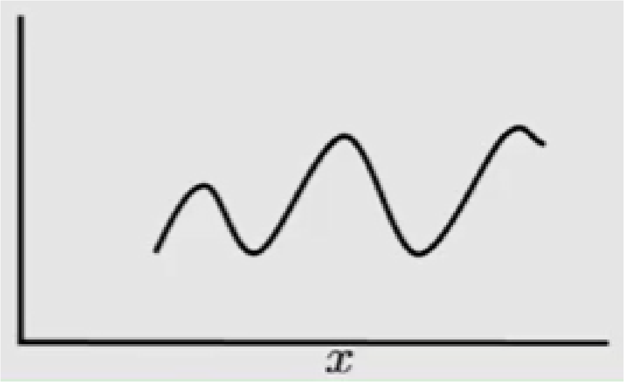
- $P_{dataset}(y|x) = P_{new}(y|x)$
- $P_{dataset}(x) \neq P_{new}(x)$
- $Support_{dataset}(x) = Support_{new}(x)$

# Covariate Shift



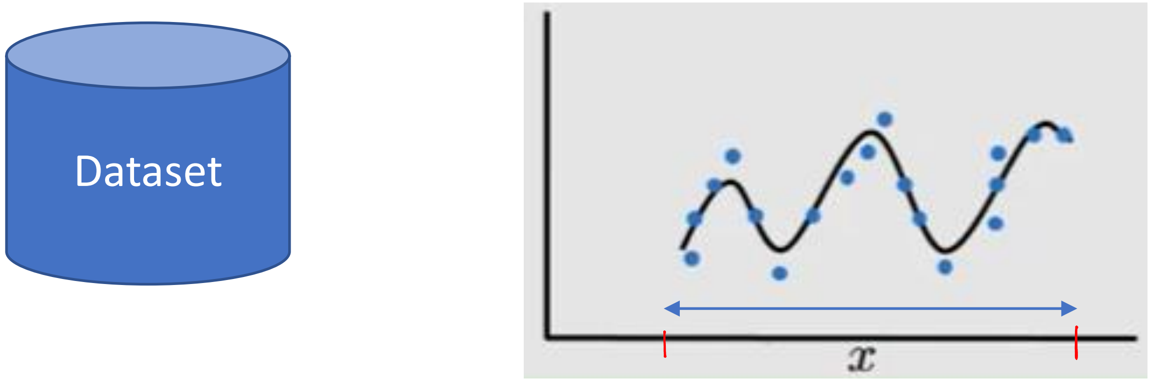

Assumption: $P_{dataset}(\mathrm{x}, \mathrm{y}) \neq P_{new}(x, y)$

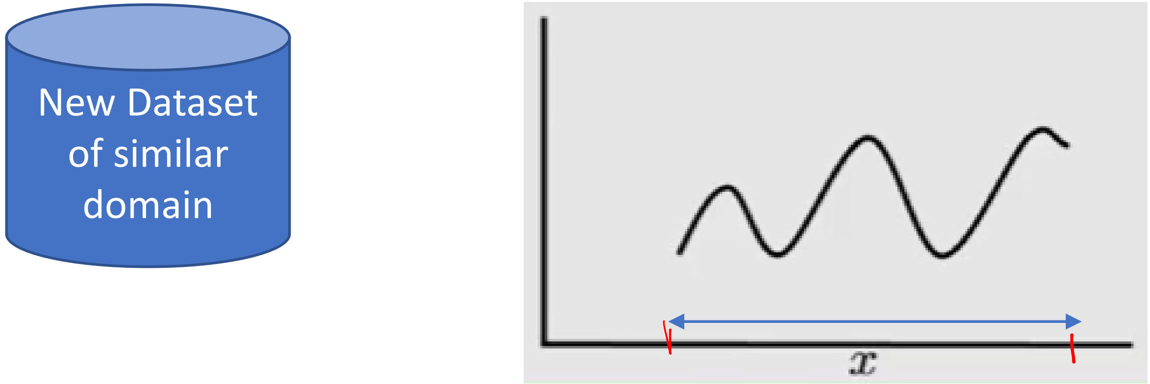
Goal: Model a predictor $C_{new}(y|x)$ using $P_{dataset}(\mathrm{x}, \mathrm{y})$

Covariate Shift Assumption:

- $P_{dataset}(y|x) = P_{new}(y|x)$
- $P_{dataset}(x) \neq P_{new}(x)$
- $Support_{dataset}(x) = Support_{new}(x)$

# Covariate Shift




Assumption: $P_{dataset}(\mathrm{x,y}) \neq P_{new}(x,y)$

Goal: Model a predictor $C_{new}(y|x)$ using $P_{dataset}(\mathrm{x,y})$

Covariate Shift Assumption:

- $P_{dataset}(y|x) = P_{new}(y|x)$
- $P_{dataset}(x) \neq P_{new}(x)$
- $Support_{dataset}(x) = Support_{new}(x)$

# Transductive Learning: VGG16 for Two Classes

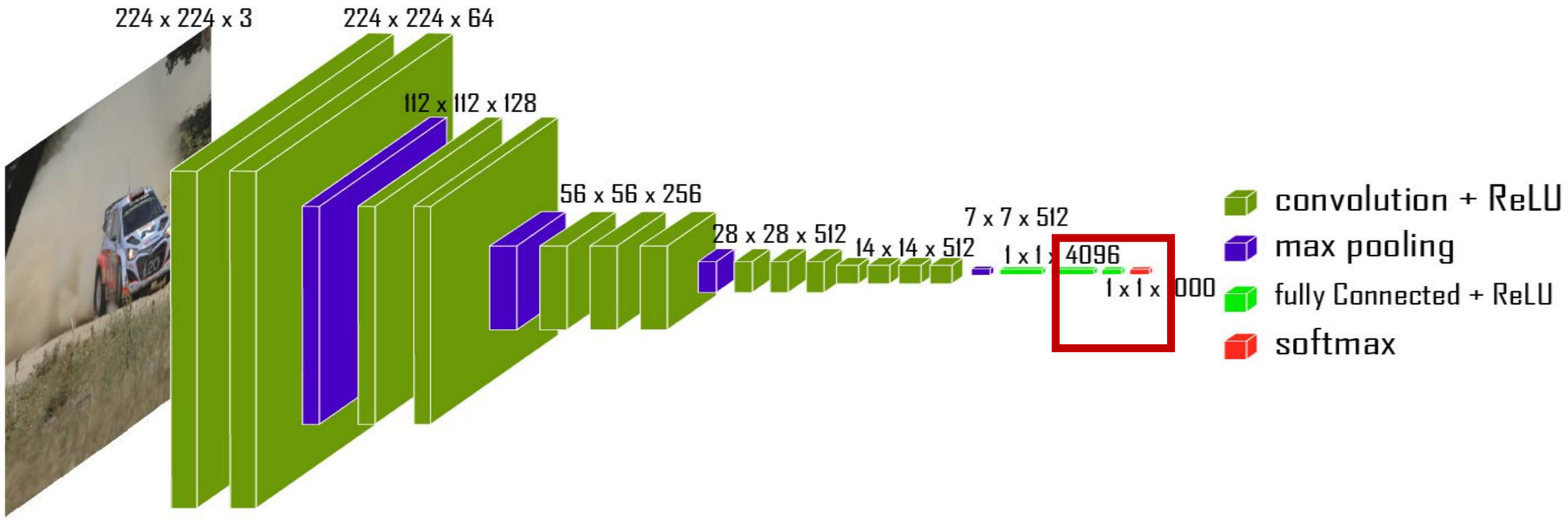
CAT




DOG




# Modify VGG16 for Two Classes




```python
from keras.applications.vgg16 import VGG16
vggmodel = VGG16(weights=None, include_top=False,input_shape = (224,224,3) )
vggmodel.summary()
```

# Modify VGG16 for Two Classes

```
import keras
from keras.models import Model
from keras.layers import Dense, Flatten
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
```

```
from keras.applications.vgg16 import VGG16
vggmodel = VGG16(weights=None, include_top=False,input_shape = (224,224,3) )
vggmodel.summary()
```

```
for  layer in vggmodel.layers:
    layer.trainable = True
```

```
x = Flatten()(vggmodel.output) # Flattened the last layer

prediction = Dense( 2 , activation = 'softmax' )(x) # Created a new layer as output

model = Model( inputs = vggmodel.input , outputs = prediction ) # Join it with the model

model.summary() # Visualize the model again
```

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

# Modify VGG16 for Two Classes

```
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="../dogs-vs-cats/train",target_size=(224,224))
vdata = ImageDataGenerator()
testdata = vdata.flow_from_directory(directory="../dogs-vs-cats/validation", target_size=(224,224))
```

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint("catVsDog-v2.h5", monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=False, m
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=3, verbose=1, mode='auto')
hist = vggmodel.fit_generator(steps_per_epoch=3,generator=traindata, validation_data= testdata, validation_steps=3,epochs=10, cal
model.save("catVsDog-v2.h5")
```

```
from keras.preprocessing import image
import numpy as np

img = image.load_img("1.jpg",target_size=(224,224))
img = np.asarray(img)
plt.imshow(img)
img = np.expand_dims(img, axis=0)


from keras.models import load_model

saved_model = load_model("catVsDog_v2.h5")
output = saved_model.predict(img)
if output[0][0] > output[0][1]:
    print("cat")
else:
    print('dog')
```

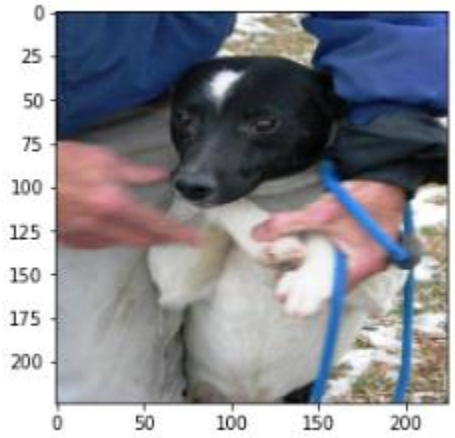
# Modify VGG16 for Two Classes

```
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="../dogs-vs-cats/train",target_size=(224,224))
vdata = ImageDataGenerator()
testdata = vdata.flow_from_directory(directory="../dogs-vs-cats/validation", target_size=(224,224))
```

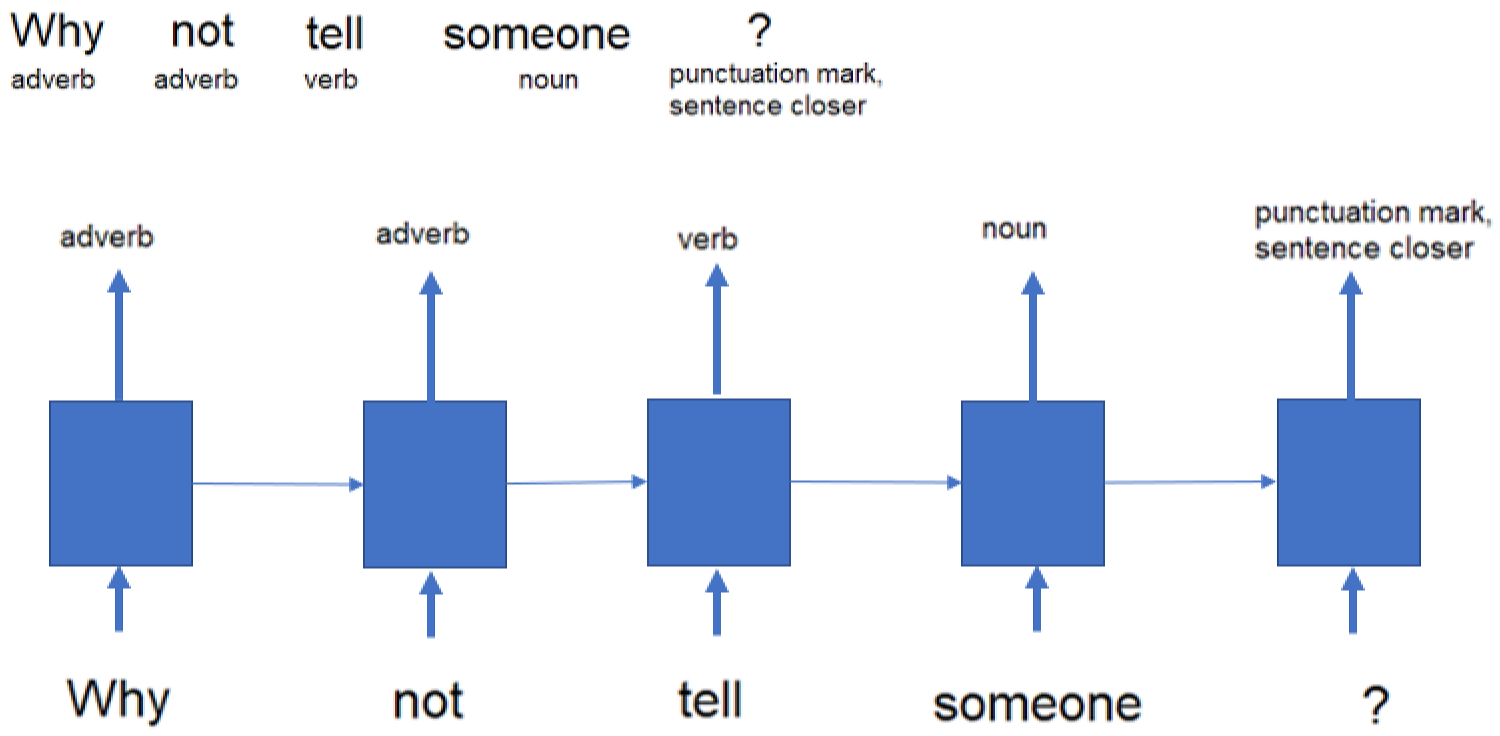
```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint("catVsDog-v2.h5", monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=False, m
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=3, verbose=1, mode='auto')
hist = vggmodel.fit_generator(steps_per_epoch=3,generator=traindata, validation_data= testdata, validation_steps=3,epochs=10, cal
model.save("catVsDog-v2.h5")
```

```
from keras.preprocessing import image
import numpy as np

img = image.load_img("1.jpg",target_size=(224,224))
img = np.asarray(img)
plt.imshow(img)
img = np.expand_dims(img, axis=0)


from keras.models import load_model

saved_model = load_model("catVsDog_v2.h5")
output = saved_model.predict(img)
if output[0][0] > output[0][1]:
    print("cat")
else:
    print('dog')
```

dog




# Transductive Learning: Part-Of-Speech Tagging using RNN



# Part-Of-Speech Tagging using RNN

Adapted from : https://www.kaggle.com/code/tanyadayanand/pos-tagging-using-rnn

```
import numpy as np
from nltk.corpus import treebank
from gensim.models import KeyedVectors
from keras_preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Embedding
from keras.layers import Dense, Input
from keras.layers import TimeDistributed
from keras.layers import LSTM, GRU, Bidirectional, SimpleRNN, RNN
from keras.models import Model
from keras.preprocessing.text import Tokenizer

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

# Treebank Dataset and Preprocessing

```
# Load the treebank POS tagged dataset
tagged_sentences = treebank.tagged_sents()

# Structure of the annotation
tagged_sentences[0]
```

# Treebank Dataset and Preprocessing

```
# Load the treebank POS tagged dataset
tagged_sentences = treebank.tagged_sents()

# Structure of the annotation
tagged_sentences[0]
```

```
X = [] # Sample
Y = [] # Labels

for sentence in tagged_sentences:
    X_sentence = []
    Y_sentence = []
    for entity in sentence:
        X_sentence.append(entity[0])  # Word in the sentence
        Y_sentence.append(entity[1])  # corresponding POS tag

    X.append(X_sentence)
    Y.append(Y_sentence)
```

# Treebank Dataset and Preprocessing

```
# encode X

word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X)
X_encoded = word_tokenizer.texts_to_sequences(X)
```

```
# encode Y

tag_tokenizer = Tokenizer()
tag_tokenizer.fit_on_texts(Y)
Y_encoded = tag_tokenizer.texts_to_sequences(Y)
```

```
# Raw data and encoding
print('X: ', X[0], '\n')
print('Y: ', Y[0], '\n')
print('X: ', X_encoded[0], '\n')
print('Y: ', Y_encoded[0], '\n')
```

```
X:  ['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'director',
'Nov.', '29', '.']

Y:  ['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', ',', 'MD', 'VB', 'DT', 'NN', 'IN', 'DT', 'JJ', 'NN', 'NNP', 'CD', '.']

X:  [5601, 3746, 1, 2024, 86, 331, 1, 46, 2405, 2, 131, 27, 6, 2025, 332, 459, 2026, 3]

Y:  [3, 3, 8, 10, 6, 7, 8, 21, 13, 4, 1, 2, 4, 7, 1, 3, 10, 9]
```

# Treebank Dataset and Preprocessing

```
MAX_SEQ_LENGTH = 50  # sequences greater than 50 in length will be truncated

X = pad_sequences(X_encoded, maxlen=MAX_SEQ_LENGTH, padding="pre", truncating="post")
Y = pad_sequences(Y_encoded, maxlen=MAX_SEQ_LENGTH, padding="pre", truncating="post")
```

```
# First Sentence and its label
print(X[0], "\n"*3)
print(Y[0])
```

```
[    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0 5601 3746    1 2024   86  331    1   46 2405    2
   131   27    6 2025  332  459 2026    3]



[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  3  3  8 10  6  7  8 21 13  4  1  2  4  7  1  3
 10  9]
```

```
# Create one hot encoding of the labels
Y = to_categorical(Y)
```

```
# Training, TEsting and Validation
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=4)
X_train, X_validation, Y_train, Y_validation = train_test_split(X_train, Y_train, test_size=0.15, random_state=4)
```

# Use Pre-trained Google Word2Vec

```python
# Used pre-trained word2vec embedding

path = './GoogleNews-vectors-negative300.bin'
word2vec = KeyedVectors.load_word2vec_format(path, binary=True)
```

```python
# Extract embedding for each word in the dataset from Word2Vec

EMBEDDING_SIZE  = 300  # embedding size
VOCABULARY_SIZE = len(word_tokenizer.word_index) + 1

# create an empty embedding matix
embedding_weights = np.zeros((VOCABULARY_SIZE, EMBEDDING_SIZE))

# create a word to index dictionary mapping
word2id = word_tokenizer.word_index

# copy vectors from word2vec model to the words present in corpus
for word, index in word2id.items():
    try:
        embedding_weights[index, :] = word2vec[word]
    except KeyError:
        pass
```

# Many-to-many Trainable RNN

```
# total number of tags
NUM_CLASSES = Y.shape[2]
```

```
# create architecture

rnn_model = Sequential()

# create embedding layer - usually the first layer in text problems
rnn_model.add(Embedding(input_dim     =  VOCABULARY_SIZE,         # vocabulary size - number of unique words in data
                        output_dim    =  EMBEDDING_SIZE,          # length of vector with which each word is represented
                        input_length  =  MAX_SEQ_LENGTH,          # length of input sequence
                        weights       = [embedding_weights],      # word embedding matrix
                        trainable     =  True                     # True - update the embeddings while training
))

# add an RNN layer which contains 64 RNN cells
rnn_model.add(SimpleRNN(64,
              return_sequences=True  # True - return whole sequence; False - return single output of the end of the sequence
))

# add time distributed (output at each sequence) layer
rnn_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax')))


rnn_model.compile(loss      =  'categorical_crossentropy',
                  optimizer =  'adam',
                  metrics   =  ['acc'])
# check summary of the model
rnn_model.summary()
```

# Prediction

```
rnn_training = rnn_model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_validation, Y_validation))
```

```
prediction=model.predict_classes(test_seq)
print(prediction)
loss, accuracy = rnn_model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
prediction=rnn_model.predict(X_test)
print(prediction[0])
loss, accuracy = rnn_model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

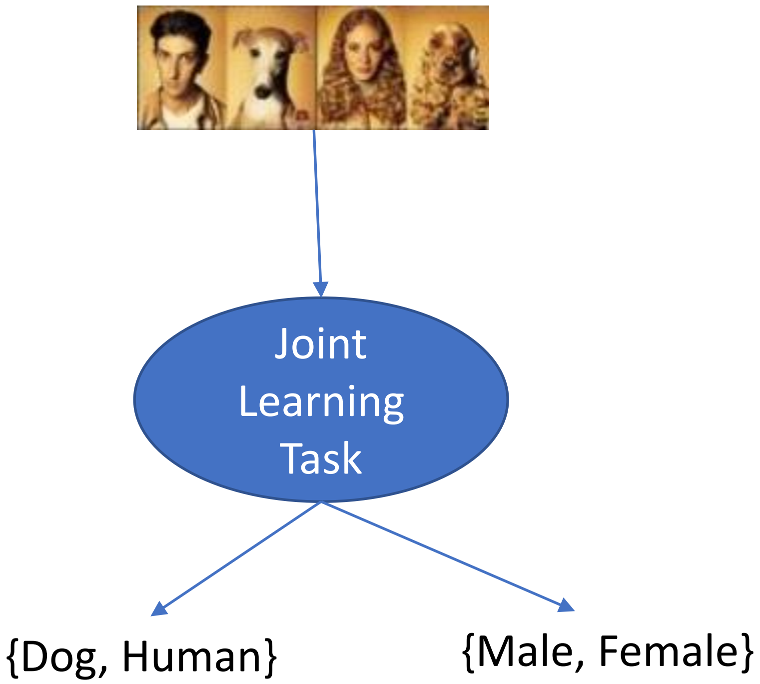
```
19/19 [==============================] - 0s 6ms/step

[[9.4129276e-01 2.0318586e-04 1.6010919e-04 ... 1.1508343e-03
  2.0732924e-03 1.8700062e-03]
 [9.9475807e-01 4.9241667e-06 1.2485887e-05 ... 1.2610275e-04
  2.2710966e-04 1.8933785e-04]
 [9.9640536e-01 3.5103747e-06 8.4279964e-06 ... 7.7903613e-05
  1.5103622e-04 1.3862057e-04]
 ...
 [9.3454997e-05 8.2431763e-01 2.5043532e-03 ... 1.1551229e-04
  2.7520786e-04 8.3007669e-04]
 [2.8277838e-04 7.2922724e-01 1.5582883e-03 ... 2.1224949e-04
  5.3602213e-04 1.2912665e-03]
 [2.1531312e-05 9.7058213e-04 1.6488490e-05 ... 6.2426268e-05
  9.4029841e-05 2.5430173e-04]]
19/19 [==============================] - 0s 7ms/step - loss: 0.1817 - acc: 0.9585
Loss: 0.1816641241312027,
Accuracy: 0.9584693908691406
```

# Multitask Learning

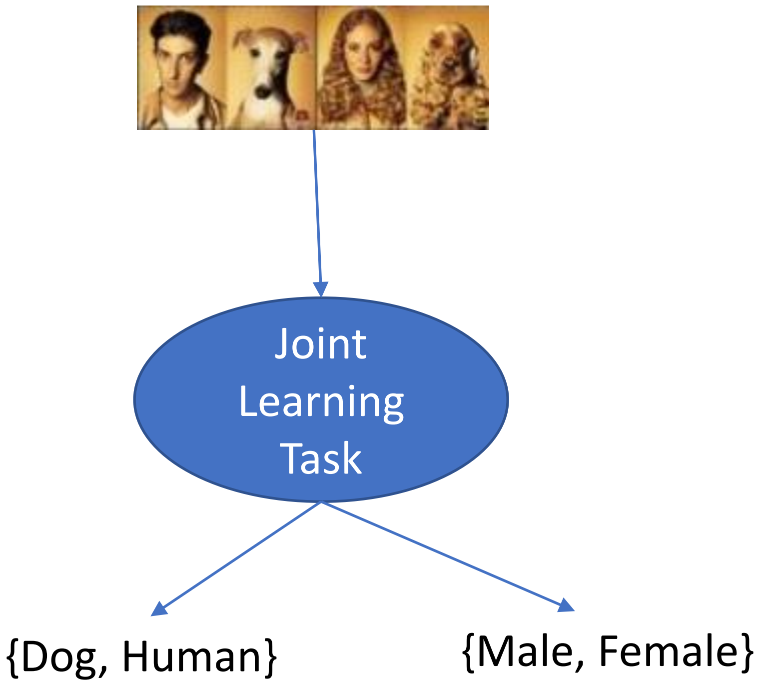
# Multitask Learning

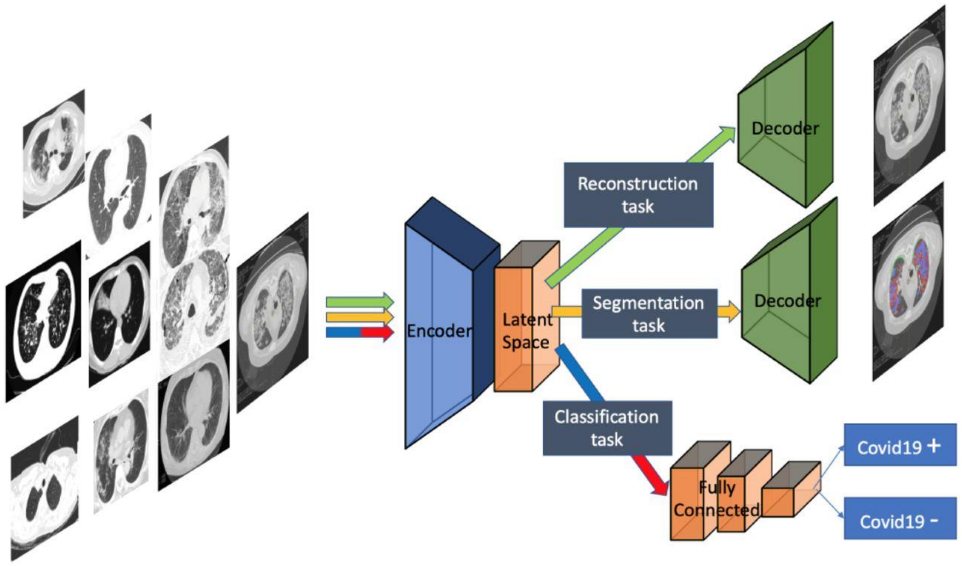
- Standard method in Machine Learning learns one task at a time.
- However, a large problem can be broken down to smaller problems

# Multitask Learning

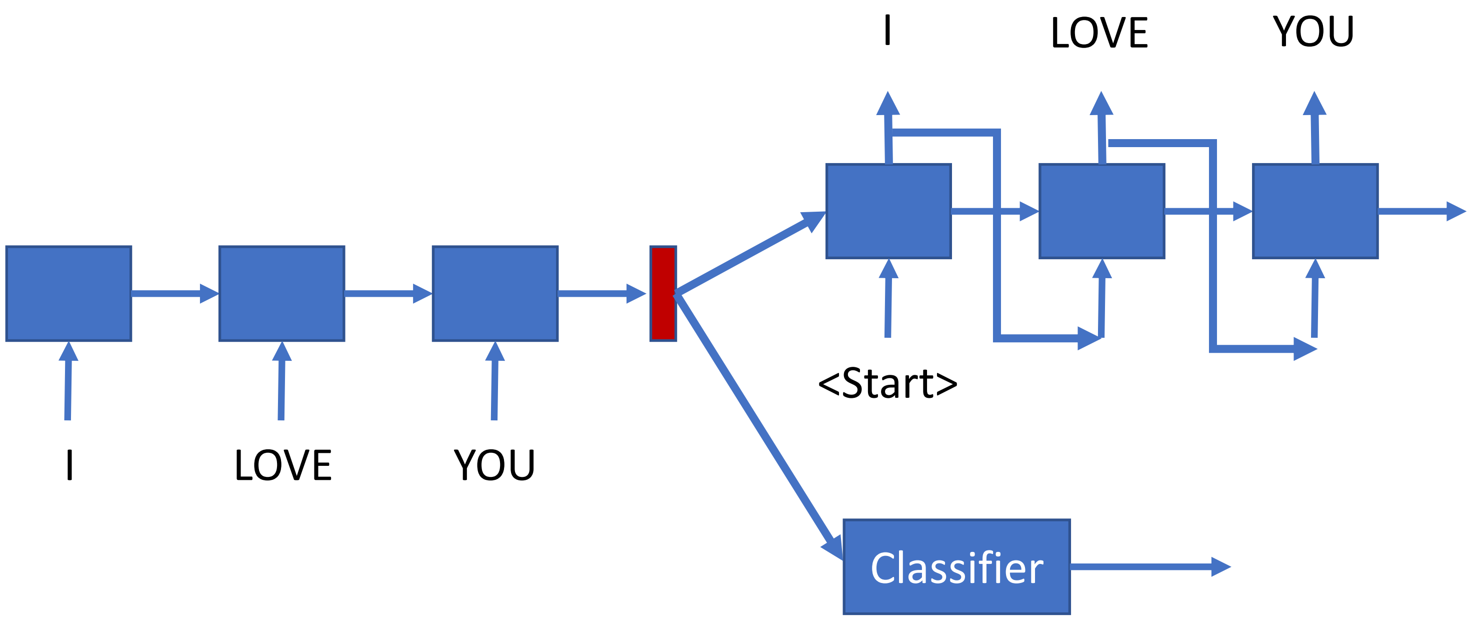


# Multitask Learning



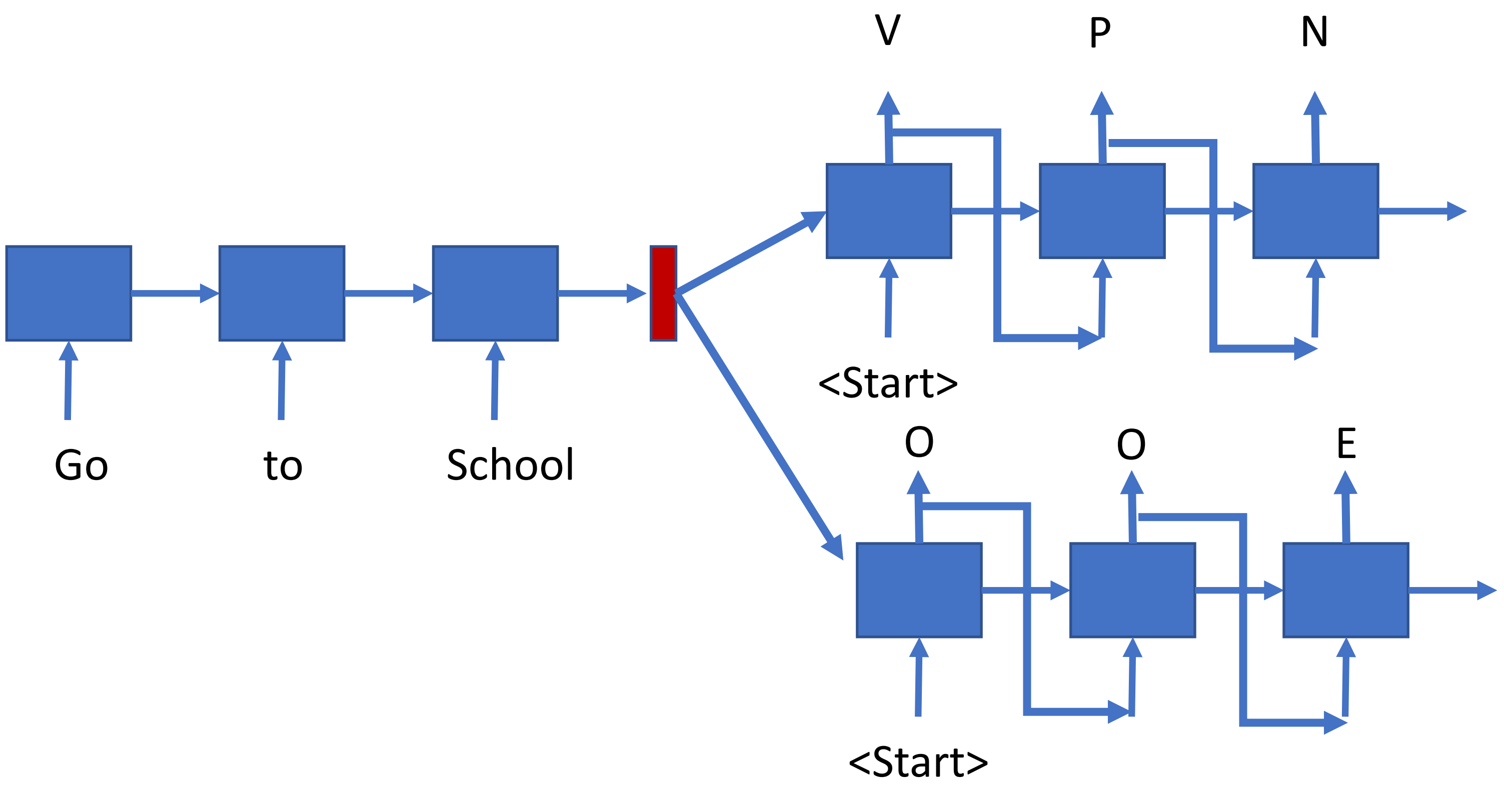



# Multitask Learning



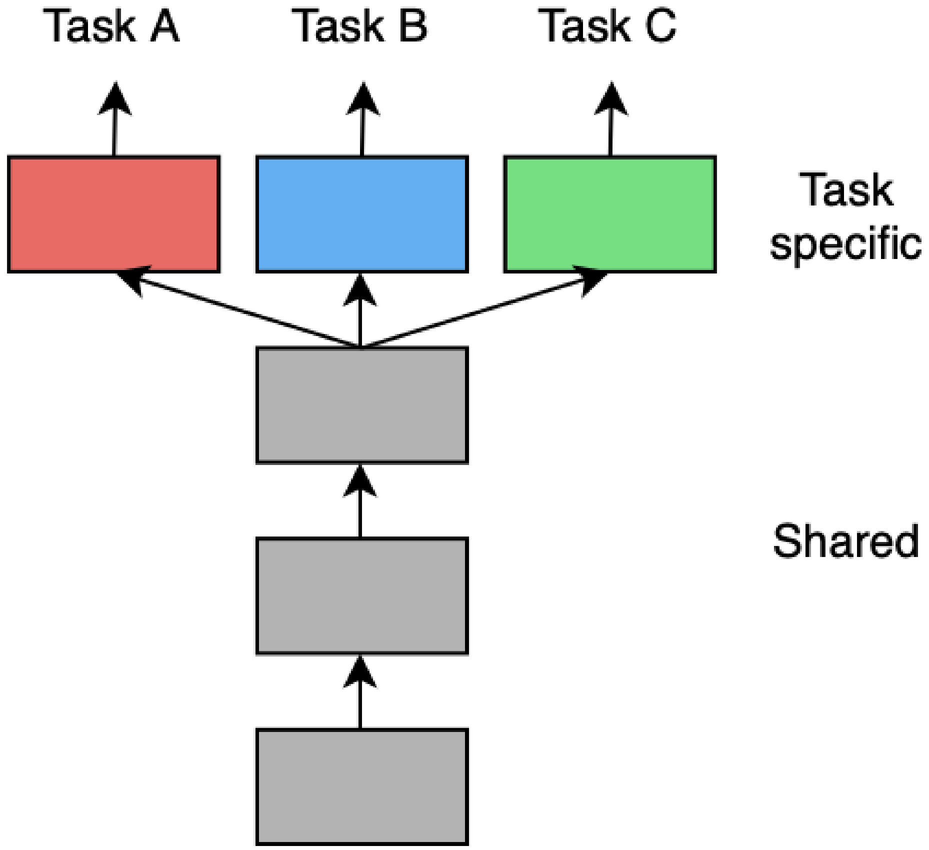

Given a text corpus, learn representation and classification together.

# Multitask Learning

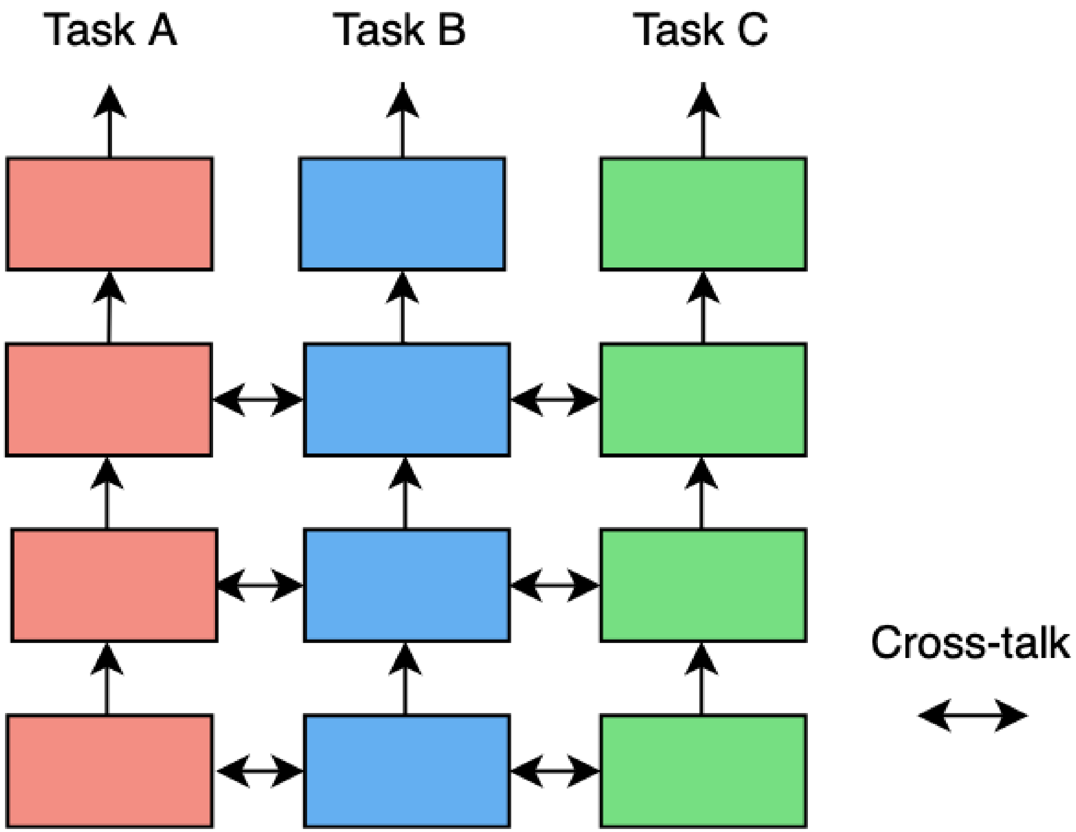



Given a text corpus, train a network to identify Part of Speech and Name Entities

# Multitask Learning

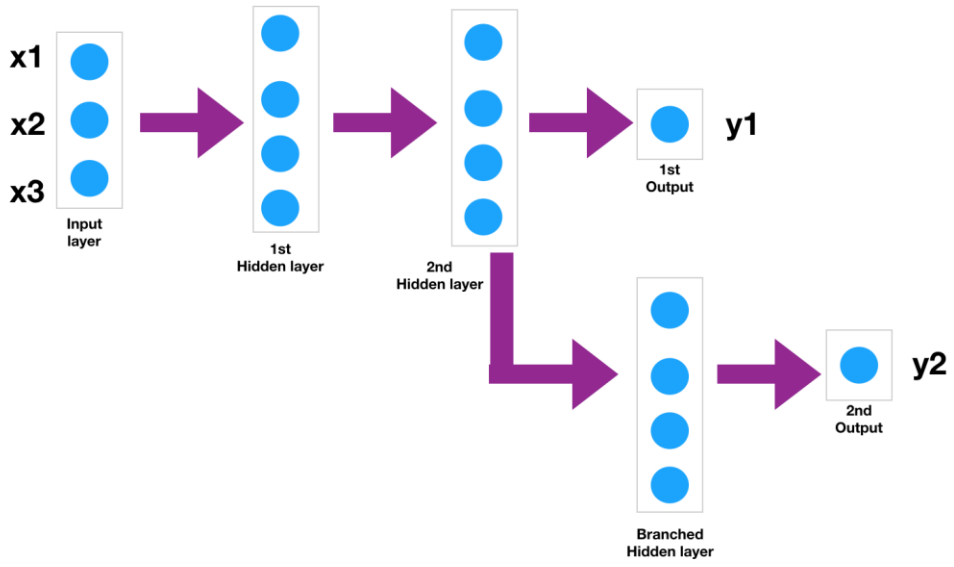



Hard parameter sharing




Soft parameter sharing

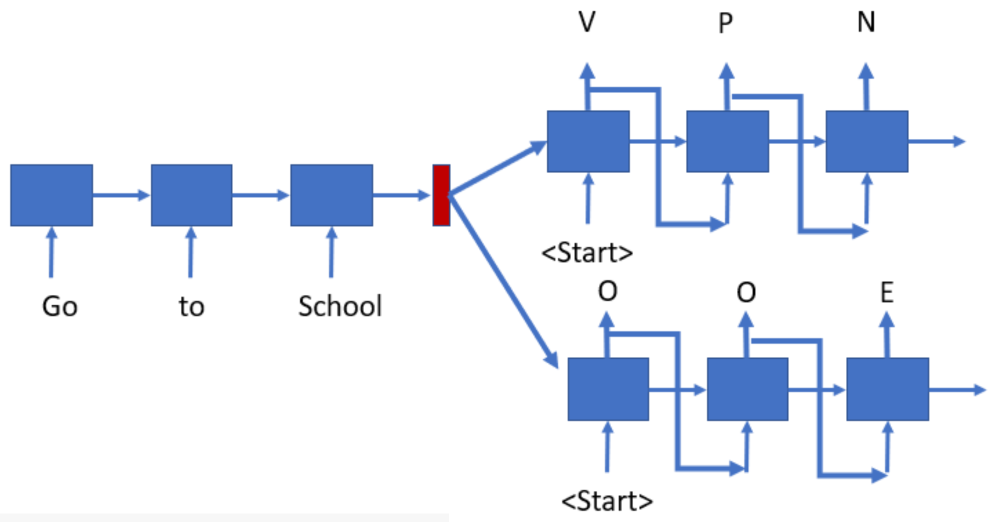
# A Simple Multi-Tasking Example




```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Dense

D = np.array([[4,500,6],
              [4,550,5.5],
              [2,200,3.5],
              [2,250,4]])
label = np.array([[1,1,0,0]]).T

## Creating the layers
input_layer = Input(shape=(3,))
layer_1 = Dense(4, activation="relu")(input_layer)
layer_2 = Dense(4, activation="relu")(layer_1)
layer_3 = Dense(4, activation="relu")(layer_2)
o1_layer= Dense(1, activation="linear")(layer_2)
o2_layer= Dense(1, activation="linear")(layer_3)

##Defining the model by specifying the input and output layers
model = Model(inputs=input_layer, outputs=[o1_layer,o2_layer])
model.summary()

## defining the optimiser and loss function
model.compile(optimizer='adam', loss='mse')

## training the model
model.fit(D, label,epochs=2, batch_size=128,validation_data=(D,label))
```

# Single Sequential input and Multiple Sequential outputs

```
from keras.models import Model
from keras.layers import Input, LSTM, Dense

# Define Encoder
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]

# Define Decoder 1
decoder1_inputs = Input(shape=(None, num_decoder1_tokens))
decoder1_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder1_outputs, _, _ = decoder1_lstm(decoder1_inputs,initial_state=encoder_states)
decoder1_dense = Dense(num_decoder1_tokens, activation='softmax')
decoder1_outputs = decoder_dense(decoder1_outputs)

# Define Decoder 2
decoder2_inputs = Input(shape=(None, num_decoder2_tokens))
decoder2_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder2_outputs, _, _ = decoder2_lstm(decoder2_inputs,initial_state=encoder_states)
decoder2_dense = Dense(num_decoder2_tokens, activation='softmax')
decoder2_outputs = decoder2_dense(decoder1_outputs)

# Define Model
model = Model([encoder_inputs, decoder1_inputs], [decoder1_outputs,decoder2_outputs])

# Run training
model.compile(optimizer='adam', loss='categorical_crossentropy')
model.fit([encoder_input_data, decoder1_input_data, decoder2_input_data], [decoder1_target_data,decoder2_target_data],
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
```



# VGG 16 with multiple outputs

```
from keras.applications.vgg16 import VGG16
vggmodel = VGG16(weights='imagenet', include_top=False,input_shape = (224,224,3) )

x = Flatten()(vggmodel.output) # Flattened the last layer
h1 = Dense( 2 , activation = 'softmax' )(x)
prediction1 = Dense( 2 , activation = 'softmax' )(h1) # First Task

h2= Dense( 2 , activation = 'softmax' )(x)
prediction2 = Dense( 2 , activation = 'softmax' )(h2# Second Task


model = Model( inputs = vggmodel.input , outputs = [prediction1, prediction2] ) # Join it with the model
```