Distillation Unleashed: Domain Knowledge Transfer with Compact Neural Networks

Hadi Abdi Khojasteh

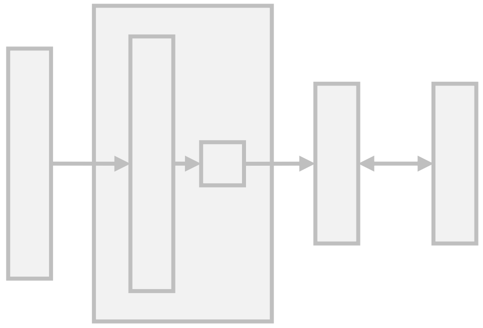
hadi.abdikhojasteh@deltatre.com

September 16th, 2023

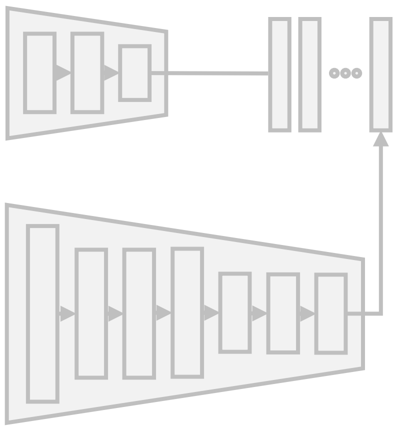
Presentation will be available online.

PyData Amsterdam 2023

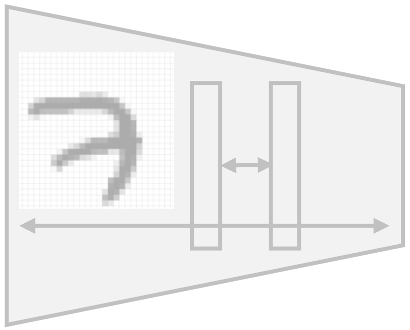
Overview



Introduction

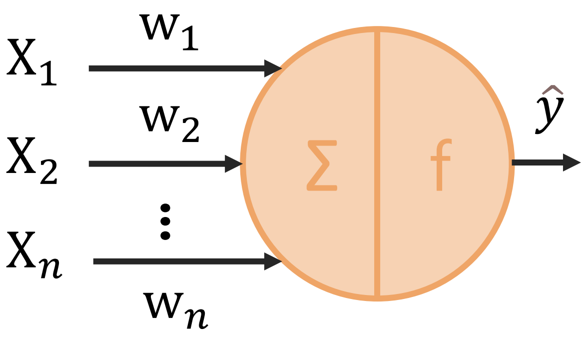


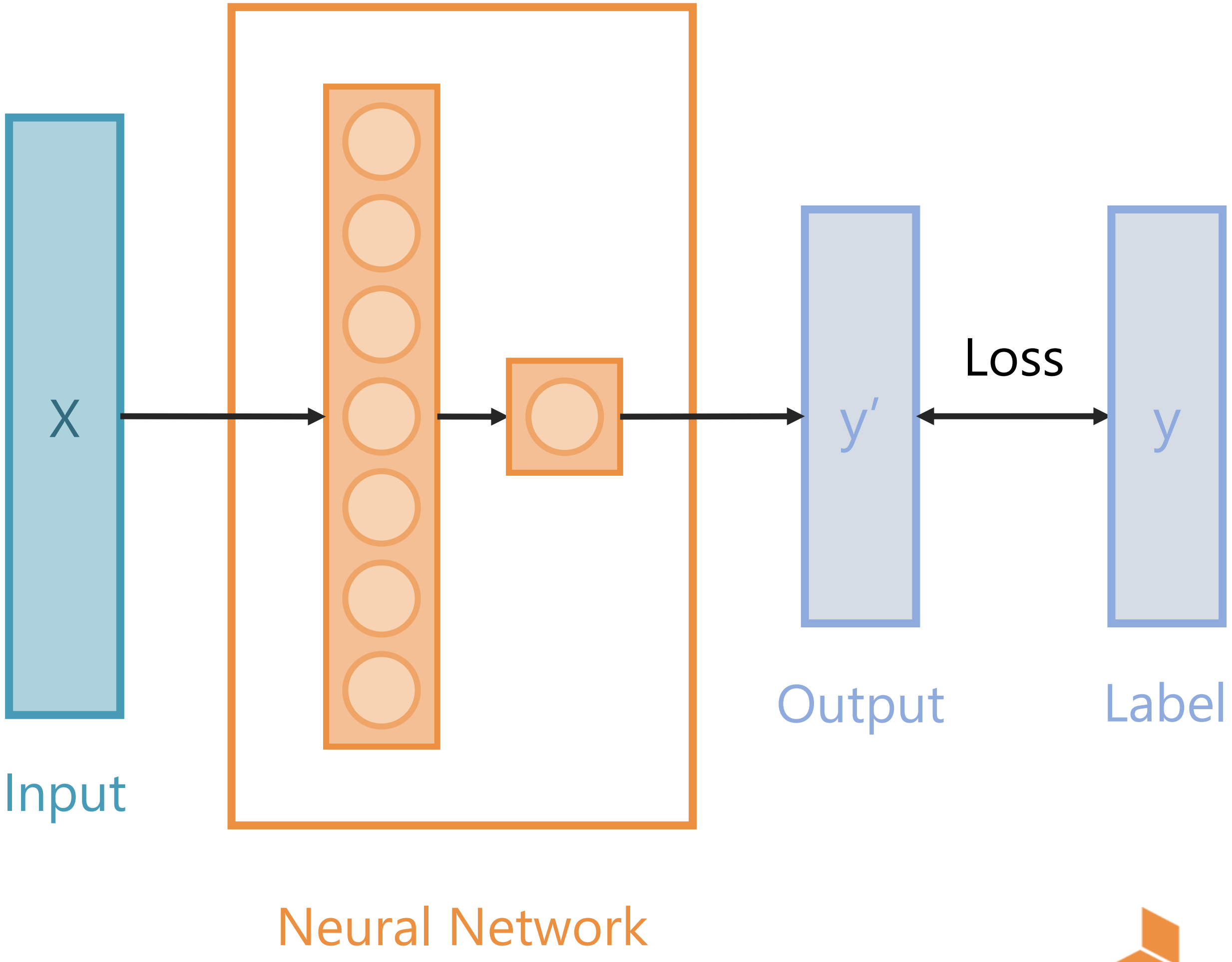
Variations



Implementation

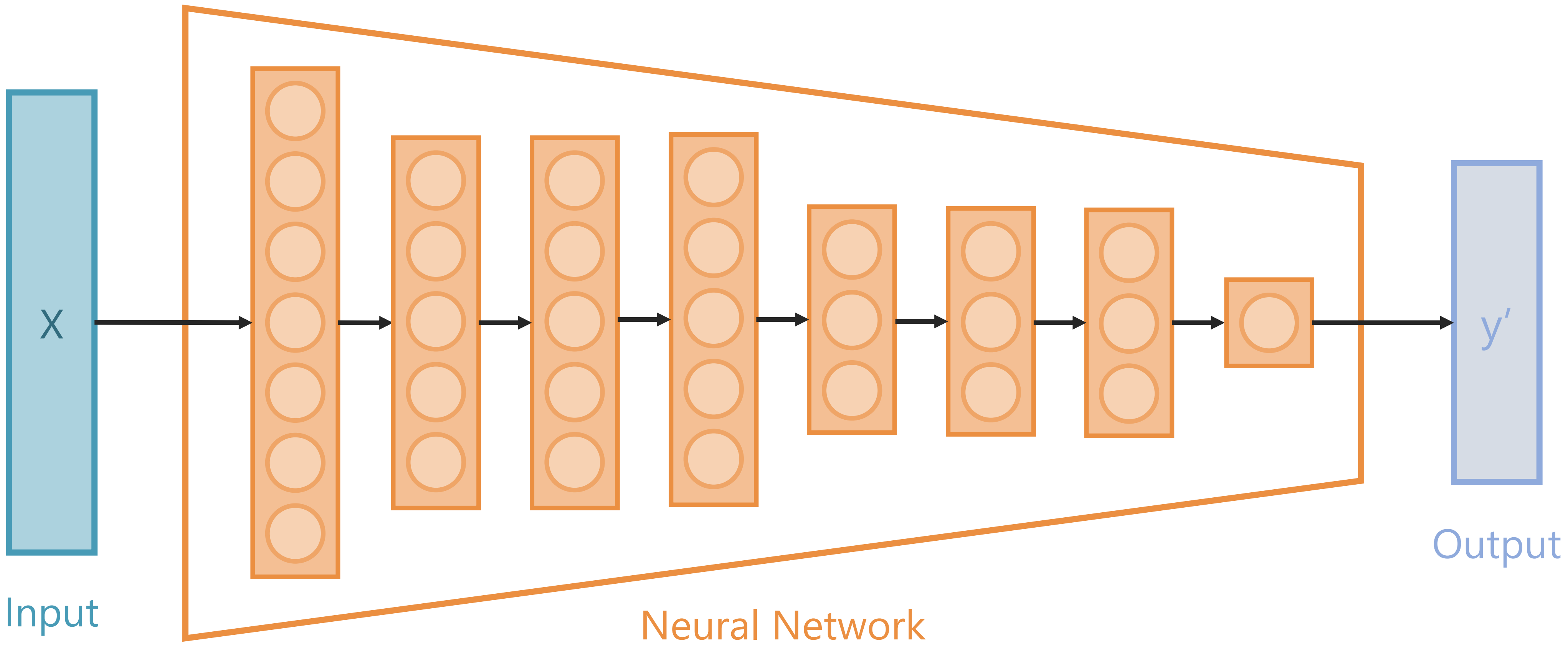
Introduction





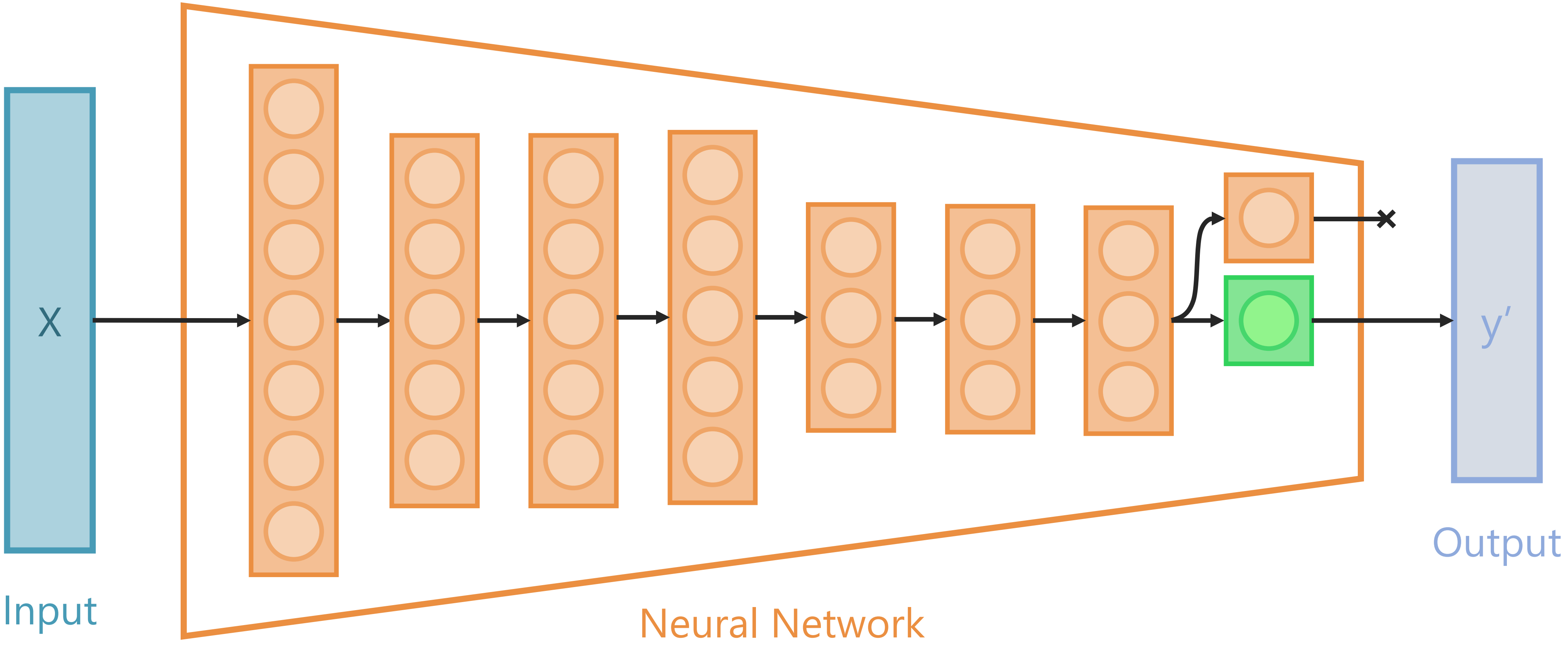
PyData
Amsterdam 2023

Deep Neural Network Demonstration



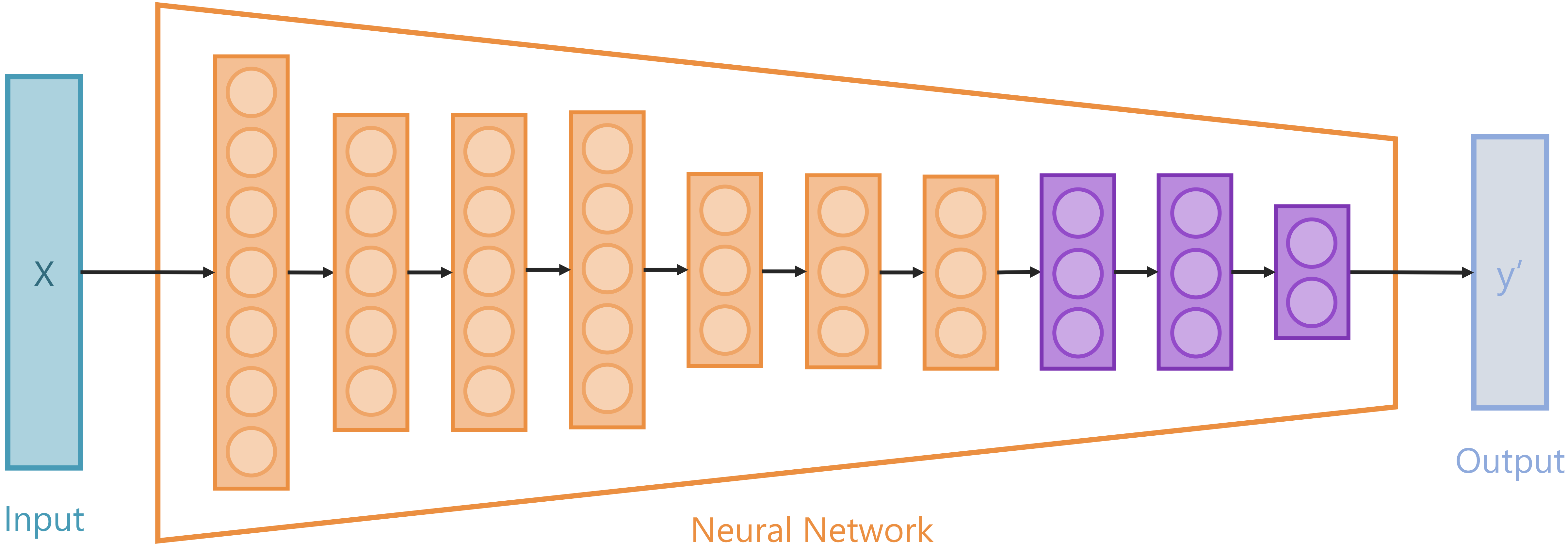
PyData
Amsterdam 2023

Transfer Learning

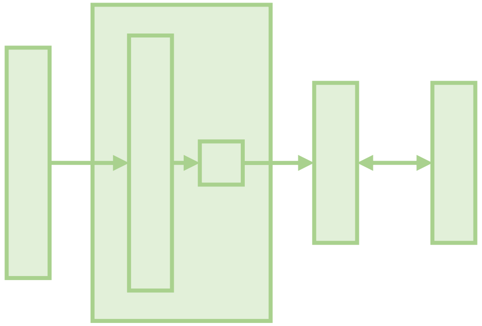


PyData
Amsterdam 2023

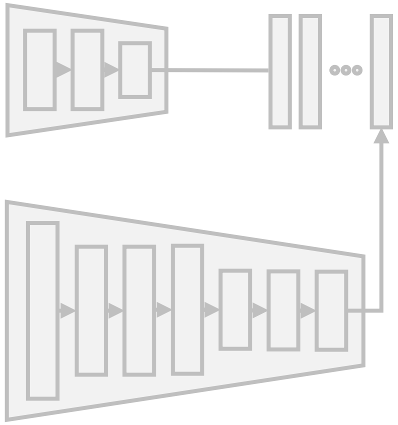
Transfer Learning



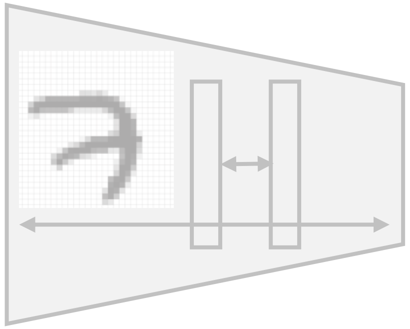
Overview



Introduction

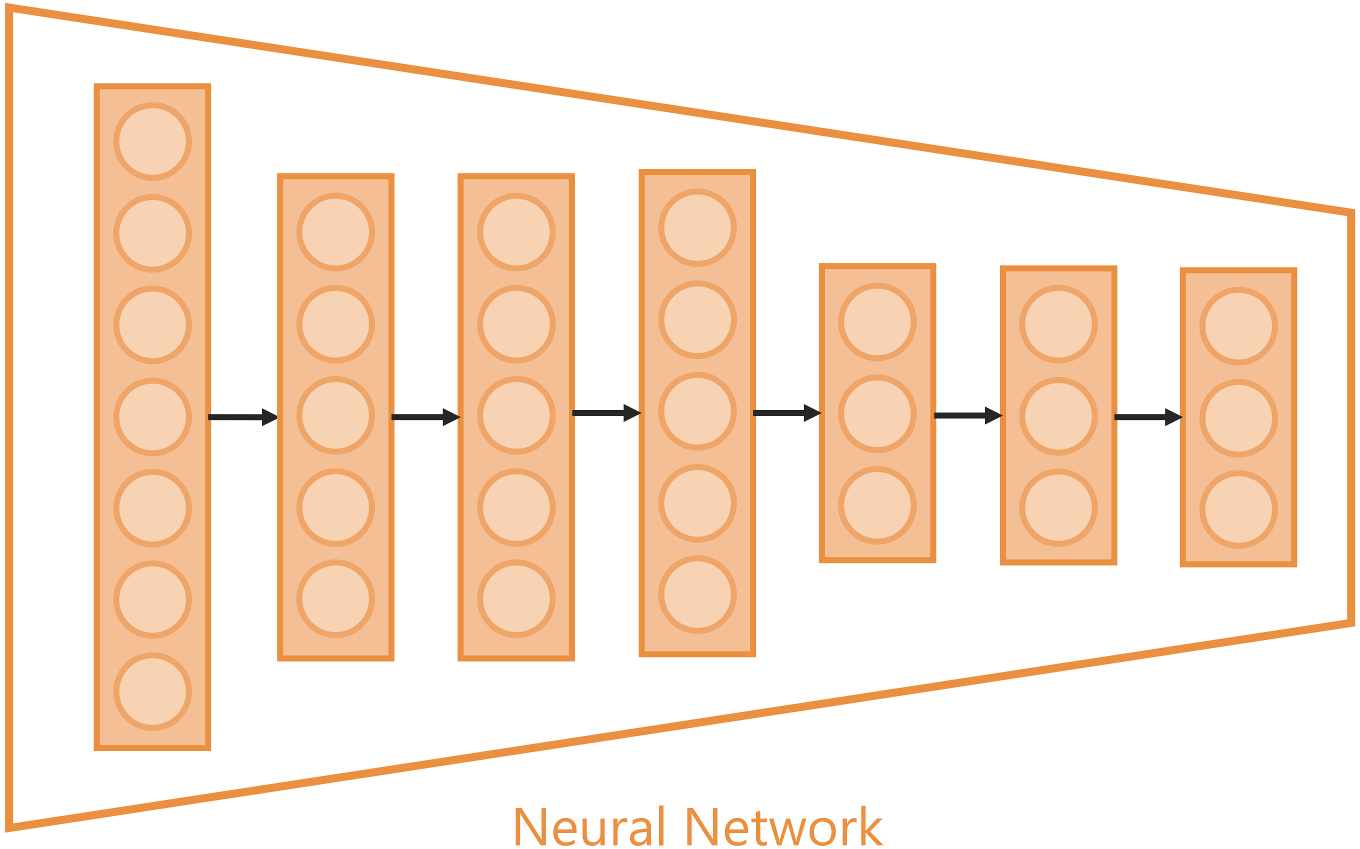


Variations

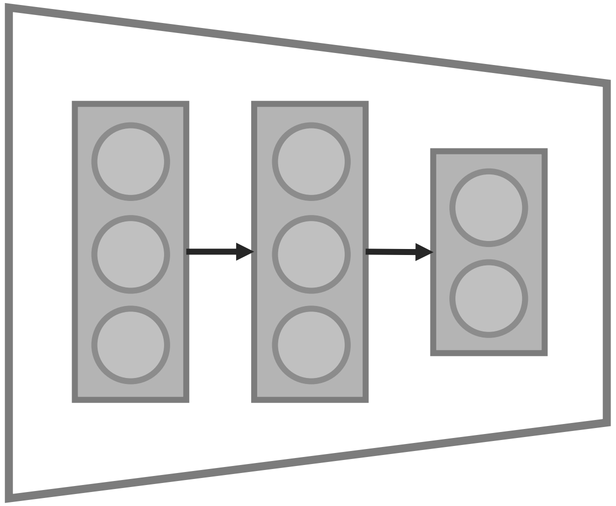


Implementation

Knowledge Transfer

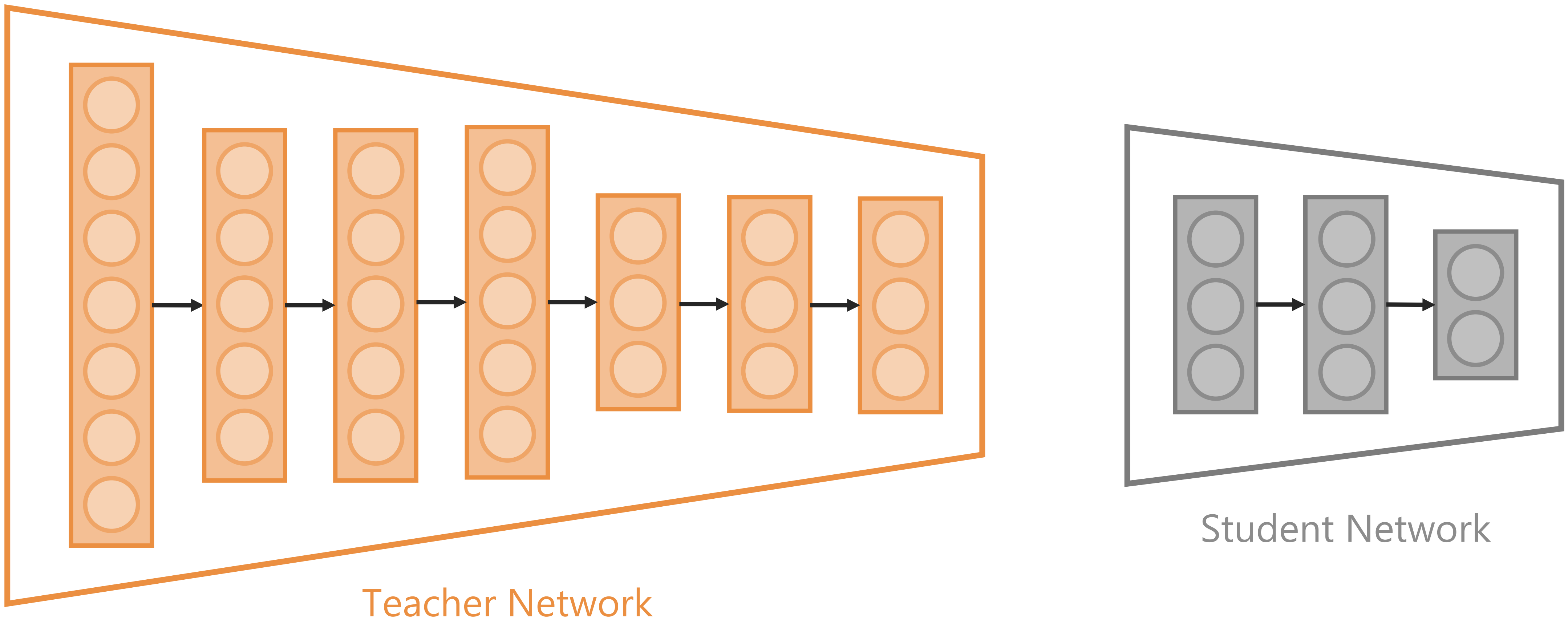


Neural Network

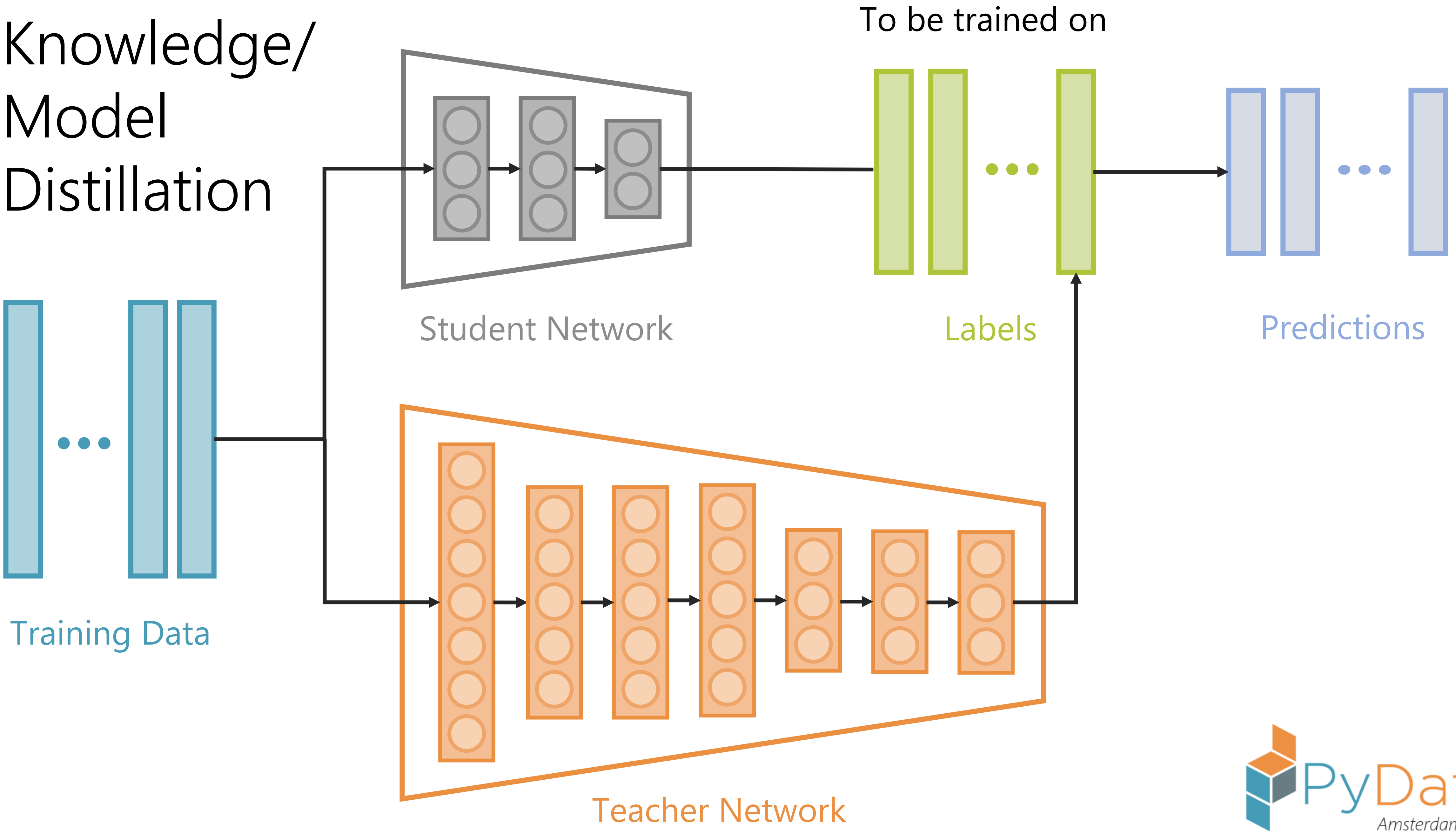


Neural Network

Knowledge Transfer

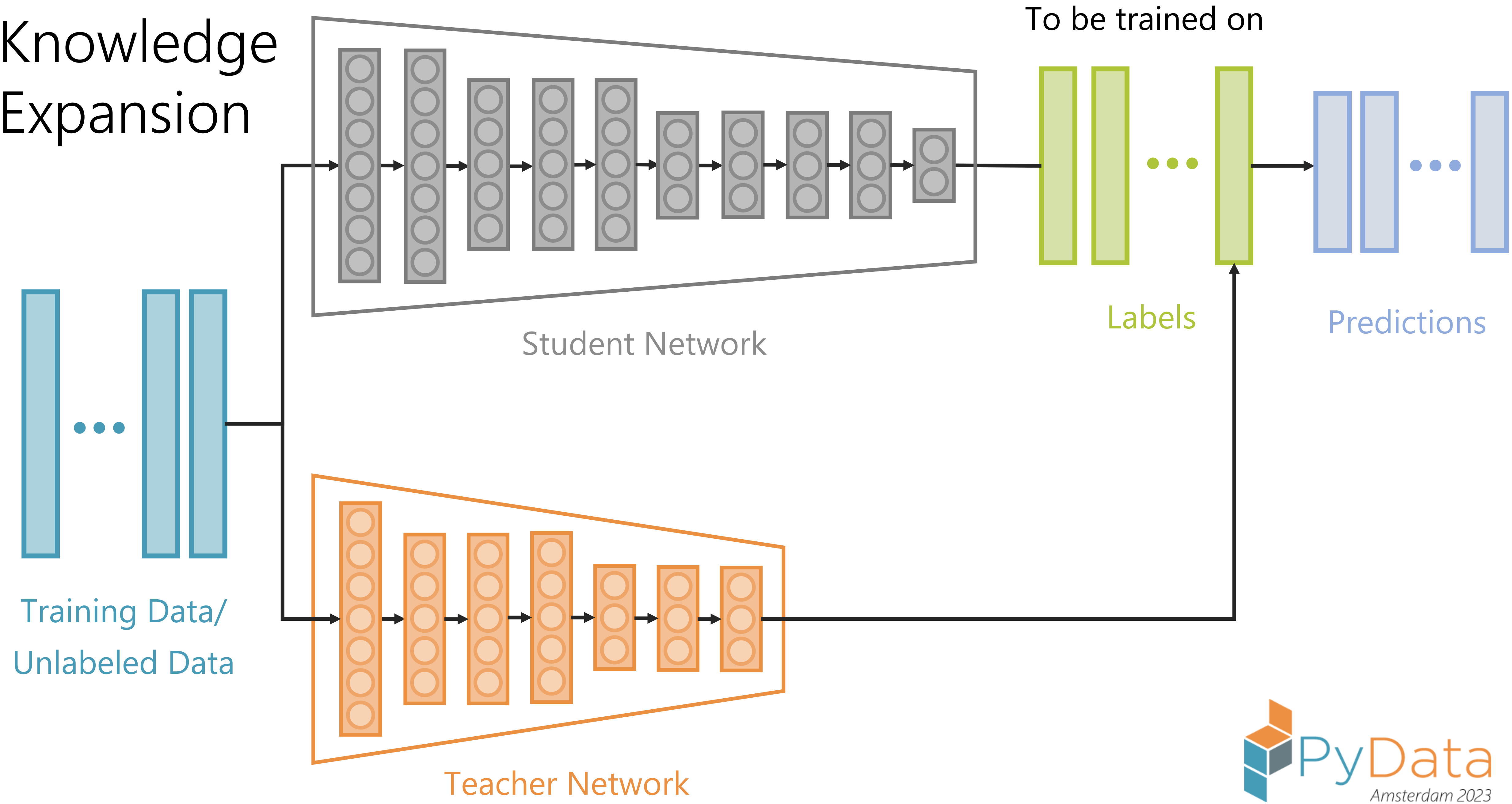


Knowledge/ Model Distillation

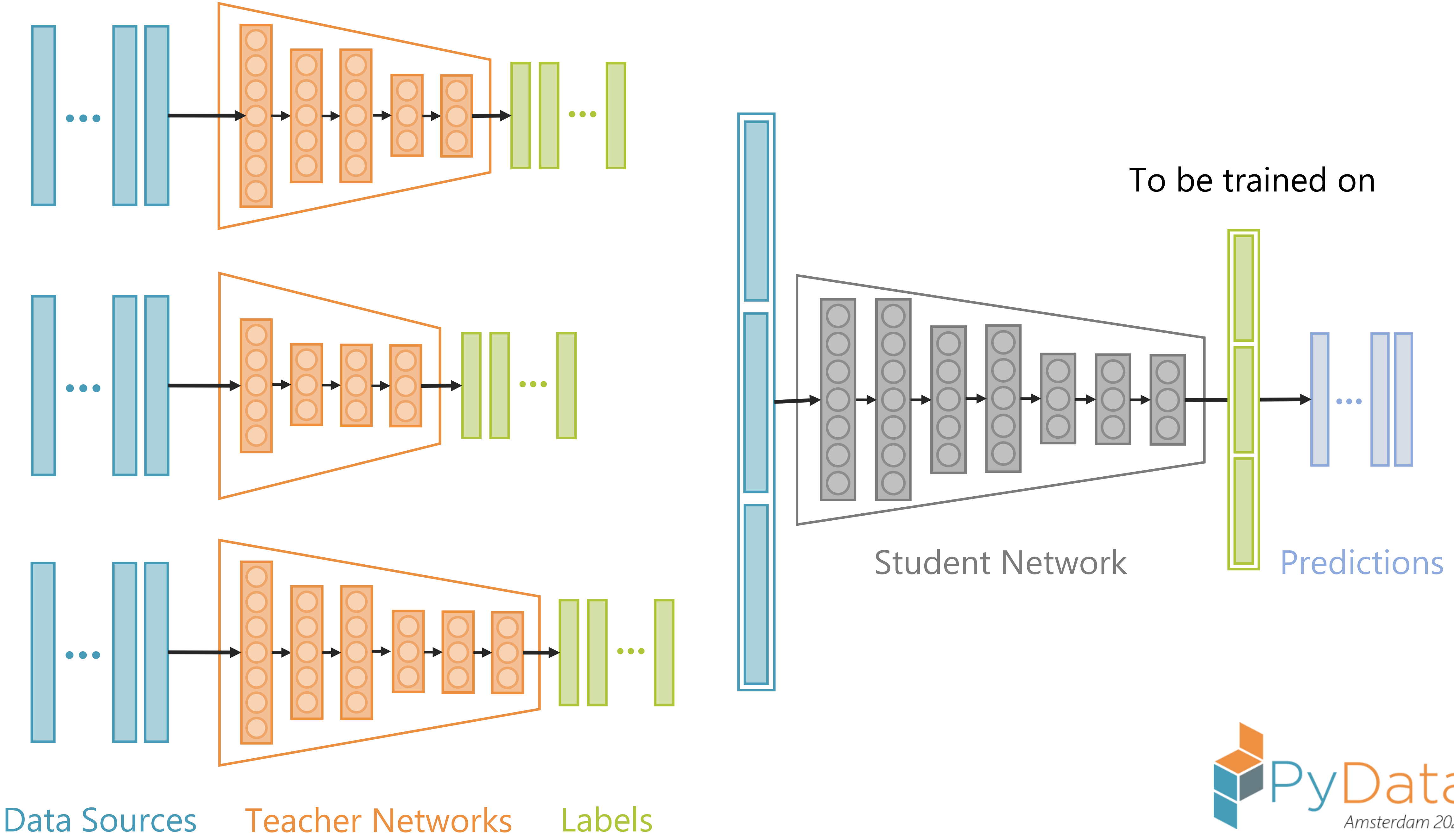


PyData Amsterdam 2023

Knowledge Expansion

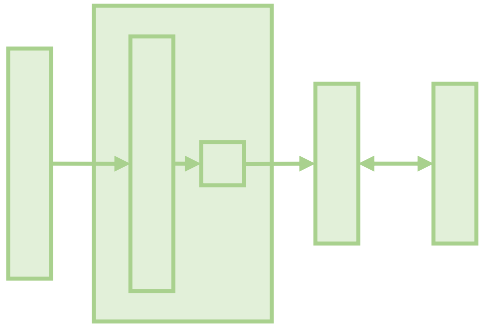


Multi-task Distillation

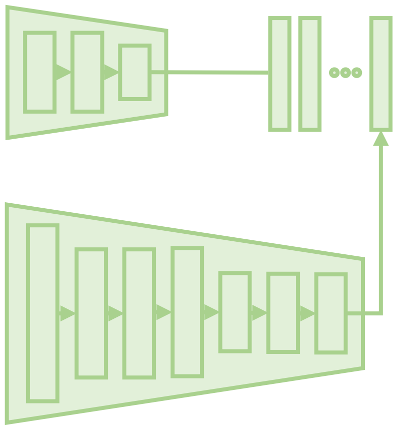


PyData Amsterdam 2023

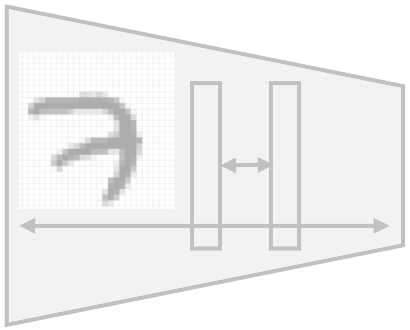
Overview



Introduction

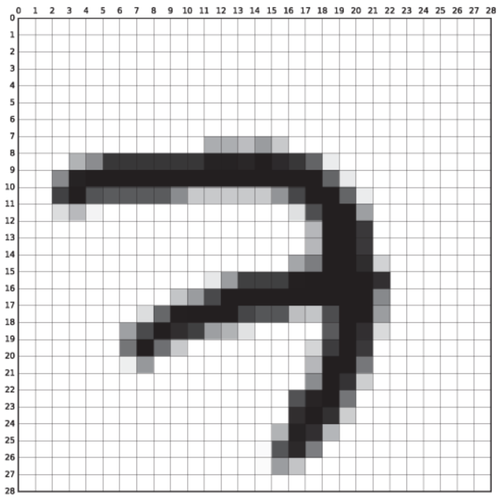


Variations



Implementation

Distillation Learning in Action



First, we import the TensorFlow library:

```python
# Importing libraries
import tensorflow as tf
```

Then load MNIST dataset and normalize the data:

```python
# Loading the data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Preprocessing the data
x_train = x_train.reshape(-1, 784).astype('float32') / 255
x_test = x_test.reshape(-1, 784).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
```

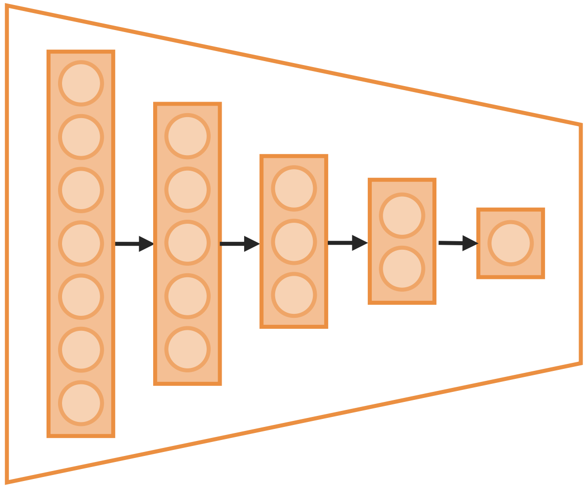
0,0,0,0,0,0,0,0,0,1

0 1 2 3 4 5 6 7 8 9

PyData Amsterdam 2023

Next, we define the teacher model:

```
# Defining the teacher model (More complex and larger neural network)
class TeacherModel(tf.keras.Model):
    def __init__(self):
        super(TeacherModel, self).__init__()
        self.layer1 = tf.keras.layers.Dense(512, input_shape=(784,), activation='relu')
        self.layer2 = tf.keras.layers.Dense(256, activation='relu')
        self.layer3 = tf.keras.layers.Dense(128, activation='relu')
        self.layer4 = tf.keras.layers.Dense(64, activation='relu')
        self.last = tf.keras.layers.Dense(10, activation='softmax')

    def call(self, x):
        # Defining the forward pass
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return self.last(x)
```

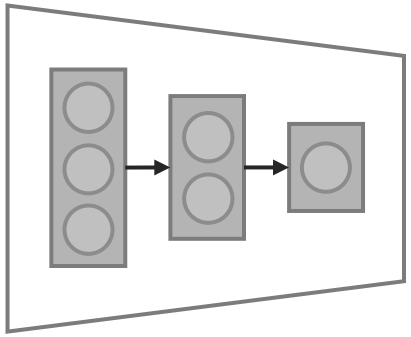


Teacher Network

PyData Amsterdam 2023

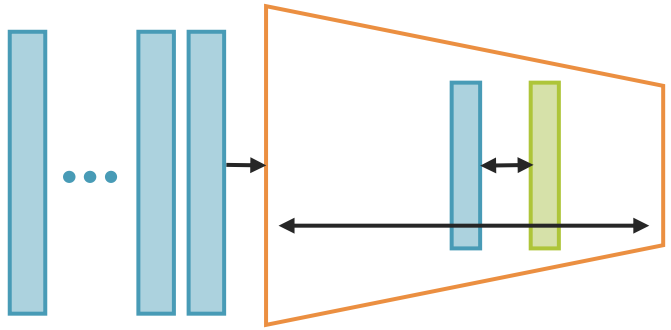
And an student model which is shallower than teacher model:

```python
# Defining the student model (More efficient and smaller neural network)
class StudentModel(tf.keras.Model):
    def __init__(self):
        super(StudentModel, self).__init__()
        self.layer1 = tf.keras.layers.Dense(512, input_shape=(784,), activation='relu’)
        self.layer2 = tf.keras.layers.Dense(256, activation='relu’)
        self.last = tf.keras.layers.Dense(10, activation='softmax’)

    def call(self, x):
        # Defining the forward pass
        x = self.layer1(x)
        x = self.layer2(x)
        return self.last(x)
```



Student Network

Afterward we call and load the teacher model first, define some of its hyperparameters and finally train it as follows:

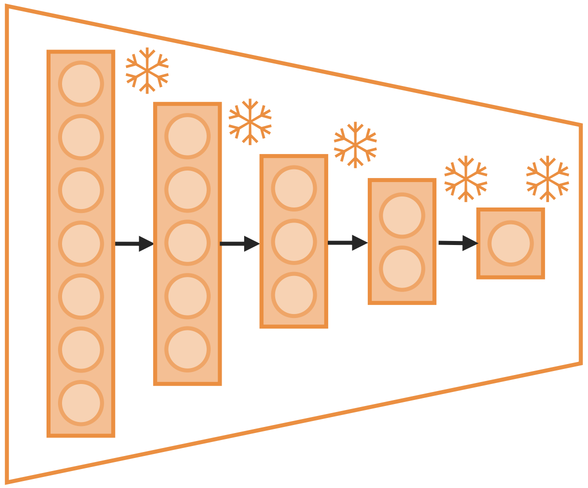


```python
# Loading the teacher model
teacher_model = TeacherModel()

# Defining the loss function and optimizer
loss_fn = tf.keras.losses.CategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()

# Training the teacher model
teacher_model.compile(optimizer=optimizer, loss=loss_fn, metrics=['accuracy'])
teacher_model.fit(x_train, y_train, epochs=5, batch_size=32,\
                  validation_data=(x_test, y_test))
```

```
Epoch 1/5 1875/1875 - loss: 0.2074 - accuracy: 0.9377 - val_loss: 0.1059 - val_accuracy: 0.9694
Epoch 2/5 1875/1875 - loss: 0.0941 - accuracy: 0.9717 - val_loss: 0.0878 - val_accuracy: 0.9736
Epoch 3/5 1875/1875 - loss: 0.0675 - accuracy: 0.9795 - val_loss: 0.0722 - val_accuracy: 0.9773
Epoch 4/5 1875/1875 - loss: 0.0530 - accuracy: 0.9842 - val_loss: 0.0808 - val_accuracy: 0.9776
Epoch 5/5 1875/1875 - loss: 0.0445 - accuracy: 0.9864 - val_loss: 0.0813 - val_accuracy: 0.9782
```

When teacher model training is done, we freeze teacher model layers to be able to use it in the student model:

```
# Loading the student model
student_model = StudentModel()

# Freezing the teacher model layers
for layer in teacher_model.layers:
    layer.trainable = False
```



Teacher Network

The loss function for student model is calculated with respect to the teacher model output not the true values from the training data:

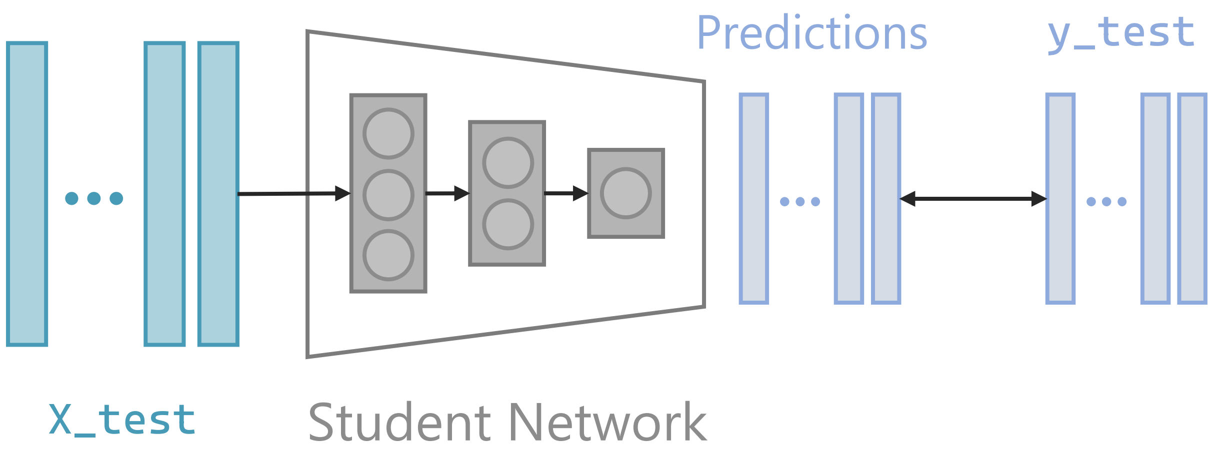
```
# Defining the distillation loss function
temp = 5
def distillation_loss(y_true, y_pred):
    y_true = tf.nn.softmax(y_true / temp)
    y_pred = tf.nn.softmax(y_pred / temp)
    return tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true, y_pred))

# Train the student model
student_model.compile(optimizer=optimizer, loss=distillation_loss, metrics=['accuracy'])
student_model.fit(x_train, teacher_model.predict(x_train), epochs=5, batch_size=32,
validation_data=(x_test, y_test))
```

```
1875/1875
Epoch 1/5 - loss: 2.3008 - accuracy: 0.9404 - val_loss: 2.3007 - val_accuracy: 0.9643
Epoch 2/5 - loss: 2.3007 - accuracy: 0.9741 - val_loss: 2.3007 - val_accuracy: 0.9721
Epoch 3/5 - loss: 2.3007 - accuracy: 0.9796 - val_loss: 2.3007 - val_accuracy: 0.9719
Epoch 4/5 - loss: 2.3007 - accuracy: 0.9837 - val_loss: 2.3007 - val_accuracy: 0.9761
Epoch 5/5 - loss: 2.3007 - accuracy: 0.9862 - val_loss: 2.3007 - val_accuracy: 0.9760
          #loss: 0.0445 - accuracy: 0.9864 - val_loss: 0.0813 - val_accuracy: 0.9782
```

Finally, we run the inference on the dataset:

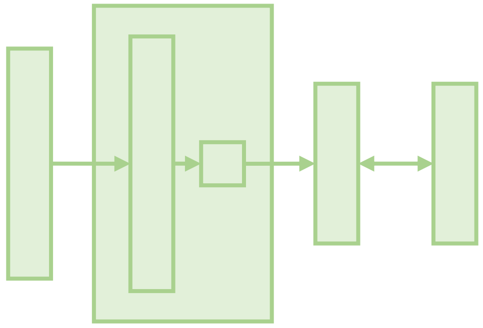
```python
# Evaluate the student model
test_loss, test_acc = student_model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

```
313/313 [============================] - 1s 2ms/step - loss: 2.3007 - accuracy: 0.9760
Test accuracy: 0.9760000109672546
```

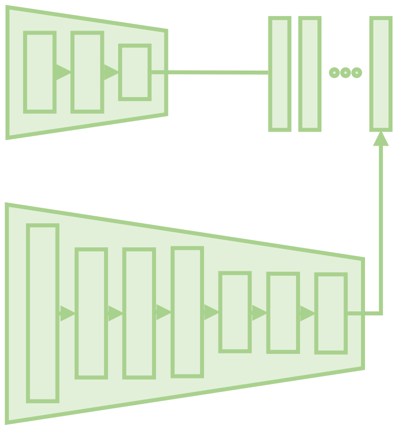


PyData
Amsterdam 2023

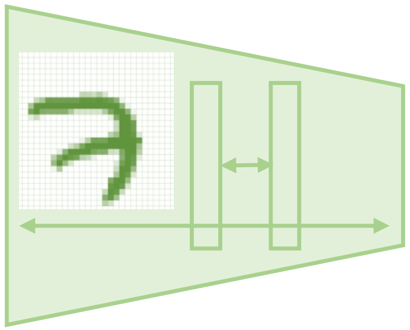
Overview



Introduction



Variations



Implementation

PyData Amsterdam 2023

Further References

Below are the recommended researches for a comprehensive exploration and in-depth understanding of the topic:

KD++	Improving Knowledge Distillation via Regularizing Feature Norm and Direction, 2023
Closer Look	A closer look at the training dynamics of knowledge distillation, 2023
DIST	Knowledge Distillation from A Stronger Teacher, 2022
ADLIK-Faster	Focal and Global Knowledge Distillation for Detectors, 2021
LSHFM	Distilling Knowledge by Mimicking Features, 2020
ADLIK-MO-P25	Ensemble Knowledge Distillation for Learning Improved and Efficient Networks, 2019
KD/ADLIK-MO	Distilling the Knowledge in a Neural Network, 2015

Further References

torchdistill (formerly kdkit) simplifies knowledge distillation through declarative config files, eliminating the need for extensive code. It also supports reproducible deep learning studies and offers flexibility for various experiments.

```
# Fine-tune Transformer models for GLUE CoLA task
!accelerate launch torchdistill/examples/legacy/hf_transformers/text_classification.py \
  --config torchdistill/configs/legacy/sample/glue/cola/ce/bert_base_uncased.yaml \
  --task cola \
  --log log/glue/cola/ce/bert_base_uncased.txt \
  --private_output leaderboard/glue/standard/bert_base_uncased/
```

Thank you for your attention :)

Distillation Unleashed:
Domain Knowledge Transfer with Compact Neural Networks

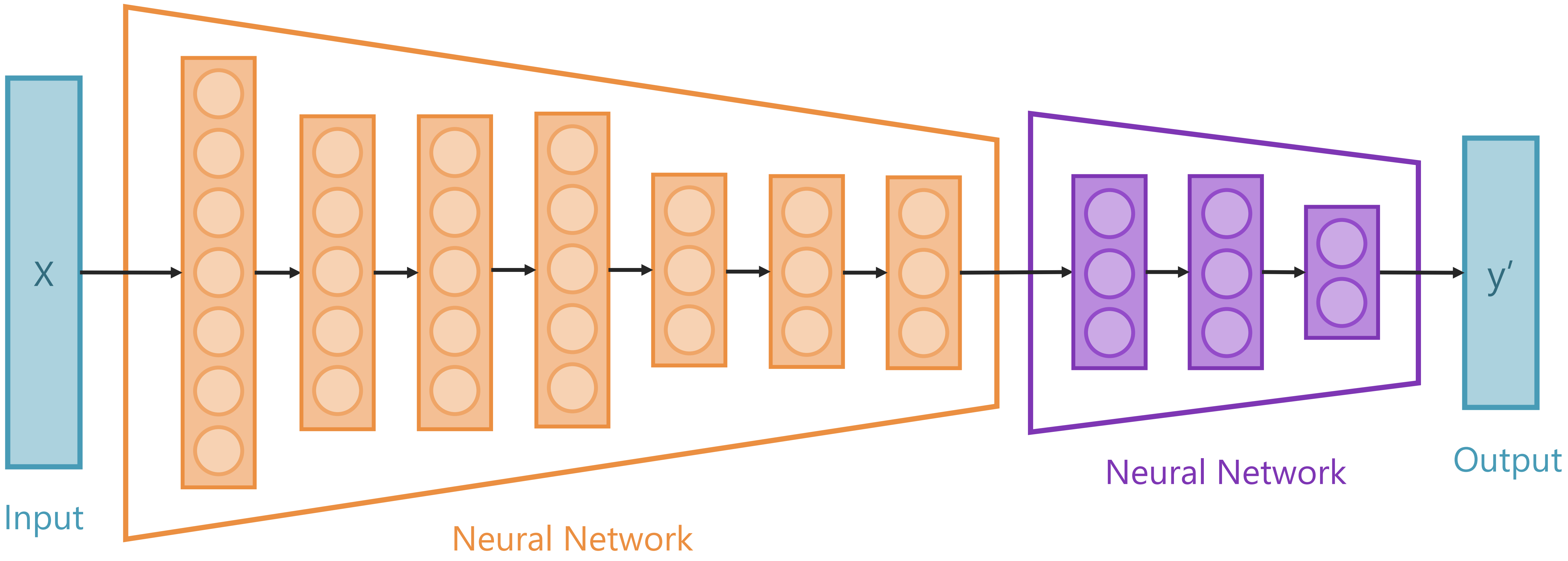
You can find me in

Presentation will be available online.

Amsterdam 2023

PyData
Amsterdam 2023

Transfer Learning



PyData
Amsterdam 2023