

Sequential Model Example

```
seq_model = Sequential([  
    Flatten(input_shape=(28, 28)),  
    Dense(128, activation='relu'),  
    Dense(10, activation='softmax')  
])
```

Sequential Model Example

```
seq_model = Sequential([  
    Flatten(input_shape=(28, 28)),  
    Dense(128, activation='relu'),  
    Dense(10, activation='softmax')  
])
```

Sequential Model Example

```
seq_model = Sequential([  
    Flatten(input_shape=(28, 28)),  
    Dense(128, activation='relu'),  
    Dense(10, activation='softmax')  
])
```

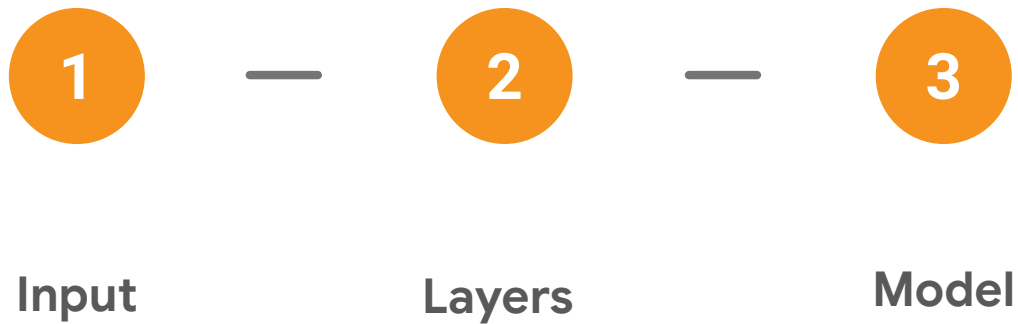
Sequential Model Example

```
seq_model = Sequential([  
    Flatten(input_shape=(28, 28)),  
    Dense(128, activation='relu'),  
    Dense(10, activation='softmax')  
])
```

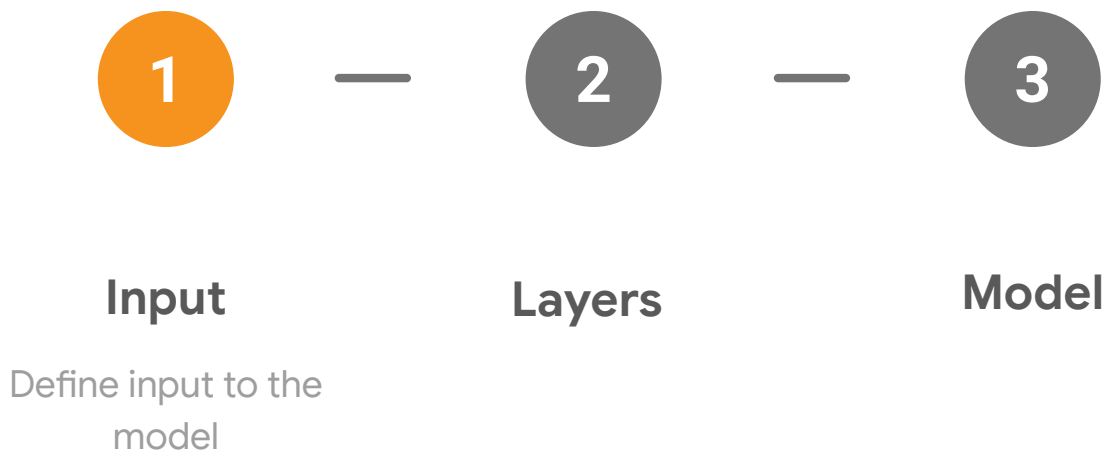
Sequential Model Example

```
seq_model = Sequential([  
    Flatten(input_shape=(28, 28)),  
    Dense(128, activation='relu'),  
    Dense(10, activation='softmax')  
])
```

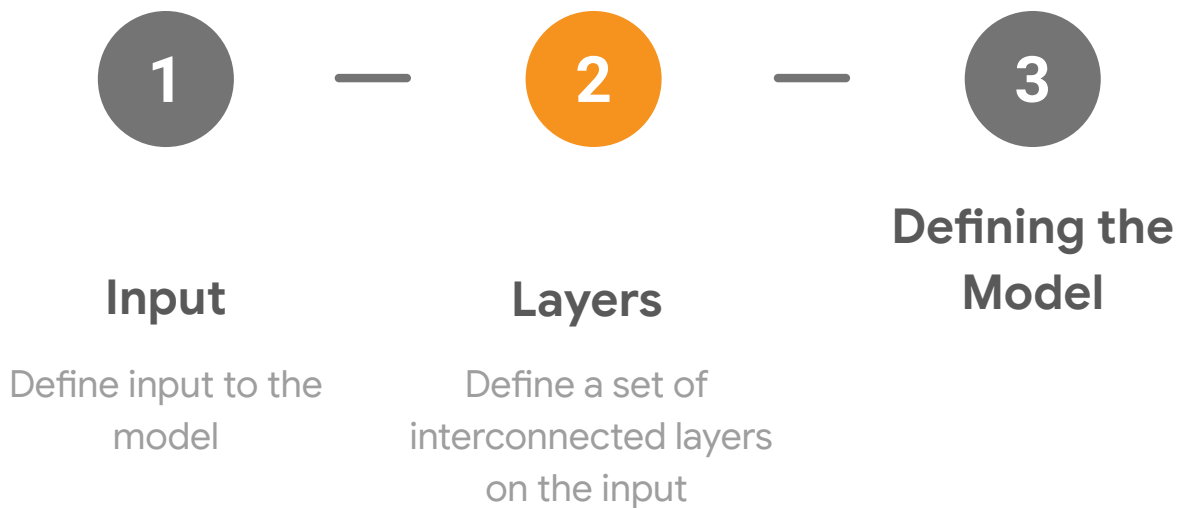
Functional API



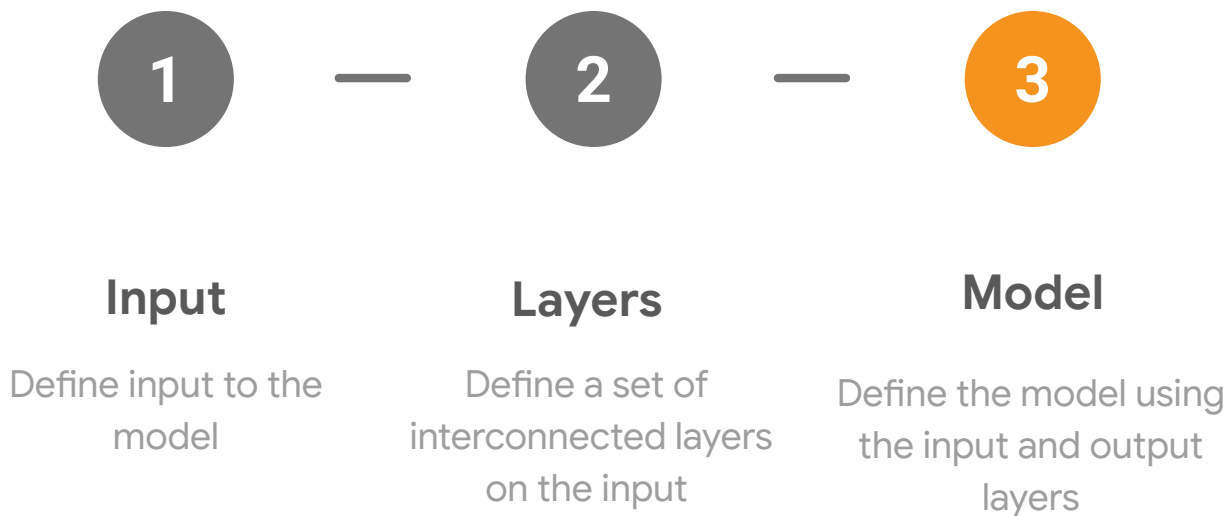
Functional API



Functional API



Functional API



Defining the Input

```
from tensorflow.keras.layers import Input
```

```
...
```

```
input = Input(shape=(28, 28))
```

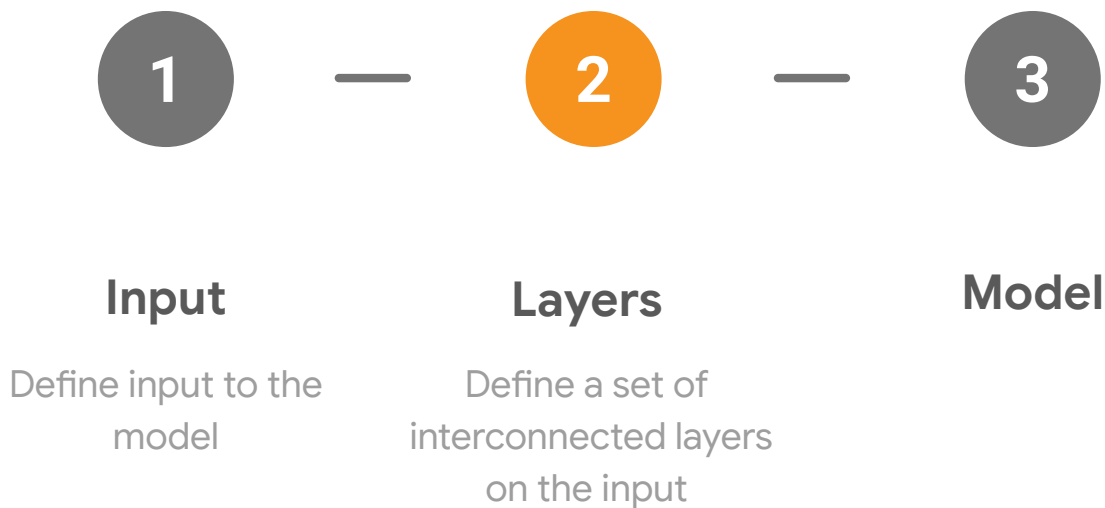
Defining the Input

```
from tensorflow.keras.layers import Input
```

```
...
```

```
input = Input(shape=(28, 28))
```

Functional API



Defining the layers

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

```
...
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

Defining the layers

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

```
...
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

Defining the layers

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

```
...
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

Defining the layers

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

```
...
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```


Defining the layers

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

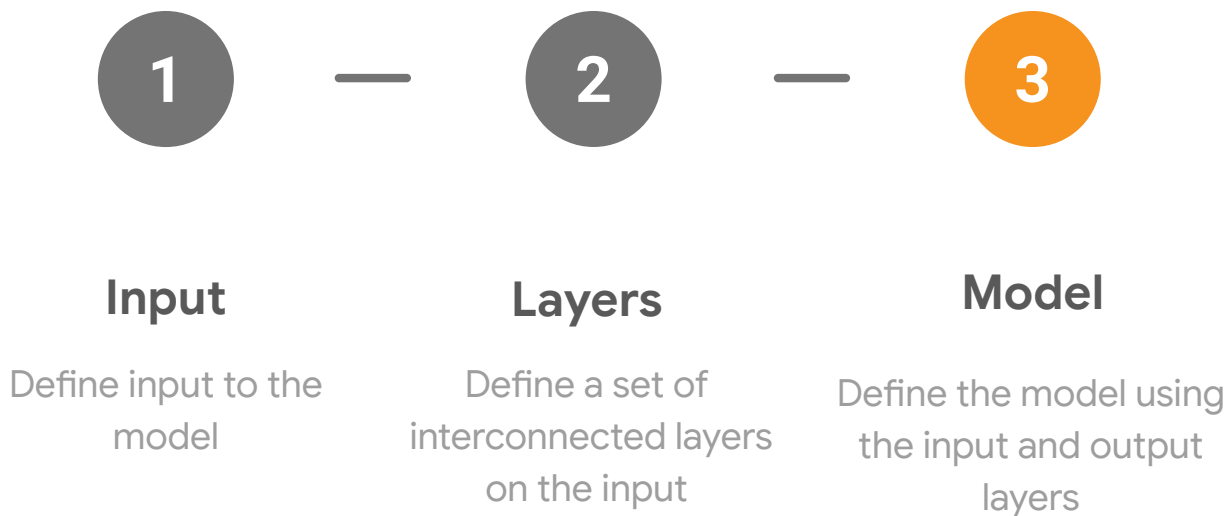
```
...
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

Functional API



Defining the Model

```
from tensorflow.keras.models import Model
```

```
...
```

```
func_model = Model(inputs=input, outputs=predictions)
```

Defining the Model

```
from tensorflow.keras.models import Model
```

```
...
```

```
func_model = Model(inputs=input, outputs=predictions)
```

Defining the Model

```
from tensorflow.keras.models import Model
```

```
...
```

```
func_model = Model(inputs=input, outputs=predictions)
```

Defining the Model

```
from tensorflow.keras.models import Model
```

```
...
```

```
func_model = Model(inputs=input, outputs=predictions)
```

Rewriting Using Functional API

```
input = Input(shape=(28,28))  
x = Flatten()(input)  
x = Dense(128, activation="relu")(x)  
predictions = Dense(10, activation="softmax")(x)  
  
func_model = Model(inputs=input, outputs=predictions)
```

Rewriting Using Functional API

```
input = Input(shape=(28,28))
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

```
func_model = Model(inputs=input, outputs=predictions)
```


Rewriting Using Functional API

```
input = Input(shape=(28,28))
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

```
func_model = Model(inputs=input, outputs=predictions)
```

Rewriting Using Functional API

```
input = Input(shape=(28,28))
```

```
x = Flatten()(input)
```

```
x = Dense(128, activation="relu")(x)
```

```
predictions = Dense(10, activation="softmax")(x)
```

```
func_model = Model(inputs=input, outputs=predictions)
```

Rewriting Using Functional API

```
input = Input(shape=(28,28))  
x = Flatten()(input)  
x = Dense(128, activation="relu")(x)  
predictions = Dense(10, activation="softmax")(x)
```

```
func_model = Model(inputs=input, outputs=predictions)
```

```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```

```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```

```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```

```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```

```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```



```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```

```
first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)
```

```
first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)
```

```
first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)
```

```
first_dense(flatten_layer)
```



```
layer1 = Dense(32)
layer2_1 = Dense(32)(layer1)
layer2_2 = Dense(32)(layer1)
layer2_3 = Dense(32)(layer1)
layer2_4 = Dense(32)(layer1)
merge = Concatenate([layer2_1, layer2_2, layer2_3, layer2_4])
```

```
layer1 = Dense(32)
```


```
layer2_1 = Dense(32)(layer1)
```

```
layer2_2 = Dense(32)(layer1)
```

```
layer2_3 = Dense(32)(layer1)
```

```
layer2_4 = Dense(32)(layer1)
```

```
merge = Concatenate([layer2_1, layer2_2, layer2_3, layer2_4])
```



Layer 1

```
layer1 = Dense(32)
```

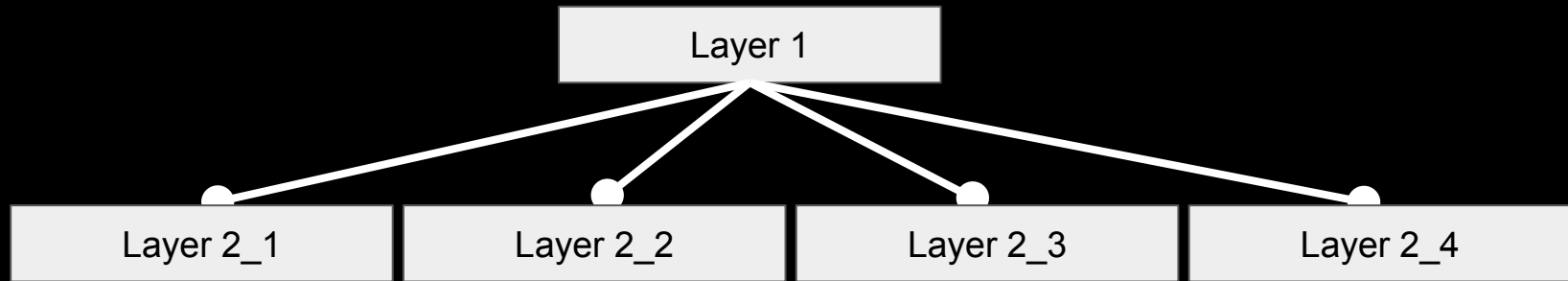
```
layer2_1 = Dense(32)(layer1)
```

```
layer2_2 = Dense(32)(layer1)
```

```
layer2_3 = Dense(32)(layer1)
```

```
layer2_4 = Dense(32)(layer1)
```

```
merge = Concatenate([layer2_1, layer2_2, layer2_3, layer2_4])
```



```
layer1 = Dense(32)
```

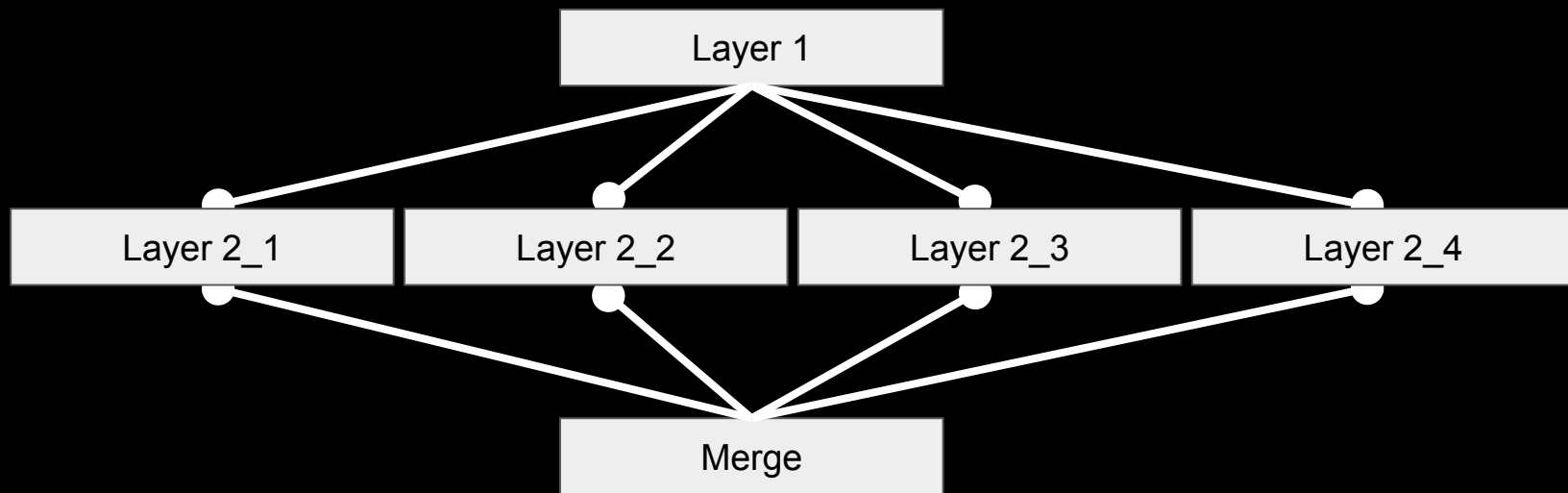
```
layer2_1 = Dense(32)(layer1)
```

```
layer2_2 = Dense(32)(layer1)
```

```
layer2_3 = Dense(32)(layer1)
```

```
layer2_4 = Dense(32)(layer1)
```

```
merge = Concatenate([layer2_1, layer2_2, layer2_3, layer2_4])
```



```
def build_model_with_functional():  
    from tensorflow.keras.models import Model  
    input_layer = tf.keras.Input(shape=(28, 28))  
    flatten_layer = tf.keras.layers.Flatten()(input_layer)  
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.relu)(flatten_layer)  
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(first_dense)  
    func_model = Model(inputs=input_layer, outputs=output_layer)  
    return func_model
```

```
func_model = Model(inputs=input_layer, outputs=output_layer)
```

```
func_model = Model(inputs=[input1, input2], outputs=[output1, output2])
```

```
func_model = Model(inputs=[input1, input2], outputs=[output1, output2])
```


<https://archive.ics.uci.edu/ml/datasets/Energy+efficiency>



Machine Learning Repository

[Center for Machine Learning and Intelligent Systems](#)

Energy efficiency Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: This study looked into assessing the heating load and cooling load requirements of buildings (that is, energy efficiency) as a function of building parameters.

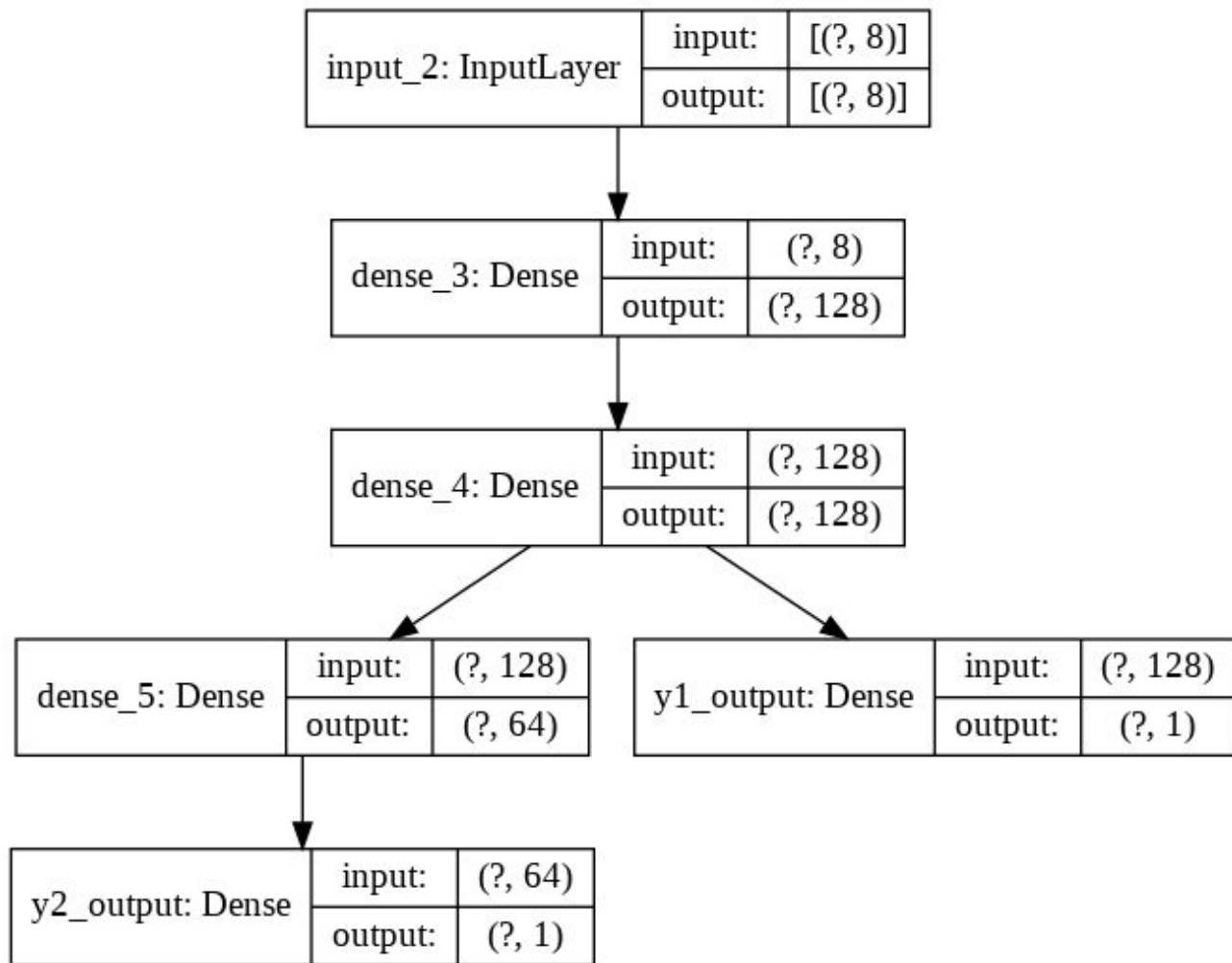
Data Set Characteristics:	Multivariate	Number of Instances:	768	Area:	Computer
Attribute Characteristics:	Integer, Real	Number of Attributes:	8	Date Donated	2012-11-30
Associated Tasks:	Classification, Regression	Missing Values?	N/A	Number of Web Hits:	301947

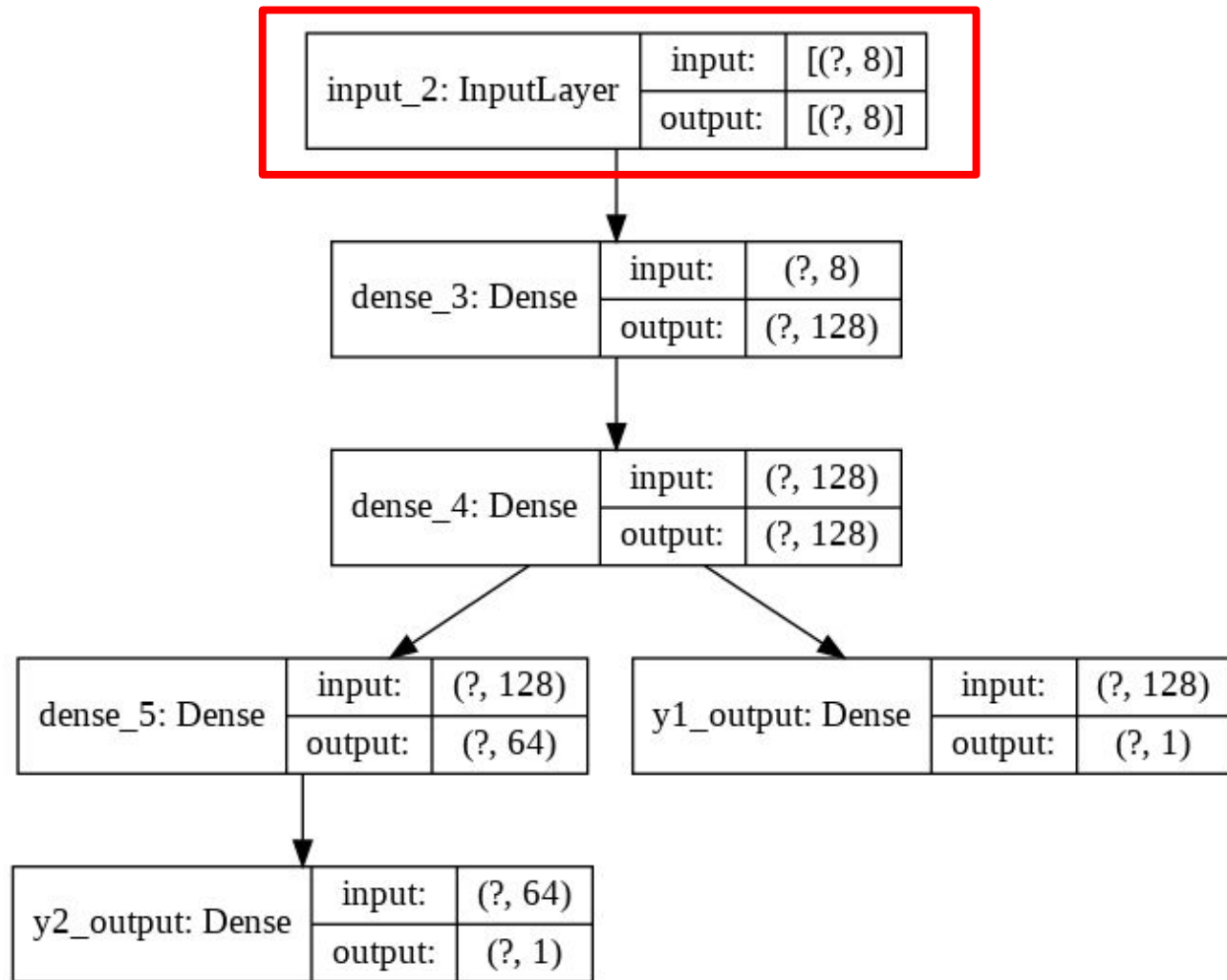
Features

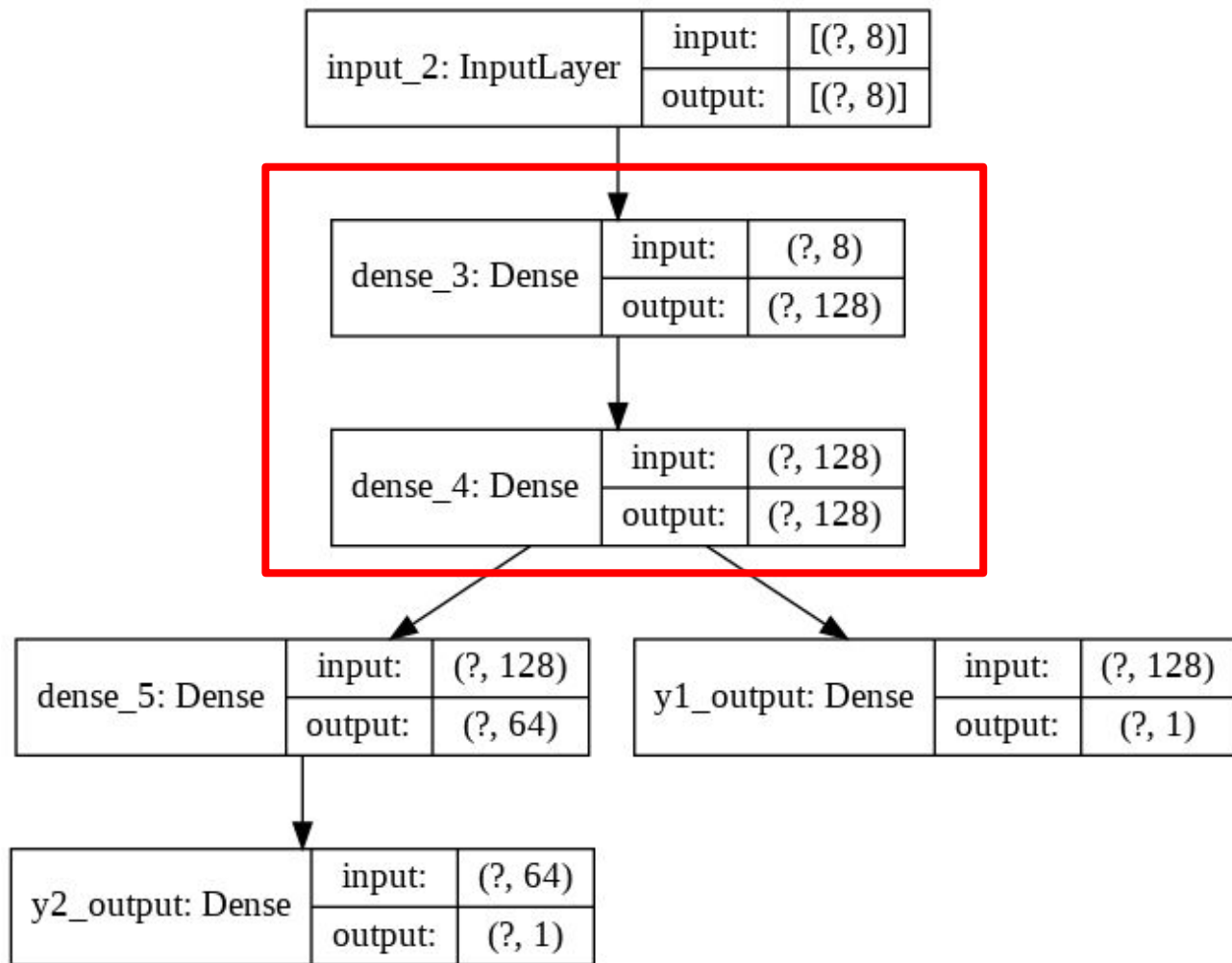
X1 Relative Compactness
X2 Surface Area
X3 Wall Area
X4 Roof Area
X5 Overall Height
X6 Orientation
X7 Glazing Area
X8 Glazing Area Distribution

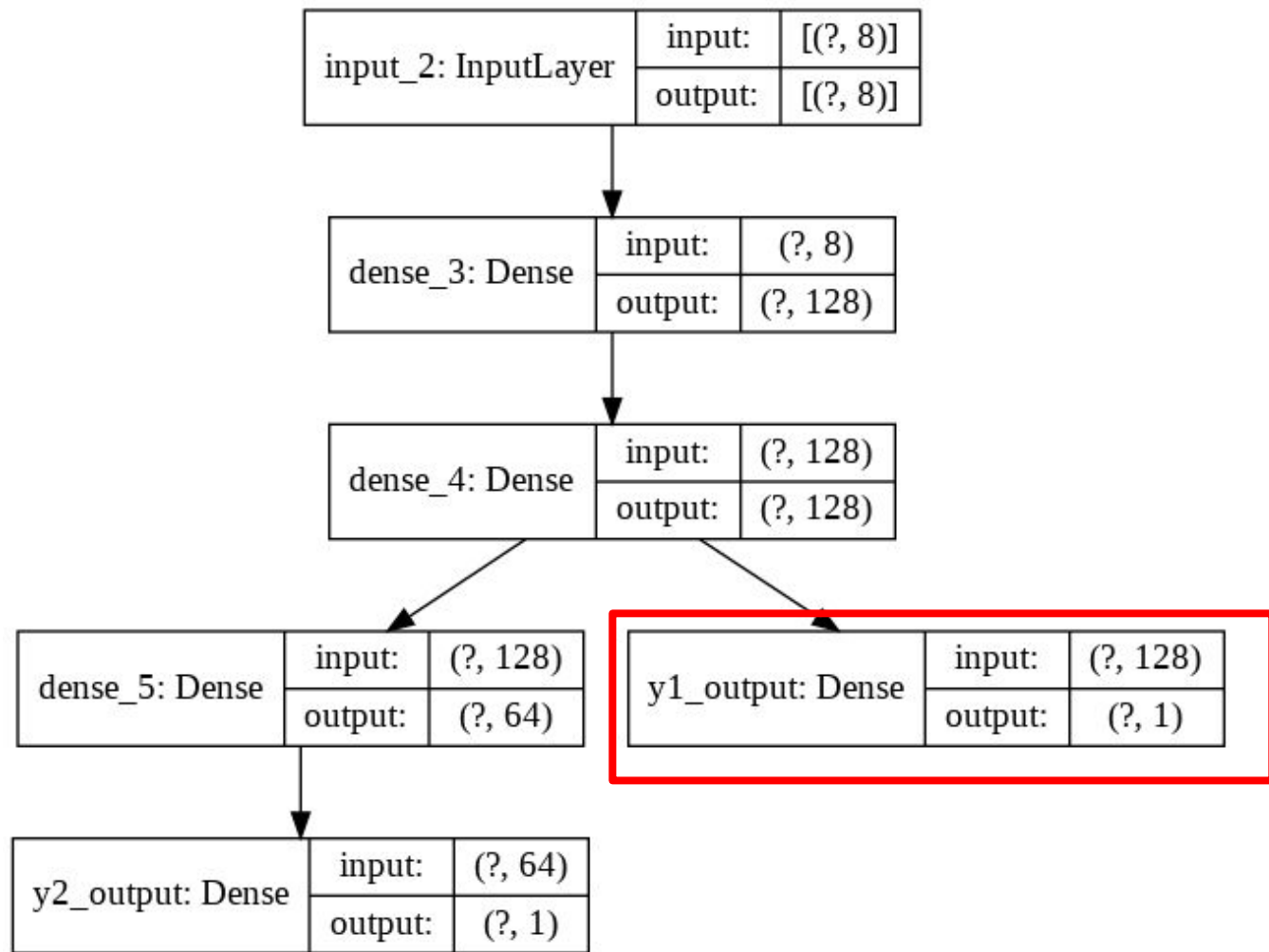
Labels

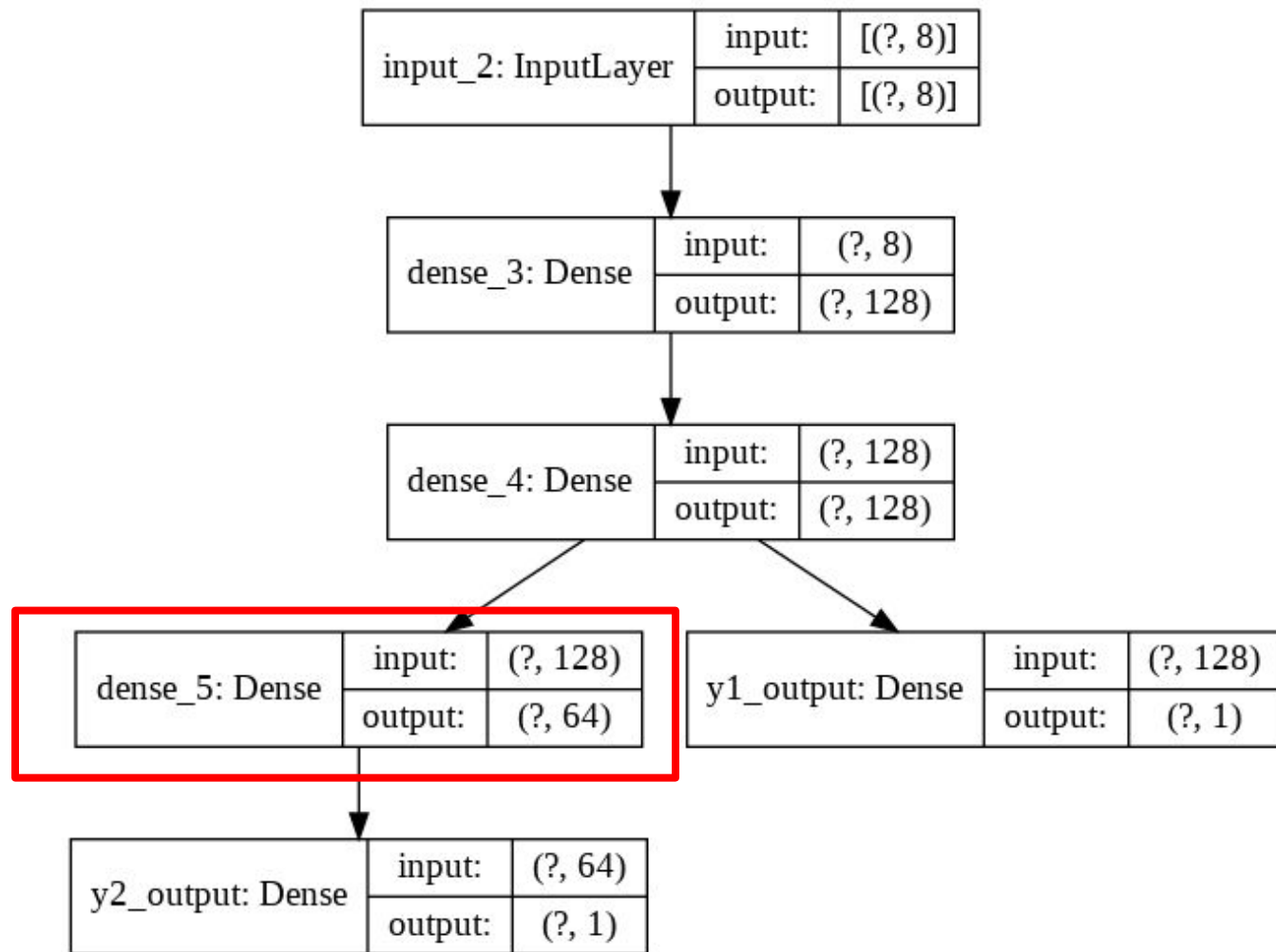
y1 Heating Load
y2 Cooling Load

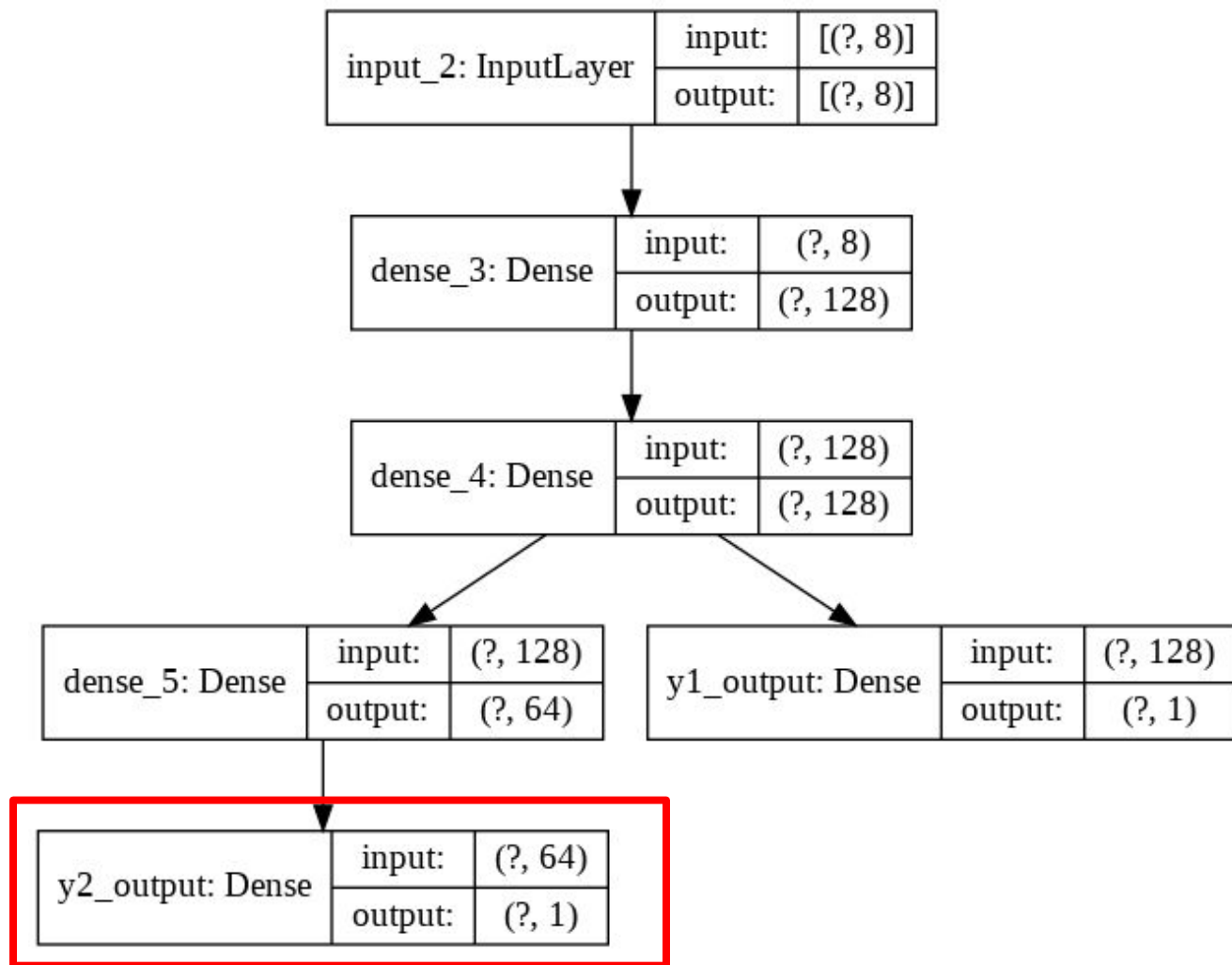






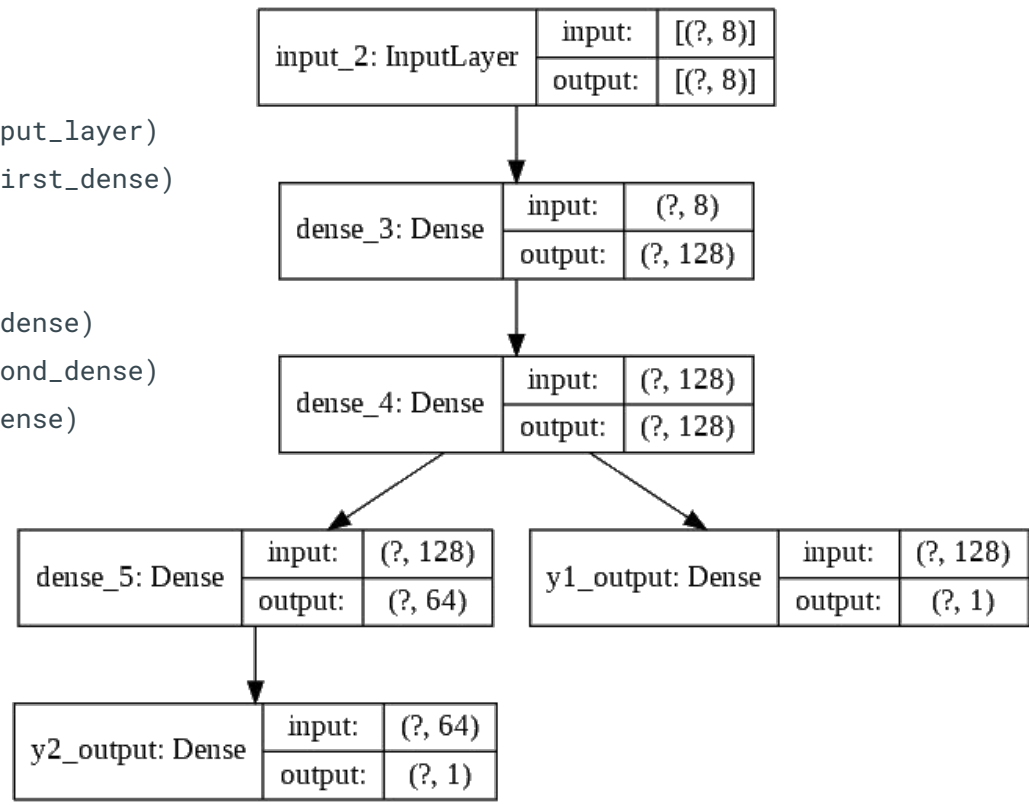







```
input_layer = Input(shape=(len(train .columns),))
first_dense = Dense(units='128', activation='relu')(input_layer)
second_dense = Dense(units='128', activation='relu')(first_dense)
```

```
y1_output = Dense(units='1', name='y1_output')(second_dense)
third_dense = Dense(units='64', activation='relu')(second_dense)
y2_output = Dense(units='1', name='y2_output')(third_dense)
```



```
# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])
```

```
input_layer = Input(shape=(len(train.columns),))
```

```
first_dense = Dense(units='128', activation='relu')(input_layer)
```

```
second_dense = Dense(units='128', activation='relu')(first_dense)
```

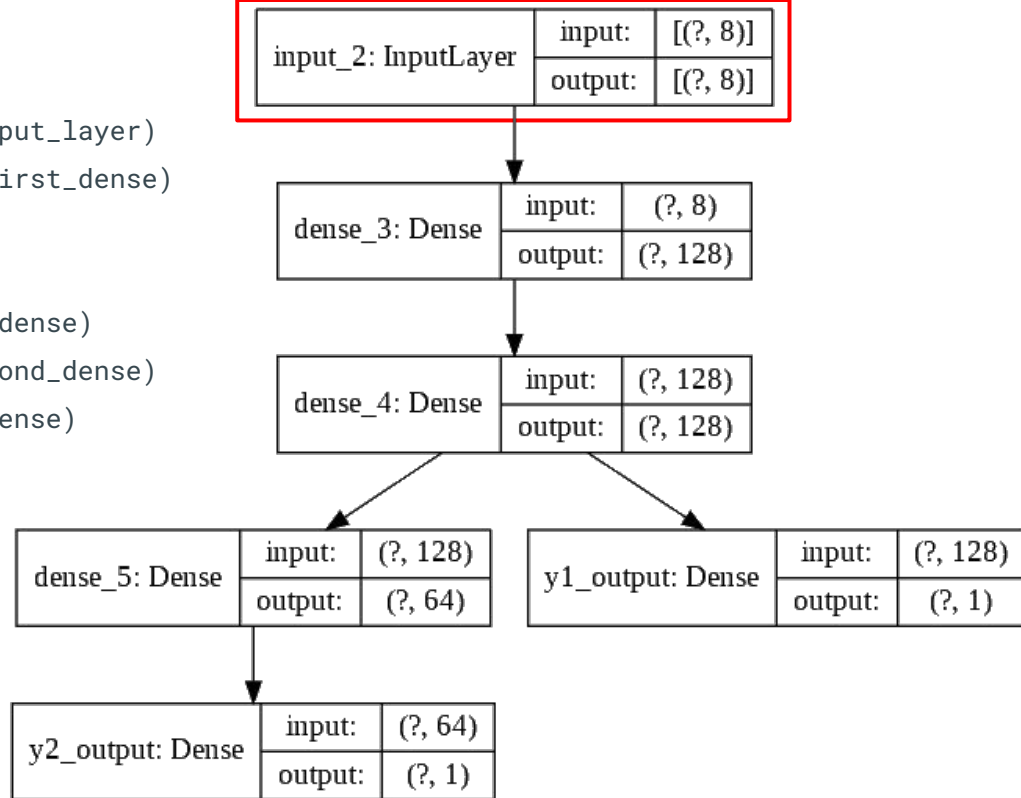
```
y1_output = Dense(units='1', name='y1_output')(second_dense)
```

```
third_dense = Dense(units='64', activation='relu')(second_dense)
```

```
y2_output = Dense(units='1', name='y2_output')(third_dense)
```

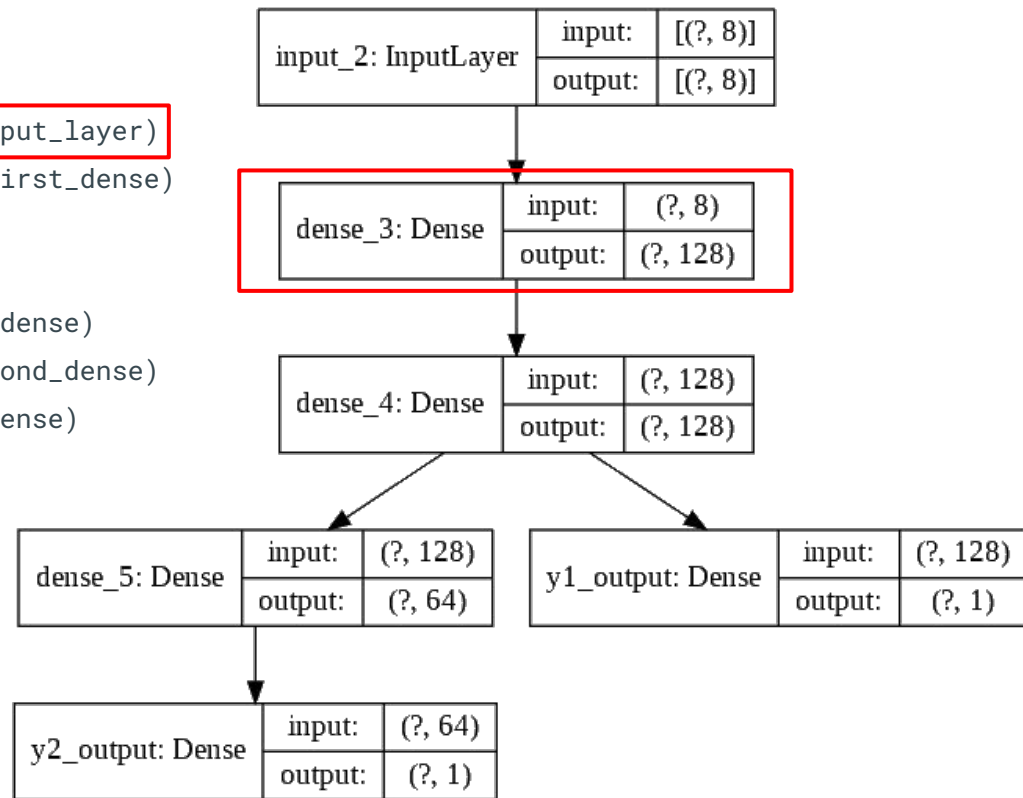
```
# Define the model with the input layer and a list of output layers
```

```
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])
```



```
input_layer = Input(shape=(len(train.columns),))
first_dense = Dense(units='128', activation='relu')(input_layer)
second_dense = Dense(units='128', activation='relu')(first_dense)
```

```
y1_output = Dense(units='1', name='y1_output')(second_dense)
third_dense = Dense(units='64', activation='relu')(second_dense)
y2_output = Dense(units='1', name='y2_output')(third_dense)
```



```
# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])
```

```

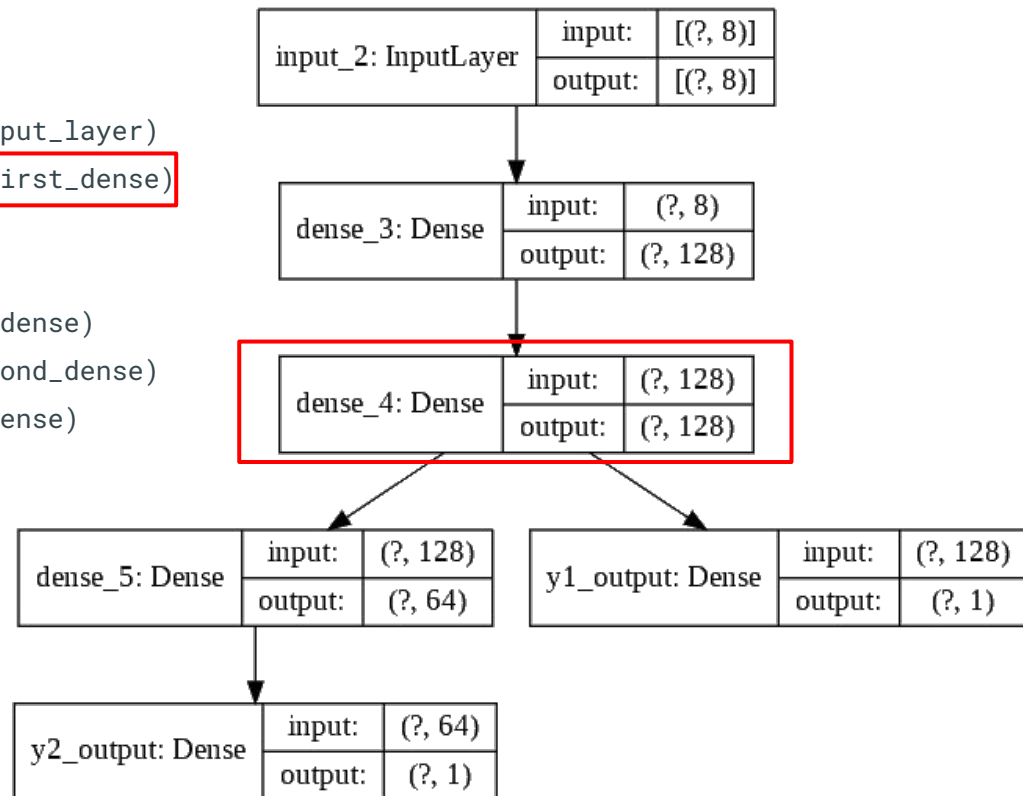
input_layer = Input(shape=(len(train.columns),))
first_dense = Dense(units='128', activation='relu')(input_layer)
second_dense = Dense(units='128', activation='relu')(first_dense)

```

```

y1_output = Dense(units='1', name='y1_output')(second_dense)
third_dense = Dense(units='64', activation='relu')(second_dense)
y2_output = Dense(units='1', name='y2_output')(third_dense)

```



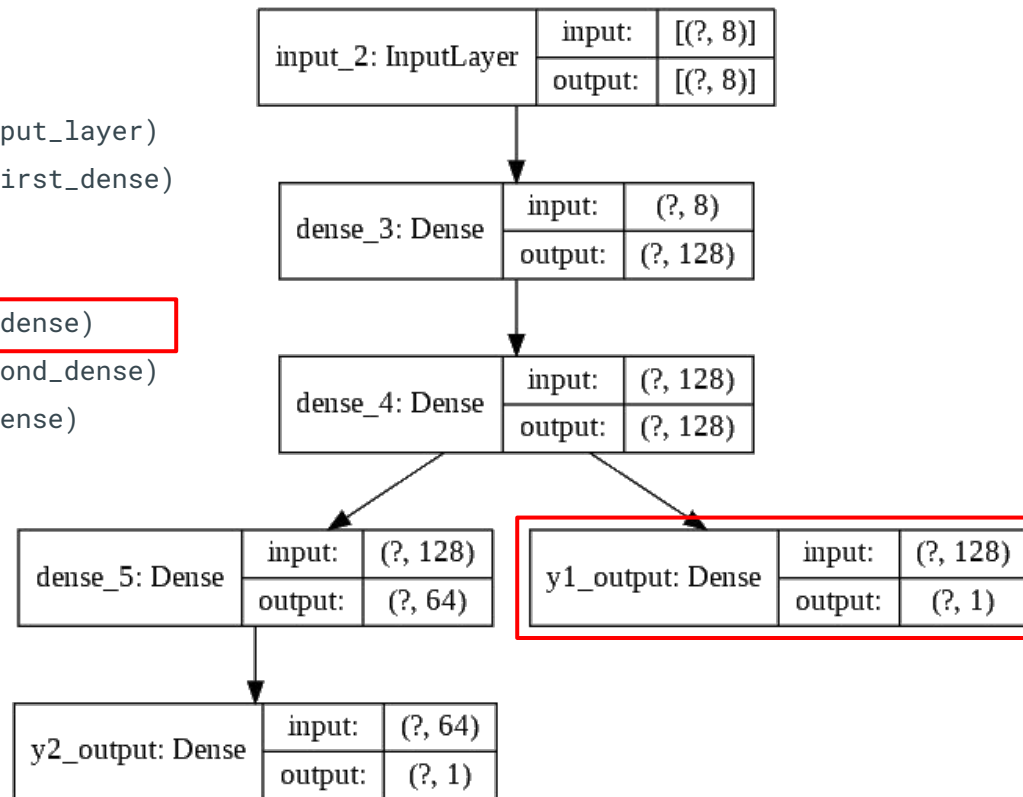
```

# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])

```

```
input_layer = Input(shape=(len(train.columns),))
first_dense = Dense(units='128', activation='relu')(input_layer)
second_dense = Dense(units='128', activation='relu')(first_dense)
```

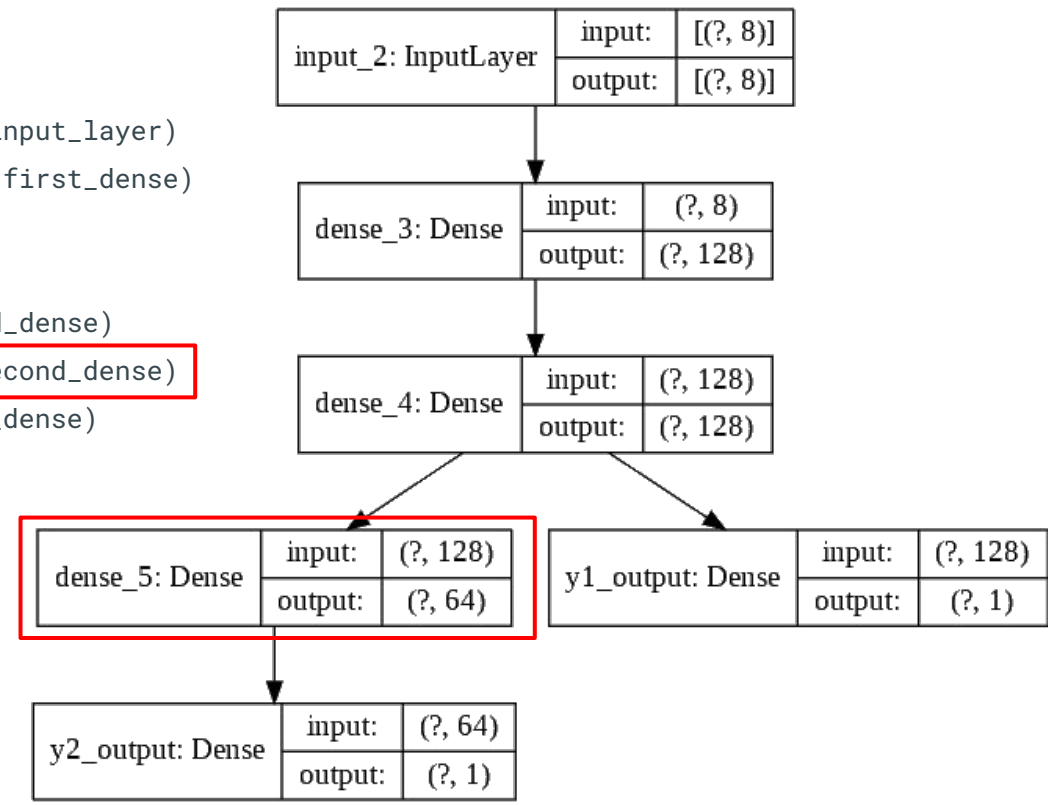
```
y1_output = Dense(units='1', name='y1_output')(second_dense)
third_dense = Dense(units='64', activation='relu')(second_dense)
y2_output = Dense(units='1', name='y2_output')(third_dense)
```



```
# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])
```

```
input_layer = Input(shape=(len(train .columns),))
first_dense = Dense(units='128', activation='relu')(input_layer)
second_dense = Dense(units='128', activation='relu')(first_dense)
```

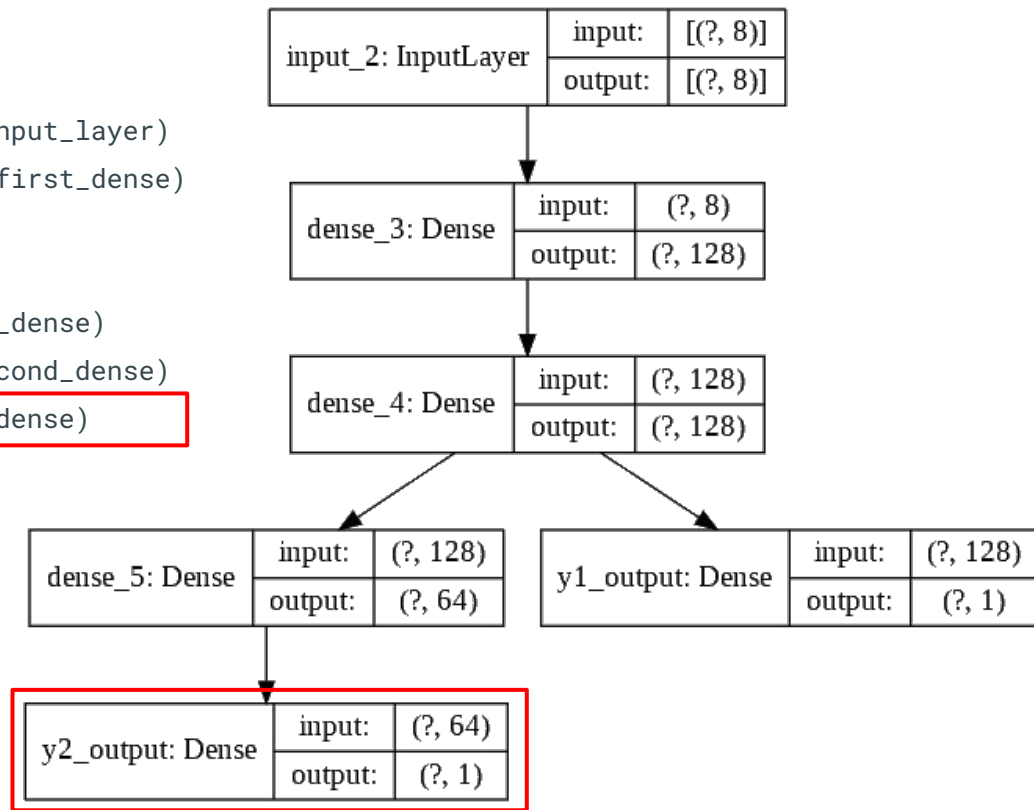
```
y1_output = Dense(units='1', name='y1_output')(second_dense)
third_dense = Dense(units='64', activation='relu')(second_dense)
y2_output = Dense(units='1', name='y2_output')(third_dense)
```



```
# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])
```

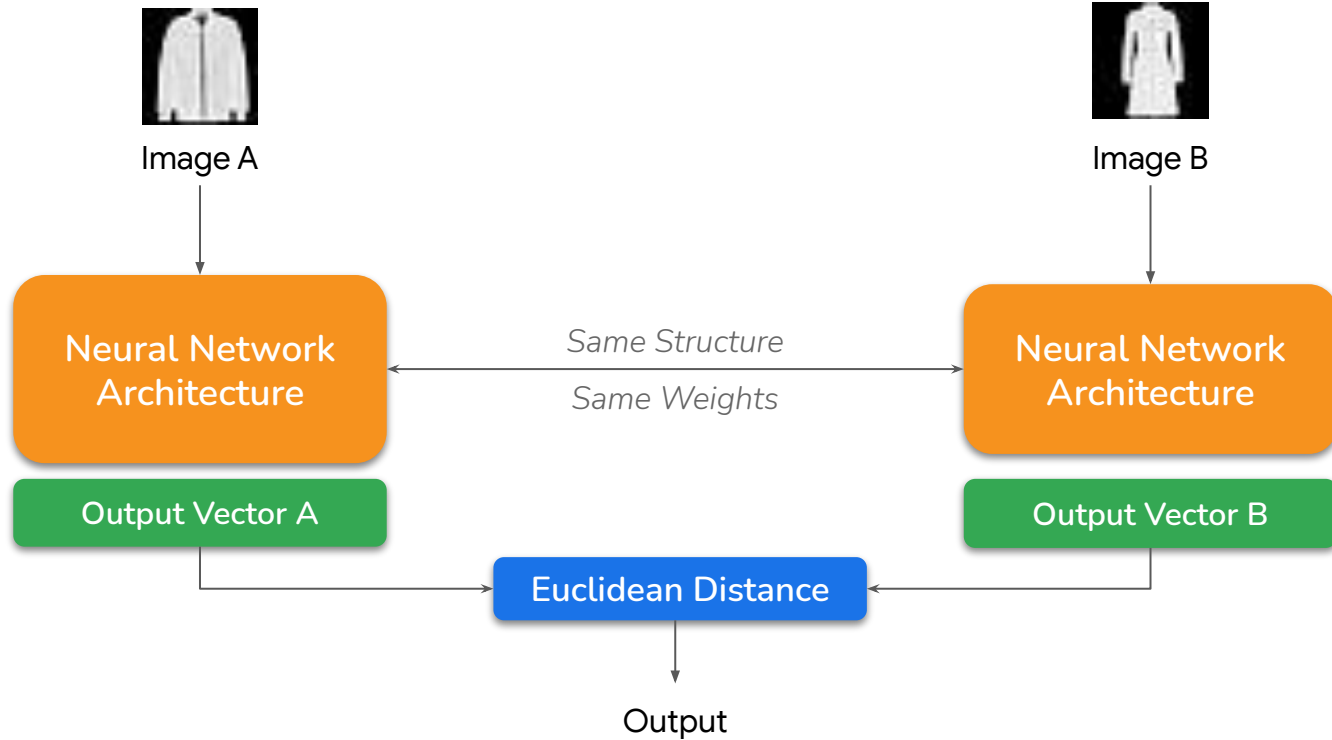
```
input_layer = Input(shape=(len(train.columns),))
first_dense = Dense(units='128', activation='relu')(input_layer)
second_dense = Dense(units='128', activation='relu')(first_dense)
```

```
y1_output = Dense(units='1', name='y1_output')(second_dense)
third_dense = Dense(units='64', activation='relu')(second_dense)
y2_output = Dense(units='1', name='y2_output')(third_dense)
```

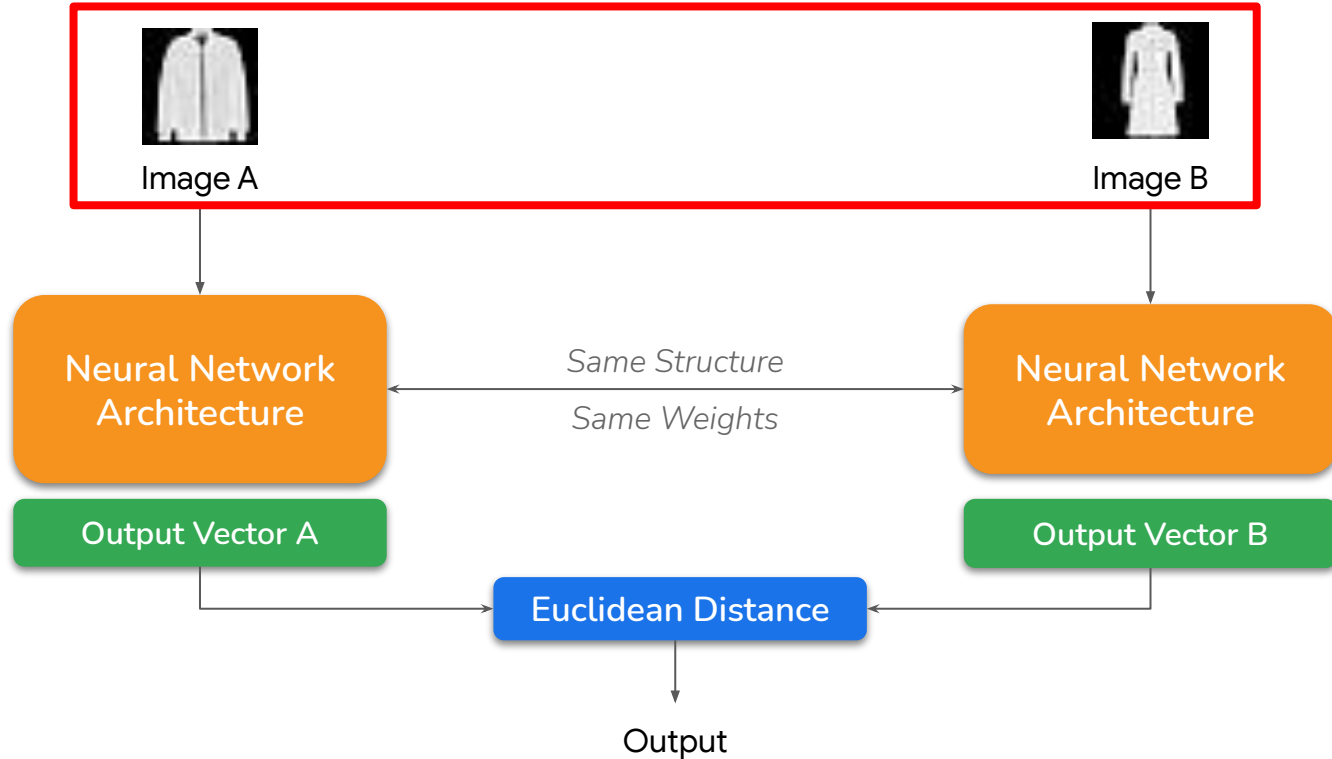


```
# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])
```

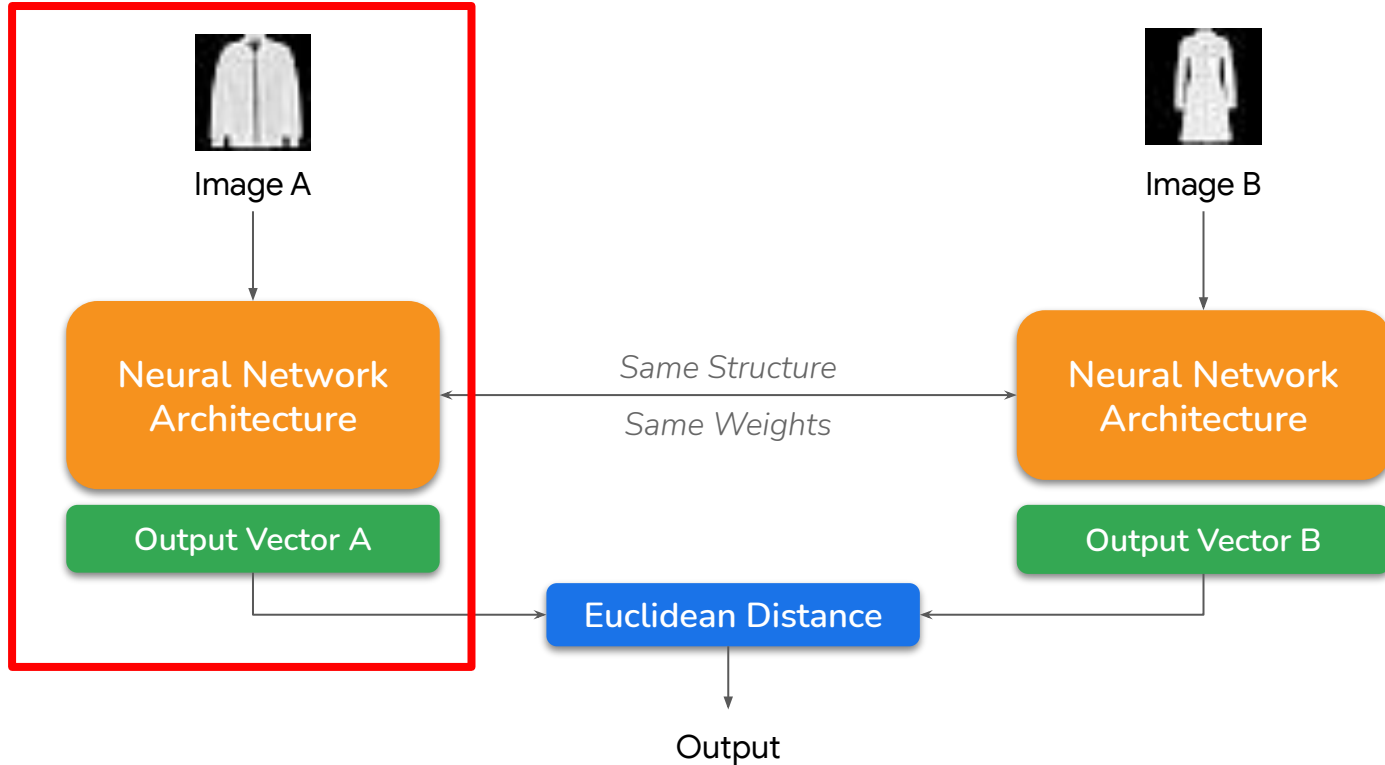
A Siamese network's architecture



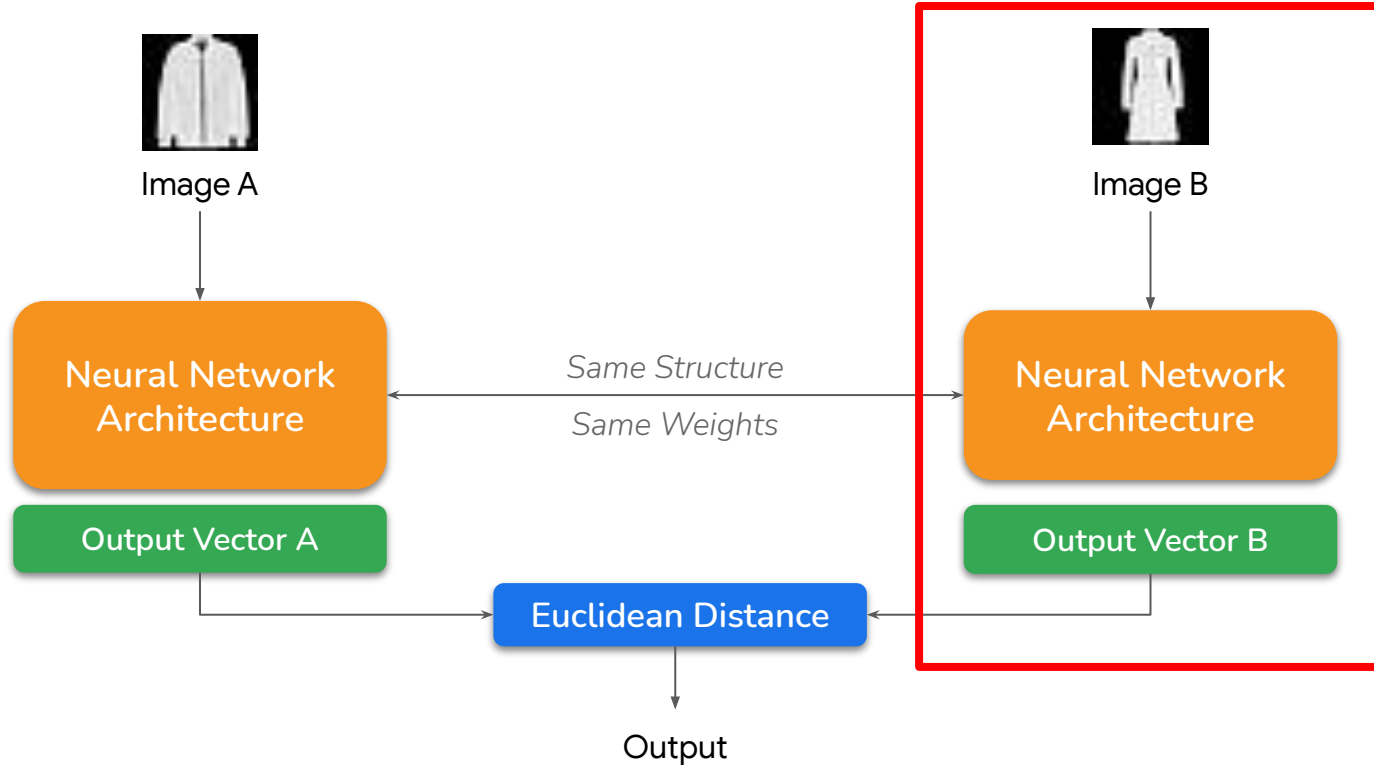
A Siamese network's architecture



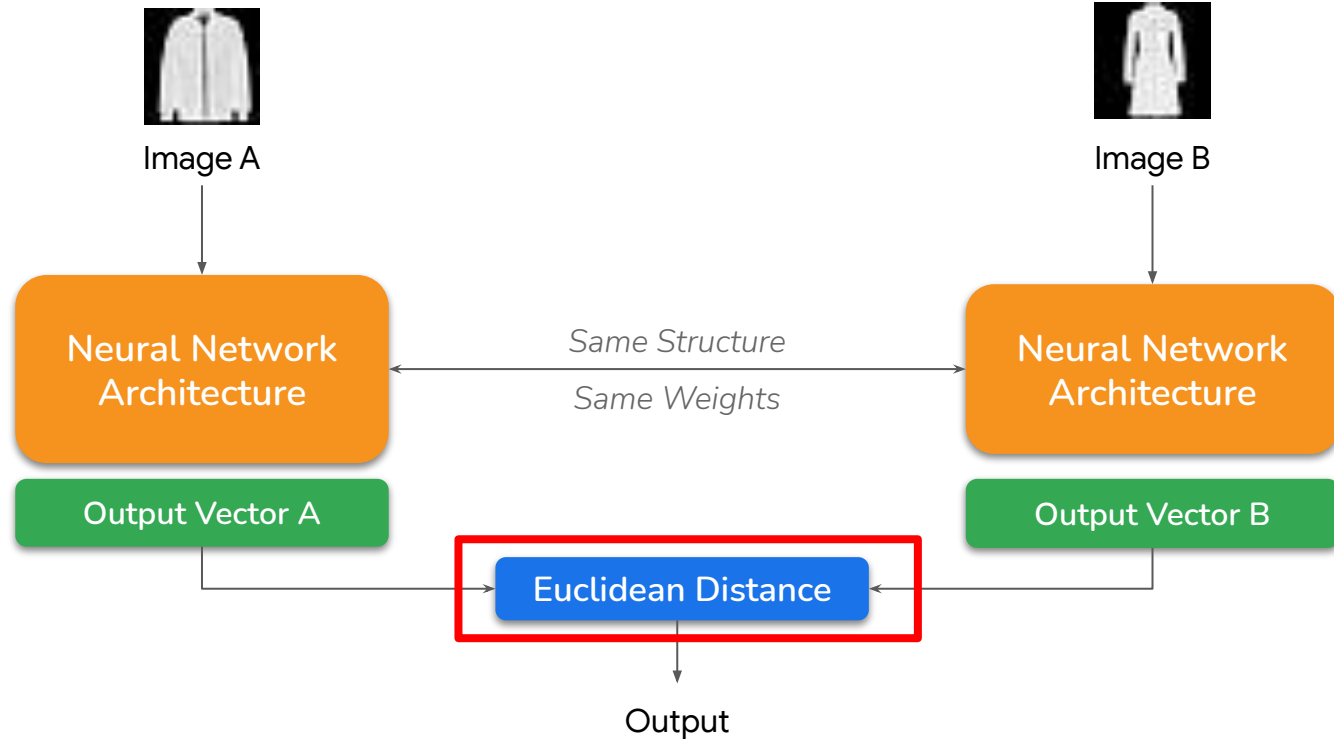
A Siamese network's architecture



A Siamese network's architecture

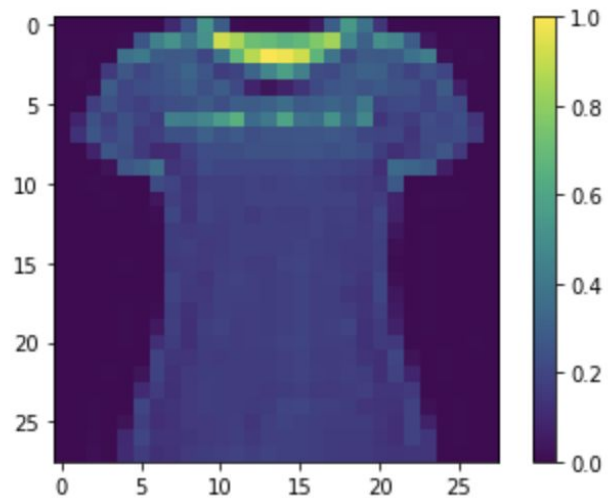
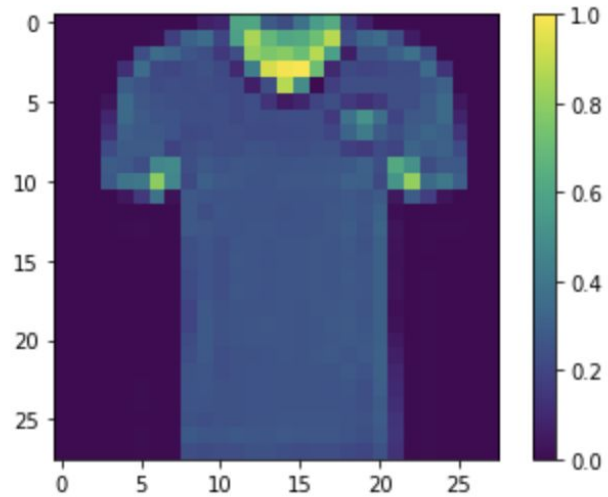


A Siamese network's architecture

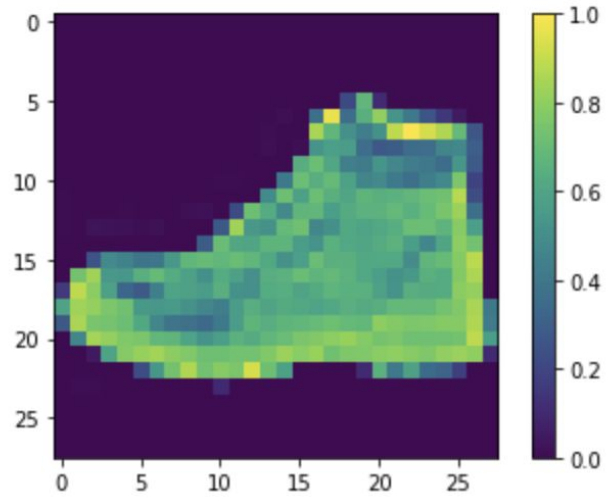
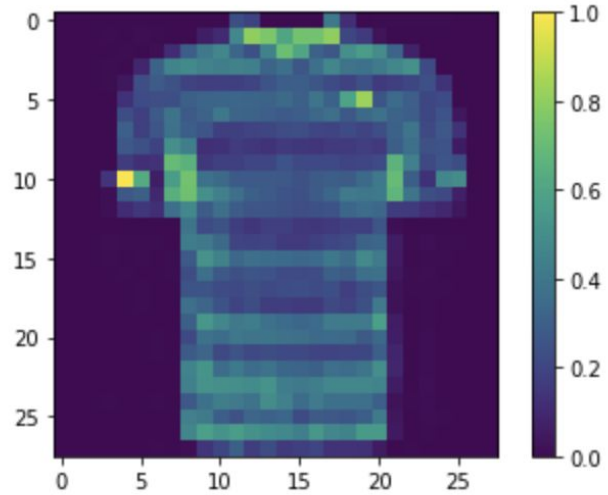


References

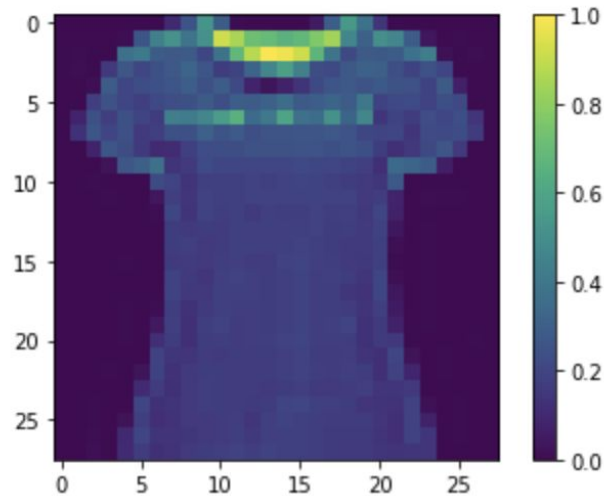
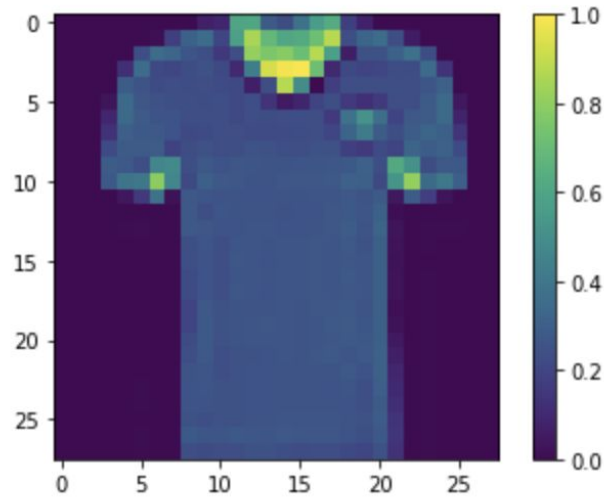
- <http://yann.lecun.com/exdb/publis/pdf/chopra-05.pdf>
- http://slazebni.cs.illinois.edu/spring17/lec09_similarity.pdf



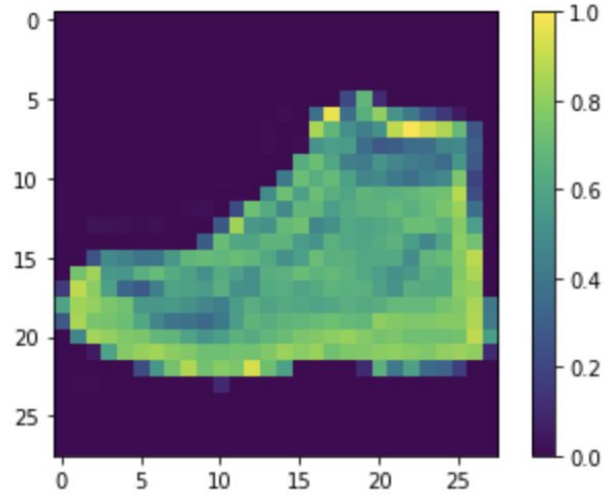
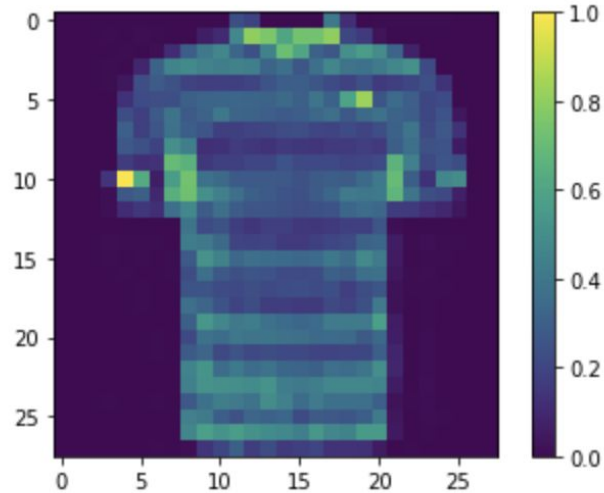
1



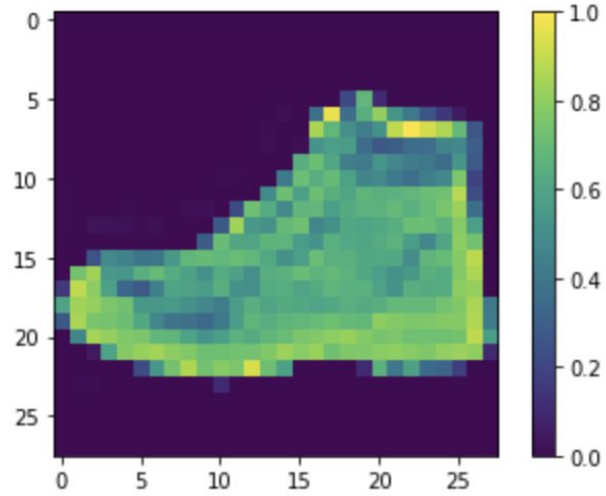
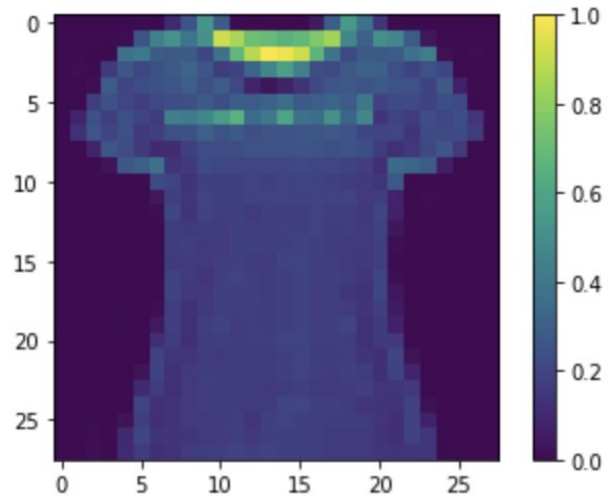
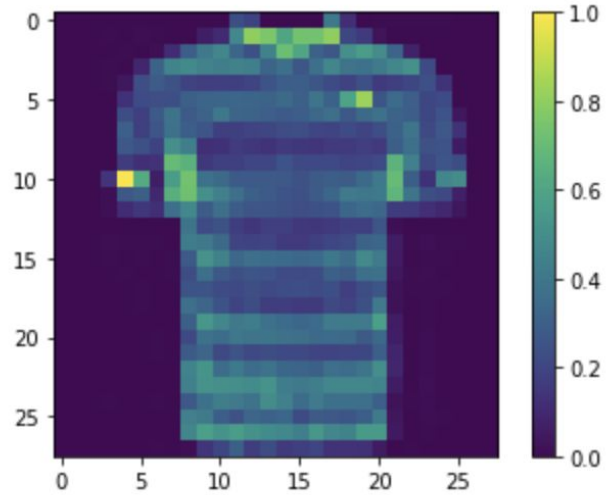
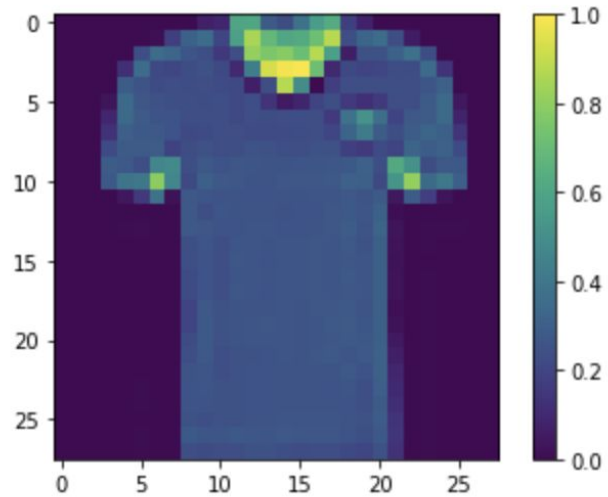
0



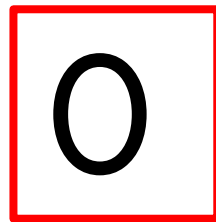
1



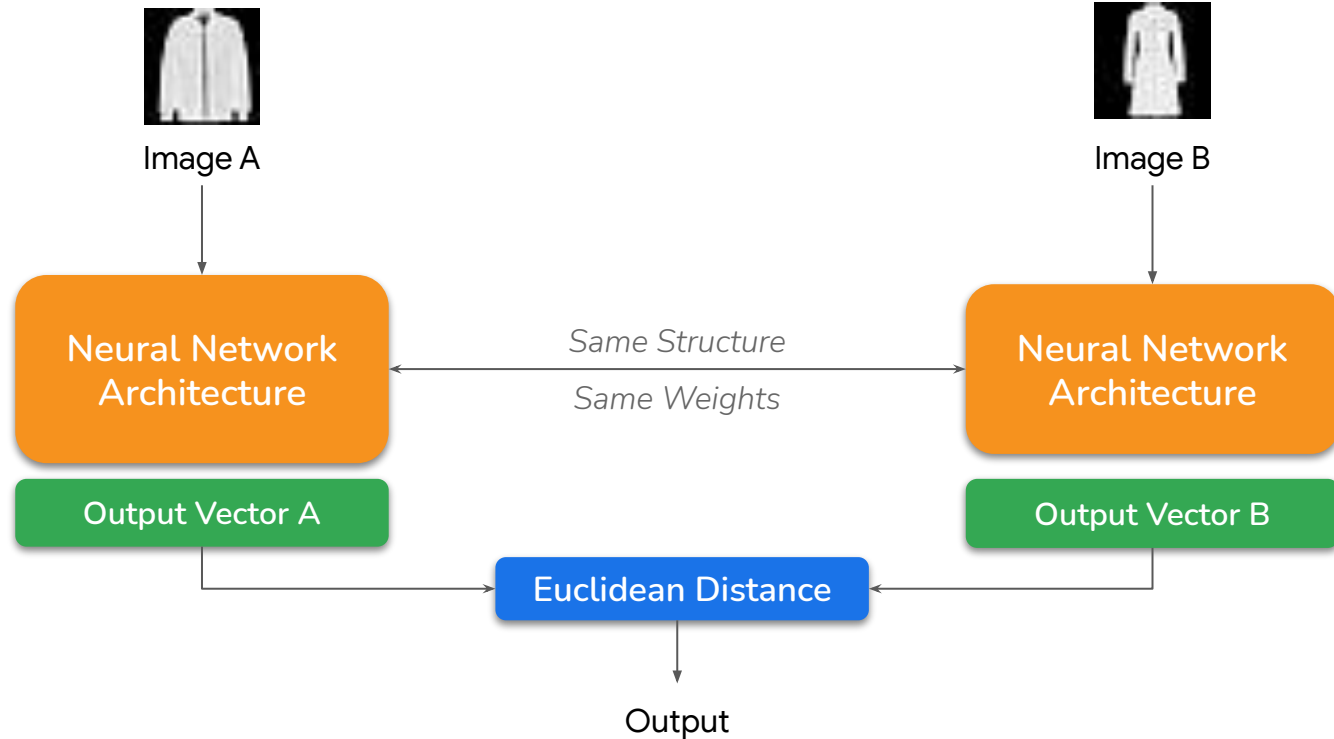
0



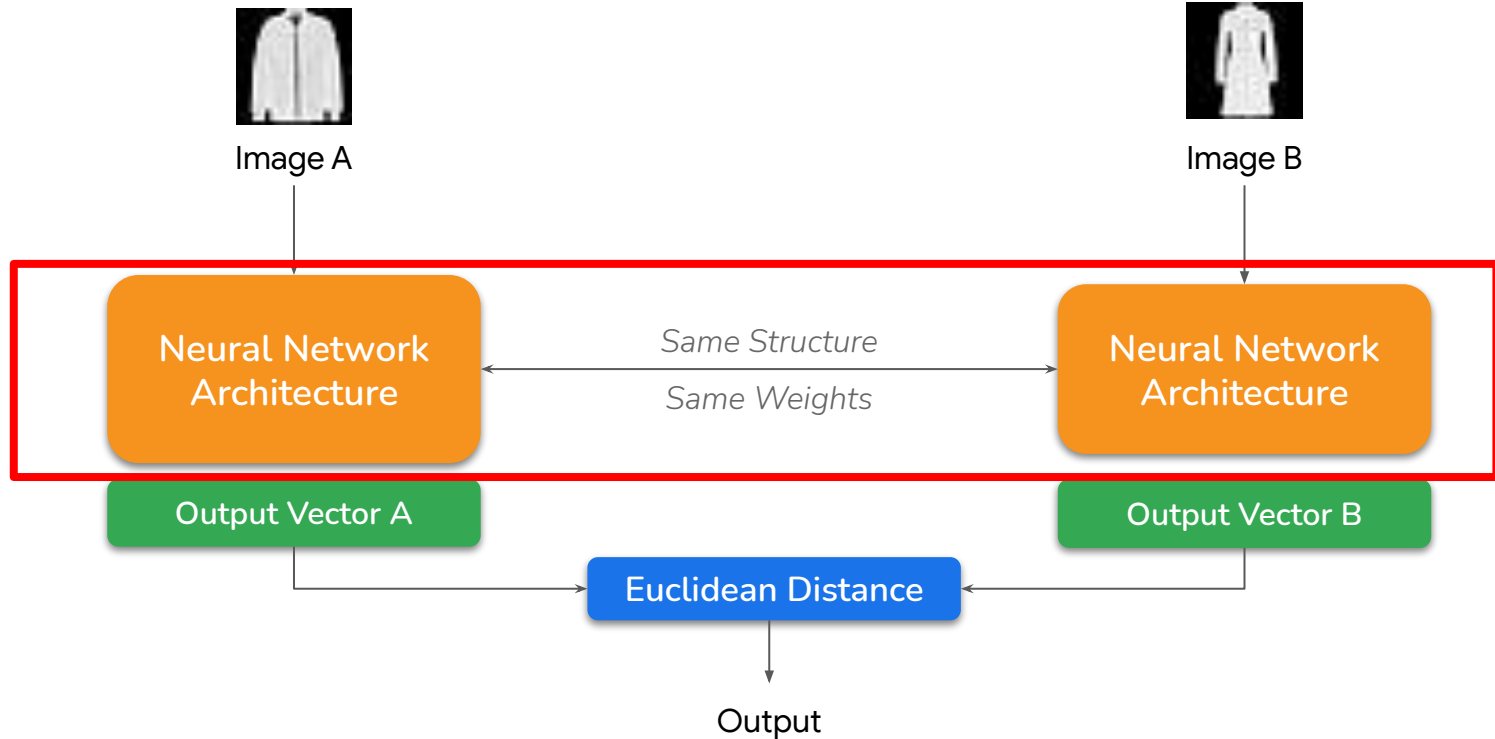
1



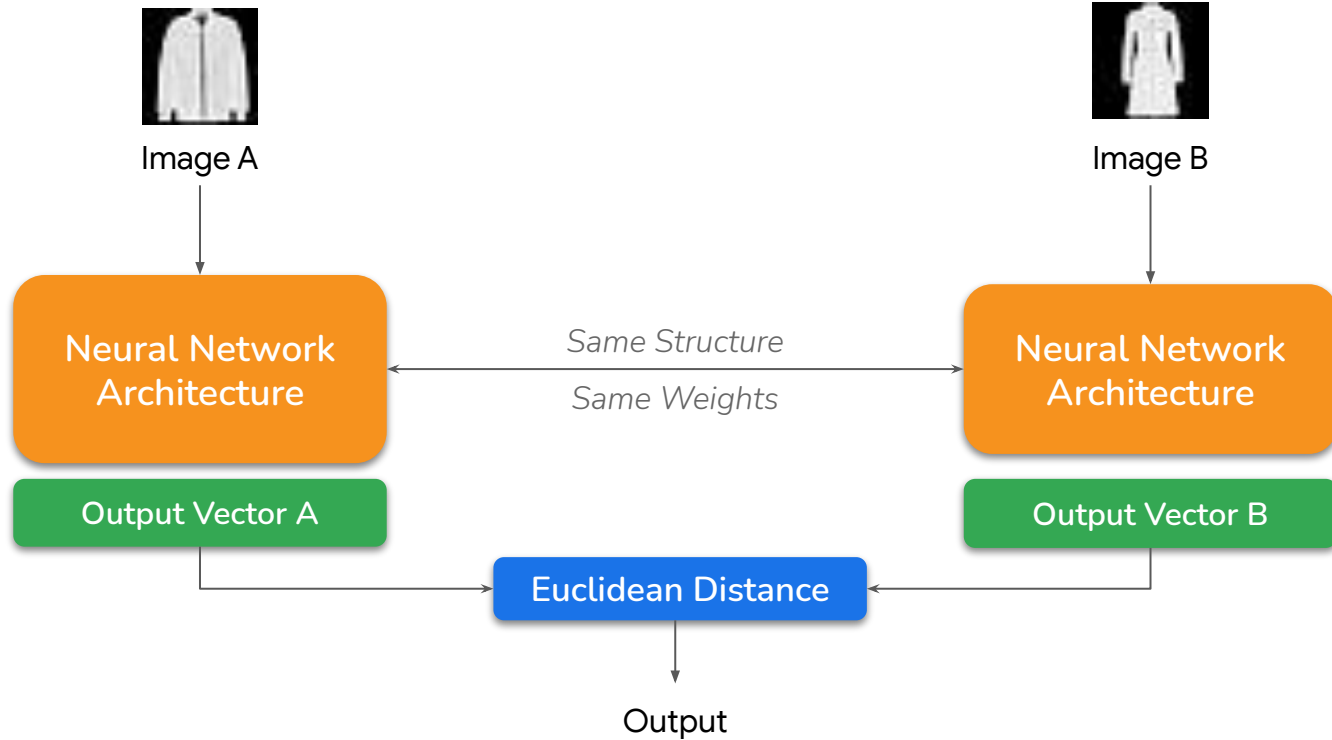
A Siamese network's architecture



A Siamese network's architecture



A Siamese network's architecture

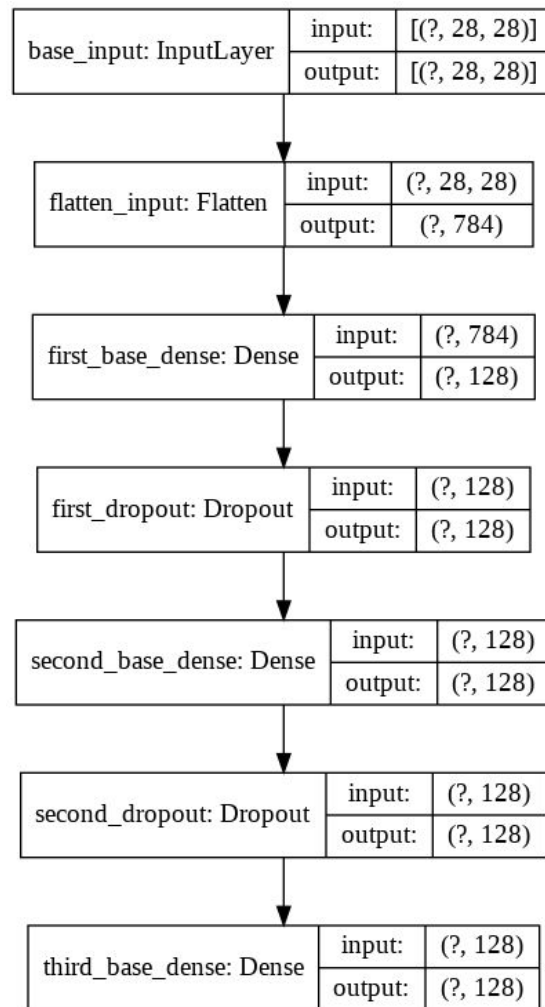


Defining the Base Network

```
def initialize_base_network():  
    input = Input(shape=(28,28,))  
    x = Flatten()(input)  
    x = Dense(128, activation='relu')(x)  
    x = Dropout(0.1)(x)  
    x = Dense(128, activation='relu')(x)  
    x = Dropout(0.1)(x)  
    x = Dense(128, activation='relu')(x)  
    return Model(inputs=input, outputs=x)
```

Defining the Base Network

```
def initialize_base_network():  
    input = Input(shape=(28,28,))  
    x = Flatten()(input)  
    x = Dense(128, activation='relu')(x)  
    x = Dropout(0.1)(x)  
    x = Dense(128, activation='relu')(x)  
    x = Dropout(0.1)(x)  
    x = Dense(128, activation='relu')(x)  
    return Model(inputs=input, outputs=x)
```



Re-using the base network

```
base_network = initialize_base_network()
```

```
input_a = Input(shape=(28,28,))
```

```
input_b = Input(shape=(28,28,))
```

```
vect_output_a = base_network(input_a)
```

```
vect_output_b = base_network(input_b)
```

Re-using the base network

```
base_network = initialize_base_network()
```

```
input_a = Input(shape=(28,28,))
```

```
input_b = Input(shape=(28,28,))
```

```
vect_output_a = base_network(input_a)
```

```
vect_output_b = base_network(input_b)
```


Re-using the base network

```
base_network = initialize_base_network()
```

```
input_a = Input(shape=(28,28,))
```

```
input_b = Input(shape=(28,28,))
```

```
vect_output_a = base_network(input_a)
```

```
vect_output_b = base_network(input_b)
```

Re-using the base network

```
base_network = initialize_base_network()
```

```
input_a = Input(shape=(28,28,))
```

```
input_b = Input(shape=(28,28,))
```

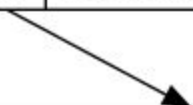
```
vect_output_a = base_network(input_a)
```

```
vect_output_b = base_network(input_b)
```

left_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

right_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

model_15: Model	input:	(?, 28, 28)
	output:	(?, 128)



left_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

right_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

model_15: Model	input:	(?, 28, 28)
	output:	(?, 128)

base_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

flatten_input: Flatten	input:	(?, 28, 28)
	output:	(?, 784)

first_base_dense: Dense	input:	(?, 784)
	output:	(?, 128)

first_dropout: Dropout	input:	(?, 128)
	output:	(?, 128)

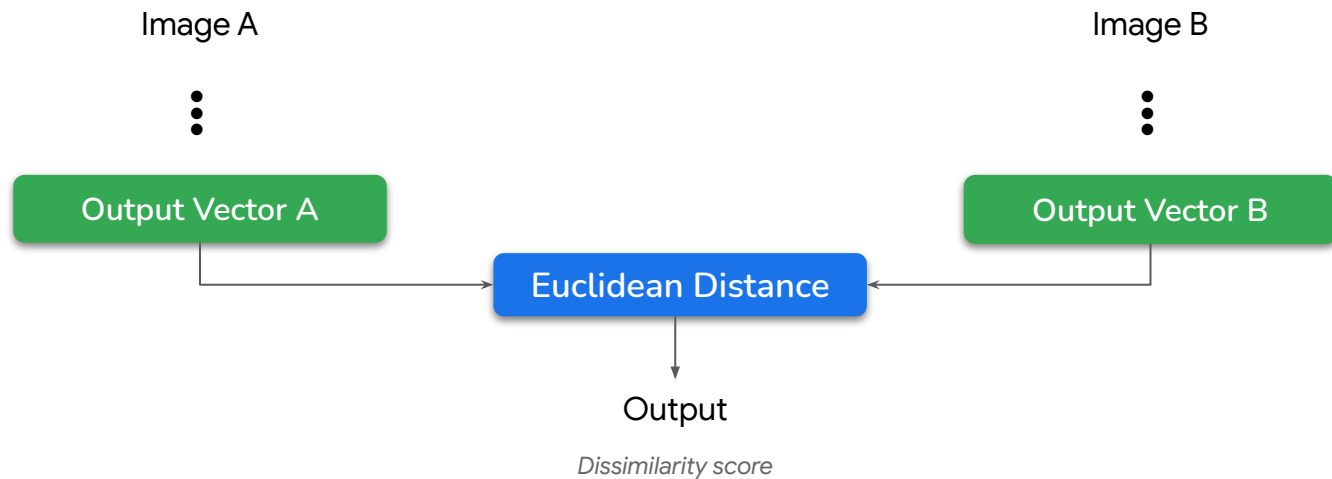
second_base_dense: Dense	input:	(?, 128)
	output:	(?, 128)

second_dropout: Dropout	input:	(?, 128)
	output:	(?, 128)

third_base_dense: Dense	input:	(?, 128)
	output:	(?, 128)

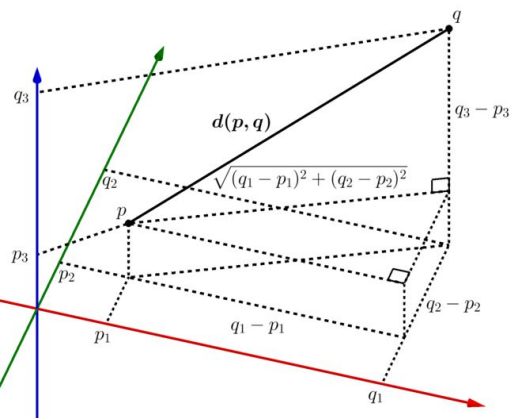
Output of the network

Similarity between two input images



```
def euclidean_distance(vects):  
    x, y = vects  
    sum_square = K.sum(K.square(x - y), axis=1, keepdims=True)  
    return K.sqrt(K.maximum(sum_square, K.epsilon()))
```

```
def eucl_dist_output_shape(shapes):  
    shape1, shape2 = shapes  
    return (shape1[0], 1)
```



Output layer is euclidean distance

```
output = Lambda(euclidean_distance,  
                output_shape=eucl_dist_output_shape)([vect_output_a, vect_output_b])
```

Output layer is euclidean distance

```
output = Lambda(euclidean_distance,  
                output_shape=eucl_dist_output_shape)([vect_output_a, vect_output_b])
```


Defining the final model

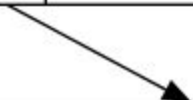
```
model = Model([input_a, input_b], output)
```

left_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

right_input: InputLayer	input:	[(?, 28, 28)]
	output:	[(?, 28, 28)]

model_15: Model	input:	(?, 28, 28)
	output:	(?, 128)

output_layer: Lambda	input:	[(?, 128), (?, 128)]
	output:	(?, 1)



Defining the final model

```
model = Model([input_a, input_b], output)
```

```
rms = RMSprop()
```

```
model.compile(loss=contrastive_loss, optimizer=rms)
```

Train the Model

```
model.fit([tr_pairs[:,0], tr_pairs[:,1]], tr_y, # Training data  
          epochs=20,  
          batch_size=128,  
          validation_data=([ts_pairs[:,0], ts_pairs[:,1]], ts_y))
```

Train the Model

```
model.fit([tr_pairs[:,0], tr_pairs[:,1]], tr_y, # Training data  
          epochs=20,  
          batch_size=128,  
          validation_data=(ts_pairs[:,0], ts_pairs[:,1], ts_y))
```

Train the Model

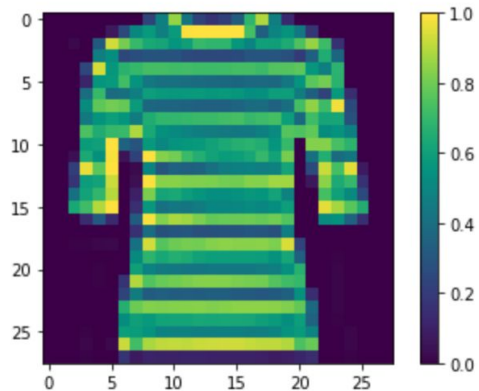
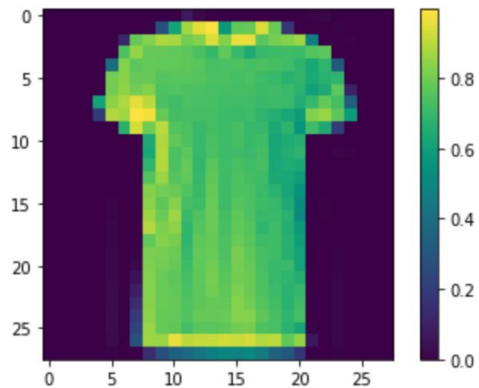
```
model.fit([tr_pairs[:,0], tr_pairs[:,1]], tr_y, # Training data  
          epochs=20,  
          batch_size=128,  
          validation_data=([ts_pairs[:,0], ts_pairs[:,1]], ts_y))
```

Train the Model

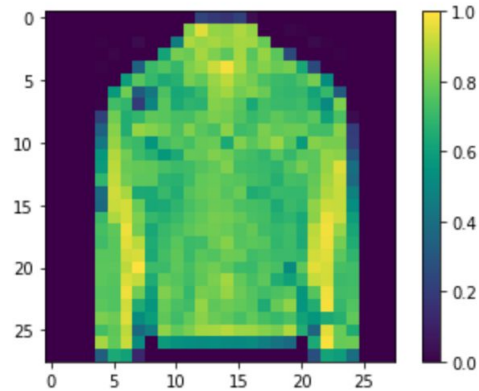
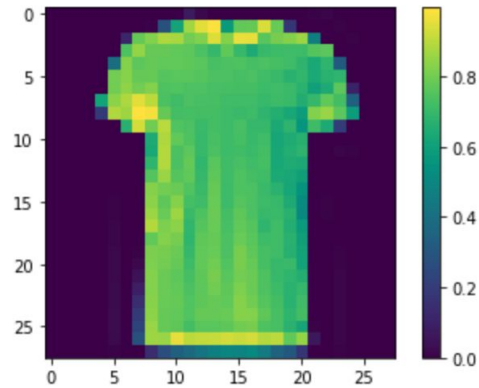
```
model.fit([tr_pairs[:,0], tr_pairs[:,1]], tr_y, # Training data  
          epochs=20,  
          batch_size=128,  
          validation_data=([ts_pairs[:,0], ts_pairs[:,1]], ts_y))
```



```
this_pair = 8  
show_image(tr_pairs[this_pair][0])  
show_image(tr_pairs[this_pair][1])  
print(tr_y[this_pair])
```



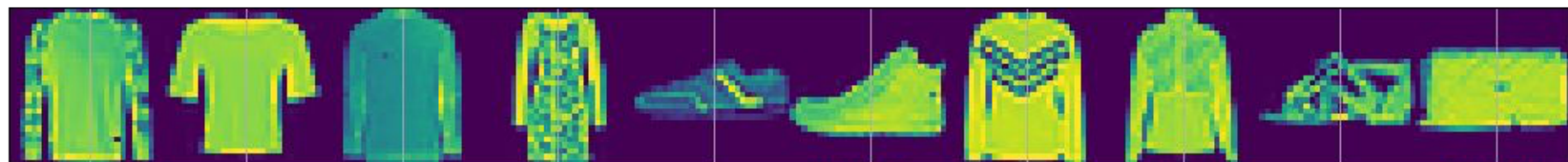
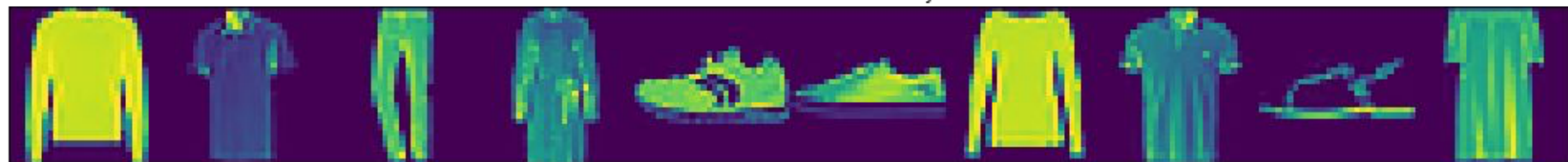
```
this_pair = 9  
show_image(tr_pairs[this_pair][0])  
show_image(tr_pairs[this_pair][1])  
print(tr_y[this_pair])
```



1.0

0.0

clothes and their dissimilarity



0.1789761

0.109004594

1.2614491

0.30694813

0.007702528

1.2948258

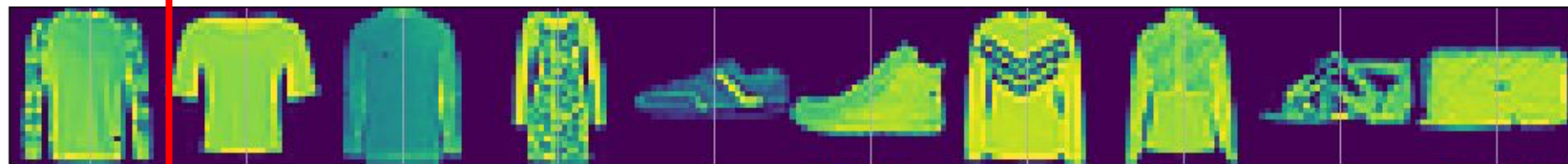
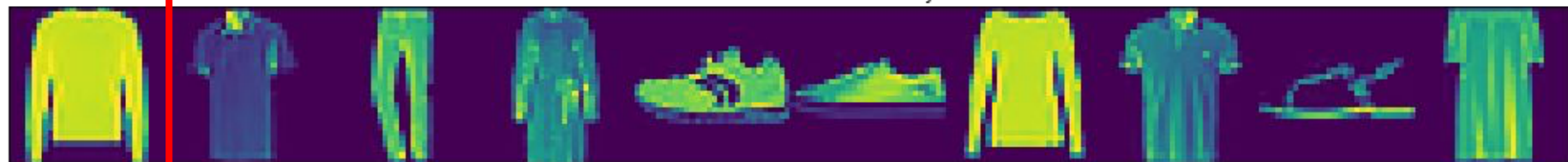
0.45779905

0.7186581

0.01438873

1.4013804

clothes and their dissimilarity



0.1789761

0.109004594

1.2614491

0.30694813

0.007702528

1.2948258

0.45779905

0.7186581

0.01438873

1.4013804

clothes and their dissimilarity

