Copyright Notice

Using Loss Functions

```
model.compile(loss='mse', optimizer='sgd')
```

or

```
from tensorflow.keras.losses import mean_squared_error
model.compile(loss=mean_squared_error, optimizer='sgd')
```

Using Loss Functions

```
model.compile(loss='mse', optimizer='sgd')
```

or

```
from tensorflow.keras.losses import mean_squared_error
model.compile(loss=mean_squared_error, optimizer='sgd')
```

Using Loss Functions

```
model.compile(loss='mse', optimizer='sgd')
```

or

```
from tensorflow.keras.losses import mean_squared_error
model.compile(loss=mean_squared_error, optimizer='sgd')
```

Using Loss Functions

```
model.compile(loss='mse', optimizer='sgd')
```

or

```
from tensorflow.keras.losses import mean_squared_error
model.compile(loss=mean_squared_error(param=value),
optimizer='sgd')
```

Creating a custom loss function

```
def my_loss_function(y_true, y_pred):
```

Creating a custom loss function

```
def my_loss_function(y_true, y_pred):
```

Creating a custom loss function

```
def my_loss_function(y_true, y_pred):
```

Creating a custom loss function

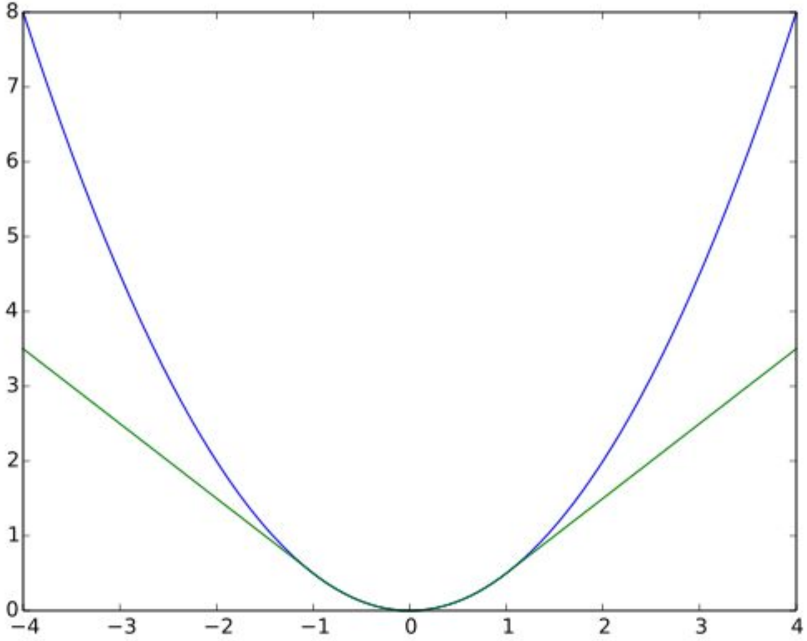
```
def my_loss_function(y_true, y_pred):
    return losses
```

Example

Huber Loss

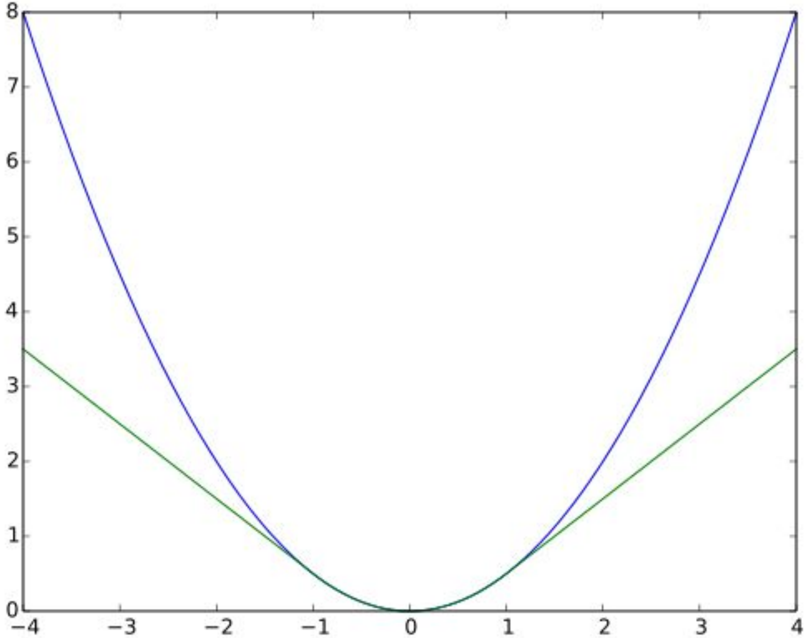
$$L_\delta(a)\begin{cases}\frac{1}{2}a^2 & for|a|\leq\delta\\ \delta\times(|a|-\frac{1}{2}\delta) & \text{otherwise}\end{cases}$$

https://en.wikipedia.org/wiki/Huber_loss



$$L_{\delta}(a)\begin{cases} \frac{1}{2}a^2 & for|a| \le \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

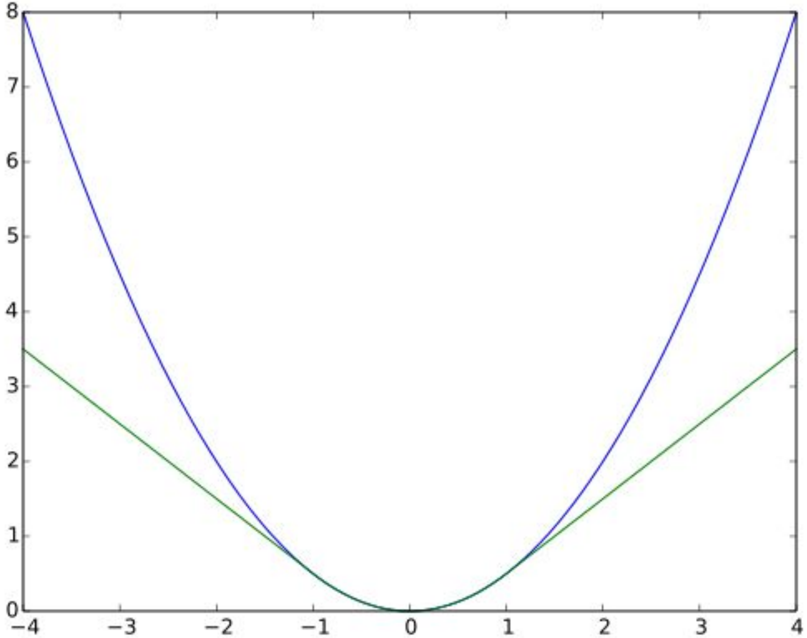
Threshold



$$L_\delta(a)\begin{cases}\frac{1}{2}a^2 & for|a| \leq \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise}\end{cases}$$

Threshold

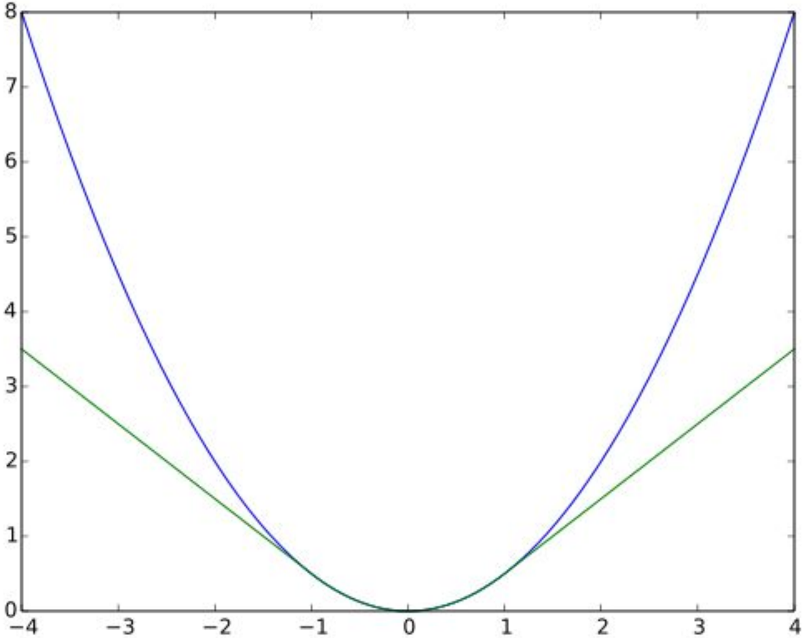
Error



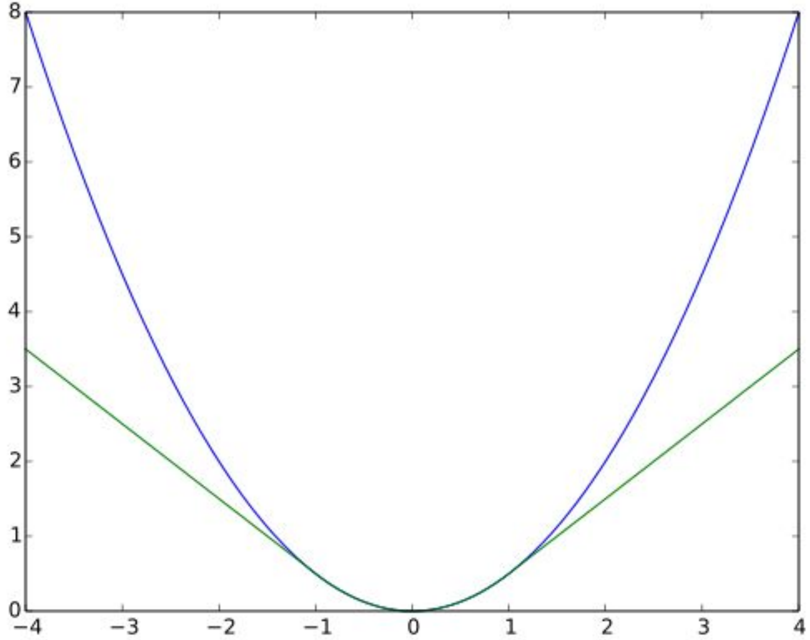
$$L_\delta(a) \begin{cases} \frac{1}{2}a^2 & for|a| \le \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

Threshold

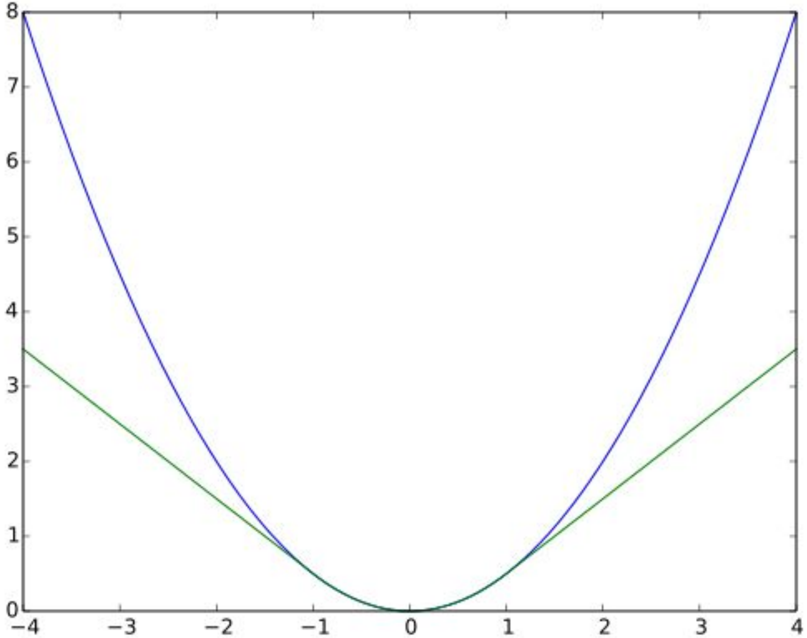
Error
(label-prediction)



$$L_\delta(a)\begin{cases}\boxed{\frac{1}{2}a^2 \qquad\qquad for|a|\leq\delta} \\ \delta\times(|a|-\frac{1}{2}\delta) & \text{otherwise}\end{cases}$$



$$L_\delta(a)\begin{cases} \frac{1}{2}a^2 & for|a| \le \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$



```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a)\begin{cases} \frac{1}{2}a^2 & for|a| \le \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a) \begin{cases} \frac{1}{2}a^2 & for|a| \leq \delta \\ \boxed{\delta} \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

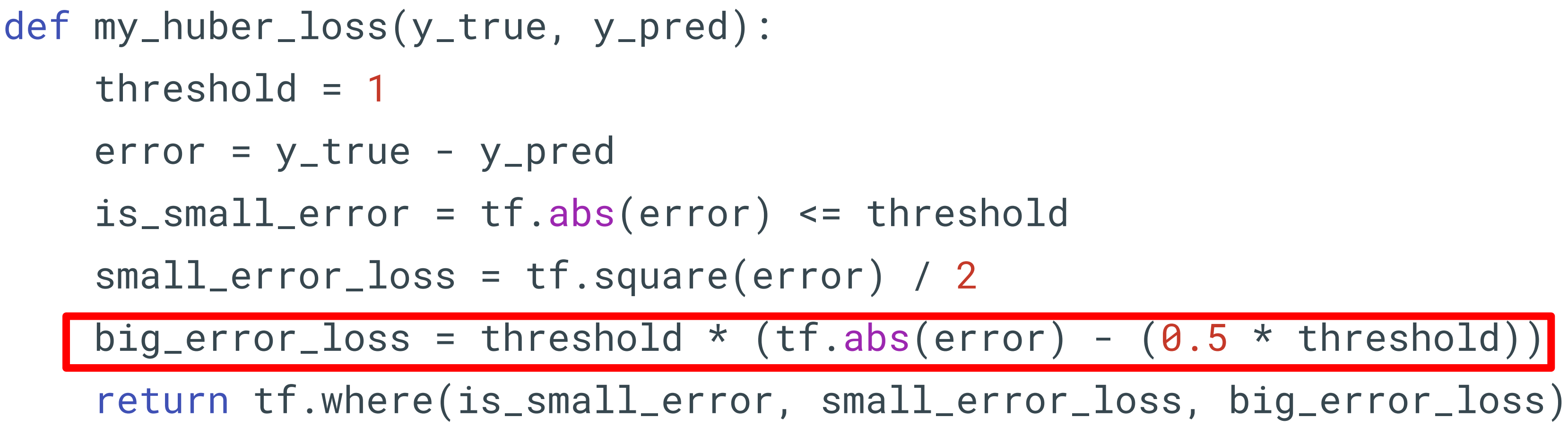
$$L_\delta(a) \begin{cases} \frac{1}{2}\boxed{a}^2 & for\, |\boxed{a}| \leq \delta \\ \delta \times (|\boxed{a}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

```
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a)\begin{cases} \frac{1}{2}a^2 & \boxed{for|a| \leq \delta} \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

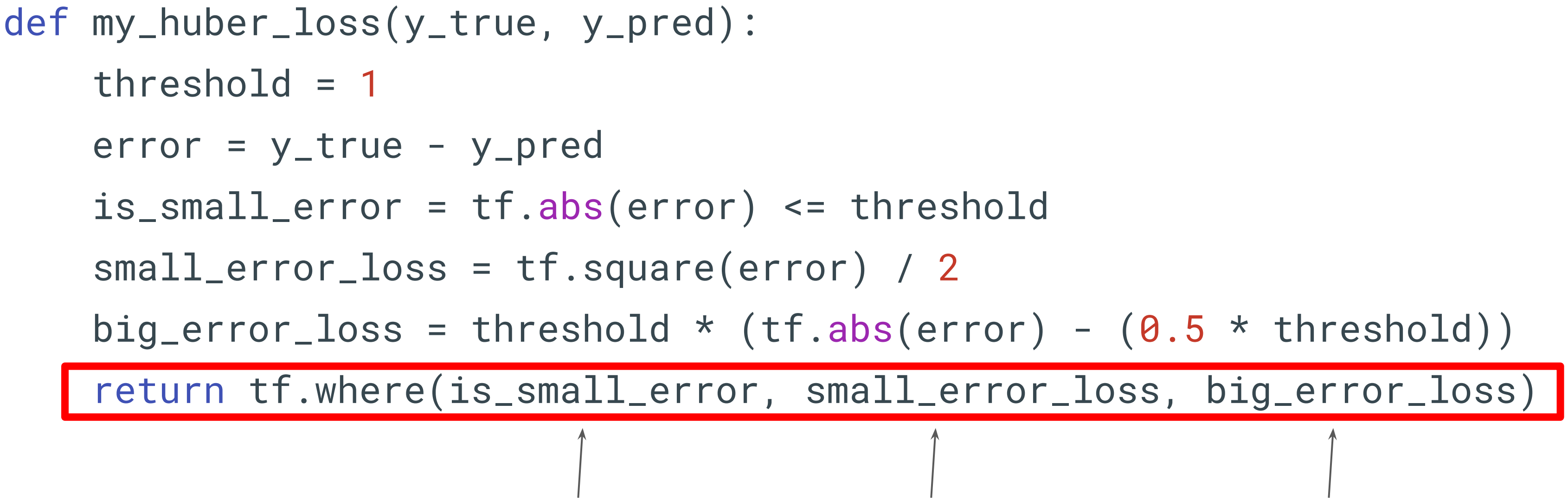
```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a)\begin{cases}\boxed{\frac{1}{2}a^2} & \xleftarrow{\quad} \boxed{for|a| \leq \delta} \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise}\end{cases}$$



```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a) \begin{cases} \frac{1}{2}a^2 & for|a| \le \delta \\ \boxed{\delta \times (|a| - \frac{1}{2}\delta)} \longleftarrow & \boxed{\text{otherwise}} \end{cases}$$



```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

Boolean to check

Value if True

Value if False

$$L_\delta(a)\begin{cases}\frac{1}{2}a^2 & for|a|\leq\delta\\ \delta\times(|a|-\frac{1}{2}\delta) & \text{otherwise}\end{cases}$$

```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])

model.compile(optimizer='sgd', loss='mean_squared_error')
```

```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
```

```python
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='my_huber_loss')




def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a)\begin{cases}\frac{1}{2}a^2 & for|a|\le\delta\\ \delta\times(|a|-\frac{1}{2}\delta) & \text{otherwise}\end{cases}$$

```
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```python
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```
def my_huber_loss(y_true, y_pred):
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```
def my_huber_loss_with_threshold(threshold):
    def my_huber_loss(y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) <= threshold
        small_error_loss = tf.square(error) / 2
        big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
        return tf.where(is_small_error, small_error_loss, big_error_loss)
    return my_huber_loss
```

```
def my_huber_loss_with_threshold(threshold):
    def my_huber_loss(y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) <= threshold
        small_error_loss = tf.square(error) / 2
        big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
        return tf.where(is_small_error, small_error_loss, big_error_loss)
    return my_huber_loss
```

```python
def my_huber_loss_with_threshold(threshold):
    def my_huber_loss(y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) <= threshold
        small_error_loss = tf.square(error) / 2
        big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
        return tf.where(is_small_error, small_error_loss, big_error_loss)
    return my_huber_loss
```

```
def my_huber_loss_with_threshold(threshold):
    def my_huber_loss(y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) <= threshold
        small_error_loss = tf.square(error) / 2
        big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
        return tf.where(is_small_error, small_error_loss, big_error_loss)
    return my_huber_loss
```

```
model.compile(
    optimizer='sgd', loss=my_huber_loss_with_threshold(threshold=1))
```

```
model.compile(
    optimizer='sgd', loss=my_huber_loss_with_threshold(threshold=1))
```

```python
from tensorflow.keras.losses import Loss

class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```python
from tensorflow.keras.losses import Loss

class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```python
from tensorflow.keras.losses import Loss

class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```
from tensorflow.keras.losses import Loss


class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```
from tensorflow.keras.losses import Loss


class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold


  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```python
from tensorflow.keras.losses import Loss

class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```python
from tensorflow.keras.losses import Loss

class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```
from tensorflow.keras.losses import Loss


class MyHuberLoss(Loss):
  threshold = 1
  def __init__(self, threshold):
    super().__init__()
    self.threshold = threshold

  def call(self, y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= self.threshold
    small_error_loss = tf.square(error) / 2
    big_error_loss = self.threshold * (tf.abs(error) - (0.5 * self.threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

```
model.compile(optimizer='sgd', loss=MyHuberLoss(threshold=1))
model.compile(optimizer='sgd', loss=MyHuberLoss(threshold=1))
```

Siamese Network for Image Similarity

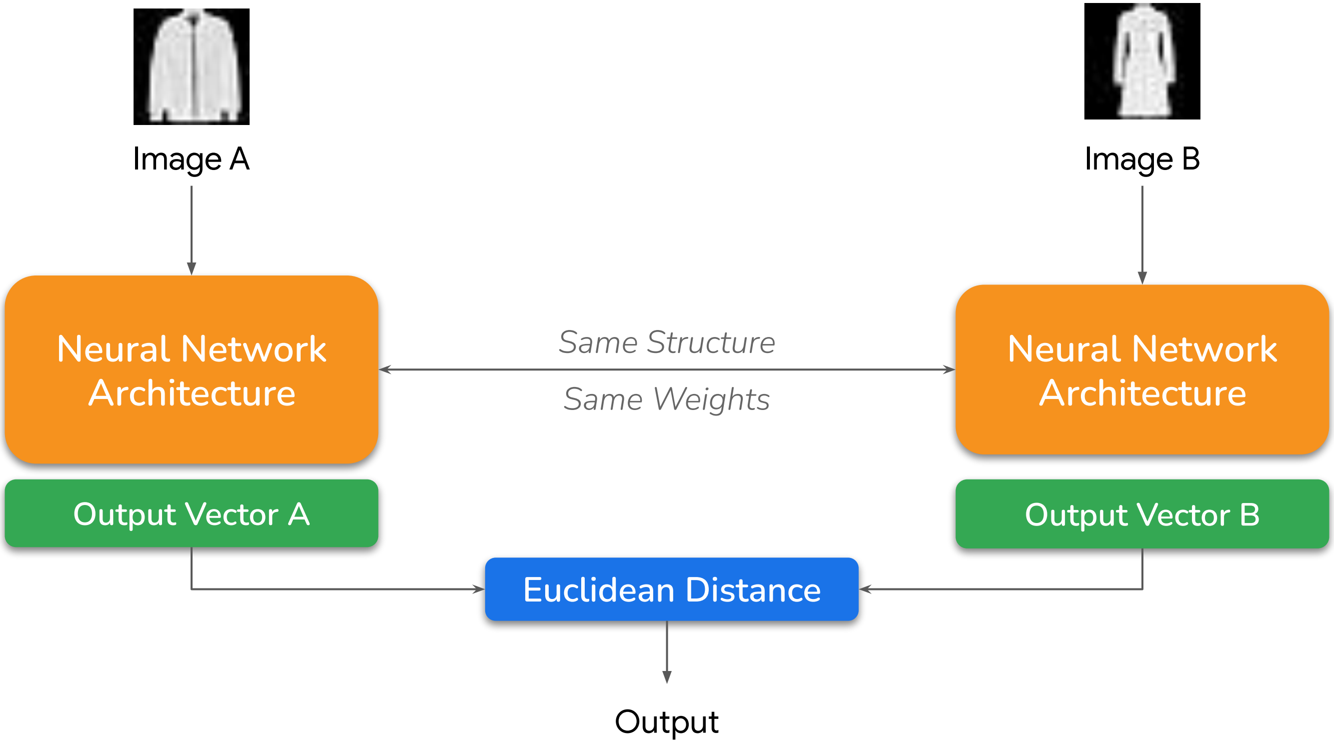


Image A

Image B

Neural Network Architecture

Same Structure

Same Weights

Neural Network Architecture

Output Vector A

Output Vector B

Euclidean Distance

Output

Contrastive Loss

- If images are similar, produce feature vectors that are very similar
- If images are different, produce feature vectors that are dissimilar.
- Based on the paper

  “Dimensionality Reduction by Learning an Invariant Mapping”

  by R. Hadsell ; S. Chopra ; Y. LeCun

  http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

Contrastive Loss - Formula

$$\boxed{Y} * D^2 + (1 - \boxed{Y}) * max(margin - D, 0)^2$$

Contrastive Loss - Formula

$$Y * \boxed{D}^2 + (1 - Y) * max(margin - \boxed{D}, 0)^2$$

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(\boxed{margin} - D, 0)^2$$

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

$$\Downarrow$$

$$1 * D^2 + (1 - 1) * max(margin - D, 0)^2$$

$$\Downarrow$$

$$D^2$$

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

$$\Downarrow$$

$$0 * D^2 + (1 - 0) * max(margin - D, 0)^2$$

$$\Downarrow$$

$$max(margin - D, 0)^2$$

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

$$Y_{true} * Y_{pred}^{2} + (1 - Y_{true}) * max(margin - Y_{pred}, 0)^2$$

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

$$Y_{true} * Y_{pred}^{\ 2} + (1 - Y_{true}) * max(margin - Y_{pred}, 0)^2$$

Contrastive Loss - Formula

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

$$\downarrow$$

$$Y_{true} * Y_{pred}{}^2 + (1 - Y_{true}) * max(margin - Y_{pred}, 0)^2$$

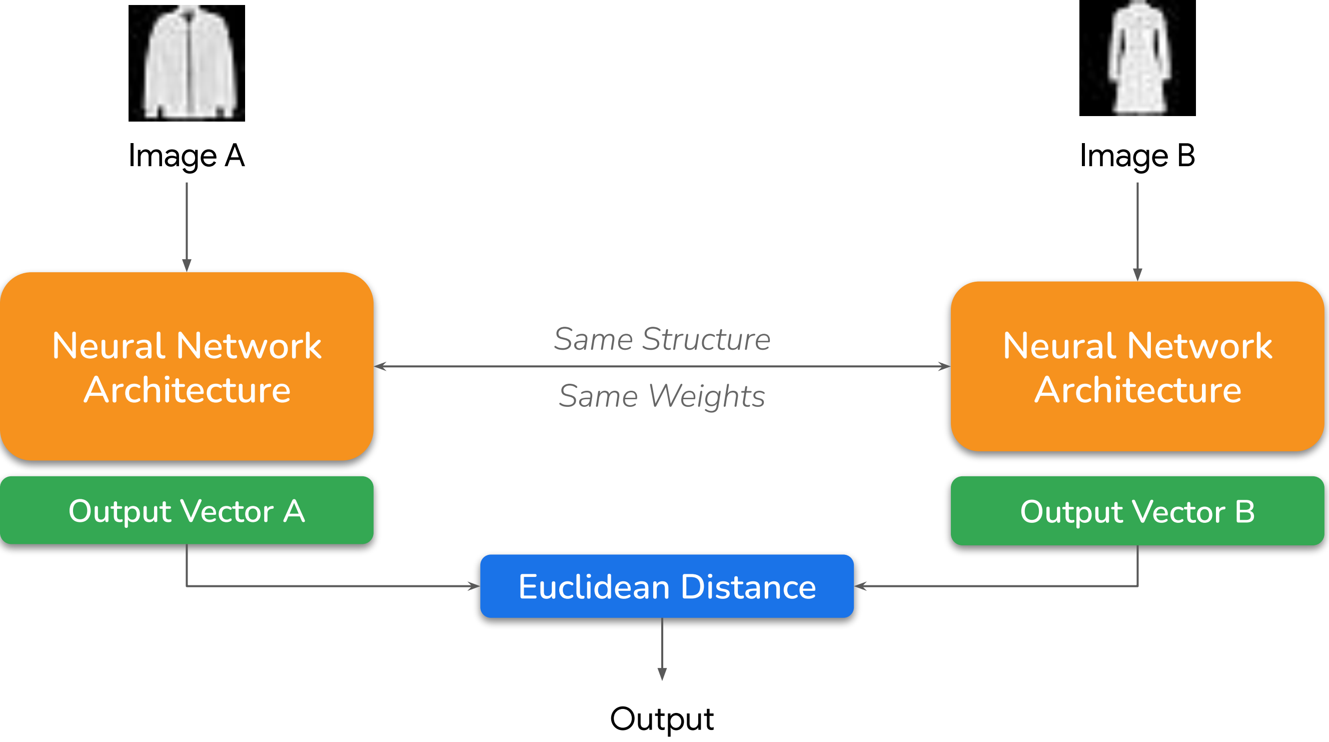


Image A

Image B

Neural Network Architecture

Same Structure

Same Weights

Neural Network Architecture

Output Vector A

Output Vector B

Euclidean Distance

Output

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

Custom Loss Function

```
def contrastive_loss(y_true, y_pred):
    margin = 1
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

$$Y_{true} * Y_{pred}^{2} + (1 - Y_{true}) * max(margin - Y_{pred}, 0)^{2}$$

Custom Loss Function

```python
def contrastive_loss(y_true, y_pred):
    margin = 1
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

$$Y_{true} * \boxed{Y_{pred}^{2}} + (1 - Y_{true}) * max(margin - Y_{pred}, 0)^{2}$$

Custom Loss Function

```python
def contrastive_loss(y_true, y_pred):
    margin = 1
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

$$Y_{true} * Y_{pred}^{2} + (1 - Y_{true}) * \boxed{max(margin - Y_{pred}, 0)^{2}}$$

Custom Loss Function

```
def contrastive_loss(y_true, y_pred):
    margin = 1
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

$$\mathrm{K.mean}[Y_{true} * Y_{pred}{}^{2} + (1 - Y_{true}) * max(margin - Y_{pred}, 0)^{2}]$$

Usage of Custom Loss

```
model.compile(loss=constrastive_loss, optimizer=RMSprop())
```

Usage of Custom Loss

```
model.compile(loss=constrastive_loss, optimizer=RMSprop())
```

Custom Loss Function with Arguments

```
def contrastive_loss_with_margin(margin):
  #Original Loss Function
  def contrastive_loss(y_true, y_pred):
      square_pred = K.square(y_pred)
      margin_square = K.square(K.maximum(margin - y_pred, 0))
      return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
  return contrastive_loss
```

Usage of Wrapper Loss Function

```
model.compile(loss=contrastive_loss_with_margin(margin=1), optimizer=rms)
```

Usage of Wrapper Loss Function

```
model.compile(loss=contrastive_loss_with_margin(margin=1), optimizer=rms)
```

Contrastive Loss - Object Oriented

```
class ContrastiveLoss(Loss):
  margin = 0
  def __init__(self,margin):
    super().__init__()
    self.margin = margin

  def call(self, y_true, y_pred):
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(self.margin - y_pred, 0))
    return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

Contrastive Loss - Object Oriented

```
class ContrastiveLoss(Loss):
 margin = 0
 def __init__(self,margin):
  super().__init__()
  self.margin = margin

 def call(self, y_true, y_pred):
  square_pred = K.square(y_pred)
  margin_square = K.square(K.maximum(self.margin - y_pred, 0))
  return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

Contrastive Loss - Object Oriented

```
class ContrastiveLoss(Loss):
  margin = 0
  def __init__(self,margin):
    super().__init__()
    self.margin = margin

  def call(self, y_true, y_pred):
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(self.margin - y_pred, 0))
    return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
```

Usage of Object Oriented Loss

```
model.compile(loss=ContrastiveLoss(margin=1), optimizer=rms)
```

Usage of Object Oriented Loss

```
model.compile(loss=ContrastiveLoss(margin=1), optimizer=rms)
```