model_training

March 17, 2019

1 Follow-Me Project

Congratulations on reaching the final project of the Robotics Nanodegree!

Previously, you worked on the Semantic Segmentation lab where you built a deep learning network that locates a particular human target within an image. For this project, you will utilize what you implemented and learned from that lab and extend it to train a deep learning model that will allow a simulated quadcopter to follow around the person that it detects!

Most of the code below is similar to the lab with some minor modifications. You can start with your existing solution, and modify and improve upon it to train the best possible model for this task.

You can click on any of the following to quickly jump to that part of this notebook: 1. Section **??** 2. Section **??** 3. Section **??** 4. Section 1.4 5. Section 1.5 6. Section 1.6

1.1 Data Collection

We have provided you with a starting dataset for this project. Download instructions can be found in the README for this project's repo. Alternatively, you can collect additional data of your own to improve your model. Check out the "Collecting Data" section in the Project Lesson in the Classroom for more details!

In [1]:
```
import os
import glob
import sys
import tensorflow as tf

from scipy import misc
import numpy as np

from tensorflow.contrib.keras.python import keras
from tensorflow.contrib.keras.python.keras import layers, models

from tensorflow import image

from utils import scoring_utils
from utils.separable_conv2d import SeparableConv2DKeras, BilinearUpSampling2D
from utils import data_iterator
from utils import plotting_tools
from utils import model_tools
```

1.2 FCN Layers

In the Classroom, we discussed the different layers that constitute a fully convolutional network (FCN). The following code will introduce you to the functions that you need to build your semantic segmentation model.

1.2.1 Separable Convolutions

The Encoder for your FCN will essentially require separable convolution layers, due to their advantages as explained in the classroom. The 1x1 convolution layer in the FCN, however, is a regular convolution. Implementations for both are provided below for your use. Each includes batch normalization with the ReLU activation function applied to the layers.

```
In [2]: def separable_conv2d_batchnorm(input_layer, filters, strides=1):
            output_layer = SeparableConv2DKeras(filters=filters,kernel_size=3, strides=strides,
                                        padding='same', activation='relu')(input_layer)

            output_layer = layers.BatchNormalization()(output_layer)
            return output_layer

        def conv2d_batchnorm(input_layer, filters, kernel_size=3, strides=1):
            output_layer = layers.Conv2D(filters=filters, kernel_size=kernel_size, strides=strid
                               padding='same', activation='relu')(input_layer)

            output_layer = layers.BatchNormalization()(output_layer)
            return output_layer
```

1.2.2 Bilinear Upsampling

The following helper function implements the bilinear upsampling layer. Upsampling by a factor of 2 is generally recommended, but you can try out different factors as well. Upsampling is used in the decoder block of the FCN.

```
In [3]: def bilinear_upsample(input_layer):
            output_layer = BilinearUpSampling2D((2,2))(input_layer)
            return output_layer
```

1.3 Build the Model

In the following cells, you will build an FCN to train a model to detect and locate the hero target within an image. The steps are: - Create an `encoder_block` - Create a `decoder_block` - Build the FCN consisting of encoder block(s), a 1x1 convolution, and decoder block(s). This step requires experimentation with different numbers of layers and filter sizes to build your model.

1.3.1 Encoder Block

Create an encoder block that includes a separable convolution layer using the `separable_conv2d_batchnorm()` function. The `filters` parameter defines the size or depth of the output layer. For example, 32 or 64.

```
In [4]: def encoder_block(input_layer, filters, strides):

            # TODO Create a separable convolution layer using the separable_conv2d_batchnorm() f
            output_layer = separable_conv2d_batchnorm(input_layer, filters, strides)
            return output_layer
```

1.3.2 Decoder Block

The decoder block is comprised of three parts: - A bilinear upsampling layer using the upsample_bilinear() function. The current recommended factor for upsampling is set to 2. - A layer concatenation step. This step is similar to skip connections. You will concatenate the upsampled small_ip_layer and the large_ip_layer. - Some (one or two) additional separable convolution layers to extract some more spatial information from prior layers.

```
In [5]: def decoder_block(small_ip_layer, large_ip_layer, filters):

            # TODO Upsample the small input layer using the bilinear_upsample() function.
            upsampled_layer = bilinear_upsample(small_ip_layer)
            print("upsampled-shape:", upsampled_layer.get_shape())
            # TODO Concatenate the upsampled and large input layers using layers.concatenate
            print("large_ip_layer-shape", large_ip_layer.get_shape())
            concatenate_layers = layers.concatenate([upsampled_layer,large_ip_layer])
            print("concatenate-shape:", concatenate_layers.get_shape())
            # TODO Add some number of separable convolution layers
            output_layer = separable_conv2d_batchnorm(concatenate_layers, filters, strides = 1)
            print("decoder_output-shpae:", output_layer.get_shape())
            return output_layer

            return output_layer
```

1.3.3 Model

Now that you have the encoder and decoder blocks ready, go ahead and build your FCN architecture!

There are three steps: - Add encoder blocks to build the encoder layers. This is similar to how you added regular convolutional layers in your CNN lab. - Add a 1x1 Convolution layer using the conv2d_batchnorm() function. Remember that 1x1 Convolutions require a kernel and stride of 1. - Add decoder blocks for the decoder layers.

```
In [6]: def fcn_model(inputs, num_classes):

            # TODO Add Encoder Blocks.
            # Remember that with each encoder layer, the depth of your model (the number of filt
            encoder_1 =  encoder_block(inputs, 64,strides =2)
            print("encoder_1-shape:", encoder_1.get_shape())
            encoder_2 =  encoder_block(encoder_1, 128,strides =2)
            print("encoder_2-shape:", encoder_2.get_shape())
            # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
            conv_1x1 = conv2d_batchnorm(encoder_2, filters = 256, kernel_size=1, strides=1)
```

```
    print("conv_1x1-shape:", conv_1x1.get_shape())
    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    decoder_1 = decoder_block(conv_1x1, encoder_1, 64)
    decoder_2 = decoder_block(decoder_1, inputs, 32)
    # The function returns the output layer of your model. "x" is the final layer obtain
    return layers.Conv2D(num_classes, 3, activation='softmax', padding='same')(decoder_2
```

1.4 Training

The following cells will use the FCN you created and define an ouput layer based on the size of the processed image and the number of classes recognized. You will define the hyperparameters to compile and train your model.

Please Note: For this project, the helper code in `data_iterator.py` will resize the copter images to 160x160x3 to speed up training.

In [7]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

image_hw = 160
image_shape = (image_hw, image_hw, 3)
inputs = layers.Input(image_shape)
num_classes = 3

# Call fcn_model()
output_layer = fcn_model(inputs, num_classes)
print("output-shpae:", output_layer.get_shape())
```

```
encoder_1-shape: (?, 80, 80, 64)
encoder_2-shape: (?, 40, 40, 128)
conv_1x1-shape: (?, 40, 40, 256)
upsampled-shape: (?, 80, 80, 256)
large_ip_layer-shape (?, 80, 80, 64)
concatenate-shape: (?, 80, 80, 320)
decoder_output-shpae: (?, 80, 80, 64)
upsampled-shape: (?, 160, 160, 64)
large_ip_layer-shape (?, 160, 160, 3)
concatenate-shape: (?, 160, 160, 67)
decoder_output-shpae: (?, 160, 160, 32)
output-shpae: (?, 160, 160, 3)
```

1.4.1 Hyperparameters

Define and tune your hyperparameters. - **batch_size**: number of training samples/images that get propagated through the network in a single pass. - **num_epochs**: number of times the entire training dataset gets propagated through the network. - **steps_per_epoch**: number of batches of

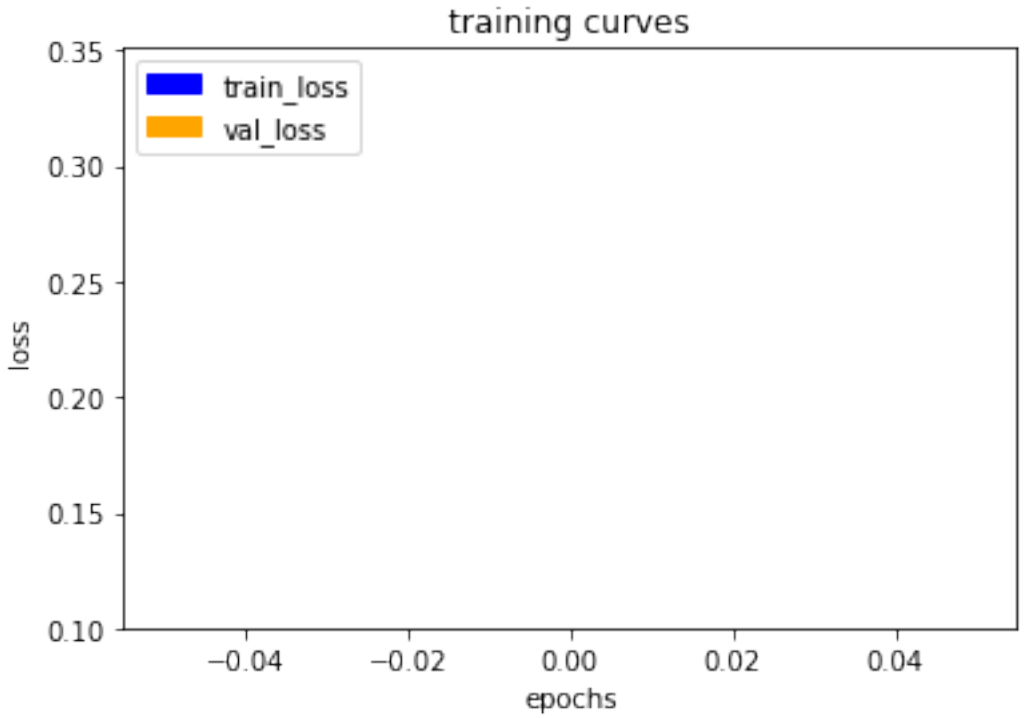
training images that go through the network in 1 epoch. We have provided you with a default value. One recommended value to try would be based on the total number of images in training dataset divided by the batch_size. - **validation_steps**: number of batches of validation images that go through the network in 1 epoch. This is similar to steps_per_epoch, except validation_steps is for the validation dataset. We have provided you with a default value for this as well. - **workers**: maximum number of processes to spin up. This can affect your training speed and is dependent on your hardware. We have provided a recommended value to work with.

In [8]:
```
learning_rate = 0.001
batch_size = 32
num_epochs = 100
steps_per_epoch = 200
validation_steps = 50
workers = 4
```

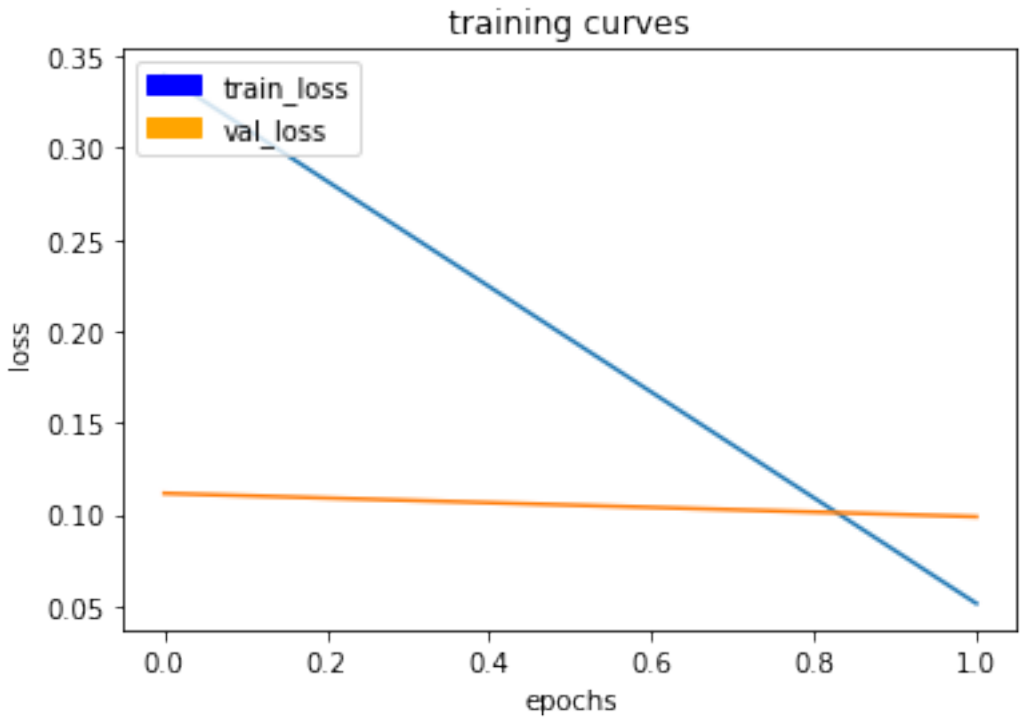
In [9]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

from workspace_utils import active_session
# Keeping Your Session Active
with active_session():
    # Define the Keras model and compile it for training
    model = models.Model(inputs=inputs, outputs=output_layer)

    model.compile(optimizer=keras.optimizers.Adam(learning_rate), loss='categorical_cros

    # Data iterators for loading the training and validation data
    train_iter = data_iterator.BatchIteratorSimple(batch_size=batch_size,
                                                   data_folder=os.path.join('..', 'data'
                                                   image_shape=image_shape,
                                                   shift_aug=True)

    val_iter = data_iterator.BatchIteratorSimple(batch_size=batch_size,
                                                 data_folder=os.path.join('..', 'data',
                                                 image_shape=image_shape)

    logger_cb = plotting_tools.LoggerPlotter()
    callbacks = [logger_cb]

    model.fit_generator(train_iter,
                        steps_per_epoch = steps_per_epoch, # the number of batches per e
                        epochs = num_epochs, # the number of epochs to train for,
                        validation_data = val_iter, # validation iterator
                        validation_steps = validation_steps, # the number of batches to
                        callbacks=callbacks,
                        workers = workers)
```

```
Epoch 1/100
```

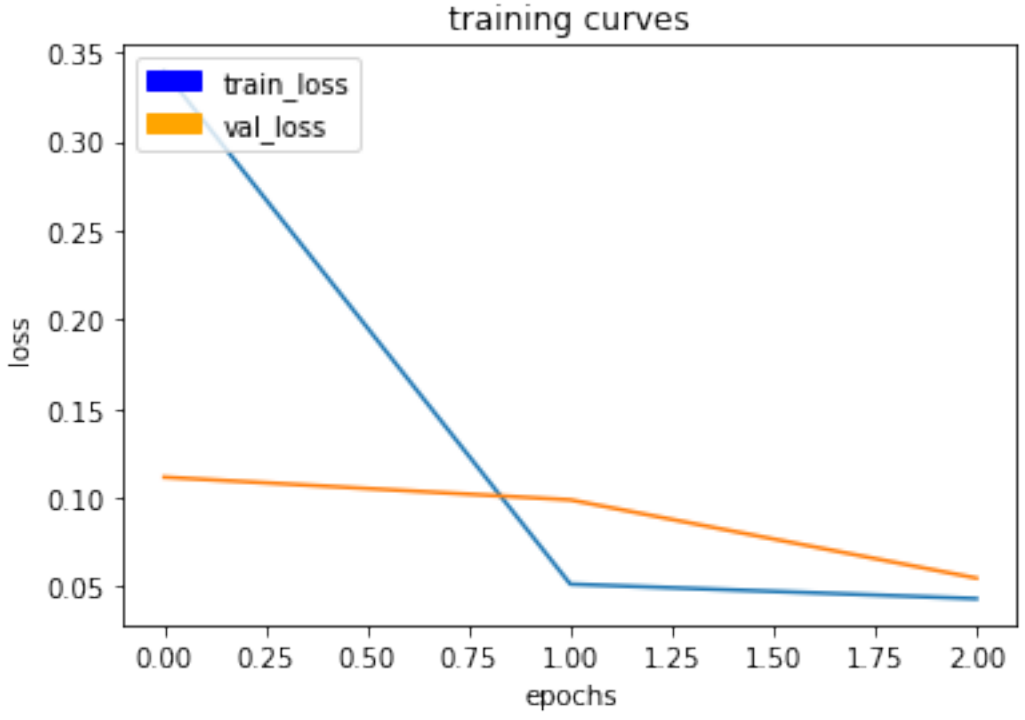
```
199/200 [============================>.] - ETA: 0s - loss: 0.3396
```



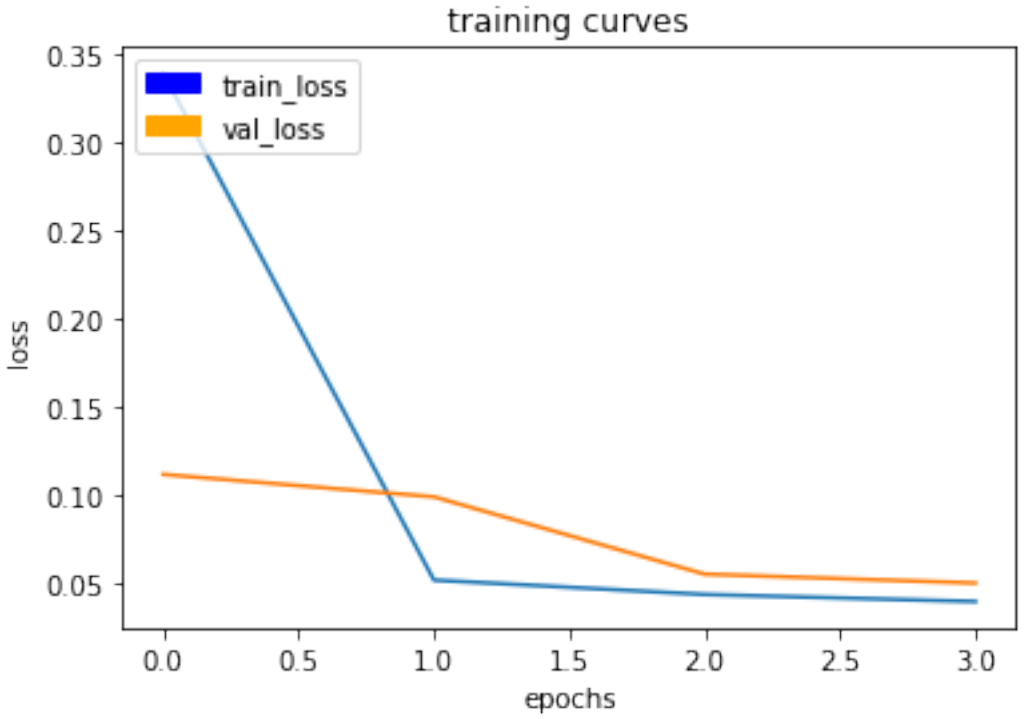

```
200/200 [==============================] - 101s - loss: 0.3383 - val_loss: 0.1114
Epoch 2/100
199/200 [============================>.] - ETA: 0s - loss: 0.0512
```



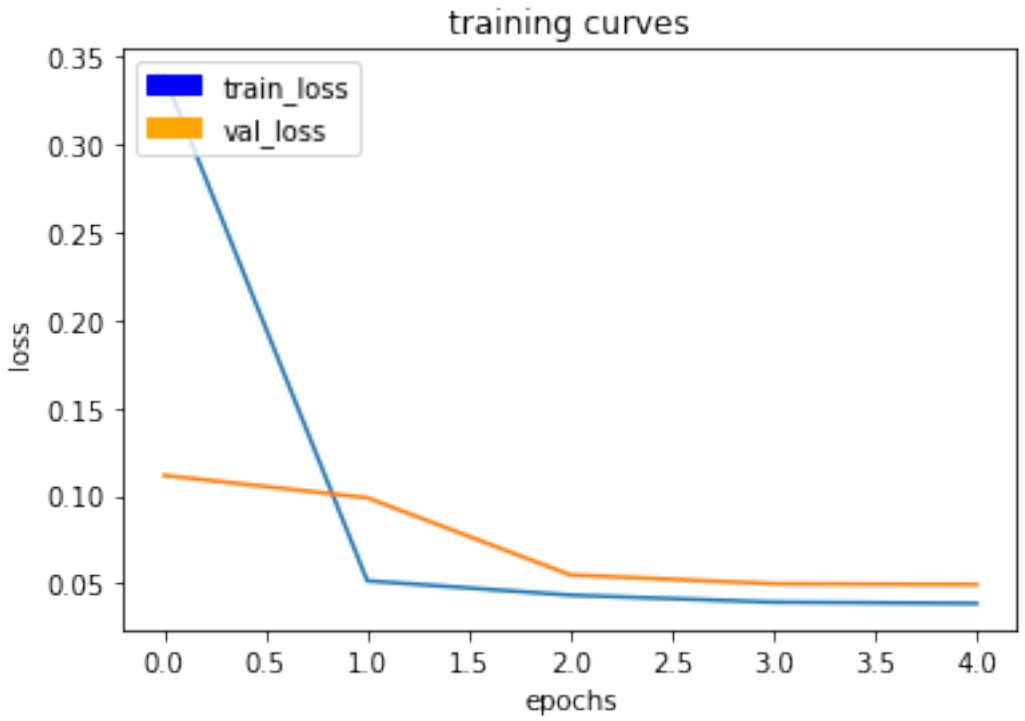

```
200/200 [==============================] - 96s - loss: 0.0513 - val_loss: 0.0987
Epoch 3/100
199/200 [============================>.] - ETA: 0s - loss: 0.0432
```



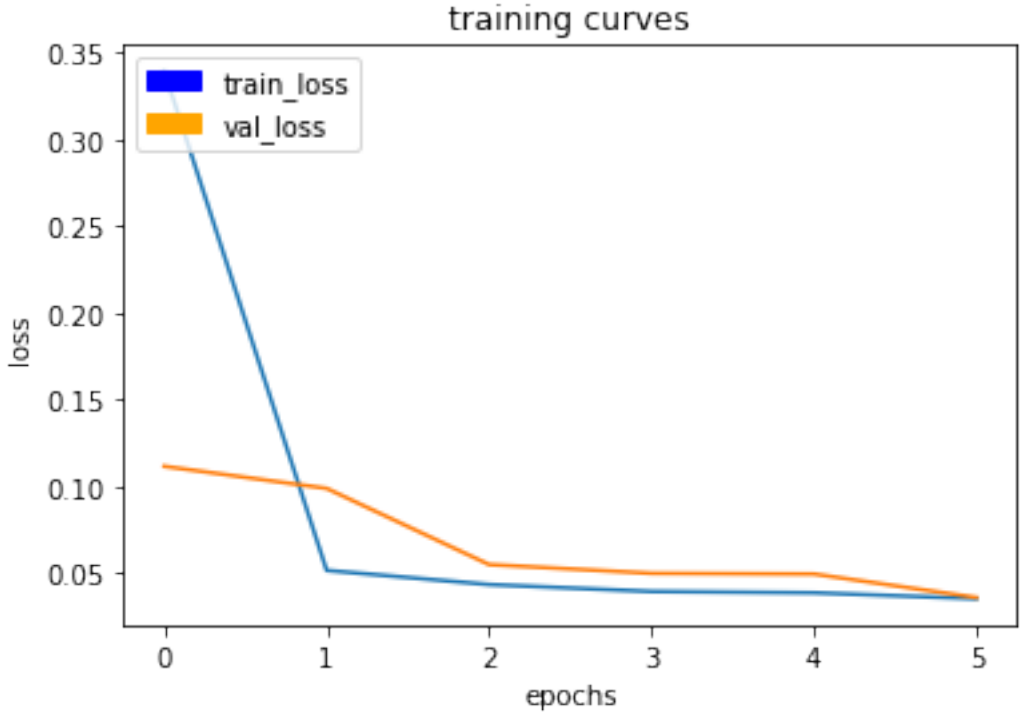
```
200/200 [==============================] - 97s - loss: 0.0432 - val_loss: 0.0547
Epoch 4/100
199/200 [============================>.] - ETA: 0s - loss: 0.0393
```



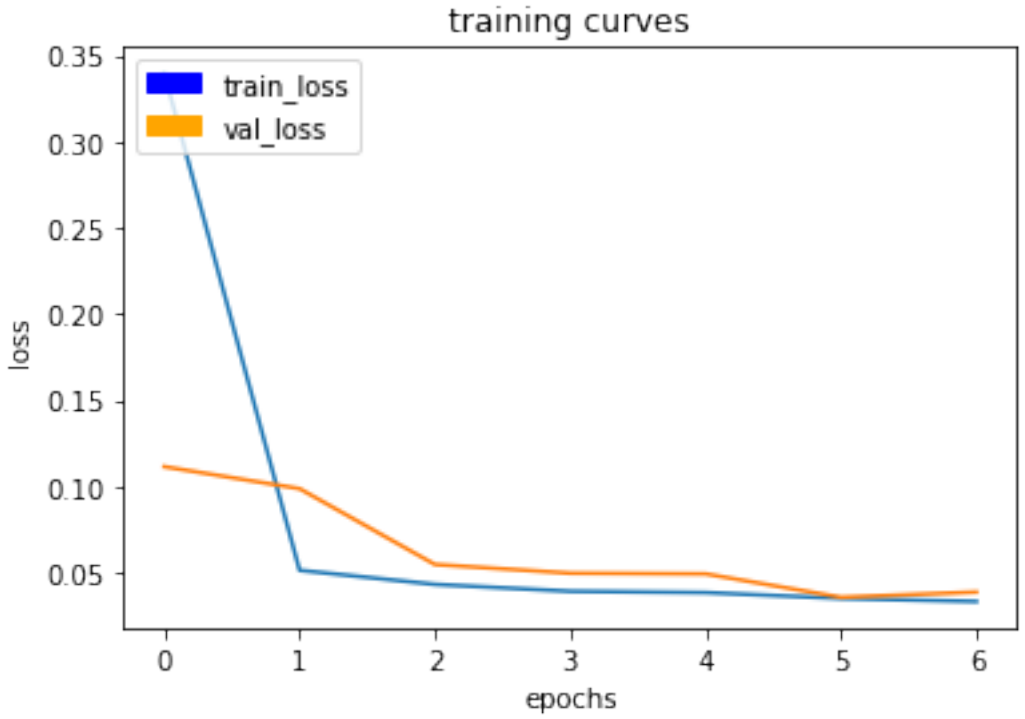

```
200/200 [==============================] - 96s - loss: 0.0393 - val_loss: 0.0497
Epoch 5/100
199/200 [============================>.] - ETA: 0s - loss: 0.0385
```



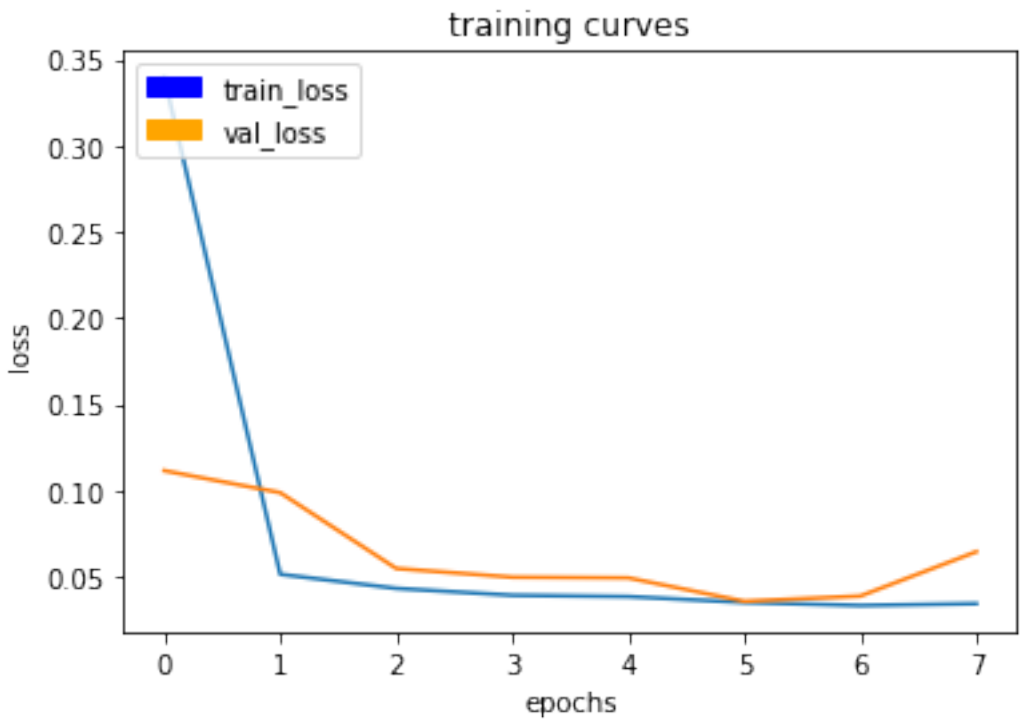

```
200/200 [==============================] - 97s - loss: 0.0385 - val_loss: 0.0493
Epoch 6/100
199/200 [============================>.] - ETA: 0s - loss: 0.0350
```



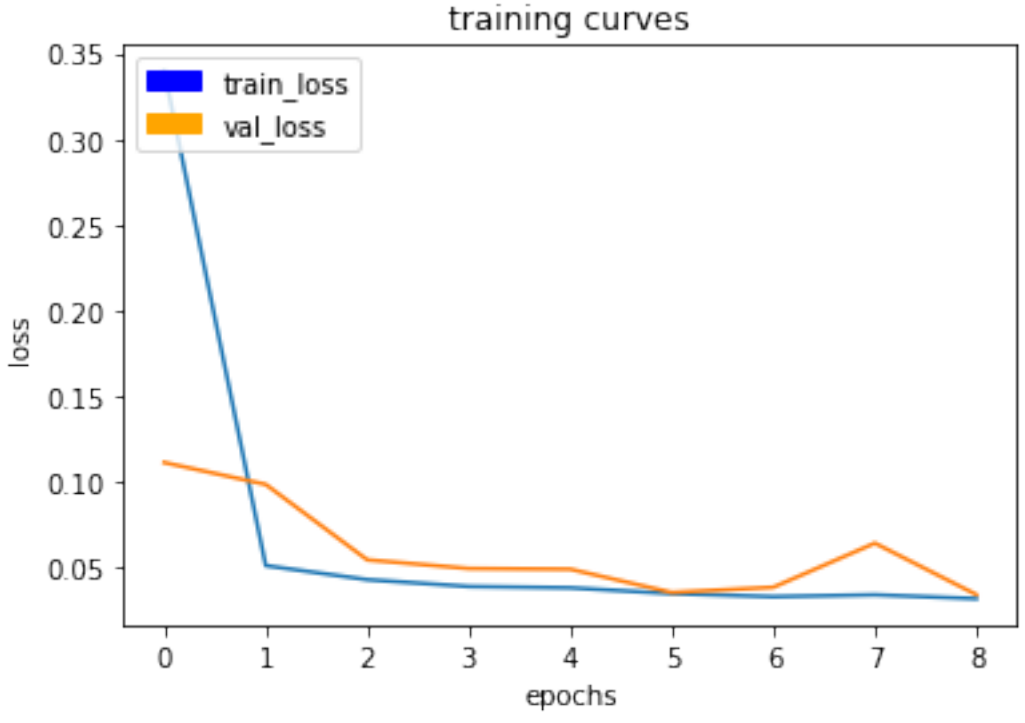
```
200/200 [==============================] - 96s - loss: 0.0350 - val_loss: 0.0357
Epoch 7/100
199/200 [============================>.] - ETA: 0s - loss: 0.0332
```



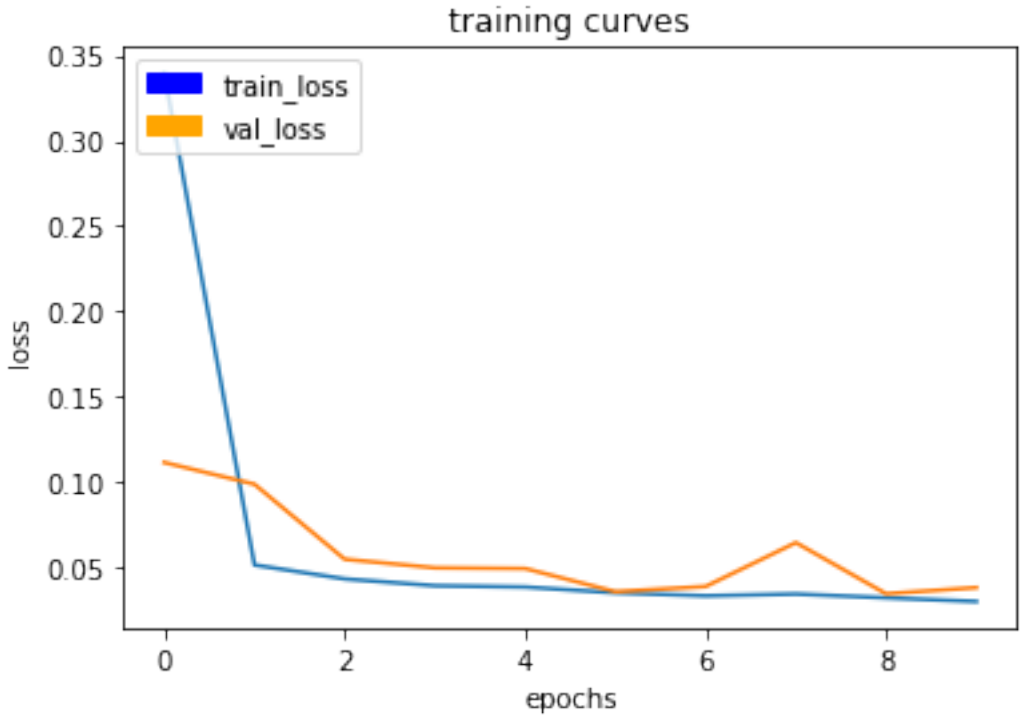

```
200/200 [==============================] - 97s - loss: 0.0332 - val_loss: 0.0387
Epoch 8/100
199/200 [============================>.] - ETA: 0s - loss: 0.0349
```



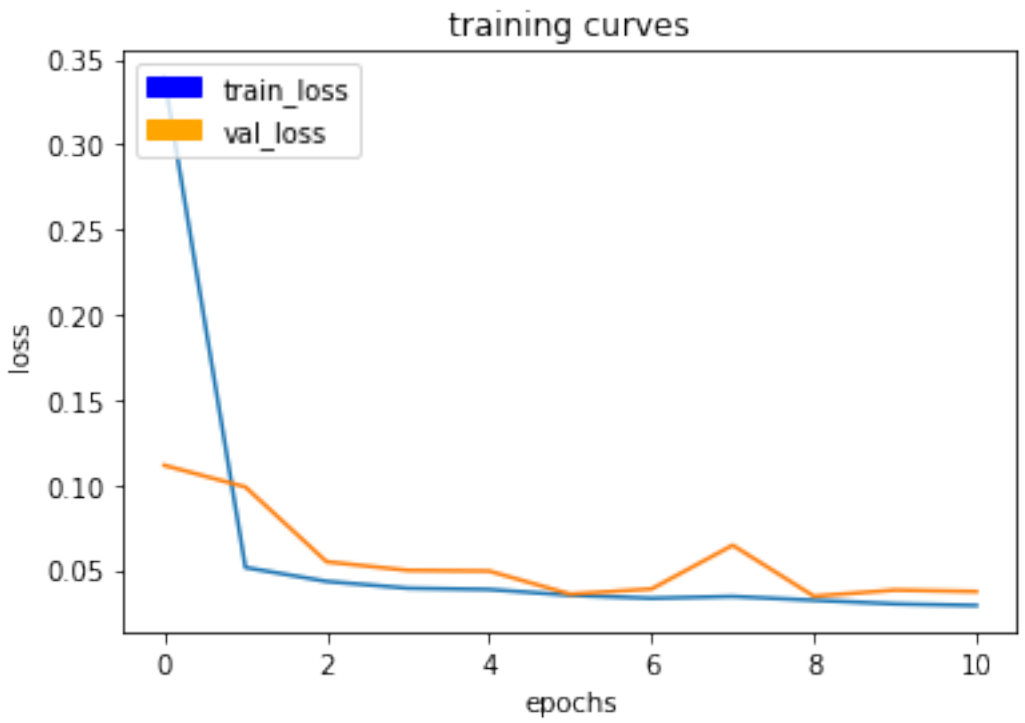

```
200/200 [==============================] - 96s - loss: 0.0349 - val_loss: 0.0645
Epoch 9/100
199/200 [============================>.] - ETA: 0s - loss: 0.0322
```



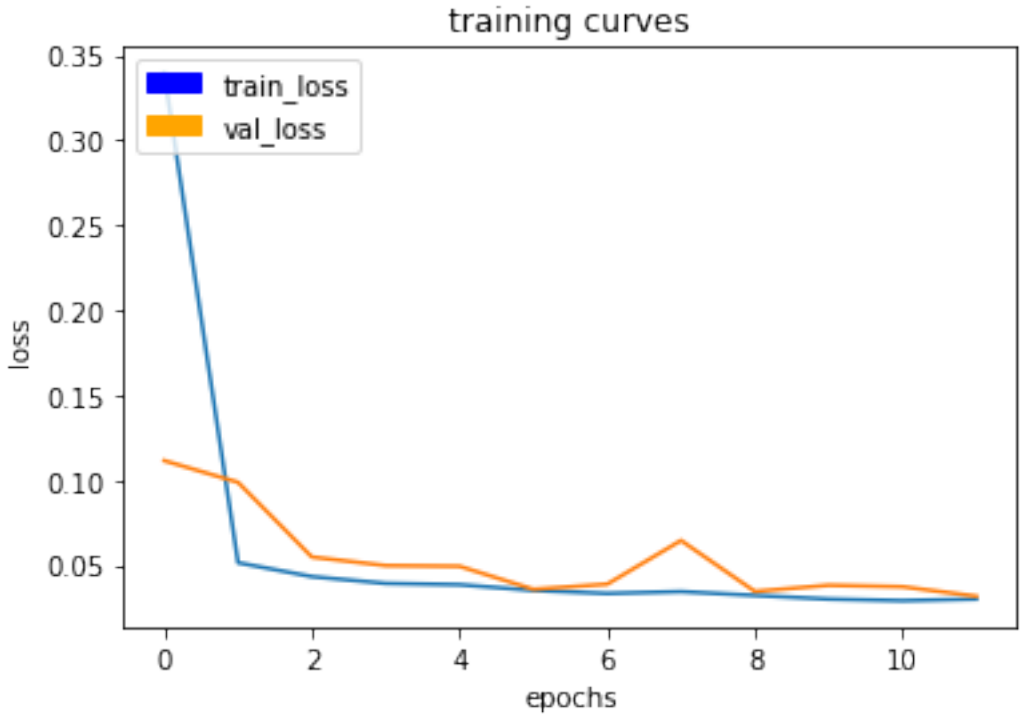
```
200/200 [==============================] - 96s - loss: 0.0321 - val_loss: 0.0345
Epoch 10/100
199/200 [============================>.] - ETA: 0s - loss: 0.0301
```



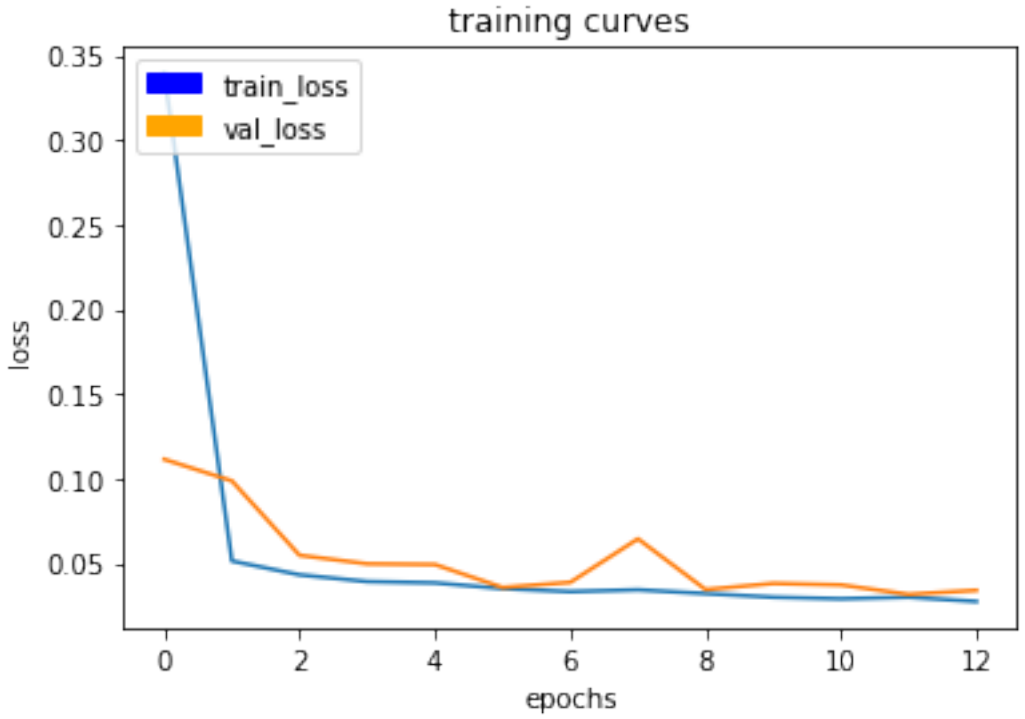

```
200/200 [==============================] - 97s - loss: 0.0301 - val_loss: 0.0381
Epoch 11/100
199/200 [============================>.] - ETA: 0s - loss: 0.0290
```



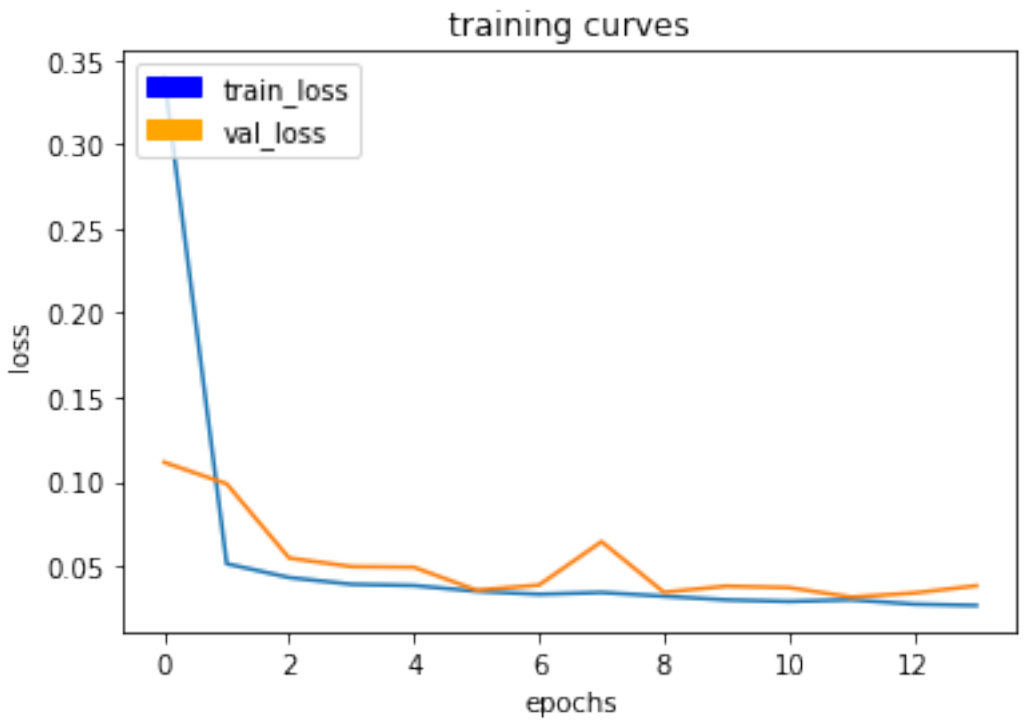

```
200/200 [==============================] - 97s - loss: 0.0290 - val_loss: 0.0372
Epoch 12/100
199/200 [============================>.] - ETA: 0s - loss: 0.0300
```



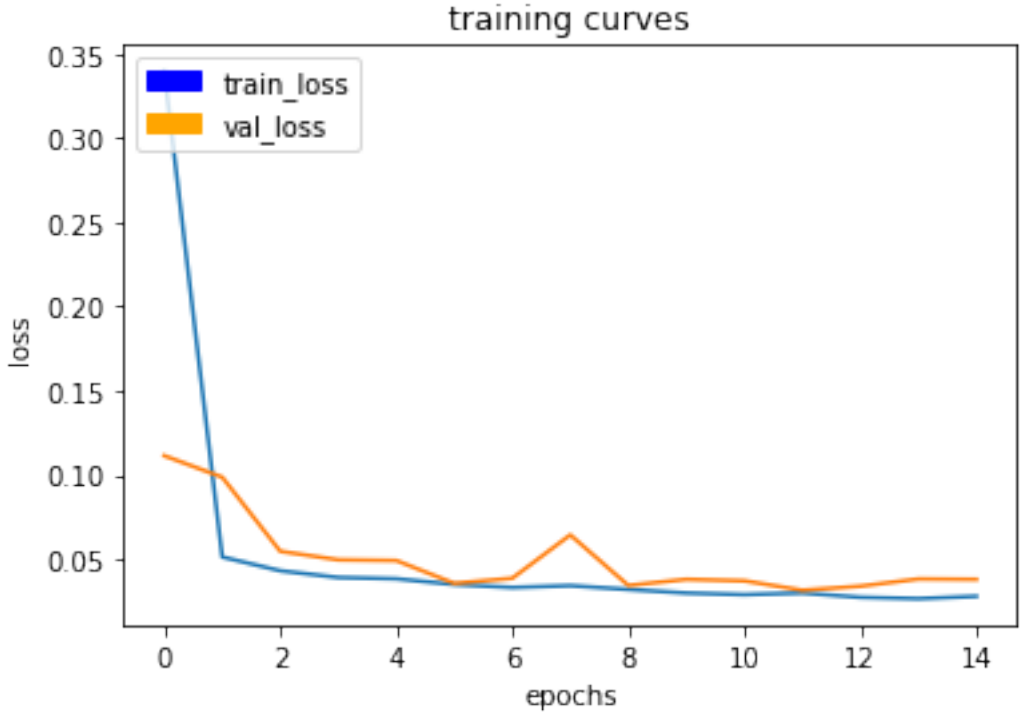
```
200/200 [==============================] - 96s - loss: 0.0300 - val_loss: 0.0316
Epoch 13/100
199/200 [============================>.] - ETA: 0s - loss: 0.0274
```



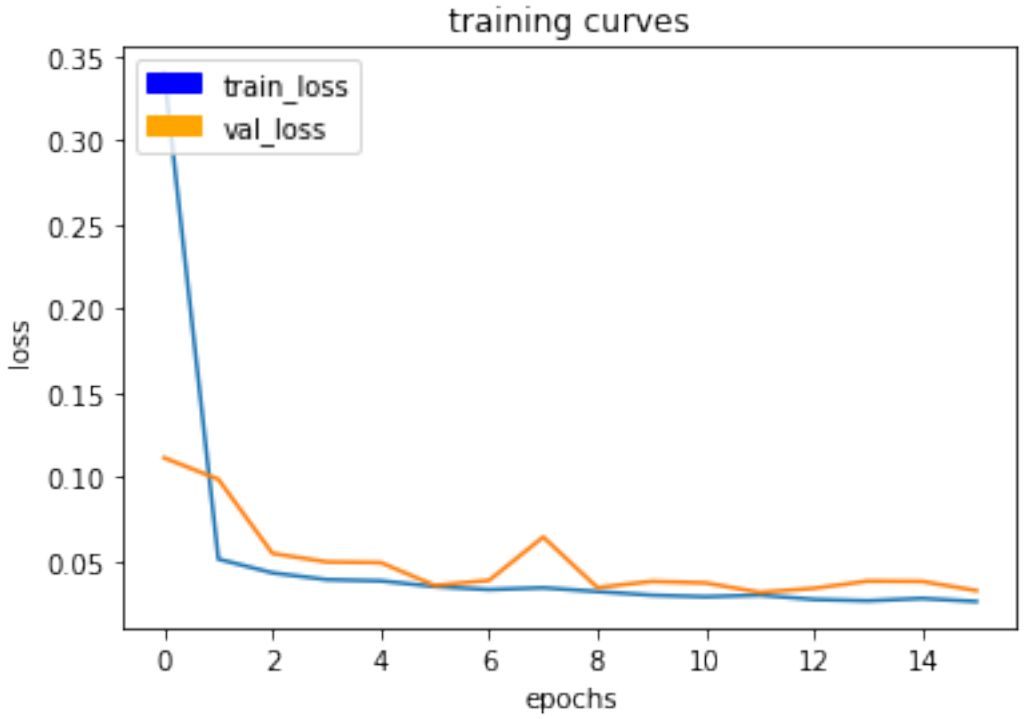

```
200/200 [==============================] - 96s - loss: 0.0274 - val_loss: 0.0340
Epoch 14/100
199/200 [============================>.] - ETA: 0s - loss: 0.0265
```



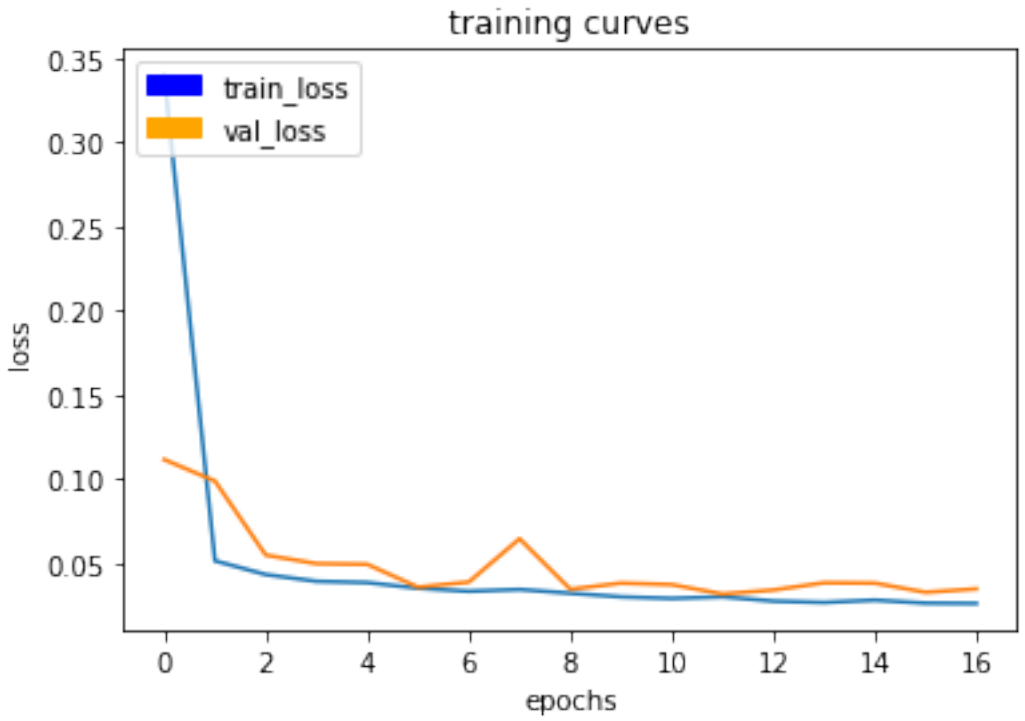

```
200/200 [==============================] - 96s - loss: 0.0267 - val_loss: 0.0383
Epoch 15/100
199/200 [============================>.] - ETA: 0s - loss: 0.0280
```



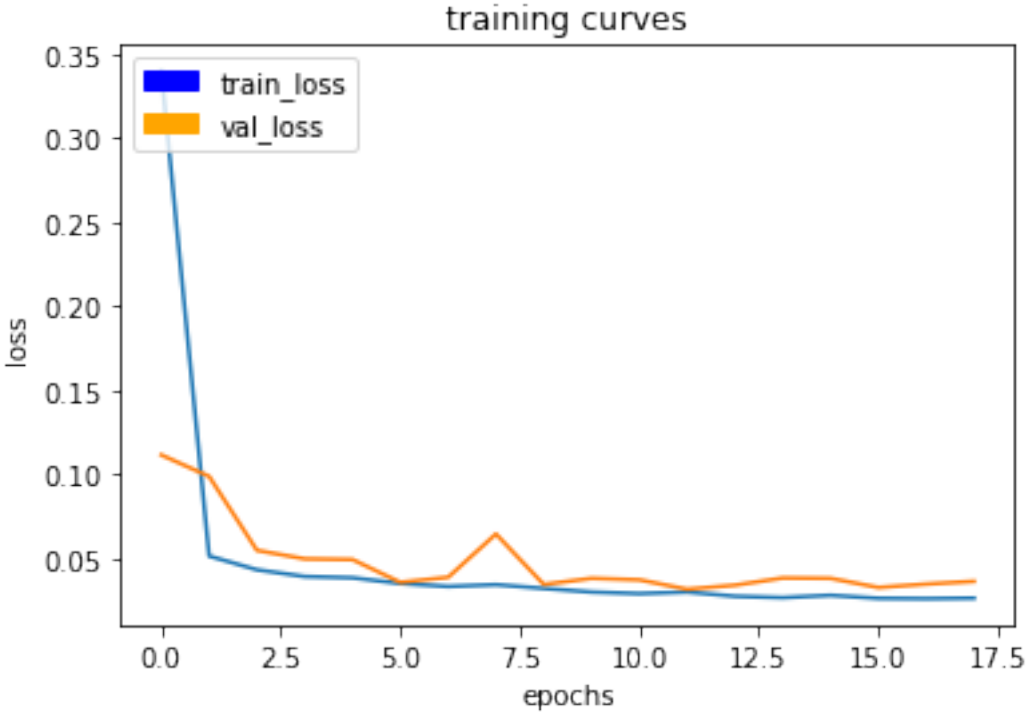
```
200/200 [==============================] - 97s - loss: 0.0280 - val_loss: 0.0381
Epoch 16/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0262
```



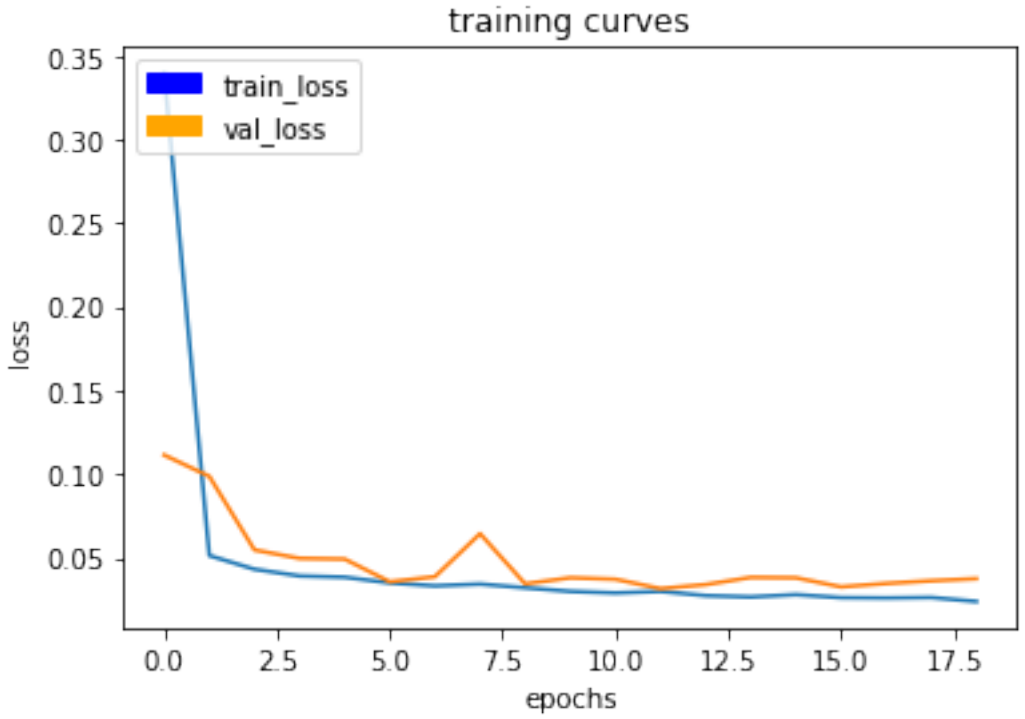

```
200/200 [==============================] - 97s - loss: 0.0262 - val_loss: 0.0327
Epoch 17/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0260
```



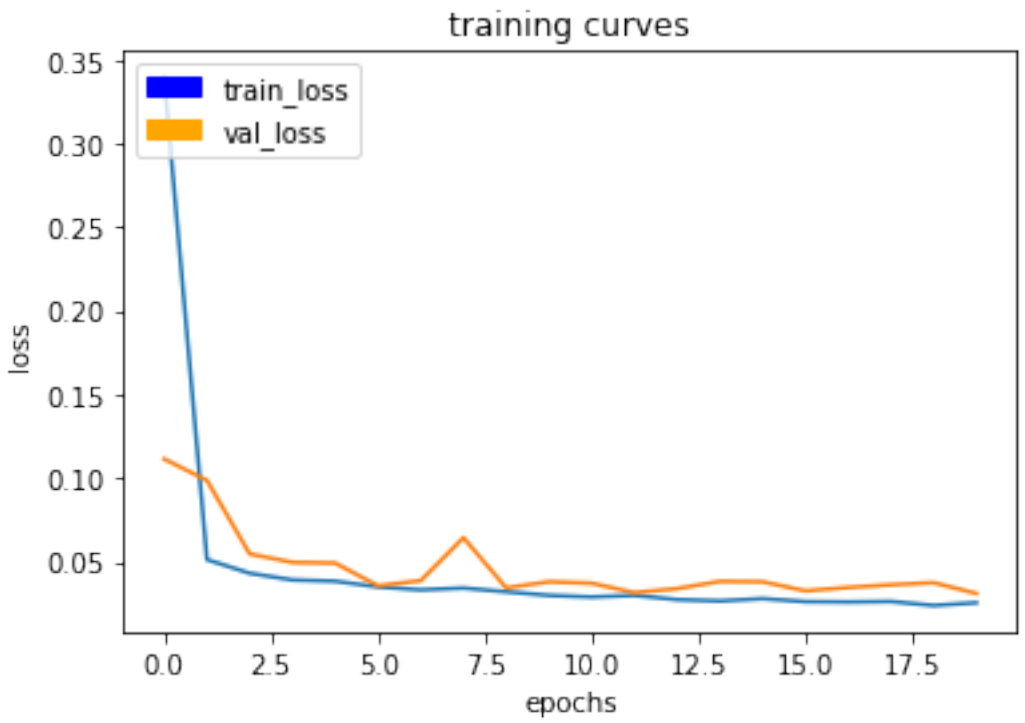

```
200/200 [==============================] - 96s - loss: 0.0260 - val_loss: 0.0347
Epoch 18/100
199/200 [============================>.] - ETA: 0s - loss: 0.0263
```



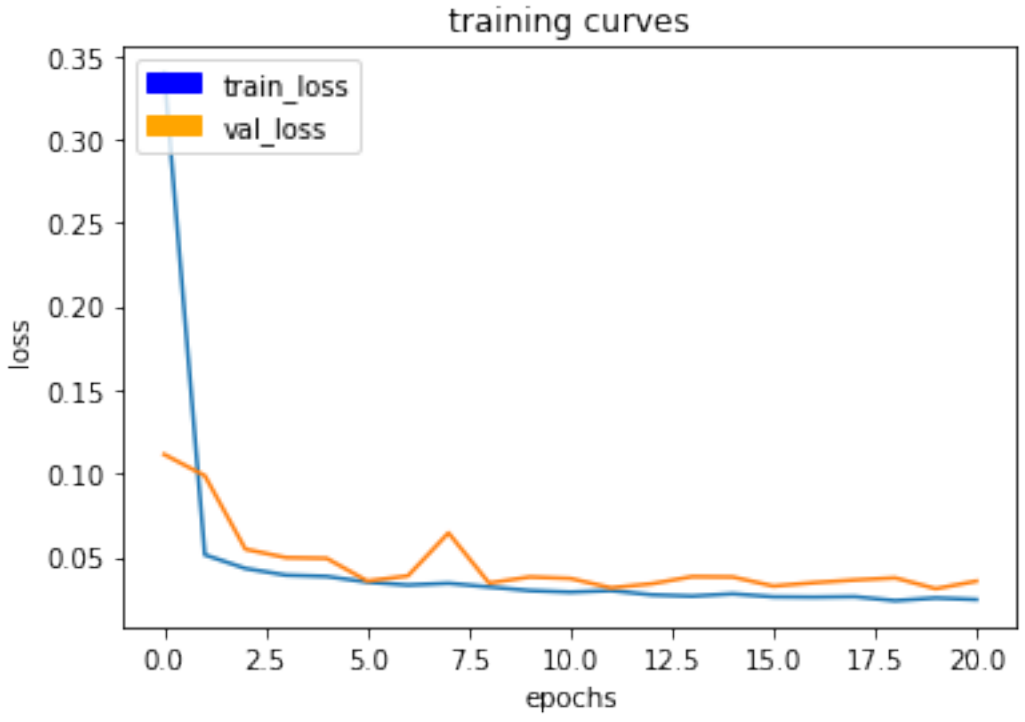
```
200/200 [==============================] - 97s - loss: 0.0263 - val_loss: 0.0363
Epoch 19/100
199/200 [============================>.] - ETA: 0s - loss: 0.0239
```



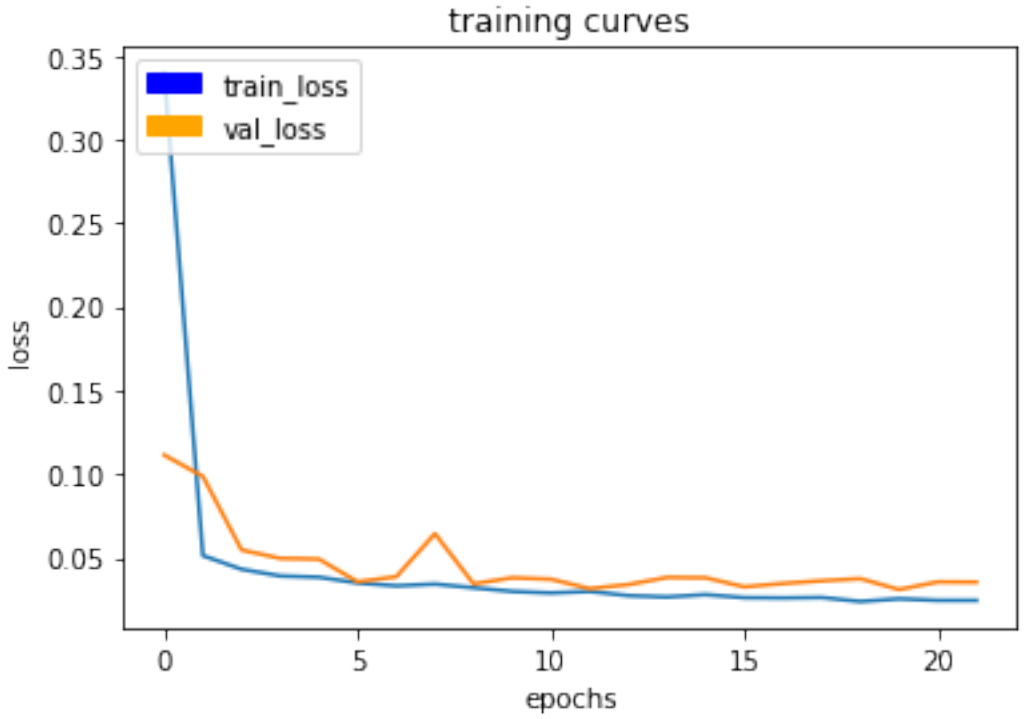

```
200/200 [==============================] - 96s - loss: 0.0239 - val_loss: 0.0375
Epoch 20/100
199/200 [============================>.] - ETA: 0s - loss: 0.0256
```



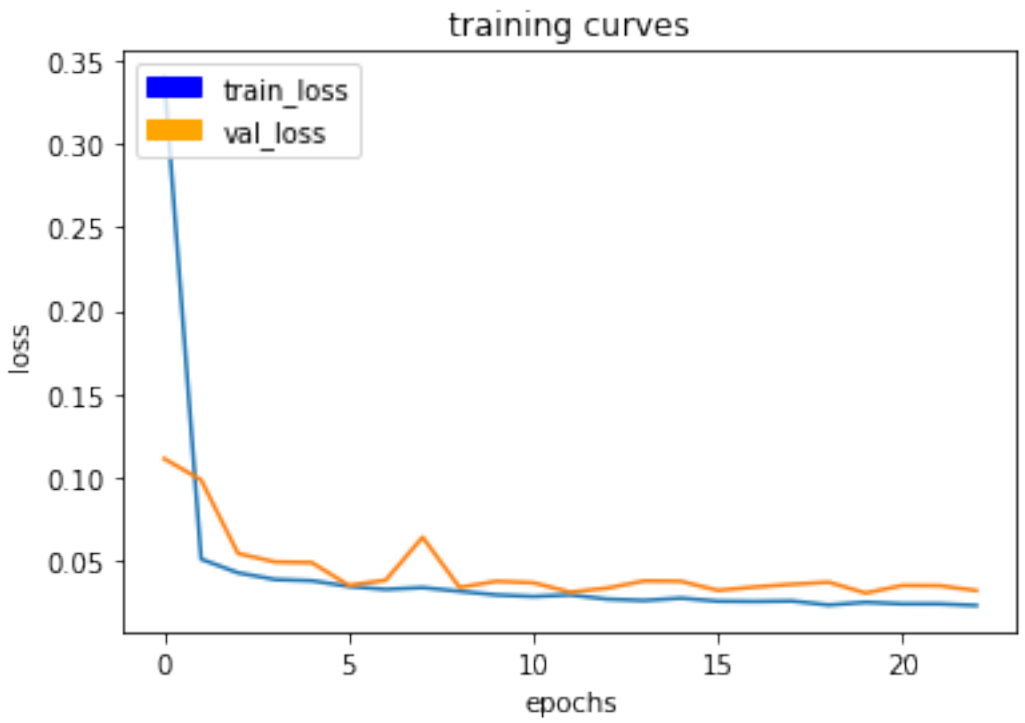

```
200/200 [==============================] - 97s - loss: 0.0256 - val_loss: 0.0311
Epoch 21/100
199/200 [============================>.] - ETA: 0s - loss: 0.0245
```



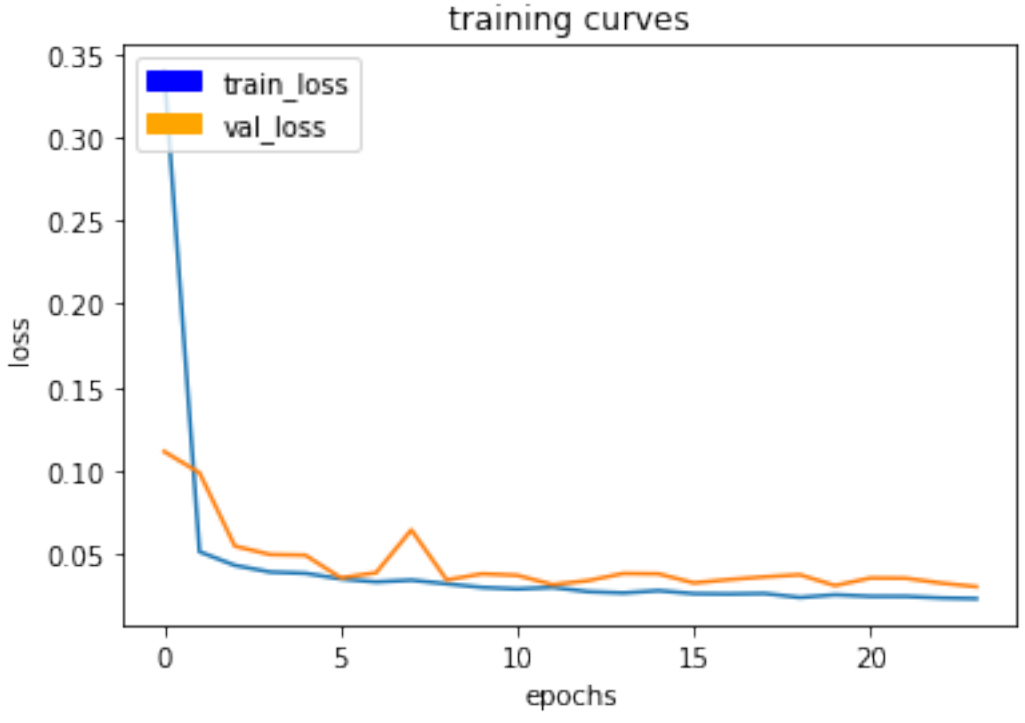
```
200/200 [==============================] - 96s - loss: 0.0245 - val_loss: 0.0356
Epoch 22/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0246
```



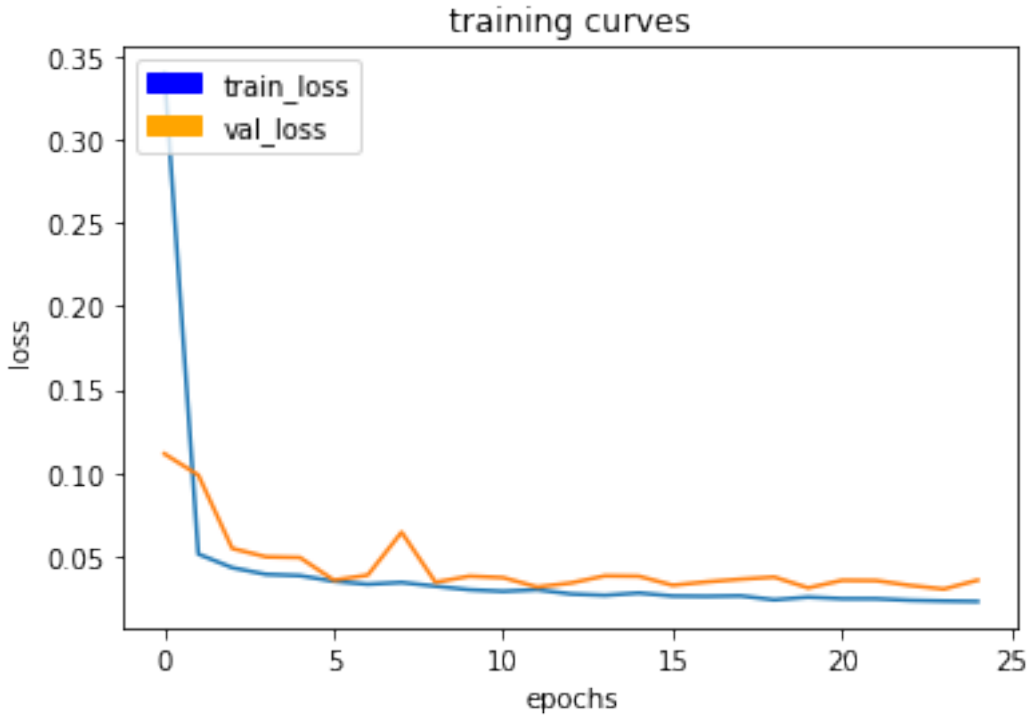

```
200/200 [==============================] - 96s - loss: 0.0248 - val_loss: 0.0355
Epoch 23/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0239
```



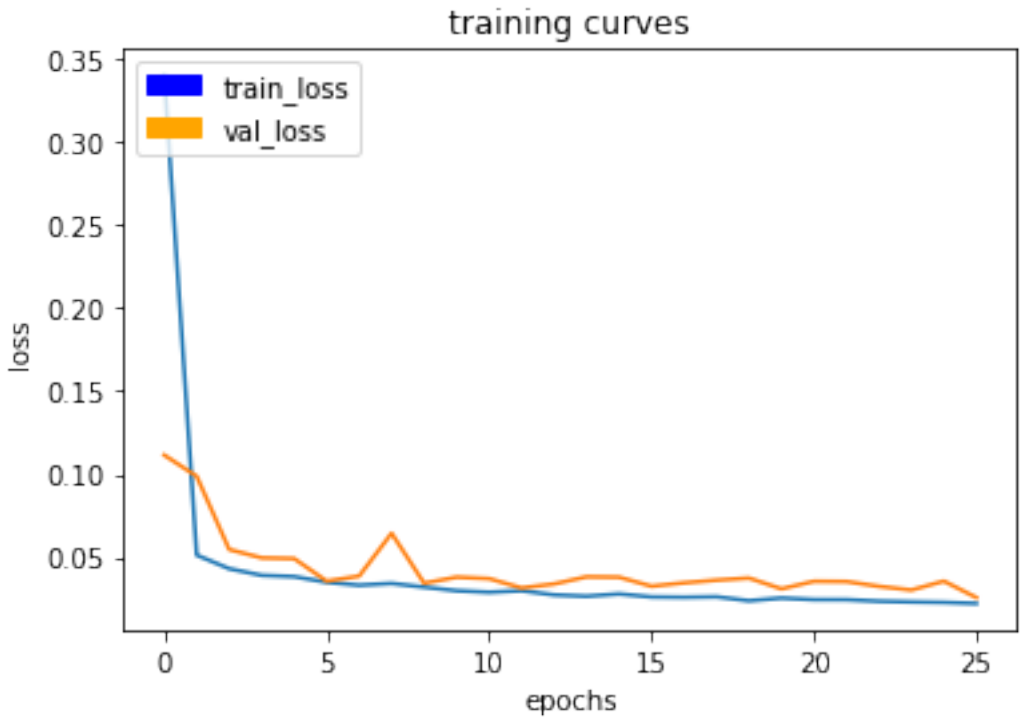

```
200/200 [==============================] - 96s - loss: 0.0239 - val_loss: 0.0326
Epoch 24/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0233
```



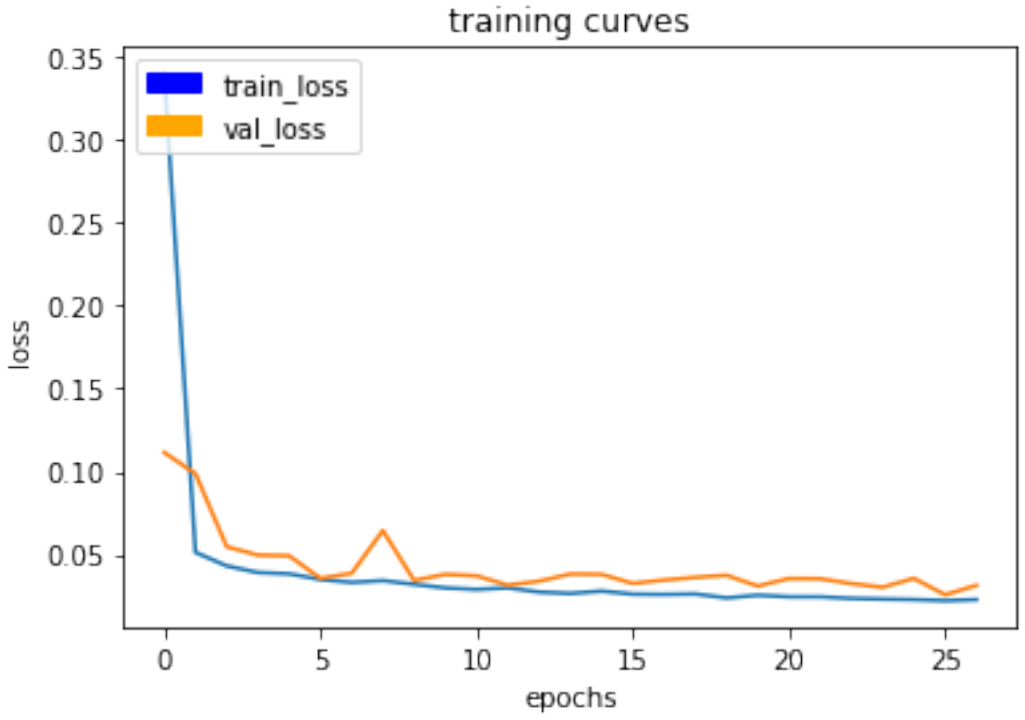
```
200/200 [==============================] - 96s - loss: 0.0232 - val_loss: 0.0304
Epoch 25/100
199/200 [============================>.] - ETA: 0s - loss: 0.0230
```



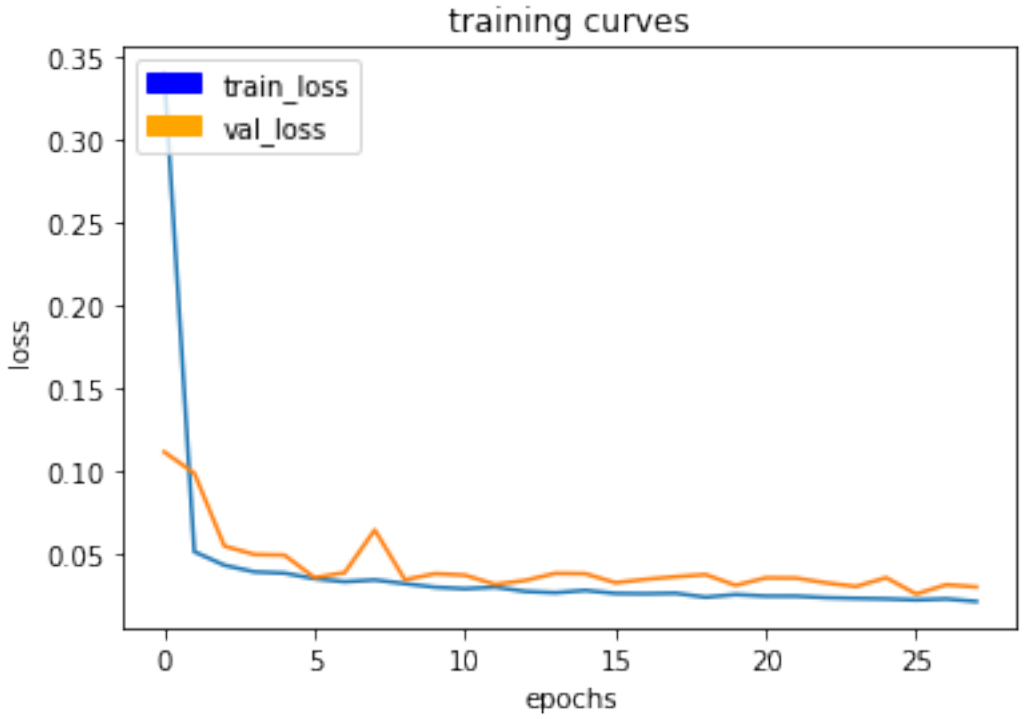

```
200/200 [==============================] - 97s - loss: 0.0230 - val_loss: 0.0357
Epoch 26/100
199/200 [============================>.] - ETA: 0s - loss: 0.0223
```



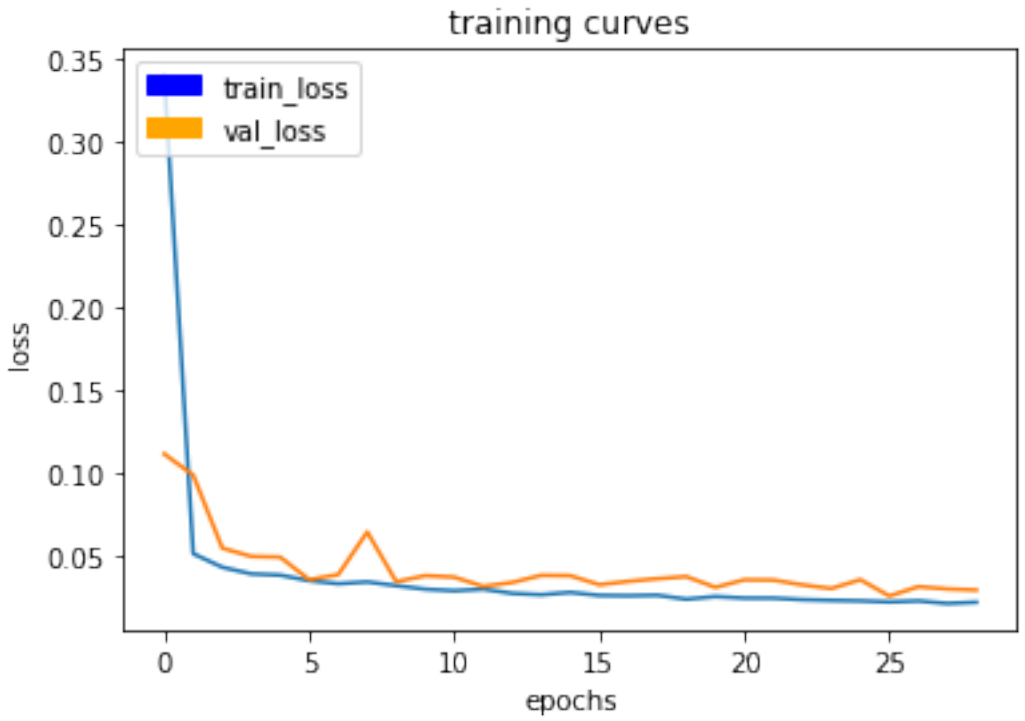

```
200/200 [==============================] - 96s - loss: 0.0223 - val_loss: 0.0258
Epoch 27/100
199/200 [============================>.] - ETA: 0s - loss: 0.0230
```



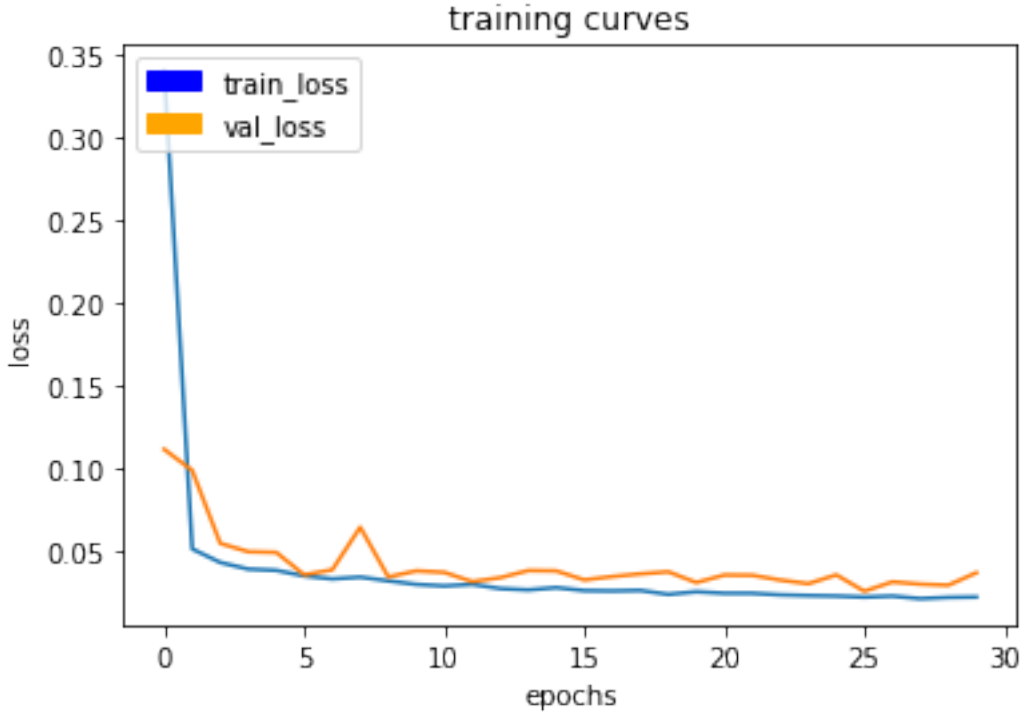
```
200/200 [==============================] - 97s - loss: 0.0230 - val_loss: 0.0315
Epoch 28/100
199/200 [============================>.] - ETA: 0s - loss: 0.0214
```



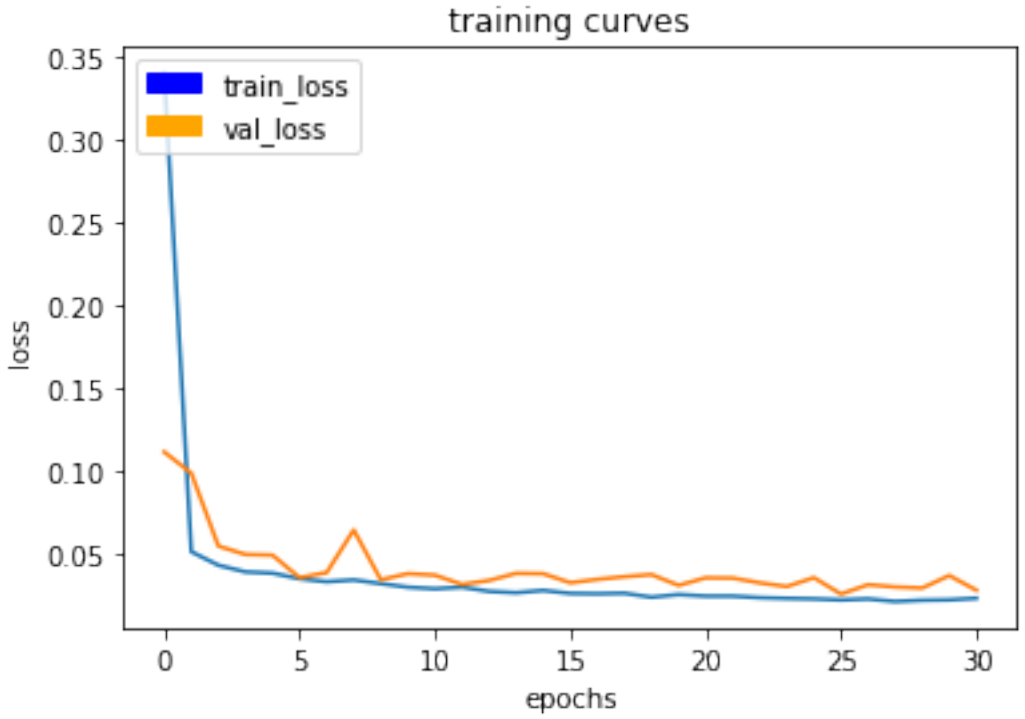

```
200/200 [==============================] - 97s - loss: 0.0213 - val_loss: 0.0301
Epoch 29/100
199/200 [============================>.] - ETA: 0s - loss: 0.0221
```



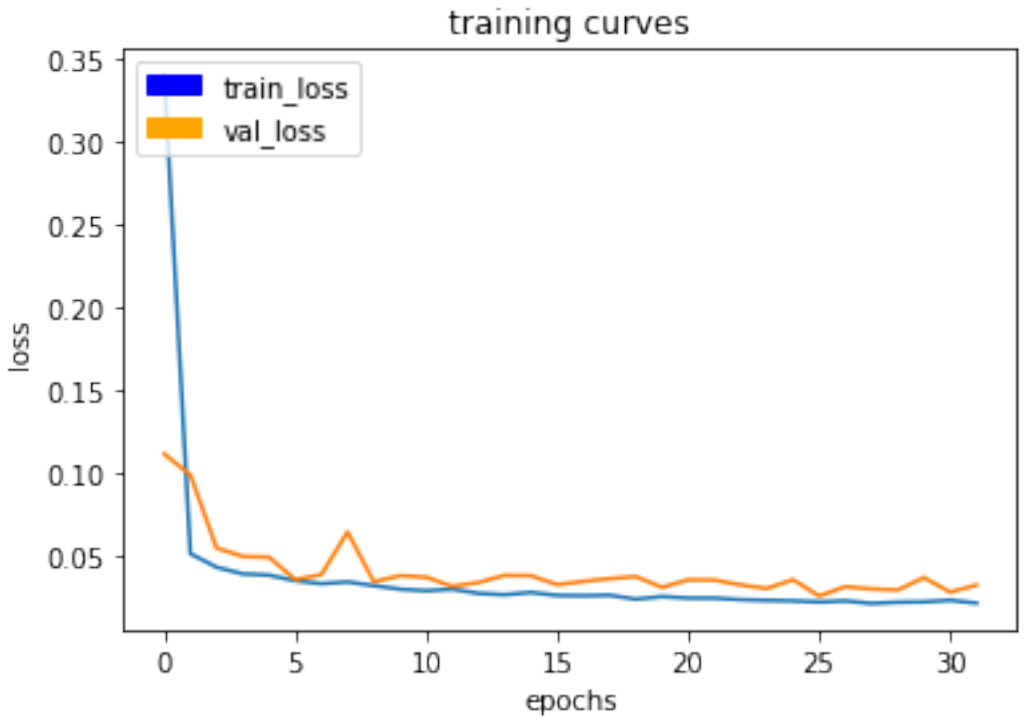

```
200/200 [==============================] - 97s - loss: 0.0221 - val_loss: 0.0295
Epoch 30/100
199/200 [============================>.] - ETA: 0s - loss: 0.0228
```



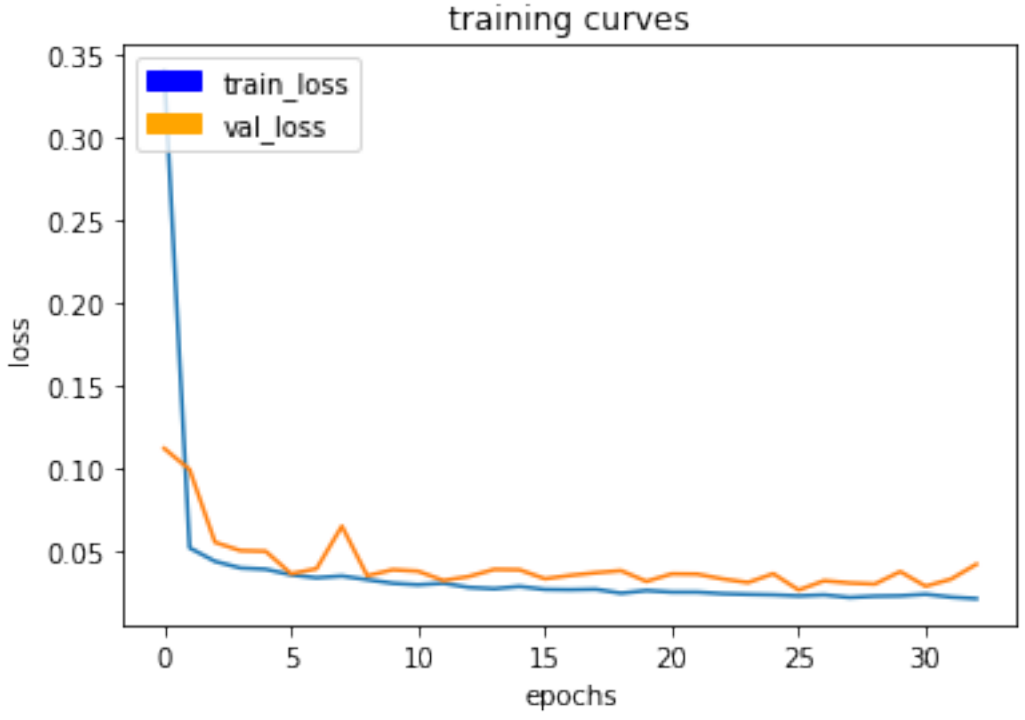
```
200/200 [==============================] - 96s - loss: 0.0228 - val_loss: 0.0370
Epoch 31/100
199/200 [============================>.] - ETA: 0s - loss: 0.0234
```



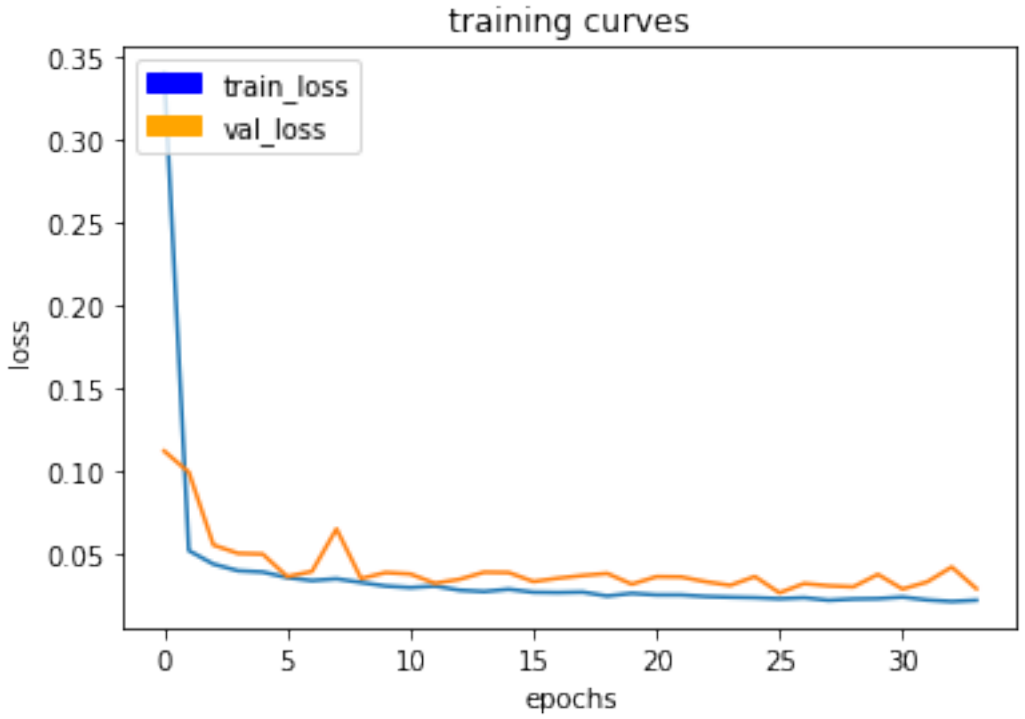

```
200/200 [==============================] - 97s - loss: 0.0233 - val_loss: 0.0281
Epoch 32/100
199/200 [============================>.] - ETA: 0s - loss: 0.0215
```



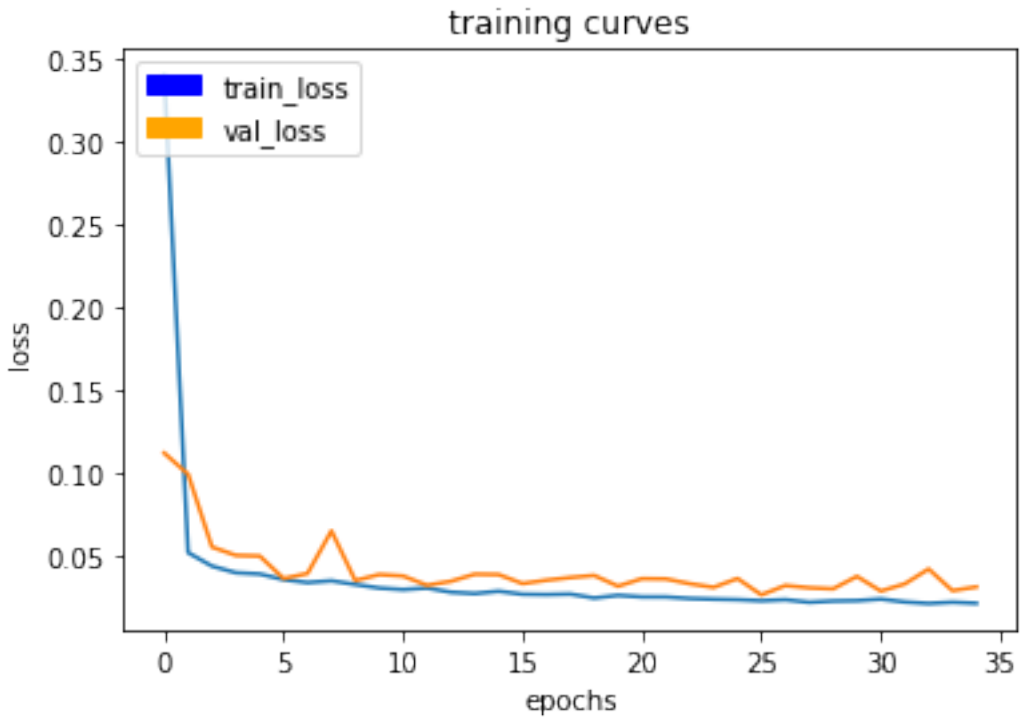

```
200/200 [==============================] - 96s - loss: 0.0215 - val_loss: 0.0324
Epoch 33/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0203
```



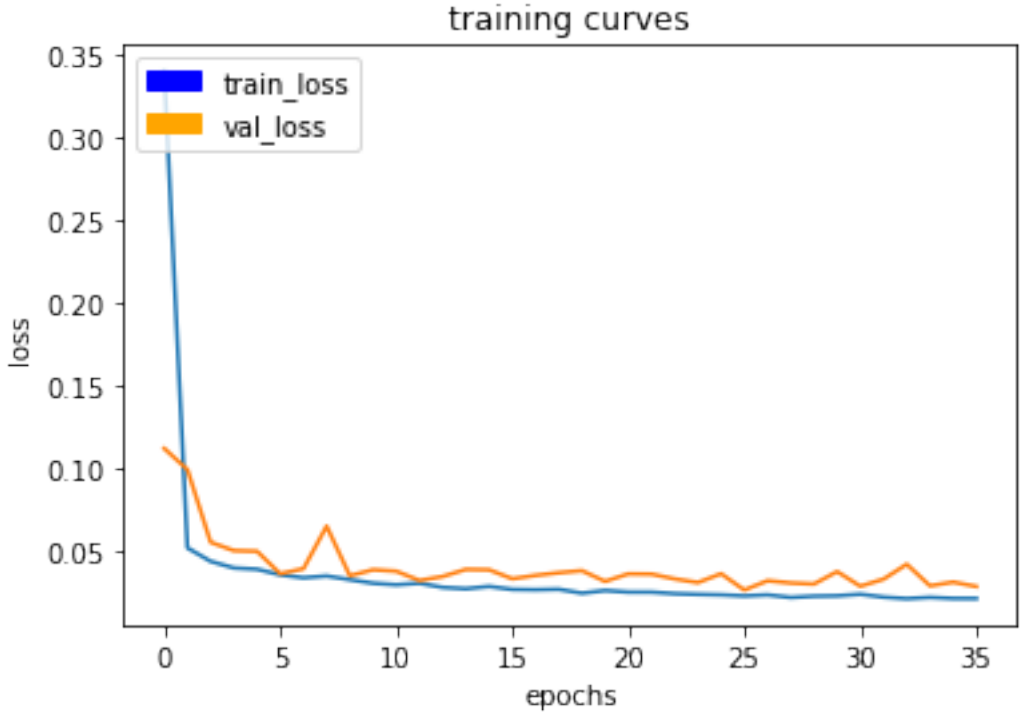
```
200/200 [==============================] - 97s - loss: 0.0206 - val_loss: 0.0414
Epoch 34/100
199/200 [============================>.] - ETA: 0s - loss: 0.0214
```



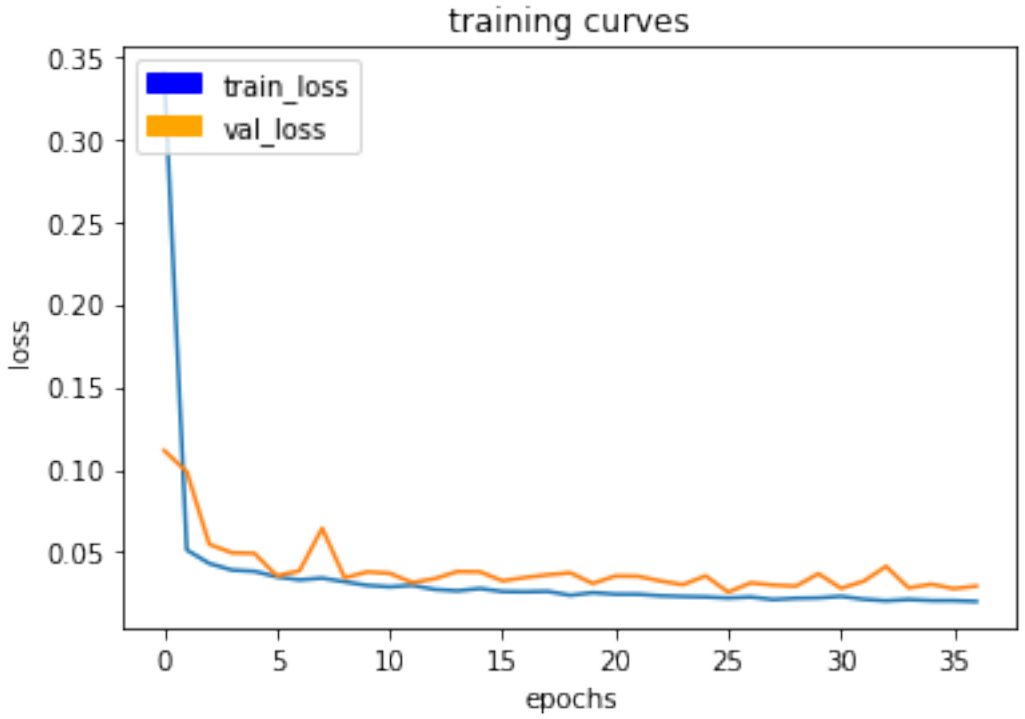

```
200/200 [==============================] - 96s - loss: 0.0215 - val_loss: 0.0284
Epoch 35/100
199/200 [============================>.] - ETA: 0s - loss: 0.0208
```



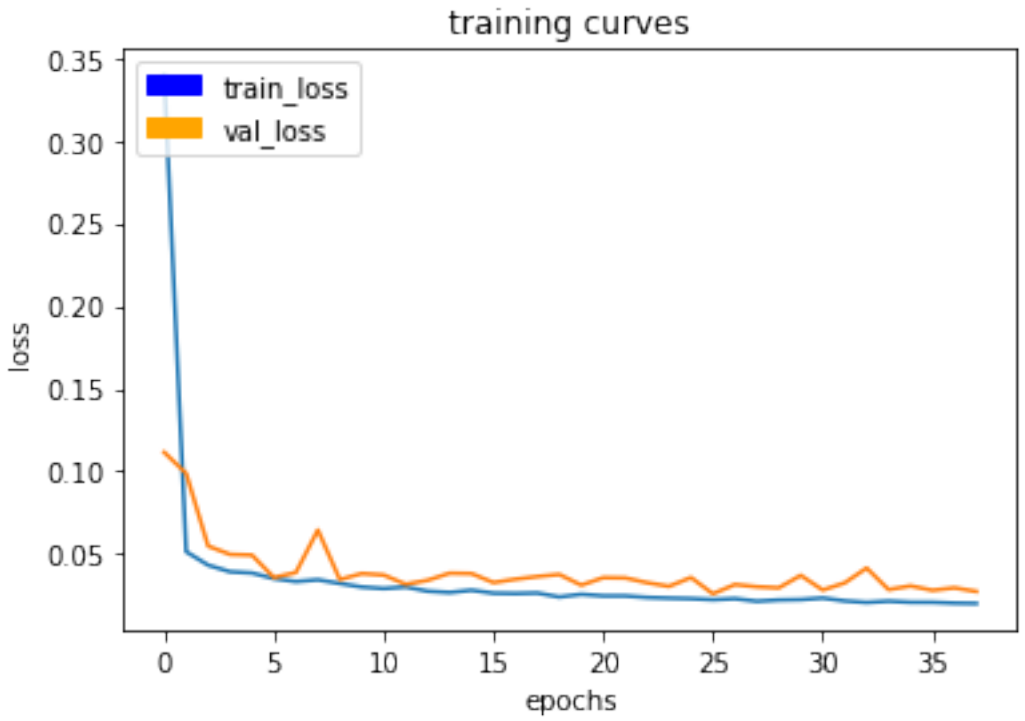

```
200/200 [==============================] - 97s - loss: 0.0208 - val_loss: 0.0306
Epoch 36/100
199/200 [============================>.] - ETA: 0s - loss: 0.0207
```



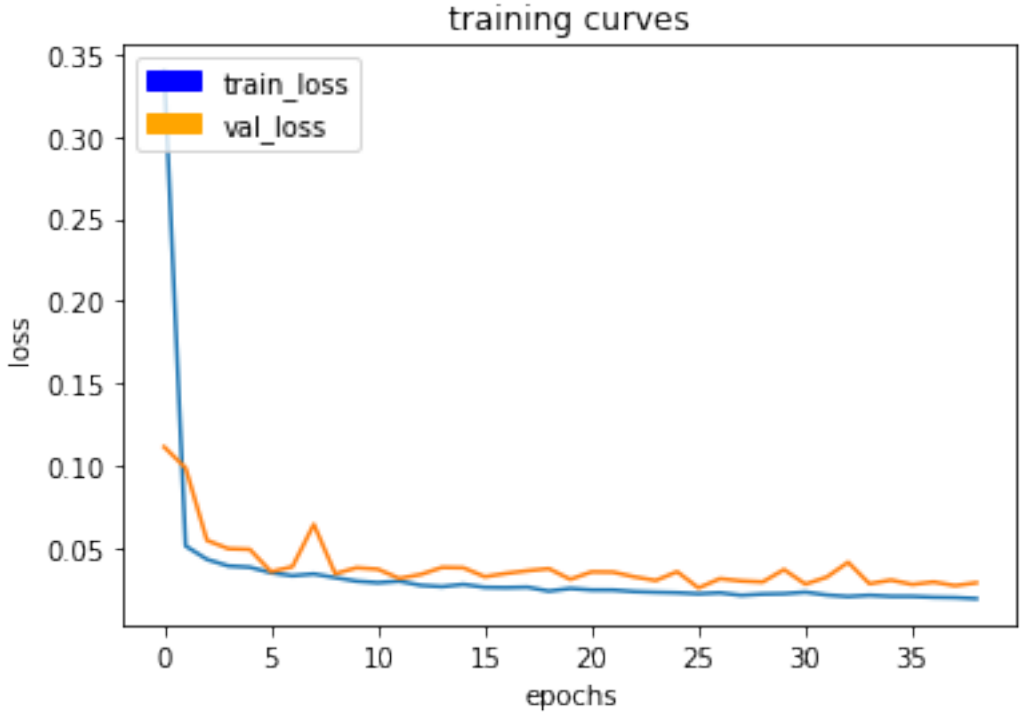
```
200/200 [==============================] - 96s - loss: 0.0207 - val_loss: 0.0280
Epoch 37/100
199/200 [============================>.] - ETA: 0s - loss: 0.0204
```



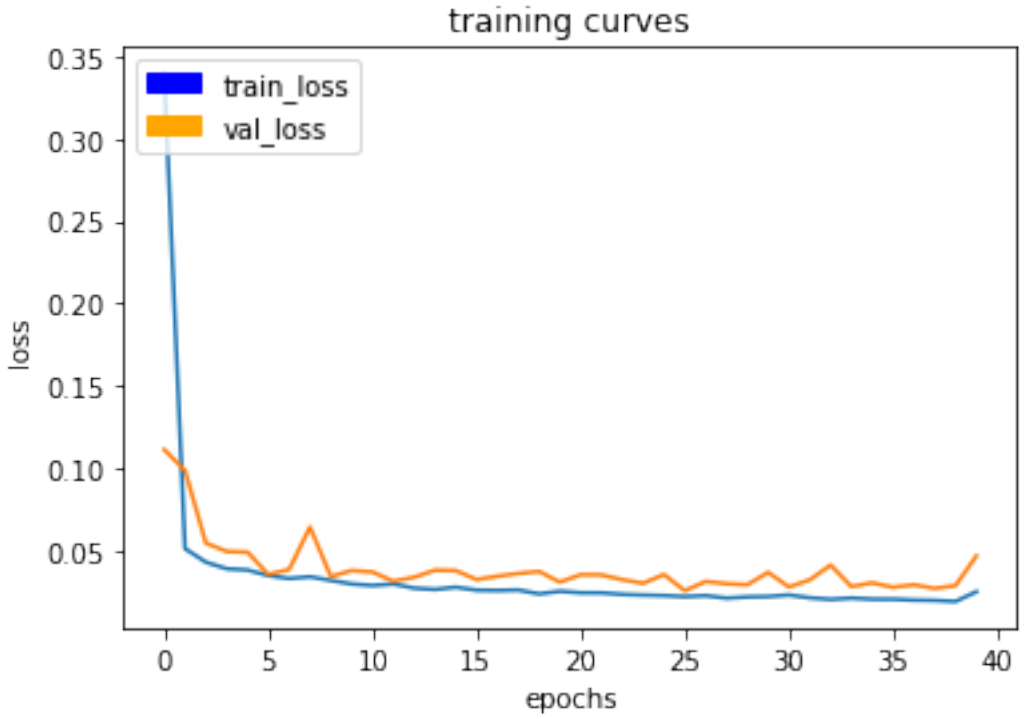

```
200/200 [==============================] - 97s - loss: 0.0204 - val_loss: 0.0295
Epoch 38/100
199/200 [============================>.] - ETA: 0s - loss: 0.0201
```



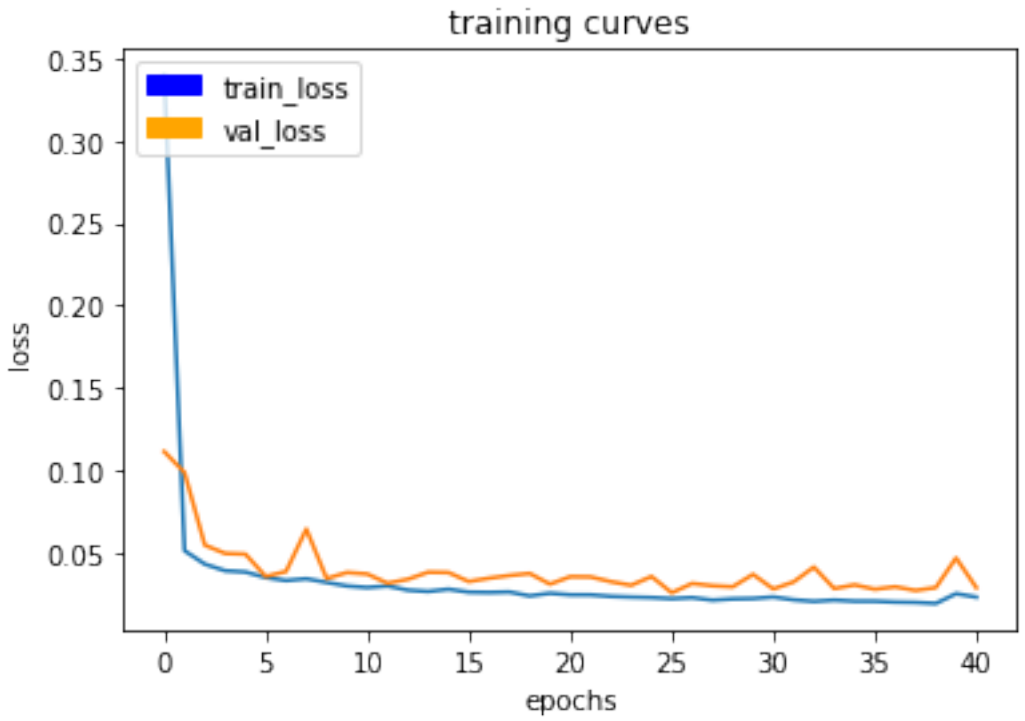

```
200/200 [==============================] - 96s - loss: 0.0200 - val_loss: 0.0272
Epoch 39/100
199/200 [============================>.] - ETA: 0s - loss: 0.0193
```



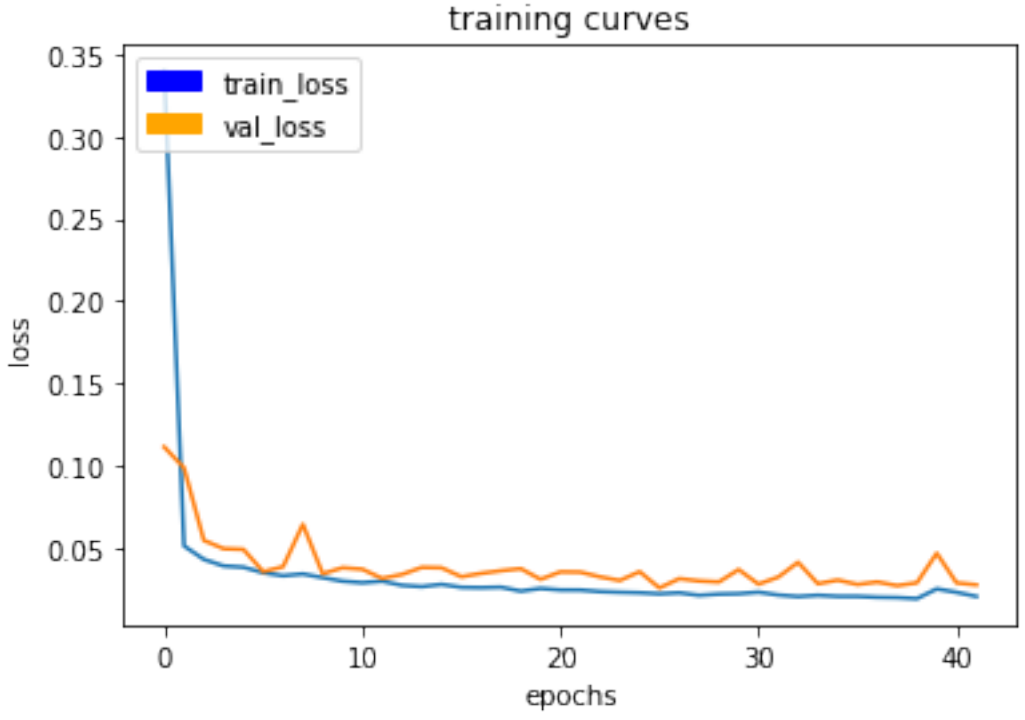
```
200/200 [==============================] - 96s - loss: 0.0193 - val_loss: 0.0290
Epoch 40/100
199/200 [============================>.] - ETA: 0s - loss: 0.0264
```



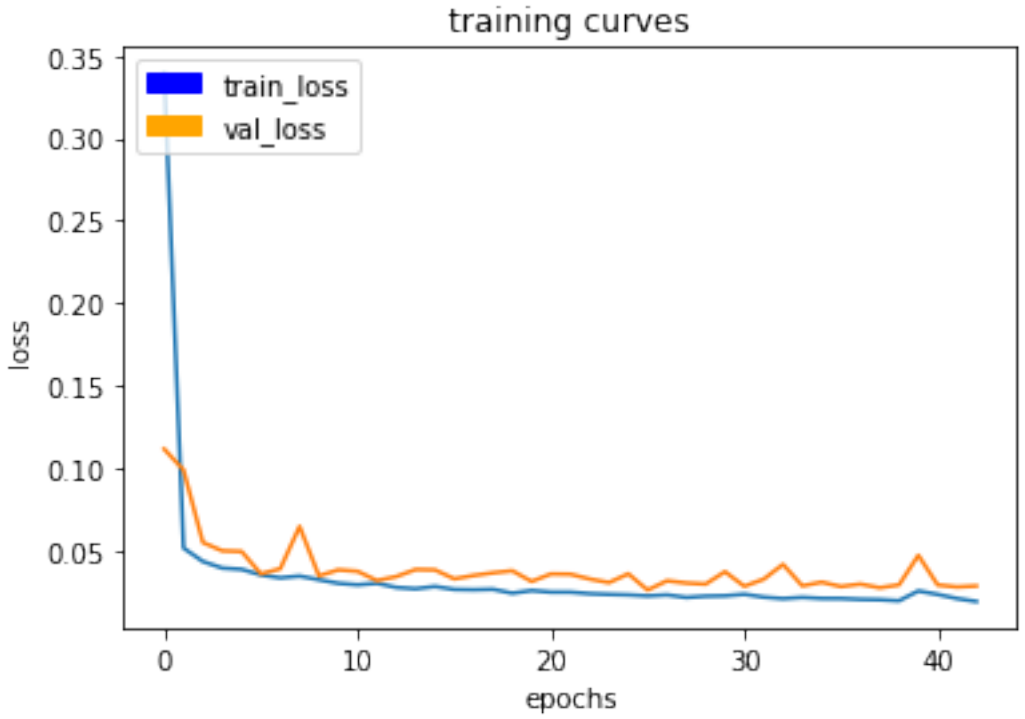

```
200/200 [==============================] - 96s - loss: 0.0263 - val_loss: 0.0469
Epoch 41/100
199/200 [============================>.] - ETA: 0s - loss: 0.0233
```



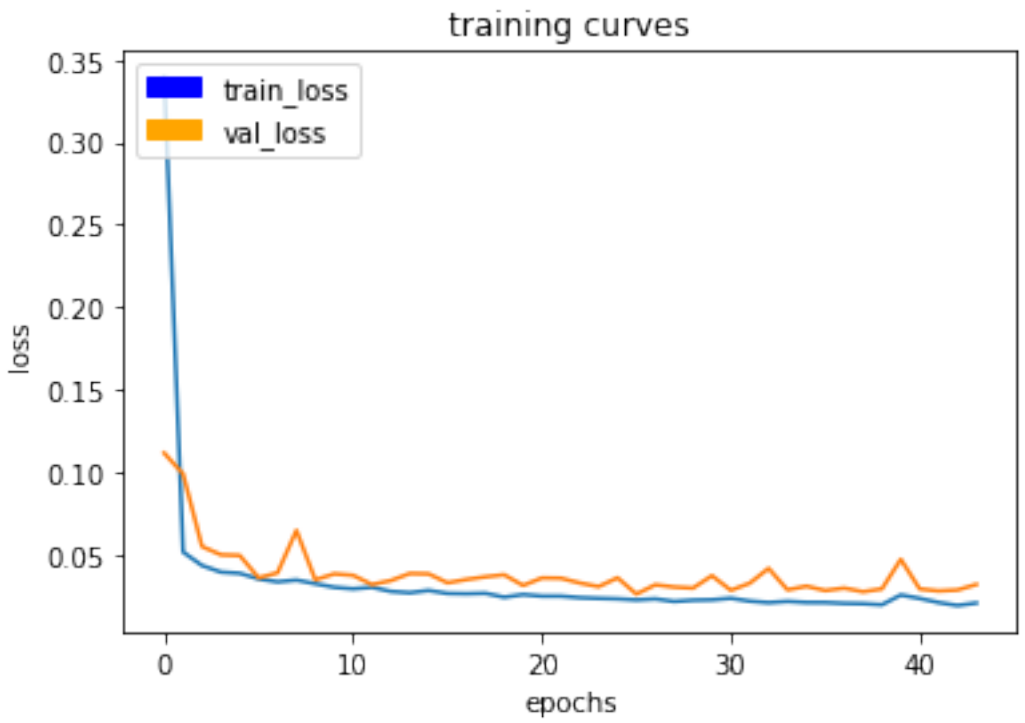

```
200/200 [==============================] - 96s - loss: 0.0232 - val_loss: 0.0289
Epoch 42/100
199/200 [============================>.] - ETA: 0s - loss: 0.0208
```



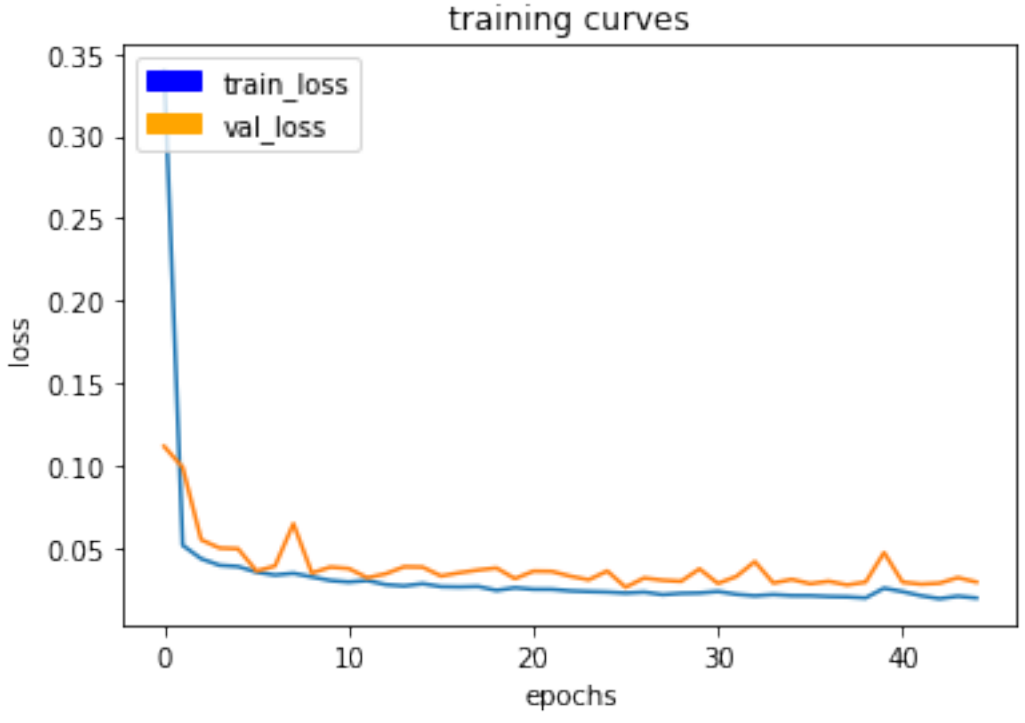
```
200/200 [==============================] - 97s - loss: 0.0208 - val_loss: 0.0277
Epoch 43/100
199/200 [============================>.] - ETA: 0s - loss: 0.0189
```



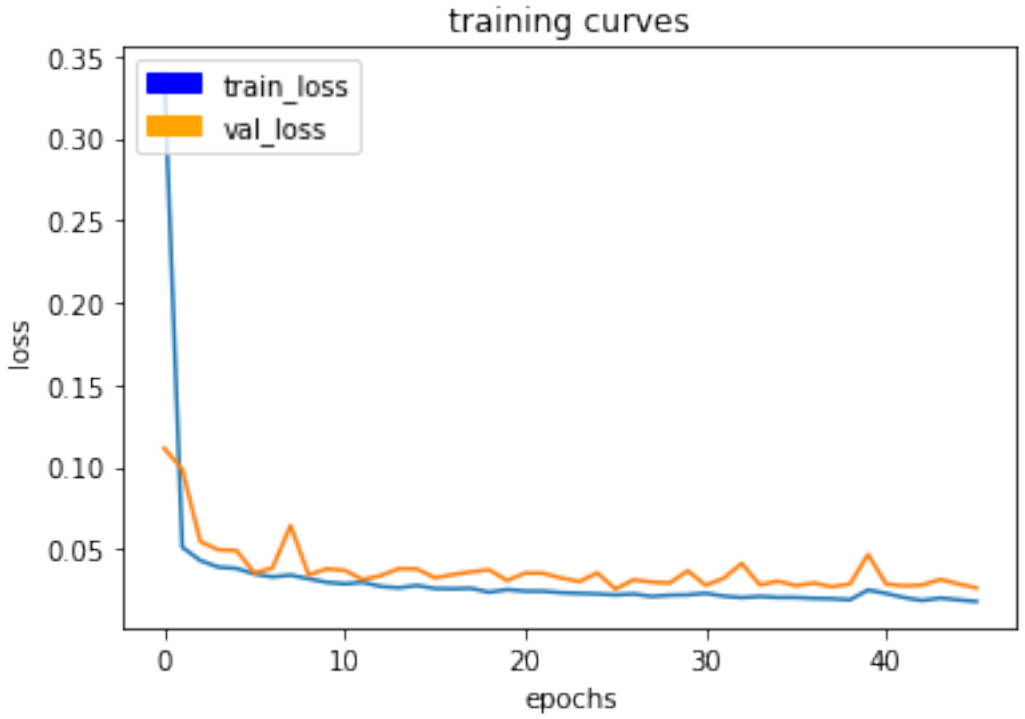

```
200/200 [==============================] - 96s - loss: 0.0189 - val_loss: 0.0283
Epoch 44/100
199/200 [============================>.] - ETA: 0s - loss: 0.0204
```



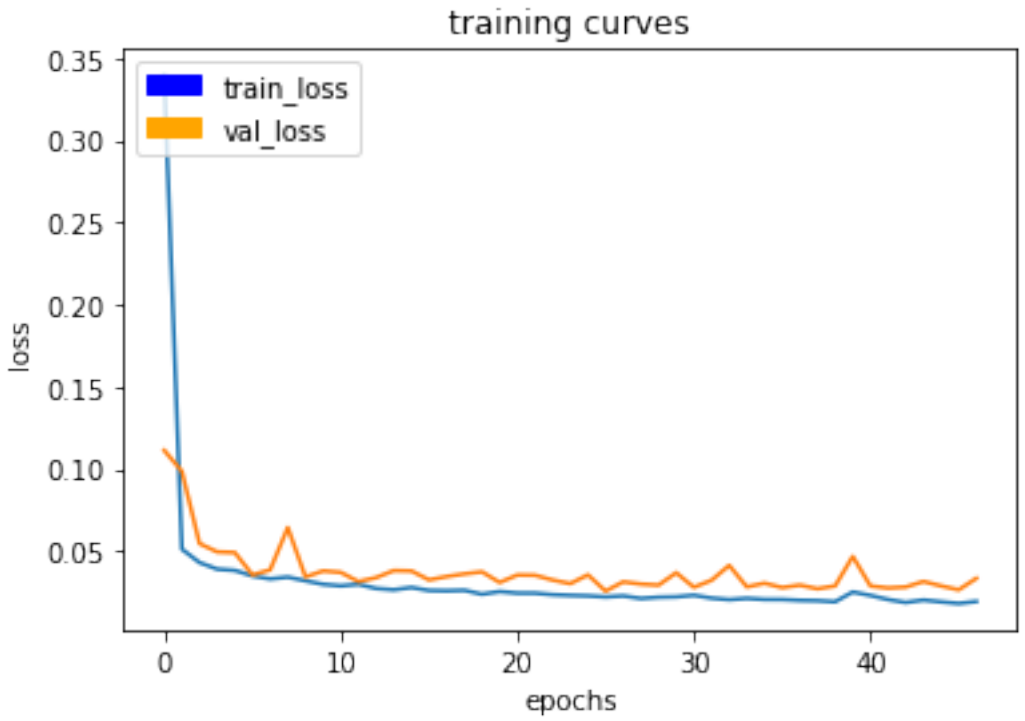

```
200/200 [==============================] - 96s - loss: 0.0204 - val_loss: 0.0317
Epoch 45/100
199/200 [============================>.] - ETA: 0s - loss: 0.0191
```



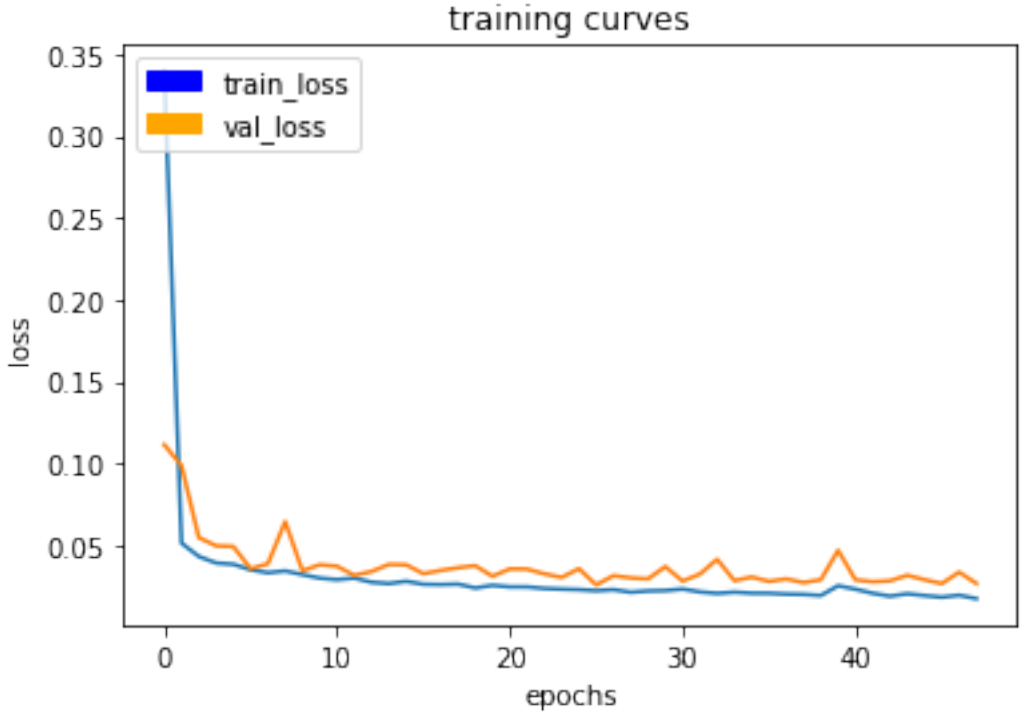
```
200/200 [==============================] - 96s - loss: 0.0194 - val_loss: 0.0290
Epoch 46/100
199/200 [============================>.] - ETA: 0s - loss: 0.0183
```



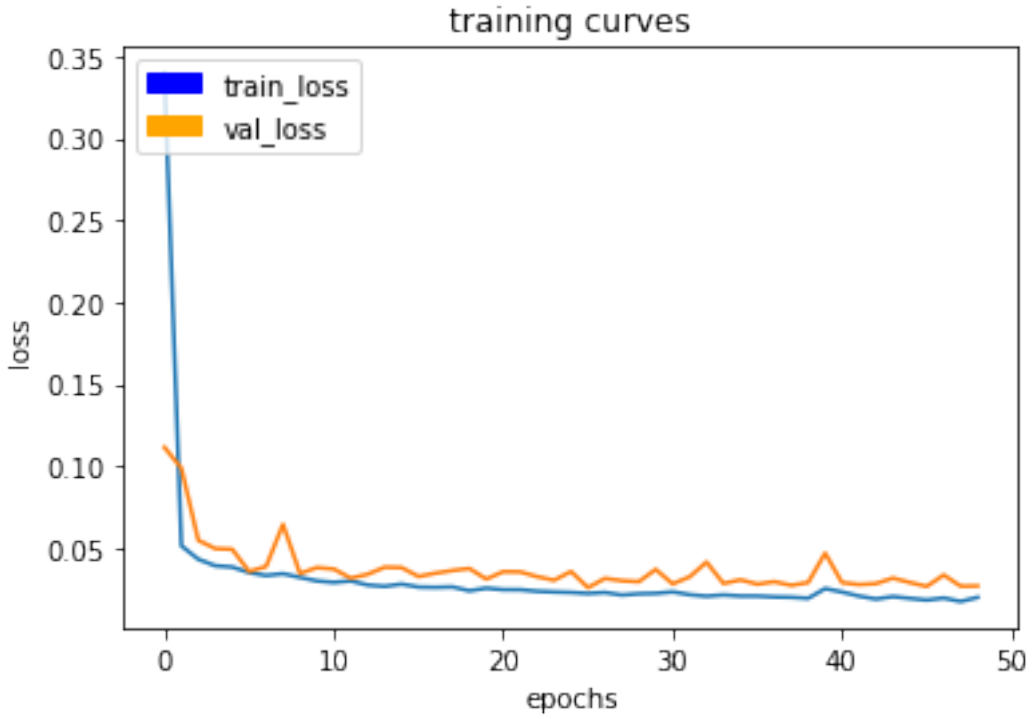

```
200/200 [==============================] - 97s - loss: 0.0183 - val_loss: 0.0266
Epoch 47/100
199/200 [============================>.] - ETA: 0s - loss: 0.0195
```



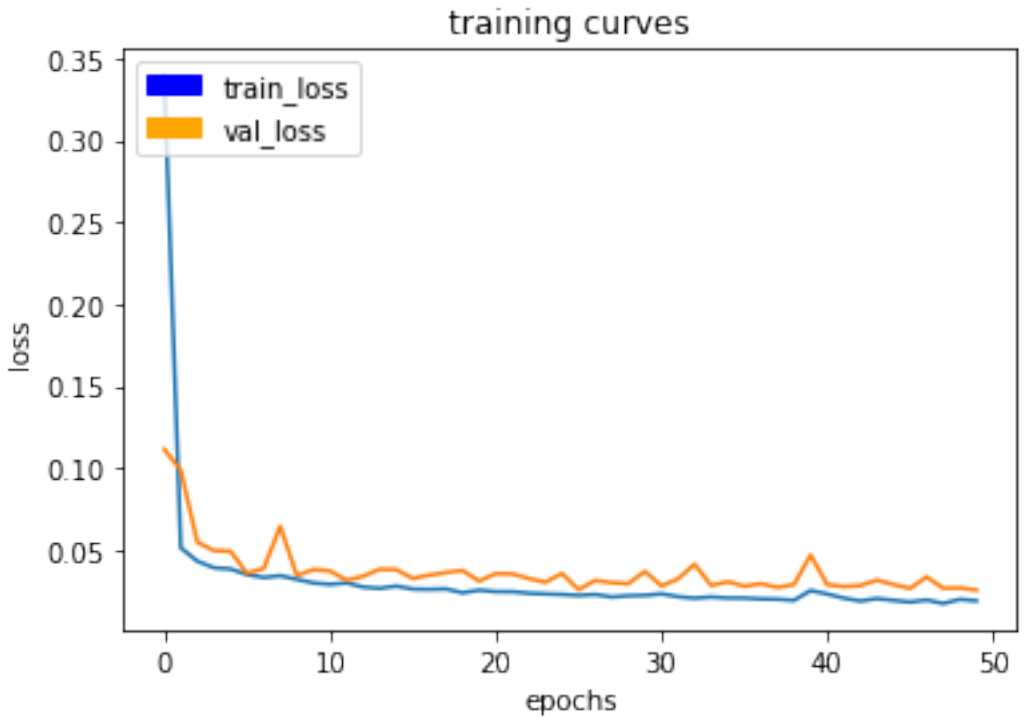

```
200/200 [==============================] - 96s - loss: 0.0195 - val_loss: 0.0336
Epoch 48/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0175
```



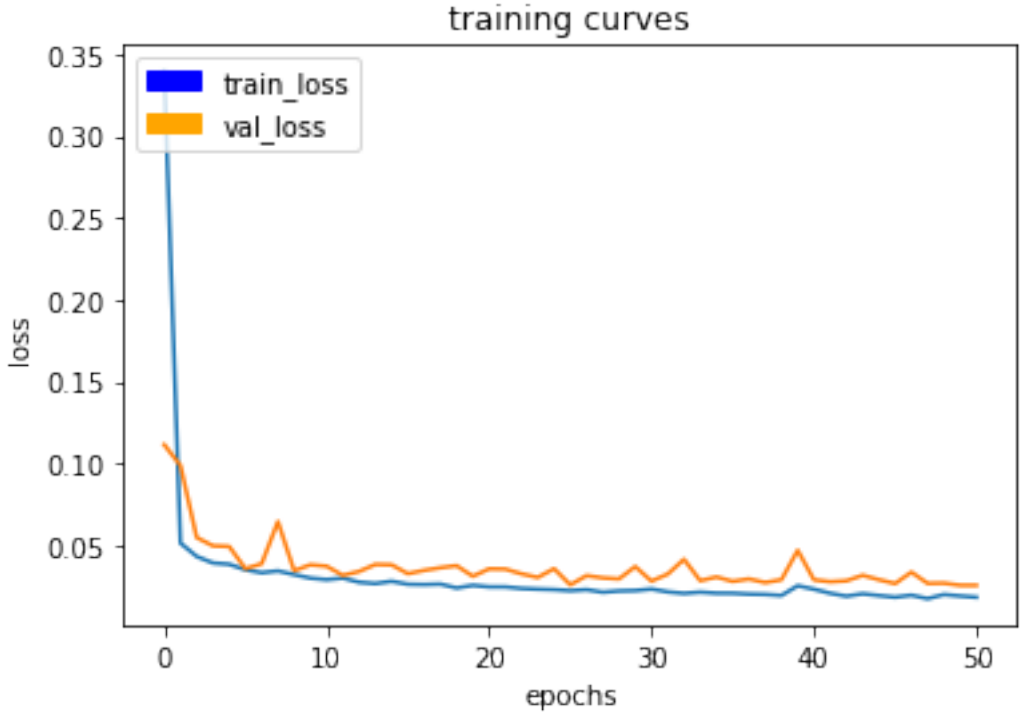
```
200/200 [==============================] - 97s - loss: 0.0174 - val_loss: 0.0267
Epoch 49/100
199/200 [============================>.] - ETA: 0s - loss: 0.0200
```



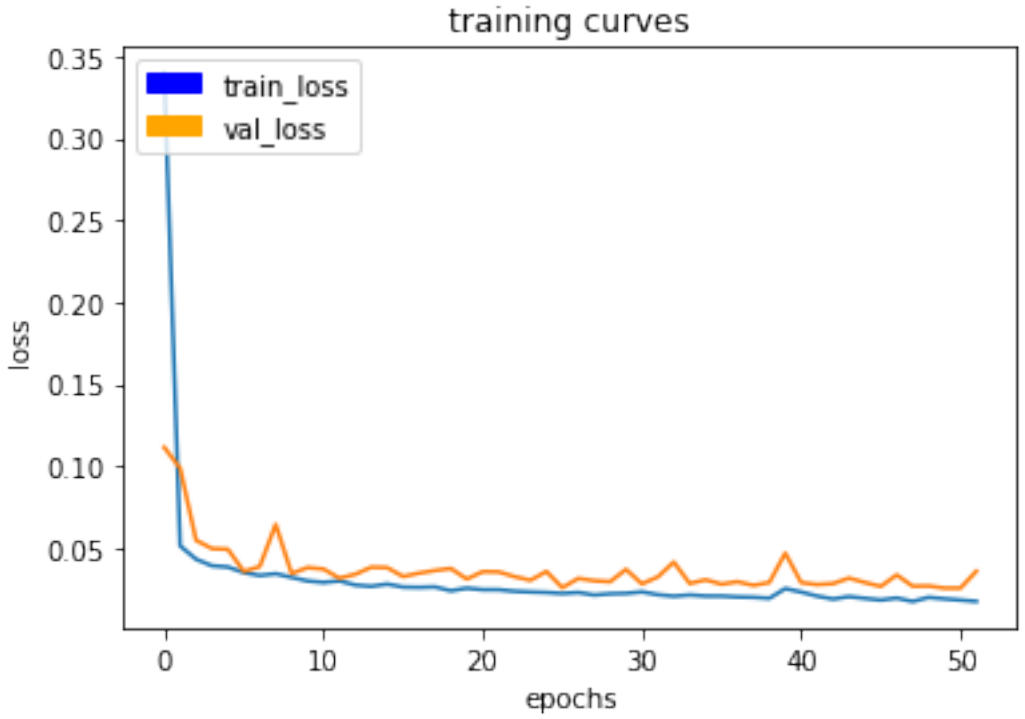

```
200/200 [==============================] - 96s - loss: 0.0200 - val_loss: 0.0269
Epoch 50/100
199/200 [============================>.] - ETA: 0s - loss: 0.0190
```



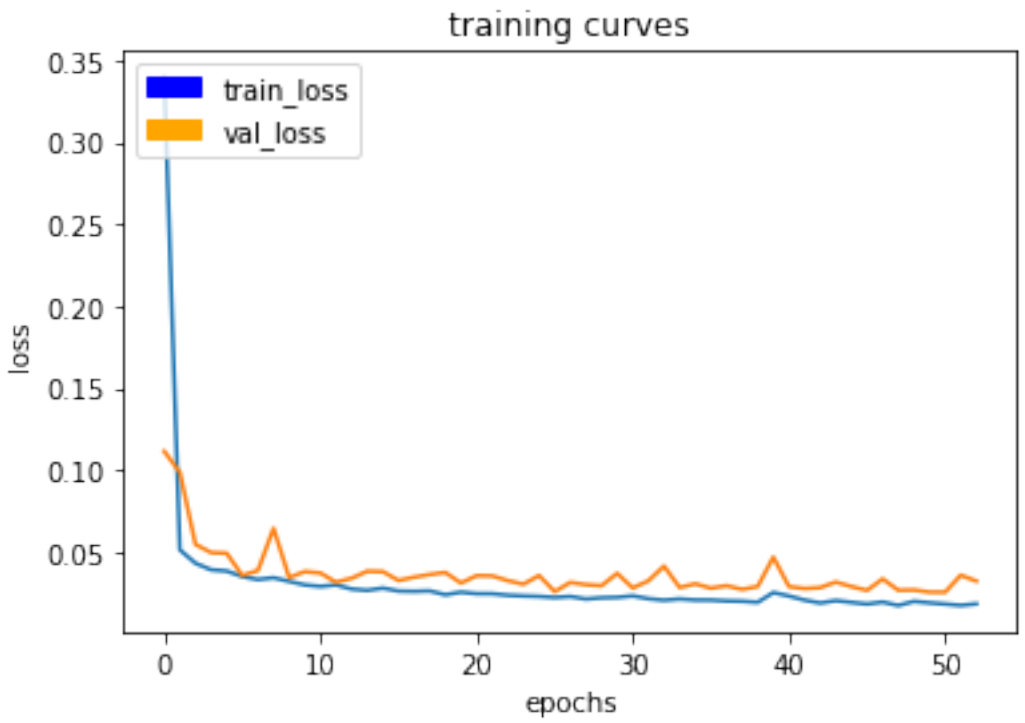

```
200/200 [==============================] - 96s - loss: 0.0190 - val_loss: 0.0256
Epoch 51/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0183
```



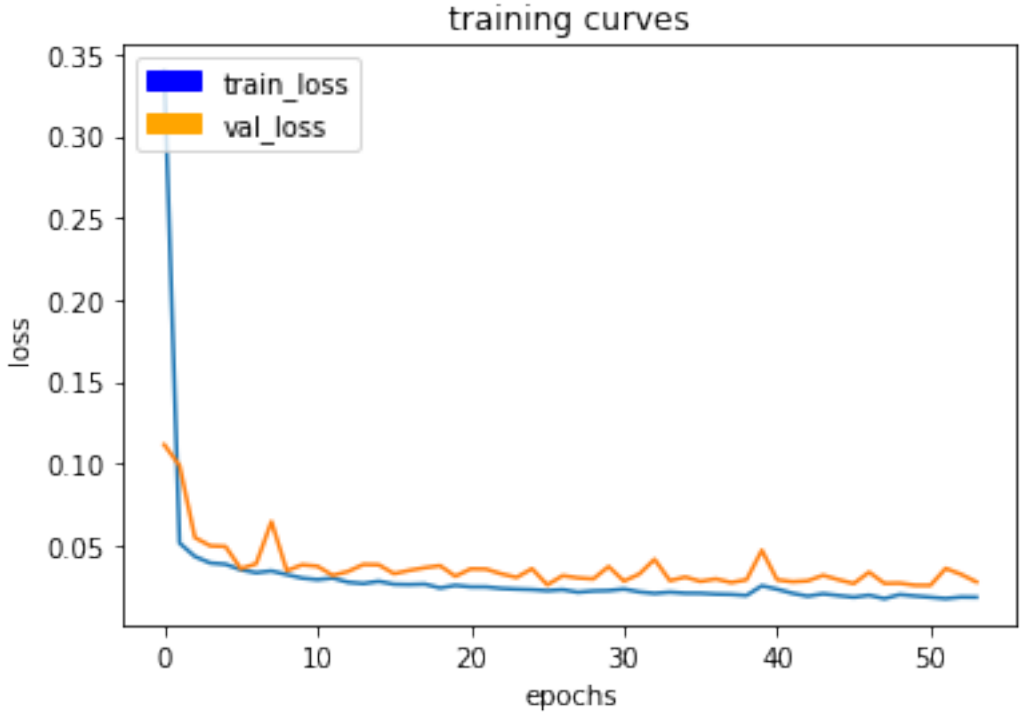
```
200/200 [==============================] - 96s - loss: 0.0183 - val_loss: 0.0255
Epoch 52/100
199/200 [============================>.] - ETA: 0s - loss: 0.0174
```



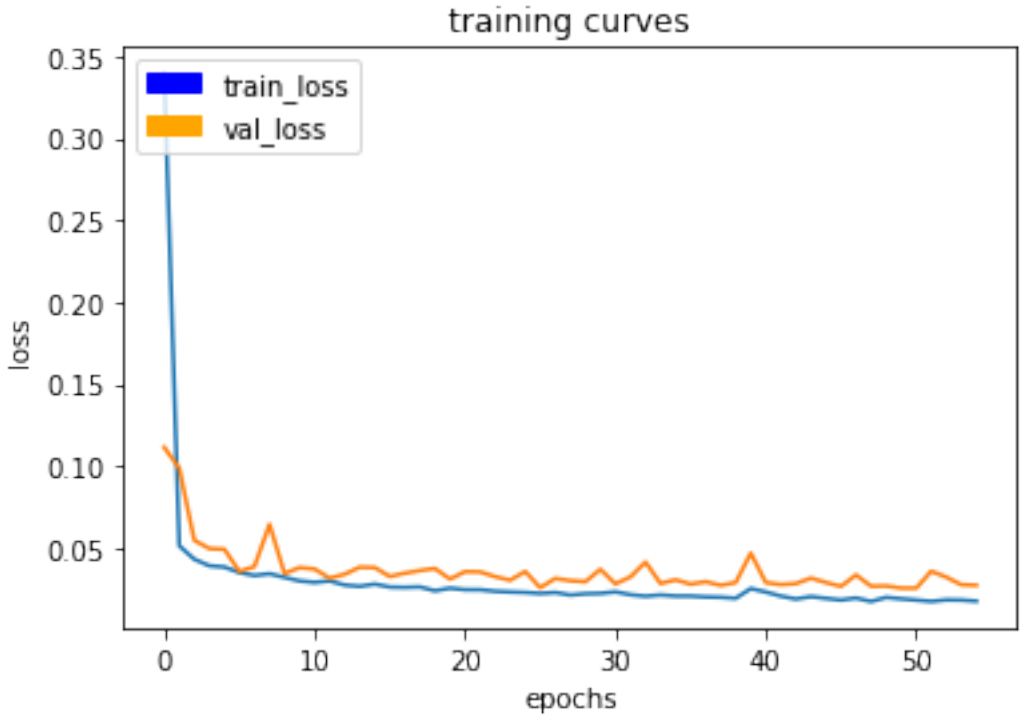

```
200/200 [==============================] - 96s - loss: 0.0174 - val_loss: 0.0359
Epoch 53/100
199/200 [============================>.] - ETA: 0s - loss: 0.0184
```



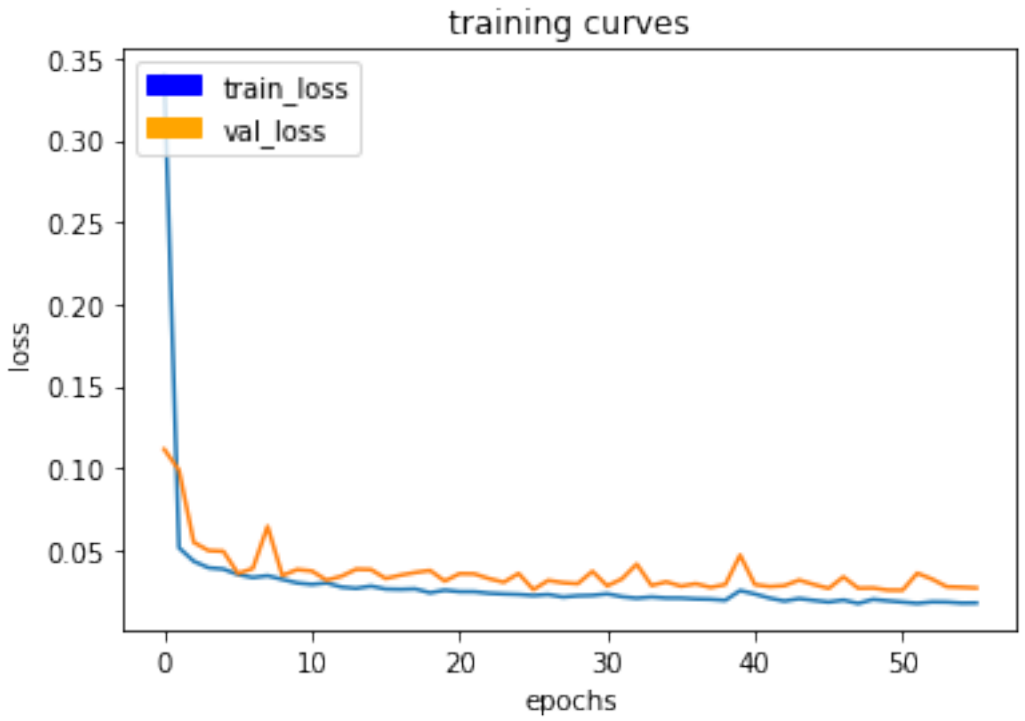

```
200/200 [==============================] - 97s - loss: 0.0184 - val_loss: 0.0323
Epoch 54/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0183
```



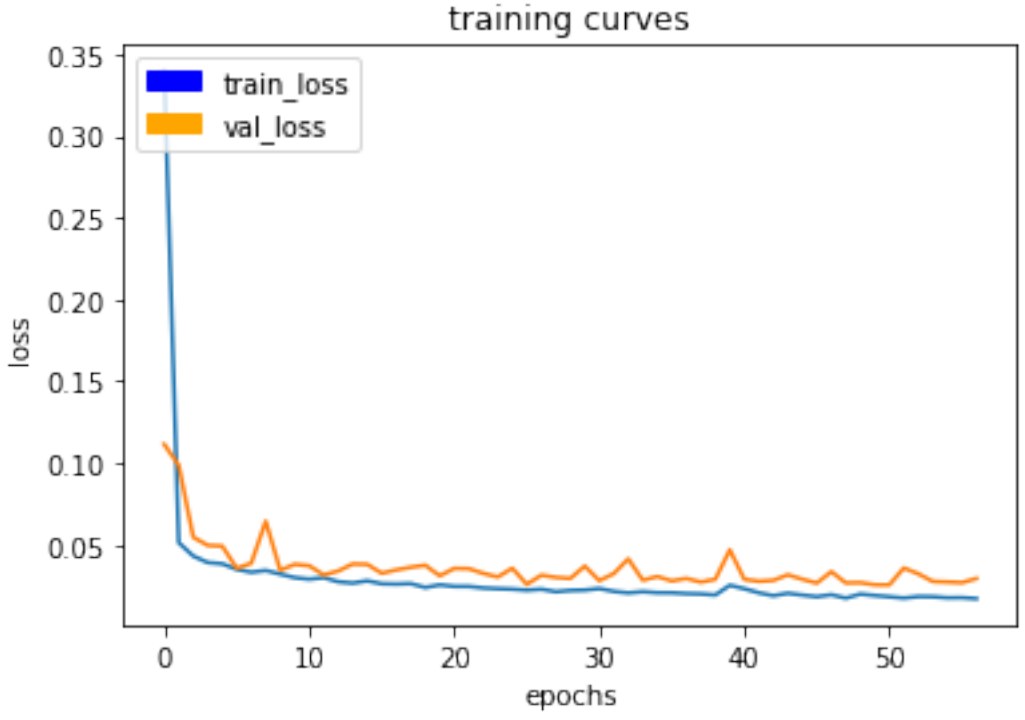
```
200/200 [==============================] - 96s - loss: 0.0183 - val_loss: 0.0276
Epoch 55/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0176
```



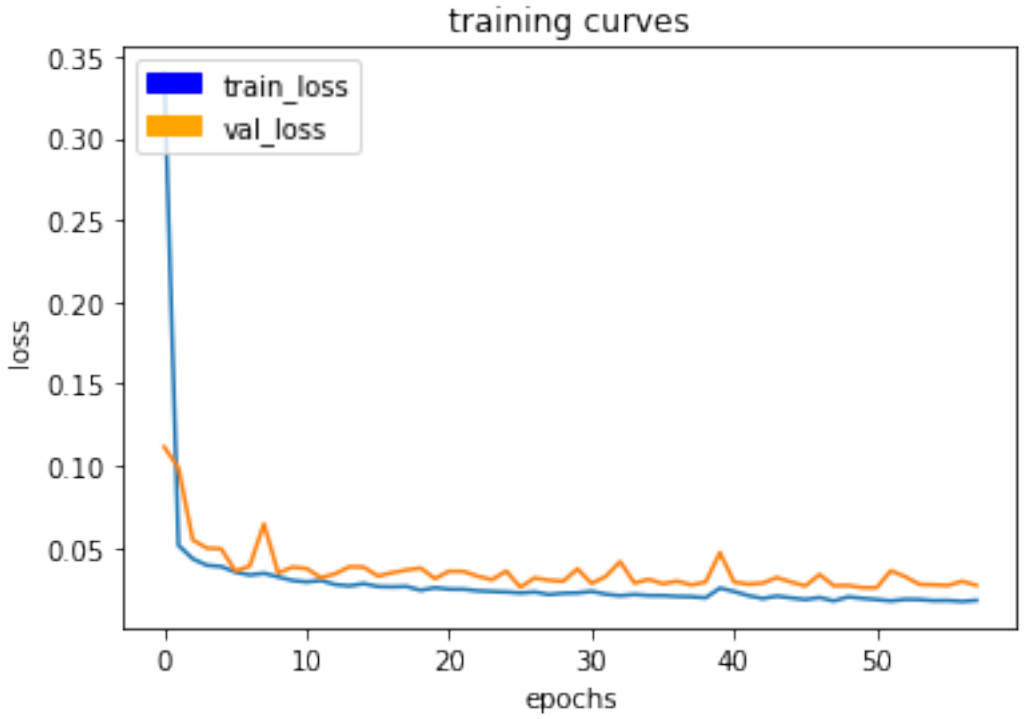

```
200/200 [==============================] - 97s - loss: 0.0176 - val_loss: 0.0273
Epoch 56/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0177
```



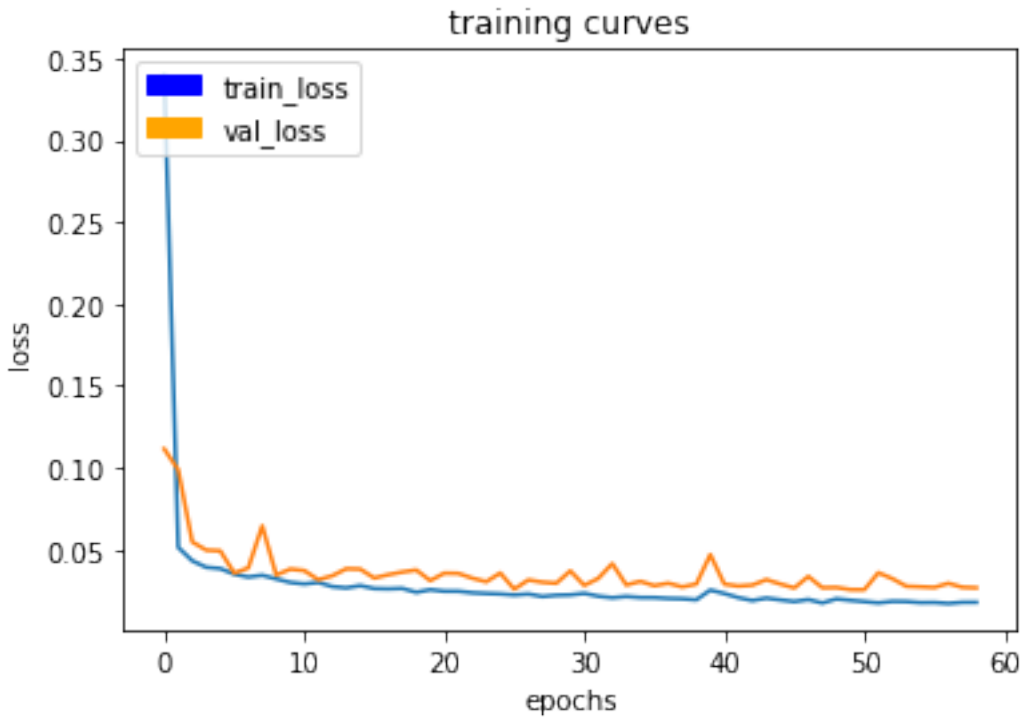

```
200/200 [==============================] - 96s - loss: 0.0177 - val_loss: 0.0269
Epoch 57/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0171
```



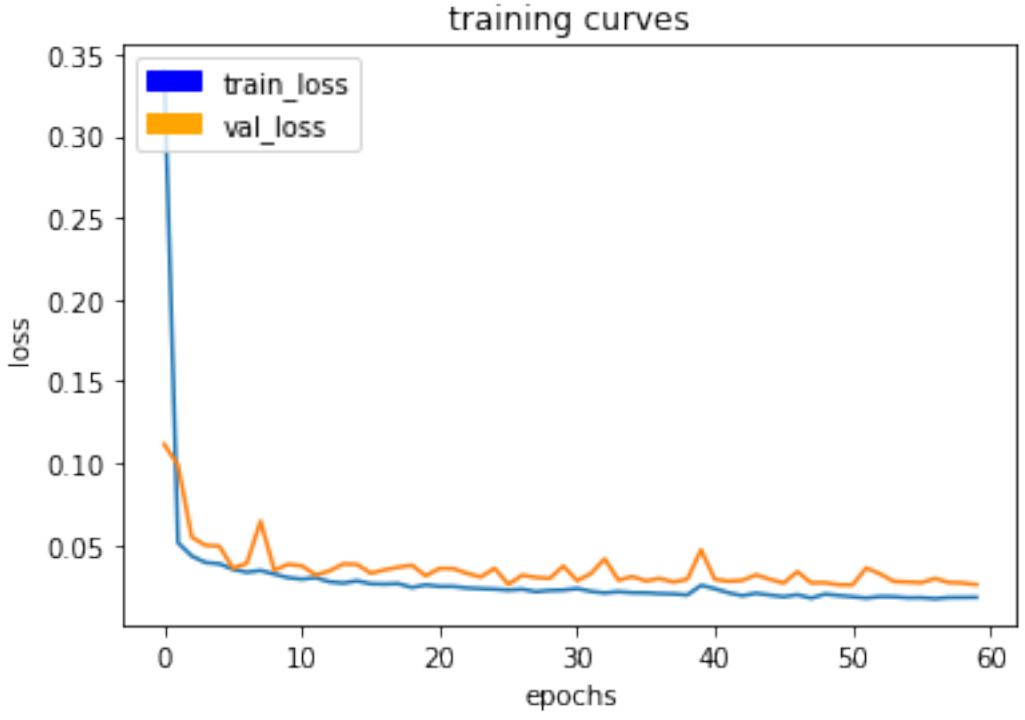
```
200/200 [==============================] - 97s - loss: 0.0171 - val_loss: 0.0294
Epoch 58/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0178
```



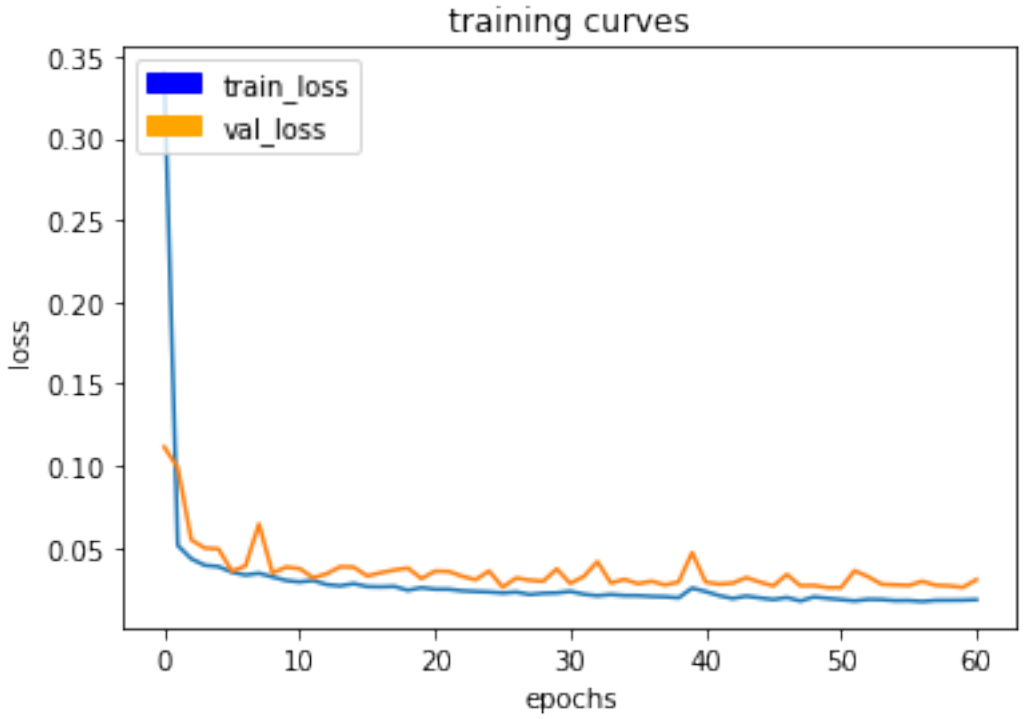

```
200/200 [==============================] - 97s - loss: 0.0178 - val_loss: 0.0270
Epoch 59/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0179
```



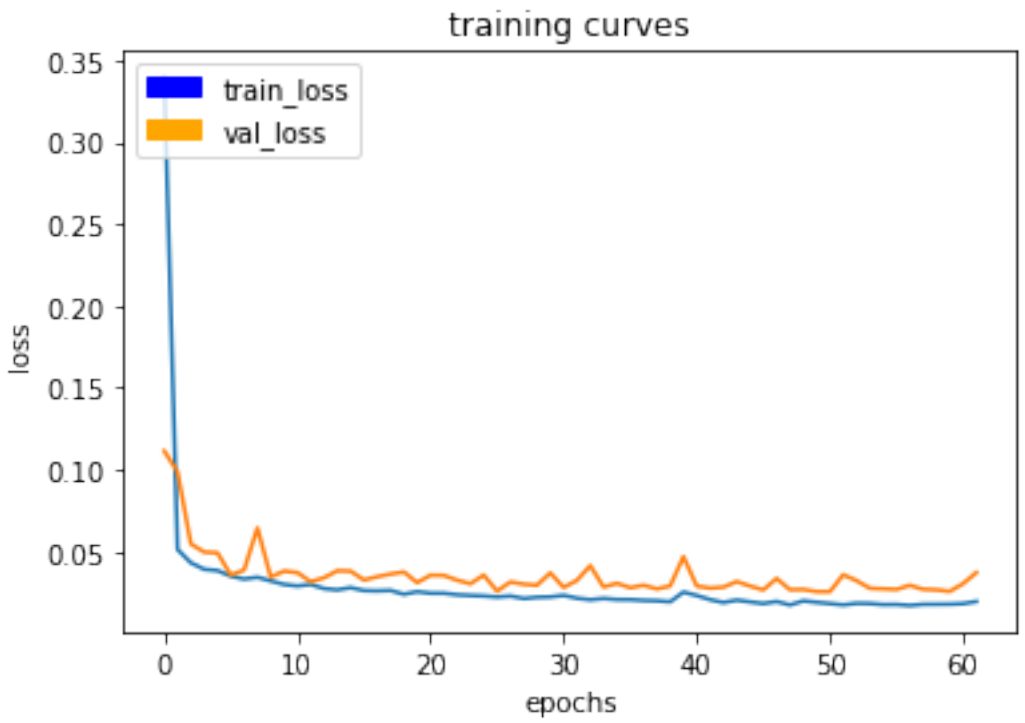

```
200/200 [==============================] - 97s - loss: 0.0179 - val_loss: 0.0267
Epoch 60/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0181
```



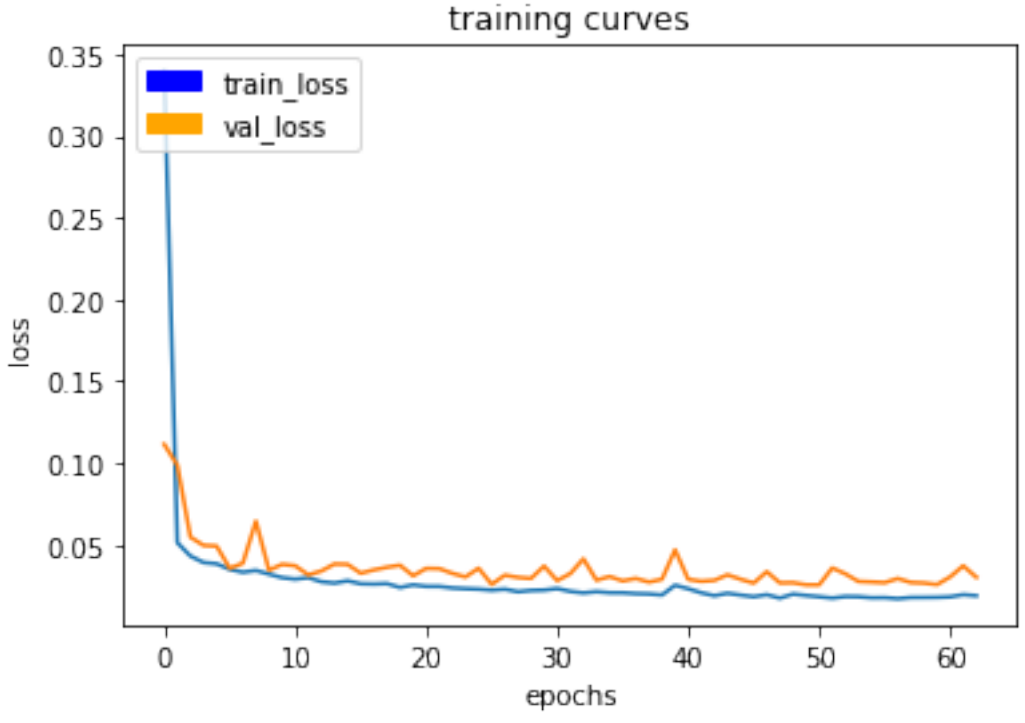
```
200/200 [==============================] - 96s - loss: 0.0181 - val_loss: 0.0257
Epoch 61/100
199/200 [============================>.] - ETA: 0s - loss: 0.0182
```



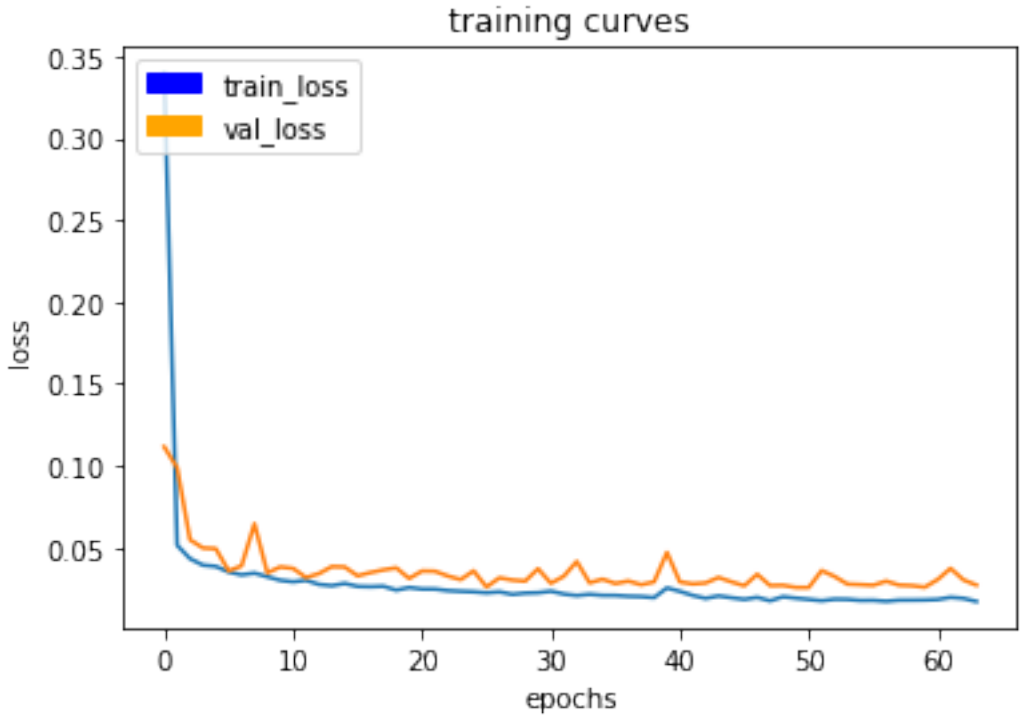

```
200/200 [==============================] - 97s - loss: 0.0182 - val_loss: 0.0304
Epoch 62/100
199/200 [============================>.] - ETA: 0s - loss: 0.0201
```



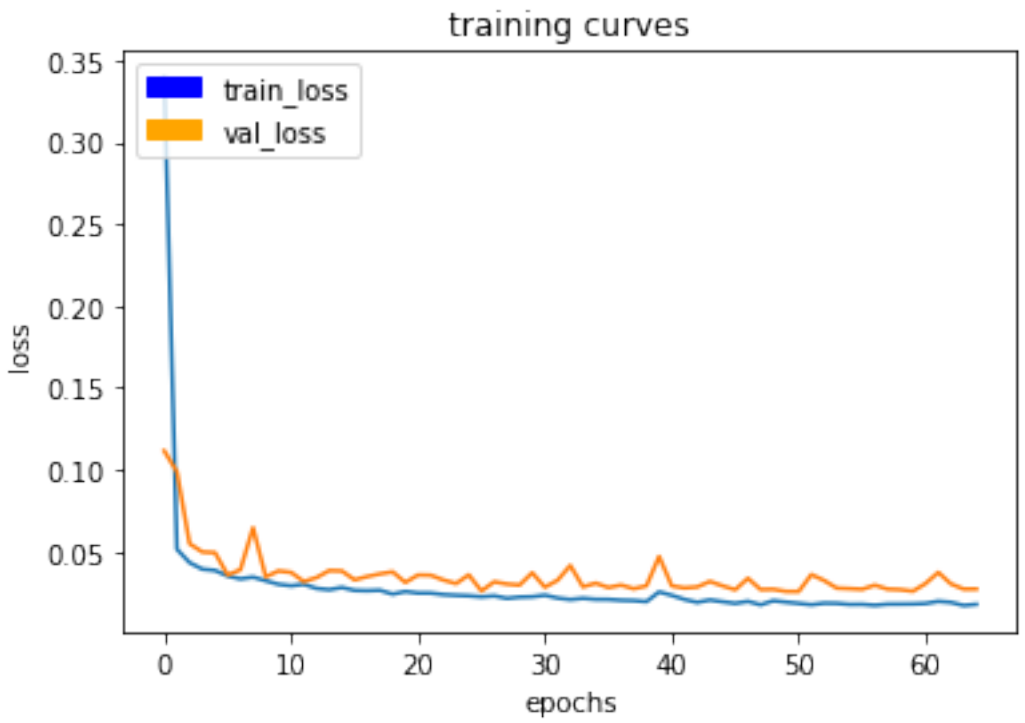

```
200/200 [==============================] - 97s - loss: 0.0201 - val_loss: 0.0372
Epoch 63/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0189
```



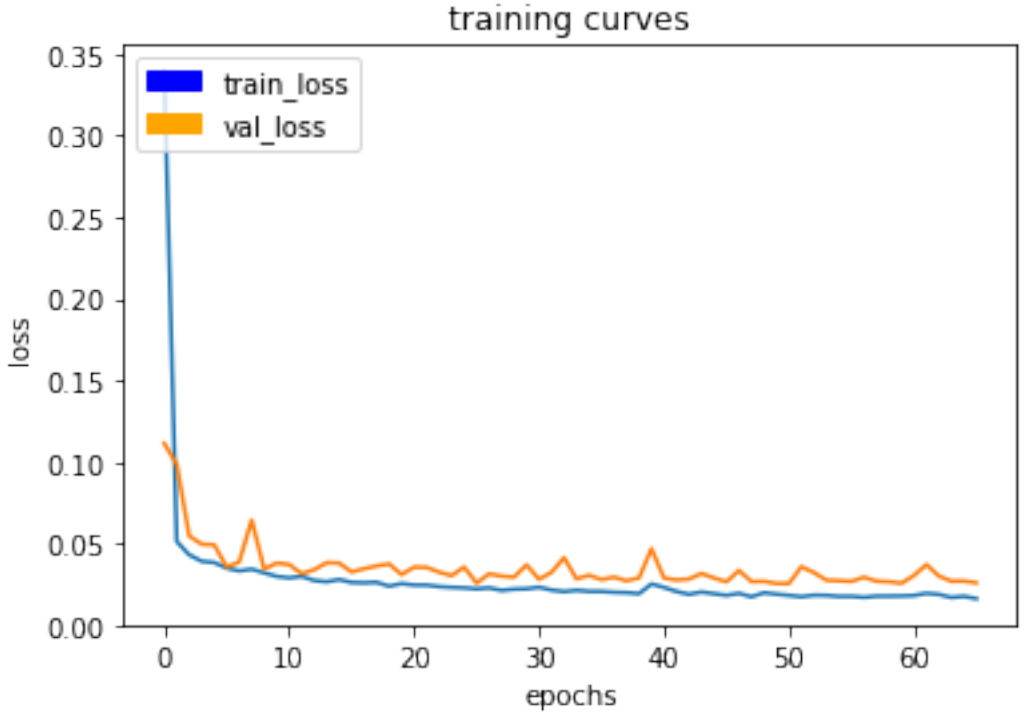
```
200/200 [==============================] - 97s - loss: 0.0189 - val_loss: 0.0302
Epoch 64/100
199/200 [============================>.] - ETA: 0s - loss: 0.0170
```



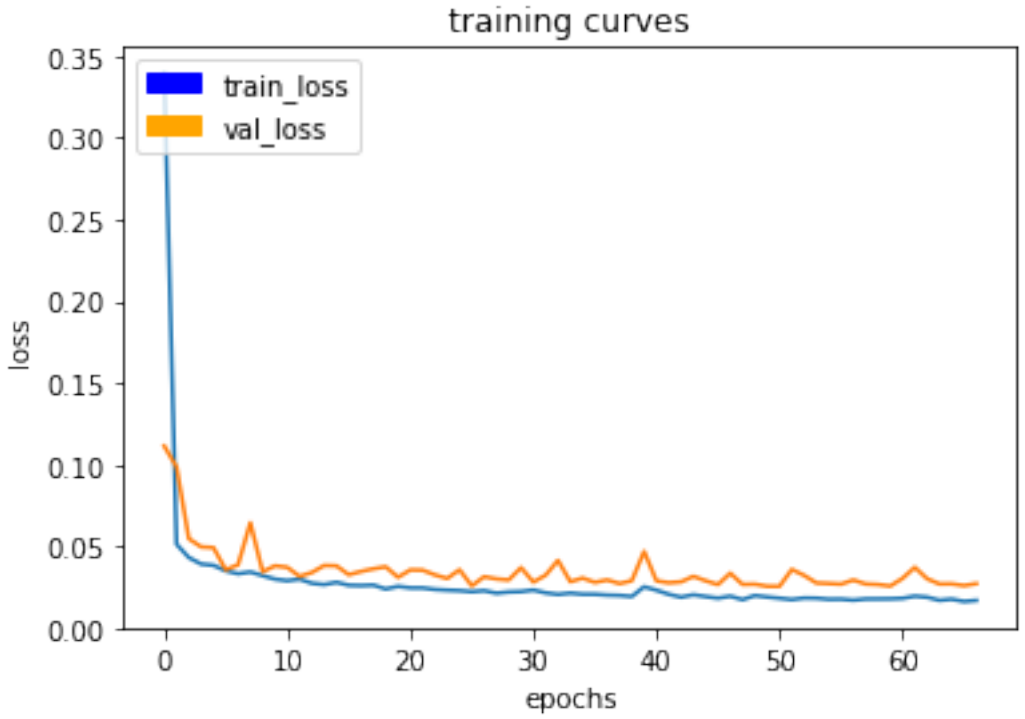

```
200/200 [==============================] - 96s - loss: 0.0170 - val_loss: 0.0270
Epoch 65/100
199/200 [============================>.] - ETA: 0s - loss: 0.0177
```



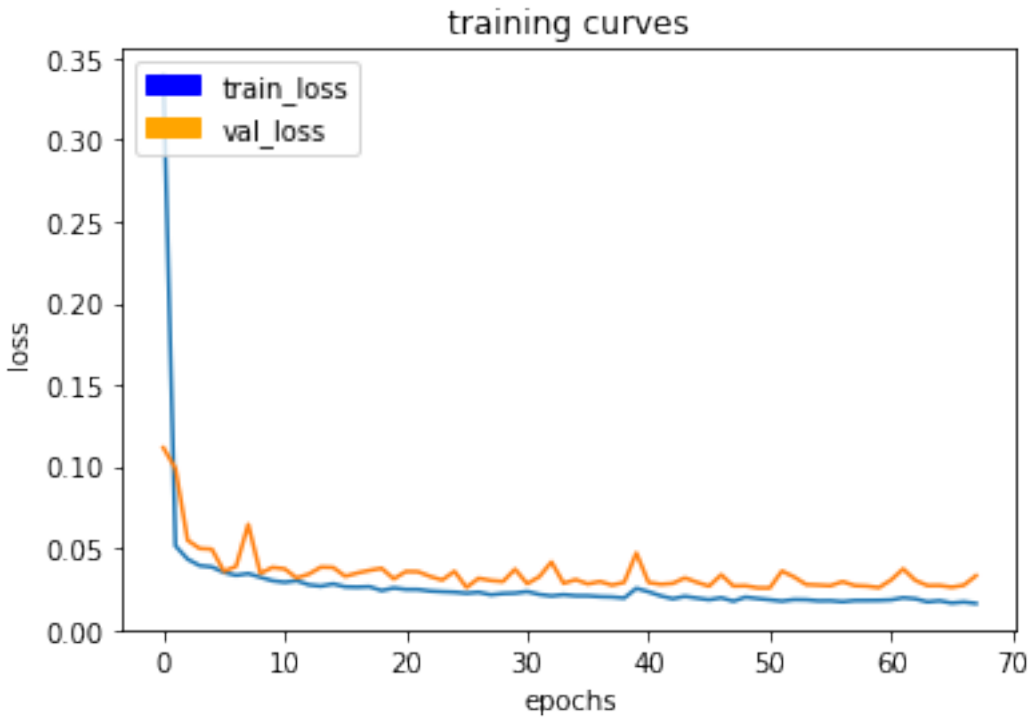

```
200/200 [==============================] - 96s - loss: 0.0177 - val_loss: 0.0270
Epoch 66/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0162
```



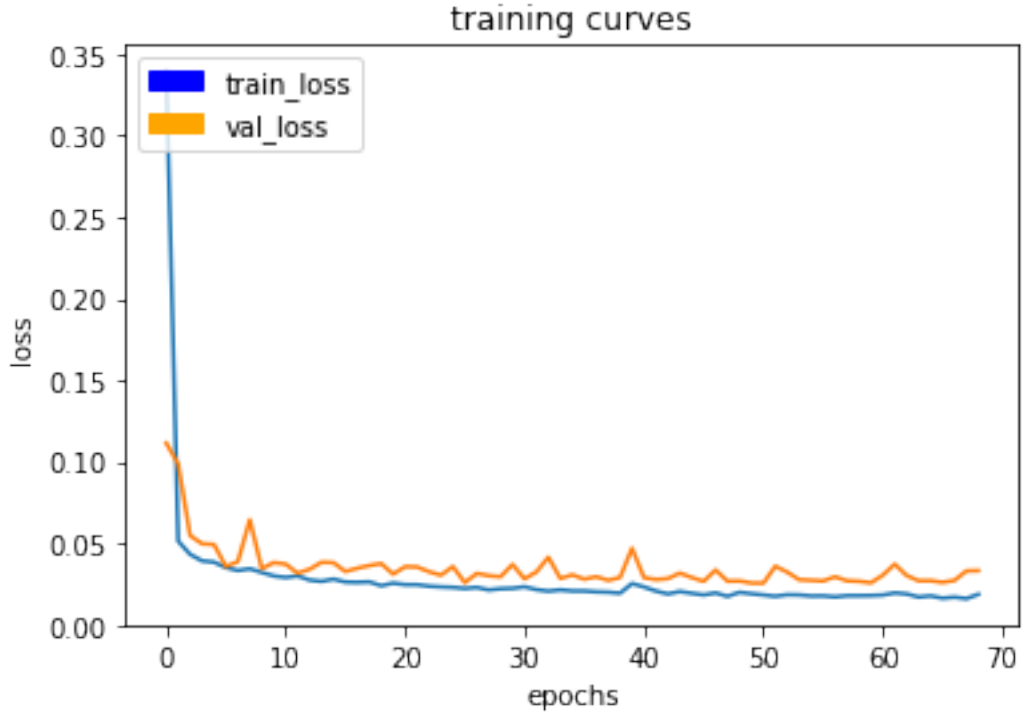
```
200/200 [==============================] - 97s - loss: 0.0163 - val_loss: 0.0260
Epoch 67/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0170
```



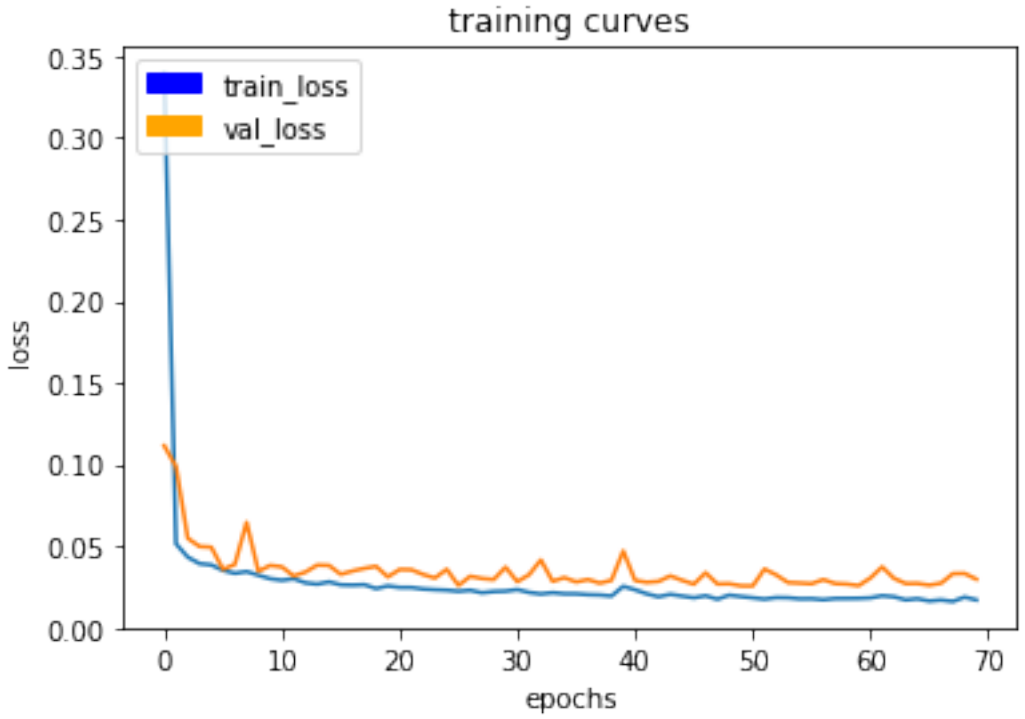

```
200/200 [==============================] - 97s - loss: 0.0170 - val_loss: 0.0272
Epoch 68/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0160
```



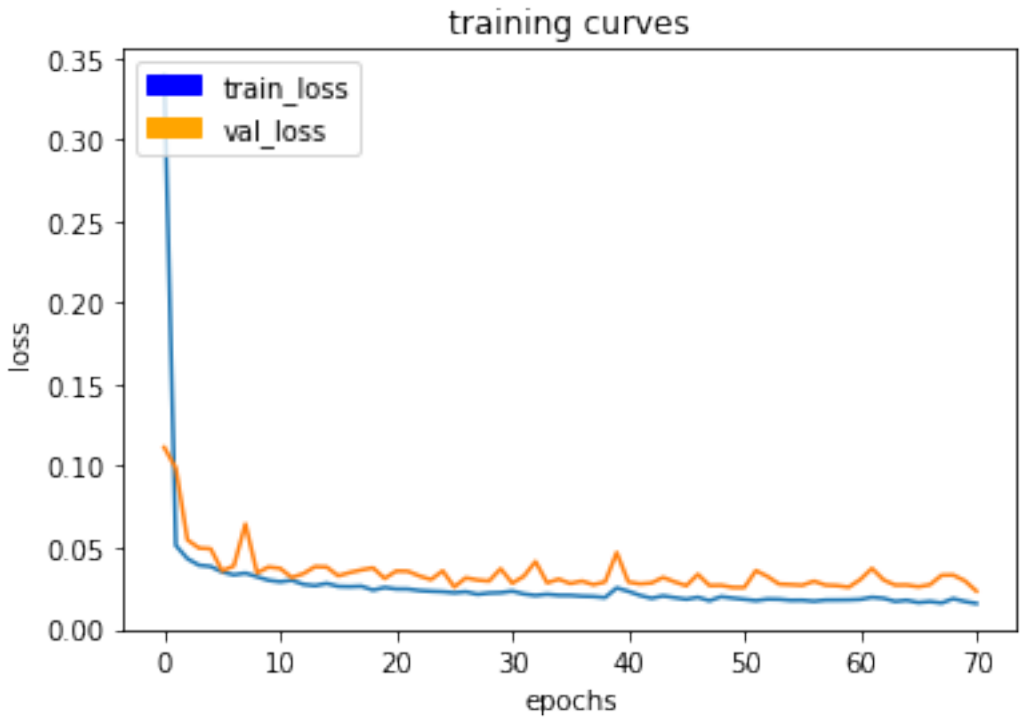

```
200/200 [==============================] - 96s - loss: 0.0160 - val_loss: 0.0330
Epoch 69/100
199/200 [============================>.] - ETA: 0s - loss: 0.0188
```



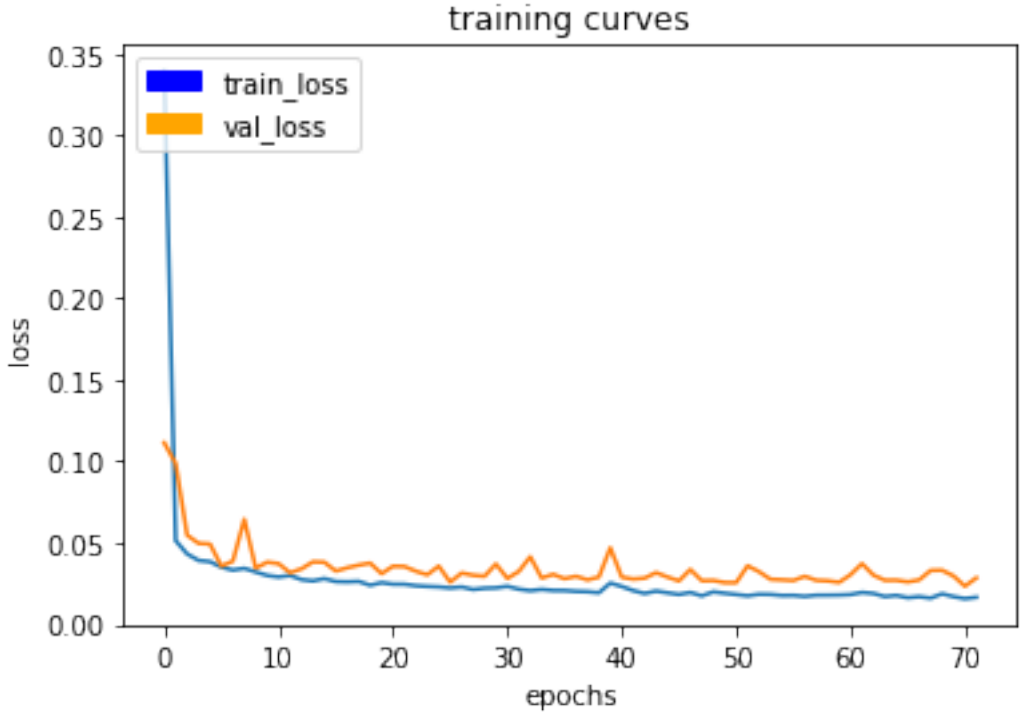
```
200/200 [==============================] - 96s - loss: 0.0188 - val_loss: 0.0332
Epoch 70/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0169
```



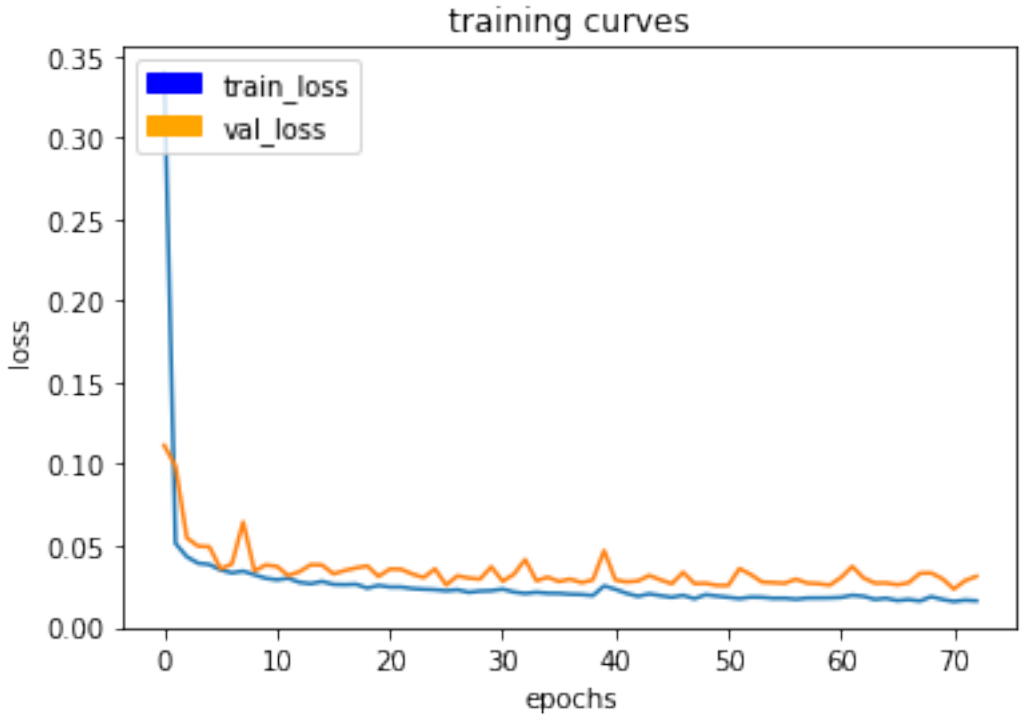

```
200/200 [==============================] - 97s - loss: 0.0169 - val_loss: 0.0297
Epoch 71/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0157
```



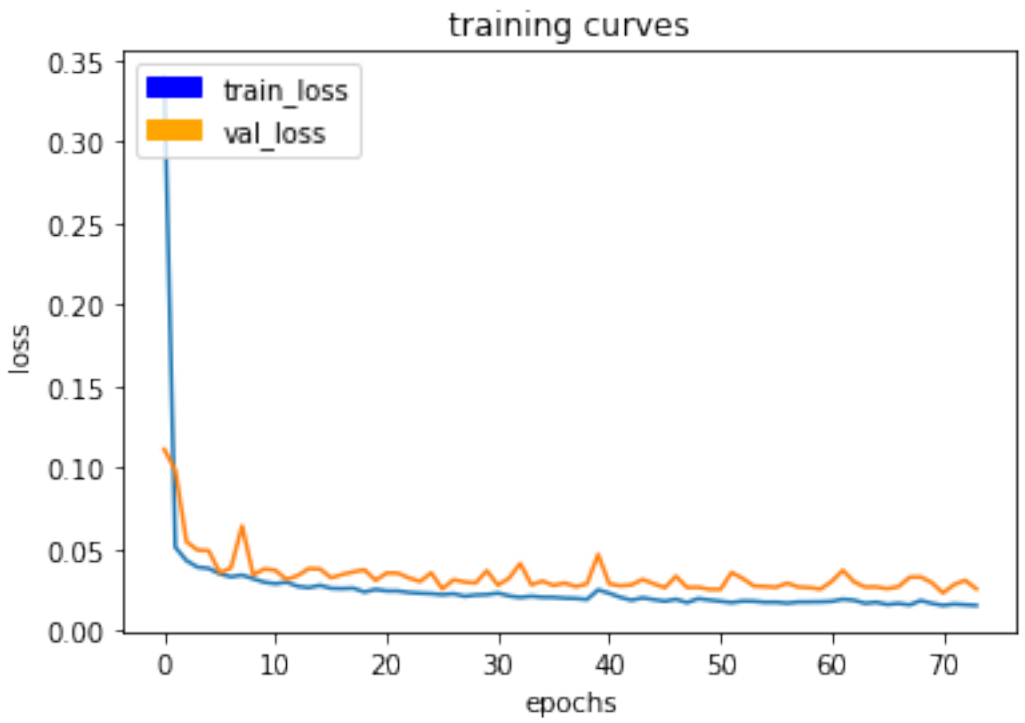

```
200/200 [==============================] - 96s - loss: 0.0157 - val_loss: 0.0234
Epoch 72/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0166
```



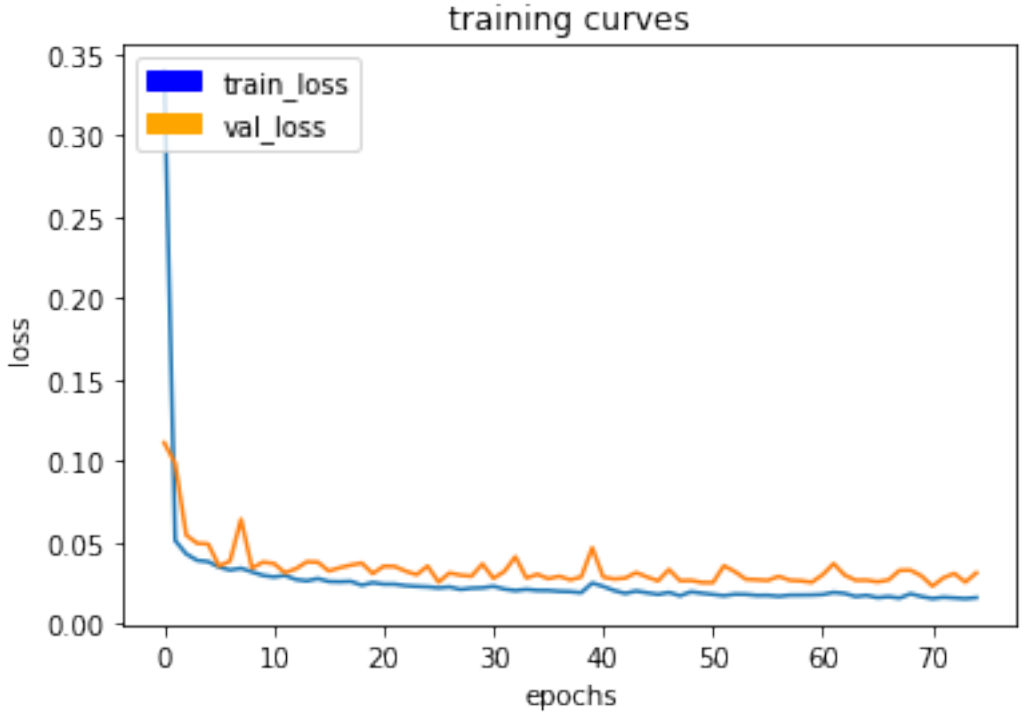
```
200/200 [==============================] - 97s - loss: 0.0166 - val_loss: 0.0285
Epoch 73/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0161
```



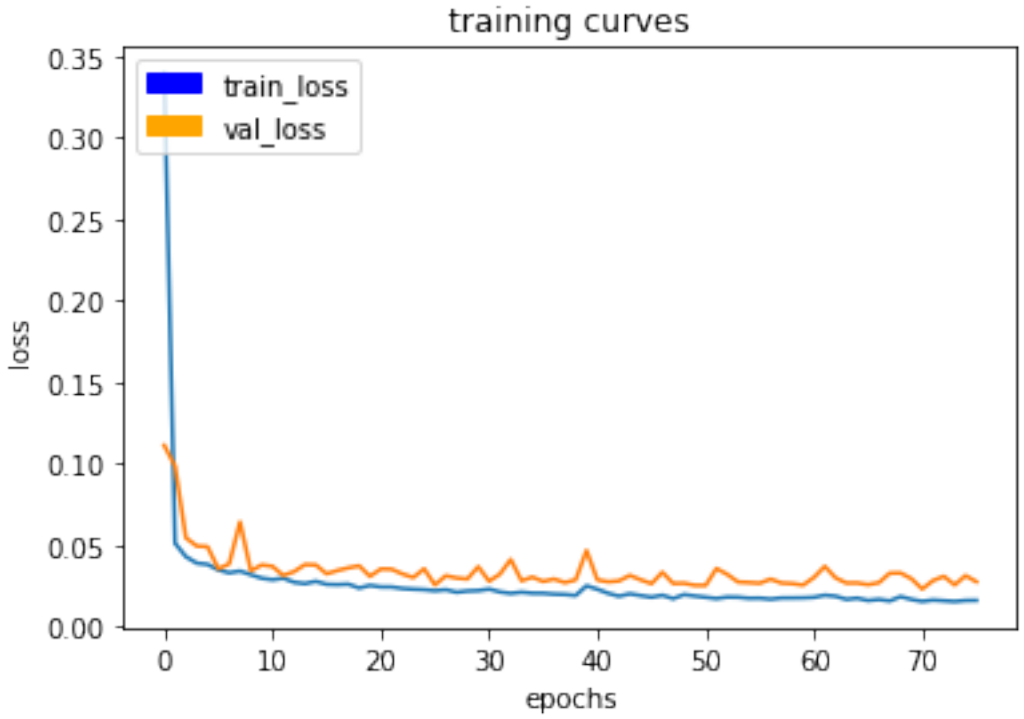

```
200/200 [==============================] - 97s - loss: 0.0161 - val_loss: 0.0311
Epoch 74/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0157
```



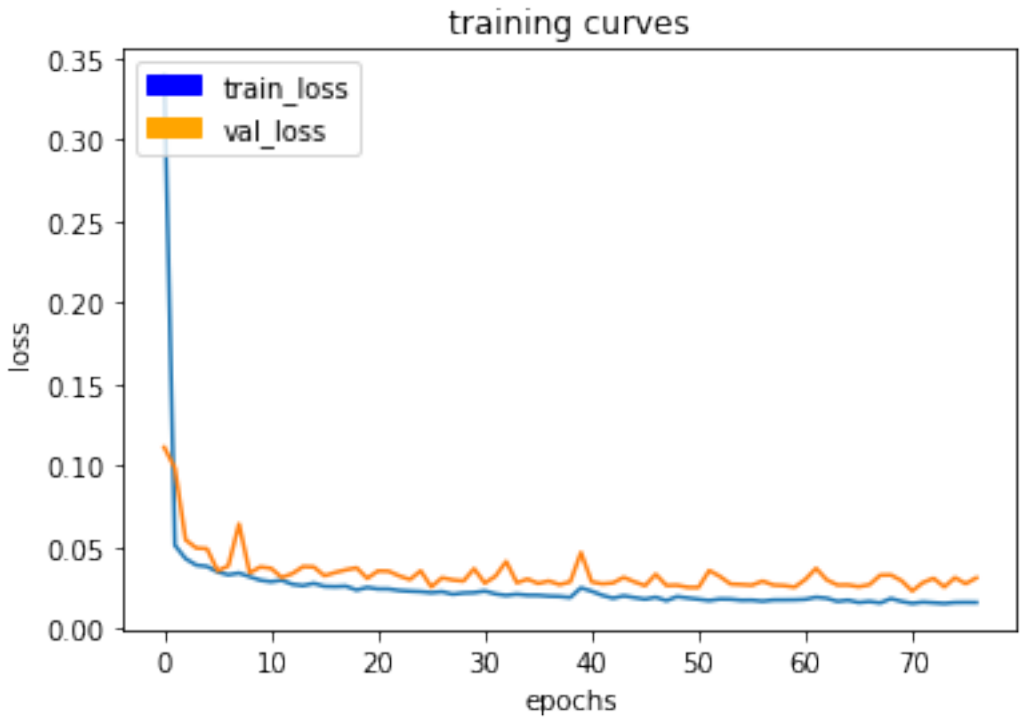

```
200/200 [==============================] - 96s - loss: 0.0157 - val_loss: 0.0258
Epoch 75/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0164
```



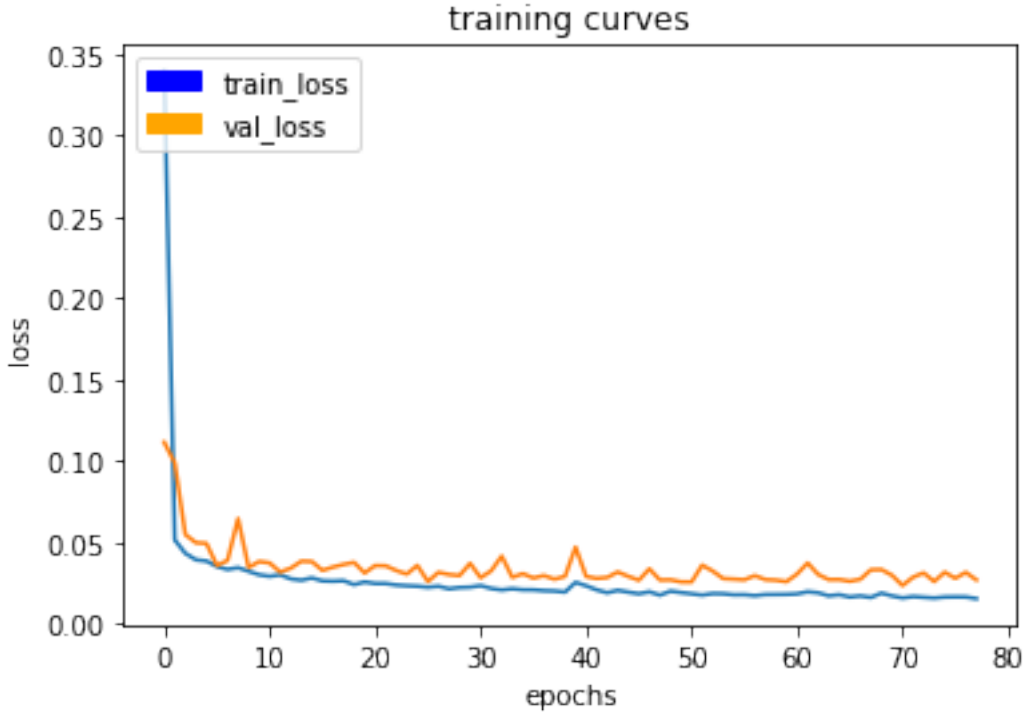
```
200/200 [==============================] - 97s - loss: 0.0164 - val_loss: 0.0315
Epoch 76/100
199/200 [============================>.] - ETA: 0s - loss: 0.0163
```



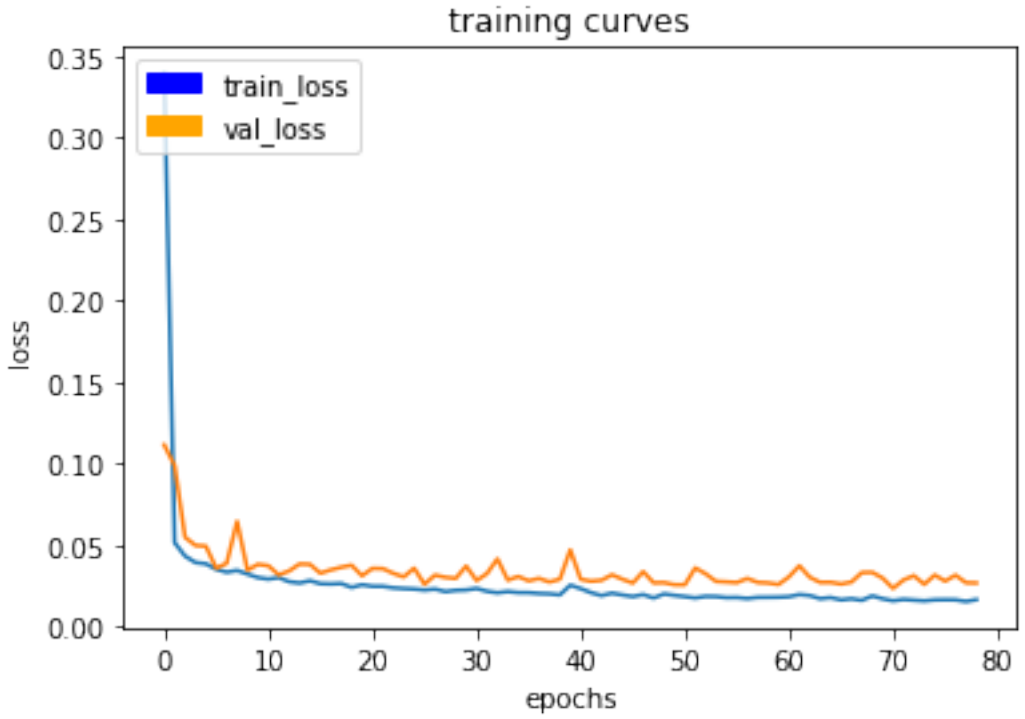

```
200/200 [==============================] - 96s - loss: 0.0163 - val_loss: 0.0278
Epoch 77/100
199/200 [============================>.] - ETA: 0s - loss: 0.0165
```



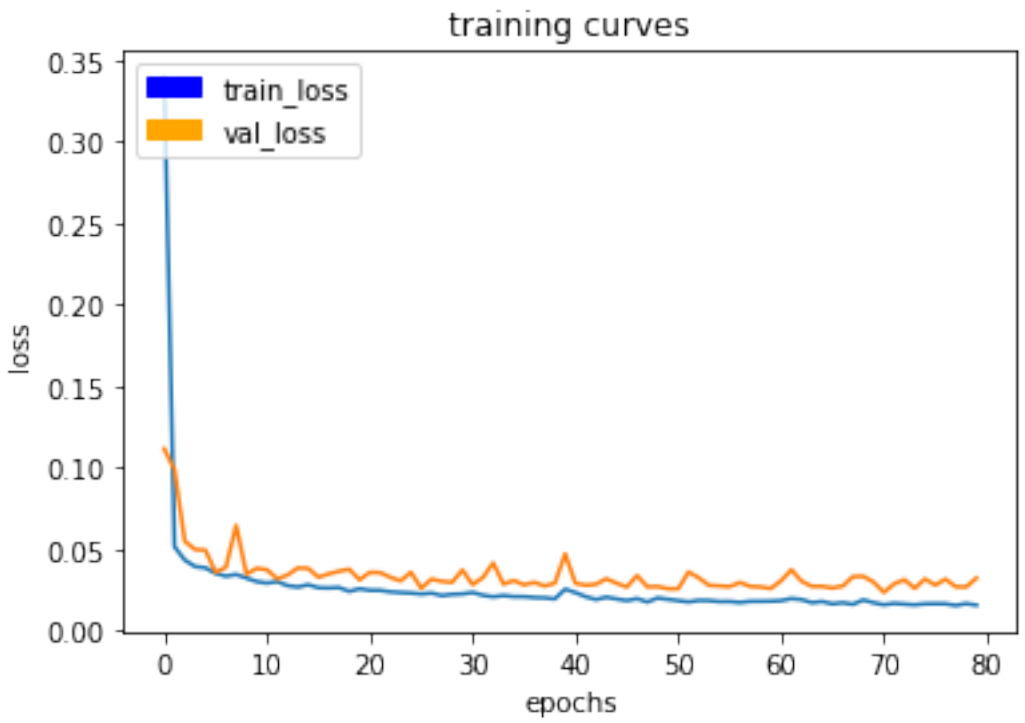

```
200/200 [==============================] - 97s - loss: 0.0164 - val_loss: 0.0314
Epoch 78/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0153
```



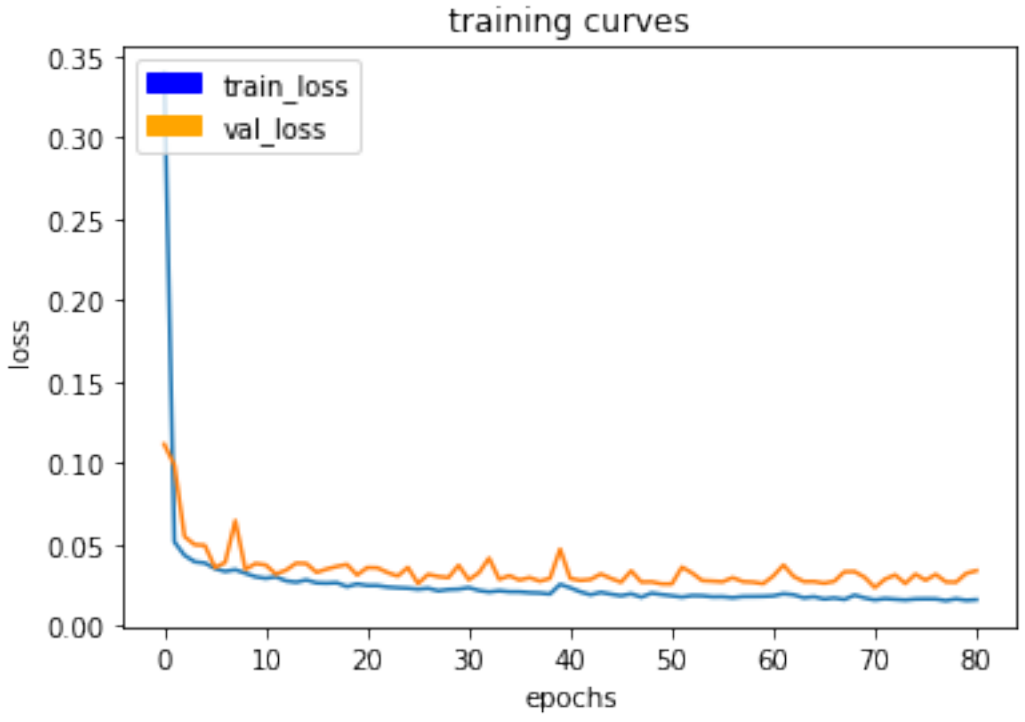
```
200/200 [==============================] - 96s - loss: 0.0152 - val_loss: 0.0268
Epoch 79/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0164
```



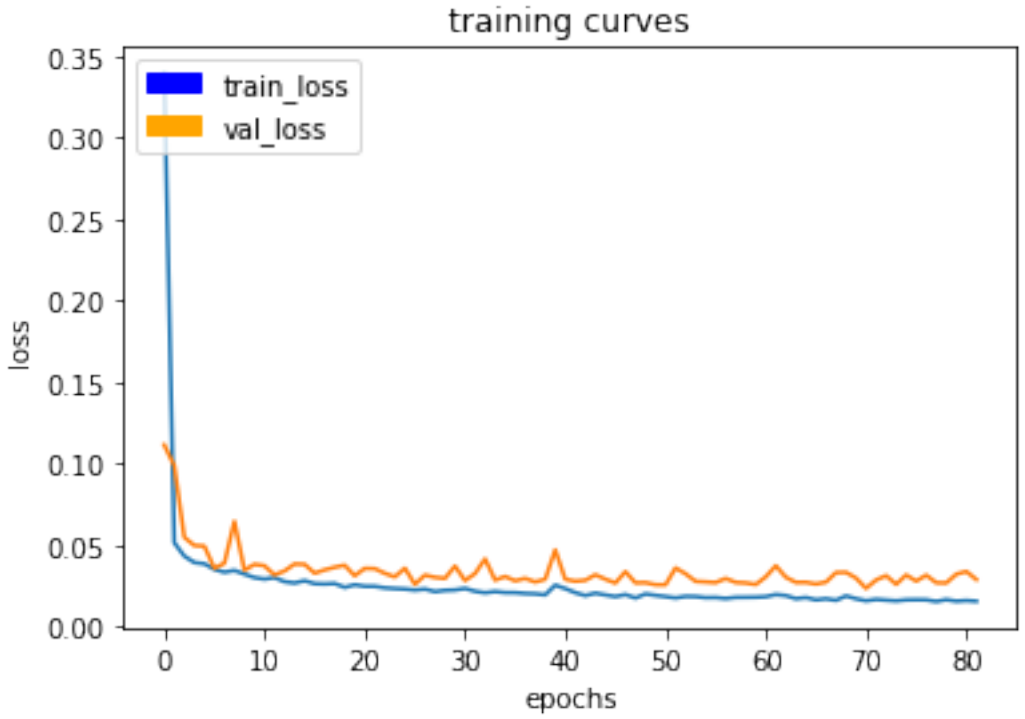

```
200/200 [==============================] - 96s - loss: 0.0165 - val_loss: 0.0266
Epoch 80/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0154
```



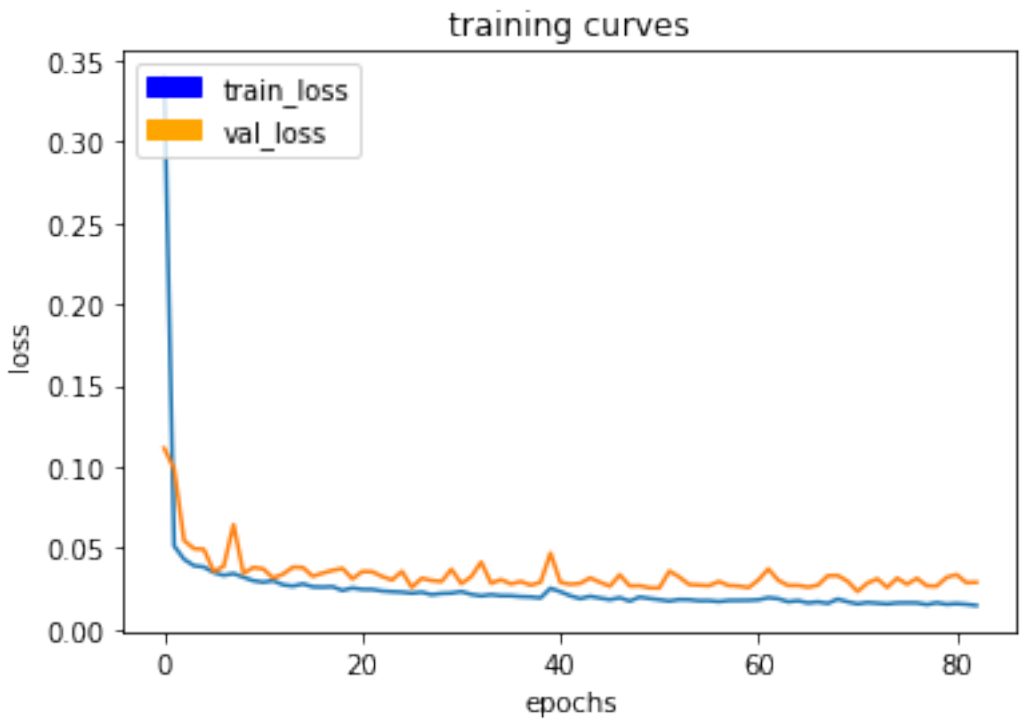

```
200/200 [==============================] - 96s - loss: 0.0154 - val_loss: 0.0320
Epoch 81/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0159
```



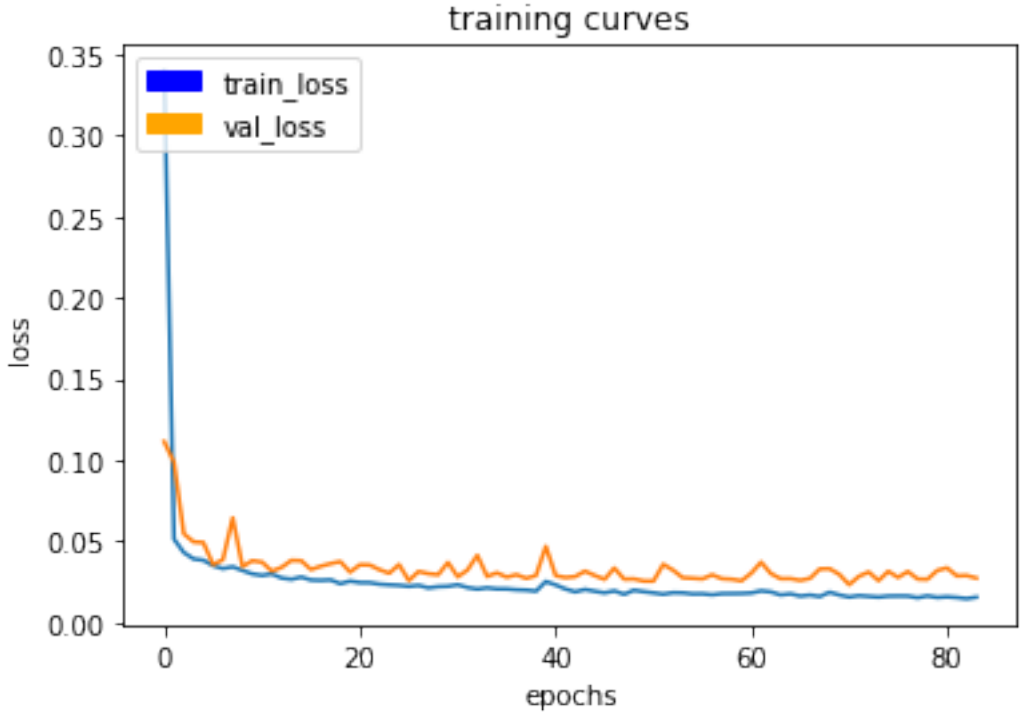
```
200/200 [==============================] - 97s - loss: 0.0159 - val_loss: 0.0337
Epoch 82/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0154
```



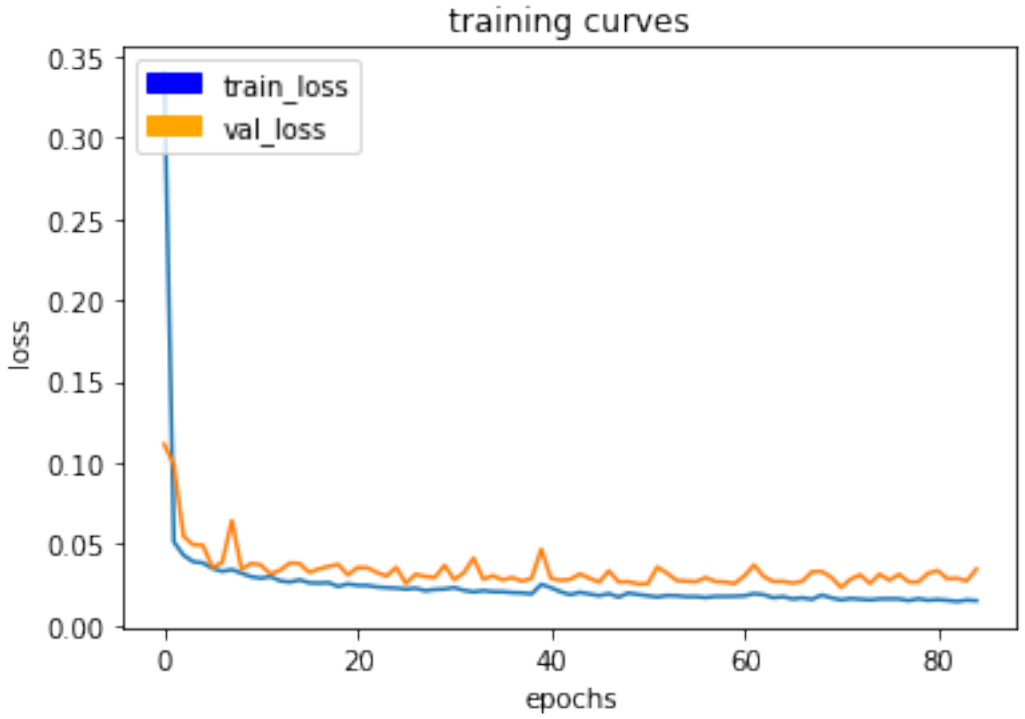

```
200/200 [==============================] - 96s - loss: 0.0154 - val_loss: 0.0287
Epoch 83/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0147
```



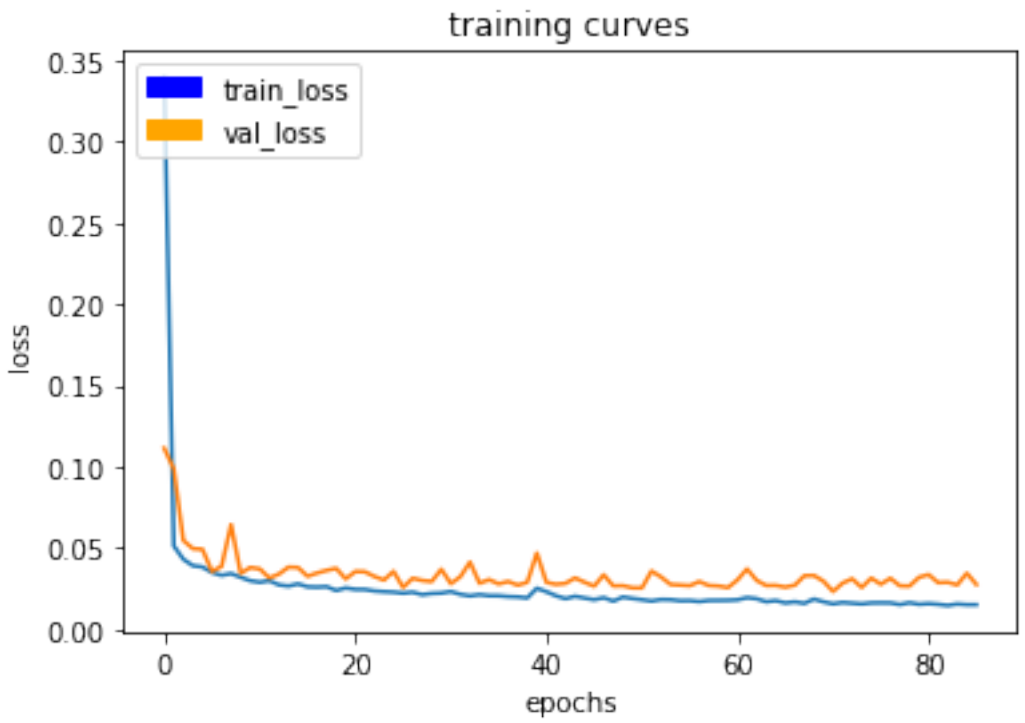

```
200/200 [==============================] - 96s - loss: 0.0147 - val_loss: 0.0290
Epoch 84/100
199/200 [============================>.] - ETA: 0s - loss: 0.0156
```



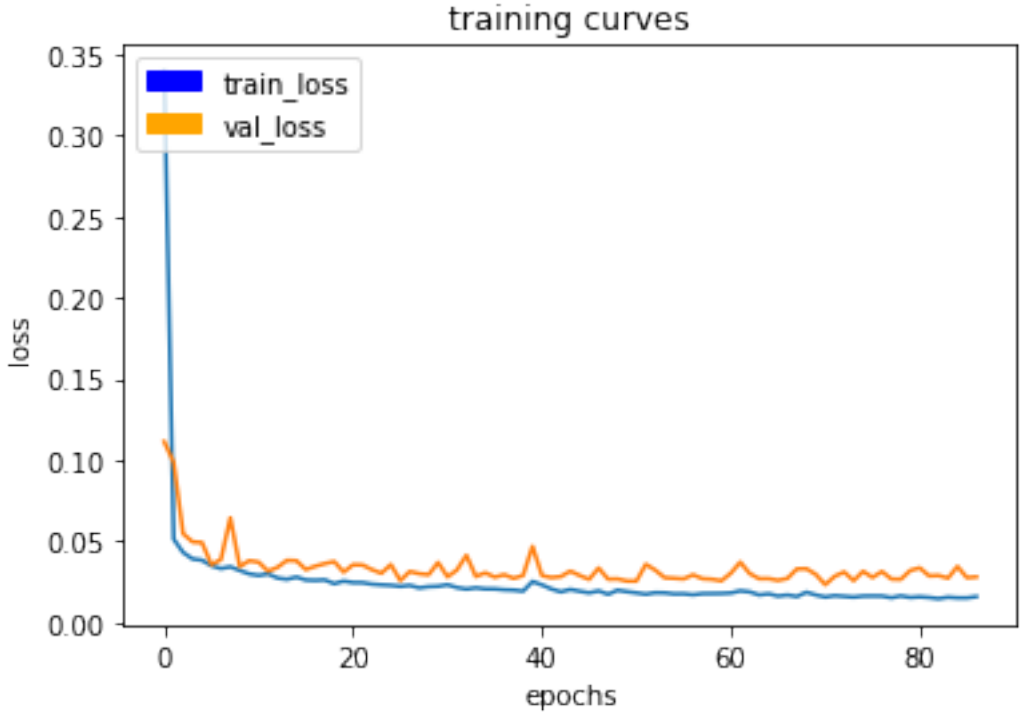
```
200/200 [==============================] - 96s - loss: 0.0156 - val_loss: 0.0274
Epoch 85/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0152
```



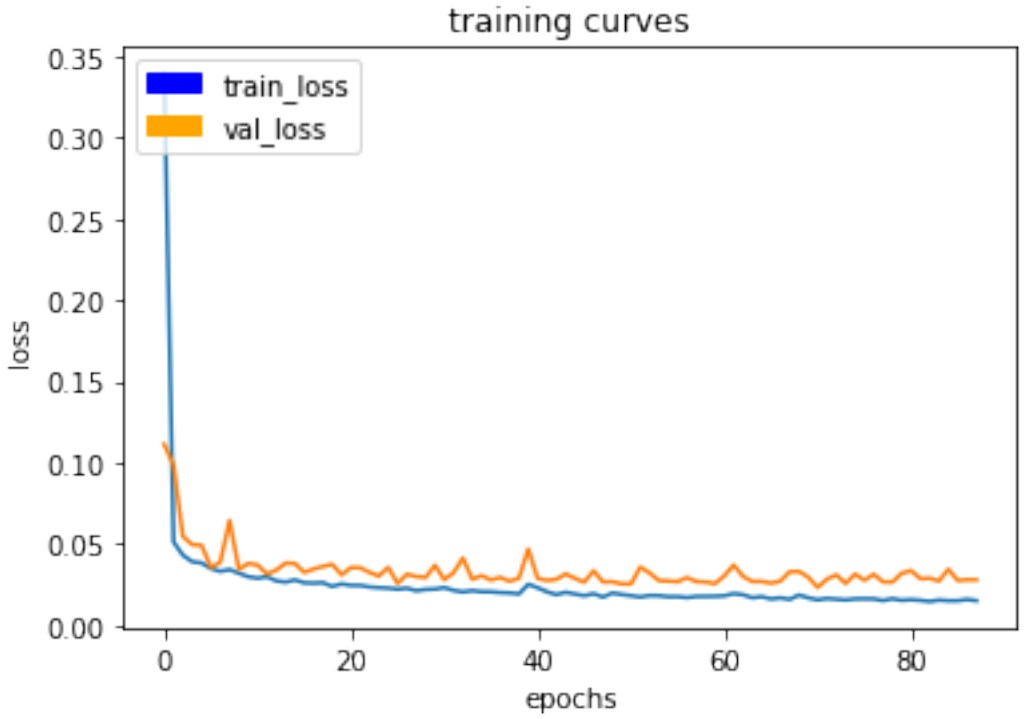

```
200/200 [==============================] - 96s - loss: 0.0152 - val_loss: 0.0345
Epoch 86/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0152
```



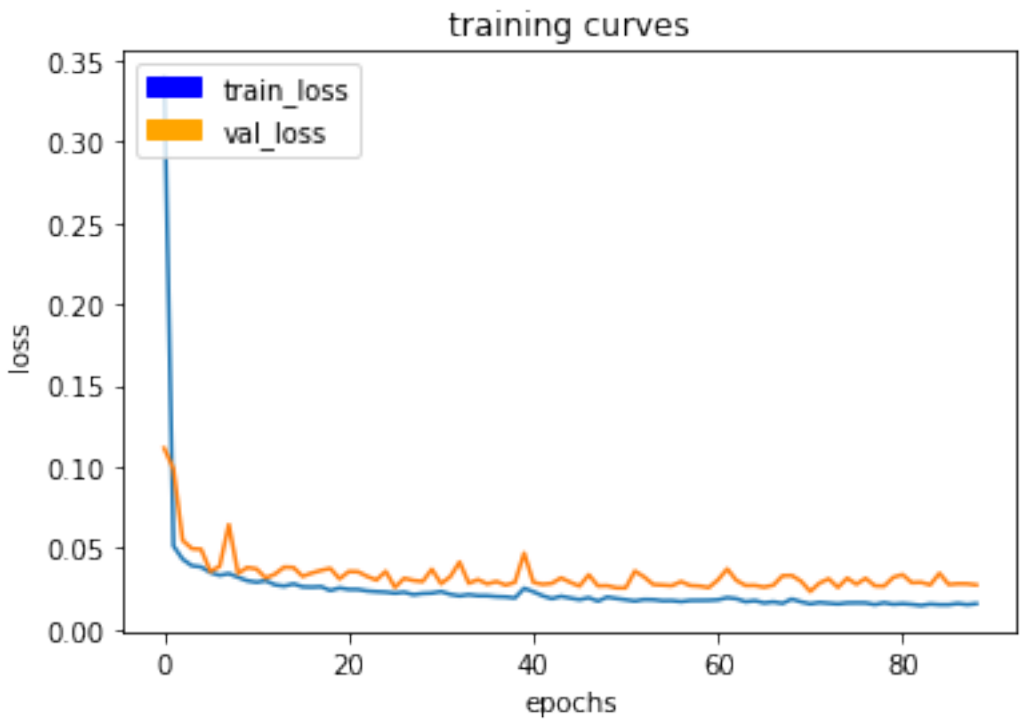

```
200/200 [==============================] - 96s - loss: 0.0152 - val_loss: 0.0276
Epoch 87/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0159
```



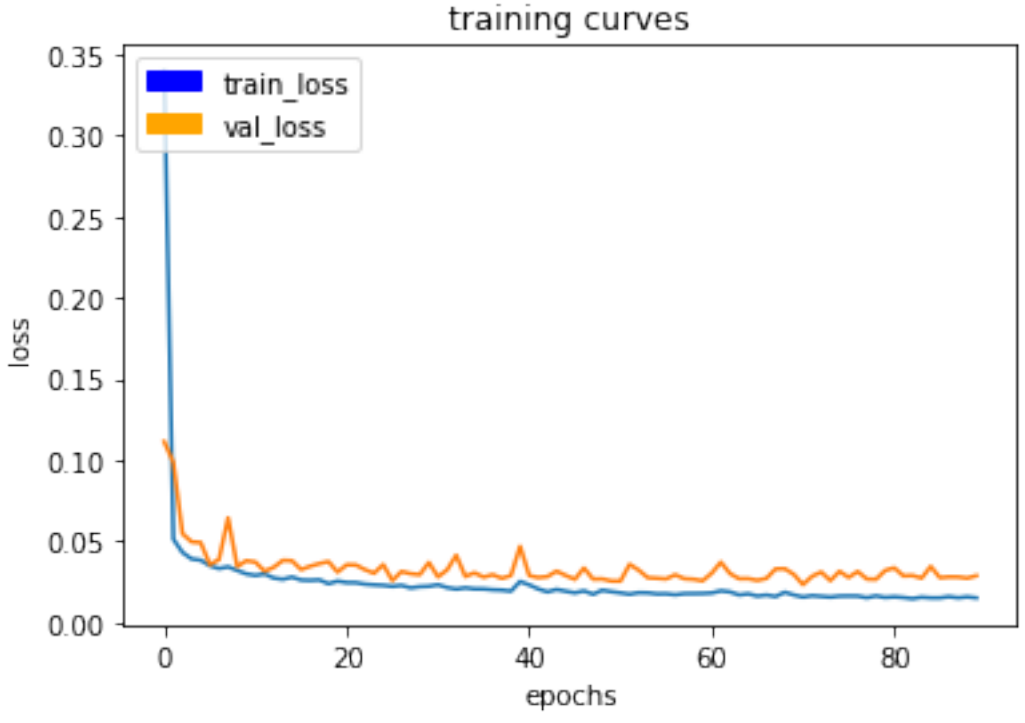
```
200/200 [==============================] - 97s - loss: 0.0159 - val_loss: 0.0281
Epoch 88/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0153
```



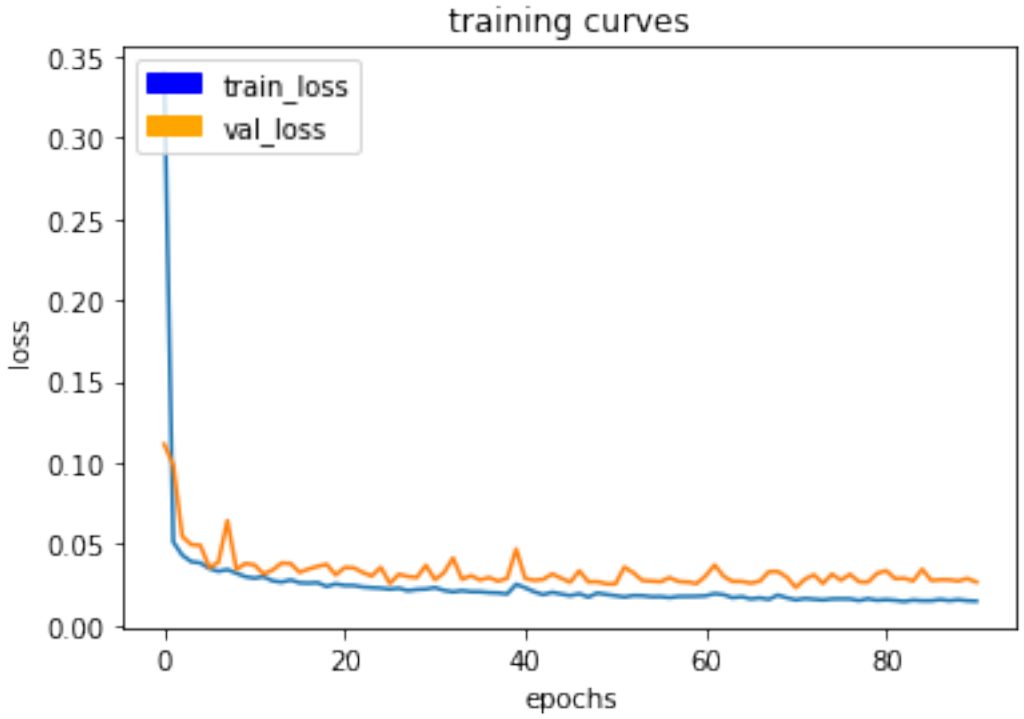

```
200/200 [==============================] - 96s - loss: 0.0153 - val_loss: 0.0280
Epoch 89/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0160
```



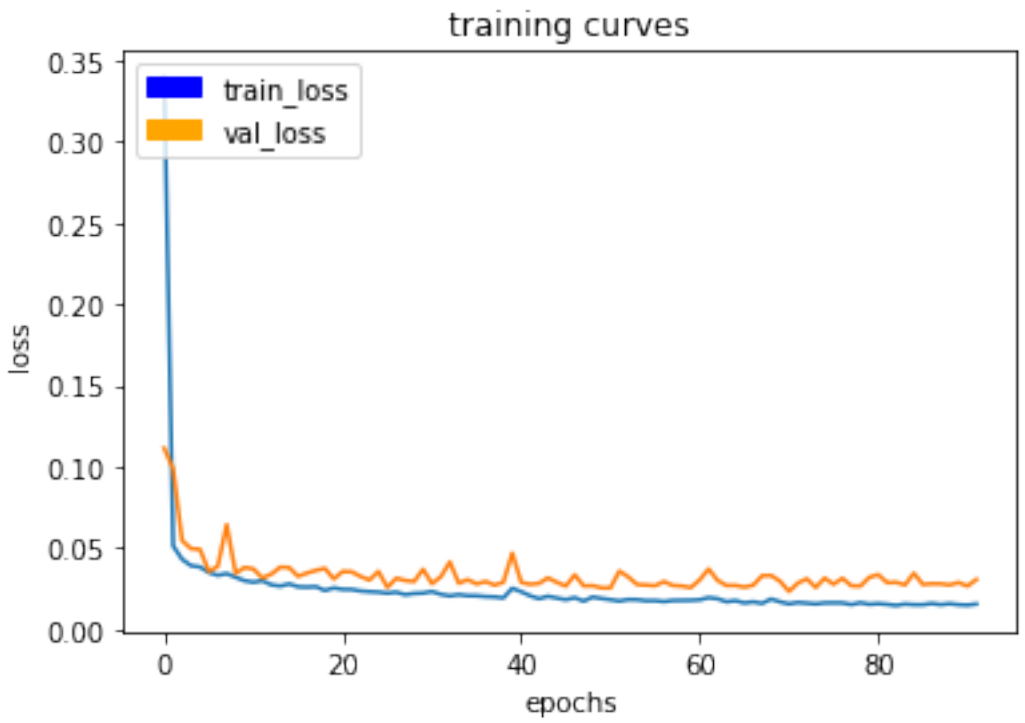

```
200/200 [==============================] - 97s - loss: 0.0160 - val_loss: 0.0274
Epoch 90/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0154
```



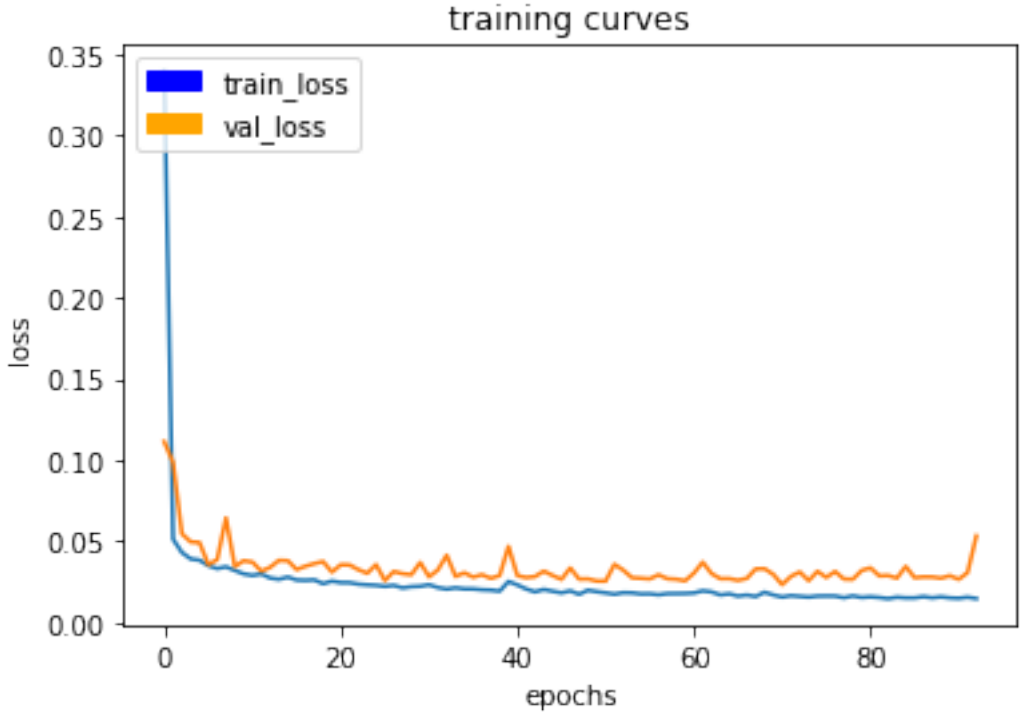
```
200/200 [==============================] - 97s - loss: 0.0154 - val_loss: 0.0288
Epoch 91/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0150
```



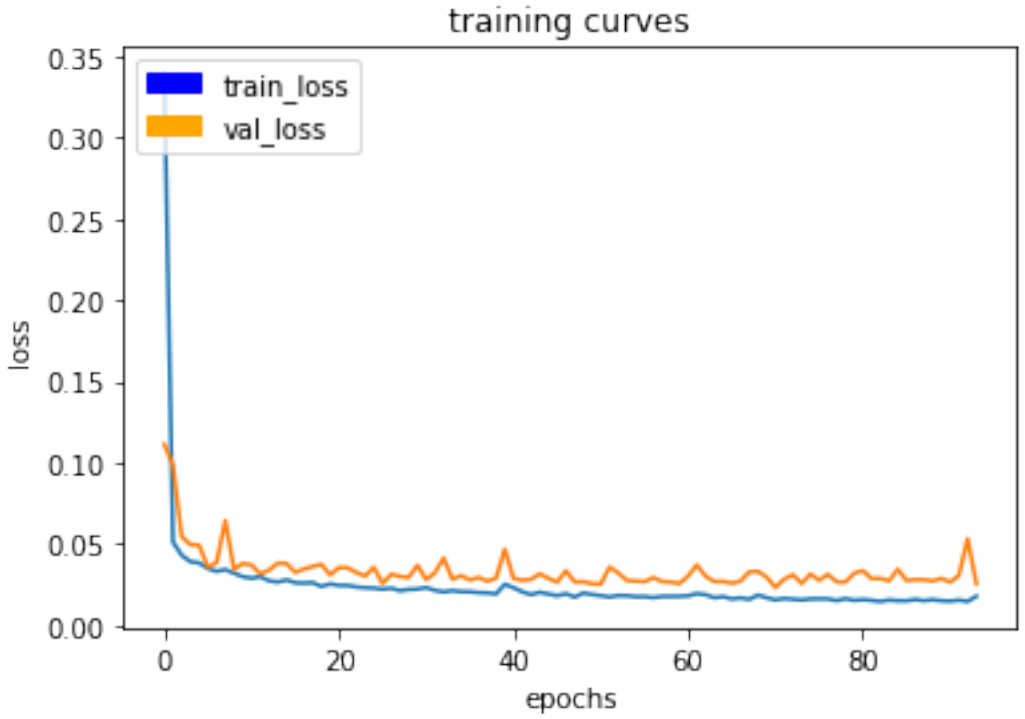

```
200/200 [==============================] - 96s - loss: 0.0150 - val_loss: 0.0266
Epoch 92/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0157
```



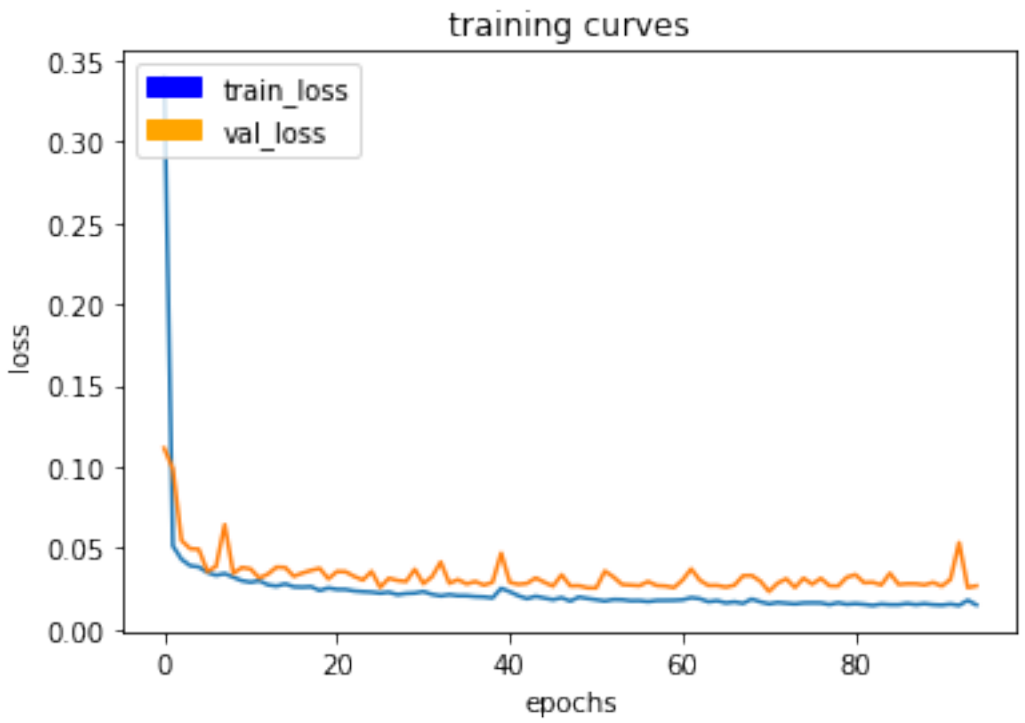

```
200/200 [==============================] - 97s - loss: 0.0156 - val_loss: 0.0307
Epoch 93/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0148
```



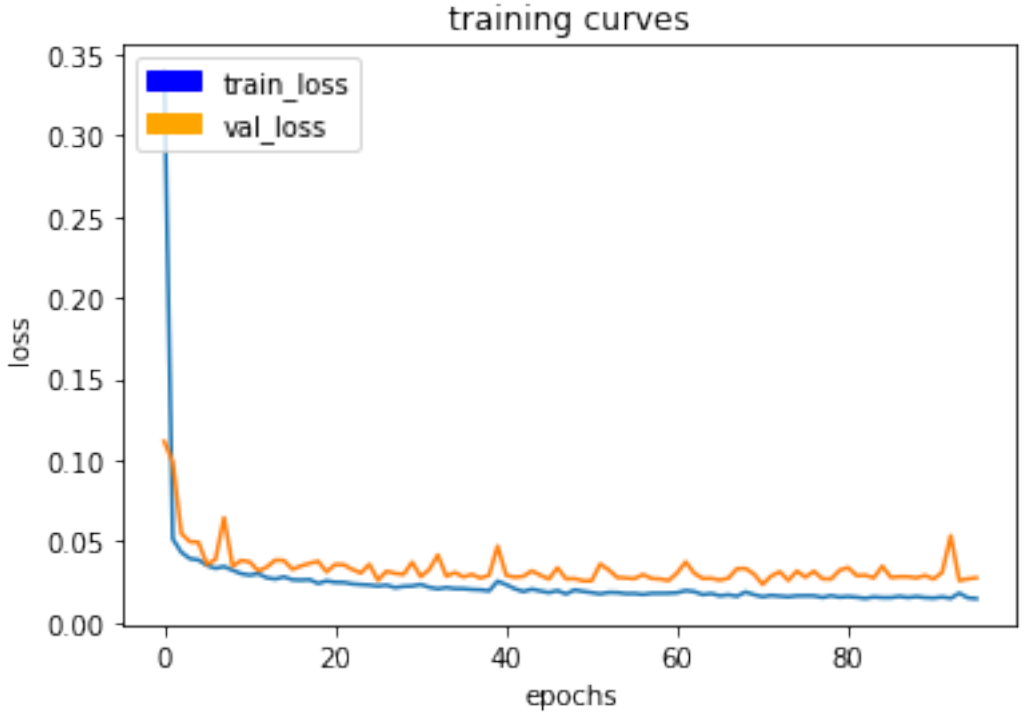
```
200/200 [==============================] - 96s - loss: 0.0148 - val_loss: 0.0534
Epoch 94/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0181
```



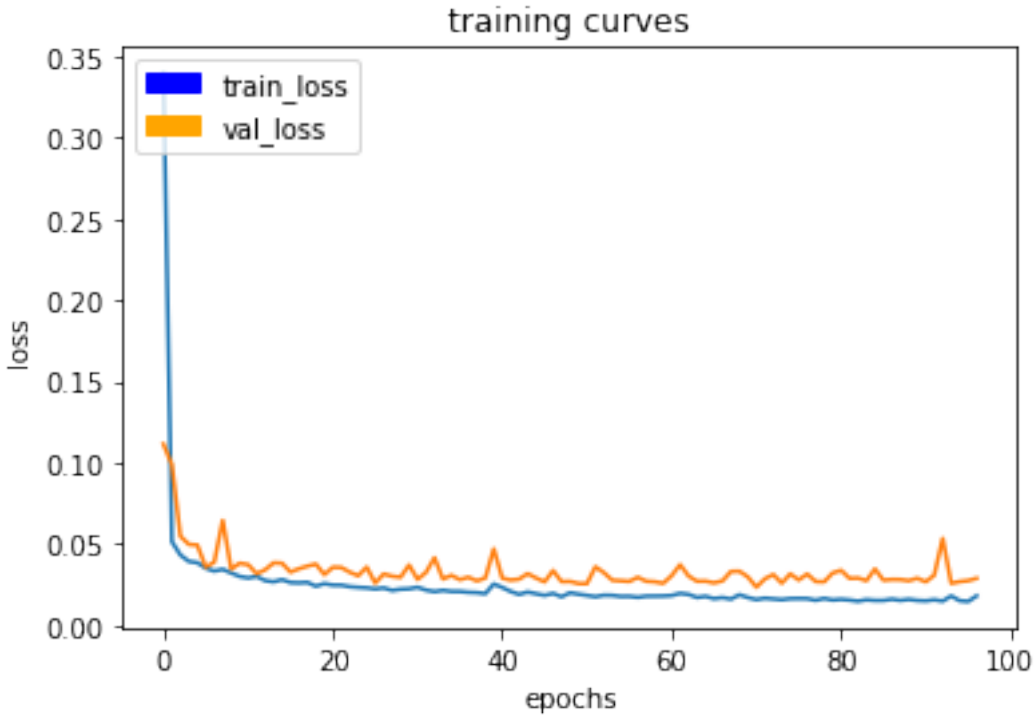

```
200/200 [==============================] - 96s - loss: 0.0180 - val_loss: 0.0257
Epoch 95/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0151
```



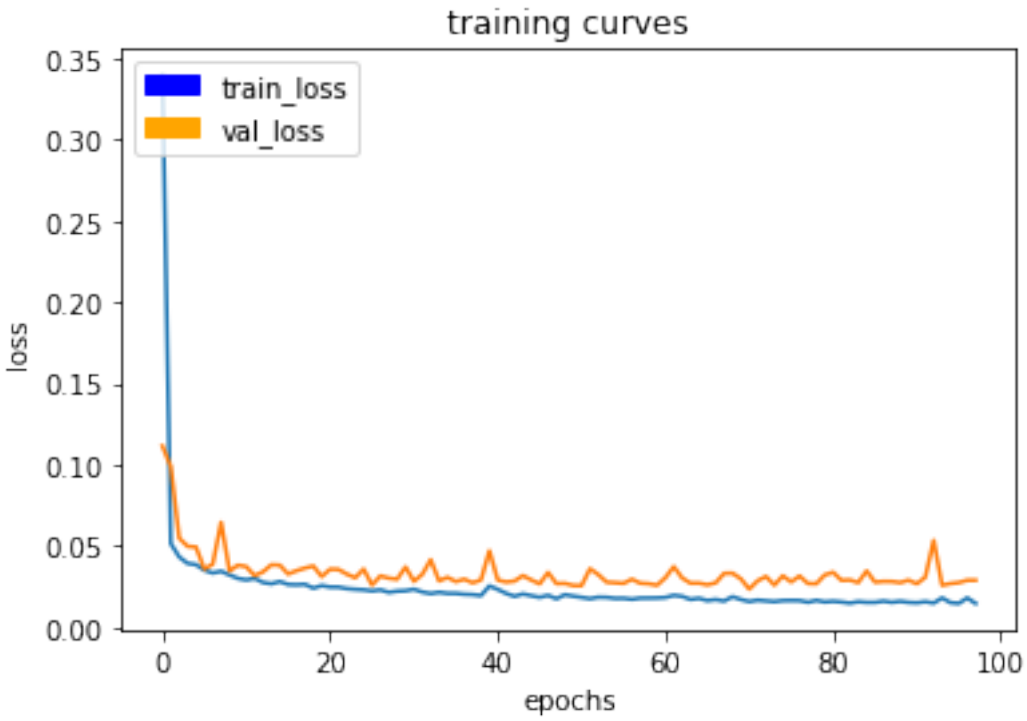

```
200/200 [==============================] - 97s - loss: 0.0151 - val_loss: 0.0268
Epoch 96/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0146
```



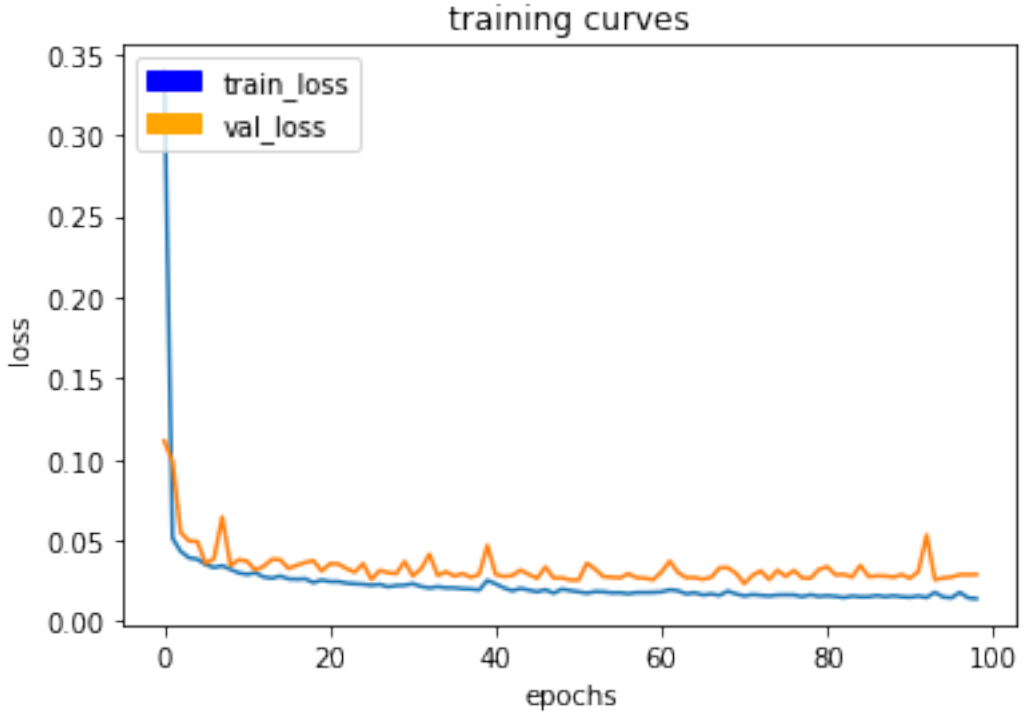
```
200/200 [==============================] - 97s - loss: 0.0146 - val_loss: 0.0274
Epoch 97/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0181
```



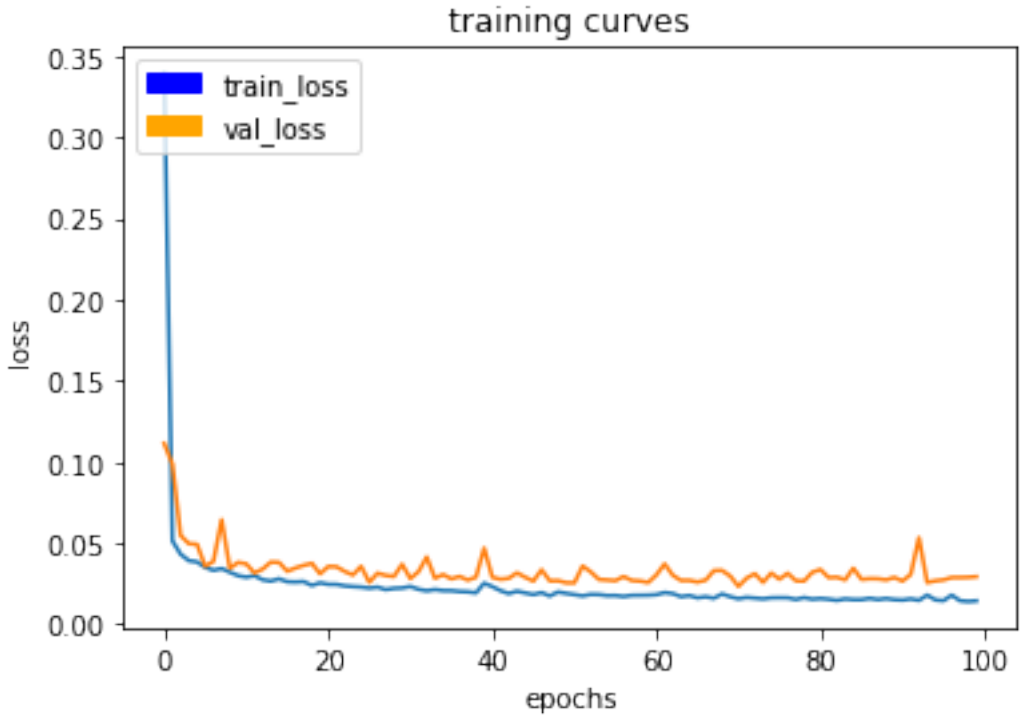

```
200/200 [==============================] - 96s - loss: 0.0180 - val_loss: 0.0288
Epoch 98/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0145
```




```
200/200 [==============================] - 96s - loss: 0.0145 - val_loss: 0.0288
Epoch 99/100
199/200 [==============================>.] - ETA: 0s - loss: 0.0141
```



```
200/200 [==============================] - 96s - loss: 0.0141 - val_loss: 0.0289
Epoch 100/100
199/200 [============================>.] - ETA: 0s - loss: 0.0143
```




```
200/200 [==============================] - 97s - loss: 0.0144 - val_loss: 0.0293
```

```
In [10]: # Save your trained model weights
         weight_file_name = 'model_weights'
         model_tools.save_network(model, weight_file_name)
```

1.5 Prediction

Now that you have your model trained and saved, you can make predictions on your validation dataset. These predictions can be compared to the mask images, which are the ground truth labels, to evaluate how well your model is doing under different conditions.

There are three different predictions available from the helper code provided: - **patrol_with_targ**: Test how well the network can detect the hero from a distance. - **patrol_non_targ**: Test how often the network makes a mistake and identifies the wrong person as the target. - **following_images**: Test how well the network can identify the target while following them.

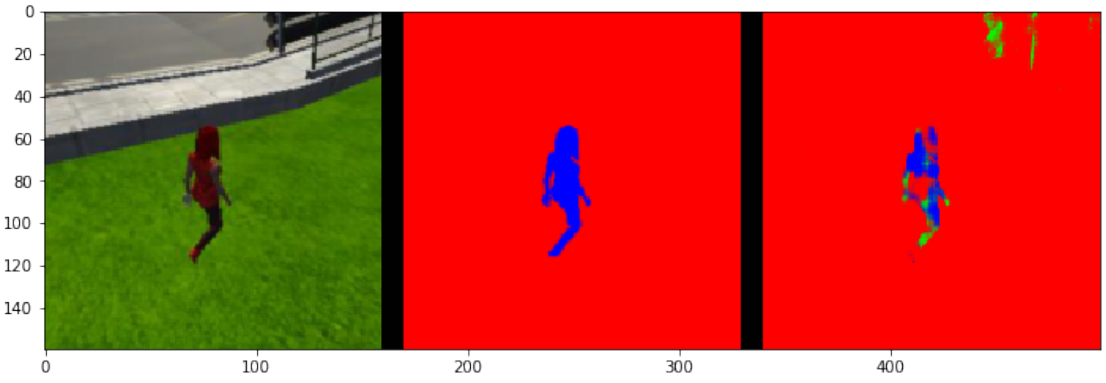
```
In [11]: # If you need to load a model which you previously trained you can uncomment the codeli

         # weight_file_name = 'model_weights'
         # restored_model = model_tools.load_network(weight_file_name)
```

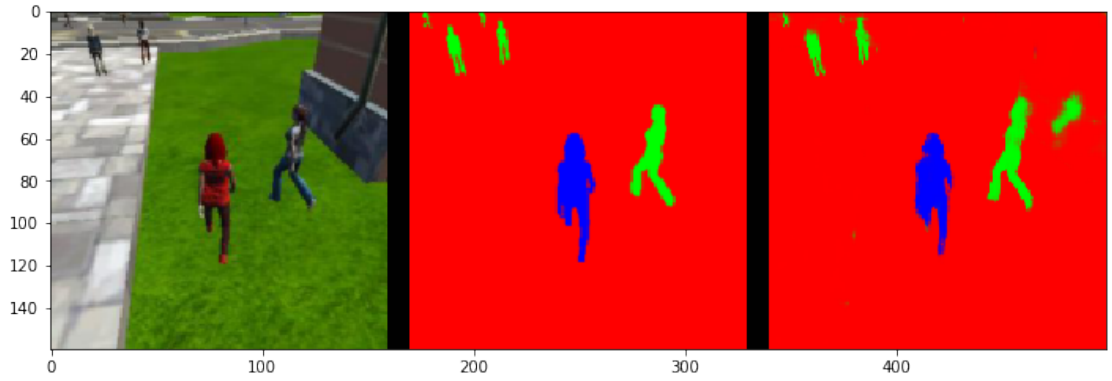
The following cell will write predictions to files and return paths to the appropriate directories. The `run_num` parameter is used to define or group all the data for a particular model run. You can change it for different runs. For example, 'run_1', 'run_2' etc.

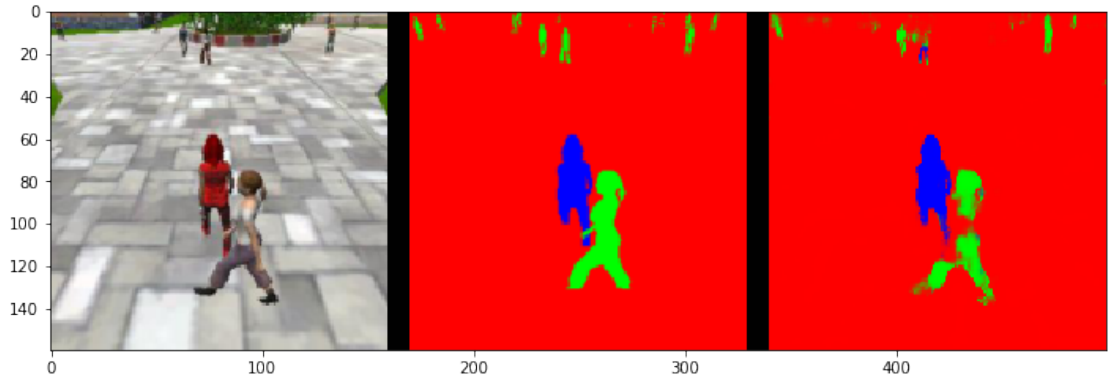
```
In [12]: run_num = 'run_1'

         val_with_targ, pred_with_targ = model_tools.write_predictions_grade_set(model,
                                                 run_num,'patrol_with_targ', 'sample_evaluation_

         val_no_targ, pred_no_targ = model_tools.write_predictions_grade_set(model,
                                                 run_num,'patrol_non_targ', 'sample_evaluation_d

         val_following, pred_following = model_tools.write_predictions_grade_set(model,
                                                 run_num,'following_images', 'sample_evaluation_
```

Now lets look at your predictions, and compare them to the ground truth labels and original images. Run each of the following cells to visualize some sample images from the predictions in the validation set.

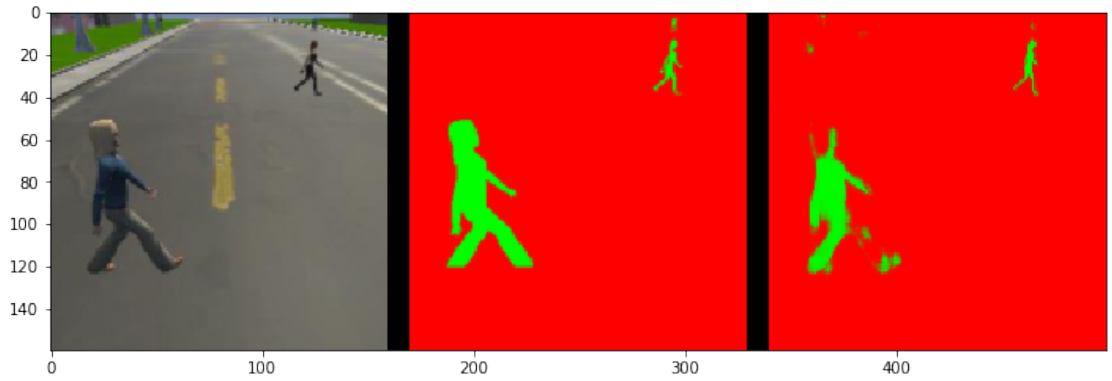
```
In [17]: # images while following the target
         im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','following_images
         for i in range(3):
             im_tuple = plotting_tools.load_images(im_files[i])
             plotting_tools.show_images(im_tuple)
```

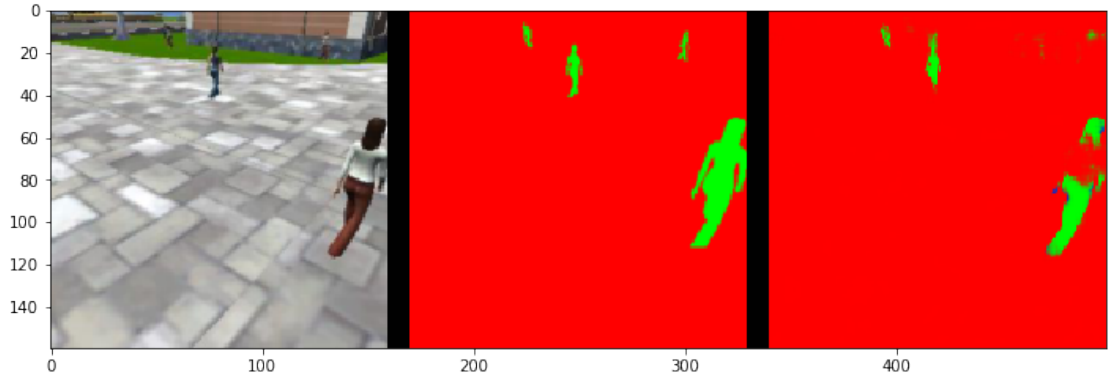


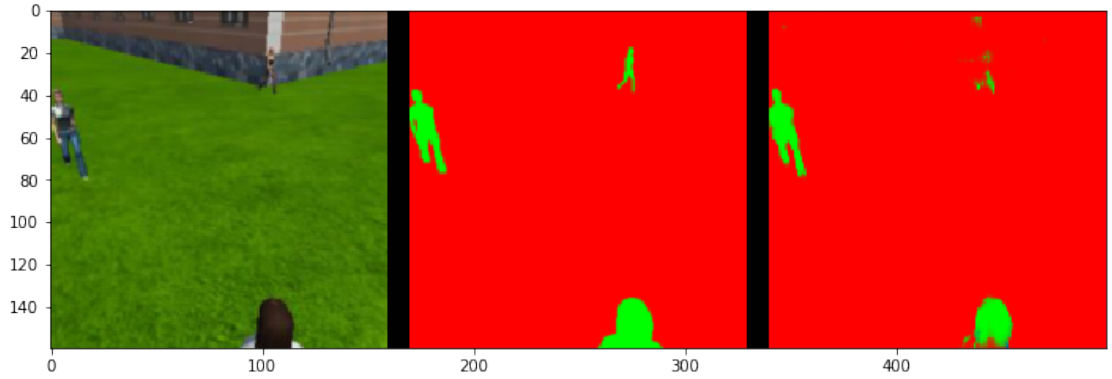




In [18]:
```
# images while at patrol without target
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_non_targ'
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

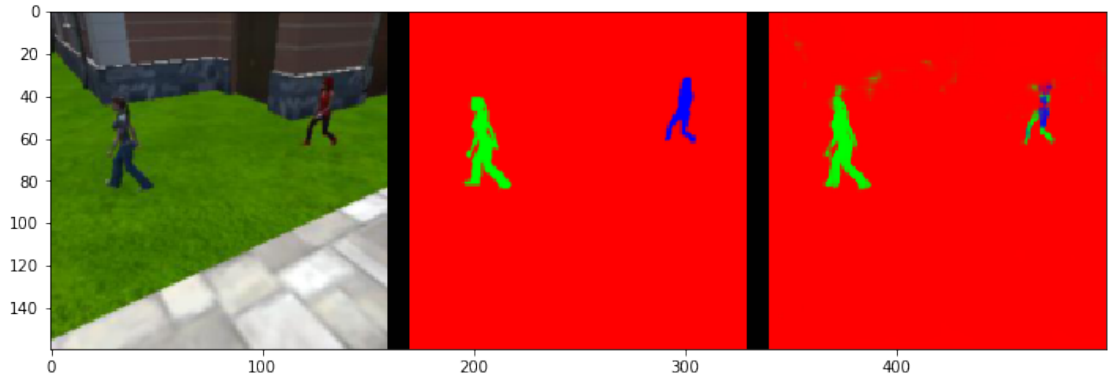


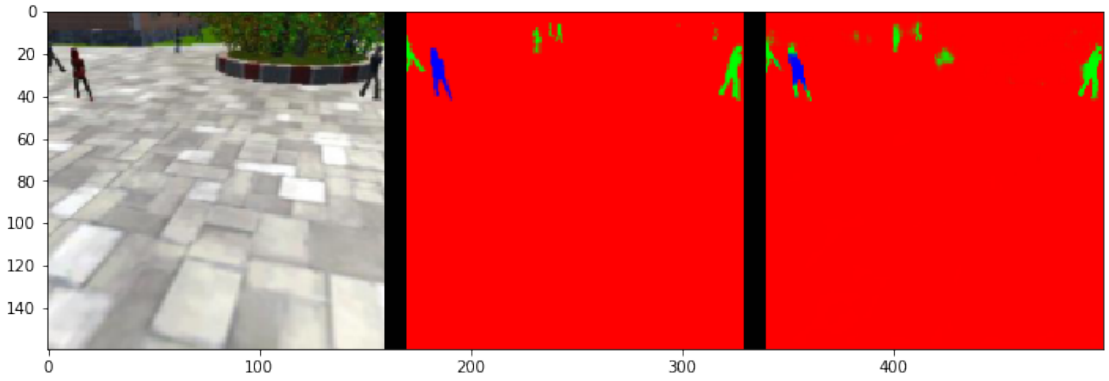


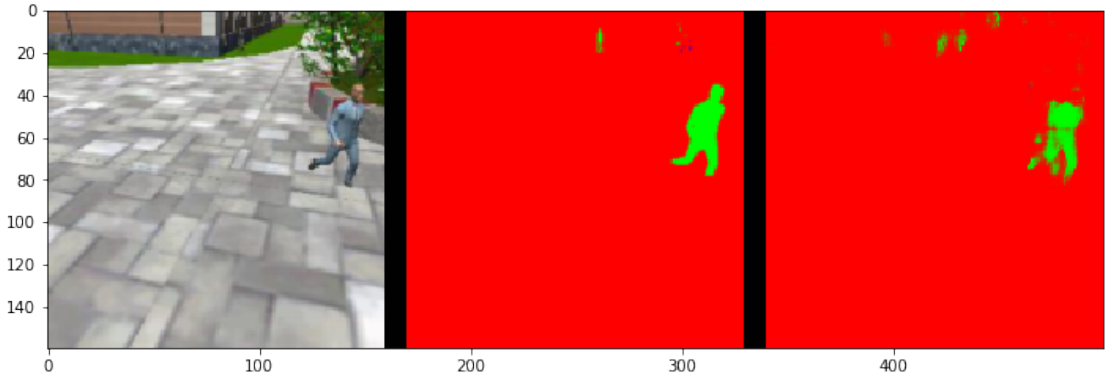


In [19]:

```
# images while at patrol with target
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_with_targ
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```







1.6 Evaluation

Evaluate your model! The following cells include several different scores to help you evaluate your model under the different conditions discussed during the Prediction step.

```
In [20]: # Scores for while the quad is following behind the target.
         true_pos1, false_pos1, false_neg1, iou1 = scoring_utils.score_run_iou(val_following, pr
```

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9952069229249436
average intersection over union for other people is 0.348246367183434
average intersection over union for the hero is 0.8749019619801721
number true positives: 539, number false positives: 0, number false negatives: 0
```

```
In [21]: # Scores for images while the quad is on patrol and the target is not visable
         true_pos2, false_pos2, false_neg2, iou2 = scoring_utils.score_run_iou(val_no_targ, pred
```

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.9861884958680348
average intersection over union for other people is 0.721654260088537
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 62, number false negatives: 0
```

```
In [22]: # This score measures how well the neural network can detect the target from far away
         true_pos3, false_pos3, false_neg3, iou3 = scoring_utils.score_run_iou(val_with_targ, pr
```

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9960621111155227
average intersection over union for other people is 0.43813107303103743
average intersection over union for the hero is 0.22699085853805873
number true positives: 146, number false positives: 1, number false negatives: 155
```

```
In [24]: # Sum all the true positives, etc from the three datasets to get a weight for the score
         true_pos = true_pos1 + true_pos2 + true_pos3
         false_pos = false_pos1 + false_pos2 + false_pos3
         false_neg = false_neg1 + false_neg2 + false_neg3

         weight = true_pos/(true_pos+false_neg+false_pos)
         print(weight)
```

```
0.7585825027685493
```

```
In [25]: # The IoU for the dataset that never includes the hero is excluded from grading
         final_IoU = (iou1 + iou3)/2
         print(final_IoU)
```

```
0.550946410259
```

```
In [26]: # And the final grade score is
         final_score = final_IoU * weight
         print(final_score)
```

```
0.417938306786
```

```
In [ ]:
```