model_training

May 29, 2018

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=str
                          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):

    output_layer = separable_conv2d_batchnorm(input_layer, filters, strides=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.
    sampled = bilinear_upsample(small_ip_layer)

    # TODO Concatenate the upsampled and large input layers using layers.concatenate
    cat = layers.concatenate([sampled, large_ip_layer])


    # TODO Add some number of separable convolution layers

    output_layer = separable_conv2d_batchnorm(cat, filters)
    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 fi
    encoder_1 = encoder_block(inputs, 32, strides=2)
    encoder_2 = encoder_block(encoder_1, 64, strides=2)


    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv2d_batchnormed = conv2d_batchnorm(encoder_2, 128, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
```

```
    decoder_1 = decoder_block(conv2d_batchnormed, 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 obta
    return layers.Conv2D(num_classes, 1, activation='softmax', padding='same')(decoder_
```

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)
```

1.4.1 Hyperparameters

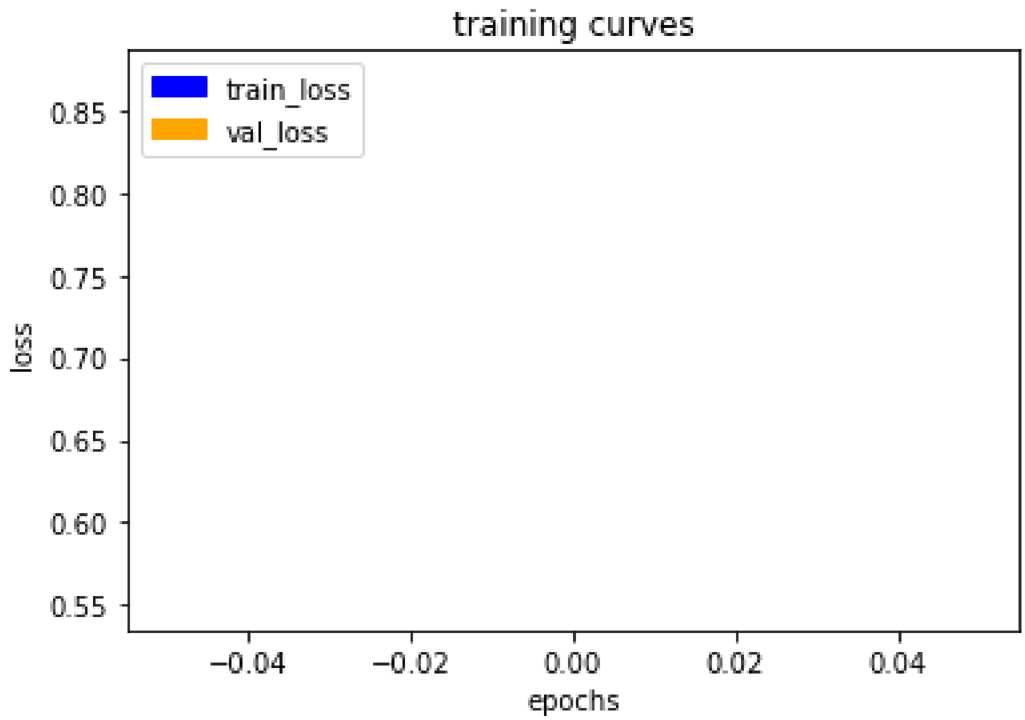
Define and tune your hyperparameters. - **batch_size**: number of training samples/images that get propagated through the network in a single pass. - **num_epochs**: number of times the entire training dataset gets propagated through the network. - **steps_per_epoch**: number of batches of training images 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 = 0.002
        batch_size = 100
        num_epochs = 40
        steps_per_epoch = 200
        validation_steps = 50
        workers = 2
```

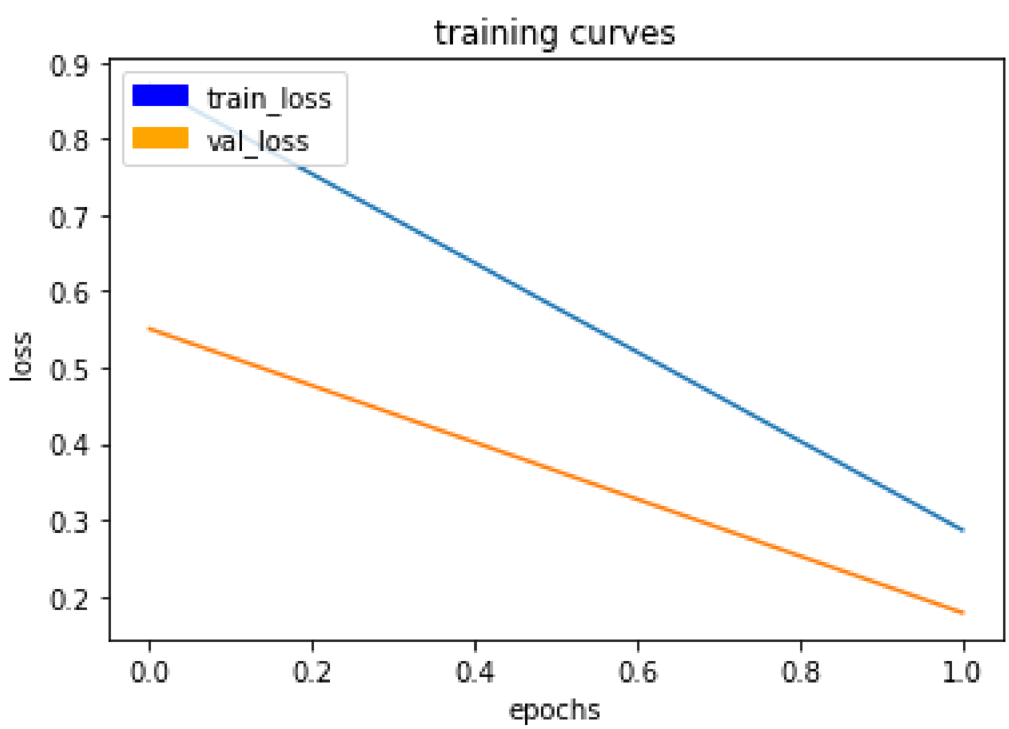
```
In [9]: """
        DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
        """
```

```python
# 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_crosse

# 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', 'va
                                             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 epo
                    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 va
                    callbacks=callbacks,
                    workers = workers)
```

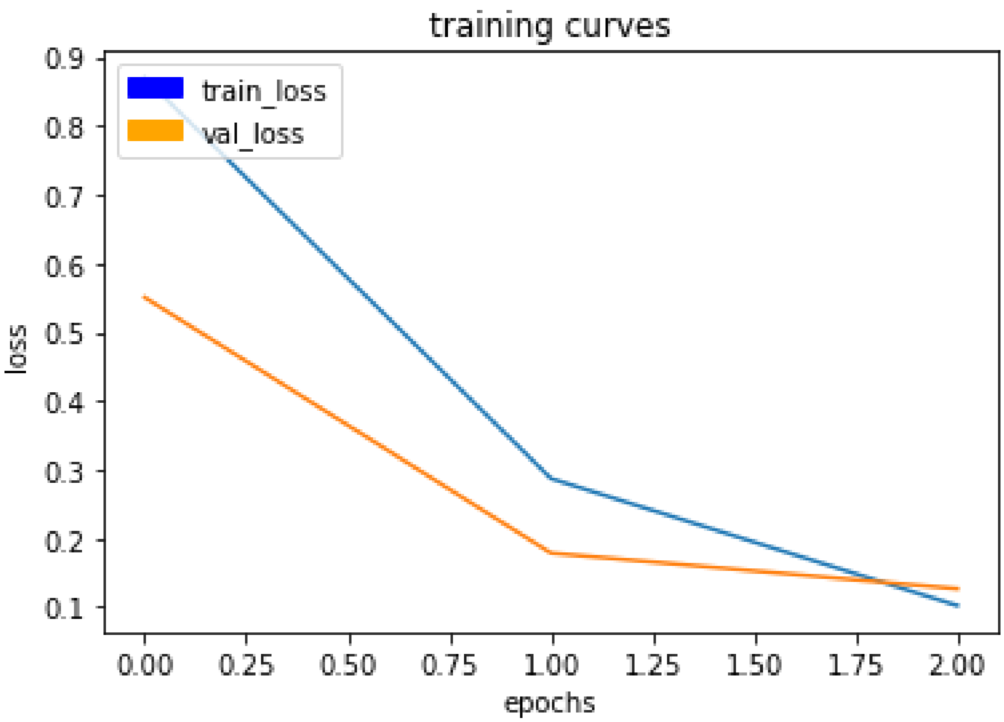
```
Epoch 1/55
49/50 [============================>.] - ETA: 0s - loss: 0.8773
```



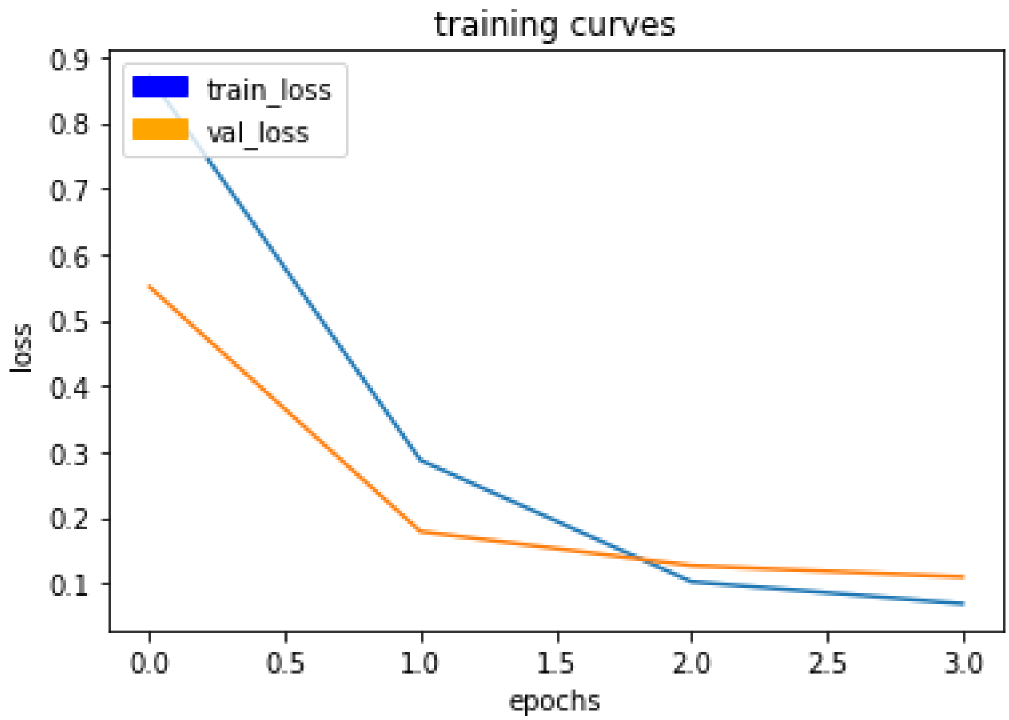

```
50/50 [==============================] - 41s - loss: 0.8704 - val_loss: 0.5507
Epoch 2/55
49/50 [============================>.] - ETA: 0s - loss: 0.2895
```



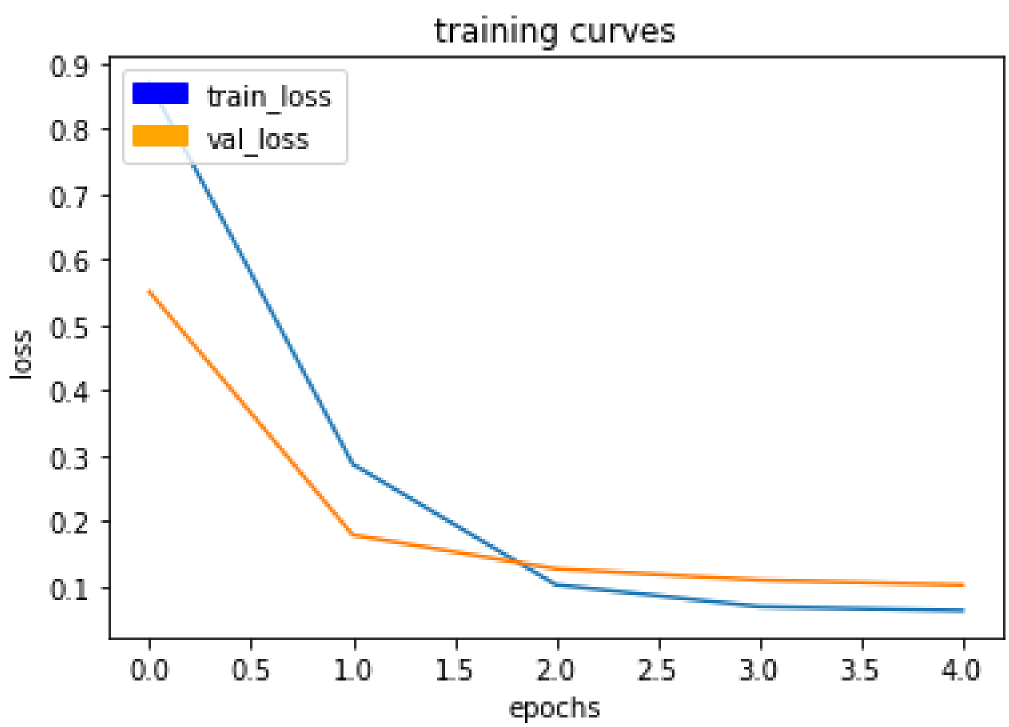
```
50/50 [==============================] - 35s - loss: 0.2867 - val_loss: 0.1784
Epoch 3/55
49/50 [============================>.] - ETA: 0s - loss: 0.1026
```



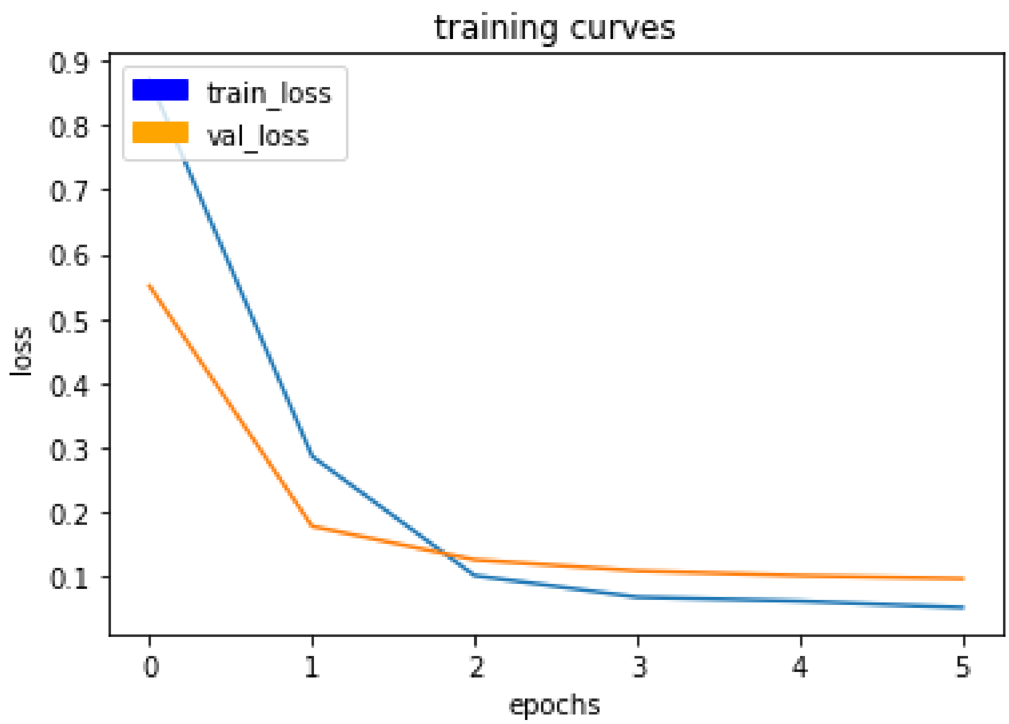

```
50/50 [==============================] - 36s - loss: 0.1020 - val_loss: 0.1265
Epoch 4/55
49/50 [============================>.] - ETA: 0s - loss: 0.0684
```



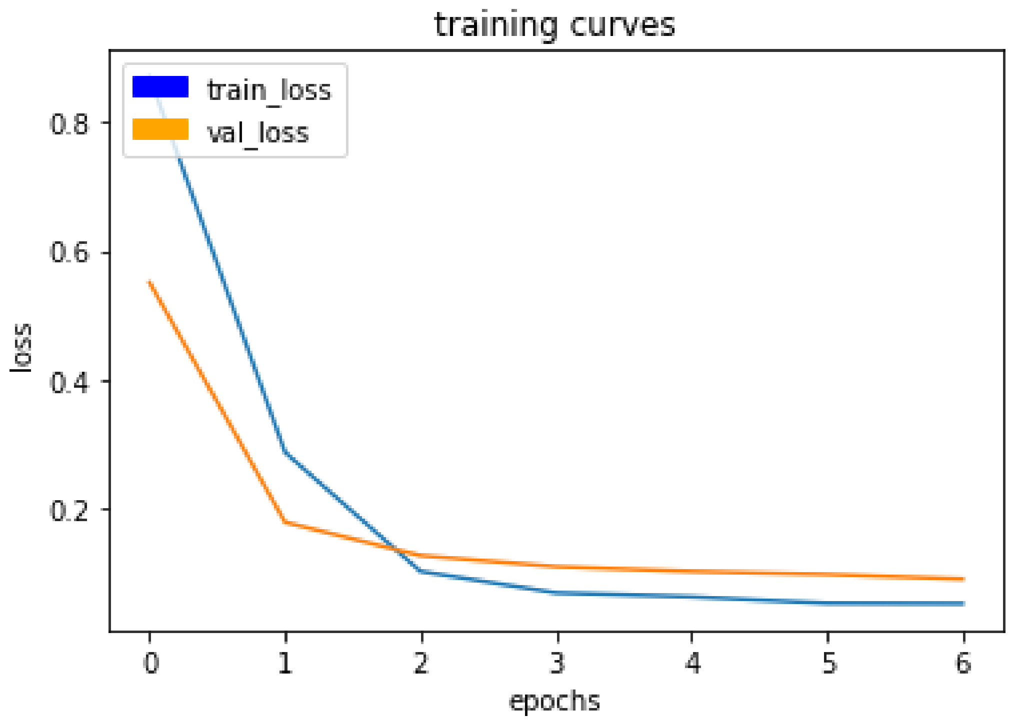

```
50/50 [==============================] - 36s - loss: 0.0685 - val_loss: 0.1093
Epoch 5/55
49/50 [============================>.] - ETA: 0s - loss: 0.0626
```



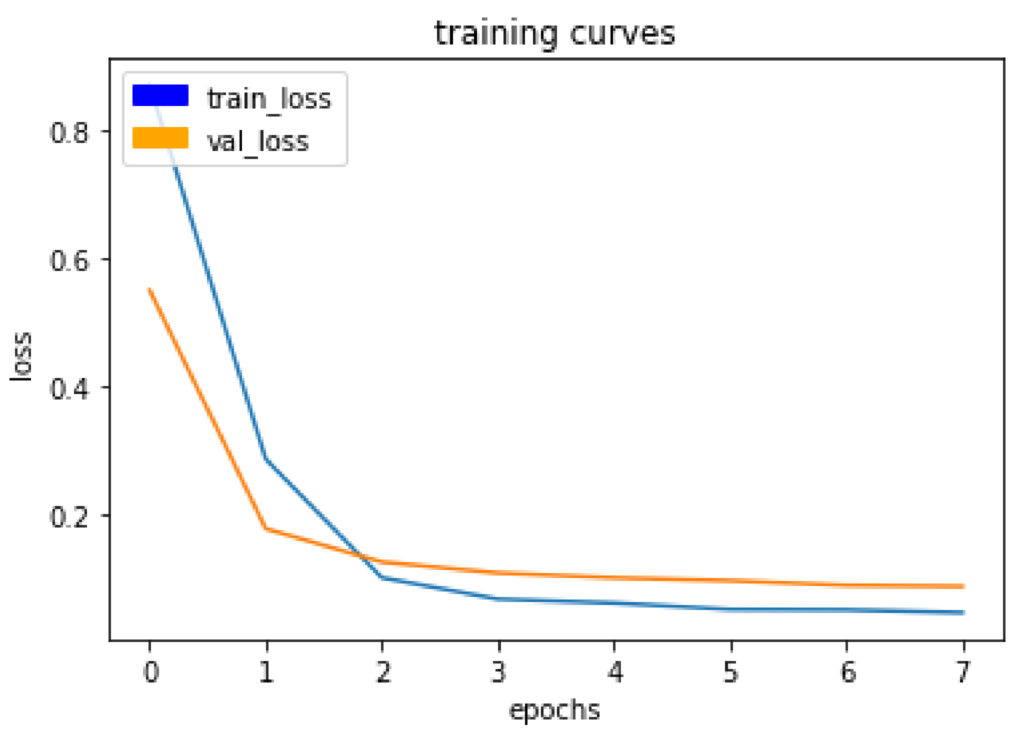
```
50/50 [==============================] - 36s - loss: 0.0624 - val_loss: 0.1018
Epoch 6/55
49/50 [============================>.] - ETA: 0s - loss: 0.0523
```



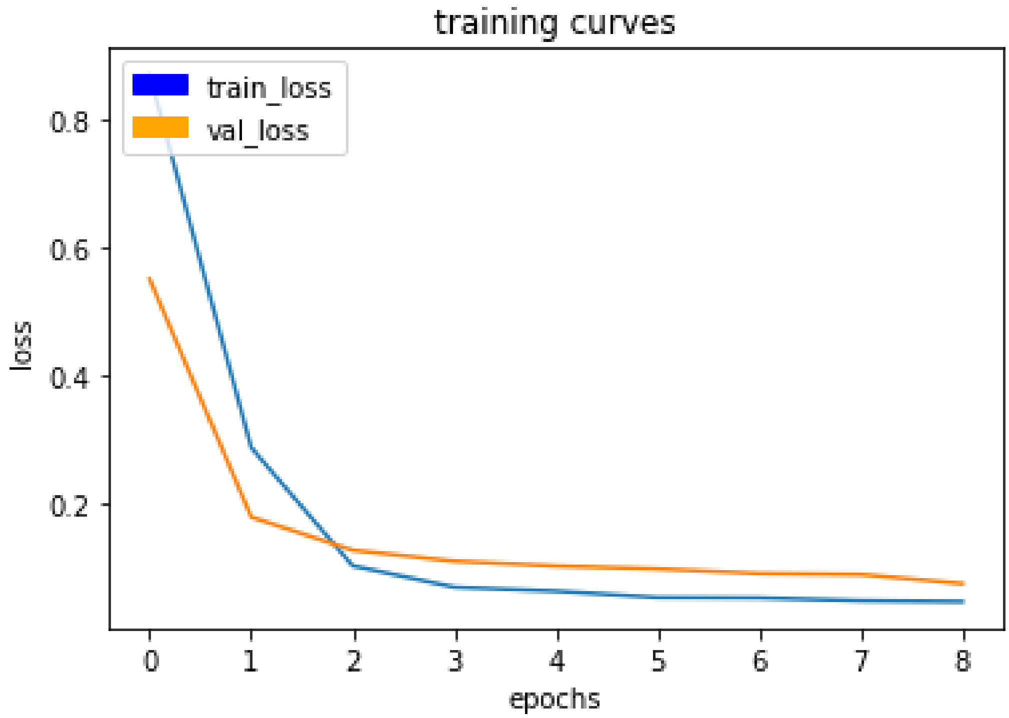

```
50/50 [==============================] - 36s - loss: 0.0524 - val_loss: 0.0973
Epoch 7/55
49/50 [============================>.] - ETA: 0s - loss: 0.0521
```



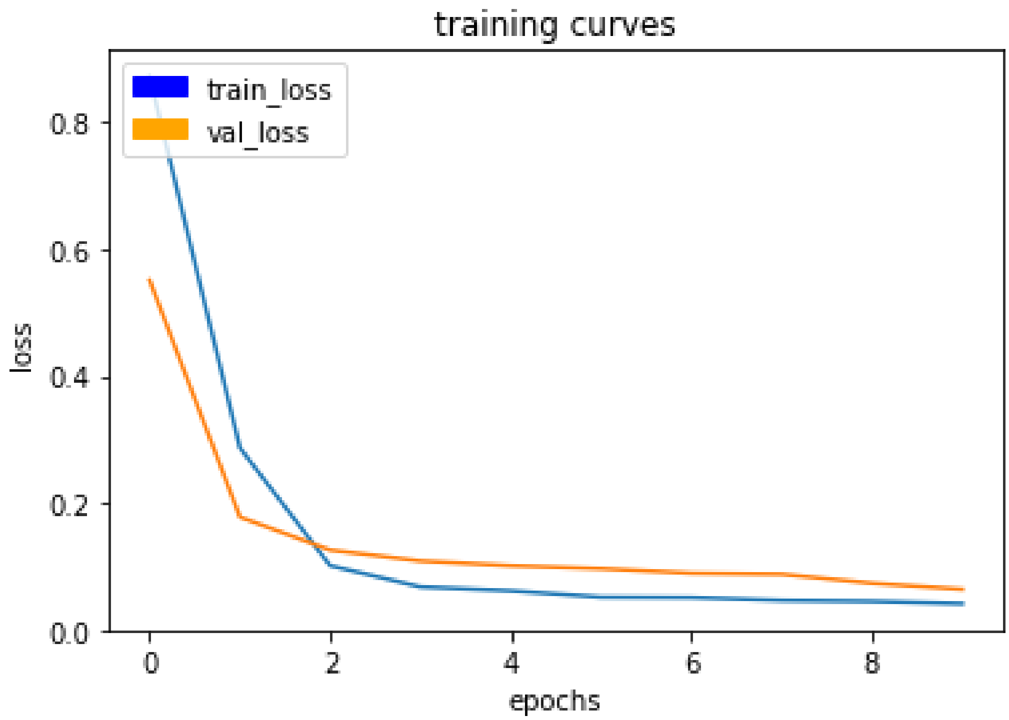

```
50/50 [==============================] - 36s - loss: 0.0518 - val_loss: 0.0901
Epoch 8/55
49/50 [============================>.] - ETA: 0s - loss: 0.0471
```



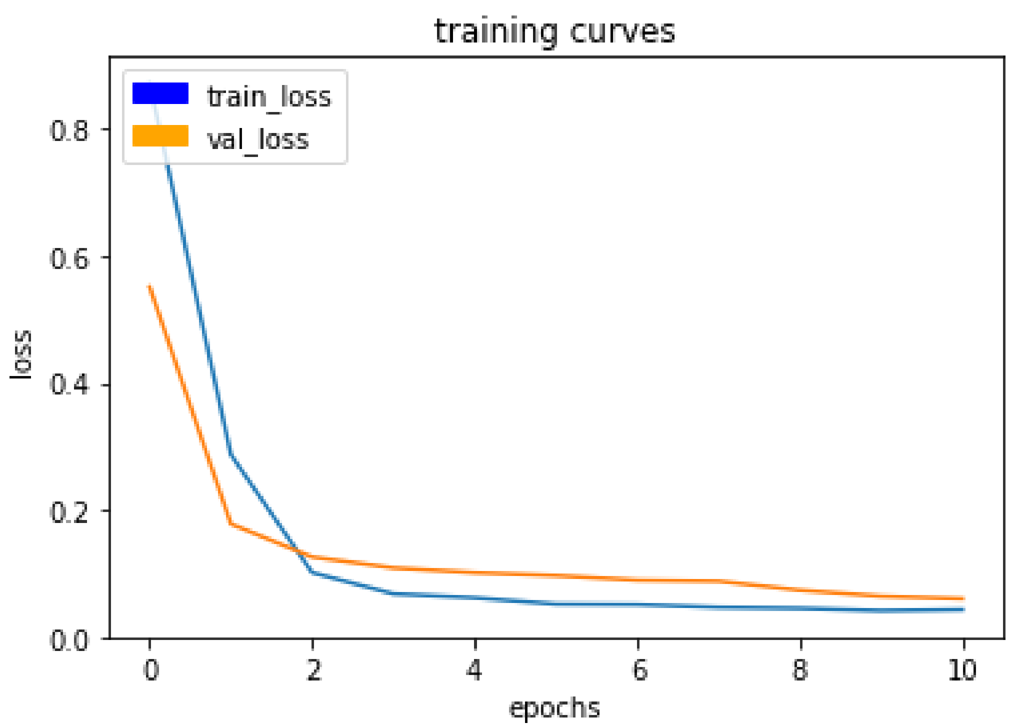
```
50/50 [==============================] - 36s - loss: 0.0474 - val_loss: 0.0885
Epoch 9/55
49/50 [============================>.] - ETA: 0s - loss: 0.0459
```



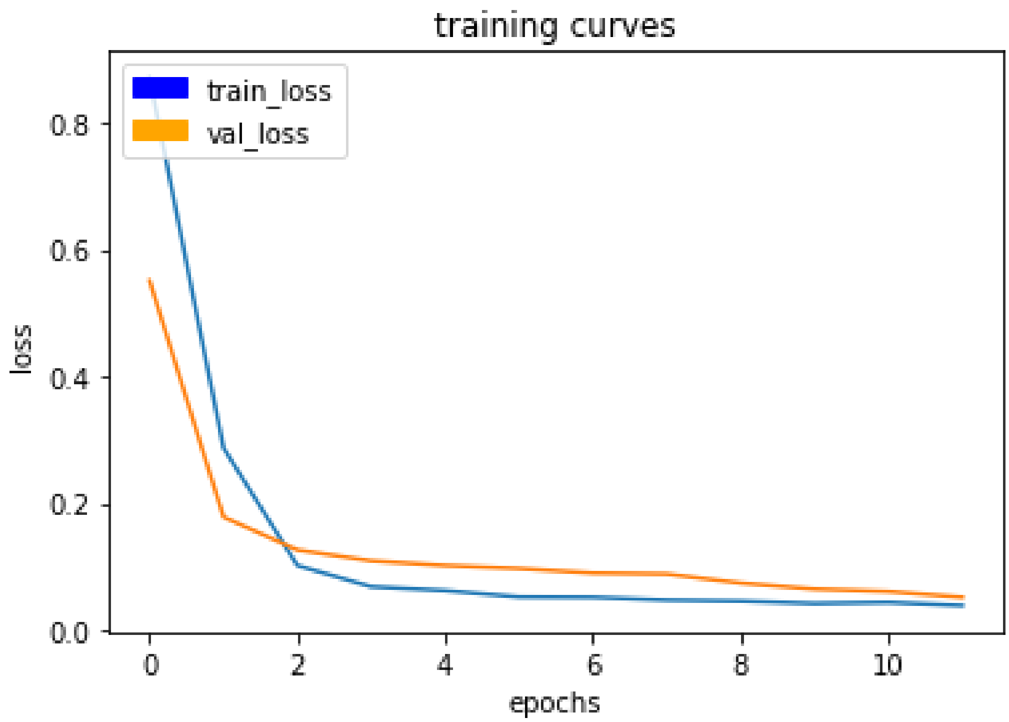

```
50/50 [==============================] - 36s - loss: 0.0458 - val_loss: 0.0743
Epoch 10/55
49/50 [============================>.] - ETA: 0s - loss: 0.0423
```



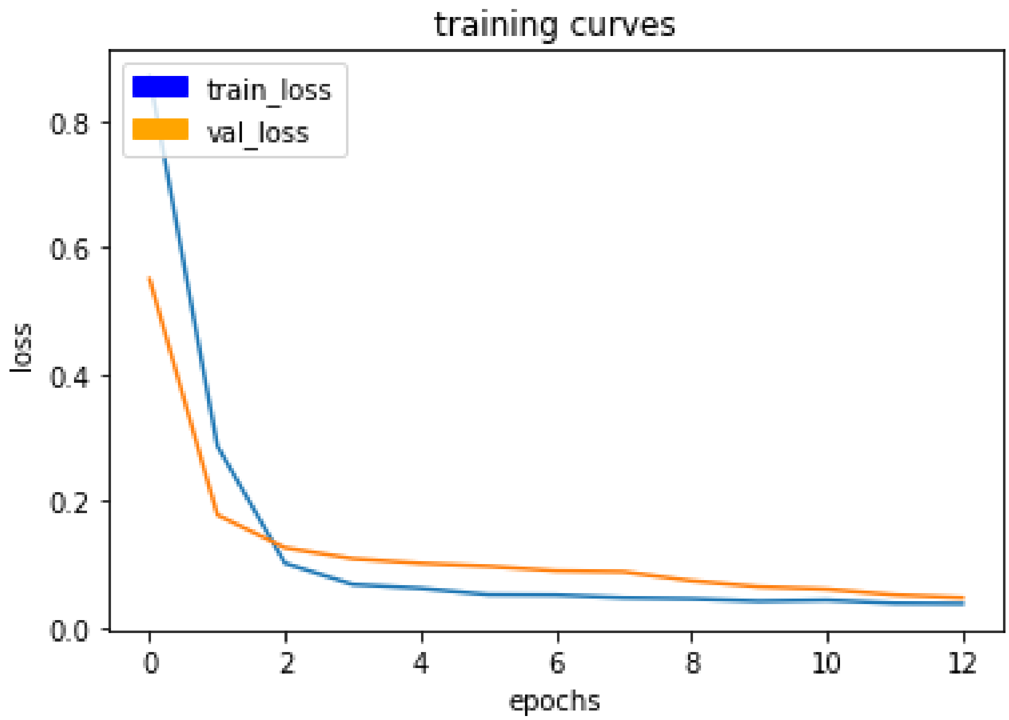

```
50/50 [==============================] - 36s - loss: 0.0421 - val_loss: 0.0648
Epoch 11/55
49/50 [============================>.] - ETA: 0s - loss: 0.0438
```



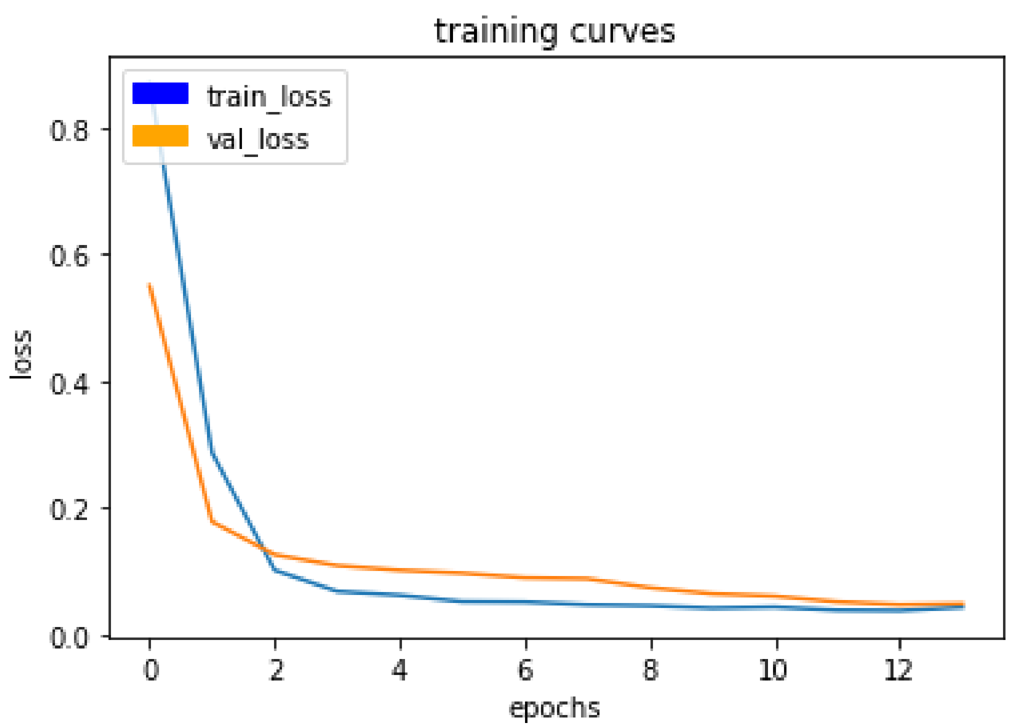
```
50/50 [==============================] - 36s - loss: 0.0438 - val_loss: 0.0610
Epoch 12/55
49/50 [============================>.] - ETA: 0s - loss: 0.0390
```



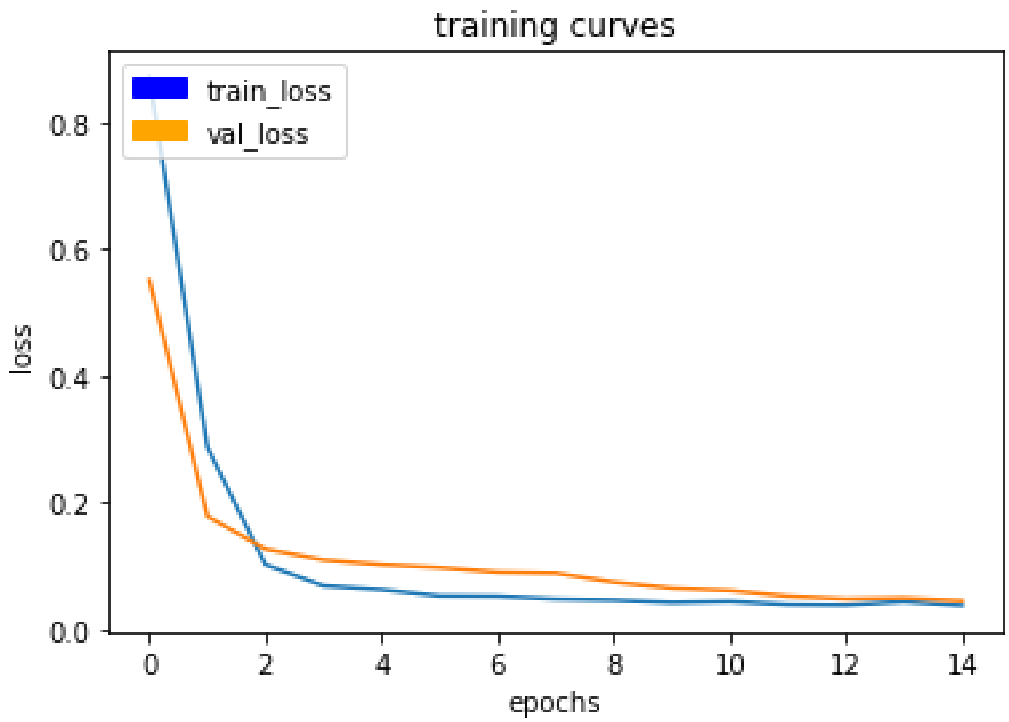

```
50/50 [==============================] - 36s - loss: 0.0389 - val_loss: 0.0520
Epoch 13/55
49/50 [============================>.] - ETA: 0s - loss: 0.0383
```



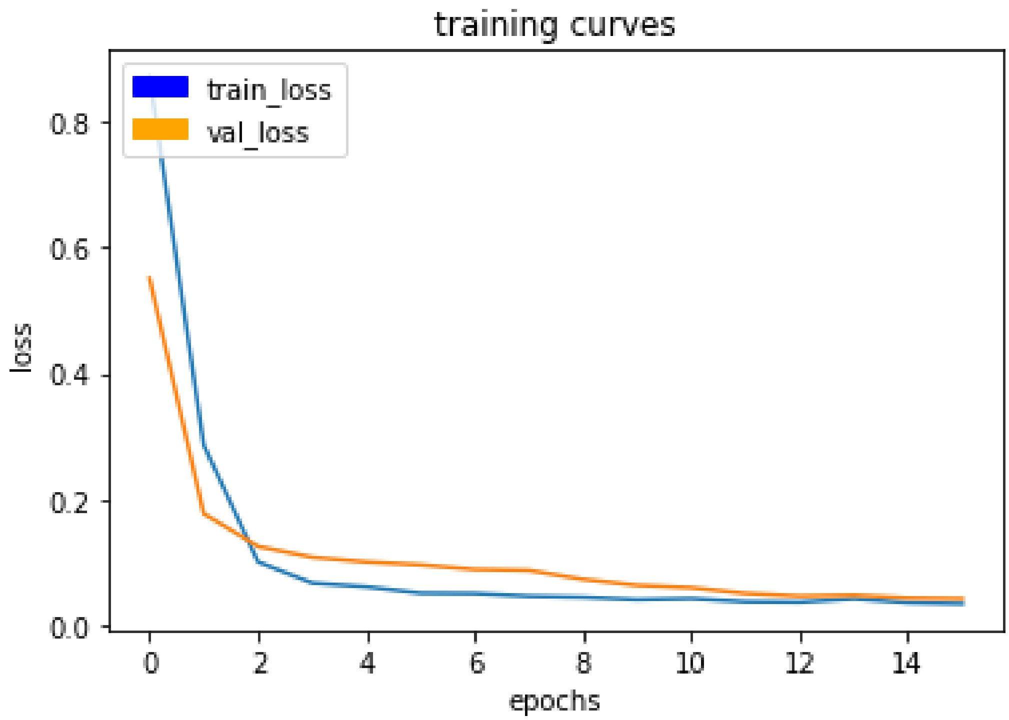

```
50/50 [==============================] - 36s - loss: 0.0381 - val_loss: 0.0476
Epoch 14/55
49/50 [============================>.] - ETA: 0s - loss: 0.0431
```



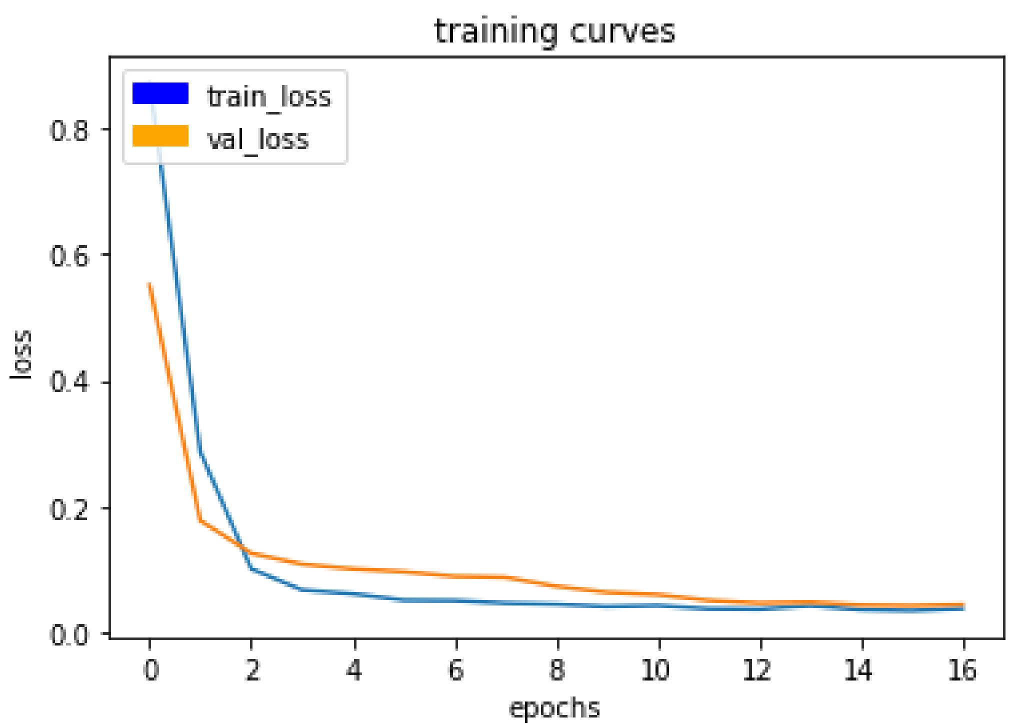
```
50/50 [==============================] - 36s - loss: 0.0434 - val_loss: 0.0491
Epoch 15/55
49/50 [============================>.] - ETA: 0s - loss: 0.0374
```



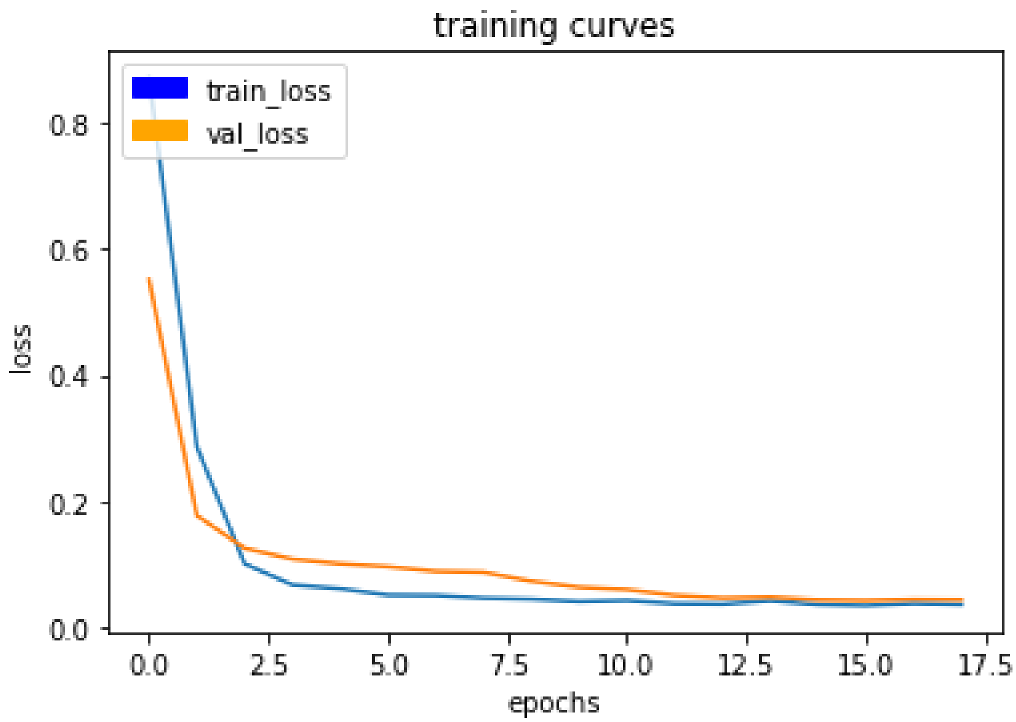

```
50/50 [==============================] - 36s - loss: 0.0374 - val_loss: 0.0447
Epoch 16/55
49/50 [============================>.] - ETA: 0s - loss: 0.0356
```



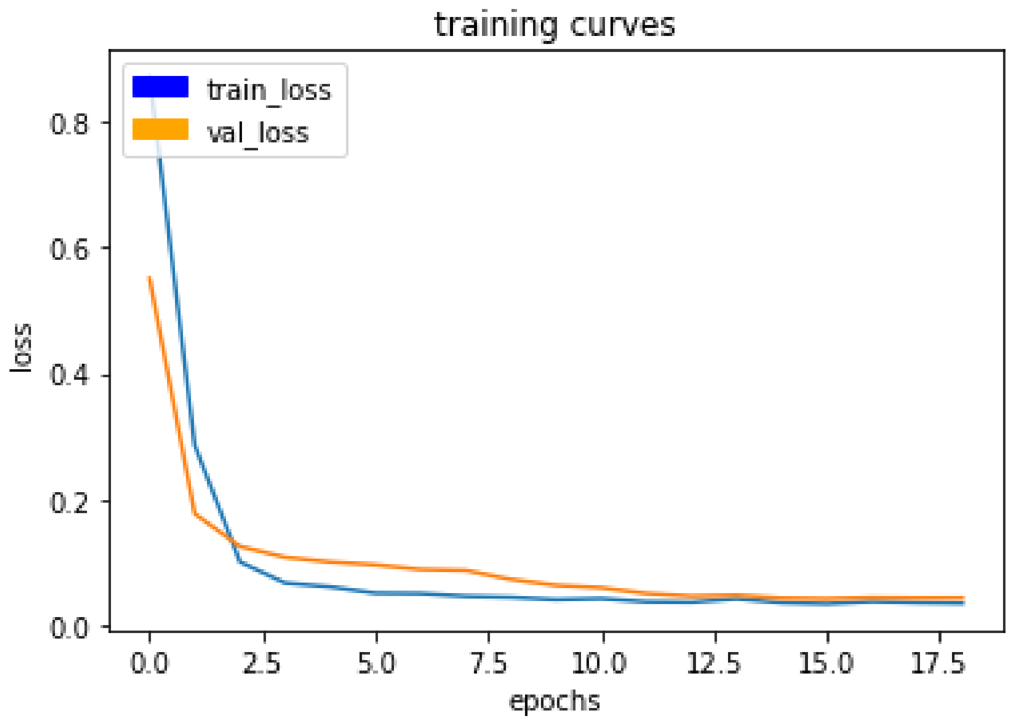

```
50/50 [==============================] - 36s - loss: 0.0357 - val_loss: 0.0438
Epoch 17/55
49/50 [============================>.] - ETA: 0s - loss: 0.0393
```



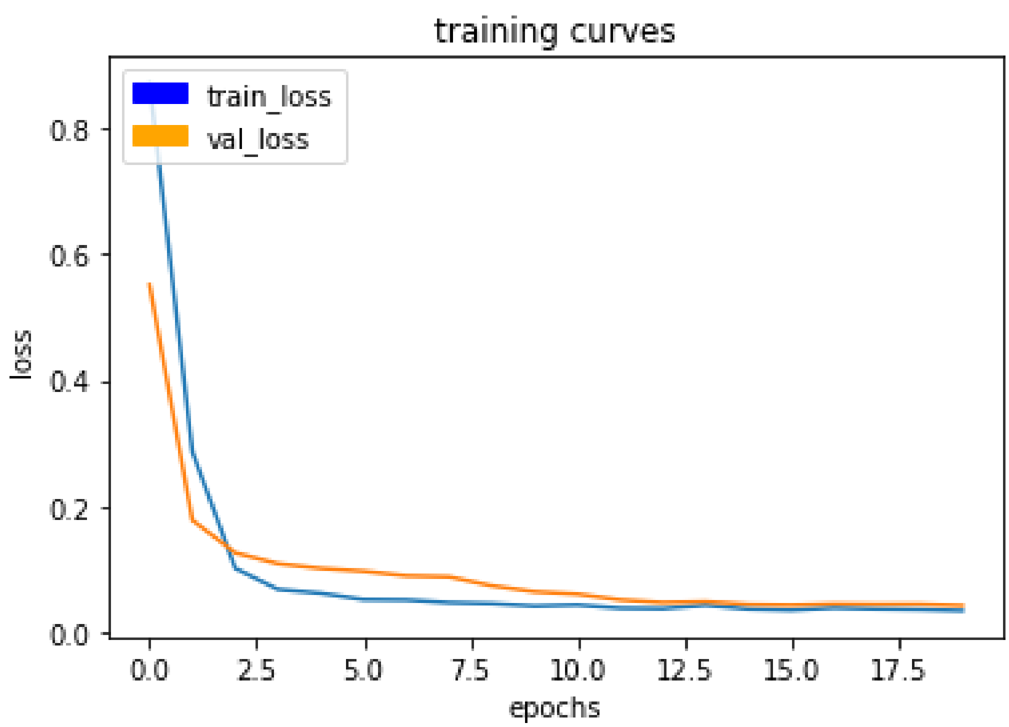
```
50/50 [==============================] - 36s - loss: 0.0391 - val_loss: 0.0455
Epoch 18/55
49/50 [============================>.] - ETA: 0s - loss: 0.0377
```



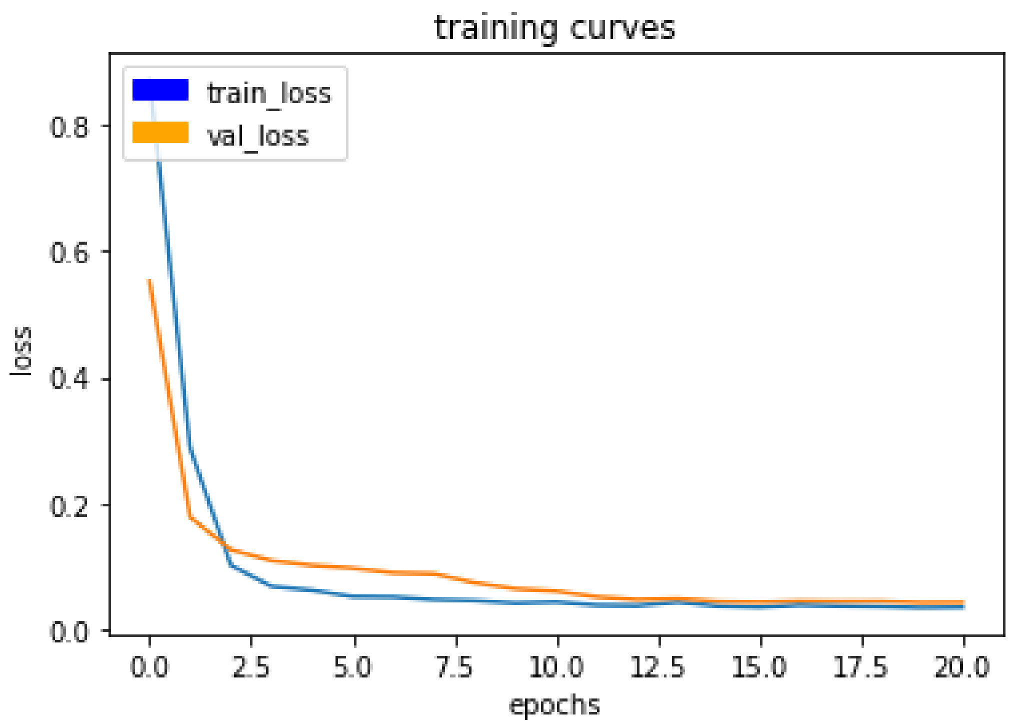

```
50/50 [==============================] - 36s - loss: 0.0375 - val_loss: 0.0450
Epoch 19/55
49/50 [============================>.] - ETA: 0s - loss: 0.0362
```



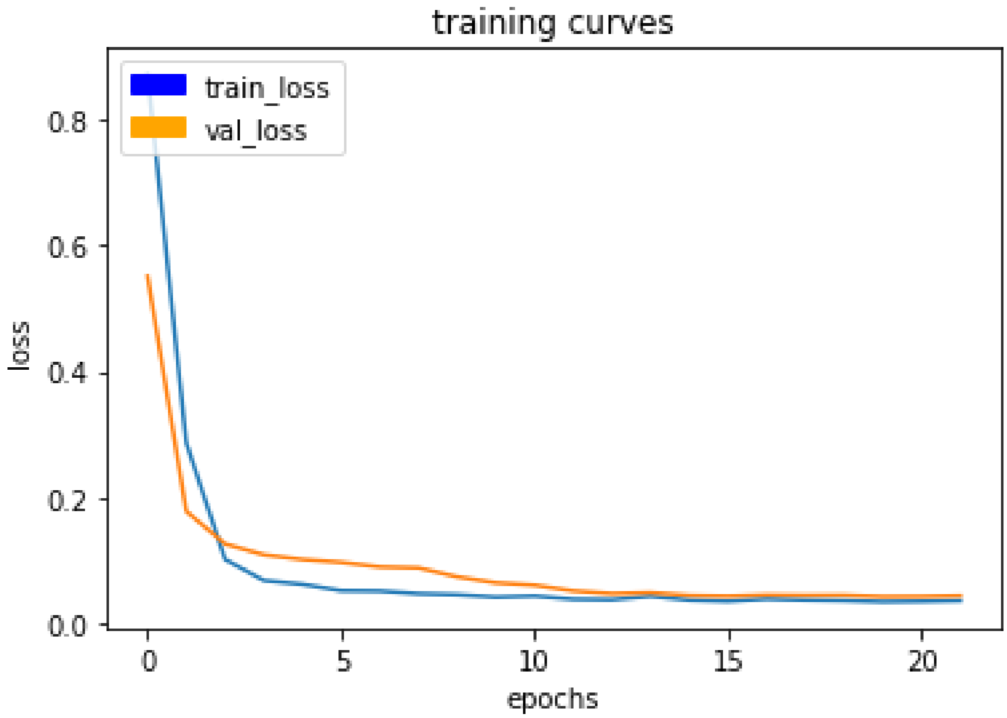

```
50/50 [==============================] - 36s - loss: 0.0366 - val_loss: 0.0452
Epoch 20/55
49/50 [============================>.] - ETA: 0s - loss: 0.0349
```



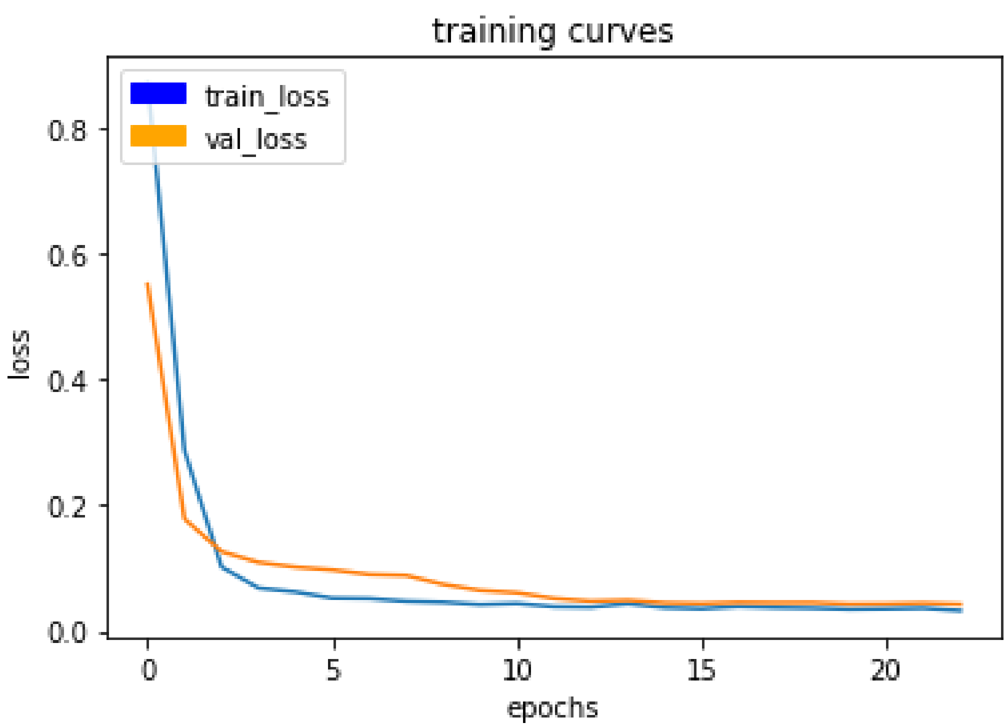
```
50/50 [==============================] - 36s - loss: 0.0348 - val_loss: 0.0430
Epoch 21/55
49/50 [============================>.] - ETA: 0s - loss: 0.0356
```



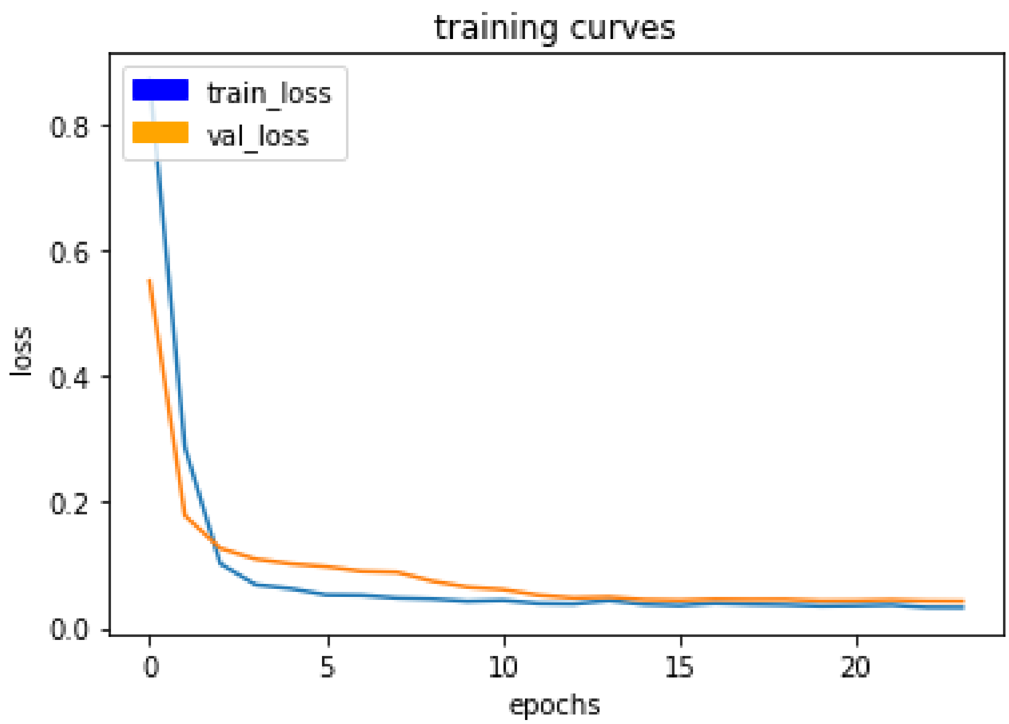

```
50/50 [==============================] - 36s - loss: 0.0355 - val_loss: 0.0433
Epoch 22/55
49/50 [============================>.] - ETA: 0s - loss: 0.0364
```



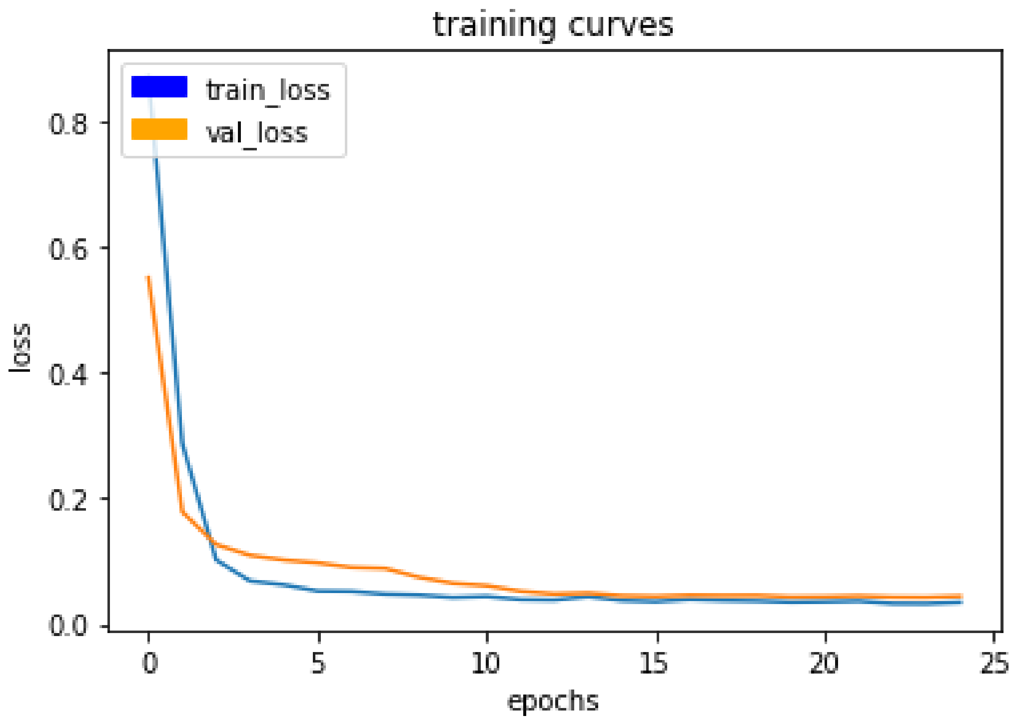

```
50/50 [==============================] - 36s - loss: 0.0365 - val_loss: 0.0444
Epoch 23/55
49/50 [============================>.] - ETA: 0s - loss: 0.0326
```



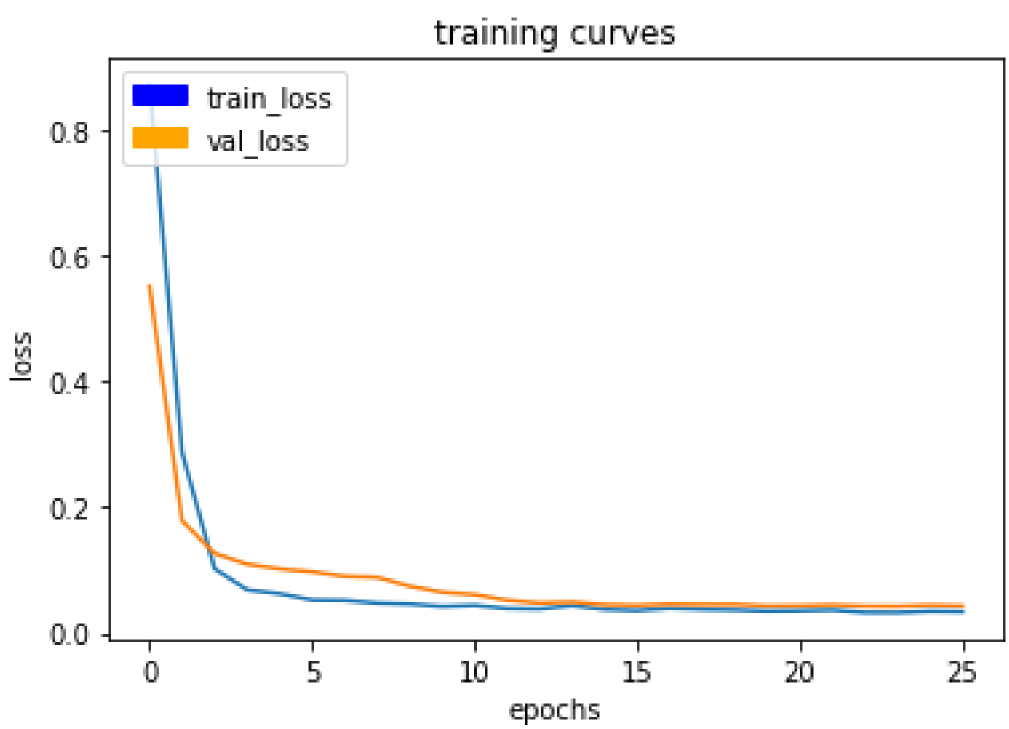
```
50/50 [==============================] - 35s - loss: 0.0326 - val_loss: 0.0428
Epoch 24/55
49/50 [============================>.] - ETA: 0s - loss: 0.0324
```



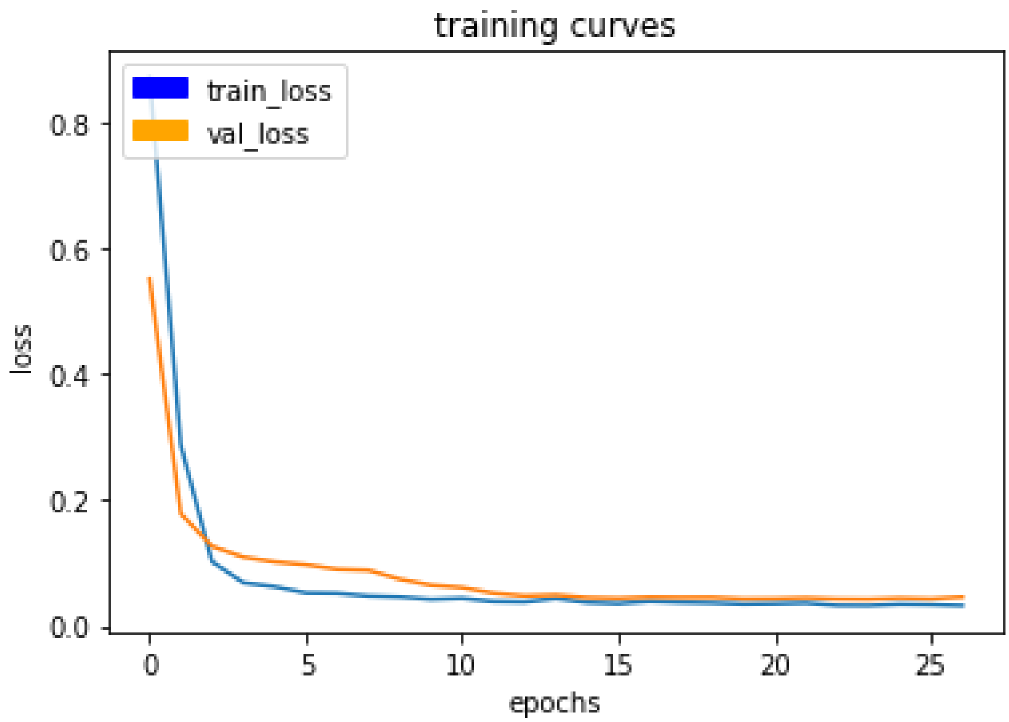

```
50/50 [==============================] - 36s - loss: 0.0323 - val_loss: 0.0425
Epoch 25/55
49/50 [============================>.] - ETA: 0s - loss: 0.0344
```



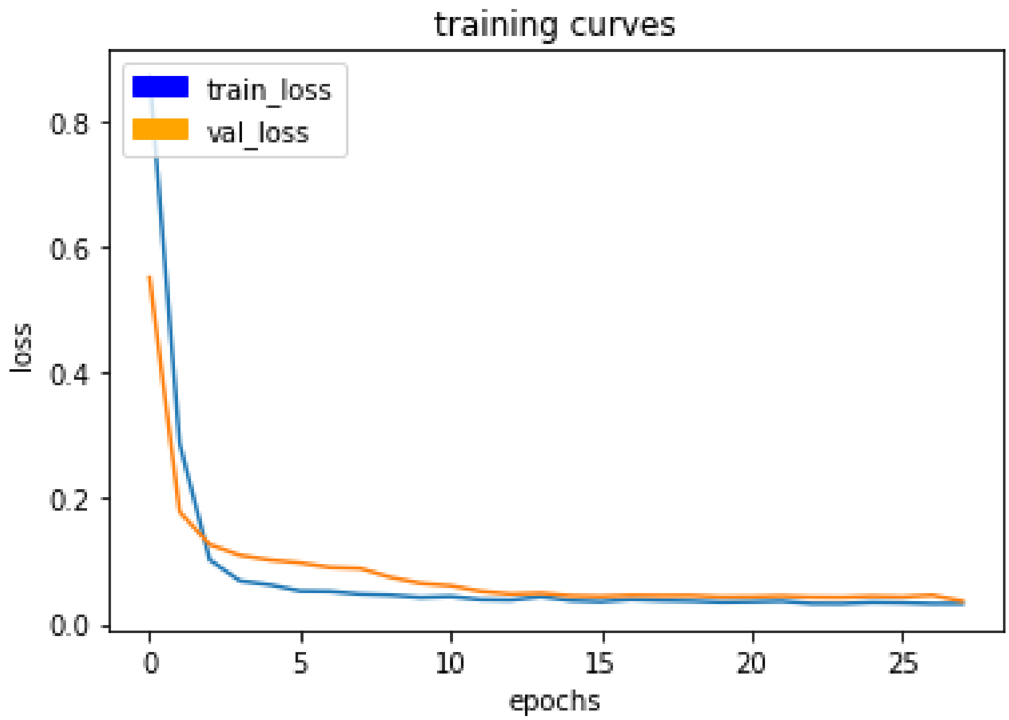

```
50/50 [==============================] - 36s - loss: 0.0345 - val_loss: 0.0440
Epoch 26/55
49/50 [============================>.] - ETA: 0s - loss: 0.0340
```



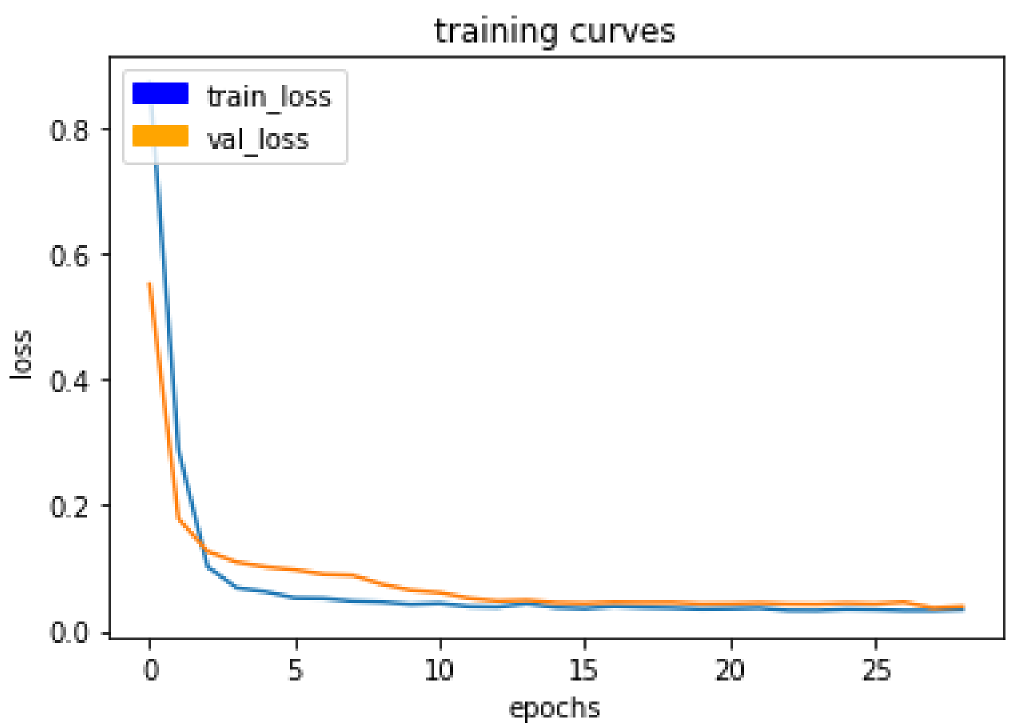
```
50/50 [==============================] - 36s - loss: 0.0339 - val_loss: 0.0430
Epoch 27/55
49/50 [============================>.] - ETA: 0s - loss: 0.0325
```



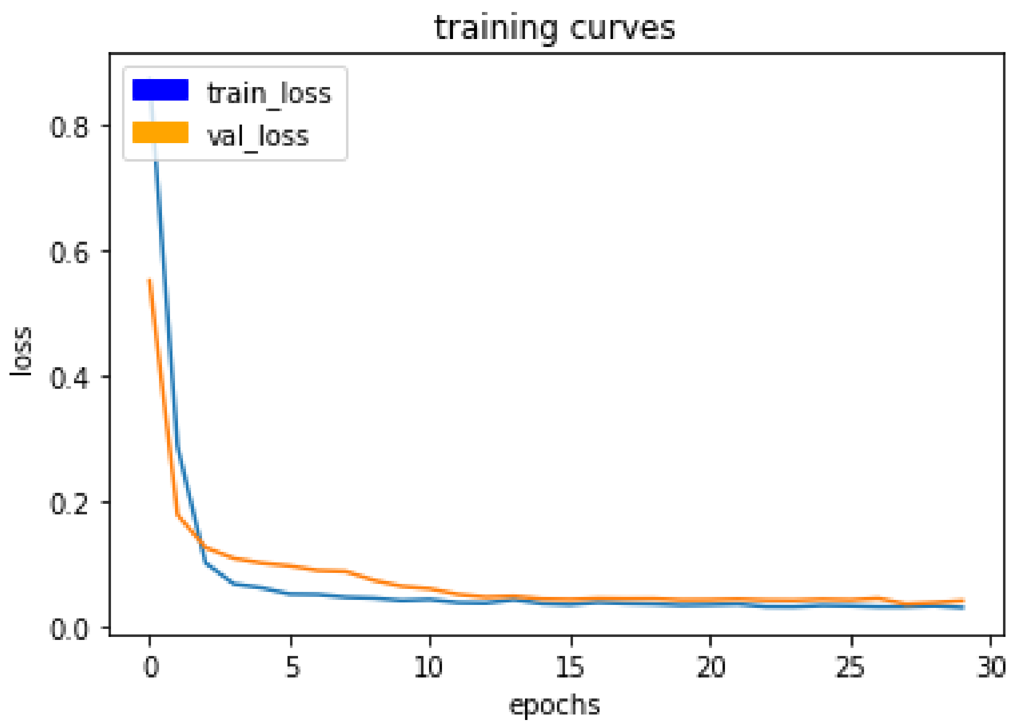

```
50/50 [==============================] - 35s - loss: 0.0324 - val_loss: 0.0457
Epoch 28/55
49/50 [============================>.] - ETA: 0s - loss: 0.0323
```



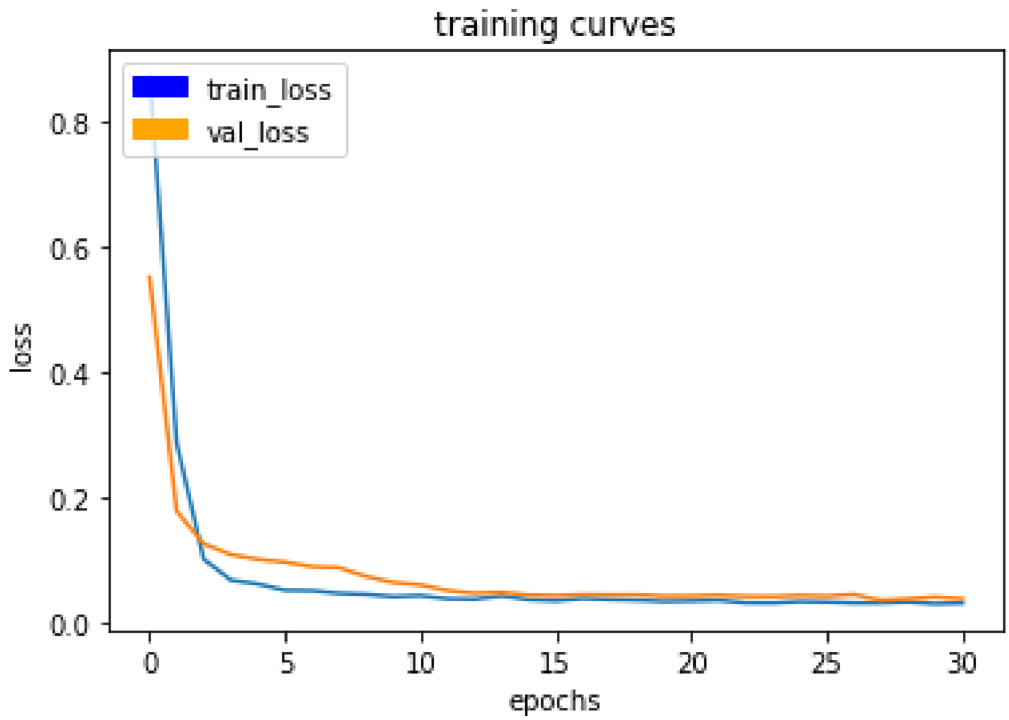

```
50/50 [==============================] - 36s - loss: 0.0323 - val_loss: 0.0366
Epoch 29/55
49/50 [============================>.] - ETA: 0s - loss: 0.0342
```



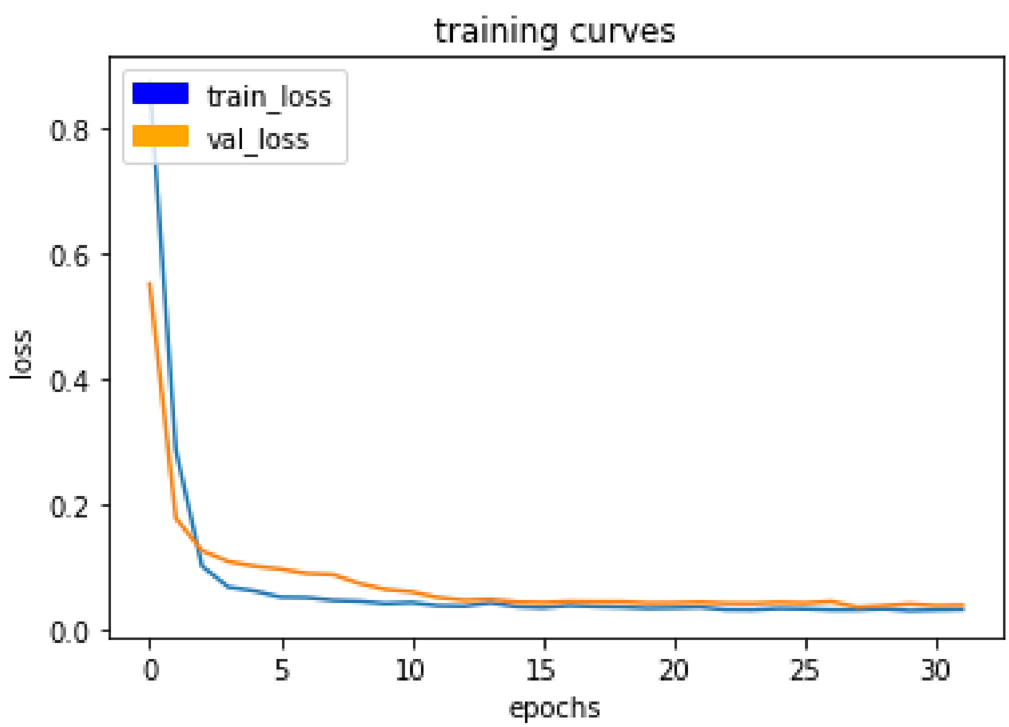
```
50/50 [==============================] - 36s - loss: 0.0339 - val_loss: 0.0389
Epoch 30/55
49/50 [============================>.] - ETA: 0s - loss: 0.0310
```



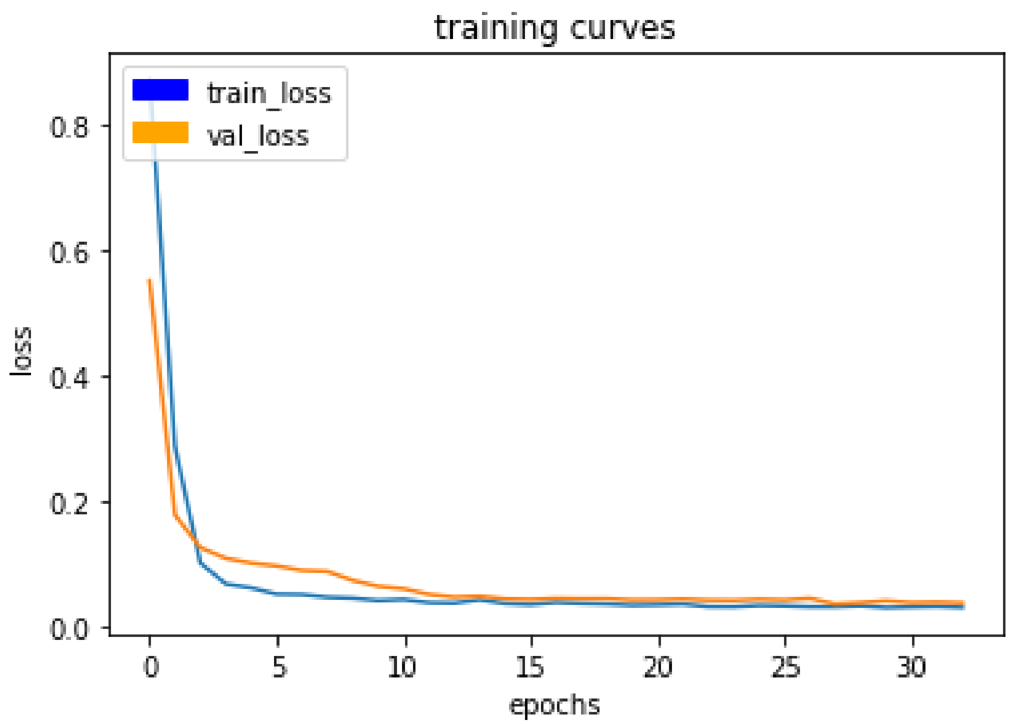

```
50/50 [==============================] - 36s - loss: 0.0311 - val_loss: 0.0419
Epoch 31/55
49/50 [============================>.] - ETA: 0s - loss: 0.0321
```



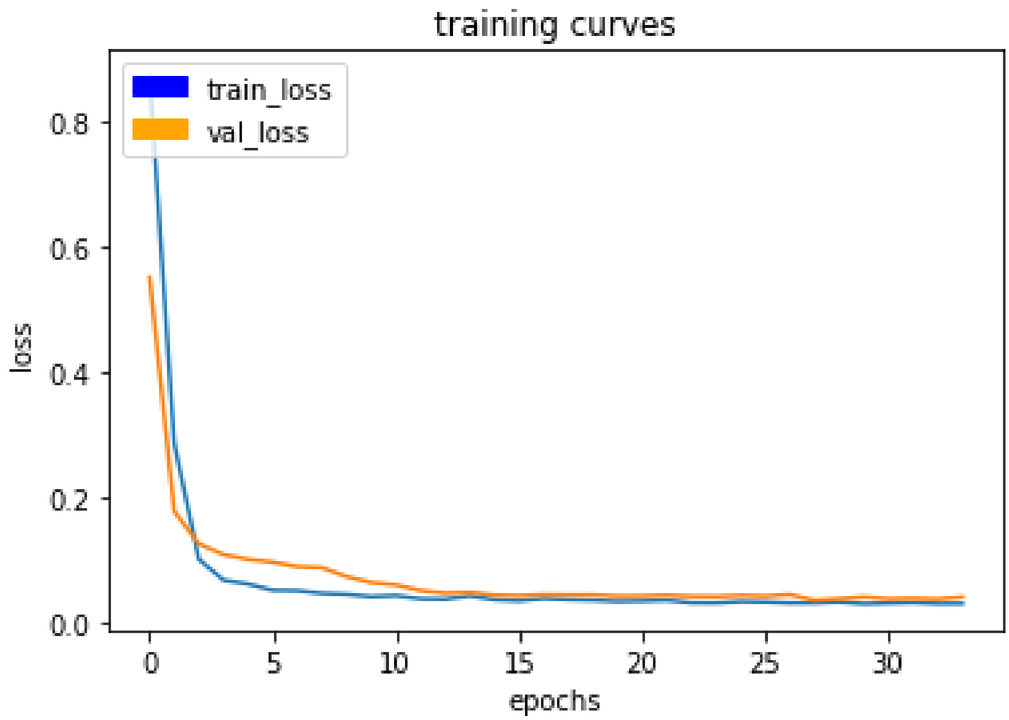

```
50/50 [==============================] - 36s - loss: 0.0321 - val_loss: 0.0389
Epoch 32/55
49/50 [============================>.] - ETA: 0s - loss: 0.0331
```



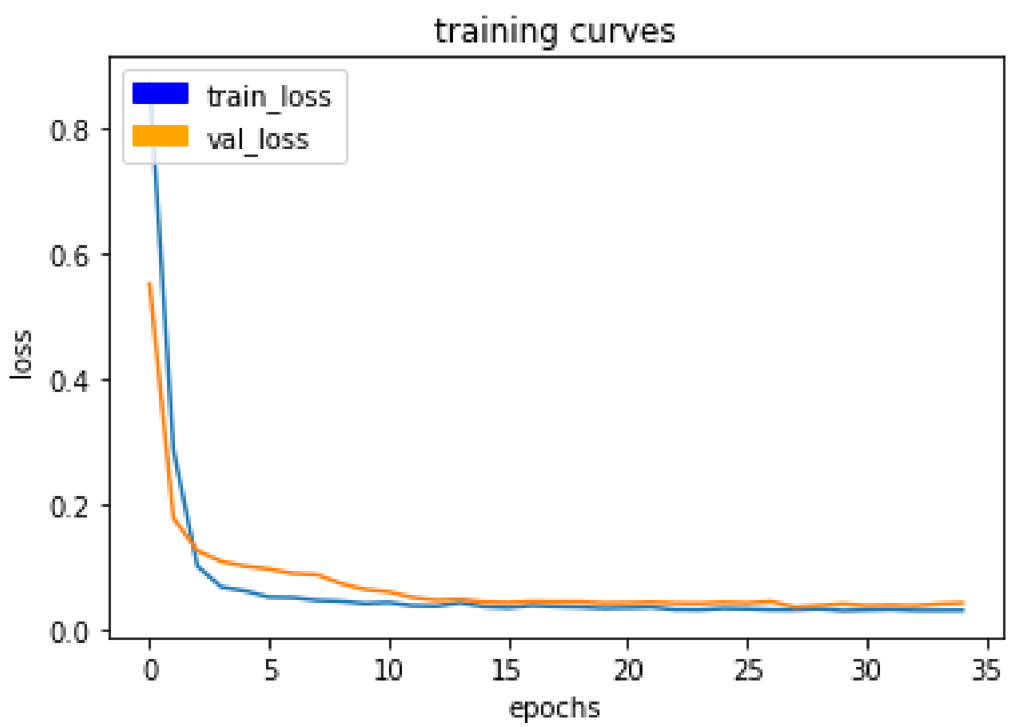
```
50/50 [==============================] - 36s - loss: 0.0329 - val_loss: 0.0397
Epoch 33/55
49/50 [============================>.] - ETA: 0s - loss: 0.0314
```



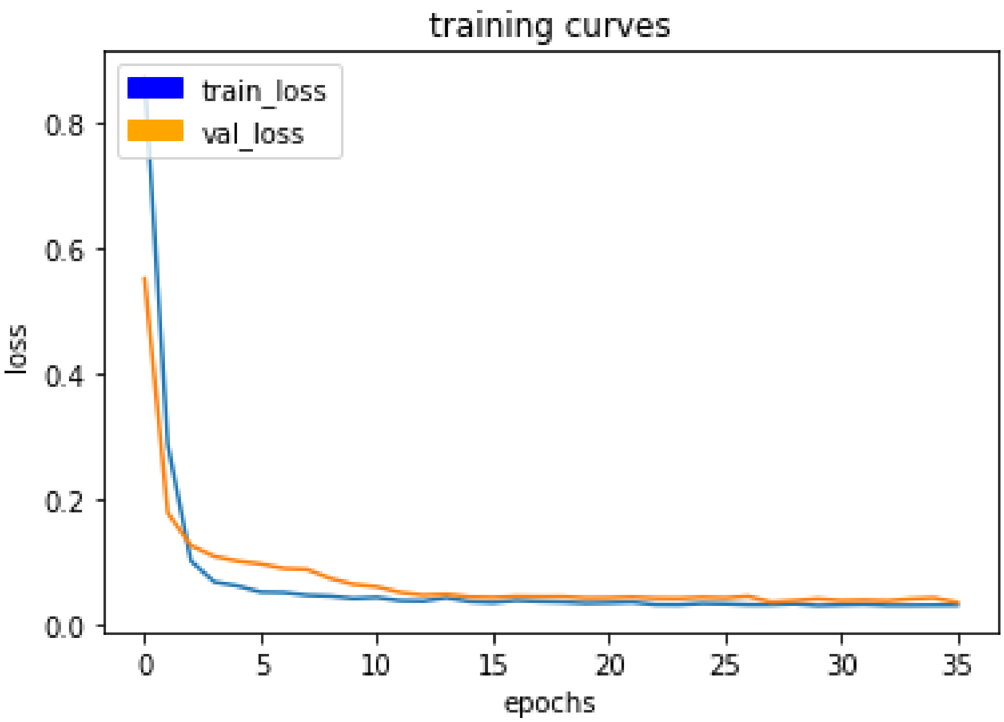

```
50/50 [==============================] - 36s - loss: 0.0313 - val_loss: 0.0385
Epoch 34/55
49/50 [============================>.] - ETA: 0s - loss: 0.0314
```



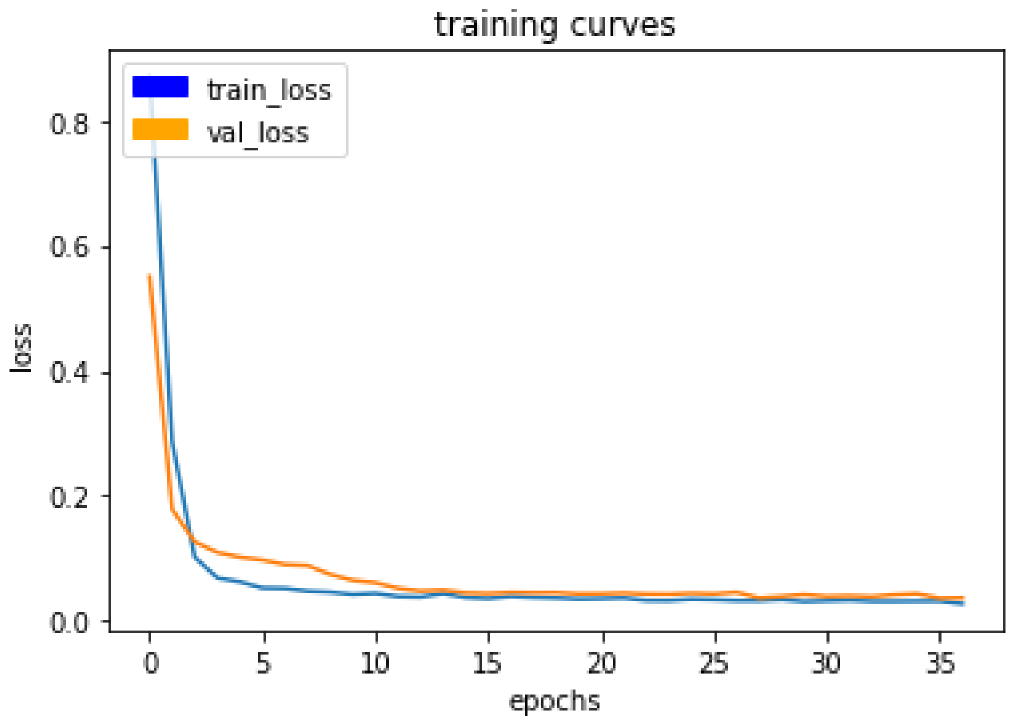

```
50/50 [==============================] - 36s - loss: 0.0313 - val_loss: 0.0417
Epoch 35/55
49/50 [============================>.] - ETA: 0s - loss: 0.0316
```



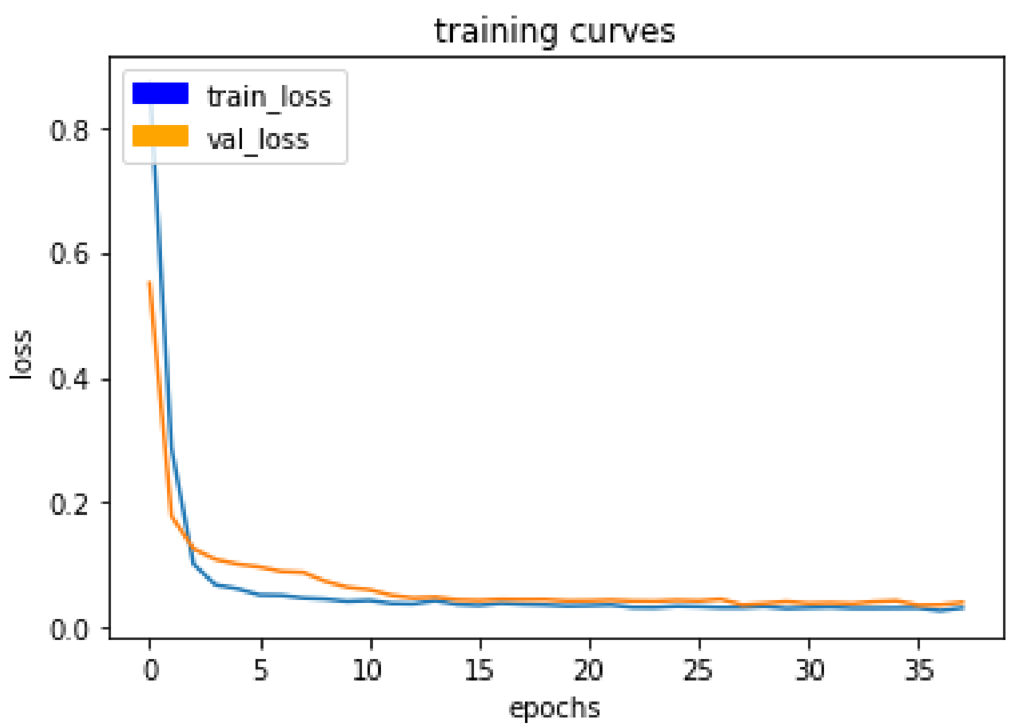
```
50/50 [==============================] - 35s - loss: 0.0316 - val_loss: 0.0432
Epoch 36/55
49/50 [============================>.] - ETA: 0s - loss: 0.0314
```



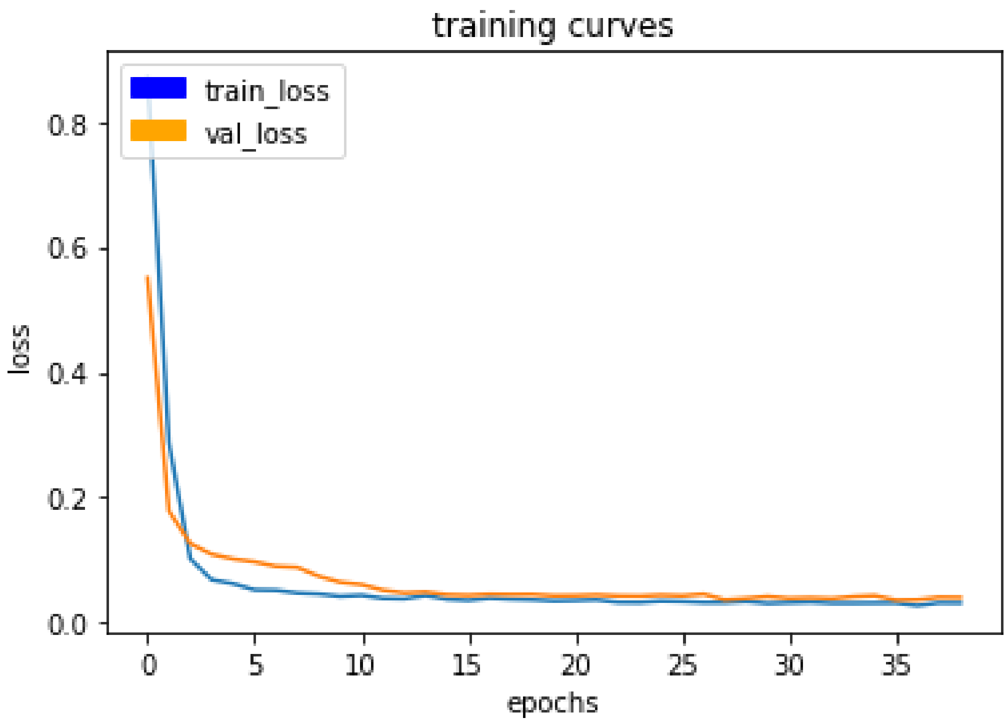

```
50/50 [==============================] - 36s - loss: 0.0317 - val_loss: 0.0361
Epoch 37/55
49/50 [============================>.] - ETA: 0s - loss: 0.0274
```



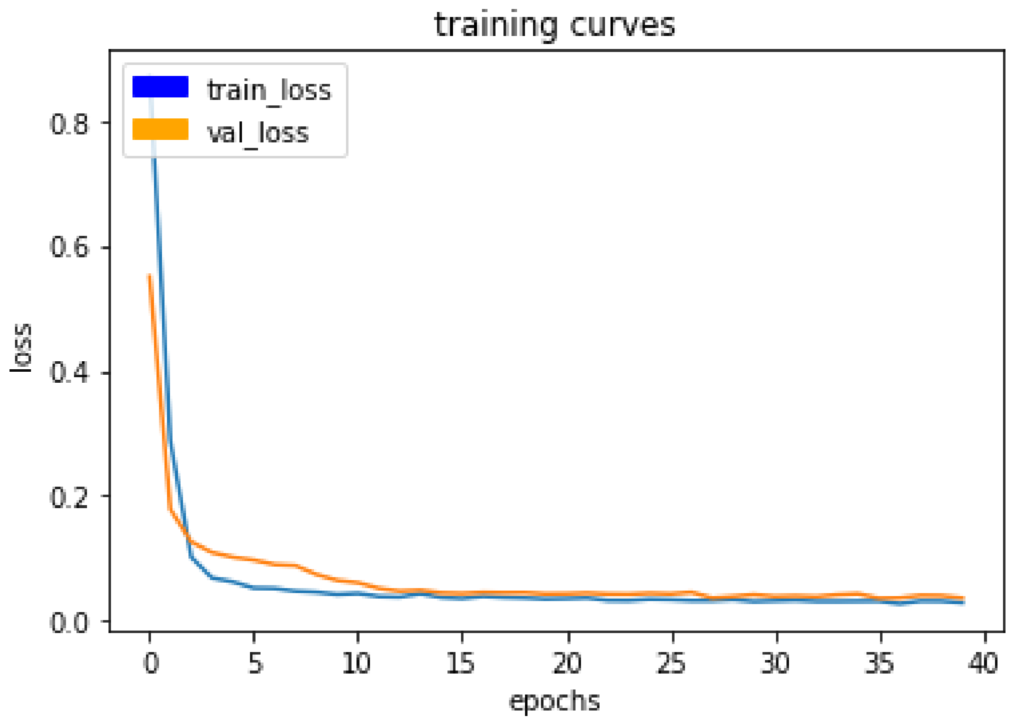

```
50/50 [==============================] - 36s - loss: 0.0274 - val_loss: 0.0367
Epoch 38/55
49/50 [============================>.] - ETA: 0s - loss: 0.0313
```



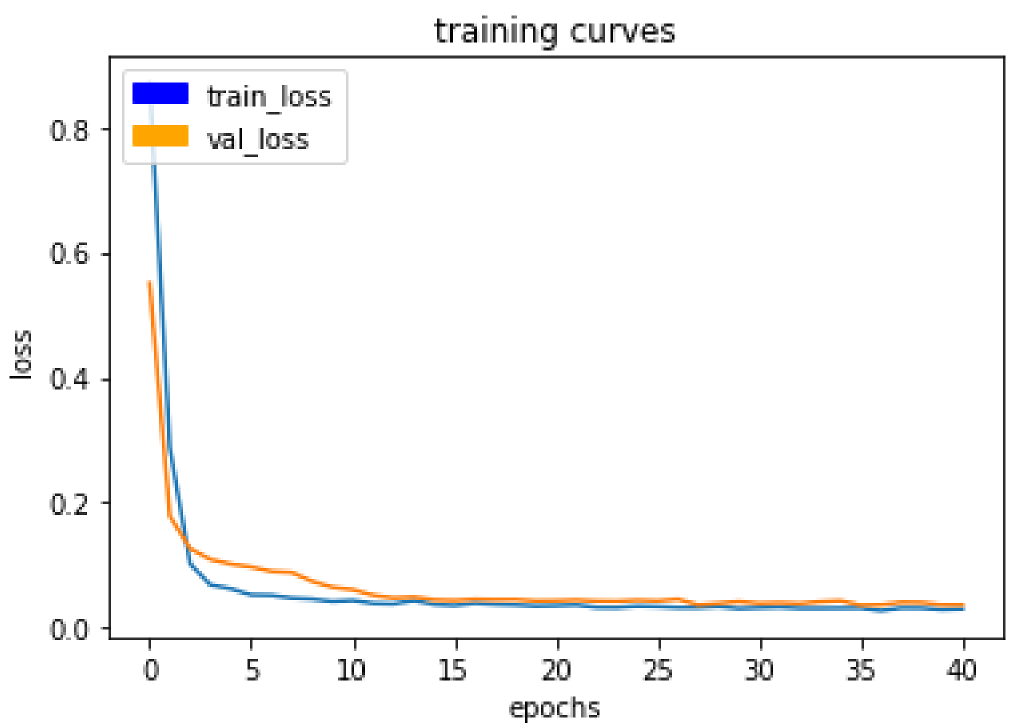
```
50/50 [==============================] - 36s - loss: 0.0318 - val_loss: 0.0405
Epoch 39/55
49/50 [============================>.] - ETA: 0s - loss: 0.0321
```



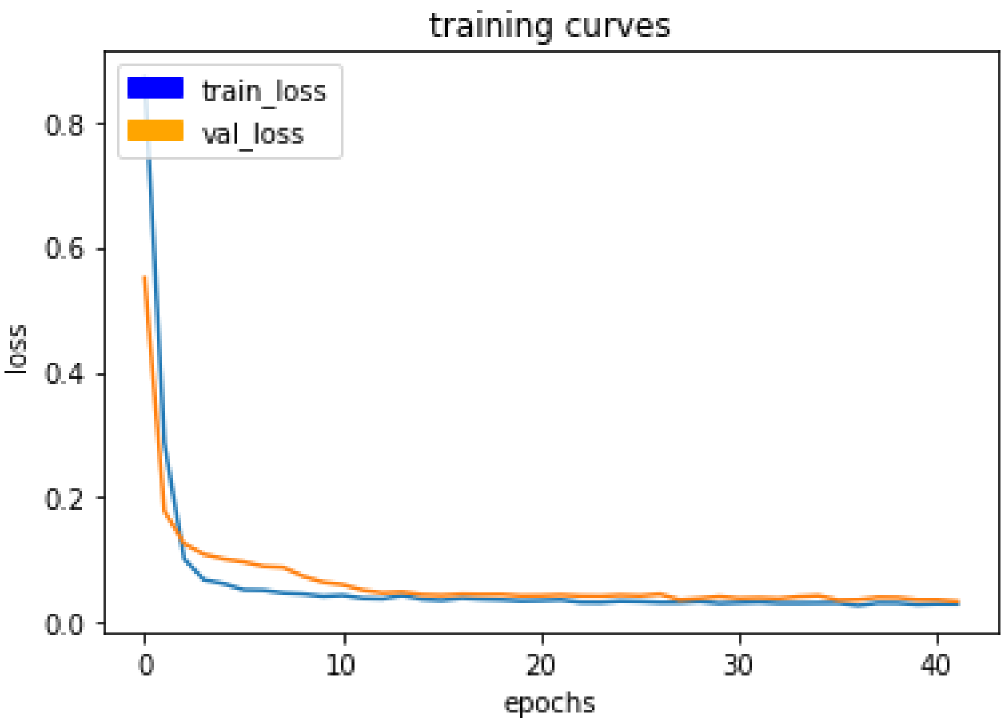

```
50/50 [==============================] - 35s - loss: 0.0320 - val_loss: 0.0401
Epoch 40/55
49/50 [============================>.] - ETA: 0s - loss: 0.0291
```



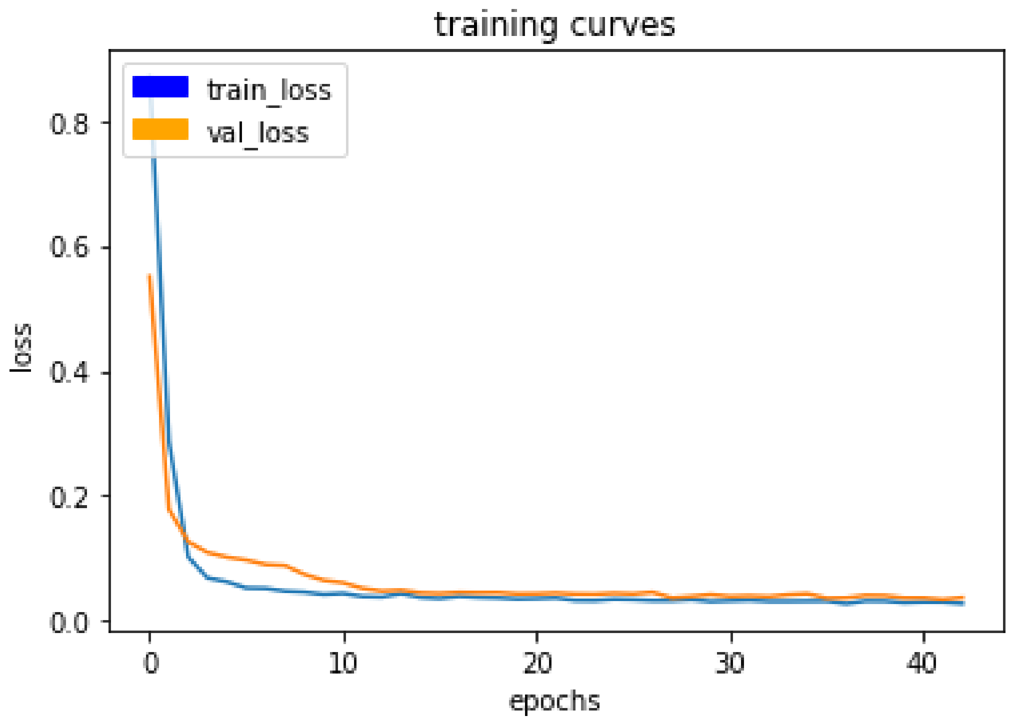

```
50/50 [==============================] - 36s - loss: 0.0291 - val_loss: 0.0366
Epoch 41/55
49/50 [============================>.] - ETA: 0s - loss: 0.0303
```



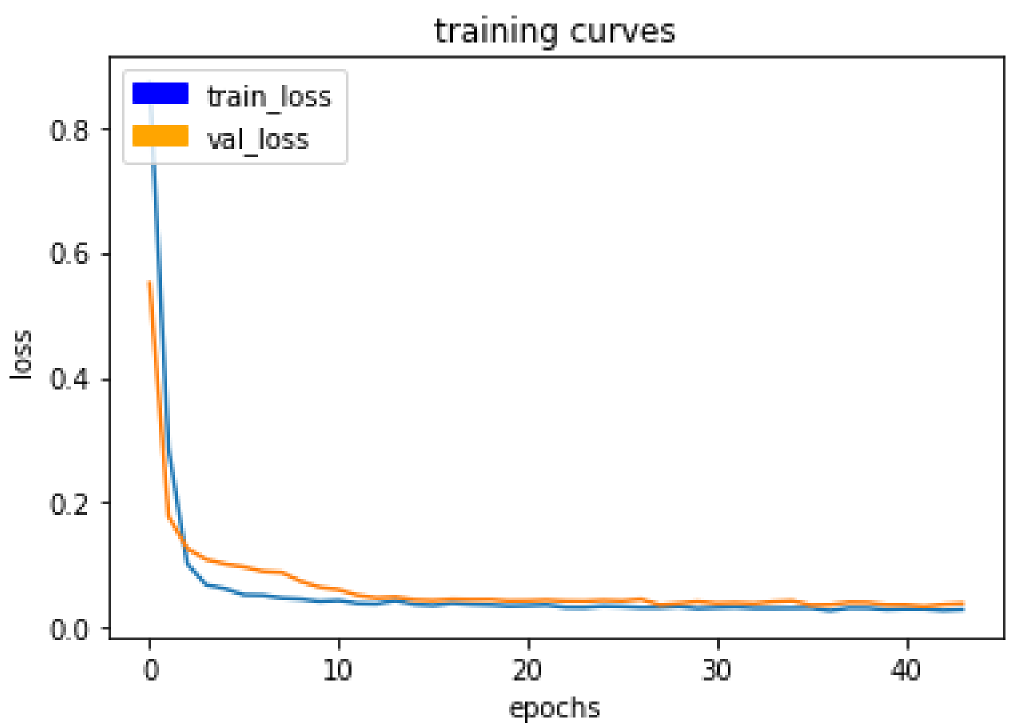
```
50/50 [==============================] - 36s - loss: 0.0302 - val_loss: 0.0366
Epoch 42/55
49/50 [============================>.] - ETA: 0s - loss: 0.0302
```



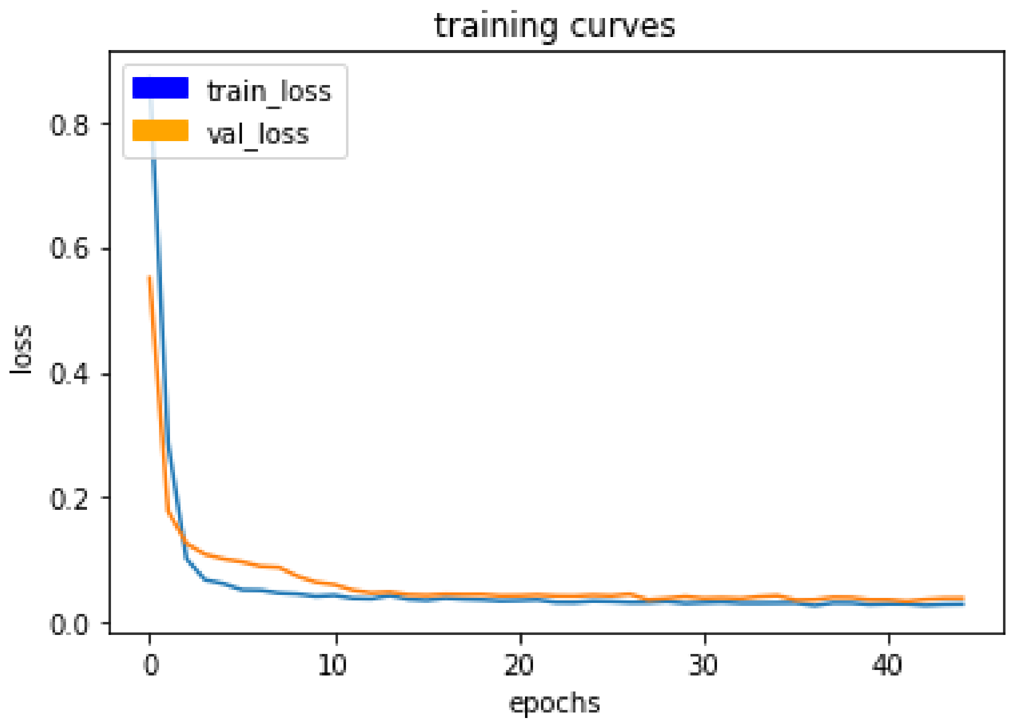

```
50/50 [==============================] - 36s - loss: 0.0300 - val_loss: 0.0341
Epoch 43/55
49/50 [============================>.] - ETA: 0s - loss: 0.0278
```



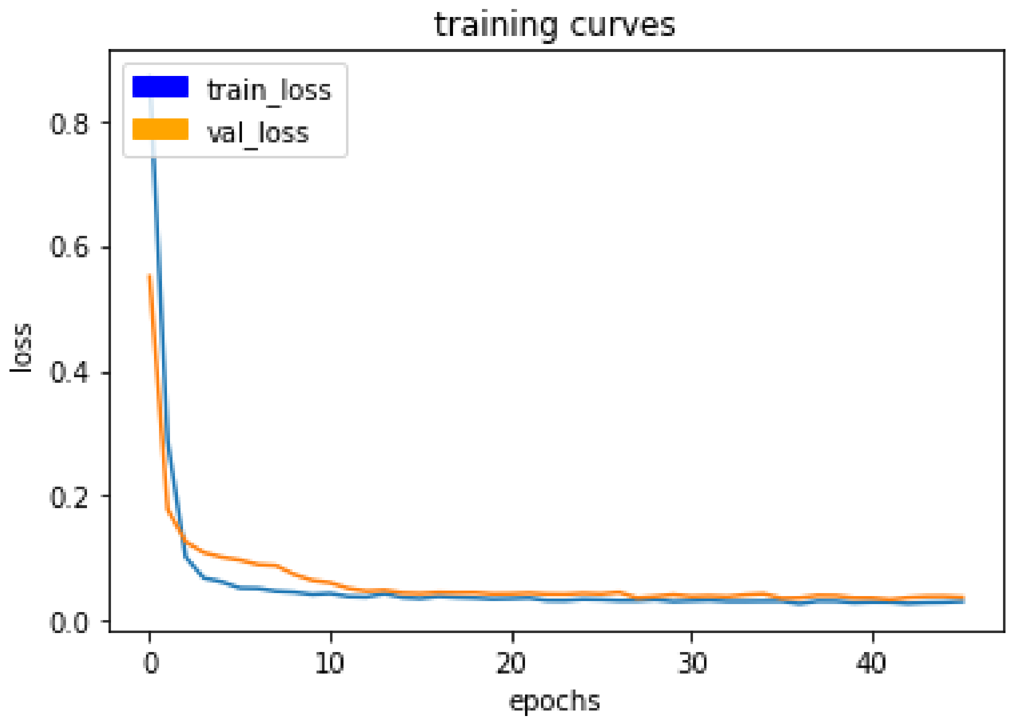

```
50/50 [==============================] - 36s - loss: 0.0278 - val_loss: 0.0371
Epoch 44/55
49/50 [============================>.] - ETA: 0s - loss: 0.0294
```



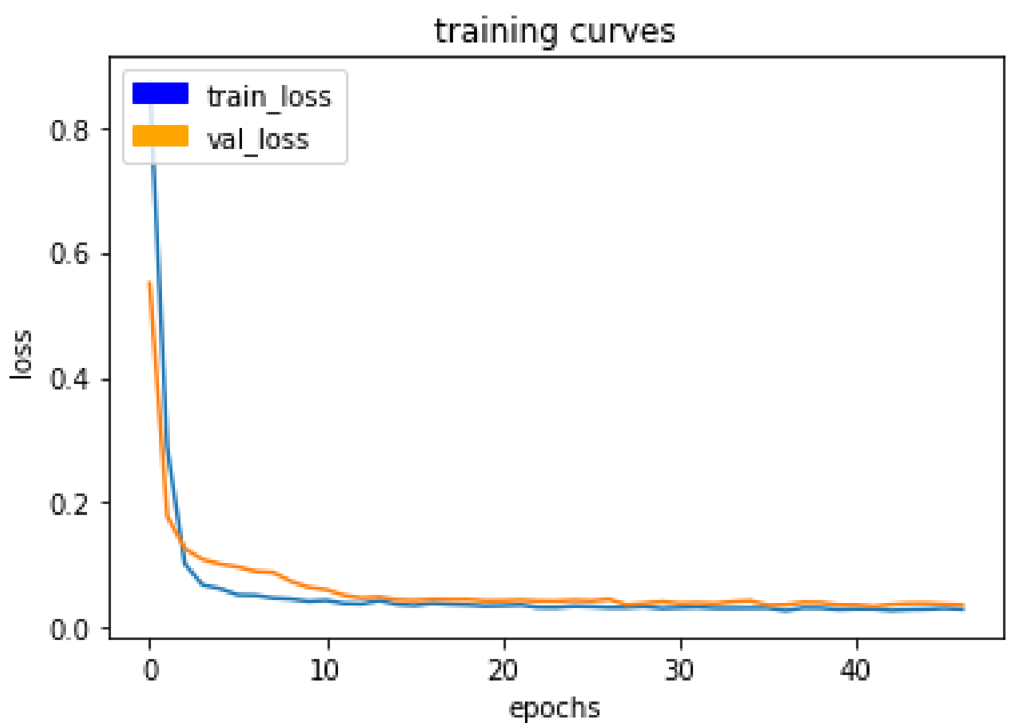
```
50/50 [==============================] - 36s - loss: 0.0293 - val_loss: 0.0389
Epoch 45/55
49/50 [============================>.] - ETA: 0s - loss: 0.0298
```



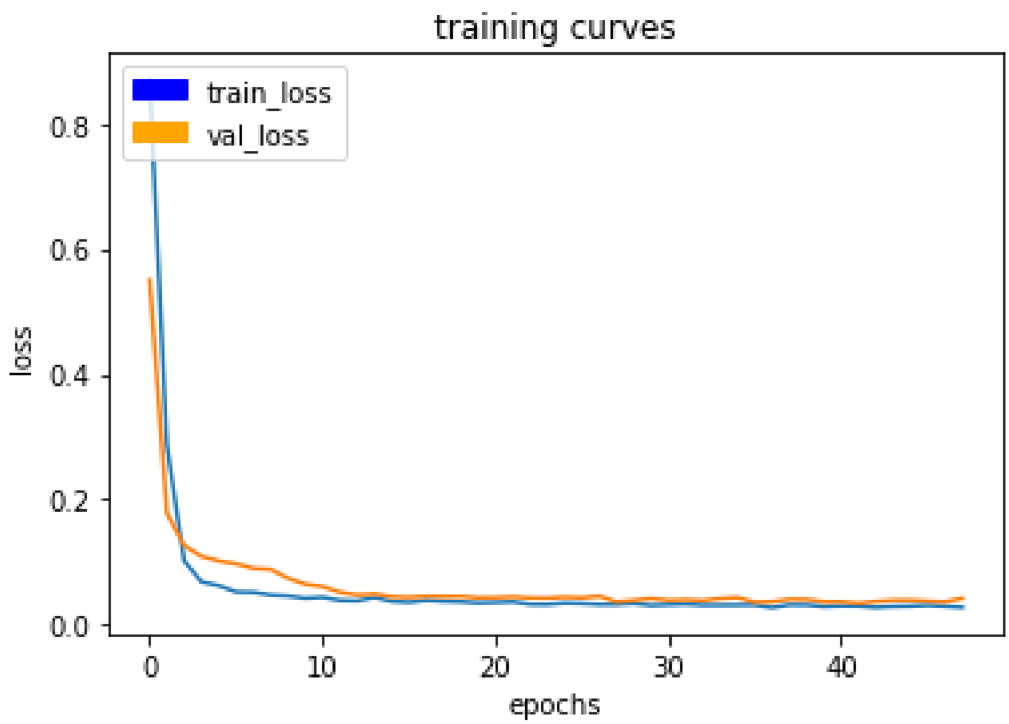

```
50/50 [==============================] - 36s - loss: 0.0297 - val_loss: 0.0391
Epoch 46/55
49/50 [============================>.] - ETA: 0s - loss: 0.0312
```



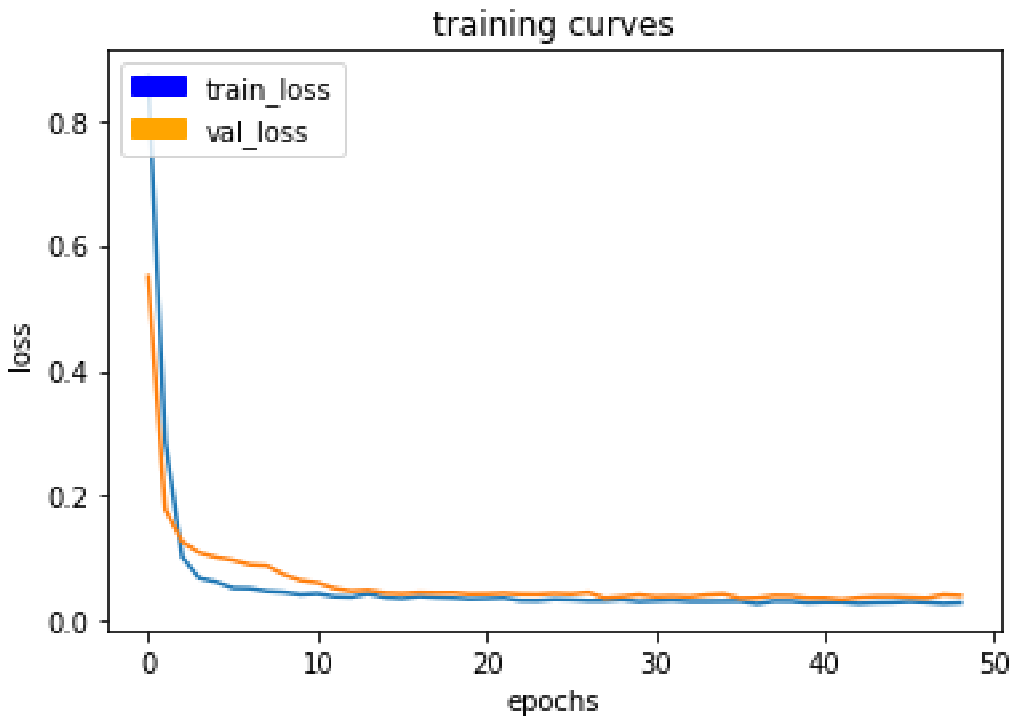

```
50/50 [==============================] - 35s - loss: 0.0312 - val_loss: 0.0377
Epoch 47/55
49/50 [============================>.] - ETA: 0s - loss: 0.0298
```



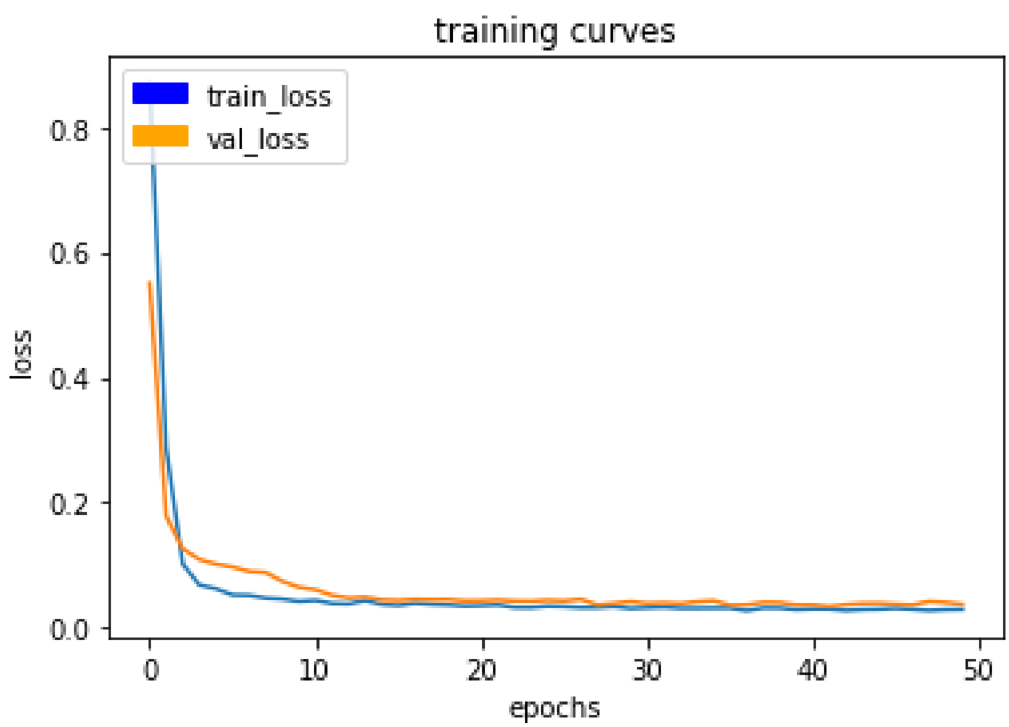
```
50/50 [==============================] - 36s - loss: 0.0296 - val_loss: 0.0365
Epoch 48/55
49/50 [============================>.] - ETA: 0s - loss: 0.0276
```



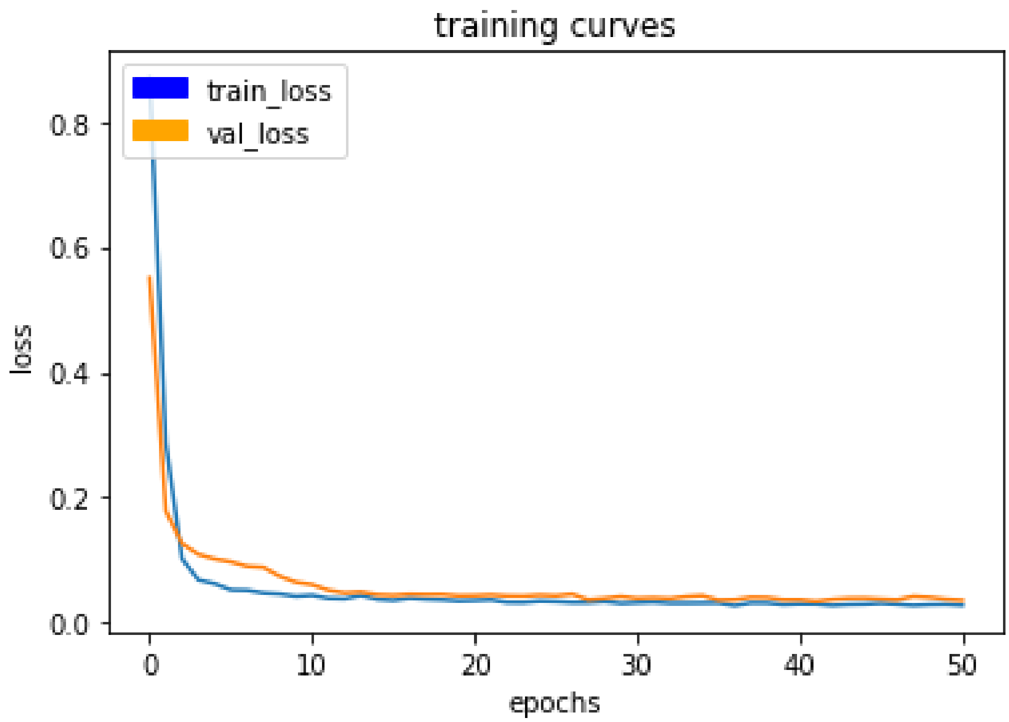

```
50/50 [==============================] - 36s - loss: 0.0277 - val_loss: 0.0421
Epoch 49/55
49/50 [============================>.] - ETA: 0s - loss: 0.0291
```



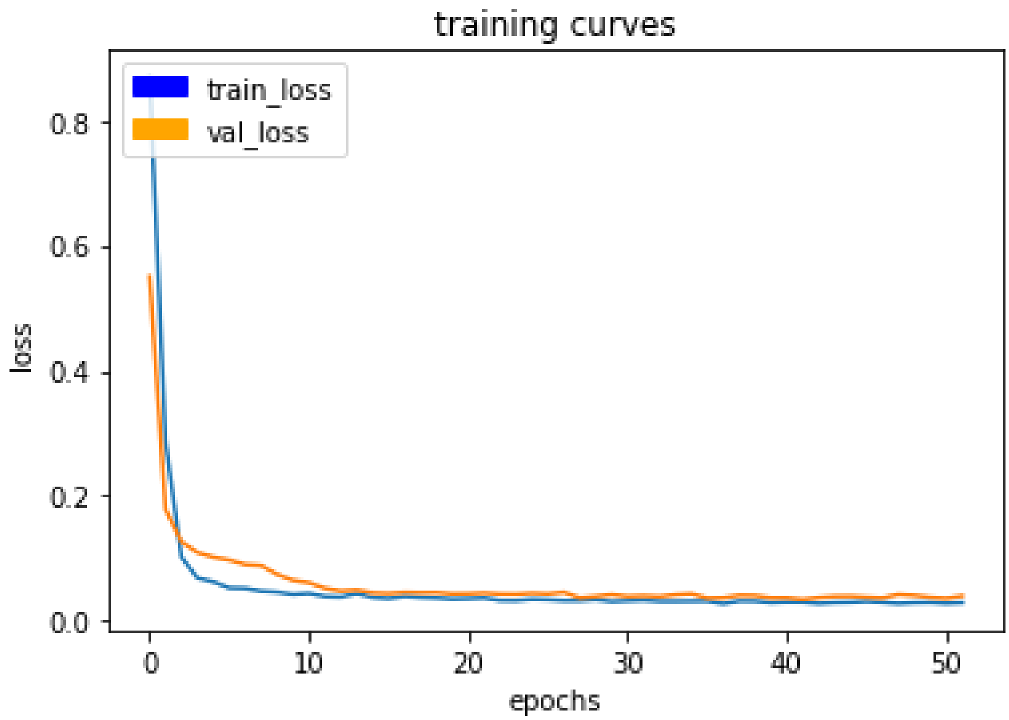

```
50/50 [==============================] - 36s - loss: 0.0290 - val_loss: 0.0405
Epoch 50/55
49/50 [============================>.] - ETA: 0s - loss: 0.0296
```



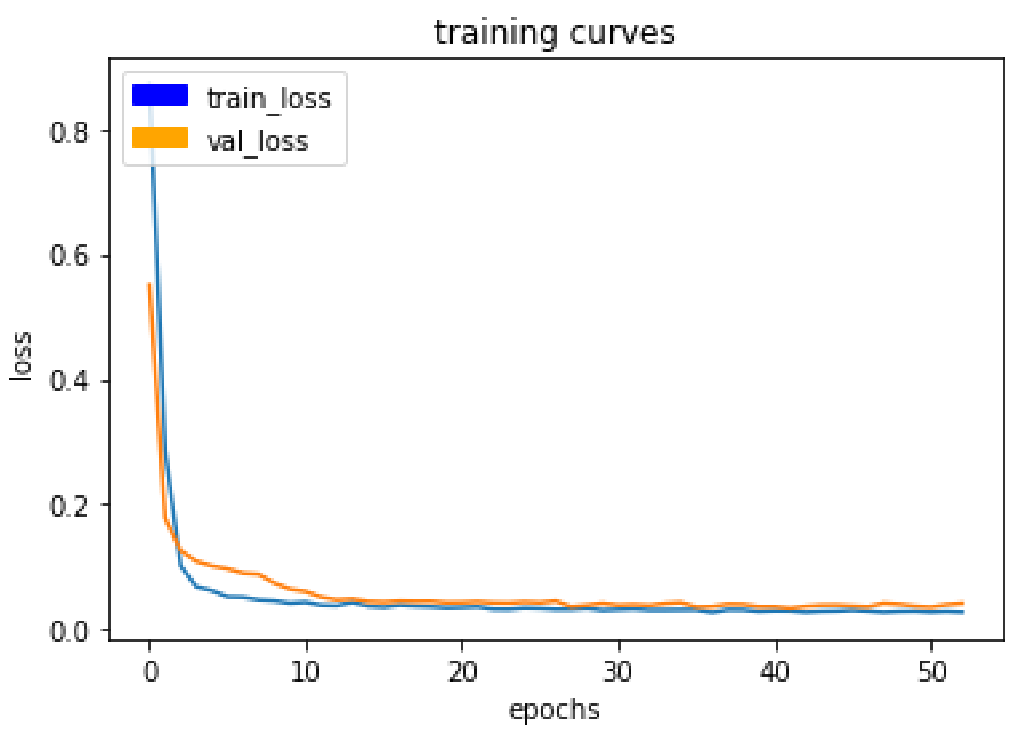
```
50/50 [==============================] - 35s - loss: 0.0295 - val_loss: 0.0375
Epoch 51/55
49/50 [============================>.] - ETA: 0s - loss: 0.0280
```



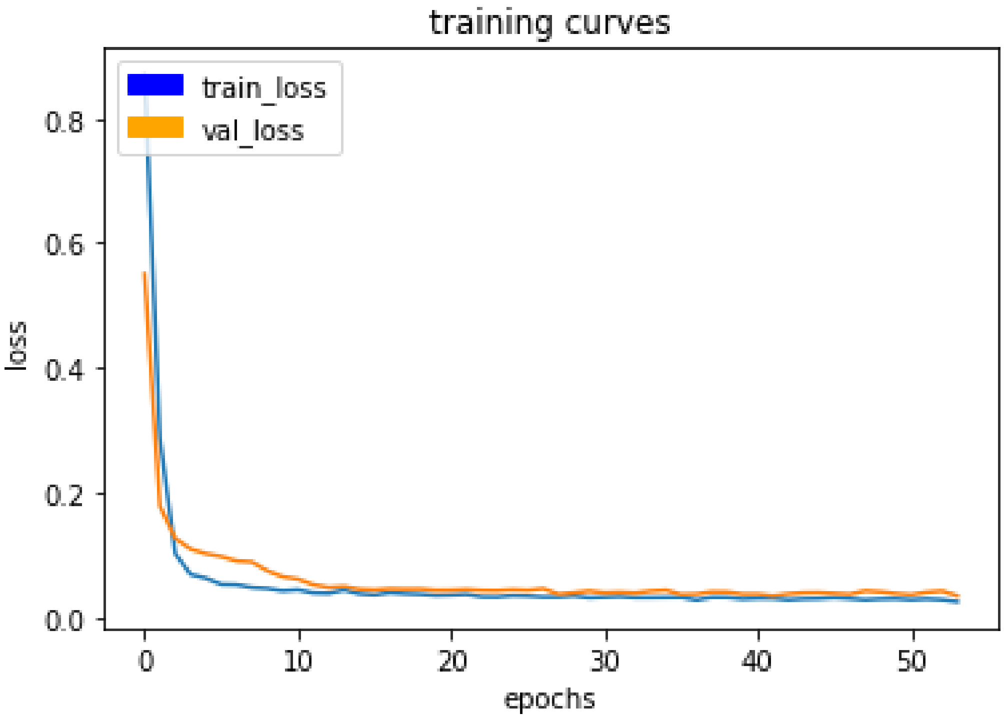

```
50/50 [==============================] - 36s - loss: 0.0280 - val_loss: 0.0358
Epoch 52/55
49/50 [============================>.] - ETA: 0s - loss: 0.0289
```



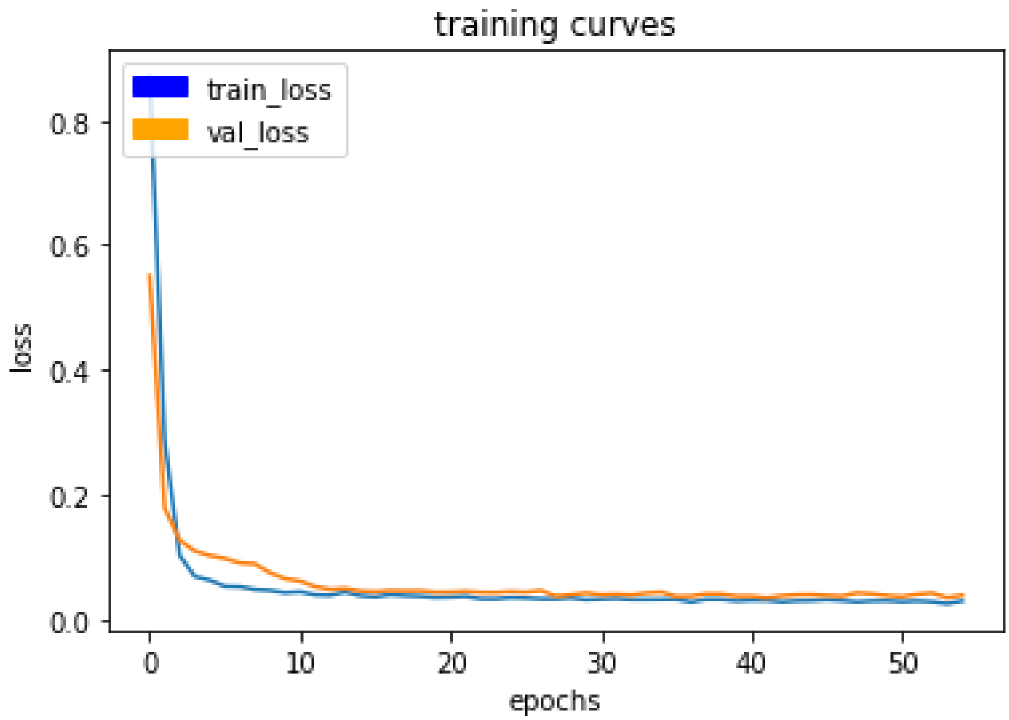

```
50/50 [==============================] - 36s - loss: 0.0290 - val_loss: 0.0397
Epoch 53/55
49/50 [============================>.] - ETA: 0s - loss: 0.0279
```



```
50/50 [==============================] - 36s - loss: 0.0279 - val_loss: 0.0419
Epoch 54/55
49/50 [============================>.] - ETA: 0s - loss: 0.0248
```




```
50/50 [==============================] - 35s - loss: 0.0247 - val_loss: 0.0343
Epoch 55/55
49/50 [============================>.] - ETA: 0s - loss: 0.0294
```




```
50/50 [==============================] - 36s - loss: 0.0294 - val_loss: 0.0381
```

```
Out[9]: <tensorflow.contrib.keras.python.keras.callbacks.History at 0x7fb9e93be6d8>
```

```
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.

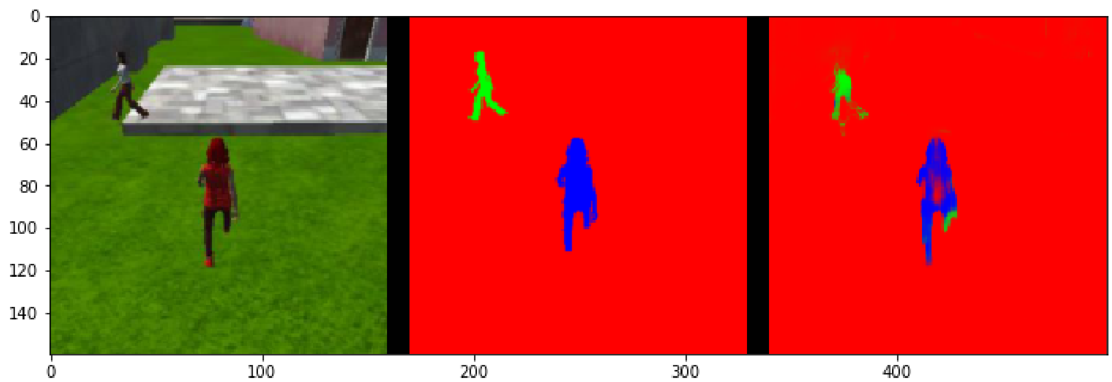
```
In [11]: # If you need to load a model which you previously trained you can uncomment the code

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

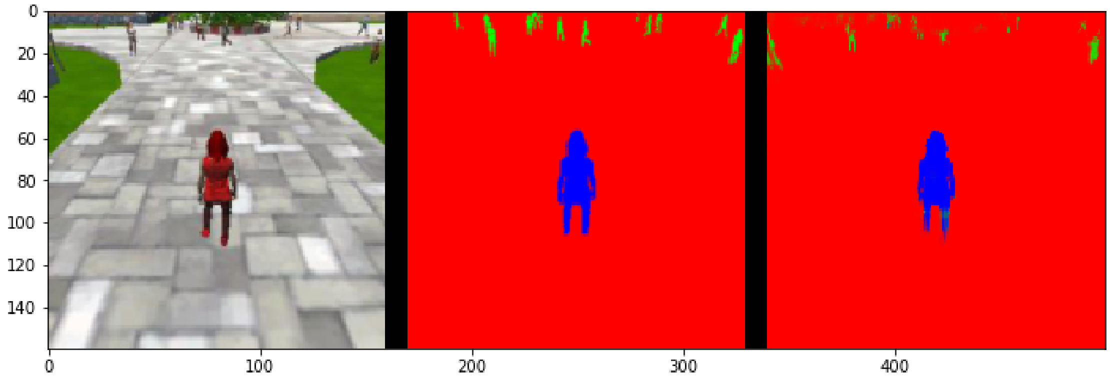
The following cell will write predictions to files and return paths to the appropriate directories. The run_num parameter is used to define or group all the data for a particular model run. You can change it for different runs. For example, 'run_1', 'run_2' etc.

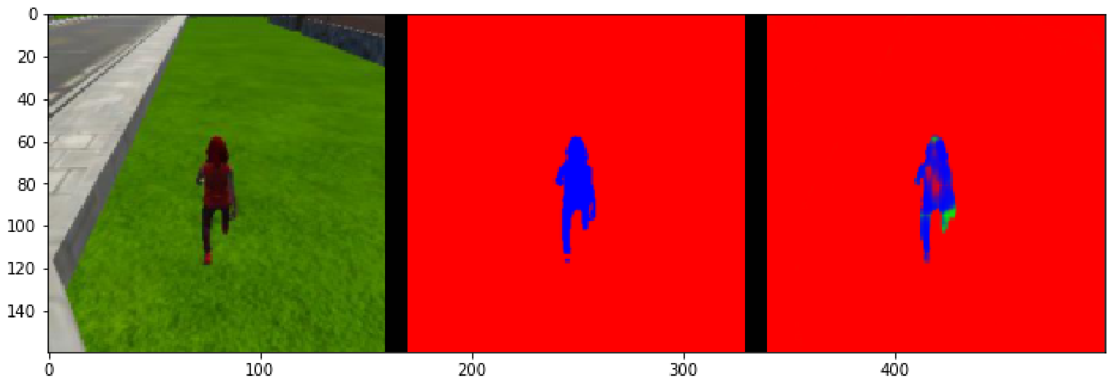
```
In [12]: run_num = 'run_1'

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

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

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

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.

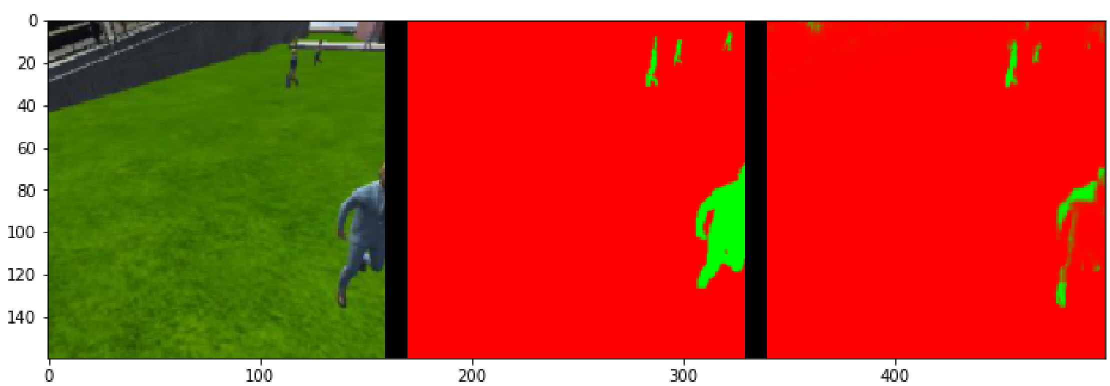
```
In [13]: # images while following the target
         im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','following_image
         for i in range(3):
             im_tuple = plotting_tools.load_images(im_files[i])
             plotting_tools.show_images(im_tuple)
```

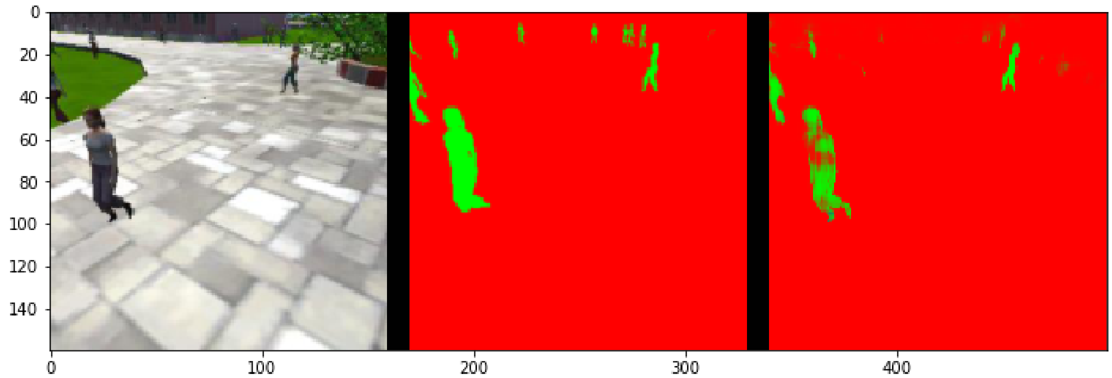


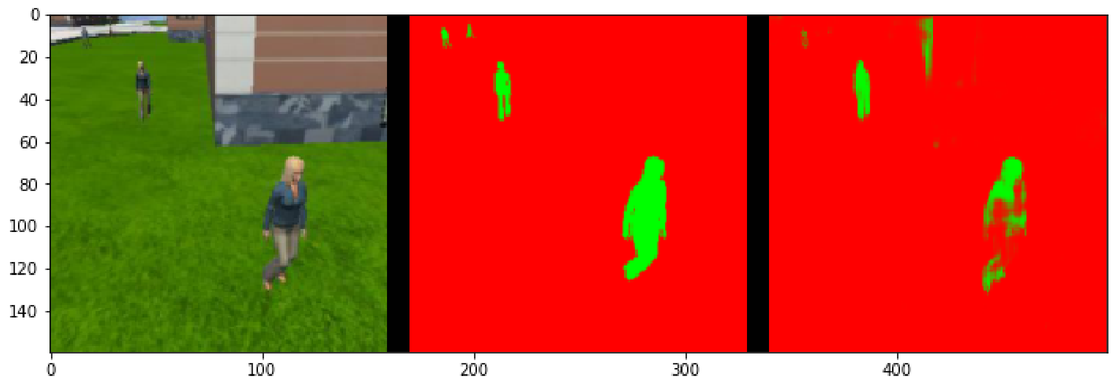





```
In [14]: # images while at patrol without target
         im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_non_targ
         for i in range(3):
             im_tuple = plotting_tools.load_images(im_files[i])
             plotting_tools.show_images(im_tuple)
```

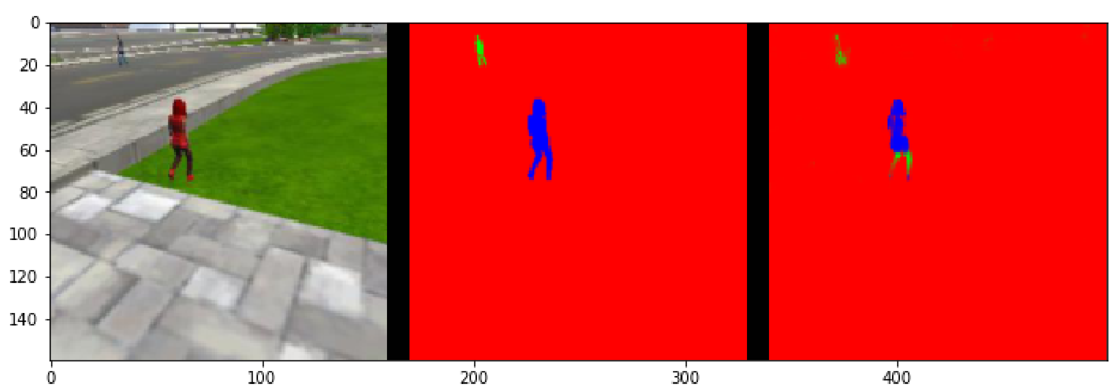


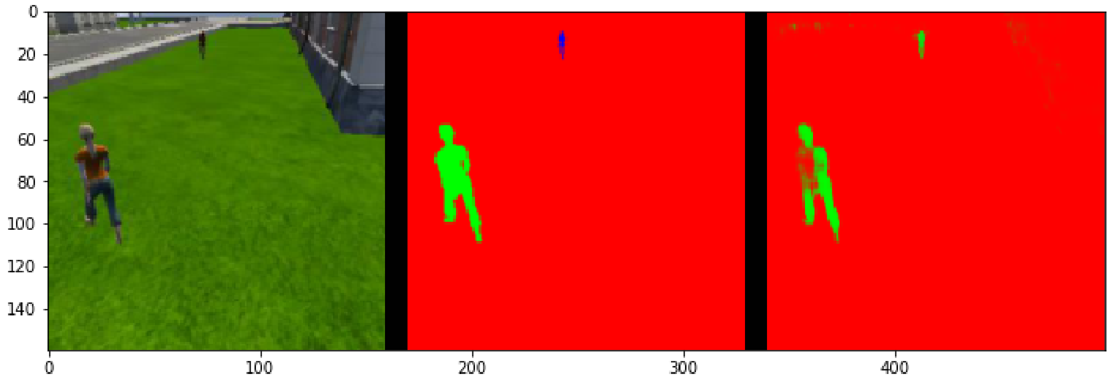


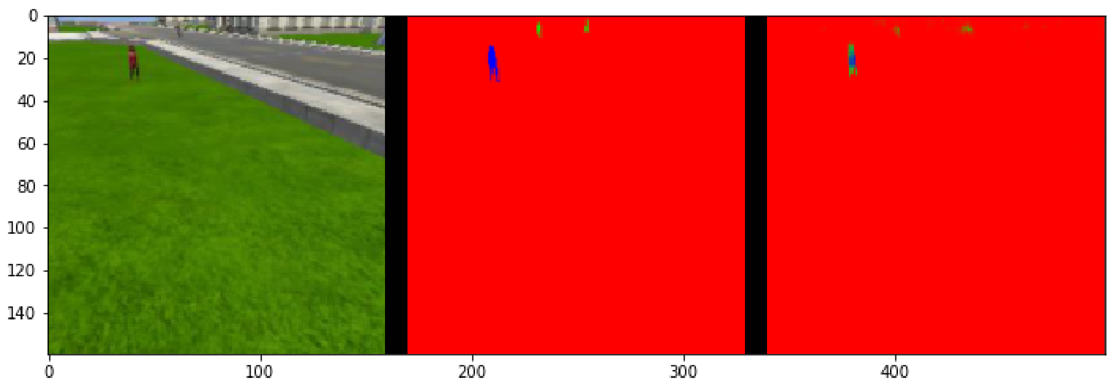


In [15]:

```
# images while at patrol with target
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_with_ta
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 [16]: # Scores for while the quad is following behind the target.
         true_pos1, false_pos1, false_neg1, iou1 = scoring_utils.score_run_iou(val_following, p
```

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9927287077655694
average intersection over union for other people is 0.25983523214722426
average intersection over union for the hero is 0.7539231332782744
number true positives: 539, number false positives: 0, number false negatives: 0
```

```
In [17]: # 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, pre
```

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.976278688127623
average intersection over union for other people is 0.5196730301828965
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 42, number false negatives: 0
```

```
In [18]: # 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, p
```

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9950078124538203
average intersection over union for other people is 0.3556042555330591
average intersection over union for the hero is 0.1391417052876219
number true positives: 118, number false positives: 2, number false negatives: 183
```

```
In [19]: # Sum all the true positives, etc from the three datasets to get a weight for the sco
         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)
```

```
0.7432126696832579
```

```
In [20]: # The IoU for the dataset that never includes the hero is excluded from grading
         final_IoU = (iou1 + iou3)/2
         print(final_IoU)
```

```
0.512332419283
```

```
In [21]: # And the final grade score is
         final_score = final_IoU * weight
         print(final_score)
```

```
0.409768551435
```