Open in Colab

T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 9 Material

- **Part 9.1: Introduction to Keras Transfer Learning** [Video] [Notebook]
- Part 9.2: Keras Transfer Learning for Computer Vision [Video] [Notebook]
- Part 9.3: Transfer Learning for NLP with Keras [Video] [Notebook]
- Part 9.4: Transfer Learning for Facial Feature Recognition [Video] [Notebook]
- Part 9.5: Transfer Learning for Style Transfer [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [ ]:
```
# Start CoLab
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: not using Google CoLab
```

Part 9.1: Introduction to Keras Transfer Learning

Human beings learn new skills throughout their entire lives. However, this learning is rarely from scratch. No matter what task a human learns, they are most likely drawing on experiences to learn this new skill early in life. In this way, humans learn much differently than most deep learning projects.

A human being learns to tell the difference between a cat and a dog at some point. To teach a neural network, you would obtain many cat pictures and dog pictures. The neural network would iterate over all of these pictures and train on the differences. The human child that learned to distinguish between the two animals would probably need to see a few examples when parents told them the name of each type of animal. The human child would use previous knowledge of looking at different living and non-living objects to help make this classification. The child would already know the physical appearance of sub-objects, such as fur, eyes, ears, noses, tails, and teeth.

Transfer learning attempts to teach a neural network by similar means. Rather than training your neural network from scratch, you begin training with a preloaded set of weights. Usually, you will remove the topmost layers of the pretrained neural network and retrain it with new uppermost layers. The layers from the previous neural network will be locked so that training does not change these weights. Only the newly added layers will be trained.

It can take much computing power to train a neural network for a large image dataset. Google, Facebook, Microsoft, and other tech companies have utilized GPU arrays for training high-quality neural networks for various applications. Transferring these weights into your neural network can save considerable effort and compute time. It is unlikely that a pretrained model will exactly fit the application that you seek to implement. Finding the closest pretrained model and using transfer learning is essential for a deep learning engineer.

Transfer Learning Example

Let's look at a simple example of using transfer learning to build upon an imagenet neural network. We will begin by training a neural network for Fisher's Iris Dataset. This network takes four measurements and classifies each observation into three iris species. However, what if later we received a data set that included the four measurements, plus a cost as the target? This dataset does not contain the species; as a result, it uses the same four inputs as the base model we just trained.

We can take our previously trained iris network and transfer the weights to a new neural network that will learn to predict the cost through transfer learning. Also of note, the original neural network was a classification network, yet we now use it to build a regression neural network. Such a transformation is common for transfer learning. As a reference point, I randomly created this iris cost dataset.

The first step is to train our neural network for the regular Iris Dataset. The code presented here is the same as we saw in Module 3.

In [ ]:
```
import pandas as pd
import io
import requests
import numpy as np
from sklearn import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/iris.csv",
    na_values=['NA', '?'])

# Convert to numpy - Classification
x = df[['sepal_l', 'sepal_w', 'petal_l', 'petal_w']].values
dummies = pd.get_dummies(df['species']) # Classification
species = dummies.columns
y = dummies.values


# Build neural network
model = Sequential()
model.add(Dense(50, input_dim=x.shape[1], activation='relu')) # Hidden 1
model.add(Dense(25, activation='relu')) # Hidden 2
model.add(Dense(y.shape[1],activation='softmax')) # Output

model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit(x,y,verbose=2,epochs=100)
```

```
Epoch 1/100
5/5 - 1s - loss: 1.7638 - 634ms/epoch - 127ms/step
Epoch 2/100
5/5 - 0s - loss: 1.2951 - 9ms/epoch - 2ms/step
Epoch 3/100
5/5 - 0s - loss: 1.0713 - 8ms/epoch - 2ms/step
Epoch 4/100
5/5 - 0s - loss: 1.0110 - 12ms/epoch - 2ms/step
Epoch 5/100
5/5 - 0s - loss: 0.9364 - 9ms/epoch - 2ms/step
Epoch 6/100
5/5 - 0s - loss: 0.8444 - 8ms/epoch - 2ms/step
Epoch 7/100
5/5 - 0s - loss: 0.7800 - 12ms/epoch - 2ms/step
Epoch 8/100
5/5 - 0s - loss: 0.7321 - 13ms/epoch - 3ms/step
Epoch 9/100
5/5 - 0s - loss: 0.6806 - 13ms/epoch - 3ms/step
Epoch 10/100
5/5 - 0s - loss: 0.6377 - 12ms/epoch - 2ms/step
Epoch 11/100
5/5 - 0s - loss: 0.6021 - 13ms/epoch - 3ms/step
Epoch 12/100
5/5 - 0s - loss: 0.5693 - 10ms/epoch - 2ms/step
Epoch 13/100
5/5 - 0s - loss: 0.5470 - 11ms/epoch - 2ms/step
Epoch 14/100
5/5 - 0s - loss: 0.5219 - 11ms/epoch - 2ms/step
Epoch 15/100
5/5 - 0s - loss: 0.4992 - 24ms/epoch - 5ms/step
Epoch 16/100
5/5 - 0s - loss: 0.4757 - 15ms/epoch - 3ms/step
Epoch 17/100
5/5 - 0s - loss: 0.4576 - 11ms/epoch - 2ms/step
Epoch 18/100
5/5 - 0s - loss: 0.4422 - 11ms/epoch - 2ms/step
Epoch 19/100
5/5 - 0s - loss: 0.4267 - 13ms/epoch - 3ms/step
Epoch 20/100
5/5 - 0s - loss: 0.4119 - 17ms/epoch - 3ms/step
Epoch 21/100
5/5 - 0s - loss: 0.3994 - 12ms/epoch - 2ms/step
Epoch 22/100
5/5 - 0s - loss: 0.3877 - 16ms/epoch - 3ms/step
Epoch 23/100
5/5 - 0s - loss: 0.3783 - 16ms/epoch - 3ms/step
Epoch 24/100
5/5 - 0s - loss: 0.3640 - 16ms/epoch - 3ms/step
Epoch 25/100
5/5 - 0s - loss: 0.3537 - 16ms/epoch - 3ms/step
Epoch 26/100
5/5 - 0s - loss: 0.3443 - 9ms/epoch - 2ms/step
Epoch 27/100
5/5 - 0s - loss: 0.3375 - 11ms/epoch - 2ms/step
Epoch 28/100
5/5 - 0s - loss: 0.3245 - 14ms/epoch - 3ms/step
```

```
Epoch 29/100
5/5 - 0s - loss: 0.3152 - 16ms/epoch - 3ms/step
Epoch 30/100
5/5 - 0s - loss: 0.3050 - 15ms/epoch - 3ms/step
Epoch 31/100
5/5 - 0s - loss: 0.2934 - 16ms/epoch - 3ms/step
Epoch 32/100
5/5 - 0s - loss: 0.2830 - 14ms/epoch - 3ms/step
Epoch 33/100
5/5 - 0s - loss: 0.2738 - 13ms/epoch - 3ms/step
Epoch 34/100
5/5 - 0s - loss: 0.2651 - 17ms/epoch - 3ms/step
Epoch 35/100
5/5 - 0s - loss: 0.2609 - 19ms/epoch - 4ms/step
Epoch 36/100
5/5 - 0s - loss: 0.2499 - 13ms/epoch - 3ms/step
Epoch 37/100
5/5 - 0s - loss: 0.2436 - 12ms/epoch - 2ms/step
Epoch 38/100
5/5 - 0s - loss: 0.2353 - 13ms/epoch - 3ms/step
Epoch 39/100
5/5 - 0s - loss: 0.2291 - 12ms/epoch - 2ms/step
Epoch 40/100
5/5 - 0s - loss: 0.2226 - 14ms/epoch - 3ms/step
Epoch 41/100
5/5 - 0s - loss: 0.2171 - 16ms/epoch - 3ms/step
Epoch 42/100
5/5 - 0s - loss: 0.2140 - 16ms/epoch - 3ms/step
Epoch 43/100
5/5 - 0s - loss: 0.2037 - 17ms/epoch - 3ms/step
Epoch 44/100
5/5 - 0s - loss: 0.2039 - 17ms/epoch - 3ms/step
Epoch 45/100
5/5 - 0s - loss: 0.1982 - 14ms/epoch - 3ms/step
Epoch 46/100
5/5 - 0s - loss: 0.1897 - 9ms/epoch - 2ms/step
Epoch 47/100
5/5 - 0s - loss: 0.1862 - 18ms/epoch - 4ms/step
Epoch 48/100
5/5 - 0s - loss: 0.1812 - 13ms/epoch - 3ms/step
Epoch 49/100
5/5 - 0s - loss: 0.1798 - 15ms/epoch - 3ms/step
Epoch 50/100
5/5 - 0s - loss: 0.1726 - 19ms/epoch - 4ms/step
Epoch 51/100
5/5 - 0s - loss: 0.1685 - 14ms/epoch - 3ms/step
Epoch 52/100
5/5 - 0s - loss: 0.1674 - 15ms/epoch - 3ms/step
Epoch 53/100
5/5 - 0s - loss: 0.1651 - 16ms/epoch - 3ms/step
Epoch 54/100
5/5 - 0s - loss: 0.1576 - 10ms/epoch - 2ms/step
Epoch 55/100
5/5 - 0s - loss: 0.1550 - 14ms/epoch - 3ms/step
Epoch 56/100
5/5 - 0s - loss: 0.1515 - 16ms/epoch - 3ms/step
```

```
Epoch 57/100
5/5 - 0s - loss: 0.1526 - 16ms/epoch - 3ms/step
Epoch 58/100
5/5 - 0s - loss: 0.1487 - 14ms/epoch - 3ms/step
Epoch 59/100
5/5 - 0s - loss: 0.1458 - 17ms/epoch - 3ms/step
Epoch 60/100
5/5 - 0s - loss: 0.1411 - 14ms/epoch - 3ms/step
Epoch 61/100
5/5 - 0s - loss: 0.1401 - 9ms/epoch - 2ms/step
Epoch 62/100
5/5 - 0s - loss: 0.1355 - 14ms/epoch - 3ms/step
Epoch 63/100
5/5 - 0s - loss: 0.1344 - 18ms/epoch - 4ms/step
Epoch 64/100
5/5 - 0s - loss: 0.1321 - 15ms/epoch - 3ms/step
Epoch 65/100
5/5 - 0s - loss: 0.1295 - 12ms/epoch - 2ms/step
Epoch 66/100
5/5 - 0s - loss: 0.1278 - 12ms/epoch - 2ms/step
Epoch 67/100
5/5 - 0s - loss: 0.1261 - 10ms/epoch - 2ms/step
Epoch 68/100
5/5 - 0s - loss: 0.1255 - 15ms/epoch - 3ms/step
Epoch 69/100
5/5 - 0s - loss: 0.1237 - 9ms/epoch - 2ms/step
Epoch 70/100
5/5 - 0s - loss: 0.1210 - 16ms/epoch - 3ms/step
Epoch 71/100
5/5 - 0s - loss: 0.1184 - 15ms/epoch - 3ms/step
Epoch 72/100
5/5 - 0s - loss: 0.1168 - 15ms/epoch - 3ms/step
Epoch 73/100
5/5 - 0s - loss: 0.1146 - 12ms/epoch - 2ms/step
Epoch 74/100
5/5 - 0s - loss: 0.1169 - 15ms/epoch - 3ms/step
Epoch 75/100
5/5 - 0s - loss: 0.1123 - 10ms/epoch - 2ms/step
Epoch 76/100
5/5 - 0s - loss: 0.1109 - 15ms/epoch - 3ms/step
Epoch 77/100
5/5 - 0s - loss: 0.1089 - 15ms/epoch - 3ms/step
Epoch 78/100
5/5 - 0s - loss: 0.1079 - 15ms/epoch - 3ms/step
Epoch 79/100
5/5 - 0s - loss: 0.1074 - 12ms/epoch - 2ms/step
Epoch 80/100
5/5 - 0s - loss: 0.1060 - 10ms/epoch - 2ms/step
Epoch 81/100
5/5 - 0s - loss: 0.1054 - 10ms/epoch - 2ms/step
Epoch 82/100
5/5 - 0s - loss: 0.1030 - 18ms/epoch - 4ms/step
Epoch 83/100
5/5 - 0s - loss: 0.1037 - 15ms/epoch - 3ms/step
Epoch 84/100
5/5 - 0s - loss: 0.1014 - 15ms/epoch - 3ms/step
```

```
Epoch 85/100
5/5 - 0s - loss: 0.0989 - 17ms/epoch - 3ms/step
Epoch 86/100
5/5 - 0s - loss: 0.0989 - 9ms/epoch - 2ms/step
Epoch 87/100
5/5 - 0s - loss: 0.1000 - 13ms/epoch - 3ms/step
Epoch 88/100
5/5 - 0s - loss: 0.0953 - 8ms/epoch - 2ms/step
Epoch 89/100
5/5 - 0s - loss: 0.0994 - 11ms/epoch - 2ms/step
Epoch 90/100
5/5 - 0s - loss: 0.0941 - 12ms/epoch - 2ms/step
Epoch 91/100
5/5 - 0s - loss: 0.0947 - 15ms/epoch - 3ms/step
Epoch 92/100
5/5 - 0s - loss: 0.0963 - 18ms/epoch - 4ms/step
Epoch 93/100
5/5 - 0s - loss: 0.0913 - 14ms/epoch - 3ms/step
Epoch 94/100
5/5 - 0s - loss: 0.0922 - 20ms/epoch - 4ms/step
Epoch 95/100
5/5 - 0s - loss: 0.0916 - 13ms/epoch - 3ms/step
Epoch 96/100
5/5 - 0s - loss: 0.0905 - 12ms/epoch - 2ms/step
Epoch 97/100
5/5 - 0s - loss: 0.0886 - 9ms/epoch - 2ms/step
Epoch 98/100
5/5 - 0s - loss: 0.0900 - 15ms/epoch - 3ms/step
Epoch 99/100
5/5 - 0s - loss: 0.0868 - 15ms/epoch - 3ms/step
Epoch 100/100
5/5 - 0s - loss: 0.0892 - 8ms/epoch - 2ms/step
```

Out[ ]:

```
<keras.callbacks.History at 0x7fea3fb1ef50>
```

To keep this example simple, we are not setting aside a validation set. The goal of this example is to show how to create a multi-layer neural network, where we transfer the weights to another network. We begin by evaluating the accuracy of the network on the training set.

In [ ]:

```
from sklearn.metrics import accuracy_score
pred = model.predict(x)
predict_classes = np.argmax(pred,axis=1)
expected_classes = np.argmax(y,axis=1)
correct = accuracy_score(expected_classes,predict_classes)
print(f"Training Accuracy: {correct}")
```

```
Training Accuracy: 0.9866666666666667
```

Viewing the model summary is as expected; we can see the three layers previously defined.

In [ ]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                250

 dense_1 (Dense)             (None, 25)                1275

 dense_2 (Dense)             (None, 3)                 78

=================================================================
Total params: 1,603
Trainable params: 1,603
Non-trainable params: 0
_________________________________________________________________
```

Create a New Iris Network

Now that we've trained a neural network on the iris dataset, we can transfer the knowledge of this neural network to other neural networks. It is possible to create a new neural network from some or all of the layers of this neural network. We will create a new neural network that is essentially a clone of the first neural network to demonstrate the technique. We now transfer all of the layers from the original neural network into the new one.

In [ ]:

```python
model2 = Sequential()
for layer in model.layers:
    model2.add(layer)
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                250

 dense_1 (Dense)             (None, 25)                1275

 dense_2 (Dense)             (None, 3)                 78

=================================================================
Total params: 1,603
Trainable params: 1,603
Non-trainable params: 0
_________________________________________________________________
```

As a sanity check, we would like to calculate the accuracy of the newly created model. The in-sample accuracy should be the same as the previous model that the new model transferred.

In [ ]:

```python
from sklearn.metrics import accuracy_score
pred = model2.predict(x)
```

```
predict_classes = np.argmax(pred,axis=1)
expected_classes = np.argmax(y,axis=1)
correct = accuracy_score(expected_classes,predict_classes)
print(f"Training Accuracy: {correct}")
```

```
Training Accuracy: 0.9866666666666667
```

The in-sample accuracy of the newly created neural network is the same as the first neural network. We've successfully transferred all of the layers from the original neural network.

Transfering to a Regression Network

The Iris Cost Dataset has measurements for samples of these flowers that conform to the predictors contained in the original iris dataset: sepal width, sepal length, petal width, and petal length. We present the cost dataset here.

In [ ]:

```
df_cost = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/iris_cost.csv",
    na_values=['NA', '?'])

df_cost
```

Out[ ]:

	sepal_l	sepal_w	petal_l	petal_w	cost
0	7.8	3.0	6.2	2.0	10.740
1	5.0	2.2	1.7	1.5	2.710
2	6.9	2.6	3.7	1.4	4.624
3	5.9	2.2	3.7	2.4	6.558
4	5.1	3.9	6.8	0.7	7.395
...	...	...	...	...	...
245	4.7	2.1	4.0	2.3	5.721
246	7.2	3.0	4.3	1.1	5.266
247	6.6	3.4	4.6	1.4	5.776
248	5.7	3.7	3.1	0.4	2.233
249	7.6	4.0	5.1	1.4	7.508

250 rows × 5 columns

For transfer learning to be effective, the input for the newly trained neural network most closely conforms to the first neural network we transfer.

We will strip away the last output layer that contains the softmax activation function that performs this final classification. We will create a new output layer that will output the cost prediction. We will only train the weights in this new layer. We will mark the first two layers as non-trainable. The hope is that the first few layers have learned to abstract the raw input data in a way that is also helpful to the new neural network. This process is accomplished by looping over the first few layers and copying them to the new neural network. We output a summary of the new neural network to verify that Keras stripped the previous output layer.

In [ ]:
```
model3 = Sequential()
for i in range(2):
    layer = model.layers[i]
    layer.trainable = False
    model3.add(layer)
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                250

 dense_1 (Dense)             (None, 25)                1275

=================================================================
Total params: 1,525
Trainable params: 0
Non-trainable params: 1,525
_________________________________________________________________
```

We add a final regression output layer to complete the new neural network.

In [ ]:
```
model3.add(Dense(1)) # Output

model3.compile(loss='mean_squared_error', optimizer='adam')
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                250

 dense_1 (Dense)             (None, 25)                1275

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 1,551
Trainable params: 26
Non-trainable params: 1,525
_________________________________________________________________
```

Now we train just the output layer to predict the cost. The cost in the made-up dataset is dependent on the species, so the previous learning should be helpful.

In [ ]:
```
# Convert to numpy - Classification
x = df_cost[['sepal_l', 'sepal_w', 'petal_l', 'petal_w']].values
y = df_cost.cost.values

# Train the last layer of the network
model3.fit(x,y,verbose=2,epochs=100)
```

```
Epoch 1/100
8/8 - 0s - loss: 14.0400 - 379ms/epoch - 47ms/step
Epoch 2/100
8/8 - 0s - loss: 12.6133 - 10ms/epoch - 1ms/step
Epoch 3/100
8/8 - 0s - loss: 11.3224 - 12ms/epoch - 1ms/step
Epoch 4/100
8/8 - 0s - loss: 10.1006 - 19ms/epoch - 2ms/step
Epoch 5/100
8/8 - 0s - loss: 9.0898 - 19ms/epoch - 2ms/step
Epoch 6/100
8/8 - 0s - loss: 8.1514 - 13ms/epoch - 2ms/step
Epoch 7/100
8/8 - 0s - loss: 7.3497 - 11ms/epoch - 1ms/step
Epoch 8/100
8/8 - 0s - loss: 6.6789 - 14ms/epoch - 2ms/step
Epoch 9/100
8/8 - 0s - loss: 6.0785 - 11ms/epoch - 1ms/step
Epoch 10/100
8/8 - 0s - loss: 5.5620 - 11ms/epoch - 1ms/step
Epoch 11/100
8/8 - 0s - loss: 5.1035 - 11ms/epoch - 1ms/step
Epoch 12/100
8/8 - 0s - loss: 4.7415 - 12ms/epoch - 2ms/step
Epoch 13/100
8/8 - 0s - loss: 4.4169 - 13ms/epoch - 2ms/step
Epoch 14/100
8/8 - 0s - loss: 4.1181 - 19ms/epoch - 2ms/step
Epoch 15/100
8/8 - 0s - loss: 3.8847 - 20ms/epoch - 3ms/step
Epoch 16/100
8/8 - 0s - loss: 3.6586 - 13ms/epoch - 2ms/step
Epoch 17/100
8/8 - 0s - loss: 3.4690 - 17ms/epoch - 2ms/step
Epoch 18/100
8/8 - 0s - loss: 3.3085 - 16ms/epoch - 2ms/step
Epoch 19/100
8/8 - 0s - loss: 3.1461 - 17ms/epoch - 2ms/step
Epoch 20/100
8/8 - 0s - loss: 3.0206 - 20ms/epoch - 2ms/step
Epoch 21/100
8/8 - 0s - loss: 2.8936 - 14ms/epoch - 2ms/step
Epoch 22/100
8/8 - 0s - loss: 2.7855 - 11ms/epoch - 1ms/step
Epoch 23/100
8/8 - 0s - loss: 2.6949 - 10ms/epoch - 1ms/step
Epoch 24/100
8/8 - 0s - loss: 2.6056 - 13ms/epoch - 2ms/step
Epoch 25/100
8/8 - 0s - loss: 2.5355 - 24ms/epoch - 3ms/step
Epoch 26/100
8/8 - 0s - loss: 2.4600 - 10ms/epoch - 1ms/step
Epoch 27/100
8/8 - 0s - loss: 2.4041 - 13ms/epoch - 2ms/step
Epoch 28/100
8/8 - 0s - loss: 2.3449 - 14ms/epoch - 2ms/step
```

```
Epoch 29/100
8/8 - 0s - loss: 2.2991 - 17ms/epoch - 2ms/step
Epoch 30/100
8/8 - 0s - loss: 2.2528 - 15ms/epoch - 2ms/step
Epoch 31/100
8/8 - 0s - loss: 2.2143 - 15ms/epoch - 2ms/step
Epoch 32/100
8/8 - 0s - loss: 2.1818 - 17ms/epoch - 2ms/step
Epoch 33/100
8/8 - 0s - loss: 2.1527 - 15ms/epoch - 2ms/step
Epoch 34/100
8/8 - 0s - loss: 2.1262 - 17ms/epoch - 2ms/step
Epoch 35/100
8/8 - 0s - loss: 2.1046 - 13ms/epoch - 2ms/step
Epoch 36/100
8/8 - 0s - loss: 2.0811 - 13ms/epoch - 2ms/step
Epoch 37/100
8/8 - 0s - loss: 2.0657 - 10ms/epoch - 1ms/step
Epoch 38/100
8/8 - 0s - loss: 2.0487 - 15ms/epoch - 2ms/step
Epoch 39/100
8/8 - 0s - loss: 2.0368 - 17ms/epoch - 2ms/step
Epoch 40/100
8/8 - 0s - loss: 2.0257 - 21ms/epoch - 3ms/step
Epoch 41/100
8/8 - 0s - loss: 2.0131 - 15ms/epoch - 2ms/step
Epoch 42/100
8/8 - 0s - loss: 2.0053 - 24ms/epoch - 3ms/step
Epoch 43/100
8/8 - 0s - loss: 1.9972 - 22ms/epoch - 3ms/step
Epoch 44/100
8/8 - 0s - loss: 1.9898 - 17ms/epoch - 2ms/step
Epoch 45/100
8/8 - 0s - loss: 1.9838 - 14ms/epoch - 2ms/step
Epoch 46/100
8/8 - 0s - loss: 1.9788 - 12ms/epoch - 1ms/step
Epoch 47/100
8/8 - 0s - loss: 1.9743 - 14ms/epoch - 2ms/step
Epoch 48/100
8/8 - 0s - loss: 1.9702 - 19ms/epoch - 2ms/step
Epoch 49/100
8/8 - 0s - loss: 1.9662 - 15ms/epoch - 2ms/step
Epoch 50/100
8/8 - 0s - loss: 1.9646 - 14ms/epoch - 2ms/step
Epoch 51/100
8/8 - 0s - loss: 1.9604 - 15ms/epoch - 2ms/step
Epoch 52/100
8/8 - 0s - loss: 1.9578 - 16ms/epoch - 2ms/step
Epoch 53/100
8/8 - 0s - loss: 1.9552 - 14ms/epoch - 2ms/step
Epoch 54/100
8/8 - 0s - loss: 1.9528 - 11ms/epoch - 1ms/step
Epoch 55/100
8/8 - 0s - loss: 1.9511 - 19ms/epoch - 2ms/step
Epoch 56/100
8/8 - 0s - loss: 1.9489 - 18ms/epoch - 2ms/step
```

```
Epoch 57/100
8/8 - 0s - loss: 1.9468 - 17ms/epoch - 2ms/step
Epoch 58/100
8/8 - 0s - loss: 1.9462 - 19ms/epoch - 2ms/step
Epoch 59/100
8/8 - 0s - loss: 1.9435 - 14ms/epoch - 2ms/step
Epoch 60/100
8/8 - 0s - loss: 1.9421 - 11ms/epoch - 1ms/step
Epoch 61/100
8/8 - 0s - loss: 1.9406 - 17ms/epoch - 2ms/step
Epoch 62/100
8/8 - 0s - loss: 1.9389 - 14ms/epoch - 2ms/step
Epoch 63/100
8/8 - 0s - loss: 1.9387 - 15ms/epoch - 2ms/step
Epoch 64/100
8/8 - 0s - loss: 1.9358 - 19ms/epoch - 2ms/step
Epoch 65/100
8/8 - 0s - loss: 1.9347 - 17ms/epoch - 2ms/step
Epoch 66/100
8/8 - 0s - loss: 1.9332 - 20ms/epoch - 3ms/step
Epoch 67/100
8/8 - 0s - loss: 1.9329 - 16ms/epoch - 2ms/step
Epoch 68/100
8/8 - 0s - loss: 1.9300 - 17ms/epoch - 2ms/step
Epoch 69/100
8/8 - 0s - loss: 1.9289 - 22ms/epoch - 3ms/step
Epoch 70/100
8/8 - 0s - loss: 1.9286 - 17ms/epoch - 2ms/step
Epoch 71/100
8/8 - 0s - loss: 1.9259 - 13ms/epoch - 2ms/step
Epoch 72/100
8/8 - 0s - loss: 1.9250 - 16ms/epoch - 2ms/step
Epoch 73/100
8/8 - 0s - loss: 1.9231 - 13ms/epoch - 2ms/step
Epoch 74/100
8/8 - 0s - loss: 1.9217 - 14ms/epoch - 2ms/step
Epoch 75/100
8/8 - 0s - loss: 1.9199 - 17ms/epoch - 2ms/step
Epoch 76/100
8/8 - 0s - loss: 1.9189 - 21ms/epoch - 3ms/step
Epoch 77/100
8/8 - 0s - loss: 1.9176 - 15ms/epoch - 2ms/step
Epoch 78/100
8/8 - 0s - loss: 1.9163 - 16ms/epoch - 2ms/step
Epoch 79/100
8/8 - 0s - loss: 1.9146 - 14ms/epoch - 2ms/step
Epoch 80/100
8/8 - 0s - loss: 1.9132 - 15ms/epoch - 2ms/step
Epoch 81/100
8/8 - 0s - loss: 1.9118 - 14ms/epoch - 2ms/step
Epoch 82/100
8/8 - 0s - loss: 1.9100 - 15ms/epoch - 2ms/step
Epoch 83/100
8/8 - 0s - loss: 1.9083 - 15ms/epoch - 2ms/step
Epoch 84/100
8/8 - 0s - loss: 1.9070 - 25ms/epoch - 3ms/step
```

```
Epoch 85/100
8/8 - 0s - loss: 1.9074 - 12ms/epoch - 2ms/step
Epoch 86/100
8/8 - 0s - loss: 1.9039 - 18ms/epoch - 2ms/step
Epoch 87/100
8/8 - 0s - loss: 1.9027 - 13ms/epoch - 2ms/step
Epoch 88/100
8/8 - 0s - loss: 1.9010 - 15ms/epoch - 2ms/step
Epoch 89/100
8/8 - 0s - loss: 1.8994 - 16ms/epoch - 2ms/step
Epoch 90/100
8/8 - 0s - loss: 1.8978 - 15ms/epoch - 2ms/step
Epoch 91/100
8/8 - 0s - loss: 1.8968 - 14ms/epoch - 2ms/step
Epoch 92/100
8/8 - 0s - loss: 1.8952 - 15ms/epoch - 2ms/step
Epoch 93/100
8/8 - 0s - loss: 1.8956 - 20ms/epoch - 2ms/step
Epoch 94/100
8/8 - 0s - loss: 1.8925 - 16ms/epoch - 2ms/step
Epoch 95/100
8/8 - 0s - loss: 1.8906 - 16ms/epoch - 2ms/step
Epoch 96/100
8/8 - 0s - loss: 1.8891 - 13ms/epoch - 2ms/step
Epoch 97/100
8/8 - 0s - loss: 1.8877 - 11ms/epoch - 1ms/step
Epoch 98/100
8/8 - 0s - loss: 1.8860 - 14ms/epoch - 2ms/step
Epoch 99/100
8/8 - 0s - loss: 1.8851 - 17ms/epoch - 2ms/step
Epoch 100/100
8/8 - 0s - loss: 1.8838 - 9ms/epoch - 1ms/step
```

Out[ ]:
```
<keras.callbacks.History at 0x7fea3f9bc890>
```

We can evaluate the in-sample RMSE for the new model containing transferred layers from the previous model.

In [ ]:
```
from sklearn.metrics import accuracy_score
pred = model3.predict(x)
score = np.sqrt(metrics.mean_squared_error(pred,y))
print(f"Final score (RMSE): {score}")
```

```
Final score (RMSE): 1.3716589625823072
```

Module 9 Assignment

You can find the first assignment here: assignment 9

In [ ]:
```

```

Open in Colab

T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis

- For more information visit the class website.

Module 9 Material

- Part 9.1: Introduction to Keras Transfer Learning [Video] [Notebook]
- **Part 9.2: Keras Transfer Learning for Computer Vision** [Video] [Notebook]
- Part 9.3: Transfer Learning for NLP with Keras [Video] [Notebook]
- Part 9.4: Transfer Learning for Facial Feature Recognition [Video] [Notebook]
- Part 9.5: Transfer Learning for Style Transfer [Video] [Notebook]

Part 9.2: Keras Transfer Learning for Computer Vision

We will take a look at several popular pretrained neural networks for Keras. The following two sites, among others, can be great starting points to find pretrained models for use in your projects:

- TensorFlow Model Zoo
- Papers with Code

Keras contains built-in support for several pretrained models. In the Keras documentation, you can find the complete list.

Transfering Computer Vision

There are many pretrained models for computer vision. This section will show you how to obtain a pretrained model for computer vision and train just the output layer. Additionally, once we train the output layer, we will fine-tune the entire network by training all weights using by applying a low learning rate.

The Kaggle Cats vs. Dogs Dataset

We will train a neural network to recognize cats and dogs for this example. The [cats and dogs dataset] comes from a classic Kaggle competition. We can achieve a very high score on this data set through modern training techniques and ensemble learning. I based this module on a tutorial provided by Francois Chollet, one of the creators of Keras. I made some changes to his example to fit with this course.

We begin by downloading this dataset from Keras. We do not need the entire dataset to achieve high accuracy. Using a portion also speeds up training. We will use 40% of the original training data (25,000 images) for training and 10% for validation.

The dogs and cats dataset is relatively large and will not easily fit into a less than 12GB system, such as Colab. Because of this memory size, you must take additional steps to handle the data. Rather than loading the dataset as a Numpy array, as done previously in this book, we will load it as a prefetched dataset so that only the portions of the dataset currently needed are in RAM. If you wish to load the dataset, in its entirety as a Numpy array, add the batch_size=-1 option to the load command below.

In [ ]:
```
import tensorflow_datasets as tfds
import tensorflow as tf

tfds.disable_progress_bar()

train_ds, validation_ds = tfds.load(
    "cats_vs_dogs",
    split=["train[:40%]", "train[40%:50%]"],
    as_supervised=True,  # Include labels
)

num_train = tf.data.experimental.cardinality(train_ds)
num_test = tf.data.experimental.cardinality(validation_ds)

print(f"Number of training samples: {num_train}")
print(f"Number of validation samples: {num_test}")
```

```
Number of training samples: 9305
Number of validation samples: 2326
```

Looking at the Data and Augmentations

We begin by displaying several of the images from this dataset. The labels are above each image. As can be seen from the images below, 1 indicates a dog, and 0 indicates a cat.

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for i, (image, label) in enumerate(train_ds.take(9)):
```

```
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image)
    plt.title(int(label))
    plt.axis("off")
```

Upon examining the above images, another problem becomes evident. The images are of various sizes. We will standardize all images to 190x190 with the following code.

In [ ]:
```
size = (150, 150)

train_ds = train_ds.map(lambda x, y: (tf.image.resize(x, size), y))
validation_ds = validation_ds.map(lambda x, y: \
                                  (tf.image.resize(x, size), y))
```

We will batch the data and use caching and prefetching to optimize loading speed.

In [ ]:
```
batch_size = 32

train_ds = train_ds.cache().batch(batch_size).prefetch(buffer_size=10)
```

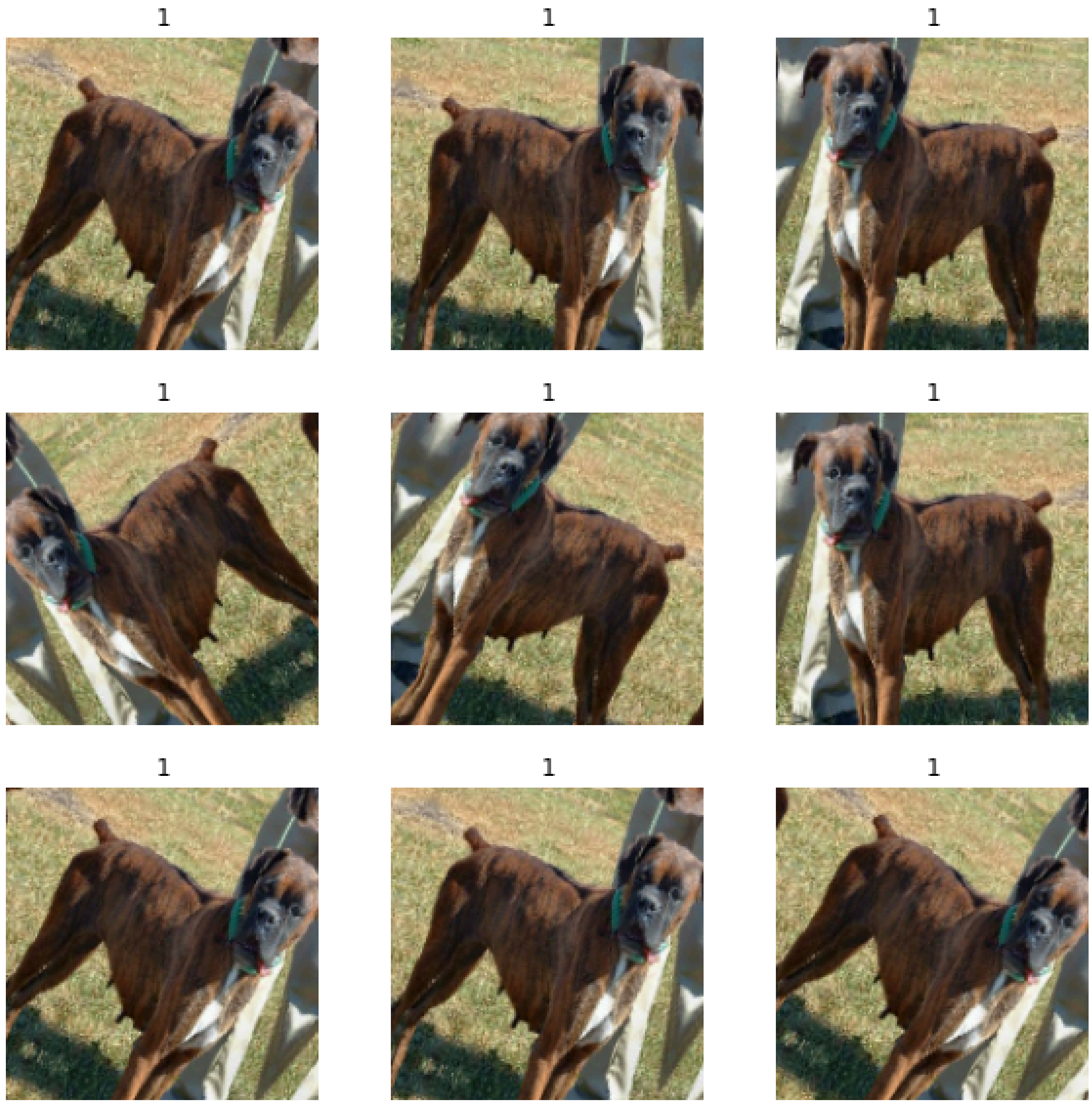
```
validation_ds = validation_ds.cache() \
    .batch(batch_size).prefetch(buffer_size=10)
```

Augmentation is a powerful computer vision technique that increases the amount of training data available to your model by altering the images in the training data. To use augmentation, we will allow horizontal flips of the images. A horizontal flip makes much more sense for cats and dogs in the real world than a vertical flip. How often do you see upside-down dogs or cats? We also include a limited degree of rotation.

In [ ]:
```
from tensorflow import keras
from tensorflow.keras import layers

data_augmentation = keras.Sequential(
    [layers.RandomFlip("horizontal"), layers.RandomRotation(0.1),]
)
```

The following code allows us to visualize the augmentation.

In [ ]:
```
import numpy as np

for images, labels in train_ds.take(1):
    plt.figure(figsize=(10, 10))
    first_image = images[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(
            tf.expand_dims(first_image, 0), training=True
        )
        plt.imshow(augmented_image[0].numpy().astype("int32"))
        plt.title(int(labels[0]))
        plt.axis("off")
```



Create a Network and Transfer Weights

We are now ready to create our new neural network with transferred weights. We will transfer the weights from an Xception neural network that contains weights trained for imagenet. We load the existing Xception neural network with **keras.applications**. There is quite a bit going on with the loading of the **base_model**, so we will examine this call piece by piece.

The base Xception neural network accepts an image of 299x299. However, we would like to use 150x150. It turns out that it is relatively easy to overcome this difference. Convolutional neural networks move a kernel across an image tensor as they scan. Keras defines the number of weights by the size of the layer's kernel, not the image that the kernel scans. As a result, we can discard the old input layer and recreate an input layer consistent with our desired image size. We specify **include_top** as false and specify our input shape.

We freeze the base model so that the model will not update existing weights as training occurs. We create the new input layer that consists of 150x150 by 3 RGB color components. These RGB components are integer numbers between 0 and 255. Neural networks deal better with floating-point numbers when you distribute them around zero. To accomplish this neural network advantage, we normalize each RGB component to between -1 and 1.

The batch normalization layers do require special consideration. We need to keep these layers in inference mode when we unfreeze the base model for fine-tuning. To do this, we make sure that the base model is running in inference mode here.

In [ ]:

```
base_model = keras.applications.Xception(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 3),
    include_top=False,
)  # Do not include the ImageNet classifier at the top.

# Freeze the base_model
base_model.trainable = False

# Create new model on top
inputs = keras.Input(shape=(150, 150, 3))
x = data_augmentation(inputs)  # Apply random data augmentation

# Pre-trained Xception weights requires that input be scaled
# from (0, 255) to a range of (-1., +1.), the rescaling layer
# outputs: `(inputs * scale) + offset`
scale_layer = keras.layers.Rescaling(scale=1 / 127.5, offset=-1)
x = scale_layer(x)

# The base model contains batchnorm layers.
# We want to keep them in inference mode
# when we unfreeze the base model for fine-tuning,
# so we make sure that the
# base_model is running in inference mode here.
x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)  # Regularize with dropout
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0         
                                                                 
 sequential (Sequential)     (None, 150, 150, 3)       0         
                                                                 
 rescaling (Rescaling)       (None, 150, 150, 3)       0         
                                                                 
 xception (Functional)       (None, 5, 5, 2048)        20861480  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout (Dropout)           (None, 2048)              0         
                                                                 
 dense (Dense)               (None, 1)                 2049      
                                                                 
=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
```

Next, we compile and fit the model. The fitting will use the Adam optimizer; because we are performing binary classification, we use the binary cross-entropy loss function, as we have done before.

In [ ]:

```
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)

epochs = 20
model.fit(train_ds, epochs=epochs, validation_data=validation_ds)
```

```
Epoch 1/20
291/291 [==============================] - 31s 70ms/step - loss: 0.1735 - bi
nary_accuracy: 0.9240 - val_loss: 0.0831 - val_binary_accuracy: 0.9678
Epoch 2/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1256 - bi
nary_accuracy: 0.9463 - val_loss: 0.0773 - val_binary_accuracy: 0.9703
Epoch 3/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1140 - bi
nary_accuracy: 0.9536 - val_loss: 0.0750 - val_binary_accuracy: 0.9708
Epoch 4/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1097 - bi
nary_accuracy: 0.9556 - val_loss: 0.0729 - val_binary_accuracy: 0.9729
Epoch 5/20
291/291 [==============================] - 12s 41ms/step - loss: 0.0996 - bi
nary_accuracy: 0.9589 - val_loss: 0.0717 - val_binary_accuracy: 0.9746
Epoch 6/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0979 - bi
nary_accuracy: 0.9587 - val_loss: 0.0734 - val_binary_accuracy: 0.9690
Epoch 7/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1007 - bi
nary_accuracy: 0.9601 - val_loss: 0.0738 - val_binary_accuracy: 0.9703
Epoch 8/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0968 - bi
nary_accuracy: 0.9591 - val_loss: 0.0709 - val_binary_accuracy: 0.9729
Epoch 9/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0940 - bi
nary_accuracy: 0.9607 - val_loss: 0.0713 - val_binary_accuracy: 0.9733
Epoch 10/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1019 - bi
nary_accuracy: 0.9599 - val_loss: 0.0731 - val_binary_accuracy: 0.9690
Epoch 11/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0904 - bi
nary_accuracy: 0.9640 - val_loss: 0.0694 - val_binary_accuracy: 0.9738
Epoch 12/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0953 - bi
nary_accuracy: 0.9629 - val_loss: 0.0777 - val_binary_accuracy: 0.9712
Epoch 13/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0922 - bi
nary_accuracy: 0.9635 - val_loss: 0.0735 - val_binary_accuracy: 0.9738
Epoch 14/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0975 - bi
nary_accuracy: 0.9610 - val_loss: 0.0714 - val_binary_accuracy: 0.9733
Epoch 15/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0916 - bi
nary_accuracy: 0.9634 - val_loss: 0.0770 - val_binary_accuracy: 0.9699
Epoch 16/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0957 - bi
nary_accuracy: 0.9605 - val_loss: 0.0713 - val_binary_accuracy: 0.9721
Epoch 17/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0899 - bi
nary_accuracy: 0.9638 - val_loss: 0.0731 - val_binary_accuracy: 0.9725
Epoch 18/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0937 - bi
nary_accuracy: 0.9620 - val_loss: 0.0755 - val_binary_accuracy: 0.9712
Epoch 19/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0907 - bi
```

```
nary_accuracy: 0.9627 - val_loss: 0.0718 - val_binary_accuracy: 0.9729
Epoch 20/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0899 - bi
nary_accuracy: 0.9652 - val_loss: 0.0694 - val_binary_accuracy: 0.9746
```

Out[ ]:
```
<keras.callbacks.History at 0x7f0c2cfbf410>
```

The training above shows that the validation accuracy reaches the mid 90% range. This accuracy is good; however, we can do better.

Fine-Tune the Model

Finally, we will fine-tune the model. First, we set all weights to trainable and then train the neural network with a low learning rate (1e-5). This fine-tuning results in an accuracy in the upper 90% range. The fine-tuning allows all weights in the neural network to adjust slightly to optimize for the dogs/cats data.

In [ ]:
```
# Unfreeze the base_model. Note that it keeps running in inference mode
# since we passed `training=False` when calling it. This means that
# the batchnorm layers will not update their batch statistics.
# This prevents the batchnorm layers from undoing all the training
# we've done so far.
base_model.trainable = True
model.summary()

model.compile(
    optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)

epochs = 10
model.fit(train_ds, epochs=epochs, validation_data=validation_ds)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0         
                                                                 
 sequential (Sequential)     (None, 150, 150, 3)       0         
                                                                 
 rescaling (Rescaling)       (None, 150, 150, 3)       0         
                                                                 
 xception (Functional)       (None, 5, 5, 2048)        20861480  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout (Dropout)           (None, 2048)              0         
                                                                 
 dense (Dense)               (None, 1)                 2049      
                                                                 
=================================================================
Total params: 20,863,529
Trainable params: 20,809,001
Non-trainable params: 54,528
_________________________________________________________________
Epoch 1/10
291/291 [==============================] - 48s 144ms/step - loss: 0.0771 - b
inary_accuracy: 0.9712 - val_loss: 0.0533 - val_binary_accuracy: 0.9772
Epoch 2/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0571 - b
inary_accuracy: 0.9774 - val_loss: 0.0484 - val_binary_accuracy: 0.9811
Epoch 3/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0412 - b
inary_accuracy: 0.9830 - val_loss: 0.0467 - val_binary_accuracy: 0.9819
Epoch 4/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0354 - b
inary_accuracy: 0.9861 - val_loss: 0.0543 - val_binary_accuracy: 0.9785
Epoch 5/10
291/291 [==============================] - 41s 141ms/step - loss: 0.0313 - b
inary_accuracy: 0.9876 - val_loss: 0.0490 - val_binary_accuracy: 0.9807
Epoch 6/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0231 - b
inary_accuracy: 0.9911 - val_loss: 0.0625 - val_binary_accuracy: 0.9776
Epoch 7/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0227 - b
inary_accuracy: 0.9909 - val_loss: 0.0579 - val_binary_accuracy: 0.9798
Epoch 8/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0195 - b
inary_accuracy: 0.9937 - val_loss: 0.0493 - val_binary_accuracy: 0.9837
Epoch 9/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0136 - b
inary_accuracy: 0.9952 - val_loss: 0.0447 - val_binary_accuracy: 0.9837
Epoch 10/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0162 - b
inary_accuracy: 0.9944 - val_loss: 0.0548 - val_binary_accuracy: 0.9819
```

Out[ ]:
```
<keras.callbacks.History at 0x7f0c2ceaafd0>
```

t81_558_class_09_3_transfer_nlp

May 24, 2025

1 T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning * Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis * For more information visit the class website.

2 Module 9 Material

- Part 9.1: Introduction to Keras Transfer Learning [Video] [Notebook]
- Part 9.2: Keras Transfer Learning for Computer Vision [Video] [Notebook]
- **Part 9.3: Transfer Learning for NLP with Keras** [Video] [Notebook]
- Part 9.4: Transfer Learning for Facial Feature Recognition [Video] [Notebook]
- Part 9.5: Transfer Learning for Style Transfer [Video] [Notebook]

3 Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

[ ]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

4 Part 9.3: Transfer Learning for NLP with Keras

You will commonly use transfer learning with Natural Language Processing (NLP). Word embeddings are a common means of transfer learning in NLP where network layers map words to vectors. Third parties trained neural networks on a large corpus of text to learn these embeddings. We will use these vectors as the input to the neural network rather than the actual characters of words.

This course has an entire module covering NLP; however, we use word embeddings to perform sentiment analysis in this module. We will specifically attempt to classify if a text sample is speaking in a positive or negative tone.

The following three sources were helpful for the creation of this section.

- Universal sentence encoder [Cite:cer2018universal]. arXiv preprint arXiv:1803.11175)
- Deep Transfer Learning for Natural Language Processing: Text Classification with Universal Embeddings [Cite:howard2018universal]
- Keras Tutorial: How to Use Google's Universal Sentence Encoder for Spam Classification

These examples use TensorFlow Hub, which allows pretrained models to be loaded into TensorFlow easily. To install TensorHub use the following commands.

[ ]:
```
# HIDE OUTPUT
!pip install tensorflow_hub
```

```
Requirement already satisfied: tensorflow_hub in /usr/local/lib/python3.7/dist-
packages (0.12.0)
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_hub) (1.19.5)
Requirement already satisfied: protobuf>=3.8.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_hub) (3.17.3)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-
packages (from protobuf>=3.8.0->tensorflow_hub) (1.15.0)
```

It is also necessary to install TensorFlow Datasets, which you can install with the following command.

[ ]:
```
# HIDE OUTPUT
!pip install tensorflow_datasets
```

```
Requirement already satisfied: tensorflow_datasets in
/usr/local/lib/python3.7/dist-packages (4.0.1)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.7/dist-packages (from tensorflow_datasets) (5.4.0)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_datasets) (3.17.3)
Requirement already satisfied: attrs>=18.1.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_datasets) (21.4.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.7/dist-
packages (from tensorflow_datasets) (1.1.0)
Requirement already satisfied: promise in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (2.3)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (0.3.4)
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (1.0.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (1.15.0)
Requirement already satisfied: tensorflow-metadata in
/usr/local/lib/python3.7/dist-packages (from tensorflow_datasets) (1.6.0)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (0.16.0)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow_datasets) (2.23.0)
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (4.62.3)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (0.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (1.19.5)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests>=2.19.0->tensorflow_datasets) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (2021.10.8)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-
packages (from importlib-resources->tensorflow_datasets) (3.7.0)
Requirement already satisfied: googleapis-common-protos<2,>=1.52.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-
metadata->tensorflow_datasets) (1.54.0)
```

Movie reviews are a good source of training data for sentiment analysis. These reviews are textual, and users give them a star rating which indicates if the viewer had a positive or negative experience with the movie. Load the Internet Movie DataBase (IMDB) reviews data set. This example is based on a TensorFlow example that you can find here.

[ ]:
```python
# HIDE OUTPUT
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds

train_data, test_data = tfds.load(name="imdb_reviews",
                                  split=["train", "test"],
                                  batch_size=-1, as_supervised=True)

train_examples, train_labels = tfds.as_numpy(train_data)
test_examples, test_labels = tfds.as_numpy(test_data)

# /Users/jheaton/tensorflow_datasets/imdb_reviews/plain_text/0.1.0
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download:
80.23 MiB, generated: Unknown size, total: 80.23 MiB) to
/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0…
Dl Completed…: 0 url [00:00, ? url/s]
```

```
Dl Size…: 0 MiB [00:00, ? MiB/s]
```

```
0 examples [00:00, ? examples/s]
```

```
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete0GRP97/imdb_reviews-train.tfrecord
```

```
  0%|          | 0/25000 [00:00<?, ? examples/s]
```

```
0 examples [00:00, ? examples/s]
```

```
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete0GRP97/imdb_reviews-test.tfrecord
```

```
  0%|          | 0/25000 [00:00<?, ? examples/s]
```

```
0 examples [00:00, ? examples/s]
```

```
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete0GRP97/imdb_reviews-unsupervised.tfrecord
```

```
  0%|          | 0/50000 [00:00<?, ? examples/s]
```

```
WARNING:absl:Dataset is using deprecated text encoder API which will be removed
soon. Please use the plain_text version of the dataset and migrate to
`tensorflow_text`.
```

Dataset imdb_reviews downloaded and prepared to

/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0. Subsequent calls will

reuse this data.

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-
packages/tensorflow_datasets/core/dataset_builder.py:598: get_single_element
(from tensorflow.python.data.experimental.ops.get_single_element) is deprecated
and will be removed in a future version.
Instructions for updating:
Use `tf.data.Dataset.get_single_element()`.
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-
packages/tensorflow_datasets/core/dataset_builder.py:598: get_single_element
(from tensorflow.python.data.experimental.ops.get_single_element) is deprecated
and will be removed in a future version.
Instructions for updating:
Use `tf.data.Dataset.get_single_element()`.
```

Load a pretrained embedding model called gnews-swivel-20dim. Google trained this network on GNEWS data and can convert raw text into vectors.

[ ]:
```
model = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"
hub_layer = hub.KerasLayer(model, output_shape=[20], input_shape=[],
                           dtype=tf.string, trainable=True)
```

The following code displays three movie reviews. This display allows you to see the actual data.

```
[ ]: train_examples[:3]
```

```
[ ]: array([b"This was an absolutely terrible movie. Don't be lured in by Christopher
     Walken or Michael Ironside. Both are great actors, but this must simply be their
     worst role in history. Even their great acting could not redeem this movie's
     ridiculous storyline. This movie is an early nineties US propaganda piece. The
     most pathetic scenes were those when the Columbian rebels were making their
     cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love
     affair with Walken was nothing but a pathetic emotional plug in a movie that was
     devoid of any real meaning. I am disappointed that there are movies like this,
     ruining actor's like Christopher Walken's good name. I could barely sit through
     it.",
            b'I have been known to fall asleep during films, but this is usually due
     to a combination of things including, really tired, being warm and comfortable
     on the sette and having just eaten a lot. However on this occasion I fell asleep
     because the film was rubbish. The plot development was constant. Constantly slow
     and boring. Things seemed to happen, but with no explanation of what was causing
     them or why. I admit, I may have missed part of the film, but i watched the
     majority of it and everything just seemed to happen of its own accord without
     any real concern for anything else. I cant recommend this film at all.',
            b'Mann photographs the Alberta Rocky Mountains in a superb fashion, and
     Jimmy Stewart and Walter Brennan give enjoyable performances as they always seem
     to do. <br /><br />But come on Hollywood - a Mountie telling the people of
     Dawson City, Yukon to elect themselves a marshal (yes a marshal!) and to enforce
     the law themselves, then gunfighters battling it out on the streets for control
     of the town? <br /><br />Nothing even remotely resembling that happened on the
     Canadian side of the border during the Klondike gold rush. Mr. Mann and company
     appear to have mistaken Dawson City for Deadwood, the Canadian North for the
     American Wild West.<br /><br />Canadian viewers be prepared for a Reefer Madness
     type of enjoyable howl with this ludicrous plot, or, to shake your head in
     disgust.'],
           dtype=object)
```

The embedding layer can convert each to 20-number vectors, which the neural network receives as input in place of the actual words.

```
[ ]: hub_layer(train_examples[:3])
```

```
[ ]: <tf.Tensor: shape=(3, 20), dtype=float32, numpy=
     array([[ 1.7657859 , -3.882232  ,  3.913424  , -1.5557289 , -3.3362343 ,
             -1.7357956 , -1.9954445 ,  1.298955  ,  5.081597  , -1.1041285 ,
             -2.0503852 , -0.7267516 , -0.6567596 ,  0.24436145, -3.7208388 ,
              2.0954835 ,  2.2969332 , -2.0689783 , -2.9489715 , -1.1315986 ],
            [ 1.8804485 , -2.5852385 ,  3.4066994 ,  1.0982676 , -4.056685  ,
             -4.891284  , -2.7855542 ,  1.3874227 ,  3.8476458 , -0.9256539 ,
             -1.896706  ,  1.2113281 ,  0.11474716,  0.76209456, -4.8791065 ,
```

```
     2.906149  ,  4.7087674 , -2.3652055 , -3.5015903 , -1.6390051 ],
   [ 0.71152216, -0.63532174,  1.7385626 , -1.1168287 , -0.54515934,
    -1.1808155 ,  0.09504453,  1.4653089 ,  0.66059506,  0.79308075,
    -2.2268343 ,  0.07446616, -1.4075902 , -0.706454  , -1.907037  ,
     1.4419788 ,  1.9551864 , -0.42660046, -2.8022065 ,  0.43727067]],
  dtype=float32)>
```

We add additional layers to classify the movie reviews as either positive or negative.

```
[ ]: model = tf.keras.Sequential()
     model.add(hub_layer)
     model.add(tf.keras.layers.Dense(16, activation='relu'))
     model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 20)                400020

 dense (Dense)               (None, 16)                336

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 400,373
Trainable params: 400,373
Non-trainable params: 0
_________________________________________________________________
```

We are now ready to compile the neural network. For this application, we use the adam training method for binary classification. We also save the initial random weights for later to start over easily.

```
[ ]: model.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
     init_weights = model.get_weights()
```

Before fitting, we split the training data into the train and validation sets.

```
[ ]: x_val = train_examples[:10000]
     partial_x_train = train_examples[10000:]

     y_val = train_labels[:10000]
     partial_y_train = train_labels[10000:]
```

We can now fit the neural network. This fitting will run for 40 epochs and allow us to evaluate the effectiveness of the neural network, as measured by the training set.

```
[ ]: history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=40,
                         batch_size=512,
                         validation_data=(x_val, y_val),
                         verbose=1)
```

```
Epoch 1/40
30/30 [==============================] - 4s 80ms/step - loss: 1.5554 - accuracy:
0.5515 - val_loss: 0.8048 - val_accuracy: 0.5865
Epoch 2/40
30/30 [==============================] - 2s 73ms/step - loss: 0.7600 - accuracy:
0.6011 - val_loss: 0.7107 - val_accuracy: 0.6230
Epoch 3/40
30/30 [==============================] - 2s 82ms/step - loss: 0.6541 - accuracy:
0.6561 - val_loss: 0.6263 - val_accuracy: 0.6662
Epoch 4/40
30/30 [==============================] - 2s 70ms/step - loss: 0.5864 - accuracy:
0.6953 - val_loss: 0.5818 - val_accuracy: 0.6978
Epoch 5/40
30/30 [==============================] - 2s 77ms/step - loss: 0.5493 - accuracy:
0.7248 - val_loss: 0.5551 - val_accuracy: 0.7190
Epoch 6/40
30/30 [==============================] - 2s 77ms/step - loss: 0.5222 - accuracy:
0.7452 - val_loss: 0.5336 - val_accuracy: 0.7338
Epoch 7/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4990 - accuracy:
0.7618 - val_loss: 0.5146 - val_accuracy: 0.7477
Epoch 8/40
30/30 [==============================] - 1s 36ms/step - loss: 0.4765 - accuracy:
0.7768 - val_loss: 0.4967 - val_accuracy: 0.7637
Epoch 9/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4551 - accuracy:
0.7925 - val_loss: 0.4798 - val_accuracy: 0.7739
Epoch 10/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4335 - accuracy:
0.8062 - val_loss: 0.4629 - val_accuracy: 0.7864
Epoch 11/40
30/30 [==============================] - 1s 36ms/step - loss: 0.4129 - accuracy:
0.8191 - val_loss: 0.4466 - val_accuracy: 0.7971
Epoch 12/40
30/30 [==============================] - 1s 36ms/step - loss: 0.3915 - accuracy:
0.8315 - val_loss: 0.4309 - val_accuracy: 0.8086
Epoch 13/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3710 - accuracy:
0.8431 - val_loss: 0.4159 - val_accuracy: 0.8180
```

```
Epoch 14/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3510 - accuracy:
0.8544 - val_loss: 0.4017 - val_accuracy: 0.8262
Epoch 15/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3310 - accuracy:
0.8643 - val_loss: 0.3883 - val_accuracy: 0.8315
Epoch 16/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3118 - accuracy:
0.8740 - val_loss: 0.3754 - val_accuracy: 0.8385
Epoch 17/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2932 - accuracy:
0.8846 - val_loss: 0.3638 - val_accuracy: 0.8454
Epoch 18/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2747 - accuracy:
0.8930 - val_loss: 0.3533 - val_accuracy: 0.8495
Epoch 19/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2568 - accuracy:
0.9030 - val_loss: 0.3434 - val_accuracy: 0.8539
Epoch 20/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2396 - accuracy:
0.9094 - val_loss: 0.3360 - val_accuracy: 0.8567
Epoch 21/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2246 - accuracy:
0.9183 - val_loss: 0.3298 - val_accuracy: 0.8605
Epoch 22/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2104 - accuracy:
0.9248 - val_loss: 0.3234 - val_accuracy: 0.8633
Epoch 23/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1971 - accuracy:
0.9295 - val_loss: 0.3192 - val_accuracy: 0.8663
Epoch 24/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1856 - accuracy:
0.9359 - val_loss: 0.3173 - val_accuracy: 0.8678
Epoch 25/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1739 - accuracy:
0.9417 - val_loss: 0.3147 - val_accuracy: 0.8704
Epoch 26/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1631 - accuracy:
0.9464 - val_loss: 0.3144 - val_accuracy: 0.8713
Epoch 27/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1538 - accuracy:
0.9508 - val_loss: 0.3134 - val_accuracy: 0.8725
Epoch 28/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1454 - accuracy:
0.9540 - val_loss: 0.3158 - val_accuracy: 0.8723
Epoch 29/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1372 - accuracy:
0.9573 - val_loss: 0.3174 - val_accuracy: 0.8739
```

```
Epoch 30/40
30/30 [==============================] - 1s 38ms/step - loss: 0.1287 - accuracy:
0.9605 - val_loss: 0.3174 - val_accuracy: 0.8748
Epoch 31/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1211 - accuracy:
0.9634 - val_loss: 0.3202 - val_accuracy: 0.8752
Epoch 32/40
30/30 [==============================] - 1s 38ms/step - loss: 0.1140 - accuracy:
0.9657 - val_loss: 0.3226 - val_accuracy: 0.8745
Epoch 33/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1067 - accuracy:
0.9683 - val_loss: 0.3272 - val_accuracy: 0.8738
Epoch 34/40
30/30 [==============================] - 1s 38ms/step - loss: 0.1001 - accuracy:
0.9707 - val_loss: 0.3323 - val_accuracy: 0.8737
Epoch 35/40
30/30 [==============================] - 1s 38ms/step - loss: 0.0934 - accuracy:
0.9734 - val_loss: 0.3352 - val_accuracy: 0.8737
Epoch 36/40
30/30 [==============================] - 1s 38ms/step - loss: 0.0871 - accuracy:
0.9761 - val_loss: 0.3403 - val_accuracy: 0.8740
Epoch 37/40
30/30 [==============================] - 1s 38ms/step - loss: 0.0817 - accuracy:
0.9788 - val_loss: 0.3449 - val_accuracy: 0.8729
Epoch 38/40
30/30 [==============================] - 1s 36ms/step - loss: 0.0765 - accuracy:
0.9805 - val_loss: 0.3508 - val_accuracy: 0.8739
Epoch 39/40
30/30 [==============================] - 1s 37ms/step - loss: 0.0711 - accuracy:
0.9820 - val_loss: 0.3562 - val_accuracy: 0.8738
Epoch 40/40
30/30 [==============================] - 1s 37ms/step - loss: 0.0661 - accuracy:
0.9847 - val_loss: 0.3626 - val_accuracy: 0.8728
```

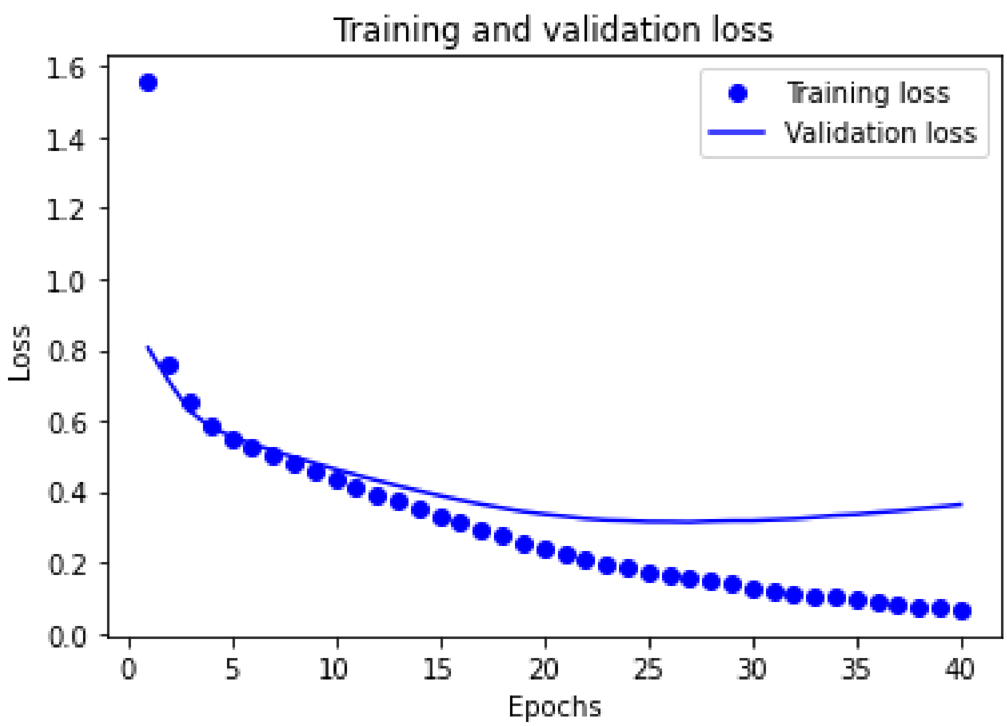
4.1 Benefits of Early Stopping

While we used a validation set, we fit the neural network without early stopping. This dataset is complex enough to allow us to see the benefit of early stopping. We will examine how accuracy and loss progressed for training and validation sets. Loss measures the degree to which the neural network was confident in incorrect answers. Accuracy is the percentage of correct classifications, regardless of the neural network's confidence.

We begin by looking at the loss as we fit the neural network.

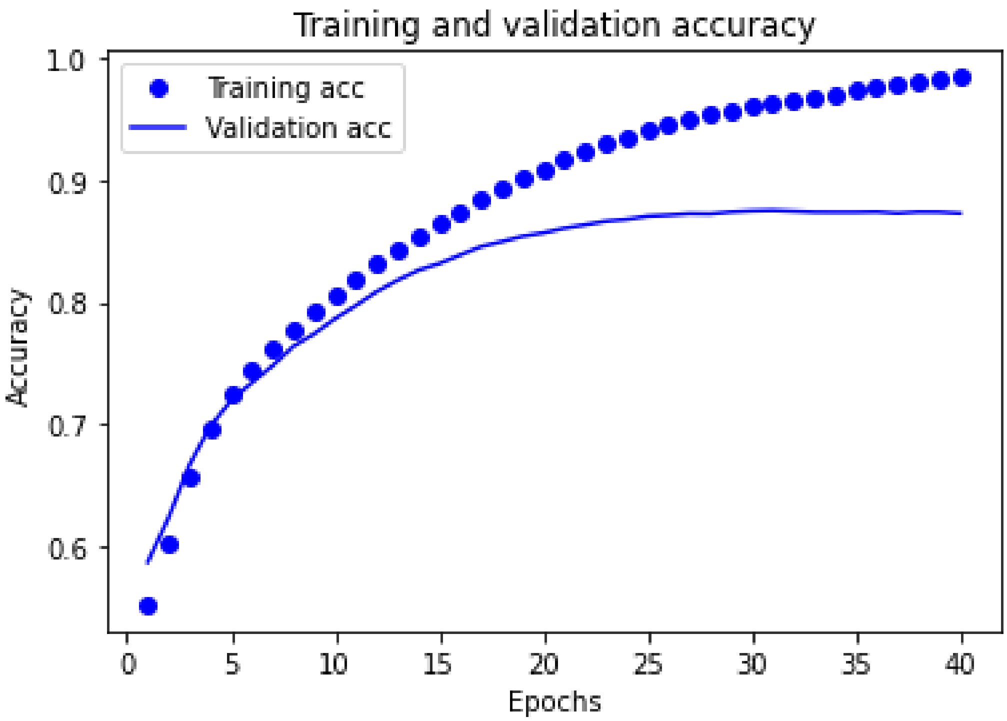
[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

history_dict = history.history
```

```python
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



We can see that training and validation loss are similar early in the fitting. However, as fitting continues and overfitting sets in, training and validation loss diverge from each other. Training loss continues to fall consistently. However, once overfitting happens, the validation loss no longer falls and eventually begins to increase a bit. Early stopping, which we saw earlier in this course, can prevent some overfitting.

[ ]:
```
plt.clf()   # clear figure

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



The accuracy graph tells a similar story. Now let's repeat the fitting with early stopping. We begin by creating an early stopping monitor and restoring the network's weights to random. Once this is complete, we can fit the neural network with the early stopping monitor enabled.

[ ]:
```
from tensorflow.keras.callbacks import EarlyStopping

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=5, verbose=1, mode='auto',
        restore_best_weights=True)

model.set_weights(init_weights)

history = model.fit(partial_x_train,
```

```
                    partial_y_train,
                    epochs=40,
                    batch_size=512,
                    callbacks=[monitor],
                    validation_data=(x_val, y_val),
                    verbose=1)
```

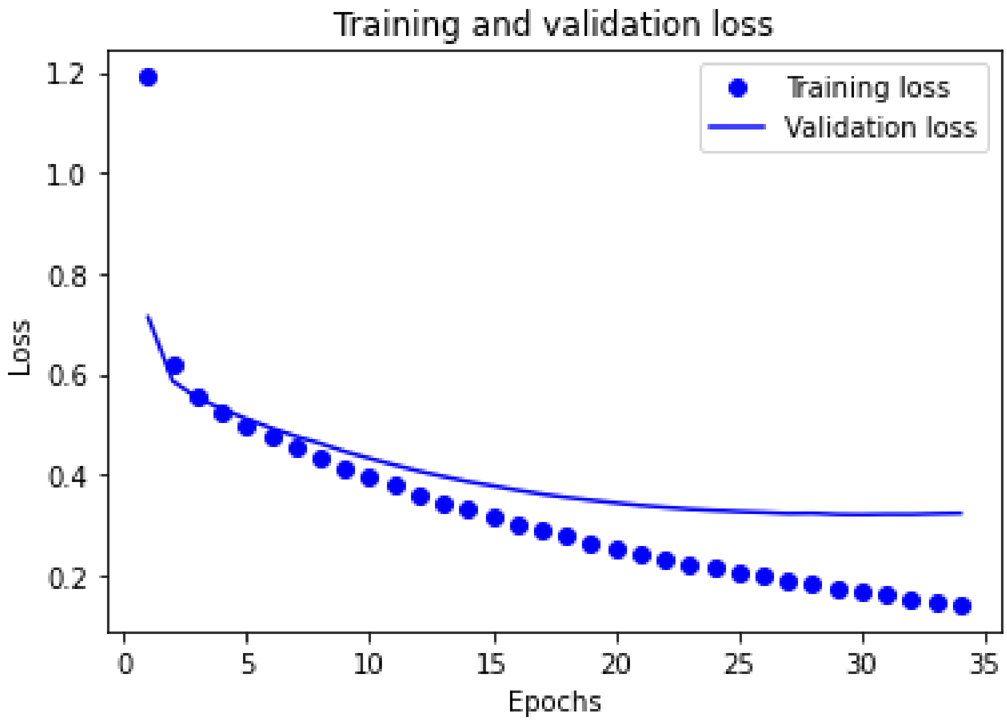
```
Epoch 1/40
30/30 [==============================] - 1s 38ms/step - loss: 1.1912 - accuracy:
0.5643 - val_loss: 0.7129 - val_accuracy: 0.6332
Epoch 2/40
30/30 [==============================] - 1s 37ms/step - loss: 0.6202 - accuracy:
0.6749 - val_loss: 0.5862 - val_accuracy: 0.6922
Epoch 3/40
30/30 [==============================] - 1s 37ms/step - loss: 0.5531 - accuracy:
0.7251 - val_loss: 0.5525 - val_accuracy: 0.7208
Epoch 4/40
30/30 [==============================] - 1s 36ms/step - loss: 0.5228 - accuracy:
0.7483 - val_loss: 0.5308 - val_accuracy: 0.7403
Epoch 5/40
30/30 [==============================] - 1s 38ms/step - loss: 0.4981 - accuracy:
0.7671 - val_loss: 0.5106 - val_accuracy: 0.7569
Epoch 6/40
30/30 [==============================] - 1s 38ms/step - loss: 0.4754 - accuracy:
0.7823 - val_loss: 0.4925 - val_accuracy: 0.7690
Epoch 7/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4542 - accuracy:
0.7959 - val_loss: 0.4759 - val_accuracy: 0.7803
Epoch 8/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4342 - accuracy:
0.8089 - val_loss: 0.4619 - val_accuracy: 0.7883
Epoch 9/40
30/30 [==============================] - 1s 38ms/step - loss: 0.4143 - accuracy:
0.8215 - val_loss: 0.4455 - val_accuracy: 0.7998
Epoch 10/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3960 - accuracy:
0.8318 - val_loss: 0.4319 - val_accuracy: 0.8090
Epoch 11/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3784 - accuracy:
0.8390 - val_loss: 0.4190 - val_accuracy: 0.8166
Epoch 12/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3613 - accuracy:
0.8483 - val_loss: 0.4068 - val_accuracy: 0.8237
Epoch 13/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3453 - accuracy:
0.8570 - val_loss: 0.3962 - val_accuracy: 0.8291
Epoch 14/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3301 - accuracy:
```

```
0.8644 - val_loss: 0.3861 - val_accuracy: 0.8360
Epoch 15/40
30/30 [==============================] - 1s 39ms/step - loss: 0.3157 - accuracy:
0.8735 - val_loss: 0.3778 - val_accuracy: 0.8366
Epoch 16/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3022 - accuracy:
0.8784 - val_loss: 0.3690 - val_accuracy: 0.8419
Epoch 17/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2888 - accuracy:
0.8866 - val_loss: 0.3612 - val_accuracy: 0.8465
Epoch 18/40
30/30 [==============================] - 1s 39ms/step - loss: 0.2765 - accuracy:
0.8917 - val_loss: 0.3546 - val_accuracy: 0.8500
Epoch 19/40
30/30 [==============================] - 1s 36ms/step - loss: 0.2645 - accuracy:
0.8961 - val_loss: 0.3490 - val_accuracy: 0.8524
Epoch 20/40
30/30 [==============================] - 1s 36ms/step - loss: 0.2533 - accuracy:
0.9024 - val_loss: 0.3436 - val_accuracy: 0.8554
Epoch 21/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2433 - accuracy:
0.9065 - val_loss: 0.3386 - val_accuracy: 0.8567
Epoch 22/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2333 - accuracy:
0.9108 - val_loss: 0.3348 - val_accuracy: 0.8590
Epoch 23/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2231 - accuracy:
0.9165 - val_loss: 0.3312 - val_accuracy: 0.8615
Epoch 24/40
30/30 [==============================] - 1s 50ms/step - loss: 0.2142 - accuracy:
0.9206 - val_loss: 0.3287 - val_accuracy: 0.8630
Epoch 25/40
30/30 [==============================] - 1s 48ms/step - loss: 0.2054 - accuracy:
0.9247 - val_loss: 0.3264 - val_accuracy: 0.8639
Epoch 26/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1972 - accuracy:
0.9299 - val_loss: 0.3247 - val_accuracy: 0.8660
Epoch 27/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1891 - accuracy:
0.9327 - val_loss: 0.3225 - val_accuracy: 0.8668
Epoch 28/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1818 - accuracy:
0.9354 - val_loss: 0.3231 - val_accuracy: 0.8656
Epoch 29/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1746 - accuracy:
0.9384 - val_loss: 0.3208 - val_accuracy: 0.8685
Epoch 30/40
30/30 [==============================] - 1s 36ms/step - loss: 0.1671 - accuracy:
```

```
0.9419 - val_loss: 0.3203 - val_accuracy: 0.8694
Epoch 31/40
30/30 [==============================] - 1s 36ms/step - loss: 0.1605 - accuracy:
0.9445 - val_loss: 0.3210 - val_accuracy: 0.8688
Epoch 32/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1539 - accuracy:
0.9480 - val_loss: 0.3209 - val_accuracy: 0.8699
Epoch 33/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1475 - accuracy:
0.9508 - val_loss: 0.3220 - val_accuracy: 0.8700
Epoch 34/40
29/30 [============================>.] - ETA: 0s - loss: 0.1419 - accuracy:
0.9528Restoring model weights from the end of the best epoch: 29.
30/30 [==============================] - 1s 38ms/step - loss: 0.1414 - accuracy:
0.9531 - val_loss: 0.3231 - val_accuracy: 0.8704
Epoch 00034: early stopping
```

The training history chart is now shorter because we stopped earlier.

[ ]:
```
history_dict = history.history
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



Finally, we evaluate the accuracy for the best neural network before early stopping occured.

[ ]:
```
from sklearn.metrics import accuracy_score
import numpy as np

pred = model.predict(x_val)
# Use 0.5 as the threshold
predict_classes = pred.flatten()>0.5

correct = accuracy_score(y_val,predict_classes)
print(f"Accuracy: {correct}")
```

```
Accuracy: 0.8685
```

t81_558_class_09_4_facial_points

May 24, 2025

1 T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning * Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis * For more information visit the class website.

2 Module 9 Material

- Part 9.1: Introduction to Keras Transfer Learning [Video] [Notebook]
- Part 9.2: Keras Transfer Learning for Computer Vision [Video] [Notebook]
- Part 9.3: Transfer Learning for NLP with Keras [Video] [Notebook]
- **Part 9.4: Transfer Learning for Facial Feature Recognition** [Video] [Notebook]
- Part 9.5: Transfer Learning for Style Transfer [Video] [Notebook]

3 Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

[ ]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

4 Part 9.4: Transfer Learning for Facial Points and GANs

I designed this notebook to work with Google Colab. You can run it locally; however, you might need to adjust some of the installation scripts contained in this notebook.

This part will see how we can use a 3rd party neural network to detect facial features, particularly the location of an individual's eyes. By locating eyes, we can crop portraits consistently. Previously, we saw that GANs could convert a random vector into a realistic-looking portrait. We can also perform the reverse and convert an actual photograph into a numeric vector. If we convert two images into these vectors, we can produce a video that transforms between the two images.

NVIDIA trained StyleGAN on portraits consistently cropped with the eyes always in the same location. To successfully convert an image to a vector, we must crop the image similarly to how NVIDIA used cropping.

The code presented here allows you to choose a starting and ending image and use StyleGAN2 to produce a "morph" video between the two pictures. The preprocessing code will lock in on the exact positioning of each image, so your crop does not have to be perfect. The main point of your crop is for you to remove anything else that might be confused for a face. If multiple faces are detected, you will receive an error.

Also, make sure you have selected a GPU Runtime from CoLab. Choose "Runtime," then "Change Runtime Type," and choose GPU for "Hardware Accelerator."

These settings allow you to change the high-level configuration. The number of steps determines how long your resulting video is. The video plays at 30 frames a second, so 150 is 5 seconds. You can also specify freeze steps to leave the video unchanged at the beginning and end. You will not likely need to change the network.

[ ]:
```
NETWORK = "https://nvlabs-fi-cdn.nvidia.com/"\
  "stylegan2-ada-pytorch/pretrained/ffhq.pkl"
STEPS = 150
FPS = 30
FREEZE_STEPS = 30
```

4.1 Upload Starting and Ending Images

We will begin by uploading a starting and ending image. The Colab service uploads these images. If you are running this code outside of Colab, these images are likely somewhere on your computer, and you provide the path to these files using the **SOURCE** and **TARGET** variables.

Choose your starting image.

[ ]:
```
# HIDE OUTPUT
import os
from google.colab import files

uploaded = files.upload()

if len(uploaded) != 1:
  print("Upload exactly 1 file for source.")
else:
  for k, v in uploaded.items():
    _, ext = os.path.splitext(k)
    os.remove(k)
    SOURCE_NAME = f"source{ext}"
    open(SOURCE_NAME, 'wb').write(v)
```

```
<IPython.core.display.HTML object>
```

```
Saving about-jeff-heaton-2020.jpg to about-jeff-heaton-2020.jpg
```

Also, choose your ending image.

[ ]:
```
# HIDE OUTPUT
uploaded = files.upload()

if len(uploaded) != 1:
  print("Upload exactly 1 file for target.")
else:
  for k, v in uploaded.items():
    _, ext = os.path.splitext(k)
    os.remove(k)
    TARGET_NAME = f"target{ext}"
    open(TARGET_NAME, 'wb').write(v)
```

```
<IPython.core.display.HTML object>
```

```
Saving thor.jpg to thor.jpg
```

4.2 Install Software

Some software must be installed into Colab, for this notebook to work. We are specifically using these technologies:

- Training Generative Adversarial Networks with Limited Data Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, Timo Aila
- One-millisecond face alignment with an ensemble of regression trees Vahid Kazemi, Josephine Sullivan

[ ]:
```
# HIDE OUTPUT
!wget http://dlib.net/files/shape_predictor_5_face_landmarks.dat.bz2
!bzip2 -d shape_predictor_5_face_landmarks.dat.bz2
```

```
--2022-01-31 02:50:46--
http://dlib.net/files/shape_predictor_5_face_landmarks.dat.bz2
Resolving dlib.net (dlib.net)… 107.180.26.78
Connecting to dlib.net (dlib.net)|107.180.26.78|:80… connected.
HTTP request sent, awaiting response… 200 OK
Length: 5706710 (5.4M)
Saving to: ‘shape_predictor_5_face_landmarks.dat.bz2.1’

shape_predictor_5_f 100%[===================>]   5.44M  22.4MB/s    in 0.2s

2022-01-31 02:50:46 (22.4 MB/s) - ‘shape_predictor_5_face_landmarks.dat.bz2.1’
saved [5706710/5706710]

bzip2: Output file shape_predictor_5_face_landmarks.dat already exists.
```

[ ]:
```
# HIDE OUTPUT
import sys
!git clone https://github.com/NVlabs/stylegan2-ada-pytorch.git
```

```
!pip install ninja
sys.path.insert(0, "/content/stylegan2-ada-pytorch")
```

```
fatal: destination path 'stylegan2-ada-pytorch' already exists and is not an
empty directory.
Requirement already satisfied: ninja in /usr/local/lib/python3.7/dist-packages
(1.10.2.3)
```

4.3 Detecting Facial Features

First, I will demonstrate how to detect the facial features we will use for consistent cropping and centering of the images. To accomplish this, we will use the dlib package, a neural network library that gives us access to several pretrained models. The DLIB Face Recognition ResNET Model V1 is the model we will use; This is a 5-point landmarking model which identifies the corners of the eyes and bottom of the nose. The creators of this network trained it on the dlib 5-point face landmark dataset, which consists of 7198 faces.

We begin by initializing dlib and loading the facial features neural network.

[ ]:
```
import cv2
import numpy as np
from PIL import Image
import dlib
from matplotlib import pyplot as plt

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('shape_predictor_5_face_landmarks.dat')
```

Let's start by looking at the facial features of the source image. The following code detects the five facial features and displays their coordinates.

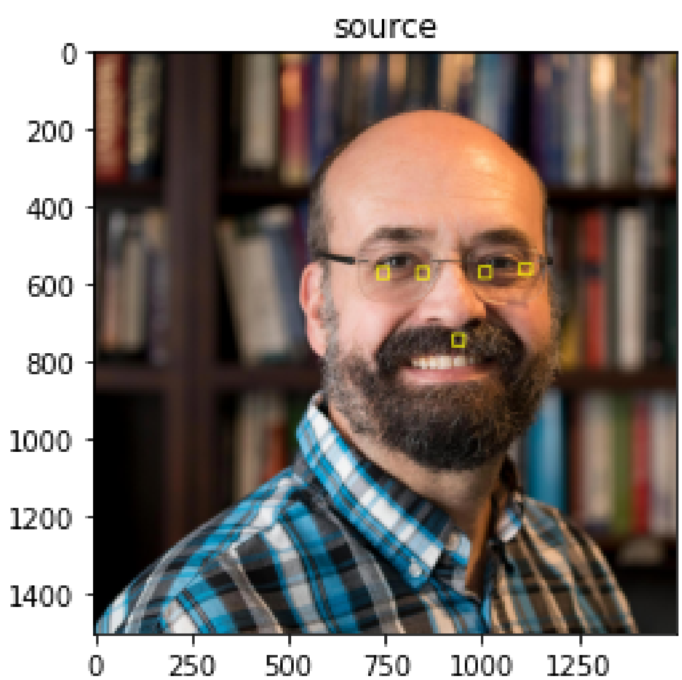
[ ]:
```
img = cv2.imread(SOURCE_NAME)
if img is None:
    raise ValueError("Source image not found")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
rects = detector(gray, 0)

if len(rects) == 0:
  raise ValueError("No faces detected")
elif len(rects) > 1:
  raise ValueError("Multiple faces detected")

shape = predictor(gray, rects[0])

w = img.shape[0]//50

for i in range(0, 5):
  pt1 = (shape.part(i).x, shape.part(i).y)
```

```
    pt2 = (shape.part(i).x+w, shape.part(i).y+w)
    cv2.rectangle(img,pt1,pt2,(0,255,255),4)
    print(pt1,pt2)
```

```
(1098, 546) (1128, 576)
(994, 554) (1024, 584)
(731, 556) (761, 586)
(833, 556) (863, 586)
(925, 729) (955, 759)
```

We can easily plot these features onto the source image. You can see the corners of the eyes and the base of the nose.

```
[ ]: img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
     plt.imshow(img)
     plt.title('source')
     plt.show()
```



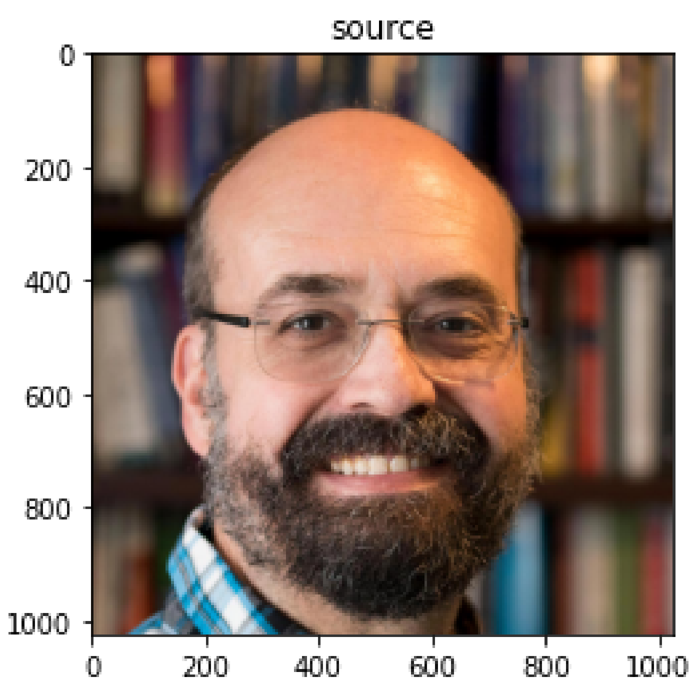
4.4 Preprocess Images for Best StyleGAN Results

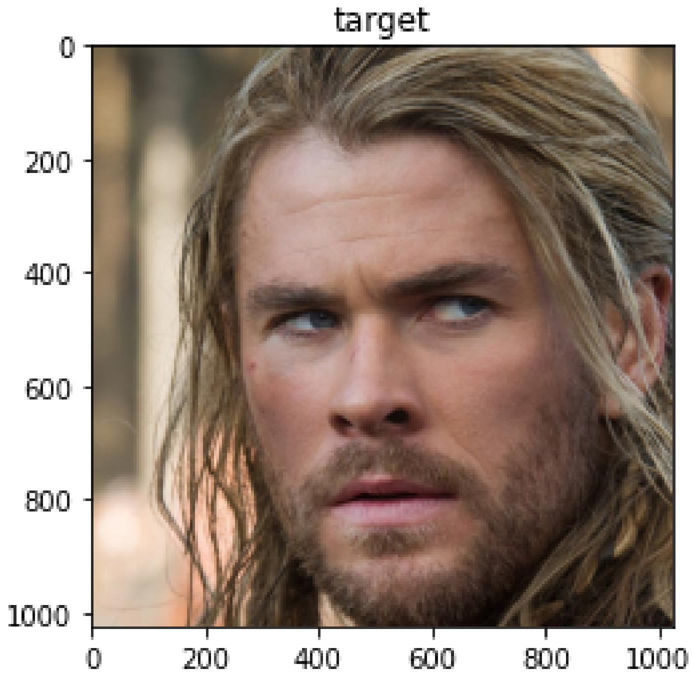
Using dlib, we will center and crop the source and target image, using the eye positions as reference. I created two functions to accomplish this task. The first calls dlib and find the locations of the person's eyes. The second uses the eye locations to center the image around the eyes. We do not exactly center; we are offsetting slightly to center, similar to the original StyleGAN training set. I determined this offset by detecting the eyes of a generated StyleGAN face. The distance between the eyes gives us a means of telling how big the face is, which we use to scale the images consistently.

[ ]:
```python
def find_eyes(img):
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  rects = detector(gray, 0)

  if len(rects) == 0:
    raise ValueError("No faces detected")
  elif len(rects) > 1:
    raise ValueError("Multiple faces detected")

  shape = predictor(gray, rects[0])
  features = []

  for i in range(0, 5):
    features.append((i, (shape.part(i).x, shape.part(i).y)))

  return (int(features[3][1][0] + features[2][1][0]) // 2, \
    int(features[3][1][1] + features[2][1][1]) // 2), \
    (int(features[1][1][0] + features[0][1][0]) // 2, \
    int(features[1][1][1] + features[0][1][1]) // 2)

def crop_stylegan(img):
  left_eye, right_eye = find_eyes(img)
  # Calculate the size of the face
  d = abs(right_eye[0] - left_eye[0])
  z = 255/d
  # Consider the aspect ratio
  ar = img.shape[0]/img.shape[1]
  w = img.shape[1] * z
  img2 = cv2.resize(img, (int(w), int(w*ar)))
  bordersize = 1024
  img3 = cv2.copyMakeBorder(
      img2,
      top=bordersize,
      bottom=bordersize,
      left=bordersize,
      right=bordersize,
      borderType=cv2.BORDER_REPLICATE)

  left_eye2, right_eye2 = find_eyes(img3)

  # Adjust to the offset used by StyleGAN2
  crop1 = left_eye2[0] - 385
  crop0 = left_eye2[1] - 490
  return img3[crop0:crop0+1024,crop1:crop1+1024]
```

The following code will preprocess and crop your images. If you receive an error indicating multiple faces were found, try to crop your image better or obscure the background. If the program does not see a face, then attempt to obtain a clearer and more high-resolution image.

[ ]:
```python
image_source = cv2.imread(SOURCE_NAME)
if image_source is None:
    raise ValueError("Source image not found")

image_target = cv2.imread(TARGET_NAME)
if image_target is None:
    raise ValueError("Source image not found")

cropped_source = crop_stylegan(image_source)
cropped_target = crop_stylegan(image_target)

img = cv2.cvtColor(cropped_source, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('source')
plt.show()

img = cv2.cvtColor(cropped_target, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('target')
plt.show()

cv2.imwrite("cropped_source.png", cropped_source)
cv2.imwrite("cropped_target.png", cropped_target)

#print(find_eyes(cropped_source))
#print(find_eyes(cropped_target))
```





[ ]: True

The two images are now 1024x1024 and cropped similarly to the ffhq dataset that NVIDIA used to train StyleGAN.

4.5 Convert Source to a GAN

We will use StyleGAN2, rather than the latest StyleGAN3, because StyleGAN2 contains a projector.py utility that converts images to latent vectors. StyleGAN3 does not have as good support for this projection. First, we convert the source to a GAN latent vector. This process will take several minutes.

[ ]:
```
# HIDE OUTPUT
cmd = f"python /content/stylegan2-ada-pytorch/projector.py "\
  f"--save-video 0 --num-steps 1000 --outdir=out_source "\
  f"--target=cropped_source.png --network={NETWORK}"
!{cmd}
```

```
Loading networks from "https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/ffhq.pkl"…
Downloading https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/ffhq.pkl … done
Computing W midpoint and stddev using 10000 samples…
Setting up PyTorch plugin "bias_act_plugin"… Done.
```

```
Downloading https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/metrics/vgg16.pt … done
Setting up PyTorch plugin "upfirdn2d_plugin"… Done.
step    1/1000: dist 0.69 loss 24568.84
step    2/1000: dist 0.68 loss 27642.19
step    3/1000: dist 0.69 loss 27167.21
step    4/1000: dist 0.64 loss 26253.41
step    5/1000: dist 0.68 loss 24959.88
step    6/1000: dist 0.67 loss 23356.19
step    7/1000: dist 0.68 loss 21512.25
step    8/1000: dist 0.64 loss 19487.28
step    9/1000: dist 0.65 loss 17341.38
step   10/1000: dist 0.64 loss 15140.43
step   11/1000: dist 0.69 loss 12949.55
step   12/1000: dist 0.66 loss 10820.28
step   13/1000: dist 0.67 loss 8802.95
step   14/1000: dist 0.70 loss 6946.31
step   15/1000: dist 0.68 loss 5316.80
step   16/1000: dist 0.59 loss 3971.21
step   17/1000: dist 0.59 loss 2941.14
step   18/1000: dist 0.66 loss 2216.37
step   19/1000: dist 0.63 loss 1758.90
step   20/1000: dist 0.61 loss 1567.61
step   21/1000: dist 0.58 loss 1602.35
step   22/1000: dist 0.56 loss 1787.89
step   23/1000: dist 0.57 loss 2053.43
step   24/1000: dist 0.57 loss 2327.65
step   25/1000: dist 0.55 loss 2538.83
step   26/1000: dist 0.57 loss 2637.41
step   27/1000: dist 0.56 loss 2603.98
step   28/1000: dist 0.55 loss 2477.22
step   29/1000: dist 0.56 loss 2317.67
step   30/1000: dist 0.56 loss 2120.18
step   31/1000: dist 0.55 loss 1884.70
step   32/1000: dist 0.57 loss 1628.14
step   33/1000: dist 0.57 loss 1388.85
step   34/1000: dist 0.55 loss 1184.02
step   35/1000: dist 0.56 loss 1026.62
step   36/1000: dist 0.54 loss 909.77
step   37/1000: dist 0.55 loss 830.95
step   38/1000: dist 0.55 loss 805.37
step   39/1000: dist 0.53 loss 812.69
step   40/1000: dist 0.53 loss 834.78
step   41/1000: dist 0.52 loss 828.74
step   42/1000: dist 0.53 loss 761.30
step   43/1000: dist 0.56 loss 651.21
step   44/1000: dist 0.52 loss 521.51
step   45/1000: dist 0.50 loss 402.90
```

```
step   46/1000: dist 0.50 loss 318.31
step   47/1000: dist 0.55 loss 263.56
step   48/1000: dist 0.53 loss 241.12
step   49/1000: dist 0.49 loss 251.14
step   50/1000: dist 0.51 loss 304.16
step   51/1000: dist 0.50 loss 369.90
step   52/1000: dist 0.52 loss 402.65
step   53/1000: dist 0.53 loss 403.27
step   54/1000: dist 0.54 loss 373.34
step   55/1000: dist 0.48 loss 286.43
step   56/1000: dist 0.49 loss 204.31
step   57/1000: dist 0.48 loss 142.15
step   58/1000: dist 0.49 loss 72.53
step   59/1000: dist 0.49 loss 64.59
step   60/1000: dist 0.49 loss 60.36
step   61/1000: dist 0.48 loss 57.51
step   62/1000: dist 0.48 loss 96.37
step   63/1000: dist 0.49 loss 109.48
step   64/1000: dist 0.47 loss 129.66
step   65/1000: dist 0.49 loss 132.66
step   66/1000: dist 0.49 loss 117.68
step   67/1000: dist 0.48 loss 103.61
step   68/1000: dist 0.49 loss 69.93
step   69/1000: dist 0.47 loss 52.00
step   70/1000: dist 0.48 loss 27.73
step   71/1000: dist 0.47 loss 19.48
step   72/1000: dist 0.47 loss 19.65
step   73/1000: dist 0.47 loss 20.25
step   74/1000: dist 0.47 loss 29.61
step   75/1000: dist 0.47 loss 33.64
step   76/1000: dist 0.46 loss 39.67
step   77/1000: dist 0.47 loss 36.08
step   78/1000: dist 0.46 loss 32.92
step   79/1000: dist 0.46 loss 31.11
step   80/1000: dist 0.47 loss 26.28
step   81/1000: dist 0.48 loss 22.76
step   82/1000: dist 0.48 loss 13.80
step   83/1000: dist 0.46 loss 12.49
step   84/1000: dist 0.48 loss 16.18
step   85/1000: dist 0.45 loss 15.87
step   86/1000: dist 0.44 loss 12.29
step   87/1000: dist 0.46 loss 10.47
step   88/1000: dist 0.48 loss 12.08
step   89/1000: dist 0.46 loss 10.22
step   90/1000: dist 0.45 loss 7.48
step   91/1000: dist 0.49 loss 6.97
step   92/1000: dist 0.45 loss 6.56
step   93/1000: dist 0.48 loss 7.02
```

```
step   94/1000: dist 0.46 loss 8.58
step   95/1000: dist 0.45 loss 10.76
step   96/1000: dist 0.47 loss 12.40
step   97/1000: dist 0.48 loss 11.84
step   98/1000: dist 0.45 loss 9.08
step   99/1000: dist 0.45 loss 6.71
step  100/1000: dist 0.45 loss 7.39
step  101/1000: dist 0.43 loss 9.75
step  102/1000: dist 0.43 loss 9.56
step  103/1000: dist 0.43 loss 5.58
step  104/1000: dist 0.44 loss 2.44
step  105/1000: dist 0.44 loss 4.01
step  106/1000: dist 0.45 loss 7.38
step  107/1000: dist 0.43 loss 8.32
step  108/1000: dist 0.43 loss 6.85
step  109/1000: dist 0.42 loss 5.39
step  110/1000: dist 0.44 loss 5.18
step  111/1000: dist 0.45 loss 5.21
step  112/1000: dist 0.43 loss 4.84
step  113/1000: dist 0.44 loss 4.69
step  114/1000: dist 0.43 loss 5.06
step  115/1000: dist 0.43 loss 5.76
step  116/1000: dist 0.42 loss 6.62
step  117/1000: dist 0.43 loss 7.34
step  118/1000: dist 0.44 loss 7.84
step  119/1000: dist 0.42 loss 9.06
step  120/1000: dist 0.44 loss 11.57
step  121/1000: dist 0.43 loss 12.81
step  122/1000: dist 0.42 loss 8.81
step  123/1000: dist 0.43 loss 2.64
step  124/1000: dist 0.41 loss 2.90
step  125/1000: dist 0.42 loss 7.96
step  126/1000: dist 0.41 loss 8.38
step  127/1000: dist 0.43 loss 3.88
step  128/1000: dist 0.42 loss 2.75
step  129/1000: dist 0.41 loss 4.95
step  130/1000: dist 0.42 loss 4.01
step  131/1000: dist 0.42 loss 1.47
step  132/1000: dist 0.43 loss 2.52
step  133/1000: dist 0.41 loss 4.18
step  134/1000: dist 0.40 loss 3.20
step  135/1000: dist 0.42 loss 3.41
step  136/1000: dist 0.41 loss 6.76
step  137/1000: dist 0.44 loss 10.63
step  138/1000: dist 0.42 loss 14.50
step  139/1000: dist 0.41 loss 15.30
step  140/1000: dist 0.42 loss 8.50
step  141/1000: dist 0.42 loss 3.20
```

```
step  142/1000: dist 0.41 loss 8.34
step  143/1000: dist 0.40 loss 13.73
step  144/1000: dist 0.40 loss 11.53
step  145/1000: dist 0.41 loss 13.75
step  146/1000: dist 0.39 loss 23.21
step  147/1000: dist 0.42 loss 23.69
step  148/1000: dist 0.41 loss 13.22
step  149/1000: dist 0.40 loss 7.29
step  150/1000: dist 0.42 loss 8.80
step  151/1000: dist 0.42 loss 10.39
step  152/1000: dist 0.41 loss 10.74
step  153/1000: dist 0.39 loss 10.47
step  154/1000: dist 0.40 loss 12.42
step  155/1000: dist 0.40 loss 20.22
step  156/1000: dist 0.40 loss 27.15
step  157/1000: dist 0.39 loss 24.20
step  158/1000: dist 0.42 loss 17.26
step  159/1000: dist 0.40 loss 14.65
step  160/1000: dist 0.40 loss 13.90
step  161/1000: dist 0.40 loss 13.30
step  162/1000: dist 0.39 loss 15.58
step  163/1000: dist 0.41 loss 20.83
step  164/1000: dist 0.41 loss 22.71
step  165/1000: dist 0.43 loss 19.80
step  166/1000: dist 0.40 loss 17.83
step  167/1000: dist 0.39 loss 17.91
step  168/1000: dist 0.40 loss 14.52
step  169/1000: dist 0.40 loss 10.05
step  170/1000: dist 0.39 loss 8.76
step  171/1000: dist 0.38 loss 7.54
step  172/1000: dist 0.40 loss 7.03
step  173/1000: dist 0.38 loss 9.81
step  174/1000: dist 0.40 loss 9.86
step  175/1000: dist 0.41 loss 4.71
step  176/1000: dist 0.39 loss 2.85
step  177/1000: dist 0.39 loss 5.94
step  178/1000: dist 0.41 loss 6.41
step  179/1000: dist 0.39 loss 3.66
step  180/1000: dist 0.39 loss 2.83
step  181/1000: dist 0.40 loss 3.92
step  182/1000: dist 0.41 loss 4.22
step  183/1000: dist 0.38 loss 3.22
step  184/1000: dist 0.39 loss 2.18
step  185/1000: dist 0.39 loss 2.63
step  186/1000: dist 0.38 loss 4.38
step  187/1000: dist 0.38 loss 5.70
step  188/1000: dist 0.39 loss 7.27
step  189/1000: dist 0.40 loss 11.58
```

```
step  190/1000: dist 0.38 loss 18.20
step  191/1000: dist 0.38 loss 22.91
step  192/1000: dist 0.38 loss 19.88
step  193/1000: dist 0.40 loss 10.03
step  194/1000: dist 0.38 loss 7.92
step  195/1000: dist 0.39 loss 16.66
step  196/1000: dist 0.39 loss 19.63
step  197/1000: dist 0.38 loss 12.89
step  198/1000: dist 0.38 loss 13.33
step  199/1000: dist 0.38 loss 20.81
step  200/1000: dist 0.38 loss 17.69
step  201/1000: dist 0.39 loss 7.83
step  202/1000: dist 0.39 loss 6.01
step  203/1000: dist 0.36 loss 7.21
step  204/1000: dist 0.39 loss 6.24
step  205/1000: dist 0.38 loss 8.44
step  206/1000: dist 0.38 loss 9.41
step  207/1000: dist 0.38 loss 4.84
step  208/1000: dist 0.37 loss 3.23
step  209/1000: dist 0.38 loss 6.27
step  210/1000: dist 0.37 loss 8.15
step  211/1000: dist 0.37 loss 10.04
step  212/1000: dist 0.38 loss 11.97
step  213/1000: dist 0.38 loss 11.18
step  214/1000: dist 0.37 loss 9.16
step  215/1000: dist 0.38 loss 6.61
step  216/1000: dist 0.37 loss 3.30
step  217/1000: dist 0.37 loss 3.33
step  218/1000: dist 0.37 loss 6.29
step  219/1000: dist 0.37 loss 6.12
step  220/1000: dist 0.36 loss 3.18
step  221/1000: dist 0.37 loss 2.60
step  222/1000: dist 0.37 loss 4.20
step  223/1000: dist 0.37 loss 5.04
step  224/1000: dist 0.37 loss 4.37
step  225/1000: dist 0.37 loss 3.59
step  226/1000: dist 0.37 loss 4.55
step  227/1000: dist 0.37 loss 6.34
step  228/1000: dist 0.37 loss 5.99
step  229/1000: dist 0.37 loss 3.80
step  230/1000: dist 0.37 loss 2.65
step  231/1000: dist 0.36 loss 2.51
step  232/1000: dist 0.37 loss 1.80
step  233/1000: dist 0.37 loss 1.14
step  234/1000: dist 0.37 loss 1.69
step  235/1000: dist 0.37 loss 2.71
step  236/1000: dist 0.37 loss 2.88
step  237/1000: dist 0.36 loss 2.07
```

```
step  238/1000: dist 0.37 loss 1.26
step  239/1000: dist 0.36 loss 1.30
step  240/1000: dist 0.37 loss 2.15
step  241/1000: dist 0.36 loss 3.45
step  242/1000: dist 0.36 loss 5.85
step  243/1000: dist 0.36 loss 10.44
step  244/1000: dist 0.36 loss 16.26
step  245/1000: dist 0.37 loss 18.68
step  246/1000: dist 0.37 loss 11.90
step  247/1000: dist 0.37 loss 2.40
step  248/1000: dist 0.36 loss 2.70
step  249/1000: dist 0.36 loss 10.08
step  250/1000: dist 0.38 loss 11.57
step  251/1000: dist 0.36 loss 8.12
step  252/1000: dist 0.35 loss 13.61
step  253/1000: dist 0.35 loss 25.04
step  254/1000: dist 0.36 loss 24.02
step  255/1000: dist 0.36 loss 9.58
step  256/1000: dist 0.36 loss 2.80
step  257/1000: dist 0.36 loss 9.33
step  258/1000: dist 0.36 loss 12.41
step  259/1000: dist 0.35 loss 6.26
step  260/1000: dist 0.37 loss 3.37
step  261/1000: dist 0.35 loss 7.18
step  262/1000: dist 0.36 loss 6.98
step  263/1000: dist 0.36 loss 3.34
step  264/1000: dist 0.36 loss 5.19
step  265/1000: dist 0.36 loss 8.05
step  266/1000: dist 0.36 loss 7.96
step  267/1000: dist 0.36 loss 12.16
step  268/1000: dist 0.36 loss 18.35
step  269/1000: dist 0.37 loss 17.42
step  270/1000: dist 0.37 loss 10.68
step  271/1000: dist 0.36 loss 4.65
step  272/1000: dist 0.36 loss 3.72
step  273/1000: dist 0.37 loss 9.11
step  274/1000: dist 0.36 loss 11.94
step  275/1000: dist 0.36 loss 7.31
step  276/1000: dist 0.36 loss 7.62
step  277/1000: dist 0.35 loss 16.34
step  278/1000: dist 0.35 loss 22.40
step  279/1000: dist 0.35 loss 25.18
step  280/1000: dist 0.35 loss 28.86
step  281/1000: dist 0.36 loss 28.18
step  282/1000: dist 0.37 loss 24.31
step  283/1000: dist 0.36 loss 24.22
step  284/1000: dist 0.35 loss 24.12
step  285/1000: dist 0.35 loss 21.50
```

```
step  286/1000: dist 0.35 loss 22.67
step  287/1000: dist 0.36 loss 28.68
step  288/1000: dist 0.34 loss 29.07
step  289/1000: dist 0.37 loss 19.28
step  290/1000: dist 0.36 loss 13.45
step  291/1000: dist 0.35 loss 19.68
step  292/1000: dist 0.35 loss 26.27
step  293/1000: dist 0.35 loss 25.12
step  294/1000: dist 0.35 loss 21.81
step  295/1000: dist 0.36 loss 22.30
step  296/1000: dist 0.35 loss 26.51
step  297/1000: dist 0.36 loss 29.40
step  298/1000: dist 0.35 loss 25.66
step  299/1000: dist 0.34 loss 20.94
step  300/1000: dist 0.34 loss 22.96
step  301/1000: dist 0.34 loss 24.71
step  302/1000: dist 0.35 loss 20.26
step  303/1000: dist 0.36 loss 14.66
step  304/1000: dist 0.35 loss 10.48
step  305/1000: dist 0.34 loss 8.68
step  306/1000: dist 0.34 loss 12.03
step  307/1000: dist 0.34 loss 14.06
step  308/1000: dist 0.35 loss 7.56
step  309/1000: dist 0.35 loss 2.77
step  310/1000: dist 0.35 loss 7.47
step  311/1000: dist 0.35 loss 10.36
step  312/1000: dist 0.34 loss 5.24
step  313/1000: dist 0.35 loss 2.05
step  314/1000: dist 0.34 loss 5.11
step  315/1000: dist 0.35 loss 7.07
step  316/1000: dist 0.35 loss 5.08
step  317/1000: dist 0.34 loss 4.33
step  318/1000: dist 0.35 loss 7.67
step  319/1000: dist 0.34 loss 12.37
step  320/1000: dist 0.34 loss 16.78
step  321/1000: dist 0.35 loss 20.85
step  322/1000: dist 0.34 loss 21.44
step  323/1000: dist 0.33 loss 16.52
step  324/1000: dist 0.33 loss 14.56
step  325/1000: dist 0.34 loss 18.90
step  326/1000: dist 0.34 loss 18.37
step  327/1000: dist 0.35 loss 8.28
step  328/1000: dist 0.34 loss 2.29
step  329/1000: dist 0.35 loss 8.59
step  330/1000: dist 0.34 loss 13.68
step  331/1000: dist 0.34 loss 7.50
step  332/1000: dist 0.35 loss 1.79
step  333/1000: dist 0.35 loss 5.54
```

```
step  334/1000: dist 0.34 loss 8.76
step  335/1000: dist 0.35 loss 5.35
step  336/1000: dist 0.34 loss 3.83
step  337/1000: dist 0.33 loss 7.30
step  338/1000: dist 0.34 loss 9.16
step  339/1000: dist 0.34 loss 9.84
step  340/1000: dist 0.35 loss 14.98
step  341/1000: dist 0.34 loss 21.81
step  342/1000: dist 0.34 loss 25.12
step  343/1000: dist 0.34 loss 24.57
step  344/1000: dist 0.34 loss 20.08
step  345/1000: dist 0.34 loss 13.54
step  346/1000: dist 0.35 loss 13.06
step  347/1000: dist 0.34 loss 19.40
step  348/1000: dist 0.33 loss 20.69
step  349/1000: dist 0.33 loss 10.90
step  350/1000: dist 0.33 loss 2.13
step  351/1000: dist 0.35 loss 5.08
step  352/1000: dist 0.34 loss 12.02
step  353/1000: dist 0.34 loss 11.44
step  354/1000: dist 0.34 loss 6.04
step  355/1000: dist 0.34 loss 5.80
step  356/1000: dist 0.34 loss 11.18
step  357/1000: dist 0.34 loss 16.00
step  358/1000: dist 0.34 loss 18.66
step  359/1000: dist 0.34 loss 18.86
step  360/1000: dist 0.33 loss 13.73
step  361/1000: dist 0.33 loss 5.32
step  362/1000: dist 0.34 loss 3.06
step  363/1000: dist 0.33 loss 7.89
step  364/1000: dist 0.33 loss 10.21
step  365/1000: dist 0.33 loss 5.65
step  366/1000: dist 0.32 loss 1.92
step  367/1000: dist 0.34 loss 4.22
step  368/1000: dist 0.33 loss 6.29
step  369/1000: dist 0.33 loss 3.76
step  370/1000: dist 0.33 loss 1.72
step  371/1000: dist 0.33 loss 3.26
step  372/1000: dist 0.33 loss 3.86
step  373/1000: dist 0.34 loss 1.95
step  374/1000: dist 0.33 loss 1.59
step  375/1000: dist 0.33 loss 2.97
step  376/1000: dist 0.32 loss 2.49
step  377/1000: dist 0.33 loss 1.11
step  378/1000: dist 0.32 loss 1.87
step  379/1000: dist 0.34 loss 3.21
step  380/1000: dist 0.33 loss 3.13
step  381/1000: dist 0.33 loss 4.06
```

```
step  382/1000: dist 0.32 loss 7.84
step  383/1000: dist 0.32 loss 13.22
step  384/1000: dist 0.32 loss 19.29
step  385/1000: dist 0.32 loss 23.98
step  386/1000: dist 0.33 loss 21.28
step  387/1000: dist 0.33 loss 13.27
step  388/1000: dist 0.33 loss 12.23
step  389/1000: dist 0.32 loss 21.08
step  390/1000: dist 0.32 loss 27.59
step  391/1000: dist 0.32 loss 24.08
step  392/1000: dist 0.32 loss 17.60
step  393/1000: dist 0.33 loss 14.27
step  394/1000: dist 0.33 loss 11.27
step  395/1000: dist 0.33 loss 8.59
step  396/1000: dist 0.32 loss 9.70
step  397/1000: dist 0.33 loss 11.47
step  398/1000: dist 0.32 loss 7.58
step  399/1000: dist 0.32 loss 3.13
step  400/1000: dist 0.32 loss 5.77
step  401/1000: dist 0.32 loss 9.09
step  402/1000: dist 0.32 loss 5.23
step  403/1000: dist 0.33 loss 1.28
step  404/1000: dist 0.32 loss 3.82
step  405/1000: dist 0.32 loss 5.91
step  406/1000: dist 0.32 loss 3.10
step  407/1000: dist 0.32 loss 1.62
step  408/1000: dist 0.32 loss 3.26
step  409/1000: dist 0.32 loss 3.00
step  410/1000: dist 0.32 loss 1.46
step  411/1000: dist 0.32 loss 2.15
step  412/1000: dist 0.32 loss 2.80
step  413/1000: dist 0.32 loss 1.41
step  414/1000: dist 0.32 loss 0.93
step  415/1000: dist 0.31 loss 2.13
step  416/1000: dist 0.32 loss 2.36
step  417/1000: dist 0.33 loss 1.89
step  418/1000: dist 0.32 loss 2.89
step  419/1000: dist 0.32 loss 5.33
step  420/1000: dist 0.32 loss 9.25
step  421/1000: dist 0.33 loss 16.09
step  422/1000: dist 0.32 loss 24.57
step  423/1000: dist 0.32 loss 26.95
step  424/1000: dist 0.31 loss 19.15
step  425/1000: dist 0.32 loss 11.93
step  426/1000: dist 0.32 loss 17.14
step  427/1000: dist 0.31 loss 23.52
step  428/1000: dist 0.32 loss 15.46
step  429/1000: dist 0.31 loss 3.44
```

```
step  430/1000: dist 0.32 loss 6.88
step  431/1000: dist 0.31 loss 17.01
step  432/1000: dist 0.31 loss 16.33
step  433/1000: dist 0.31 loss 11.61
step  434/1000: dist 0.31 loss 13.26
step  435/1000: dist 0.31 loss 13.25
step  436/1000: dist 0.31 loss 7.99
step  437/1000: dist 0.31 loss 5.13
step  438/1000: dist 0.31 loss 4.27
step  439/1000: dist 0.31 loss 3.72
step  440/1000: dist 0.31 loss 6.71
step  441/1000: dist 0.31 loss 7.85
step  442/1000: dist 0.32 loss 3.05
step  443/1000: dist 0.31 loss 1.01
step  444/1000: dist 0.31 loss 3.84
step  445/1000: dist 0.31 loss 4.11
step  446/1000: dist 0.31 loss 2.56
step  447/1000: dist 0.32 loss 2.66
step  448/1000: dist 0.31 loss 2.22
step  449/1000: dist 0.31 loss 1.51
step  450/1000: dist 0.31 loss 2.12
step  451/1000: dist 0.31 loss 2.16
step  452/1000: dist 0.31 loss 1.58
step  453/1000: dist 0.31 loss 1.72
step  454/1000: dist 0.31 loss 1.64
step  455/1000: dist 0.31 loss 1.62
step  456/1000: dist 0.31 loss 2.54
step  457/1000: dist 0.31 loss 3.23
step  458/1000: dist 0.31 loss 4.01
step  459/1000: dist 0.31 loss 6.42
step  460/1000: dist 0.31 loss 9.23
step  461/1000: dist 0.31 loss 10.20
step  462/1000: dist 0.31 loss 8.64
step  463/1000: dist 0.31 loss 4.75
step  464/1000: dist 0.31 loss 1.29
step  465/1000: dist 0.31 loss 1.23
step  466/1000: dist 0.31 loss 3.58
step  467/1000: dist 0.30 loss 4.88
step  468/1000: dist 0.31 loss 3.58
step  469/1000: dist 0.31 loss 1.32
step  470/1000: dist 0.31 loss 0.72
step  471/1000: dist 0.31 loss 2.10
step  472/1000: dist 0.31 loss 3.09
step  473/1000: dist 0.31 loss 2.07
step  474/1000: dist 0.31 loss 0.66
step  475/1000: dist 0.31 loss 0.85
step  476/1000: dist 0.31 loss 1.94
step  477/1000: dist 0.30 loss 2.23
```

```
step  478/1000: dist 0.30 loss 1.61
step  479/1000: dist 0.31 loss 1.47
step  480/1000: dist 0.31 loss 2.64
step  481/1000: dist 0.31 loss 4.61
step  482/1000: dist 0.31 loss 6.94
step  483/1000: dist 0.31 loss 10.78
step  484/1000: dist 0.31 loss 18.04
step  485/1000: dist 0.31 loss 29.05
step  486/1000: dist 0.31 loss 40.36
step  487/1000: dist 0.30 loss 43.49
step  488/1000: dist 0.30 loss 35.09
step  489/1000: dist 0.30 loss 29.96
step  490/1000: dist 0.31 loss 43.41
step  491/1000: dist 0.31 loss 56.04
step  492/1000: dist 0.30 loss 38.46
step  493/1000: dist 0.30 loss 8.22
step  494/1000: dist 0.30 loss 9.72
step  495/1000: dist 0.30 loss 29.15
step  496/1000: dist 0.30 loss 23.73
step  497/1000: dist 0.30 loss 7.04
step  498/1000: dist 0.30 loss 13.43
step  499/1000: dist 0.30 loss 23.59
step  500/1000: dist 0.30 loss 17.84
step  501/1000: dist 0.30 loss 19.33
step  502/1000: dist 0.30 loss 28.43
step  503/1000: dist 0.30 loss 21.12
step  504/1000: dist 0.30 loss 9.01
step  505/1000: dist 0.30 loss 8.26
step  506/1000: dist 0.30 loss 9.52
step  507/1000: dist 0.30 loss 13.00
step  508/1000: dist 0.30 loss 17.64
step  509/1000: dist 0.31 loss 14.73
step  510/1000: dist 0.30 loss 19.13
step  511/1000: dist 0.30 loss 33.71
step  512/1000: dist 0.30 loss 31.12
step  513/1000: dist 0.31 loss 13.14
step  514/1000: dist 0.30 loss 6.40
step  515/1000: dist 0.30 loss 14.30
step  516/1000: dist 0.30 loss 24.06
step  517/1000: dist 0.30 loss 24.87
step  518/1000: dist 0.30 loss 23.05
step  519/1000: dist 0.30 loss 30.82
step  520/1000: dist 0.30 loss 35.43
step  521/1000: dist 0.30 loss 30.63
step  522/1000: dist 0.30 loss 34.11
step  523/1000: dist 0.30 loss 39.74
step  524/1000: dist 0.30 loss 28.25
step  525/1000: dist 0.30 loss 11.24
```

```
step  526/1000: dist 0.30 loss 9.98
step  527/1000: dist 0.30 loss 17.81
step  528/1000: dist 0.30 loss 18.88
step  529/1000: dist 0.30 loss 12.85
step  530/1000: dist 0.30 loss 9.04
step  531/1000: dist 0.29 loss 13.49
step  532/1000: dist 0.30 loss 22.25
step  533/1000: dist 0.29 loss 28.80
step  534/1000: dist 0.30 loss 33.04
step  535/1000: dist 0.29 loss 32.25
step  536/1000: dist 0.30 loss 19.49
step  537/1000: dist 0.29 loss 6.74
step  538/1000: dist 0.29 loss 11.76
step  539/1000: dist 0.29 loss 22.10
step  540/1000: dist 0.29 loss 17.26
step  541/1000: dist 0.29 loss 13.31
step  542/1000: dist 0.29 loss 26.46
step  543/1000: dist 0.29 loss 37.41
step  544/1000: dist 0.29 loss 37.61
step  545/1000: dist 0.30 loss 35.10
step  546/1000: dist 0.29 loss 20.96
step  547/1000: dist 0.29 loss 4.67
step  548/1000: dist 0.29 loss 9.04
step  549/1000: dist 0.29 loss 18.12
step  550/1000: dist 0.29 loss 13.05
step  551/1000: dist 0.29 loss 7.97
step  552/1000: dist 0.29 loss 8.73
step  553/1000: dist 0.29 loss 7.92
step  554/1000: dist 0.29 loss 6.70
step  555/1000: dist 0.29 loss 5.97
step  556/1000: dist 0.29 loss 6.93
step  557/1000: dist 0.29 loss 7.92
step  558/1000: dist 0.29 loss 5.14
step  559/1000: dist 0.29 loss 6.64
step  560/1000: dist 0.29 loss 13.24
step  561/1000: dist 0.29 loss 15.03
step  562/1000: dist 0.29 loss 16.70
step  563/1000: dist 0.29 loss 20.20
step  564/1000: dist 0.29 loss 16.17
step  565/1000: dist 0.29 loss 11.33
step  566/1000: dist 0.29 loss 17.64
step  567/1000: dist 0.29 loss 32.97
step  568/1000: dist 0.29 loss 45.17
step  569/1000: dist 0.29 loss 35.84
step  570/1000: dist 0.29 loss 8.59
step  571/1000: dist 0.29 loss 3.59
step  572/1000: dist 0.29 loss 20.92
step  573/1000: dist 0.29 loss 20.77
```

```
step  574/1000: dist 0.29 loss 3.85
step  575/1000: dist 0.29 loss 5.55
step  576/1000: dist 0.29 loss 15.39
step  577/1000: dist 0.29 loss 7.31
step  578/1000: dist 0.29 loss 3.02
step  579/1000: dist 0.29 loss 10.18
step  580/1000: dist 0.29 loss 7.08
step  581/1000: dist 0.29 loss 5.36
step  582/1000: dist 0.29 loss 11.98
step  583/1000: dist 0.29 loss 11.95
step  584/1000: dist 0.29 loss 15.34
step  585/1000: dist 0.29 loss 22.79
step  586/1000: dist 0.29 loss 17.95
step  587/1000: dist 0.29 loss 10.92
step  588/1000: dist 0.29 loss 7.59
step  589/1000: dist 0.29 loss 6.25
step  590/1000: dist 0.29 loss 14.51
step  591/1000: dist 0.29 loss 23.03
step  592/1000: dist 0.29 loss 23.12
step  593/1000: dist 0.29 loss 22.27
step  594/1000: dist 0.29 loss 17.50
step  595/1000: dist 0.29 loss 8.61
step  596/1000: dist 0.29 loss 5.00
step  597/1000: dist 0.29 loss 7.29
step  598/1000: dist 0.29 loss 11.12
step  599/1000: dist 0.29 loss 8.93
step  600/1000: dist 0.29 loss 2.45
step  601/1000: dist 0.29 loss 3.66
step  602/1000: dist 0.29 loss 7.76
step  603/1000: dist 0.28 loss 4.72
step  604/1000: dist 0.28 loss 1.59
step  605/1000: dist 0.28 loss 3.54
step  606/1000: dist 0.29 loss 4.59
step  607/1000: dist 0.28 loss 2.44
step  608/1000: dist 0.28 loss 1.46
step  609/1000: dist 0.29 loss 2.96
step  610/1000: dist 0.28 loss 2.76
step  611/1000: dist 0.28 loss 1.17
step  612/1000: dist 0.28 loss 1.90
step  613/1000: dist 0.28 loss 2.54
step  614/1000: dist 0.28 loss 1.72
step  615/1000: dist 0.28 loss 2.48
step  616/1000: dist 0.28 loss 4.14
step  617/1000: dist 0.29 loss 5.72
step  618/1000: dist 0.28 loss 9.74
step  619/1000: dist 0.28 loss 16.44
step  620/1000: dist 0.29 loss 23.35
step  621/1000: dist 0.28 loss 26.36
```

```
step  622/1000: dist 0.28 loss 19.58
step  623/1000: dist 0.28 loss 6.19
step  624/1000: dist 0.28 loss 0.61
step  625/1000: dist 0.28 loss 7.34
step  626/1000: dist 0.28 loss 12.92
step  627/1000: dist 0.28 loss 7.42
step  628/1000: dist 0.28 loss 0.93
step  629/1000: dist 0.28 loss 4.01
step  630/1000: dist 0.28 loss 8.48
step  631/1000: dist 0.28 loss 5.70
step  632/1000: dist 0.28 loss 3.39
step  633/1000: dist 0.28 loss 8.37
step  634/1000: dist 0.28 loss 13.77
step  635/1000: dist 0.28 loss 15.47
step  636/1000: dist 0.28 loss 18.18
step  637/1000: dist 0.28 loss 18.48
step  638/1000: dist 0.28 loss 9.98
step  639/1000: dist 0.28 loss 1.43
step  640/1000: dist 0.28 loss 3.75
step  641/1000: dist 0.28 loss 10.17
step  642/1000: dist 0.28 loss 10.34
step  643/1000: dist 0.28 loss 6.58
step  644/1000: dist 0.28 loss 5.84
step  645/1000: dist 0.28 loss 9.31
step  646/1000: dist 0.28 loss 13.84
step  647/1000: dist 0.28 loss 15.09
step  648/1000: dist 0.28 loss 12.27
step  649/1000: dist 0.28 loss 9.42
step  650/1000: dist 0.28 loss 8.47
step  651/1000: dist 0.28 loss 7.07
step  652/1000: dist 0.28 loss 8.12
step  653/1000: dist 0.28 loss 16.30
step  654/1000: dist 0.28 loss 27.28
step  655/1000: dist 0.28 loss 33.61
step  656/1000: dist 0.28 loss 33.01
step  657/1000: dist 0.28 loss 26.69
step  658/1000: dist 0.28 loss 20.01
step  659/1000: dist 0.28 loss 18.61
step  660/1000: dist 0.28 loss 20.18
step  661/1000: dist 0.28 loss 18.51
step  662/1000: dist 0.28 loss 15.59
step  663/1000: dist 0.28 loss 16.57
step  664/1000: dist 0.28 loss 16.41
step  665/1000: dist 0.28 loss 9.32
step  666/1000: dist 0.28 loss 3.90
step  667/1000: dist 0.28 loss 8.53
step  668/1000: dist 0.28 loss 12.98
step  669/1000: dist 0.28 loss 7.80
```

```
step  670/1000: dist 0.28 loss 3.10
step  671/1000: dist 0.28 loss 6.64
step  672/1000: dist 0.28 loss 9.86
step  673/1000: dist 0.28 loss 8.57
step  674/1000: dist 0.28 loss 9.84
step  675/1000: dist 0.28 loss 14.81
step  676/1000: dist 0.28 loss 18.89
step  677/1000: dist 0.28 loss 20.58
step  678/1000: dist 0.28 loss 17.72
step  679/1000: dist 0.28 loss 8.93
step  680/1000: dist 0.28 loss 3.00
step  681/1000: dist 0.28 loss 4.18
step  682/1000: dist 0.27 loss 7.24
step  683/1000: dist 0.28 loss 8.75
step  684/1000: dist 0.27 loss 6.74
step  685/1000: dist 0.27 loss 2.29
step  686/1000: dist 0.28 loss 1.71
step  687/1000: dist 0.27 loss 5.19
step  688/1000: dist 0.27 loss 5.46
step  689/1000: dist 0.27 loss 2.11
step  690/1000: dist 0.27 loss 1.13
step  691/1000: dist 0.27 loss 2.81
step  692/1000: dist 0.28 loss 3.43
step  693/1000: dist 0.27 loss 2.14
step  694/1000: dist 0.28 loss 0.88
step  695/1000: dist 0.27 loss 1.39
step  696/1000: dist 0.27 loss 2.45
step  697/1000: dist 0.27 loss 1.76
step  698/1000: dist 0.27 loss 0.53
step  699/1000: dist 0.27 loss 0.99
step  700/1000: dist 0.27 loss 1.77
step  701/1000: dist 0.27 loss 1.13
step  702/1000: dist 0.27 loss 0.46
step  703/1000: dist 0.27 loss 0.87
step  704/1000: dist 0.27 loss 1.19
step  705/1000: dist 0.27 loss 0.77
step  706/1000: dist 0.27 loss 0.46
step  707/1000: dist 0.27 loss 0.65
step  708/1000: dist 0.27 loss 0.84
step  709/1000: dist 0.27 loss 0.67
step  710/1000: dist 0.27 loss 0.40
step  711/1000: dist 0.27 loss 0.47
step  712/1000: dist 0.27 loss 0.67
step  713/1000: dist 0.27 loss 0.56
step  714/1000: dist 0.27 loss 0.33
step  715/1000: dist 0.27 loss 0.40
step  716/1000: dist 0.27 loss 0.56
step  717/1000: dist 0.27 loss 0.45
```

```
step  718/1000: dist 0.27 loss 0.30
step  719/1000: dist 0.27 loss 0.38
step  720/1000: dist 0.27 loss 0.47
step  721/1000: dist 0.27 loss 0.40
step  722/1000: dist 0.27 loss 0.34
step  723/1000: dist 0.27 loss 0.40
step  724/1000: dist 0.27 loss 0.48
step  725/1000: dist 0.27 loss 0.53
step  726/1000: dist 0.27 loss 0.62
step  727/1000: dist 0.27 loss 0.89
step  728/1000: dist 0.27 loss 1.42
step  729/1000: dist 0.27 loss 2.33
step  730/1000: dist 0.27 loss 3.93
step  731/1000: dist 0.27 loss 6.68
step  732/1000: dist 0.27 loss 10.94
step  733/1000: dist 0.27 loss 15.81
step  734/1000: dist 0.27 loss 18.82
step  735/1000: dist 0.27 loss 15.64
step  736/1000: dist 0.27 loss 7.32
step  737/1000: dist 0.27 loss 1.40
step  738/1000: dist 0.27 loss 3.80
step  739/1000: dist 0.27 loss 10.37
step  740/1000: dist 0.27 loss 13.10
step  741/1000: dist 0.27 loss 12.50
step  742/1000: dist 0.27 loss 16.46
step  743/1000: dist 0.27 loss 24.98
step  744/1000: dist 0.27 loss 25.68
step  745/1000: dist 0.27 loss 12.94
step  746/1000: dist 0.27 loss 1.45
step  747/1000: dist 0.27 loss 4.53
step  748/1000: dist 0.27 loss 12.64
step  749/1000: dist 0.27 loss 11.11
step  750/1000: dist 0.27 loss 4.38
step  751/1000: dist 0.27 loss 6.23
step  752/1000: dist 0.27 loss 14.54
step  753/1000: dist 0.27 loss 19.38
step  754/1000: dist 0.27 loss 24.91
step  755/1000: dist 0.27 loss 38.32
step  756/1000: dist 0.27 loss 49.84
step  757/1000: dist 0.27 loss 45.00
step  758/1000: dist 0.27 loss 26.81
step  759/1000: dist 0.27 loss 10.55
step  760/1000: dist 0.27 loss 12.87
step  761/1000: dist 0.27 loss 24.01
step  762/1000: dist 0.27 loss 19.21
step  763/1000: dist 0.27 loss 4.68
step  764/1000: dist 0.27 loss 8.19
step  765/1000: dist 0.27 loss 18.31
```

```
step  766/1000: dist 0.27 loss 10.53
step  767/1000: dist 0.27 loss 4.86
step  768/1000: dist 0.27 loss 14.35
step  769/1000: dist 0.27 loss 16.69
step  770/1000: dist 0.27 loss 14.24
step  771/1000: dist 0.27 loss 18.36
step  772/1000: dist 0.27 loss 16.86
step  773/1000: dist 0.27 loss 10.16
step  774/1000: dist 0.27 loss 5.91
step  775/1000: dist 0.27 loss 2.40
step  776/1000: dist 0.27 loss 5.78
step  777/1000: dist 0.27 loss 9.85
step  778/1000: dist 0.27 loss 5.25
step  779/1000: dist 0.27 loss 2.71
step  780/1000: dist 0.27 loss 3.53
step  781/1000: dist 0.27 loss 3.63
step  782/1000: dist 0.27 loss 4.95
step  783/1000: dist 0.27 loss 3.32
step  784/1000: dist 0.27 loss 1.88
step  785/1000: dist 0.27 loss 3.50
step  786/1000: dist 0.27 loss 3.54
step  787/1000: dist 0.27 loss 3.26
step  788/1000: dist 0.27 loss 2.55
step  789/1000: dist 0.27 loss 2.22
step  790/1000: dist 0.27 loss 3.33
step  791/1000: dist 0.27 loss 1.91
step  792/1000: dist 0.27 loss 0.82
step  793/1000: dist 0.27 loss 1.64
step  794/1000: dist 0.27 loss 1.18
step  795/1000: dist 0.27 loss 0.74
step  796/1000: dist 0.27 loss 0.89
step  797/1000: dist 0.27 loss 1.26
step  798/1000: dist 0.27 loss 1.18
step  799/1000: dist 0.27 loss 0.55
step  800/1000: dist 0.27 loss 0.84
step  801/1000: dist 0.27 loss 0.76
step  802/1000: dist 0.27 loss 0.39
step  803/1000: dist 0.27 loss 0.65
step  804/1000: dist 0.27 loss 0.71
step  805/1000: dist 0.27 loss 0.57
step  806/1000: dist 0.27 loss 0.41
step  807/1000: dist 0.27 loss 0.51
step  808/1000: dist 0.27 loss 0.46
step  809/1000: dist 0.27 loss 0.35
step  810/1000: dist 0.27 loss 0.53
step  811/1000: dist 0.27 loss 0.39
step  812/1000: dist 0.27 loss 0.33
step  813/1000: dist 0.27 loss 0.39
```

```
step  814/1000: dist 0.27 loss 0.37
step  815/1000: dist 0.27 loss 0.36
step  816/1000: dist 0.27 loss 0.33
step  817/1000: dist 0.27 loss 0.35
step  818/1000: dist 0.27 loss 0.30
step  819/1000: dist 0.27 loss 0.34
step  820/1000: dist 0.27 loss 0.33
step  821/1000: dist 0.27 loss 0.29
step  822/1000: dist 0.27 loss 0.31
step  823/1000: dist 0.27 loss 0.31
step  824/1000: dist 0.27 loss 0.30
step  825/1000: dist 0.27 loss 0.29
step  826/1000: dist 0.27 loss 0.30
step  827/1000: dist 0.27 loss 0.28
step  828/1000: dist 0.27 loss 0.29
step  829/1000: dist 0.27 loss 0.28
step  830/1000: dist 0.27 loss 0.28
step  831/1000: dist 0.27 loss 0.29
step  832/1000: dist 0.27 loss 0.27
step  833/1000: dist 0.27 loss 0.28
step  834/1000: dist 0.27 loss 0.28
step  835/1000: dist 0.27 loss 0.28
step  836/1000: dist 0.27 loss 0.27
step  837/1000: dist 0.27 loss 0.28
step  838/1000: dist 0.27 loss 0.27
step  839/1000: dist 0.27 loss 0.27
step  840/1000: dist 0.27 loss 0.27
step  841/1000: dist 0.27 loss 0.27
step  842/1000: dist 0.27 loss 0.27
step  843/1000: dist 0.27 loss 0.27
step  844/1000: dist 0.27 loss 0.27
step  845/1000: dist 0.27 loss 0.27
step  846/1000: dist 0.27 loss 0.27
step  847/1000: dist 0.27 loss 0.27
step  848/1000: dist 0.27 loss 0.27
step  849/1000: dist 0.27 loss 0.27
step  850/1000: dist 0.27 loss 0.27
step  851/1000: dist 0.27 loss 0.27
step  852/1000: dist 0.27 loss 0.27
step  853/1000: dist 0.27 loss 0.27
step  854/1000: dist 0.27 loss 0.27
step  855/1000: dist 0.27 loss 0.27
step  856/1000: dist 0.27 loss 0.27
step  857/1000: dist 0.27 loss 0.27
step  858/1000: dist 0.27 loss 0.27
step  859/1000: dist 0.27 loss 0.27
step  860/1000: dist 0.27 loss 0.27
step  861/1000: dist 0.27 loss 0.27
```

```
step  862/1000: dist 0.27 loss 0.27
step  863/1000: dist 0.27 loss 0.27
step  864/1000: dist 0.27 loss 0.27
step  865/1000: dist 0.27 loss 0.27
step  866/1000: dist 0.27 loss 0.27
step  867/1000: dist 0.27 loss 0.27
step  868/1000: dist 0.27 loss 0.27
step  869/1000: dist 0.27 loss 0.27
step  870/1000: dist 0.27 loss 0.27
step  871/1000: dist 0.27 loss 0.27
step  872/1000: dist 0.27 loss 0.27
step  873/1000: dist 0.27 loss 0.27
step  874/1000: dist 0.27 loss 0.27
step  875/1000: dist 0.27 loss 0.27
step  876/1000: dist 0.27 loss 0.27
step  877/1000: dist 0.27 loss 0.27
step  878/1000: dist 0.27 loss 0.27
step  879/1000: dist 0.27 loss 0.27
step  880/1000: dist 0.27 loss 0.27
step  881/1000: dist 0.27 loss 0.27
step  882/1000: dist 0.27 loss 0.27
step  883/1000: dist 0.27 loss 0.27
step  884/1000: dist 0.27 loss 0.27
step  885/1000: dist 0.27 loss 0.27
step  886/1000: dist 0.27 loss 0.27
step  887/1000: dist 0.27 loss 0.27
step  888/1000: dist 0.27 loss 0.27
step  889/1000: dist 0.27 loss 0.27
step  890/1000: dist 0.27 loss 0.27
step  891/1000: dist 0.27 loss 0.27
step  892/1000: dist 0.27 loss 0.27
step  893/1000: dist 0.27 loss 0.27
step  894/1000: dist 0.27 loss 0.27
step  895/1000: dist 0.27 loss 0.27
step  896/1000: dist 0.27 loss 0.27
step  897/1000: dist 0.27 loss 0.27
step  898/1000: dist 0.27 loss 0.27
step  899/1000: dist 0.27 loss 0.27
step  900/1000: dist 0.27 loss 0.27
step  901/1000: dist 0.27 loss 0.27
step  902/1000: dist 0.27 loss 0.27
step  903/1000: dist 0.27 loss 0.27
step  904/1000: dist 0.27 loss 0.27
step  905/1000: dist 0.27 loss 0.27
step  906/1000: dist 0.27 loss 0.27
step  907/1000: dist 0.26 loss 0.26
step  908/1000: dist 0.26 loss 0.26
step  909/1000: dist 0.26 loss 0.26
```

```
step  910/1000: dist 0.26 loss 0.26
step  911/1000: dist 0.26 loss 0.26
step  912/1000: dist 0.26 loss 0.26
step  913/1000: dist 0.26 loss 0.26
step  914/1000: dist 0.26 loss 0.26
step  915/1000: dist 0.26 loss 0.26
step  916/1000: dist 0.26 loss 0.26
step  917/1000: dist 0.26 loss 0.26
step  918/1000: dist 0.26 loss 0.26
step  919/1000: dist 0.26 loss 0.26
step  920/1000: dist 0.26 loss 0.26
step  921/1000: dist 0.26 loss 0.26
step  922/1000: dist 0.26 loss 0.26
step  923/1000: dist 0.26 loss 0.26
step  924/1000: dist 0.26 loss 0.26
step  925/1000: dist 0.26 loss 0.26
step  926/1000: dist 0.26 loss 0.26
step  927/1000: dist 0.26 loss 0.26
step  928/1000: dist 0.26 loss 0.26
step  929/1000: dist 0.26 loss 0.26
step  930/1000: dist 0.26 loss 0.26
step  931/1000: dist 0.26 loss 0.26
step  932/1000: dist 0.26 loss 0.26
step  933/1000: dist 0.26 loss 0.26
step  934/1000: dist 0.26 loss 0.26
step  935/1000: dist 0.26 loss 0.26
step  936/1000: dist 0.26 loss 0.26
step  937/1000: dist 0.26 loss 0.26
step  938/1000: dist 0.26 loss 0.26
step  939/1000: dist 0.26 loss 0.26
step  940/1000: dist 0.26 loss 0.26
step  941/1000: dist 0.26 loss 0.26
step  942/1000: dist 0.26 loss 0.26
step  943/1000: dist 0.26 loss 0.26
step  944/1000: dist 0.26 loss 0.26
step  945/1000: dist 0.26 loss 0.26
step  946/1000: dist 0.26 loss 0.26
step  947/1000: dist 0.26 loss 0.26
step  948/1000: dist 0.26 loss 0.26
step  949/1000: dist 0.26 loss 0.26
step  950/1000: dist 0.26 loss 0.26
step  951/1000: dist 0.26 loss 0.26
step  952/1000: dist 0.26 loss 0.26
step  953/1000: dist 0.26 loss 0.26
step  954/1000: dist 0.26 loss 0.26
step  955/1000: dist 0.26 loss 0.26
step  956/1000: dist 0.26 loss 0.26
step  957/1000: dist 0.26 loss 0.26
```

```
step  958/1000: dist 0.26 loss 0.26
step  959/1000: dist 0.26 loss 0.26
step  960/1000: dist 0.26 loss 0.26
step  961/1000: dist 0.26 loss 0.26
step  962/1000: dist 0.26 loss 0.26
step  963/1000: dist 0.26 loss 0.26
step  964/1000: dist 0.26 loss 0.26
step  965/1000: dist 0.26 loss 0.26
step  966/1000: dist 0.26 loss 0.26
step  967/1000: dist 0.26 loss 0.26
step  968/1000: dist 0.26 loss 0.26
step  969/1000: dist 0.26 loss 0.26
step  970/1000: dist 0.26 loss 0.26
step  971/1000: dist 0.26 loss 0.26
step  972/1000: dist 0.26 loss 0.26
step  973/1000: dist 0.26 loss 0.26
step  974/1000: dist 0.26 loss 0.26
step  975/1000: dist 0.26 loss 0.26
step  976/1000: dist 0.26 loss 0.26
step  977/1000: dist 0.26 loss 0.26
step  978/1000: dist 0.26 loss 0.26
step  979/1000: dist 0.26 loss 0.26
step  980/1000: dist 0.26 loss 0.26
step  981/1000: dist 0.26 loss 0.26
step  982/1000: dist 0.26 loss 0.26
step  983/1000: dist 0.26 loss 0.26
step  984/1000: dist 0.26 loss 0.26
step  985/1000: dist 0.26 loss 0.26
step  986/1000: dist 0.26 loss 0.26
step  987/1000: dist 0.26 loss 0.26
step  988/1000: dist 0.26 loss 0.26
step  989/1000: dist 0.26 loss 0.26
step  990/1000: dist 0.26 loss 0.26
step  991/1000: dist 0.26 loss 0.26
step  992/1000: dist 0.26 loss 0.26
step  993/1000: dist 0.26 loss 0.26
step  994/1000: dist 0.26 loss 0.26
step  995/1000: dist 0.26 loss 0.26
step  996/1000: dist 0.26 loss 0.26
step  997/1000: dist 0.26 loss 0.26
step  998/1000: dist 0.26 loss 0.26
step  999/1000: dist 0.26 loss 0.26
step 1000/1000: dist 0.26 loss 0.26
Elapsed: 121.4 s
```

4.6 Convert Target to a GAN

Next, we convert the target to a GAN latent vector. This process will also take several minutes.

[ ]:
```
# HIDE OUTPUT
cmd = f"python /content/stylegan2-ada-pytorch/projector.py "\
  f"--save-video 0 --num-steps 1000 --outdir=out_target "\
  f"--target=cropped_target.png --network={NETWORK}"
!{cmd}
```

```
Loading networks from "https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/ffhq.pkl"…
Computing W midpoint and stddev using 10000 samples…
Setting up PyTorch plugin "bias_act_plugin"… Done.
Setting up PyTorch plugin "upfirdn2d_plugin"… Done.
step    1/1000: dist 0.63 loss 24568.77
step    2/1000: dist 0.60 loss 27642.11
step    3/1000: dist 0.60 loss 27167.12
step    4/1000: dist 0.57 loss 26253.35
step    5/1000: dist 0.61 loss 24959.81
step    6/1000: dist 0.60 loss 23356.12
step    7/1000: dist 0.56 loss 21512.13
step    8/1000: dist 0.59 loss 19487.23
step    9/1000: dist 0.55 loss 17341.27
step   10/1000: dist 0.56 loss 15140.35
step   11/1000: dist 0.63 loss 12949.49
step   12/1000: dist 0.55 loss 10820.17
step   13/1000: dist 0.56 loss 8802.83
step   14/1000: dist 0.61 loss 6946.22
step   15/1000: dist 0.58 loss 5316.71
step   16/1000: dist 0.52 loss 3971.15
step   17/1000: dist 0.55 loss 2941.10
step   18/1000: dist 0.50 loss 2216.22
step   19/1000: dist 0.51 loss 1758.78
step   20/1000: dist 0.52 loss 1567.52
step   21/1000: dist 0.51 loss 1602.28
step   22/1000: dist 0.49 loss 1787.82
step   23/1000: dist 0.48 loss 2053.34
step   24/1000: dist 0.48 loss 2327.56
step   25/1000: dist 0.48 loss 2538.77
step   26/1000: dist 0.48 loss 2637.36
step   27/1000: dist 0.49 loss 2604.07
step   28/1000: dist 0.49 loss 2477.09
step   29/1000: dist 0.51 loss 2317.29
step   30/1000: dist 0.48 loss 2120.27
step   31/1000: dist 0.47 loss 1884.58
step   32/1000: dist 0.50 loss 1627.86
step   33/1000: dist 0.48 loss 1388.84
step   34/1000: dist 0.45 loss 1183.89
step   35/1000: dist 0.45 loss 1026.39
step   36/1000: dist 0.45 loss 909.69
step   37/1000: dist 0.46 loss 830.82
```

```
step   38/1000: dist 0.48 loss 805.27
step   39/1000: dist 0.45 loss 812.57
step   40/1000: dist 0.45 loss 834.66
step   41/1000: dist 0.47 loss 828.71
step   42/1000: dist 0.47 loss 761.27
step   43/1000: dist 0.49 loss 651.15
step   44/1000: dist 0.49 loss 521.39
step   45/1000: dist 0.46 loss 402.71
step   46/1000: dist 0.47 loss 318.35
step   47/1000: dist 0.49 loss 263.81
step   48/1000: dist 0.48 loss 241.38
step   49/1000: dist 0.47 loss 251.26
step   50/1000: dist 0.46 loss 304.01
step   51/1000: dist 0.44 loss 369.49
step   52/1000: dist 0.45 loss 401.52
step   53/1000: dist 0.46 loss 400.59
step   54/1000: dist 0.46 loss 369.86
step   55/1000: dist 0.46 loss 282.55
step   56/1000: dist 0.45 loss 201.64
step   57/1000: dist 0.45 loss 149.67
step   58/1000: dist 0.47 loss 80.04
step   59/1000: dist 0.48 loss 60.75
step   60/1000: dist 0.46 loss 60.34
step   61/1000: dist 0.45 loss 62.42
step   62/1000: dist 0.42 loss 95.78
step   63/1000: dist 0.45 loss 108.14
step   64/1000: dist 0.43 loss 133.23
step   65/1000: dist 0.43 loss 133.46
step   66/1000: dist 0.44 loss 115.09
step   67/1000: dist 0.43 loss 106.48
step   68/1000: dist 0.43 loss 71.30
step   69/1000: dist 0.45 loss 49.85
step   70/1000: dist 0.43 loss 29.17
step   71/1000: dist 0.44 loss 20.93
step   72/1000: dist 0.44 loss 18.21
step   73/1000: dist 0.42 loss 20.99
step   74/1000: dist 0.43 loss 30.88
step   75/1000: dist 0.45 loss 32.53
step   76/1000: dist 0.45 loss 40.11
step   77/1000: dist 0.45 loss 37.00
step   78/1000: dist 0.44 loss 32.15
step   79/1000: dist 0.41 loss 30.82
step   80/1000: dist 0.42 loss 27.17
step   81/1000: dist 0.41 loss 22.75
step   82/1000: dist 0.42 loss 13.63
step   83/1000: dist 0.42 loss 13.14
step   84/1000: dist 0.41 loss 15.89
step   85/1000: dist 0.40 loss 15.54
```

```
step   86/1000: dist 0.42 loss 12.85
step   87/1000: dist 0.42 loss 10.69
step   88/1000: dist 0.44 loss 11.77
step   89/1000: dist 0.41 loss 10.23
step   90/1000: dist 0.41 loss 7.86
step   91/1000: dist 0.42 loss 6.77
step   92/1000: dist 0.41 loss 6.25
step   93/1000: dist 0.41 loss 7.11
step   94/1000: dist 0.41 loss 8.53
step   95/1000: dist 0.40 loss 10.37
step   96/1000: dist 0.41 loss 11.90
step   97/1000: dist 0.41 loss 11.11
step   98/1000: dist 0.41 loss 7.58
step   99/1000: dist 0.41 loss 3.91
step  100/1000: dist 0.40 loss 3.10
step  101/1000: dist 0.40 loss 4.27
step  102/1000: dist 0.40 loss 4.44
step  103/1000: dist 0.41 loss 2.91
step  104/1000: dist 0.44 loss 2.21
step  105/1000: dist 0.40 loss 3.64
step  106/1000: dist 0.42 loss 5.15
step  107/1000: dist 0.42 loss 5.45
step  108/1000: dist 0.41 loss 5.54
step  109/1000: dist 0.42 loss 5.72
step  110/1000: dist 0.40 loss 5.34
step  111/1000: dist 0.40 loss 4.88
step  112/1000: dist 0.39 loss 6.34
step  113/1000: dist 0.40 loss 10.53
step  114/1000: dist 0.41 loss 14.98
step  115/1000: dist 0.41 loss 14.98
step  116/1000: dist 0.42 loss 9.25
step  117/1000: dist 0.40 loss 5.78
step  118/1000: dist 0.42 loss 9.34
step  119/1000: dist 0.41 loss 12.45
step  120/1000: dist 0.42 loss 10.44
step  121/1000: dist 0.41 loss 11.29
step  122/1000: dist 0.41 loss 16.97
step  123/1000: dist 0.41 loss 13.82
step  124/1000: dist 0.40 loss 3.49
step  125/1000: dist 0.40 loss 4.26
step  126/1000: dist 0.40 loss 12.04
step  127/1000: dist 0.40 loss 9.42
step  128/1000: dist 0.39 loss 2.48
step  129/1000: dist 0.42 loss 5.23
step  130/1000: dist 0.40 loss 8.20
step  131/1000: dist 0.42 loss 3.68
step  132/1000: dist 0.40 loss 2.79
step  133/1000: dist 0.39 loss 6.47
```

```
step  134/1000: dist 0.39 loss 5.09
step  135/1000: dist 0.42 loss 3.70
step  136/1000: dist 0.39 loss 8.50
step  137/1000: dist 0.39 loss 13.20
step  138/1000: dist 0.39 loss 15.67
step  139/1000: dist 0.39 loss 15.73
step  140/1000: dist 0.39 loss 10.51
step  141/1000: dist 0.39 loss 8.84
step  142/1000: dist 0.39 loss 15.89
step  143/1000: dist 0.40 loss 17.21
step  144/1000: dist 0.38 loss 8.13
step  145/1000: dist 0.40 loss 7.21
step  146/1000: dist 0.39 loss 15.88
step  147/1000: dist 0.41 loss 16.82
step  148/1000: dist 0.40 loss 10.13
step  149/1000: dist 0.39 loss 6.30
step  150/1000: dist 0.39 loss 4.91
step  151/1000: dist 0.40 loss 5.57
step  152/1000: dist 0.40 loss 9.53
step  153/1000: dist 0.39 loss 9.50
step  154/1000: dist 0.39 loss 5.56
step  155/1000: dist 0.38 loss 8.20
step  156/1000: dist 0.39 loss 14.62
step  157/1000: dist 0.39 loss 13.69
step  158/1000: dist 0.39 loss 7.50
step  159/1000: dist 0.38 loss 4.26
step  160/1000: dist 0.40 loss 5.56
step  161/1000: dist 0.39 loss 9.38
step  162/1000: dist 0.40 loss 13.38
step  163/1000: dist 0.42 loss 16.43
step  164/1000: dist 0.39 loss 22.84
step  165/1000: dist 0.39 loss 35.32
step  166/1000: dist 0.38 loss 38.83
step  167/1000: dist 0.38 loss 17.69
step  168/1000: dist 0.40 loss 7.40
step  169/1000: dist 0.40 loss 23.03
step  170/1000: dist 0.41 loss 17.91
step  171/1000: dist 0.40 loss 3.20
step  172/1000: dist 0.40 loss 13.44
step  173/1000: dist 0.40 loss 13.49
step  174/1000: dist 0.39 loss 7.86
step  175/1000: dist 0.39 loss 14.67
step  176/1000: dist 0.39 loss 9.40
step  177/1000: dist 0.39 loss 10.84
step  178/1000: dist 0.38 loss 16.66
step  179/1000: dist 0.39 loss 8.86
step  180/1000: dist 0.38 loss 8.32
step  181/1000: dist 0.39 loss 6.36
```

```
step  182/1000: dist 0.39 loss 5.12
step  183/1000: dist 0.39 loss 10.92
step  184/1000: dist 0.39 loss 10.32
step  185/1000: dist 0.39 loss 13.67
step  186/1000: dist 0.39 loss 15.94
step  187/1000: dist 0.39 loss 13.11
step  188/1000: dist 0.39 loss 11.61
step  189/1000: dist 0.39 loss 8.15
step  190/1000: dist 0.38 loss 10.88
step  191/1000: dist 0.40 loss 13.93
step  192/1000: dist 0.38 loss 11.13
step  193/1000: dist 0.37 loss 10.81
step  194/1000: dist 0.37 loss 15.03
step  195/1000: dist 0.38 loss 25.86
step  196/1000: dist 0.39 loss 37.01
step  197/1000: dist 0.38 loss 37.96
step  198/1000: dist 0.39 loss 21.87
step  199/1000: dist 0.38 loss 7.17
step  200/1000: dist 0.39 loss 14.34
step  201/1000: dist 0.38 loss 19.18
step  202/1000: dist 0.38 loss 9.69
step  203/1000: dist 0.38 loss 10.58
step  204/1000: dist 0.38 loss 12.80
step  205/1000: dist 0.38 loss 4.86
step  206/1000: dist 0.38 loss 7.54
step  207/1000: dist 0.37 loss 11.50
step  208/1000: dist 0.38 loss 3.47
step  209/1000: dist 0.37 loss 4.03
step  210/1000: dist 0.37 loss 8.23
step  211/1000: dist 0.37 loss 3.73
step  212/1000: dist 0.37 loss 3.60
step  213/1000: dist 0.37 loss 4.48
step  214/1000: dist 0.37 loss 2.89
step  215/1000: dist 0.38 loss 4.53
step  216/1000: dist 0.37 loss 2.58
step  217/1000: dist 0.37 loss 1.50
step  218/1000: dist 0.37 loss 4.20
step  219/1000: dist 0.38 loss 2.48
step  220/1000: dist 0.38 loss 1.55
step  221/1000: dist 0.37 loss 2.85
step  222/1000: dist 0.38 loss 2.44
step  223/1000: dist 0.37 loss 3.21
step  224/1000: dist 0.38 loss 3.23
step  225/1000: dist 0.38 loss 3.19
step  226/1000: dist 0.38 loss 5.29
step  227/1000: dist 0.37 loss 5.61
step  228/1000: dist 0.38 loss 5.14
step  229/1000: dist 0.38 loss 4.79
```

```
step  230/1000: dist 0.37 loss 2.85
step  231/1000: dist 0.37 loss 1.50
step  232/1000: dist 0.37 loss 1.02
step  233/1000: dist 0.38 loss 1.11
step  234/1000: dist 0.37 loss 2.43
step  235/1000: dist 0.38 loss 2.87
step  236/1000: dist 0.37 loss 2.11
step  237/1000: dist 0.37 loss 1.41
step  238/1000: dist 0.37 loss 0.73
step  239/1000: dist 0.37 loss 0.76
step  240/1000: dist 0.36 loss 1.32
step  241/1000: dist 0.37 loss 1.43
step  242/1000: dist 0.37 loss 1.41
step  243/1000: dist 0.37 loss 1.16
step  244/1000: dist 0.36 loss 0.65
step  245/1000: dist 0.36 loss 0.58
step  246/1000: dist 0.37 loss 0.78
step  247/1000: dist 0.37 loss 1.00
step  248/1000: dist 0.38 loss 1.29
step  249/1000: dist 0.38 loss 1.37
step  250/1000: dist 0.39 loss 1.63
step  251/1000: dist 0.36 loss 2.78
step  252/1000: dist 0.37 loss 5.55
step  253/1000: dist 0.37 loss 10.95
step  254/1000: dist 0.36 loss 18.49
step  255/1000: dist 0.36 loss 21.99
step  256/1000: dist 0.36 loss 13.80
step  257/1000: dist 0.37 loss 3.10
step  258/1000: dist 0.37 loss 6.43
step  259/1000: dist 0.36 loss 16.95
step  260/1000: dist 0.37 loss 18.68
step  261/1000: dist 0.36 loss 19.48
step  262/1000: dist 0.35 loss 31.48
step  263/1000: dist 0.38 loss 37.36
step  264/1000: dist 0.36 loss 25.85
step  265/1000: dist 0.36 loss 16.00
step  266/1000: dist 0.36 loss 17.50
step  267/1000: dist 0.36 loss 19.14
step  268/1000: dist 0.37 loss 16.35
step  269/1000: dist 0.36 loss 10.63
step  270/1000: dist 0.35 loss 5.18
step  271/1000: dist 0.36 loss 8.53
step  272/1000: dist 0.36 loss 13.82
step  273/1000: dist 0.36 loss 8.40
step  274/1000: dist 0.35 loss 3.64
step  275/1000: dist 0.37 loss 8.15
step  276/1000: dist 0.36 loss 9.72
step  277/1000: dist 0.37 loss 8.79
```

```
step  278/1000: dist 0.36 loss 14.53
step  279/1000: dist 0.35 loss 19.90
step  280/1000: dist 0.35 loss 18.01
step  281/1000: dist 0.36 loss 12.53
step  282/1000: dist 0.36 loss 7.74
step  283/1000: dist 0.36 loss 7.64
step  284/1000: dist 0.35 loss 14.28
step  285/1000: dist 0.36 loss 22.87
step  286/1000: dist 0.35 loss 30.05
step  287/1000: dist 0.35 loss 42.14
step  288/1000: dist 0.36 loss 57.32
step  289/1000: dist 0.37 loss 58.00
step  290/1000: dist 0.37 loss 40.22
step  291/1000: dist 0.36 loss 29.92
step  292/1000: dist 0.36 loss 37.92
step  293/1000: dist 0.36 loss 39.55
step  294/1000: dist 0.35 loss 24.88
step  295/1000: dist 0.35 loss 19.24
step  296/1000: dist 0.36 loss 29.04
step  297/1000: dist 0.35 loss 28.44
step  298/1000: dist 0.35 loss 14.75
step  299/1000: dist 0.36 loss 13.28
step  300/1000: dist 0.35 loss 23.15
step  301/1000: dist 0.36 loss 21.66
step  302/1000: dist 0.36 loss 11.28
step  303/1000: dist 0.36 loss 8.81
step  304/1000: dist 0.36 loss 10.64
step  305/1000: dist 0.37 loss 8.16
step  306/1000: dist 0.35 loss 7.31
step  307/1000: dist 0.36 loss 10.09
step  308/1000: dist 0.35 loss 8.90
step  309/1000: dist 0.35 loss 4.98
step  310/1000: dist 0.35 loss 4.78
step  311/1000: dist 0.35 loss 6.37
step  312/1000: dist 0.35 loss 6.58
step  313/1000: dist 0.35 loss 7.24
step  314/1000: dist 0.35 loss 9.44
step  315/1000: dist 0.35 loss 13.21
step  316/1000: dist 0.35 loss 18.81
step  317/1000: dist 0.35 loss 21.85
step  318/1000: dist 0.35 loss 18.14
step  319/1000: dist 0.35 loss 11.98
step  320/1000: dist 0.35 loss 11.39
step  321/1000: dist 0.36 loss 14.91
step  322/1000: dist 0.35 loss 15.84
step  323/1000: dist 0.36 loss 11.16
step  324/1000: dist 0.35 loss 5.78
step  325/1000: dist 0.35 loss 6.52
```

```
step  326/1000: dist 0.36 loss 9.82
step  327/1000: dist 0.35 loss 8.05
step  328/1000: dist 0.35 loss 3.55
step  329/1000: dist 0.36 loss 3.58
step  330/1000: dist 0.35 loss 6.34
step  331/1000: dist 0.34 loss 5.66
step  332/1000: dist 0.35 loss 2.97
step  333/1000: dist 0.35 loss 2.66
step  334/1000: dist 0.35 loss 3.64
step  335/1000: dist 0.35 loss 3.52
step  336/1000: dist 0.35 loss 3.05
step  337/1000: dist 0.35 loss 2.88
step  338/1000: dist 0.35 loss 2.95
step  339/1000: dist 0.35 loss 4.19
step  340/1000: dist 0.34 loss 6.59
step  341/1000: dist 0.35 loss 9.43
step  342/1000: dist 0.35 loss 13.87
step  343/1000: dist 0.34 loss 20.12
step  344/1000: dist 0.35 loss 23.36
step  345/1000: dist 0.34 loss 20.32
step  346/1000: dist 0.35 loss 16.37
step  347/1000: dist 0.35 loss 16.01
step  348/1000: dist 0.34 loss 13.92
step  349/1000: dist 0.34 loss 8.05
step  350/1000: dist 0.34 loss 4.54
step  351/1000: dist 0.34 loss 7.58
step  352/1000: dist 0.34 loss 10.71
step  353/1000: dist 0.35 loss 7.02
step  354/1000: dist 0.34 loss 2.61
step  355/1000: dist 0.34 loss 5.11
step  356/1000: dist 0.34 loss 9.35
step  357/1000: dist 0.34 loss 8.74
step  358/1000: dist 0.34 loss 9.35
step  359/1000: dist 0.34 loss 17.84
step  360/1000: dist 0.34 loss 25.91
step  361/1000: dist 0.34 loss 22.36
step  362/1000: dist 0.34 loss 9.73
step  363/1000: dist 0.34 loss 3.08
step  364/1000: dist 0.34 loss 7.06
step  365/1000: dist 0.34 loss 11.71
step  366/1000: dist 0.33 loss 9.02
step  367/1000: dist 0.34 loss 3.32
step  368/1000: dist 0.34 loss 3.70
step  369/1000: dist 0.34 loss 7.23
step  370/1000: dist 0.34 loss 5.62
step  371/1000: dist 0.34 loss 2.02
step  372/1000: dist 0.34 loss 3.21
step  373/1000: dist 0.33 loss 4.99
```

```
step  374/1000: dist 0.34 loss 2.75
step  375/1000: dist 0.34 loss 1.43
step  376/1000: dist 0.33 loss 3.18
step  377/1000: dist 0.34 loss 3.04
step  378/1000: dist 0.33 loss 1.30
step  379/1000: dist 0.34 loss 2.00
step  380/1000: dist 0.34 loss 3.04
step  381/1000: dist 0.34 loss 2.39
step  382/1000: dist 0.34 loss 3.30
step  383/1000: dist 0.33 loss 6.57
step  384/1000: dist 0.34 loss 10.44
step  385/1000: dist 0.35 loss 16.30
step  386/1000: dist 0.35 loss 22.44
step  387/1000: dist 0.34 loss 20.03
step  388/1000: dist 0.34 loss 7.82
step  389/1000: dist 0.34 loss 0.80
step  390/1000: dist 0.34 loss 5.98
step  391/1000: dist 0.34 loss 11.64
step  392/1000: dist 0.35 loss 7.74
step  393/1000: dist 0.34 loss 1.41
step  394/1000: dist 0.33 loss 3.10
step  395/1000: dist 0.33 loss 7.24
step  396/1000: dist 0.34 loss 4.74
step  397/1000: dist 0.34 loss 0.86
step  398/1000: dist 0.34 loss 2.83
step  399/1000: dist 0.33 loss 4.92
step  400/1000: dist 0.34 loss 2.22
step  401/1000: dist 0.34 loss 0.73
step  402/1000: dist 0.34 loss 2.93
step  403/1000: dist 0.35 loss 3.00
step  404/1000: dist 0.35 loss 0.81
step  405/1000: dist 0.33 loss 1.24
step  406/1000: dist 0.33 loss 2.59
step  407/1000: dist 0.33 loss 1.49
step  408/1000: dist 0.34 loss 0.74
step  409/1000: dist 0.34 loss 2.00
step  410/1000: dist 0.34 loss 2.39
step  411/1000: dist 0.34 loss 1.92
step  412/1000: dist 0.33 loss 3.25
step  413/1000: dist 0.34 loss 5.71
step  414/1000: dist 0.34 loss 7.95
step  415/1000: dist 0.34 loss 11.13
step  416/1000: dist 0.34 loss 14.47
step  417/1000: dist 0.33 loss 13.91
step  418/1000: dist 0.33 loss 9.14
step  419/1000: dist 0.33 loss 5.72
step  420/1000: dist 0.33 loss 7.88
step  421/1000: dist 0.33 loss 13.35
```

```
step  422/1000: dist 0.34 loss 15.98
step  423/1000: dist 0.34 loss 11.85
step  424/1000: dist 0.34 loss 4.34
step  425/1000: dist 0.34 loss 1.30
step  426/1000: dist 0.33 loss 4.18
step  427/1000: dist 0.33 loss 7.01
step  428/1000: dist 0.33 loss 5.81
step  429/1000: dist 0.35 loss 3.10
step  430/1000: dist 0.33 loss 2.29
step  431/1000: dist 0.33 loss 3.03
step  432/1000: dist 0.33 loss 3.42
step  433/1000: dist 0.33 loss 3.30
step  434/1000: dist 0.33 loss 3.13
step  435/1000: dist 0.33 loss 3.17
step  436/1000: dist 0.33 loss 3.97
step  437/1000: dist 0.33 loss 6.23
step  438/1000: dist 0.33 loss 9.71
step  439/1000: dist 0.33 loss 13.88
step  440/1000: dist 0.33 loss 18.35
step  441/1000: dist 0.33 loss 20.96
step  442/1000: dist 0.33 loss 18.03
step  443/1000: dist 0.33 loss 10.45
step  444/1000: dist 0.33 loss 4.76
step  445/1000: dist 0.34 loss 5.05
step  446/1000: dist 0.33 loss 7.73
step  447/1000: dist 0.33 loss 7.25
step  448/1000: dist 0.32 loss 4.44
step  449/1000: dist 0.33 loss 4.04
step  450/1000: dist 0.32 loss 5.65
step  451/1000: dist 0.33 loss 5.03
step  452/1000: dist 0.33 loss 2.42
step  453/1000: dist 0.32 loss 2.29
step  454/1000: dist 0.33 loss 5.08
step  455/1000: dist 0.32 loss 6.55
step  456/1000: dist 0.32 loss 6.01
step  457/1000: dist 0.33 loss 7.84
step  458/1000: dist 0.32 loss 13.89
step  459/1000: dist 0.32 loss 19.44
step  460/1000: dist 0.32 loss 18.82
step  461/1000: dist 0.32 loss 11.18
step  462/1000: dist 0.33 loss 3.96
step  463/1000: dist 0.32 loss 4.37
step  464/1000: dist 0.32 loss 10.30
step  465/1000: dist 0.32 loss 13.48
step  466/1000: dist 0.32 loss 10.60
step  467/1000: dist 0.32 loss 8.22
step  468/1000: dist 0.32 loss 10.44
step  469/1000: dist 0.32 loss 11.50
```

```
step  470/1000: dist 0.32 loss 6.87
step  471/1000: dist 0.33 loss 2.77
step  472/1000: dist 0.32 loss 5.52
step  473/1000: dist 0.32 loss 11.72
step  474/1000: dist 0.32 loss 17.15
step  475/1000: dist 0.32 loss 23.26
step  476/1000: dist 0.32 loss 27.49
step  477/1000: dist 0.32 loss 19.85
step  478/1000: dist 0.32 loss 5.23
step  479/1000: dist 0.33 loss 2.14
step  480/1000: dist 0.32 loss 11.52
step  481/1000: dist 0.32 loss 14.64
step  482/1000: dist 0.32 loss 5.63
step  483/1000: dist 0.32 loss 1.35
step  484/1000: dist 0.32 loss 7.80
step  485/1000: dist 0.32 loss 10.81
step  486/1000: dist 0.33 loss 7.70
step  487/1000: dist 0.33 loss 11.75
step  488/1000: dist 0.32 loss 23.55
step  489/1000: dist 0.32 loss 30.50
step  490/1000: dist 0.32 loss 26.71
step  491/1000: dist 0.33 loss 14.44
step  492/1000: dist 0.32 loss 4.90
step  493/1000: dist 0.32 loss 9.22
step  494/1000: dist 0.32 loss 17.97
step  495/1000: dist 0.32 loss 15.02
step  496/1000: dist 0.32 loss 12.14
step  497/1000: dist 0.32 loss 25.29
step  498/1000: dist 0.31 loss 37.94
step  499/1000: dist 0.31 loss 37.77
step  500/1000: dist 0.32 loss 34.20
step  501/1000: dist 0.32 loss 22.55
step  502/1000: dist 0.32 loss 8.23
step  503/1000: dist 0.32 loss 17.44
step  504/1000: dist 0.32 loss 37.55
step  505/1000: dist 0.31 loss 39.15
step  506/1000: dist 0.32 loss 38.22
step  507/1000: dist 0.32 loss 38.50
step  508/1000: dist 0.32 loss 19.56
step  509/1000: dist 0.32 loss 7.97
step  510/1000: dist 0.32 loss 22.48
step  511/1000: dist 0.32 loss 23.17
step  512/1000: dist 0.32 loss 7.87
step  513/1000: dist 0.32 loss 15.68
step  514/1000: dist 0.32 loss 23.92
step  515/1000: dist 0.32 loss 15.09
step  516/1000: dist 0.32 loss 22.75
step  517/1000: dist 0.32 loss 31.83
```

```
step  518/1000: dist 0.31 loss 22.17
step  519/1000: dist 0.31 loss 22.18
step  520/1000: dist 0.31 loss 26.55
step  521/1000: dist 0.31 loss 22.81
step  522/1000: dist 0.32 loss 24.71
step  523/1000: dist 0.31 loss 23.00
step  524/1000: dist 0.31 loss 21.71
step  525/1000: dist 0.31 loss 29.47
step  526/1000: dist 0.31 loss 26.77
step  527/1000: dist 0.31 loss 14.91
step  528/1000: dist 0.32 loss 8.91
step  529/1000: dist 0.32 loss 13.27
step  530/1000: dist 0.31 loss 26.22
step  531/1000: dist 0.31 loss 31.21
step  532/1000: dist 0.31 loss 27.09
step  533/1000: dist 0.31 loss 23.98
step  534/1000: dist 0.31 loss 17.37
step  535/1000: dist 0.31 loss 11.66
step  536/1000: dist 0.31 loss 14.19
step  537/1000: dist 0.31 loss 23.17
step  538/1000: dist 0.31 loss 32.38
step  539/1000: dist 0.31 loss 37.40
step  540/1000: dist 0.31 loss 39.18
step  541/1000: dist 0.31 loss 27.70
step  542/1000: dist 0.31 loss 10.29
step  543/1000: dist 0.31 loss 15.83
step  544/1000: dist 0.31 loss 30.35
step  545/1000: dist 0.31 loss 24.29
step  546/1000: dist 0.31 loss 14.30
step  547/1000: dist 0.32 loss 20.68
step  548/1000: dist 0.31 loss 19.18
step  549/1000: dist 0.31 loss 4.18
step  550/1000: dist 0.31 loss 5.86
step  551/1000: dist 0.32 loss 13.53
step  552/1000: dist 0.31 loss 7.49
step  553/1000: dist 0.31 loss 7.37
step  554/1000: dist 0.31 loss 10.44
step  555/1000: dist 0.31 loss 3.34
step  556/1000: dist 0.32 loss 3.55
step  557/1000: dist 0.31 loss 9.78
step  558/1000: dist 0.32 loss 7.77
step  559/1000: dist 0.32 loss 8.57
step  560/1000: dist 0.31 loss 14.80
step  561/1000: dist 0.31 loss 18.29
step  562/1000: dist 0.31 loss 22.51
step  563/1000: dist 0.32 loss 21.73
step  564/1000: dist 0.32 loss 9.16
step  565/1000: dist 0.31 loss 1.69
```

```
step  566/1000: dist 0.31 loss 6.20
step  567/1000: dist 0.31 loss 11.43
step  568/1000: dist 0.32 loss 11.19
step  569/1000: dist 0.31 loss 6.65
step  570/1000: dist 0.31 loss 7.06
step  571/1000: dist 0.31 loss 15.96
step  572/1000: dist 0.31 loss 21.67
step  573/1000: dist 0.32 loss 20.10
step  574/1000: dist 0.32 loss 17.21
step  575/1000: dist 0.31 loss 11.33
step  576/1000: dist 0.31 loss 4.11
step  577/1000: dist 0.32 loss 4.67
step  578/1000: dist 0.31 loss 11.42
step  579/1000: dist 0.32 loss 12.86
step  580/1000: dist 0.33 loss 7.76
step  581/1000: dist 0.32 loss 7.98
step  582/1000: dist 0.32 loss 15.19
step  583/1000: dist 0.32 loss 16.73
step  584/1000: dist 0.32 loss 9.93
step  585/1000: dist 0.32 loss 5.47
step  586/1000: dist 0.32 loss 5.21
step  587/1000: dist 0.32 loss 3.95
step  588/1000: dist 0.31 loss 3.71
step  589/1000: dist 0.31 loss 6.76
step  590/1000: dist 0.31 loss 7.78
step  591/1000: dist 0.31 loss 4.75
step  592/1000: dist 0.31 loss 4.03
step  593/1000: dist 0.31 loss 8.56
step  594/1000: dist 0.31 loss 13.80
step  595/1000: dist 0.31 loss 16.86
step  596/1000: dist 0.31 loss 18.13
step  597/1000: dist 0.30 loss 14.87
step  598/1000: dist 0.31 loss 7.32
step  599/1000: dist 0.31 loss 2.98
step  600/1000: dist 0.31 loss 5.80
step  601/1000: dist 0.31 loss 10.04
step  602/1000: dist 0.31 loss 10.01
step  603/1000: dist 0.31 loss 7.76
step  604/1000: dist 0.31 loss 9.13
step  605/1000: dist 0.31 loss 14.98
step  606/1000: dist 0.30 loss 19.76
step  607/1000: dist 0.31 loss 18.18
step  608/1000: dist 0.31 loss 11.99
step  609/1000: dist 0.30 loss 6.24
step  610/1000: dist 0.30 loss 3.85
step  611/1000: dist 0.30 loss 4.49
step  612/1000: dist 0.30 loss 7.00
step  613/1000: dist 0.31 loss 8.26
```

```
step  614/1000: dist 0.30 loss 5.26
step  615/1000: dist 0.30 loss 1.53
step  616/1000: dist 0.30 loss 2.46
step  617/1000: dist 0.30 loss 5.71
step  618/1000: dist 0.30 loss 5.16
step  619/1000: dist 0.30 loss 1.92
step  620/1000: dist 0.30 loss 1.88
step  621/1000: dist 0.30 loss 4.64
step  622/1000: dist 0.30 loss 5.29
step  623/1000: dist 0.30 loss 4.01
step  624/1000: dist 0.30 loss 4.80
step  625/1000: dist 0.30 loss 7.59
step  626/1000: dist 0.30 loss 9.10
step  627/1000: dist 0.30 loss 8.84
step  628/1000: dist 0.30 loss 8.35
step  629/1000: dist 0.31 loss 7.26
step  630/1000: dist 0.30 loss 4.57
step  631/1000: dist 0.30 loss 1.69
step  632/1000: dist 0.30 loss 0.69
step  633/1000: dist 0.30 loss 1.75
step  634/1000: dist 0.30 loss 3.36
step  635/1000: dist 0.30 loss 4.01
step  636/1000: dist 0.30 loss 3.38
step  637/1000: dist 0.30 loss 2.57
step  638/1000: dist 0.30 loss 3.09
step  639/1000: dist 0.30 loss 5.89
step  640/1000: dist 0.30 loss 11.91
step  641/1000: dist 0.30 loss 22.09
step  642/1000: dist 0.30 loss 34.58
step  643/1000: dist 0.30 loss 37.51
step  644/1000: dist 0.30 loss 21.20
step  645/1000: dist 0.30 loss 2.17
step  646/1000: dist 0.30 loss 6.31
step  647/1000: dist 0.30 loss 19.65
step  648/1000: dist 0.30 loss 13.66
step  649/1000: dist 0.30 loss 1.24
step  650/1000: dist 0.30 loss 7.03
step  651/1000: dist 0.30 loss 13.04
step  652/1000: dist 0.30 loss 4.89
step  653/1000: dist 0.29 loss 4.49
step  654/1000: dist 0.30 loss 12.29
step  655/1000: dist 0.30 loss 9.99
step  656/1000: dist 0.30 loss 9.71
step  657/1000: dist 0.30 loss 17.31
step  658/1000: dist 0.30 loss 15.67
step  659/1000: dist 0.30 loss 9.88
step  660/1000: dist 0.30 loss 7.85
step  661/1000: dist 0.30 loss 2.58
```

```
step  662/1000: dist 0.30 loss 1.62
step  663/1000: dist 0.30 loss 7.38
step  664/1000: dist 0.30 loss 7.11
step  665/1000: dist 0.30 loss 3.95
step  666/1000: dist 0.30 loss 3.27
step  667/1000: dist 0.30 loss 1.98
step  668/1000: dist 0.30 loss 3.73
step  669/1000: dist 0.30 loss 6.29
step  670/1000: dist 0.30 loss 4.94
step  671/1000: dist 0.30 loss 5.91
step  672/1000: dist 0.30 loss 10.42
step  673/1000: dist 0.30 loss 16.02
step  674/1000: dist 0.29 loss 24.26
step  675/1000: dist 0.30 loss 31.65
step  676/1000: dist 0.29 loss 32.83
step  677/1000: dist 0.30 loss 25.51
step  678/1000: dist 0.30 loss 10.54
step  679/1000: dist 0.30 loss 1.28
step  680/1000: dist 0.30 loss 5.72
step  681/1000: dist 0.30 loss 13.81
step  682/1000: dist 0.30 loss 14.14
step  683/1000: dist 0.29 loss 6.32
step  684/1000: dist 0.30 loss 1.63
step  685/1000: dist 0.29 loss 7.07
step  686/1000: dist 0.30 loss 14.34
step  687/1000: dist 0.29 loss 17.52
step  688/1000: dist 0.29 loss 21.50
step  689/1000: dist 0.30 loss 23.34
step  690/1000: dist 0.30 loss 13.72
step  691/1000: dist 0.29 loss 4.13
step  692/1000: dist 0.30 loss 7.95
step  693/1000: dist 0.29 loss 13.76
step  694/1000: dist 0.29 loss 8.19
step  695/1000: dist 0.29 loss 2.99
step  696/1000: dist 0.29 loss 7.16
step  697/1000: dist 0.29 loss 7.80
step  698/1000: dist 0.29 loss 3.11
step  699/1000: dist 0.29 loss 4.45
step  700/1000: dist 0.29 loss 6.49
step  701/1000: dist 0.29 loss 3.36
step  702/1000: dist 0.29 loss 4.18
step  703/1000: dist 0.29 loss 8.17
step  704/1000: dist 0.29 loss 8.08
step  705/1000: dist 0.29 loss 10.52
step  706/1000: dist 0.29 loss 17.43
step  707/1000: dist 0.29 loss 18.52
step  708/1000: dist 0.29 loss 12.93
step  709/1000: dist 0.29 loss 7.10
```

```
step  710/1000: dist 0.29 loss 2.64
step  711/1000: dist 0.29 loss 2.74
step  712/1000: dist 0.29 loss 7.96
step  713/1000: dist 0.29 loss 9.53
step  714/1000: dist 0.29 loss 4.87
step  715/1000: dist 0.29 loss 2.97
step  716/1000: dist 0.29 loss 7.12
step  717/1000: dist 0.29 loss 12.72
step  718/1000: dist 0.29 loss 17.87
step  719/1000: dist 0.29 loss 22.91
step  720/1000: dist 0.29 loss 26.68
step  721/1000: dist 0.29 loss 23.46
step  722/1000: dist 0.29 loss 13.59
step  723/1000: dist 0.29 loss 9.42
step  724/1000: dist 0.29 loss 19.38
step  725/1000: dist 0.29 loss 30.52
step  726/1000: dist 0.29 loss 26.76
step  727/1000: dist 0.29 loss 13.86
step  728/1000: dist 0.29 loss 7.70
step  729/1000: dist 0.29 loss 8.01
step  730/1000: dist 0.29 loss 8.24
step  731/1000: dist 0.29 loss 9.47
step  732/1000: dist 0.29 loss 10.06
step  733/1000: dist 0.29 loss 5.50
step  734/1000: dist 0.29 loss 2.26
step  735/1000: dist 0.29 loss 6.17
step  736/1000: dist 0.29 loss 7.93
step  737/1000: dist 0.29 loss 2.99
step  738/1000: dist 0.29 loss 2.02
step  739/1000: dist 0.28 loss 6.60
step  740/1000: dist 0.29 loss 7.20
step  741/1000: dist 0.29 loss 6.12
step  742/1000: dist 0.29 loss 11.46
step  743/1000: dist 0.29 loss 19.38
step  744/1000: dist 0.29 loss 23.53
step  745/1000: dist 0.29 loss 22.60
step  746/1000: dist 0.29 loss 14.87
step  747/1000: dist 0.29 loss 4.25
step  748/1000: dist 0.29 loss 2.15
step  749/1000: dist 0.29 loss 8.85
step  750/1000: dist 0.29 loss 11.81
step  751/1000: dist 0.29 loss 5.97
step  752/1000: dist 0.29 loss 1.22
step  753/1000: dist 0.29 loss 4.28
step  754/1000: dist 0.28 loss 7.59
step  755/1000: dist 0.28 loss 4.51
step  756/1000: dist 0.28 loss 1.65
step  757/1000: dist 0.28 loss 4.99
```

```
step  758/1000: dist 0.28 loss 8.46
step  759/1000: dist 0.28 loss 8.44
step  760/1000: dist 0.28 loss 12.02
step  761/1000: dist 0.29 loss 20.58
step  762/1000: dist 0.28 loss 25.26
step  763/1000: dist 0.28 loss 21.08
step  764/1000: dist 0.28 loss 13.27
step  765/1000: dist 0.29 loss 8.67
step  766/1000: dist 0.29 loss 12.30
step  767/1000: dist 0.28 loss 21.45
step  768/1000: dist 0.28 loss 24.26
step  769/1000: dist 0.28 loss 15.15
step  770/1000: dist 0.28 loss 6.14
step  771/1000: dist 0.29 loss 6.03
step  772/1000: dist 0.29 loss 7.68
step  773/1000: dist 0.29 loss 6.63
step  774/1000: dist 0.28 loss 7.80
step  775/1000: dist 0.29 loss 8.91
step  776/1000: dist 0.28 loss 5.02
step  777/1000: dist 0.29 loss 2.89
step  778/1000: dist 0.28 loss 7.04
step  779/1000: dist 0.29 loss 8.93
step  780/1000: dist 0.29 loss 5.14
step  781/1000: dist 0.28 loss 3.72
step  782/1000: dist 0.28 loss 5.88
step  783/1000: dist 0.28 loss 5.14
step  784/1000: dist 0.28 loss 2.25
step  785/1000: dist 0.29 loss 1.55
step  786/1000: dist 0.28 loss 1.93
step  787/1000: dist 0.29 loss 1.91
step  788/1000: dist 0.29 loss 2.08
step  789/1000: dist 0.29 loss 1.93
step  790/1000: dist 0.29 loss 1.67
step  791/1000: dist 0.28 loss 1.87
step  792/1000: dist 0.29 loss 1.28
step  793/1000: dist 0.28 loss 0.43
step  794/1000: dist 0.29 loss 1.03
step  795/1000: dist 0.28 loss 1.52
step  796/1000: dist 0.29 loss 0.84
step  797/1000: dist 0.28 loss 0.70
step  798/1000: dist 0.29 loss 0.97
step  799/1000: dist 0.29 loss 0.59
step  800/1000: dist 0.28 loss 0.49
step  801/1000: dist 0.28 loss 0.82
step  802/1000: dist 0.28 loss 0.70
step  803/1000: dist 0.28 loss 0.46
step  804/1000: dist 0.28 loss 0.46
step  805/1000: dist 0.28 loss 0.51
```

```
step  806/1000: dist 0.28 loss 0.55
step  807/1000: dist 0.28 loss 0.45
step  808/1000: dist 0.28 loss 0.39
step  809/1000: dist 0.28 loss 0.45
step  810/1000: dist 0.28 loss 0.38
step  811/1000: dist 0.28 loss 0.39
step  812/1000: dist 0.28 loss 0.44
step  813/1000: dist 0.28 loss 0.32
step  814/1000: dist 0.28 loss 0.33
step  815/1000: dist 0.28 loss 0.41
step  816/1000: dist 0.28 loss 0.32
step  817/1000: dist 0.28 loss 0.31
step  818/1000: dist 0.28 loss 0.35
step  819/1000: dist 0.28 loss 0.32
step  820/1000: dist 0.28 loss 0.32
step  821/1000: dist 0.28 loss 0.32
step  822/1000: dist 0.28 loss 0.30
step  823/1000: dist 0.28 loss 0.32
step  824/1000: dist 0.28 loss 0.30
step  825/1000: dist 0.28 loss 0.30
step  826/1000: dist 0.28 loss 0.30
step  827/1000: dist 0.28 loss 0.30
step  828/1000: dist 0.28 loss 0.29
step  829/1000: dist 0.28 loss 0.29
step  830/1000: dist 0.28 loss 0.30
step  831/1000: dist 0.28 loss 0.29
step  832/1000: dist 0.28 loss 0.29
step  833/1000: dist 0.28 loss 0.29
step  834/1000: dist 0.28 loss 0.28
step  835/1000: dist 0.28 loss 0.29
step  836/1000: dist 0.28 loss 0.28
step  837/1000: dist 0.28 loss 0.29
step  838/1000: dist 0.28 loss 0.29
step  839/1000: dist 0.28 loss 0.28
step  840/1000: dist 0.28 loss 0.29
step  841/1000: dist 0.28 loss 0.28
step  842/1000: dist 0.28 loss 0.28
step  843/1000: dist 0.28 loss 0.28
step  844/1000: dist 0.28 loss 0.29
step  845/1000: dist 0.28 loss 0.29
step  846/1000: dist 0.28 loss 0.28
step  847/1000: dist 0.28 loss 0.28
step  848/1000: dist 0.28 loss 0.28
step  849/1000: dist 0.28 loss 0.28
step  850/1000: dist 0.28 loss 0.28
step  851/1000: dist 0.28 loss 0.28
step  852/1000: dist 0.28 loss 0.28
step  853/1000: dist 0.28 loss 0.28
```

```
step  854/1000: dist 0.28 loss 0.28
step  855/1000: dist 0.28 loss 0.28
step  856/1000: dist 0.28 loss 0.28
step  857/1000: dist 0.28 loss 0.28
step  858/1000: dist 0.28 loss 0.28
step  859/1000: dist 0.28 loss 0.28
step  860/1000: dist 0.28 loss 0.28
step  861/1000: dist 0.28 loss 0.28
step  862/1000: dist 0.28 loss 0.28
step  863/1000: dist 0.28 loss 0.28
step  864/1000: dist 0.28 loss 0.28
step  865/1000: dist 0.28 loss 0.28
step  866/1000: dist 0.28 loss 0.28
step  867/1000: dist 0.28 loss 0.28
step  868/1000: dist 0.28 loss 0.28
step  869/1000: dist 0.28 loss 0.28
step  870/1000: dist 0.28 loss 0.28
step  871/1000: dist 0.28 loss 0.28
step  872/1000: dist 0.28 loss 0.28
step  873/1000: dist 0.28 loss 0.28
step  874/1000: dist 0.28 loss 0.28
step  875/1000: dist 0.28 loss 0.28
step  876/1000: dist 0.28 loss 0.28
step  877/1000: dist 0.28 loss 0.28
step  878/1000: dist 0.28 loss 0.28
step  879/1000: dist 0.28 loss 0.28
step  880/1000: dist 0.28 loss 0.28
step  881/1000: dist 0.28 loss 0.28
step  882/1000: dist 0.28 loss 0.28
step  883/1000: dist 0.28 loss 0.28
step  884/1000: dist 0.28 loss 0.28
step  885/1000: dist 0.28 loss 0.28
step  886/1000: dist 0.28 loss 0.28
step  887/1000: dist 0.28 loss 0.28
step  888/1000: dist 0.28 loss 0.28
step  889/1000: dist 0.28 loss 0.28
step  890/1000: dist 0.28 loss 0.28
step  891/1000: dist 0.28 loss 0.28
step  892/1000: dist 0.28 loss 0.28
step  893/1000: dist 0.28 loss 0.28
step  894/1000: dist 0.28 loss 0.28
step  895/1000: dist 0.28 loss 0.28
step  896/1000: dist 0.28 loss 0.28
step  897/1000: dist 0.28 loss 0.28
step  898/1000: dist 0.28 loss 0.28
step  899/1000: dist 0.28 loss 0.28
step  900/1000: dist 0.28 loss 0.28
step  901/1000: dist 0.28 loss 0.28
```

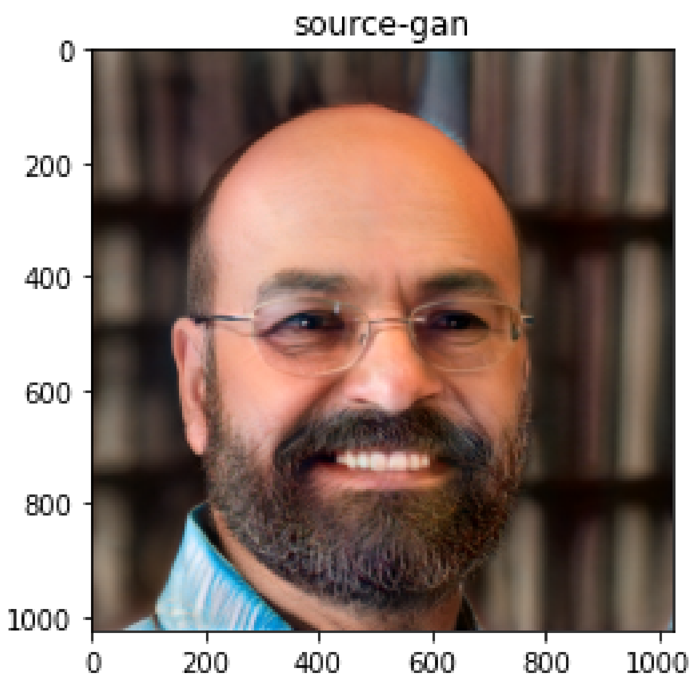
```
step  902/1000: dist 0.28 loss 0.28
step  903/1000: dist 0.27 loss 0.27
step  904/1000: dist 0.27 loss 0.27
step  905/1000: dist 0.27 loss 0.27
step  906/1000: dist 0.28 loss 0.28
step  907/1000: dist 0.28 loss 0.28
step  908/1000: dist 0.27 loss 0.27
step  909/1000: dist 0.27 loss 0.27
step  910/1000: dist 0.27 loss 0.27
step  911/1000: dist 0.27 loss 0.27
step  912/1000: dist 0.27 loss 0.27
step  913/1000: dist 0.27 loss 0.27
step  914/1000: dist 0.27 loss 0.27
step  915/1000: dist 0.27 loss 0.27
step  916/1000: dist 0.27 loss 0.27
step  917/1000: dist 0.27 loss 0.27
step  918/1000: dist 0.27 loss 0.27
step  919/1000: dist 0.27 loss 0.27
step  920/1000: dist 0.27 loss 0.27
step  921/1000: dist 0.27 loss 0.27
step  922/1000: dist 0.27 loss 0.27
step  923/1000: dist 0.27 loss 0.27
step  924/1000: dist 0.27 loss 0.27
step  925/1000: dist 0.27 loss 0.27
step  926/1000: dist 0.27 loss 0.27
step  927/1000: dist 0.27 loss 0.27
step  928/1000: dist 0.27 loss 0.27
step  929/1000: dist 0.27 loss 0.27
step  930/1000: dist 0.27 loss 0.27
step  931/1000: dist 0.27 loss 0.27
step  932/1000: dist 0.27 loss 0.27
step  933/1000: dist 0.27 loss 0.27
step  934/1000: dist 0.27 loss 0.27
step  935/1000: dist 0.27 loss 0.27
step  936/1000: dist 0.27 loss 0.27
step  937/1000: dist 0.27 loss 0.27
step  938/1000: dist 0.27 loss 0.27
step  939/1000: dist 0.27 loss 0.27
step  940/1000: dist 0.27 loss 0.27
step  941/1000: dist 0.27 loss 0.27
step  942/1000: dist 0.27 loss 0.27
step  943/1000: dist 0.27 loss 0.27
step  944/1000: dist 0.27 loss 0.27
step  945/1000: dist 0.27 loss 0.27
step  946/1000: dist 0.27 loss 0.27
step  947/1000: dist 0.27 loss 0.27
step  948/1000: dist 0.27 loss 0.27
step  949/1000: dist 0.27 loss 0.27
```

```
step  950/1000: dist 0.27 loss 0.27
step  951/1000: dist 0.27 loss 0.27
step  952/1000: dist 0.27 loss 0.27
step  953/1000: dist 0.27 loss 0.27
step  954/1000: dist 0.27 loss 0.27
step  955/1000: dist 0.27 loss 0.27
step  956/1000: dist 0.27 loss 0.27
step  957/1000: dist 0.27 loss 0.27
step  958/1000: dist 0.27 loss 0.27
step  959/1000: dist 0.27 loss 0.27
step  960/1000: dist 0.27 loss 0.27
step  961/1000: dist 0.27 loss 0.27
step  962/1000: dist 0.27 loss 0.27
step  963/1000: dist 0.27 loss 0.27
step  964/1000: dist 0.27 loss 0.27
step  965/1000: dist 0.27 loss 0.27
step  966/1000: dist 0.27 loss 0.27
step  967/1000: dist 0.27 loss 0.27
step  968/1000: dist 0.27 loss 0.27
step  969/1000: dist 0.27 loss 0.27
step  970/1000: dist 0.27 loss 0.27
step  971/1000: dist 0.27 loss 0.27
step  972/1000: dist 0.27 loss 0.27
step  973/1000: dist 0.27 loss 0.27
step  974/1000: dist 0.27 loss 0.27
step  975/1000: dist 0.27 loss 0.27
step  976/1000: dist 0.27 loss 0.27
step  977/1000: dist 0.27 loss 0.27
step  978/1000: dist 0.27 loss 0.27
step  979/1000: dist 0.27 loss 0.27
step  980/1000: dist 0.27 loss 0.27
step  981/1000: dist 0.27 loss 0.27
step  982/1000: dist 0.27 loss 0.27
step  983/1000: dist 0.27 loss 0.27
step  984/1000: dist 0.27 loss 0.27
step  985/1000: dist 0.27 loss 0.27
step  986/1000: dist 0.27 loss 0.27
step  987/1000: dist 0.27 loss 0.27
step  988/1000: dist 0.27 loss 0.27
step  989/1000: dist 0.27 loss 0.27
step  990/1000: dist 0.27 loss 0.27
step  991/1000: dist 0.27 loss 0.27
step  992/1000: dist 0.27 loss 0.27
step  993/1000: dist 0.27 loss 0.27
step  994/1000: dist 0.27 loss 0.27
step  995/1000: dist 0.27 loss 0.27
step  996/1000: dist 0.27 loss 0.27
step  997/1000: dist 0.27 loss 0.27
```

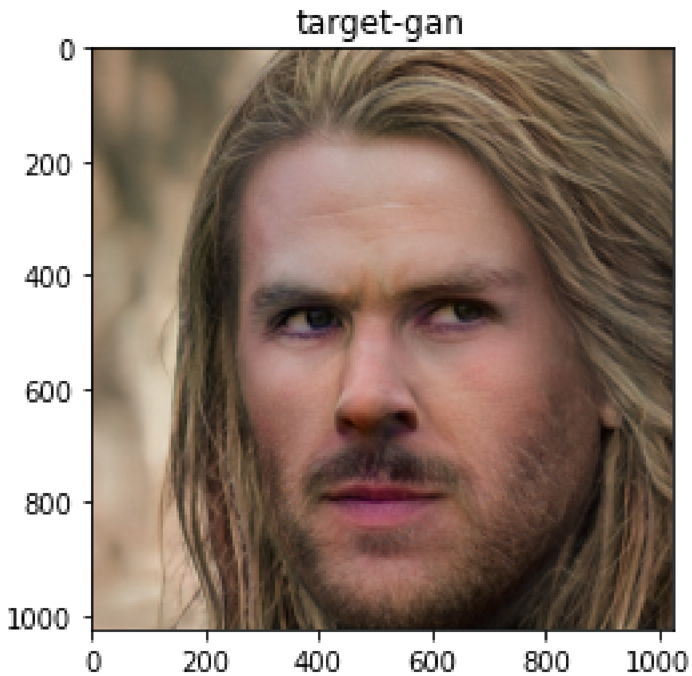
```
step  998/1000: dist 0.27 loss 0.27
step  999/1000: dist 0.27 loss 0.27
step 1000/1000: dist 0.27 loss 0.27
Elapsed: 84.2 s
```

With the conversion complete, lets have a look at the two GANs.

[ ]:
```
img_gan_source = cv2.imread('/content/out_source/proj.png')
img = cv2.cvtColor(img_gan_source, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('source-gan')
plt.show()
```



[ ]:
```
img_gan_target = cv2.imread('/content/out_target/proj.png')
img = cv2.cvtColor(img_gan_target, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('target-gan')
plt.show()
```



As you can see, the two GAN-generated images look similar to their real-world counterparts. However, they are by no means exact replicas.

4.7 Build the Video

The following code builds a transition video between the two latent vectors previously obtained.

[ ]:
```
# HIDE OUTPUT
import torch
import dnnlib
import legacy
import PIL.Image
import numpy as np
import imageio
from tqdm.notebook import tqdm

lvec1 = np.load('/content/out_source/projected_w.npz')['w']
lvec2 = np.load('/content/out_target/projected_w.npz')['w']

network_pkl = "https://nvlabs-fi-cdn.nvidia.com/stylegan2"\
  "-ada-pytorch/pretrained/ffhq.pkl"
device = torch.device('cuda')
with dnnlib.util.open_url(network_pkl) as fp:
    G = legacy.load_network_pkl(fp)['G_ema']\
      .requires_grad_(False).to(device)
```

```
diff = lvec2 - lvec1
step = diff / STEPS
current = lvec1.copy()
target_uint8 = np.array([1024,1024,3], dtype=np.uint8)

video = imageio.get_writer('/content/movie.mp4', mode='I', fps=FPS,
                           codec='libx264', bitrate='16M')

for j in tqdm(range(STEPS)):
  z = torch.from_numpy(current).to(device)
  synth_image = G.synthesis(z, noise_mode='const')
  synth_image = (synth_image + 1) * (255/2)
  synth_image = synth_image.permute(0, 2, 3, 1).clamp(0, 255)\
    .to(torch.uint8)[0].cpu().numpy()

  repeat = FREEZE_STEPS if j==0 or j==(STEPS-1) else 1

  for i in range(repeat):
    video.append_data(synth_image)
  current = current + step


video.close()
```

```
  0%|          | 0/150 [00:00<?, ?it/s]

Setting up PyTorch plugin "bias_act_plugin"… Done.
Setting up PyTorch plugin "upfirdn2d_plugin"… Done.
```

4.8 Download your Video

If you made it through all of these steps, you are now ready to download your video.

[ ]:
```
# HIDE OUTPUT
from google.colab import files
files.download("movie.mp4")
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Open in Colab

T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 9 Material

- Part 9.1: Introduction to Keras Transfer Learning [Video] [Notebook]
- Part 9.2: Keras Transfer Learning for Computer Vision [Video] [Notebook]
- Part 9.3: Transfer Learning for NLP with Keras [Video] [Notebook]
- Part 9.4: Transfer Learning for Facial Feature Recognition [Video] [Notebook]
- **Part 9.5: Transfer Learning for Style Transfer** [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

Part 9.5: Transfer Learning for Keras Style Transfer

In this part, we will implement style transfer. This technique takes two images as input and produces a third. The first image is the base image that we wish to transform. The second image represents the style we want to apply to the source image. Finally, the algorithm renders a third image that emulates the style characterized by the style image. This technique is called style transfer. [Cite:gatys2016image]

Figure 9.STYLE_TRANS: Style Transfer

I based the code presented in this part on a style transfer example in the Keras documentation created by François Chollet.

We begin by uploading two images to Colab. If running this code locally, point these two filenames at the local copies of the images you wish to use.

- **base_image_path** - The image to apply the style to.
- **style_reference_image_path** - The image whose style we wish to copy.

First, we upload the base image.

In [2]:
```
# HIDE OUTPUT
import os
from google.colab import files

uploaded = files.upload()

if len(uploaded) != 1:
  print("Upload exactly 1 file for source.")
else:
  for k, v in uploaded.items():
    _, ext = os.path.splitext(k)
    os.remove(k)
    base_image_path = f"source{ext}"
    open(base_image_path, 'wb').write(v)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving brooking-crop.jpg to brooking-crop.jpg
```

We also, upload the style image.

In [3]:
```
# HIDE OUTPUT
uploaded = files.upload()

if len(uploaded) != 1:
  print("Upload exactly 1 file for target.")
else:
  for k, v in uploaded.items():
```

```
    _, ext = os.path.splitext(k)
    os.remove(k)
    style_reference_image_path = f"style{ext}"
    open(style_reference_image_path, 'wb').write(v)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving van-gogh-crop.jpg to van-gogh-crop.jpg
```

The loss function balances three different goals defined by the following three weights. Changing these weights allows you to fine-tune the image generation.

- **total_variation_weight** - How much emphasis to place on the visual coherence of nearby pixels.
- **style_weight** - How much emphasis to place on emulating the style of the reference image.
- **content_weight** - How much emphasis to place on remaining close in appearance to the base image.

In [4]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import vgg19

result_prefix = "generated"

# Weights of the different loss components
total_variation_weight = 1e-6
style_weight = 1e-6
content_weight = 2.5e-8

# Dimensions of the generated picture.
width, height = keras.preprocessing.image.load_img(base_image_path).size
img_nrows = 400
img_ncols = int(width * img_nrows / height)
```

We now display the two images we will use, first the base image followed by the style image.

In [5]:
```
from IPython.display import Image, display

print("Source Image")
display(Image(base_image_path))
```

```
Source Image
```

In [6]:
```
print("Style Image")
display(Image(style_reference_image_path))
```

```
Style Image
```

Image Preprocessing and Postprocessing

The preprocess_image function begins by loading the image using Keras. We scale the image to the size specified by img_nrows and img_ncols. The img_to_array converts the image to a Numpy array, to which we add dimension to account for batching. The dimensions expected by VGG are colors depth, height, width, and batch. Finally, we convert the Numpy array to a tensor.

The deprocess_image performs the reverse, transforming the output of the style transfer process back to a regular image. First, we reshape the image to remove the batch dimension. Next, The outputs are moved back into the 0-255 range by adding the mean value of the RGB colors. We must also convert the BGR (blue, green, red) colorspace of VGG to the more standard RGB encoding.

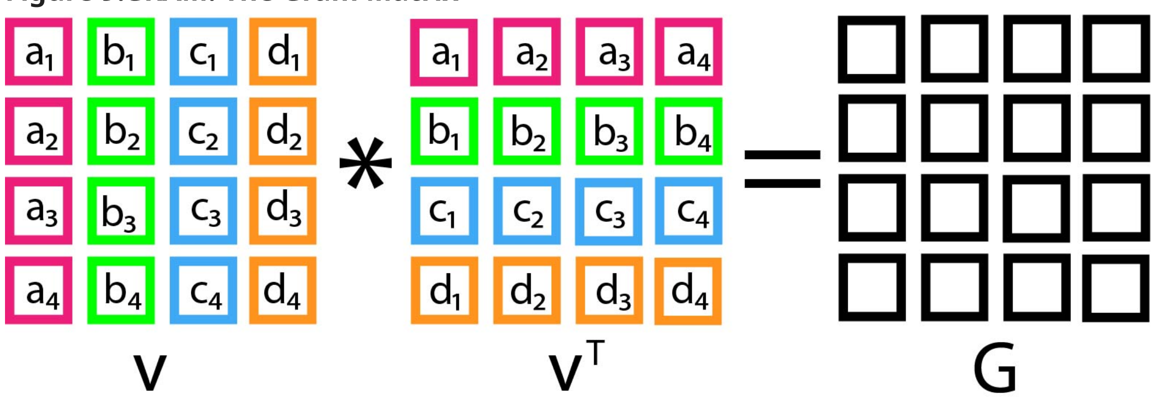
In [7]:
```
def preprocess_image(image_path):
    # Util function to open, resize and format
    # pictures into appropriate tensors
    img = keras.preprocessing.image.load_img(
        image_path, target_size=(img_nrows, img_ncols)
    )
    img = keras.preprocessing.image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
```

```
    img = vgg19.preprocess_input(img)
    return tf.convert_to_tensor(img)


def deprocess_image(x):
    # Util function to convert a tensor into a valid image
    x = x.reshape((img_nrows, img_ncols, 3))
    # Remove zero-center by mean pixel
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype("uint8")
    return x
```

Calculating the Style, Content, and Variation Loss

Before we see how to calculate the 3-part loss function, I must introduce the Gram matrix's mathematical concept. Figure 9.GRAM demonstrates this concept.

Figure 9.GRAM: The Gram Matrix



We calculate the Gram matrix by multiplying a matrix by its transpose. To calculate two parts of the loss function, we will take the Gram matrix of the outputs from several convolution layers in the VGG network. To determine both style, and similarity to the original image, we will compare the convolution layer output of VGG rather than directly comparing the image pixels. In the third part of the loss function, we will directly compare pixels near each other.

Because we are taking convolution output from several different levels of the VGG network, the Gram matrix provides a means of combining these layers. The Gram matrix of the VGG convolution layers represents the style of the image. We will calculate this style for the original image, the style-reference image, and the final output image as the algorithm generates it.

In [8]:
```
# The gram matrix of an image tensor (feature-wise outer product)
def gram_matrix(x):
```

```
    x = tf.transpose(x, (2, 0, 1))
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, tf.transpose(features))
    return gram


# The "style loss" is designed to maintain
# the style of the reference image in the generated image.
# It is based on the gram matrices (which capture style) of
# feature maps from the style reference image
# and from the generated image
def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return tf.reduce_sum(tf.square(S - C)) /\
      (4.0 * (channels ** 2) * (size ** 2))


# An auxiliary loss function
# designed to maintain the "content" of the
# base image in the generated image
def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))


# The 3rd loss function, total variation loss,
# designed to keep the generated image locally coherent
def total_variation_loss(x):
    a = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] \
          - x[:, 1:, : img_ncols - 1, :]
    )
    b = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] \
          - x[:, : img_nrows - 1, 1:, :]
    )
    return tf.reduce_sum(tf.pow(a + b, 1.25))
```

The style_loss function compares how closely the current generated image (combination) matches the style of the reference style image. The Gram matrixes of the style and current generated image are subtracted and normalized to calculate this difference in style. Precisely, it consists in a sum of L2 distances between the Gram matrices of the representations of the base image and the style reference image, extracted from different layers of VGG. The general idea is to capture color/texture information at different spatial scales (fairly large scales, as defined by the depth of the layer considered).

The content_loss function compares how closely the current generated image matches the original image. You must subtract Gram matrixes of the original and generated images to calculate this difference. Here we calculate the L2 distance between the base image's VGG features and the generated image's features, keeping the generated image close enough to the original one.

Finally, the total_variation_loss function imposes local spatial continuity between the pixels of the generated image, giving it visual coherence.

The VGG Neural Network

VGG19 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman. [Cite:simonyan2014very] The model achieves 92.7% top-5 test accuracy in ImageNet, a dataset of over 14 million images belonging to 1000 classes. We will transfer the VGG16 weights into our style transfer model. Keras provides functions to load the VGG neural network.

In [9]:
```
# HIDE OUTPUT
# Build a VGG19 model loaded with pre-trained ImageNet weights
model = vgg19.VGG19(weights="imagenet", include_top=False)

# Get the symbolic outputs of each "key" layer (we gave them unique names).
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])

# Set up a model that returns the activation values for every layer in
# VGG19 (as a dict).
feature_extractor = keras.Model(inputs=model.inputs, outputs=outputs_dict)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
```

We can now generate the complete loss function. The following images are input to the compute_loss function:

- **combination_image** - The current iteration of the generated image.
- **base_image** - The starting image.
- **style_reference_image** - The image that holds the style to reproduce.

The layers specified by style_layer_names indicate which layers should be extracted as features from VGG for each of the three images.

In [10]:
```
# List of layers to use for the style loss.
style_layer_names = [
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1",
]
# The layer to use for the content loss.
content_layer_name = "block5_conv2"
```

```
def compute_loss(combination_image, base_image, style_reference_image):
    input_tensor = tf.concat(
        [base_image, style_reference_image, combination_image], axis=0
    )
    features = feature_extractor(input_tensor)

    # Initialize the loss
    loss = tf.zeros(shape=())

    # Add content loss
    layer_features = features[content_layer_name]
    base_image_features = layer_features[0, :, :, :]
    combination_features = layer_features[2, :, :, :]
    loss = loss + content_weight * content_loss(
        base_image_features, combination_features
    )
    # Add style loss
    for layer_name in style_layer_names:
        layer_features = features[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_reference_features, combination_features)
        loss += (style_weight / len(style_layer_names)) * sl

    # Add total variation loss
    loss += total_variation_weight * \
      total_variation_loss(combination_image)
    return loss
```

Generating the Style Transferred Image

The compute_loss_and_grads function calls the loss function and computes the gradients. The parameters of this model are the actual RGB values of the current iteration of the generated images. These parameters start with the base image, and the algorithm optimizes them to the final rendered image. We are not training a model to perform the transformation; we are training/modifying the image to minimize the loss functions. We utilize gradient tape to allow Keras to modify the image in the same way the neural network training modifies weights.

In [11]:
```
@tf.function
def compute_loss_and_grads(combination_image, \
                  base_image, style_reference_image):
    with tf.GradientTape() as tape:
        loss = compute_loss(combination_image, \
                base_image, style_reference_image)
    grads = tape.gradient(loss, combination_image)
    return loss, grads
```

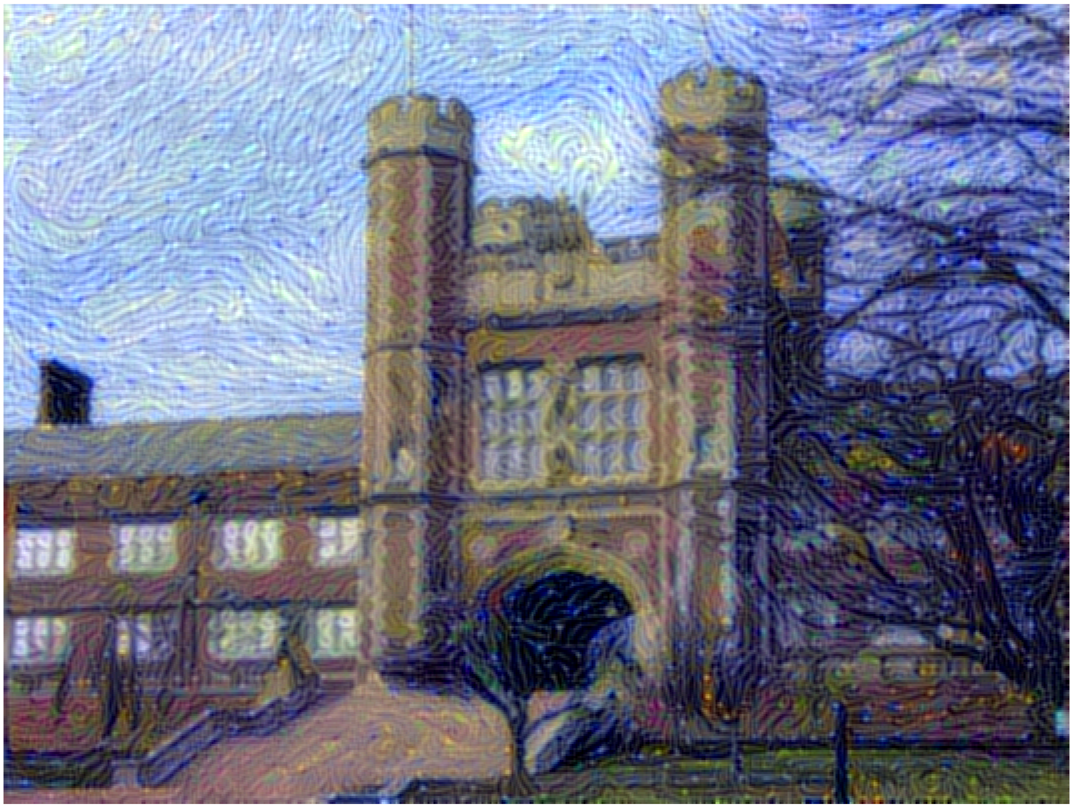
We can now optimize the image according to the loss function.

In [12]:
```
optimizer = keras.optimizers.SGD(
    keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=100.0, decay_steps=100, decay_rate=0.96
    )
)

base_image = preprocess_image(base_image_path)
style_reference_image = preprocess_image(style_reference_image_path)
combination_image = tf.Variable(preprocess_image(base_image_path))

iterations = 4000
for i in range(1, iterations + 1):
    loss, grads = compute_loss_and_grads(
        combination_image, base_image, style_reference_image
    )
    optimizer.apply_gradients([(grads, combination_image)])
    if i % 100 == 0:
        print("Iteration %d: loss=%.2f" % (i, loss))
        img = deprocess_image(combination_image.numpy())
        fname = result_prefix + "_at_iteration_%d.png" % i
        keras.preprocessing.image.save_img(fname, img)
```

```
Iteration 100: loss=4890.20
Iteration 200: loss=3527.19
Iteration 300: loss=3022.59
Iteration 400: loss=2751.59
Iteration 500: loss=2578.63
Iteration 600: loss=2457.19
Iteration 700: loss=2366.39
Iteration 800: loss=2295.66
Iteration 900: loss=2238.67
Iteration 1000: loss=2191.59
Iteration 1100: loss=2151.88
Iteration 1200: loss=2117.95
Iteration 1300: loss=2088.56
Iteration 1400: loss=2062.86
Iteration 1500: loss=2040.14
Iteration 1600: loss=2019.93
Iteration 1700: loss=2001.83
Iteration 1800: loss=1985.54
Iteration 1900: loss=1970.81
Iteration 2000: loss=1957.43
Iteration 2100: loss=1945.21
Iteration 2200: loss=1934.03
Iteration 2300: loss=1923.75
Iteration 2400: loss=1914.27
Iteration 2500: loss=1905.49
Iteration 2600: loss=1897.36
Iteration 2700: loss=1889.83
Iteration 2800: loss=1882.82
Iteration 2900: loss=1876.31
Iteration 3000: loss=1870.23
Iteration 3100: loss=1864.54
Iteration 3200: loss=1859.18
Iteration 3300: loss=1854.16
Iteration 3400: loss=1849.45
Iteration 3500: loss=1845.00
Iteration 3600: loss=1840.82
Iteration 3700: loss=1836.87
Iteration 3800: loss=1833.16
Iteration 3900: loss=1829.65
Iteration 4000: loss=1826.34
```

We can display the image.

In [13]:
```
display(Image(result_prefix + "_at_iteration_4000.png"))
```



We can download this image.

In [15]:
```
# HIDE OUTPUT
from google.colab import files
files.download(result_prefix + "_at_iteration_4000.png")
```