BST 261: Data Science II
Lecture 6
Convolutional Neural Networks (CNNs):
Data Augmentation and Pretrained networks

Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2021

Recipe of the Day!

[Rosemary crackers](#)

Pretrained Networks

Pretrained Networks

◎ Another way around having a small number of training examples to learn from is using networks that have been trained on other, bigger datasets similar to the type of data you have

◎ A **pretrained network** is a saved network that was previously trained on a large dataset

◎ If the dataset used to train the network is large enough and big enough, the features learned by the pretrained network can act as a generic model to use as a base for your network

◎ This saves an enormous amount of computing time

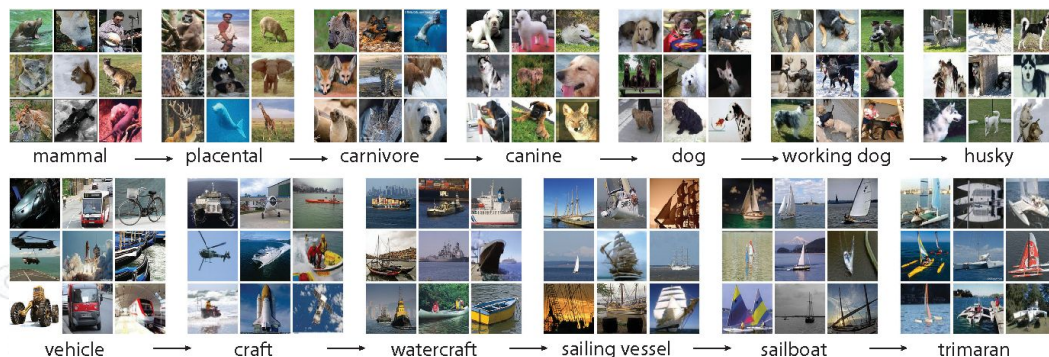◎ Pretrained networks can be used for **feature extraction** and **fine-tuning**

Pretrained Networks

◎ Commonly used pretrained networks include
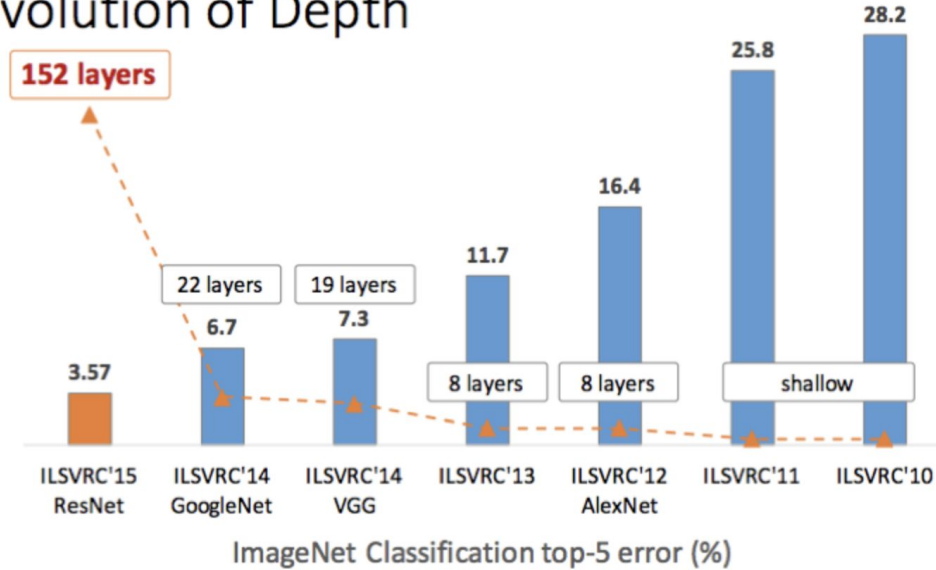- VGG16
- ResNet
- Inception
- Inception-ResNet
- Xception

◎ Commonly used dataset used to train a network is the ImageNet dataset
- 1.4 million labeled images
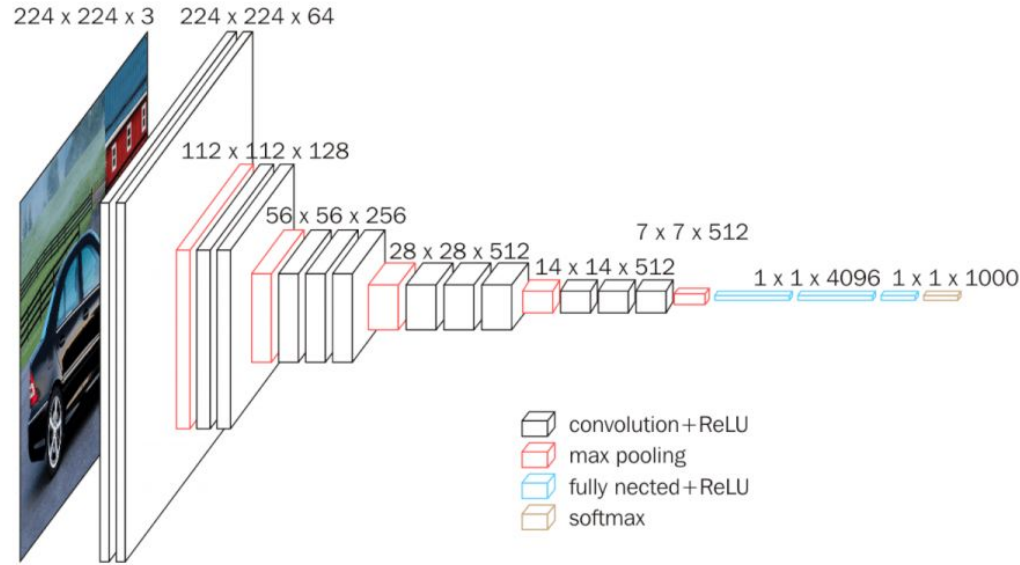- 1,000 different classes
- Mostly animals and everyday objects

mammal → placental → carnivore → canine → dog → working dog → husky

vehicle → craft → watercraft → sailing vessel → sailboat → trimaran

Pretrained Networks

Revolution of Depth

152 layers	22 layers	19 layers	8 layers	8 layers	shallow	
3.57	6.7	7.3	11.7	16.4	25.8	28.2
ILSVRC'15 ResNet	ILSVRC'14 GoogleNet	ILSVRC'14 VGG	ILSVRC'13	ILSVRC'12 AlexNet	ILSVRC'11	ILSVRC'10

ImageNet Classification top-5 error (%)

VGG-16

Input → Conv 1-1 | Conv 1-2 | Pooing | Conv 2-1 | Conv 2-2 | Pooing | Conv 3-1 | Conv 3-2 | Conv 3-3 | Pooing | Conv 4-1 | Conv 4-2 | Conv 4-3 | Pooing | Conv 5-1 | Conv 5-2 | Conv 5-3 | Pooing | Dense | Dense | Dense → Output

224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096   1 x 1 x 1000

- convolution+ReLU
- max pooling
- fully nected+ReLU
- softmax

7

Inception Models

(a) Inception module, naïve version
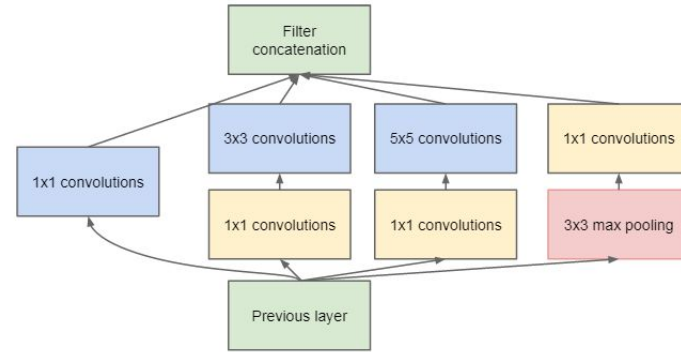
(b) Inception module with dimension reductions

Feature Extraction

◎ Consists of using the representations learned by a previous network to extract features from new samples

◎ These features are then run through a new classifier that is trained from scratch, and predictions are made
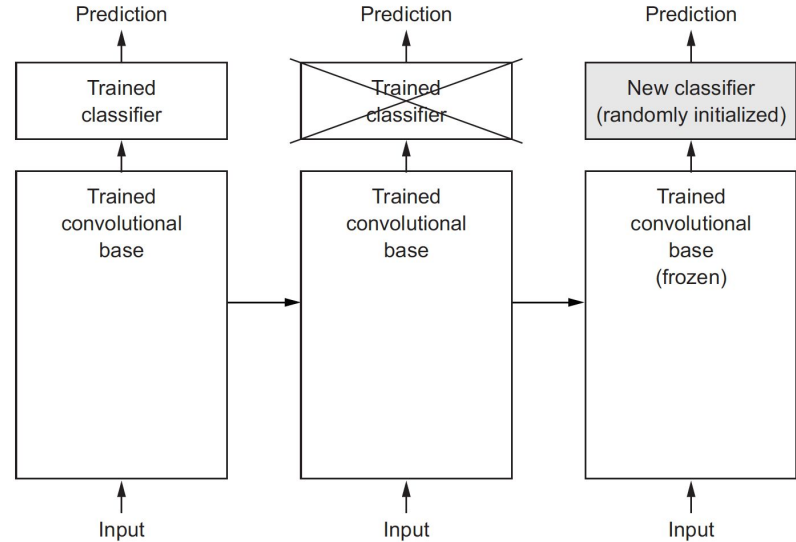
Feature Extraction

◎ For CNNs, the part of the pretrained network you use is called the **convolutional base**, which contains a series of convolution and pooling layers

◎ For feature extraction, you keep the convolutional base of the pretrained network, remove the dense / trained classifier layers, and append new dense and classifier layers to the convolutional base
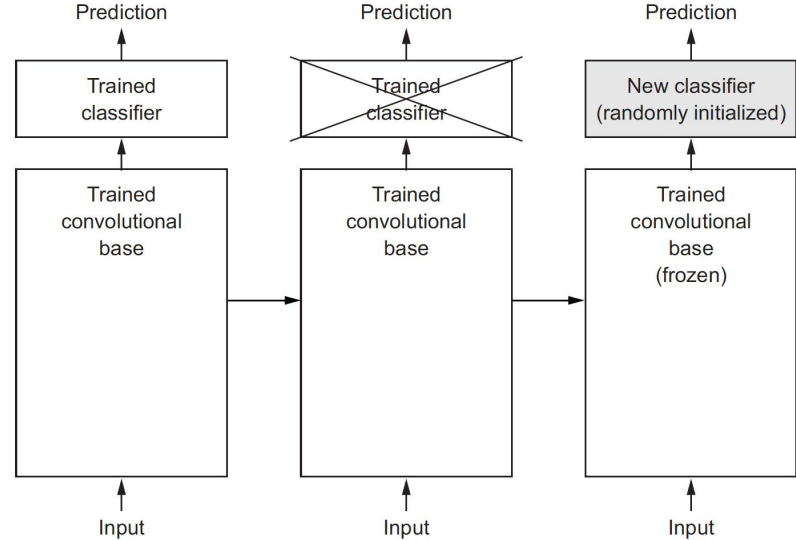
Feature Extraction

◎ We could also reuse the densely connected classifier as well, but this is not advised

◎ Representations learned by the convolutional base are likely to be more generic and thus more reusable

◎ The representations learned by the classifier will be specific to the set of classes the model was trained on

◎ They will also no longer contain information about where objects are located in the input image

  ○ This makes them especially useless when the object's location is important

Feature Extraction

◎ The level of generality depends on the depth of the layer in the model
  ○ Early layers extract local, highly generic features, i.e. edges, colors, textures
  ○ Later layers extract more abstract concepts i.e. "cat ear" or "dog eye"

◎ If your new dataset is very different from the dataset that was used to train the model, you should use only the first few layers for feature extraction rather than the entire base

Pretrained Networks in Keras

◎ Xception
◎ Inception V3
◎ ResNet50
◎ VGG16
◎ VGG19
◎ MobileNet

Instantiating the VGG16 Base

```
1 #from keras.applications import VGG16
2
3 conv_base = tf.keras.applications.VGG16(weights='imagenet',
4                                         include_top=False,
5                                         input_shape=(150, 150, 3))
```

```
conv_base.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

```
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Instantiating the VGG16 Base

```
1 #from keras.applications import VGG16
2
3 conv_base = tf.keras.applications.VGG16(weights='imagenet',
4                                         include_top=False,
5                                         input_shape=(150, 150, 3))
```

The final layer is a pooling layer and the final output shape from this base is (4, 4, 512). We need this information when adding layers to the base. This output shape will be the input shape for the densely connected layer we'll add to the base.

```
conv_base.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

```
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

15

Using a Pretrained Network

◎ The final output has shape (4, 4, 512)
◎ You have 2 options:
1. **Feature extraction without augmented data**: you can run the convolutional base over the dataset, record its output to a numpy array, and then use these values as input to a densely connected classifier
   ◎ This is fast and cheap to run
   ◎ It won't allow you to use augmented data
2. **Feature extraction with augmented data**: you can extend the convolutional base by adding dense layers on top and running the whole model on the input data
   ◎ This allows data augmentation
   ◎ This is very computationally expensive

Feature Extraction
without Augmented Data

```python
1  import os
2  import numpy as np
3  from keras.preprocessing.image import ImageDataGenerator
4
5  datagen = ImageDataGenerator(rescale=1./255)
6  batch_size = 20
7
8  def extract_features(directory, sample_count):
9      features = np.zeros(shape=(sample_count, 4, 4, 512))
10     labels = np.zeros(shape=(sample_count))
11     generator = datagen.flow_from_directory(
12         directory,
13         target_size=(150, 150),
14         batch_size=batch_size,
15         class_mode='binary')
16     i = 0
17     for inputs_batch, labels_batch in generator:
18         features_batch = conv_base.predict(inputs_batch)
19         features[i * batch_size : (i + 1) * batch_size] = features_batch
20         labels[i * batch_size : (i + 1) * batch_size] = labels_batch
21         i += 1
22         if i * batch_size >= sample_count:
23             # Note that since generators yield data indefinitely in a loop,
24             # we must `break` after every image has been seen once.
25             break
26     return features, labels
27
28 train_features, train_labels = extract_features(train_dir, 1609)
29 validation_features, validation_labels = extract_features(validation_dir, 426)
30 test_features, test_labels = extract_features(test_dir, 392)
```
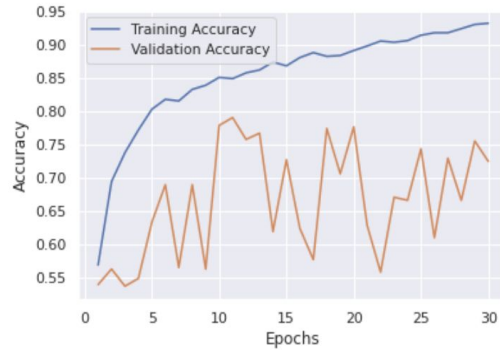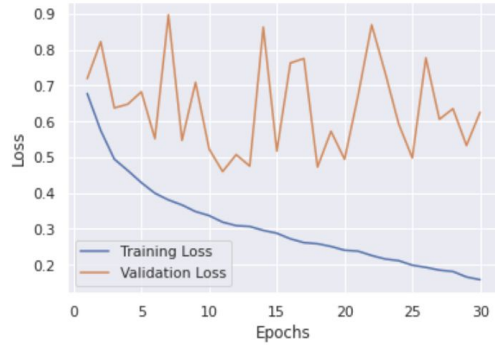
We need to reshape the outputs so we can feed them into a dense layer - recall that dense layers take in vectors.
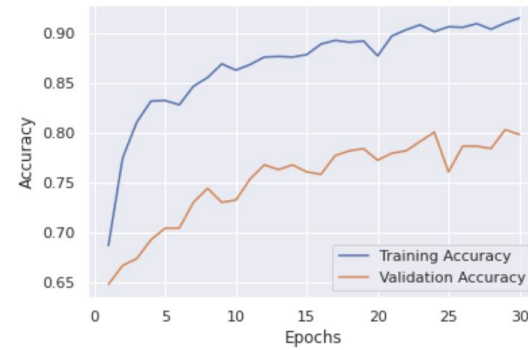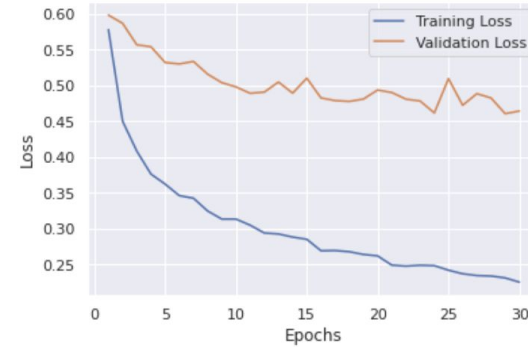
```python
1 train_features = np.reshape(train_features, (1609, 4 * 4 * 512))
2 validation_features = np.reshape(validation_features, (426, 4 * 4 * 512))
3 test_features = np.reshape(test_features, (392, 4 * 4 * 512))
```

```python
1 model = keras.Sequential([
2   layers.Dense(256, activation='relu', input_dim=4 * 4 * 512),
3   layers.Dropout(0.5),
4   layers.Dense(1, activation='sigmoid')
5 ])
6
7 model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=2e-5),
8                 loss='binary_crossentropy',
9                 metrics=['accuracy'])
10
11 history = model.fit(train_features, train_labels,
12                       epochs=30,
13                       batch_size=20,
14                       validation_data=(validation_features, validation_labels))
```

Original CNN made from scratch

CNN using pretrained base

Feature Extraction
with Augmented Data

```python
1 model = tf.keras.models.Sequential([
2   conv_base,
3   tf.keras.layers.Flatten(),
4   tf.keras.layers.Dense(256, activation='relu'),
5   tf.keras.layers.Dense(1, activation='sigmoid')
6 ])
```

```python
1 model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 256)	2097408
dense_3 (Dense)	(None, 1)	257

Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0

We can add the base just like a layer to our network

```python
1 model = tf.keras.models.Sequential([
2    conv_base,
3    tf.keras.layers.Flatten(),
4    tf.keras.layers.Dense(256, activation='relu'),
5    tf.keras.layers.Dense(1, activation='sigmoid')
6 ])
```

```python
1 model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 256)	2097408
dense_3 (Dense)	(None, 1)	257

Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0

```
1  from keras.preprocessing.image import ImageDataGenerator
2
3  train_datagen = ImageDataGenerator(
4          rescale=1./255,
5          rotation_range=40,
6          width_shift_range=0.2,
7          height_shift_range=0.2,
8          shear_range=0.2,
9          zoom_range=0.2,
10         horizontal_flip=True,
11         fill_mode='nearest')
12
13 # Note that the validation data should not be augmented!
14 test_datagen = ImageDataGenerator(rescale=1./255)
15
16 train_generator = train_datagen.flow_from_directory(
17         # This is the target directory
18         train_dir,
19         # All images will be resized to 150x150
20         target_size=(150, 150),
21         batch_size=20,
22         # Since we use binary_crossentropy loss, we need binary labels
23         class_mode='binary')
24
25 validation_generator = test_datagen.flow_from_directory(
26         validation_dir,
27         target_size=(150, 150),
28         batch_size=20,
29         class_mode='binary')
30
31 model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=2e-5),
32               loss='binary_crossentropy',
33               metrics=['accuracy'])
34
35
36 history = model.fit(
37         train_generator,
38         steps_per_epoch=81,
39         epochs=30,
40         validation_data=validation_generator,
41         validation_steps=22)
```

Note: do not run this code without access to a GPU.

Back to [Colab notebook](#)

Original CNN made from scratch with data augmentation
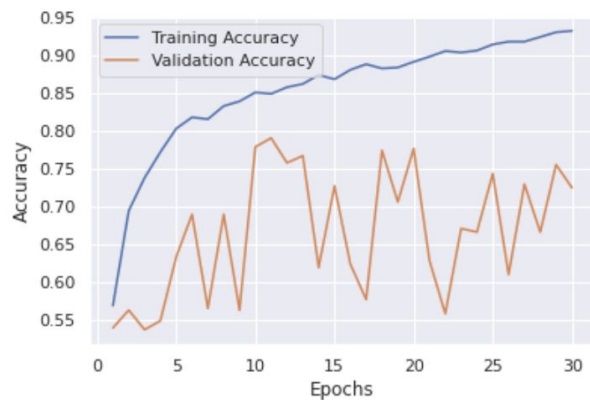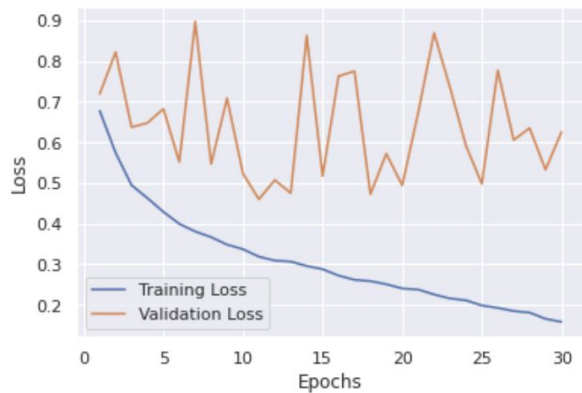
CNN using pretrained base with data augmentation