Introduction to deep learning for computer vision

Downloading the data

```
from google.colab import files
files.upload()
```

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 kaggle.json to kaggle.json
{'kaggle.json':
```

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!kaggle competitions download -c dogs-vs-cats
```

```
Downloading dogs-vs-cats.zip to /content
 97% 791M/812M [00:04<00:00, 221MB/s]
100% 812M/812M [00:04<00:00, 183MB/s]
```

```
!unzip -qq dogs-vs-cats.zip
```

```
!unzip -qq train.zip
```

Model - 1:

Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

Copying images to training, validation, and test directories

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2000)
```

Data preprocessing

Using `image_dataset_from_directory` to read images

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
```

```
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

```
import numpy as np
import tensorflow as tf
random_numbers = np.random.normal(size=(1000, 16))
dataset = tf.data.Dataset.from_tensor_slices(random_numbers)
```

```
for i, element in enumerate(dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(16,)
(16,)
(16,)
```

```
batched_dataset = dataset.batch(32)
for i, element in enumerate(batched_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(32, 16)
(32, 16)
(32, 16)
```

```
reshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))
for i, element in enumerate(reshaped_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(4, 4)
(4, 4)
(4, 4)
```

Displaying the shapes of the data and labels yielded by the `Dataset`

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

```
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
```

Building the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Configuring the model for training

```
model.compile(loss="binary_crossentropy",
              optimizer="Adam",
              metrics=["accuracy"])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dropout (Dropout)           (None, 12544)             0

 dense (Dense)               (None, 1)                 12545

=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Fitting the model using a `Dataset`

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 32/50
63/63 [==============================] - 5s 71ms/step - loss: 0.0187 - accuracy: 0.9955 - val_loss: 4.4358 - val_accuracy
Epoch 33/50
63/63 [==============================] - 5s 69ms/step - loss: 0.0371 - accuracy: 0.9925 - val_loss: 5.2315 - val_accuracy
Epoch 34/50
63/63 [==============================] - 5s 70ms/step - loss: 0.0544 - accuracy: 0.9895 - val_loss: 4.8552 - val_accuracy
Epoch 35/50
63/63 [==============================] - 5s 71ms/step - loss: 0.1080 - accuracy: 0.9860 - val_loss: 3.9726 - val_accuracy
Epoch 36/50
63/63 [==============================] - 5s 70ms/step - loss: 0.1298 - accuracy: 0.9770 - val_loss: 4.0043 - val_accuracy
Epoch 37/50
63/63 [==============================] - 5s 69ms/step - loss: 0.1399 - accuracy: 0.9835 - val_loss: 3.6744 - val_accuracy
Epoch 38/50
63/63 [==============================] - 5s 69ms/step - loss: 0.0391 - accuracy: 0.9895 - val_loss: 4.4135 - val_accuracy
Epoch 39/50
63/63 [==============================] - 5s 69ms/step - loss: 0.0284 - accuracy: 0.9930 - val_loss: 3.9840 - val_accuracy
Epoch 40/50
63/63 [==============================] - 5s 70ms/step - loss: 0.0379 - accuracy: 0.9930 - val_loss: 3.9981 - val_accuracy
Epoch 41/50
63/63 [==============================] - 5s 70ms/step - loss: 0.1033 - accuracy: 0.9845 - val_loss: 3.8504 - val_accuracy
Epoch 42/50
63/63 [==============================] - 5s 69ms/step - loss: 0.0706 - accuracy: 0.9915 - val_loss: 4.6001 - val_accuracy
Epoch 43/50
63/63 [==============================] - 5s 70ms/step - loss: 0.0502 - accuracy: 0.9915 - val_loss: 3.3963 - val_accuracy
Epoch 44/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0366 - accuracy: 0.9935 - val_loss: 3.3239 - val_accuracy
Epoch 45/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0106 - accuracy: 0.9975 - val_loss: 4.2456 - val_accuracy
Epoch 46/50
63/63 [==============================] - 5s 71ms/step - loss: 0.0101 - accuracy: 0.9985 - val_loss: 3.8701 - val_accuracy
Epoch 47/50
63/63 [==============================] - 5s 71ms/step - loss: 0.0222 - accuracy: 0.9960 - val_loss: 3.1984 - val_accuracy
Epoch 48/50
63/63 [==============================] - 5s 69ms/step - loss: 0.0184 - accuracy: 0.9970 - val_loss: 3.4274 - val_accuracy
Epoch 49/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0165 - accuracy: 0.9955 - val_loss: 3.7510 - val_accuracy
Epoch 50/50
63/63 [==============================] - 5s 70ms/step - loss: 0.0154 - accuracy: 0.9955 - val_loss: 4.0656 - val_accuracy
```

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

Training and validation accuracy

Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 34ms/step - loss: 3.5251 - accuracy: 0.7410
Test accuracy: 0.741
```

Model - 2:

Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch). What performance did you achieve?

Define a data augmentation stage to add to an image model

```
import os, shutil, pathlib

shutil.rmtree("./cats_vs_dogs_small_Q2", ignore_errors=True)

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1500)
make_subset("validation", start_index=1500, end_index=2000)
make_subset("test", start_index=2000, end_index=2500)


data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

Displaying some randomly augmented training images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

Defining a new convnet that includes image augmentation and dropout

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 39/50
63/63 [==============================] - 6s 93ms/step - loss: 0.3353 - accuracy: 0.8515 - val_loss: 0.5040 - val_accuracy
Epoch 40/50
63/63 [==============================] - 6s 94ms/step - loss: 0.3385 - accuracy: 0.8490 - val_loss: 0.6833 - val_accuracy
Epoch 41/50
63/63 [==============================] - 6s 91ms/step - loss: 0.3134 - accuracy: 0.8615 - val_loss: 0.4766 - val_accuracy
Epoch 42/50
63/63 [==============================] - 6s 93ms/step - loss: 0.3096 - accuracy: 0.8690 - val_loss: 0.3913 - val_accuracy
Epoch 43/50
63/63 [==============================] - 6s 93ms/step - loss: 0.3185 - accuracy: 0.8695 - val_loss: 0.4539 - val_accuracy
Epoch 44/50
63/63 [==============================] - 6s 93ms/step - loss: 0.3203 - accuracy: 0.8665 - val_loss: 0.4154 - val_accuracy
Epoch 45/50
63/63 [==============================] - 6s 94ms/step - loss: 0.3081 - accuracy: 0.8650 - val_loss: 0.4358 - val_accuracy
Epoch 46/50
63/63 [==============================] - 6s 91ms/step - loss: 0.2898 - accuracy: 0.8800 - val_loss: 0.5627 - val_accuracy
Epoch 47/50
63/63 [==============================] - 6s 93ms/step - loss: 0.2920 - accuracy: 0.8750 - val_loss: 0.4355 - val_accuracy
Epoch 48/50
63/63 [==============================] - 6s 94ms/step - loss: 0.2839 - accuracy: 0.8885 - val_loss: 0.5019 - val_accuracy
Epoch 49/50
63/63 [==============================] - 6s 94ms/step - loss: 0.2739 - accuracy: 0.8870 - val_loss: 0.6641 - val_accuracy
Epoch 50/50
63/63 [==============================] - 6s 94ms/step - loss: 0.2705 - accuracy: 0.8890 - val_loss: 0.5035 - val_accuracy
```

Evaluating the model on the test set

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

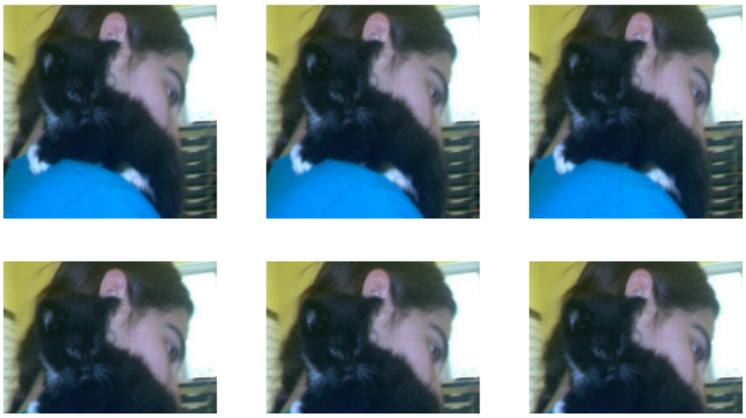
```
32/32 [==============================] - 1s 34ms/step - loss: 0.4643 - accuracy: 0.8120
Test accuracy: 0.812
```

Model - 3:

Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results.

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q3")


def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=2000)
make_subset("validation", start_index=2000, end_index=2500)
make_subset("test", start_index=2500, end_index=3000)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```



```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation1.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 39/50
63/63 [==============================] - 6s 95ms/step - loss: 0.2105 - accuracy: 0.9265 - val_loss: 0.5268 - val_accuracy
Epoch 40/50
63/63 [==============================] - 6s 91ms/step - loss: 0.2228 - accuracy: 0.9130 - val_loss: 0.7052 - val_accuracy
Epoch 41/50
63/63 [==============================] - 6s 92ms/step - loss: 0.1884 - accuracy: 0.9280 - val_loss: 0.8154 - val_accuracy
Epoch 42/50
63/63 [==============================] - 6s 93ms/step - loss: 0.2029 - accuracy: 0.9350 - val_loss: 0.7120 - val_accuracy
Epoch 43/50
63/63 [==============================] - 6s 93ms/step - loss: 0.1904 - accuracy: 0.9310 - val_loss: 0.7608 - val_accuracy
Epoch 44/50
63/63 [==============================] - 6s 93ms/step - loss: 0.1918 - accuracy: 0.9275 - val_loss: 0.6724 - val_accuracy
Epoch 45/50
63/63 [==============================] - 6s 93ms/step - loss: 0.1865 - accuracy: 0.9290 - val_loss: 0.6876 - val_accuracy
Epoch 46/50
63/63 [==============================] - 6s 94ms/step - loss: 0.2022 - accuracy: 0.9295 - val_loss: 0.8737 - val_accuracy
Epoch 47/50
63/63 [==============================] - 6s 94ms/step - loss: 0.1822 - accuracy: 0.9345 - val_loss: 0.9121 - val_accuracy
Epoch 48/50
63/63 [==============================] - 6s 93ms/step - loss: 0.1788 - accuracy: 0.9310 - val_loss: 0.6385 - val_accuracy
Epoch 49/50
63/63 [==============================] - 6s 96ms/step - loss: 0.1904 - accuracy: 0.9320 - val_loss: 1.4272 - val_accuracy
Epoch 50/50
63/63 [==============================] - 6s 93ms/step - loss: 0.1965 - accuracy: 0.9390 - val_loss: 0.8236 - val_accuracy
```

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation1.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 53ms/step - loss: 0.5215 - accuracy: 0.8260
Test accuracy: 0.826
```

Model - 4:

Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use any and all optimization techniques to get best performance

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 180, 180, 3)]     0

 block1_conv1 (Conv2D)       (None, 180, 180, 64)      1792

 block1_conv2 (Conv2D)       (None, 180, 180, 64)      36928

 block1_pool (MaxPooling2D)  (None, 90, 90, 64)        0

 block2_conv1 (Conv2D)       (None, 90, 90, 128)       73856

 block2_conv2 (Conv2D)       (None, 90, 90, 128)       147584

 block2_pool (MaxPooling2D)  (None, 45, 45, 128)       0

 block3_conv1 (Conv2D)       (None, 45, 45, 256)       295168

 block3_conv2 (Conv2D)       (None, 45, 45, 256)       590080

 block3_conv3 (Conv2D)       (None, 45, 45, 256)       590080

 block3_pool (MaxPooling2D)  (None, 22, 22, 256)       0

 block4_conv1 (Conv2D)       (None, 22, 22, 512)       1180160

 block4_conv2 (Conv2D)       (None, 22, 22, 512)       2359808

 block4_conv3 (Conv2D)       (None, 22, 22, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 11, 11, 512)       0

 block5_conv1 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv2 (Conv2D)       (None, 11, 11, 512)       2359808
```

```
 block5_conv3 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 5, 5, 512)         0

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

Fast feature extraction without data augmentation

Extracting the VGG16 features and corresponding labels

```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
test_features, test_labels =  get_features_and_labels(test_dataset)
```

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 19ms/step
```

```
train_features.shape
```

```
(2000, 5, 5, 512)
```

Defining and training the densely connected classifier

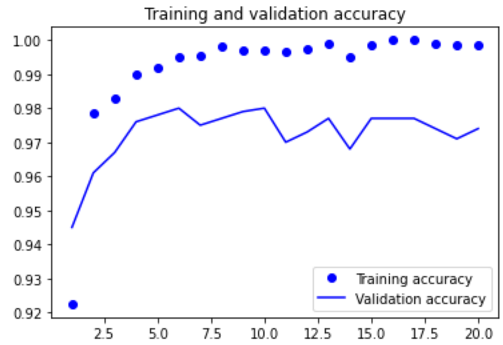
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
      filepath="feature_extraction.keras",
      save_best_only=True,
      monitor="val_loss")
]
history = model.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

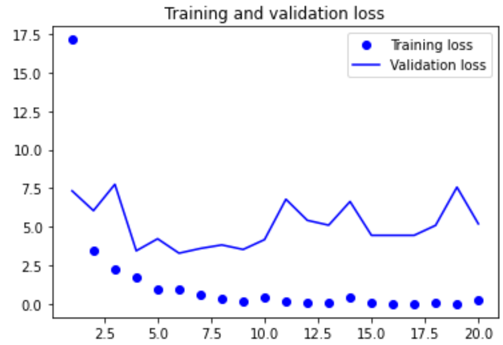
```
Epoch 1/20
63/63 [==============================] - 2s 18ms/step - loss: 17.1352 - accuracy: 0.9225 - val_loss: 7.3274 - val_accurac
Epoch 2/20
63/63 [==============================] - 0s 7ms/step - loss: 3.4957 - accuracy: 0.9785 - val_loss: 6.0471 - val_accuracy:
Epoch 3/20
63/63 [==============================] - 0s 6ms/step - loss: 2.2232 - accuracy: 0.9830 - val_loss: 7.7548 - val_accuracy:
Epoch 4/20
63/63 [==============================] - 0s 7ms/step - loss: 1.7005 - accuracy: 0.9900 - val_loss: 3.4450 - val_accuracy:
Epoch 5/20
63/63 [==============================] - 0s 7ms/step - loss: 0.9089 - accuracy: 0.9920 - val_loss: 4.2232 - val_accuracy:
Epoch 6/20
63/63 [==============================] - 0s 8ms/step - loss: 0.9305 - accuracy: 0.9950 - val_loss: 3.2892 - val_accuracy:
Epoch 7/20
63/63 [==============================] - 0s 6ms/step - loss: 0.6294 - accuracy: 0.9955 - val_loss: 3.5947 - val_accuracy:
Epoch 8/20
63/63 [==============================] - 0s 7ms/step - loss: 0.3114 - accuracy: 0.9980 - val_loss: 3.8249 - val_accuracy:
Epoch 9/20
63/63 [==============================] - 0s 6ms/step - loss: 0.1964 - accuracy: 0.9970 - val_loss: 3.5310 - val_accuracy:
Epoch 10/20
63/63 [==============================] - 0s 6ms/step - loss: 0.4063 - accuracy: 0.9970 - val_loss: 4.1690 - val_accuracy:
Epoch 11/20
63/63 [==============================] - 0s 7ms/step - loss: 0.1918 - accuracy: 0.9965 - val_loss: 6.7854 - val_accuracy:
Epoch 12/20
63/63 [==============================] - 0s 6ms/step - loss: 0.1008 - accuracy: 0.9975 - val_loss: 5.4202 - val_accuracy:
Epoch 13/20
63/63 [==============================] - 0s 7ms/step - loss: 0.0550 - accuracy: 0.9990 - val_loss: 5.1054 - val_accuracy:
Epoch 14/20
63/63 [==============================] - 0s 7ms/step - loss: 0.4349 - accuracy: 0.9950 - val_loss: 6.6374 - val_accuracy:
Epoch 15/20
63/63 [==============================] - 0s 7ms/step - loss: 0.1222 - accuracy: 0.9985 - val_loss: 4.4481 - val_accuracy:
Epoch 16/20
63/63 [==============================] - 0s 6ms/step - loss: 7.5899e-33 - accuracy: 1.0000 - val_loss: 4.4481 - val_accur
Epoch 17/20
63/63 [==============================] - 0s 7ms/step - loss: 3.9944e-27 - accuracy: 1.0000 - val_loss: 4.4481 - val_accur
Epoch 18/20
63/63 [==============================] - 0s 7ms/step - loss: 0.0474 - accuracy: 0.9990 - val_loss: 5.0921 - val_accuracy:
Epoch 19/20
63/63 [==============================] - 0s 6ms/step - loss: 0.0356 - accuracy: 0.9985 - val_loss: 7.5710 - val_accuracy:
Epoch 20/20
63/63 [==============================] - 0s 7ms/step - loss: 0.2671 - accuracy: 0.9985 - val_loss: 5.1937 - val_accuracy:
```

Plotting the results

```
import matplotlib.pyplot as plt
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
```

```
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





Feature extraction together with data augmentation

Instantiating and freezing the VGG16 convolutional base

```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

Printing the list of trainable weights before and after freezing

```
conv_base.trainable = True
print("This is the number of trainable weights "
      "before freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 26
```

```
conv_base.trainable = False
print("This is the number of trainable weights "
      "after freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 0
```

Adding a data augmentation stage and a classifier to the convolutional base

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
```

```
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 22/50
63/63 [==============================] - 12s 193ms/step - loss: 1.1954 - accuracy: 0.9830 - val_loss: 2.9859 - val_accura
Epoch 23/50
63/63 [==============================] - 12s 193ms/step - loss: 1.6731 - accuracy: 0.9825 - val_loss: 4.4921 - val_accura
Epoch 24/50
63/63 [==============================] - 12s 194ms/step - loss: 1.3528 - accuracy: 0.9830 - val_loss: 3.6913 - val_accura
Epoch 25/50
63/63 [==============================] - 12s 194ms/step - loss: 0.9937 - accuracy: 0.9855 - val_loss: 3.8712 - val_accura
Epoch 26/50
63/63 [==============================] - 13s 195ms/step - loss: 1.1740 - accuracy: 0.9865 - val_loss: 3.5527 - val_accura
Epoch 27/50
63/63 [==============================] - 12s 193ms/step - loss: 0.6175 - accuracy: 0.9890 - val_loss: 2.6135 - val_accura
Epoch 28/50
63/63 [==============================] - 12s 194ms/step - loss: 1.2226 - accuracy: 0.9825 - val_loss: 2.4289 - val_accura
Epoch 29/50
63/63 [==============================] - 12s 193ms/step - loss: 1.1220 - accuracy: 0.9810 - val_loss: 2.5970 - val_accura
Epoch 30/50
63/63 [==============================] - 13s 196ms/step - loss: 1.1136 - accuracy: 0.9815 - val_loss: 1.8565 - val_accura
Epoch 31/50
63/63 [==============================] - 13s 198ms/step - loss: 0.6502 - accuracy: 0.9880 - val_loss: 1.5259 - val_accura
Epoch 32/50
63/63 [==============================] - 13s 198ms/step - loss: 0.5585 - accuracy: 0.9875 - val_loss: 1.5048 - val_accura
Epoch 33/50
63/63 [==============================] - 12s 194ms/step - loss: 0.8992 - accuracy: 0.9870 - val_loss: 3.9911 - val_accura
Epoch 34/50
63/63 [==============================] - 12s 193ms/step - loss: 0.6869 - accuracy: 0.9880 - val_loss: 1.6864 - val_accura
Epoch 35/50
63/63 [==============================] - 12s 192ms/step - loss: 0.6020 - accuracy: 0.9845 - val_loss: 2.1977 - val_accura
Epoch 36/50
63/63 [==============================] - 12s 193ms/step - loss: 0.8081 - accuracy: 0.9840 - val_loss: 4.0316 - val_accura
Epoch 37/50
63/63 [==============================] - 12s 192ms/step - loss: 0.7218 - accuracy: 0.9875 - val_loss: 2.4624 - val_accura
Epoch 38/50
63/63 [==============================] - 12s 193ms/step - loss: 0.5975 - accuracy: 0.9865 - val_loss: 2.1100 - val_accura
Epoch 39/50
63/63 [==============================] - 13s 196ms/step - loss: 0.6047 - accuracy: 0.9900 - val_loss: 1.9450 - val_accura
Epoch 40/50
63/63 [==============================] - 12s 194ms/step - loss: 0.4830 - accuracy: 0.9890 - val_loss: 1.5697 - val_accura
Epoch 41/50
63/63 [==============================] - 13s 195ms/step - loss: 0.3343 - accuracy: 0.9895 - val_loss: 1.7509 - val_accura
Epoch 42/50
63/63 [==============================] - 13s 195ms/step - loss: 0.4581 - accuracy: 0.9850 - val_loss: 1.9371 - val_accura
Epoch 43/50
63/63 [==============================] - 12s 194ms/step - loss: 0.9432 - accuracy: 0.9840 - val_loss: 4.6797 - val_accura
Epoch 44/50
63/63 [==============================] - 12s 193ms/step - loss: 0.9636 - accuracy: 0.9875 - val_loss: 2.4952 - val_accura
Epoch 45/50
63/63 [==============================] - 12s 194ms/step - loss: 0.3675 - accuracy: 0.9895 - val_loss: 1.8321 - val_accura
Epoch 46/50
63/63 [==============================] - 12s 194ms/step - loss: 0.6764 - accuracy: 0.9875 - val_loss: 1.8318 - val_accura
Epoch 47/50
63/63 [==============================] - 13s 197ms/step - loss: 0.6043 - accuracy: 0.9870 - val_loss: 2.7244 - val_accura
Epoch 48/50
63/63 [==============================] - 13s 196ms/step - loss: 0.5413 - accuracy: 0.9880 - val_loss: 2.5047 - val_accura
Epoch 49/50
63/63 [==============================] - 13s 195ms/step - loss: 0.5012 - accuracy: 0.9880 - val_loss: 2.1289 - val_accura
Epoch 50/50
63/63 [==============================] - 13s 198ms/step - loss: 0.3326 - accuracy: 0.9910 - val_loss: 2.2701 - val_accura
```

Evaluating the model on the test set

```
test_model = keras.models.load_model(
    "feature_extraction_with_data_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 4s 117ms/step - loss: 2.6057 - accuracy: 0.9780
Test accuracy: 0.978
```

A pretrained VGG16 model with Fine-tuning

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

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

Freezing all the layers except the last fourth one

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

Fine Tuning the model

```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 2/30
```

```
63/63 [==============================] - 15s 236ms/step - loss: 0.1823 - accuracy: 0.9940 - val_loss: 1.7751 - val_accura
Epoch 4/30
63/63 [==============================] - 15s 234ms/step - loss: 0.1019 - accuracy: 0.9965 - val_loss: 1.4731 - val_accura
Epoch 5/30
63/63 [==============================] - 15s 229ms/step - loss: 0.5098 - accuracy: 0.9860 - val_loss: 1.3598 - val_accura
Epoch 6/30
63/63 [==============================] - 15s 233ms/step - loss: 0.4768 - accuracy: 0.9880 - val_loss: 1.5014 - val_accura
Epoch 7/30
63/63 [==============================] - 15s 233ms/step - loss: 0.5004 - accuracy: 0.9905 - val_loss: 1.4305 - val_accura
Epoch 8/30
63/63 [==============================] - 15s 232ms/step - loss: 0.3937 - accuracy: 0.9905 - val_loss: 1.4639 - val_accura
Epoch 9/30
63/63 [==============================] - 15s 230ms/step - loss: 0.2225 - accuracy: 0.9940 - val_loss: 1.2223 - val_accura
Epoch 10/30
63/63 [==============================] - 15s 231ms/step - loss: 0.1668 - accuracy: 0.9935 - val_loss: 1.2244 - val_accura
Epoch 11/30
63/63 [==============================] - 15s 234ms/step - loss: 0.2652 - accuracy: 0.9935 - val_loss: 1.6508 - val_accura
Epoch 12/30
63/63 [==============================] - 15s 229ms/step - loss: 0.2628 - accuracy: 0.9930 - val_loss: 1.6190 - val_accura
Epoch 13/30
63/63 [==============================] - 15s 230ms/step - loss: 0.2663 - accuracy: 0.9930 - val_loss: 1.5505 - val_accura
Epoch 14/30
63/63 [==============================] - 15s 237ms/step - loss: 0.1638 - accuracy: 0.9970 - val_loss: 1.4319 - val_accura
Epoch 15/30
63/63 [==============================] - 15s 235ms/step - loss: 0.1947 - accuracy: 0.9940 - val_loss: 1.5176 - val_accura
Epoch 16/30
63/63 [==============================] - 15s 232ms/step - loss: 0.0989 - accuracy: 0.9930 - val_loss: 1.3109 - val_accura
Epoch 17/30
63/63 [==============================] - 15s 232ms/step - loss: 0.1010 - accuracy: 0.9965 - val_loss: 1.4065 - val_accura
Epoch 18/30
63/63 [==============================] - 15s 234ms/step - loss: 0.1406 - accuracy: 0.9960 - val_loss: 1.5586 - val_accura
Epoch 19/30
63/63 [==============================] - 15s 235ms/step - loss: 0.1583 - accuracy: 0.9945 - val_loss: 1.2478 - val_accura
Epoch 20/30
63/63 [==============================] - 15s 233ms/step - loss: 0.0933 - accuracy: 0.9965 - val_loss: 1.9716 - val_accura
Epoch 21/30
63/63 [==============================] - 15s 237ms/step - loss: 0.2197 - accuracy: 0.9940 - val_loss: 1.4839 - val_accura
Epoch 22/30
63/63 [==============================] - 15s 236ms/step - loss: 0.0411 - accuracy: 0.9975 - val_loss: 1.4054 - val_accura
Epoch 23/30
63/63 [==============================] - 15s 234ms/step - loss: 0.2325 - accuracy: 0.9940 - val_loss: 1.7094 - val_accura
Epoch 24/30
63/63 [==============================] - 15s 232ms/step - loss: 0.1976 - accuracy: 0.9930 - val_loss: 2.1511 - val_accura
Epoch 25/30
63/63 [==============================] - 15s 233ms/step - loss: 0.1505 - accuracy: 0.9960 - val_loss: 1.8563 - val_accura
Epoch 26/30
63/63 [==============================] - 15s 235ms/step - loss: 0.1040 - accuracy: 0.9955 - val_loss: 1.3652 - val_accura
Epoch 27/30
63/63 [==============================] - 15s 231ms/step - loss: 0.0817 - accuracy: 0.9965 - val_loss: 1.5703 - val_accura
Epoch 28/30
63/63 [==============================] - 15s 229ms/step - loss: 0.2087 - accuracy: 0.9955 - val_loss: 1.5683 - val_accura
Epoch 29/30
63/63 [==============================] - 15s 230ms/step - loss: 0.1724 - accuracy: 0.9950 - val_loss: 1.4953 - val_accura
Epoch 30/30
63/63 [==============================] - 15s 234ms/step - loss: 0.0774 - accuracy: 0.9960 - val_loss: 1.5092 - val_accura
```

```
model = keras.models.load_model("fine_tuning.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 4s 116ms/step - loss: 1.5497 - accuracy: 0.9800
Test accuracy: 0.980
```

Summary

• **Model-1:**

o Test Accuracy: 74% and loss: 3.52

o Training sample = 1000, Validation sample = 500, Test sample = 500

o Method to reduce overfitting: Regularization

o *Conclusion:* The accuracy in the initial model without any regularization was 71%. Then I tried to change the optimizer to Adam instead of rmsprop with 30 epochs and Dropout, the test accuracy cameout to be 68.3% and loss is 0.61. Finally, I ran a model with 50 epochs and Adam then the test accuracy is 74%.

• **Model – 2:**

o Test Accuracy: 82% and loss: 0.42

o Training sample = 1500 Validation sample = 500, Test sample = 500

o Methods: Regularization and Data Augmentation

o* Conclusion:* I tried a model with Adam, Data Augmentation and regularization and 50 epochs the accuracy of the model was 78% and loss was 0.49. The final model had rms prop as an optimizer and the final accuracy was 81% and loss was 0.46. The result obtained was better than the model with only regularization.

• **Model – 3:**

o Test Accuracy: 83% and loss: 0.52

o Training sample = 2000 , Validation sample = 500 and Test sample = 500

o Methods: Regularization and Data Augmentation

o Conclusion: Increased the training sample to 2000 and the training and validation sample remain the same. Initially, I ran a model with the same metrics but Adam as an optimizer then the accuracy was 76% and loss was 0.52. Then changed the optimizer to rmsprop then I achieved better results with accuracy as 83% and loss as 0.52

• **Model – 4:**

o VGG16 Pretrained Convnet Network

o Without data augmentation: Test Accuracy: 98% and loss: 1.55

Using a pretrained model to apply deep learning to tiny image datasets is a highly effective method. A pretrained model is one that has been trained earlier on a big dataset, usually for a large – scale image classification problem.

Fine tuning the model was to unfreeze the top layers of the frozen model for feature extraction and training. It is more on to adjusting the representations of the model.

There were three different scenarios considered while running the model • Pre – trained model with Data Augmentation • Pre – trained model without Data Augmentation • Pre – trained model with fine tuning

Pre-trained model with fine tuning had a test accuracy of 98% whereas, Pre – trained model with Data Augmentation has a test accuracy of 97%

Conclusion: Using a pretrained model to apply deep learning techniques proves to be much more effective than training a model from scratch.