Assignment 2: Advanced Machine Learning

Names: Atshaya Suresh, Anusha Banda

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?**

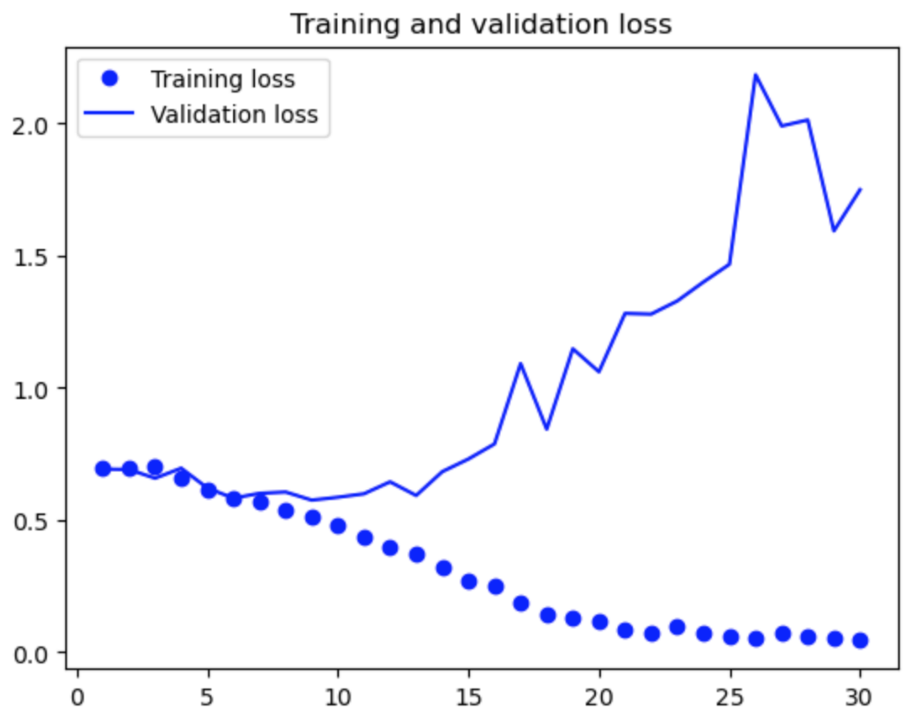
We started with processing the cats-vs-dogs zip file. We unzipped it and extracted the training set. From the training set which contained nearly 25000 images, we created a subset for our training, validation and test set. We took a subset of the images, i.e., 1000 images for training, 500 images for validation and 500 images for testing. We built the model with the following architecture:

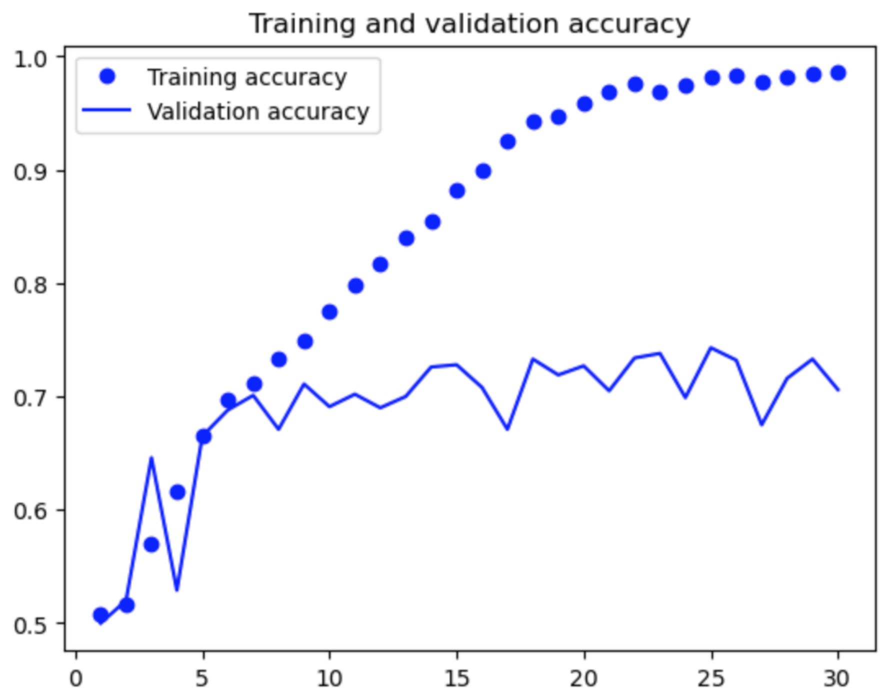
Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d_8 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_9 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_10 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_10 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_11 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_11 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_12 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_2 (Flatten)	(None, 12544)	0
dense_4 (Dense)	(None, 1)	12,545

Total params: 991,041 (3.78 MB)
Trainable params: 991,041 (3.78 MB)
Non-trainable params: 0 (0.00 B)

Since it was balanced data, we used Accuracy as the performance metric, Optimizer as "rmsprop" and loss function we used is binary cross entropy since it is a binary classification problem (dogs/cats). We executed 30 Epochs. The model converged in the 9^{th} Epoch. In the 9^{th} Epoch, the training accuracy was 0.7355 and validation accuracy was 0.7110. Also, the loss was 0.5750. Subsequently, in the 10^{th} Epoch, though the training accuracy increased to 0.7671, the validation accuracy dropped to 0.6910 and the loss went up to 0.5859. That is when we conclude that the model is starting to overfit. We can also infer the same from the following plot which shows the path of both Accuracy and loss function





Clearly, after the 9th Epoch (Epoch: x-axis), the model is overfitting. To overcome this problem, we can early stop the model to reduce the chances of overfitting. Finally, we achieved an accuracy of around 69% on the test set.

The consolidated value of the accuracy are:

Data Subset	Accuracy
Training	0.7355
Validation	0.7110
Test	0.6950

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?**

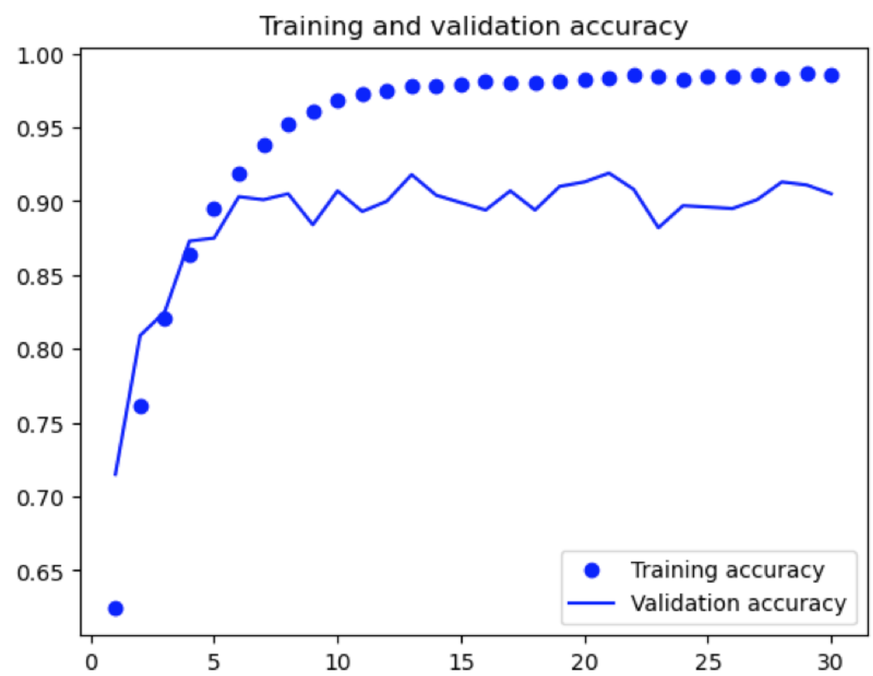
As the next step, we increased the training sample from 1000 to 10,000. This process has its own advantages and disadvantages. Some of the advantages include better learning during training by the model, improved generalization, potential for higher accuracy, and a chance for the model to learn complex patterns. On the other hand, it also has its own disadvantages, which include more requirements for computational resources and increased training time. To give an overview of the code, The code compiles a neural network model using binary cross-entropy as the loss function, RMSprop as the optimizer, and accuracy as the metric to monitor during training. It loads datasets for training, validation, and testing using the image_dataset_from_directory function provided by TensorFlow. These datasets are set to use images resized to 180x180 pixels and batched into sets of 32 samples.

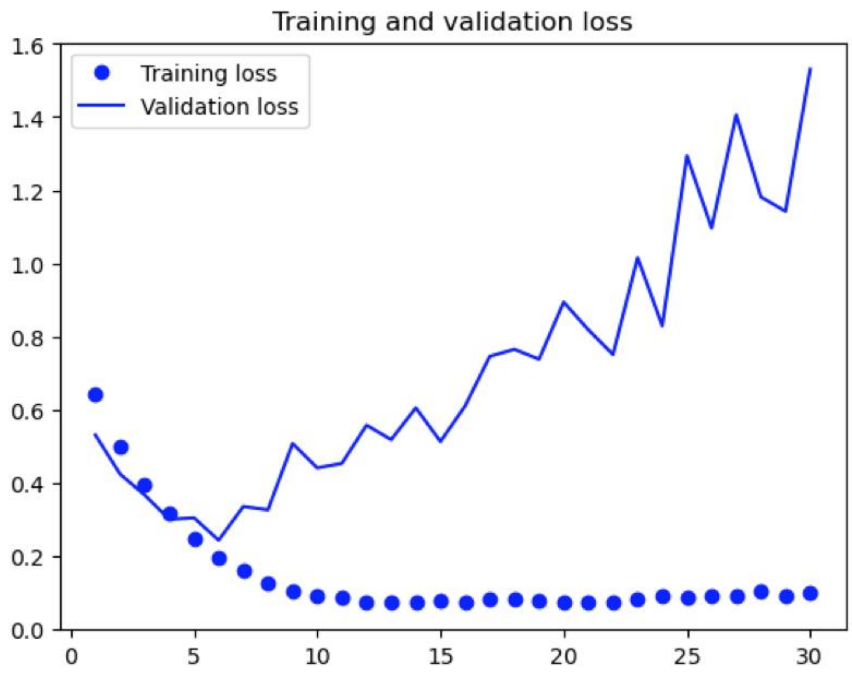
The architecture of the model is as follows:

```
Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ input_layer_1 (InputLayer)      │ (None, 180, 180, 3)    │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ rescaling_1 (Rescaling)         │ (None, 180, 180, 3)    │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_5 (Conv2D)               │ (None, 178, 178, 32)   │
896 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_4 (MaxPooling2D)  │ (None, 89, 89, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_6 (Conv2D)               │ (None, 87, 87, 64)     │
18,496 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_5 (MaxPooling2D)  │ (None, 43, 43, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_7 (Conv2D)               │ (None, 41, 41, 128)    │
73,856 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_6 (MaxPooling2D)  │ (None, 20, 20, 128)    │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_8 (Conv2D)               │ (None, 18, 18, 256)    │
295,168 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_7 (MaxPooling2D)  │ (None, 9, 9, 256)      │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_9 (Conv2D)               │ (None, 7, 7, 256)      │
590,080 │
```

```
├─────────────────────────────────┼─────────────────────────┼──────────
──────┤
│ flatten_1 (Flatten)             │ (None, 12544)           │
0 │
├─────────────────────────────────┼─────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                 │ (None, 1)               │
12,545 │
└─────────────────────────────────┴─────────────────────────┴──────────
──────┘

 Total params: 991,041 (3.78 MB)

 Trainable params: 991,041 (3.78 MB)

 Non-trainable params: 0 (0.00 B)
```

We executed 30 Epochs. The model converged in the 6th Epoch. In the 6th Epoch, the training accuracy was 0.9129 and validation accuracy was 0.9030. Also, the loss was 0.2421. Subsequently, in the next 5 epochs, the loss did not reduce. Also, the validation accuracy also did not significantly increase and in fact, started dropping.That is when we conclude that the model is starting to overfit. We can also infer the same from the following plot which shows the path of both Accuracy and loss function.





The consolidated value of the accuracy are:

Data Subset	Accuracy
Training	0.9129
Validation	0.9030
Test	0.872

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 the best prediction results.**

There is an ultimate truth in Machine Learning, more samples imply better predictions. Overfitting is caused by having too few samples to learn from, rendering you unable to train a model that can generalize to new data. Given infinite data, your model would be exposed to every possible aspect of the data distribution at hand: you would never overfit. Data augmentation takes the approach of generating more training data from existing training samples by augmenting the samples via a number of random transformations that yield believable-looking images. The goal is that, at training time, your model will never see the exact same picture twice. This helps expose the model to more aspects of the data so it can generalize better. In Keras, this can be done by adding a number of data augmentation layers at the start of your model. In our code, we used the following aspects to augment the images we had:

- RandomFlip("horizontal")—Applies horizontal flipping to a random 50% of the images that go through it.

- RandomRotation(0.1)—Rotates the input images by a random value in the range [–10%, +10%] (these are fractions of a full circle—in degrees, the range would be [–36 degrees, +36 degrees])
- RandomZoom(0.2)—Zooms in or out of the image by a random factor in the range [-20%, +20%]

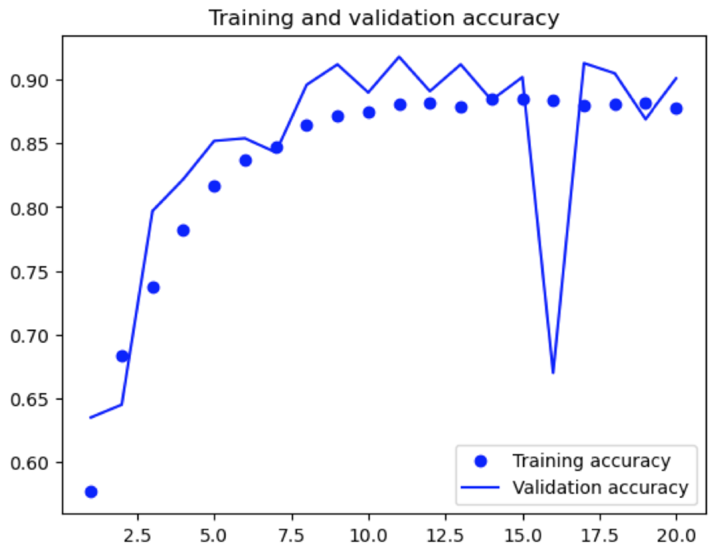
Further, we used dropout method to decrease the overfitting. In our model, the following is the significance of using the dropout:

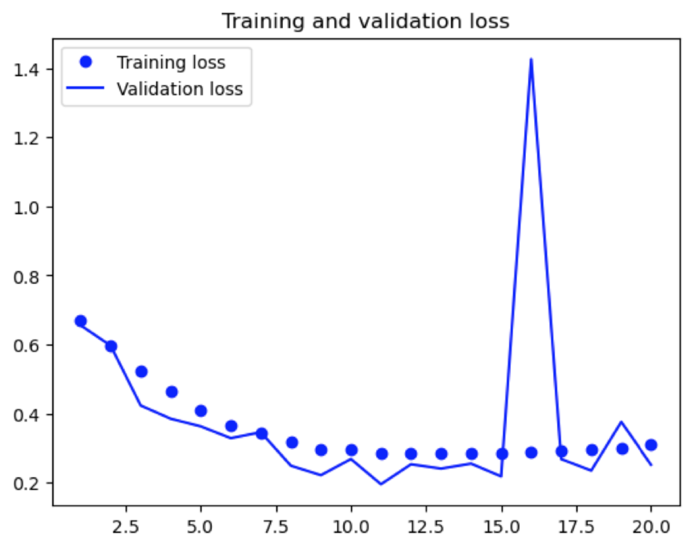
- **layers.Dropout(0.5)**: This creates a Dropout layer with a dropout rate of 0.5, which means that during training, 50% of the input units to this layer will randomly set to zero at each update, effectively "dropping out" those units from the network. Dropout is a regularization technique used to prevent overfitting by reducing the interdependence of neurons.
- **(x)**: This applies the Dropout layer to the previous layer **x**. In the context of the code snippet you provided, **x** is the output of the Flatten layer, which in turn comes after several convolutional and pooling layers.

The purpose of adding a Dropout layer in a convolutional neural network (CNN) like this is to help prevent overfitting. By randomly dropping units during training, Dropout introduces noise into the network and forces it to learn more robust features. This can lead to better generalization and performance on unseen data.

In summary, the Dropout layer in this code snippet with a dropout rate of 0.5 helps regularize the CNN model by randomly dropping 50% of the units during training, which can improve the model's ability to generalize and avoid overfitting to the training data. Clearly, by implementing data augmentation and dropout method, we increased our performance, with an **accuracy of 0.893** from 0.695.

We executed 20 Epochs. The model converged in the 15^{th} Epoch. In the 15^{th} Epoch, the training accuracy was 0.8828 and validation accuracy was 0.9020. Also, the loss was 0.2180. Subsequently, in the next 5 Epochs, the loss function never dropped. That is when we conclude that the model is starting to overfit. We can also infer the same from the following plot which shows the path of both Accuracy and loss function.





The consolidated value of the accuracy are:

Data Subset	Accuracy
Training	0.8828
Validation	0.9020
Test	0.893

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

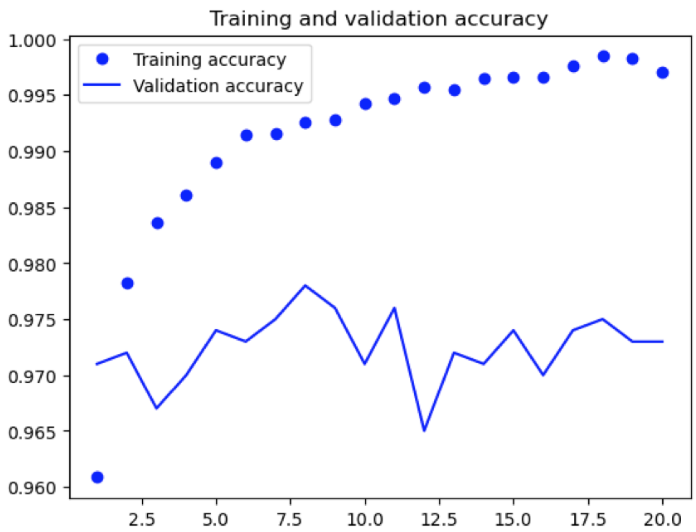
where you trained from scratch. Again, use any and all optimization techniques to get best performance.

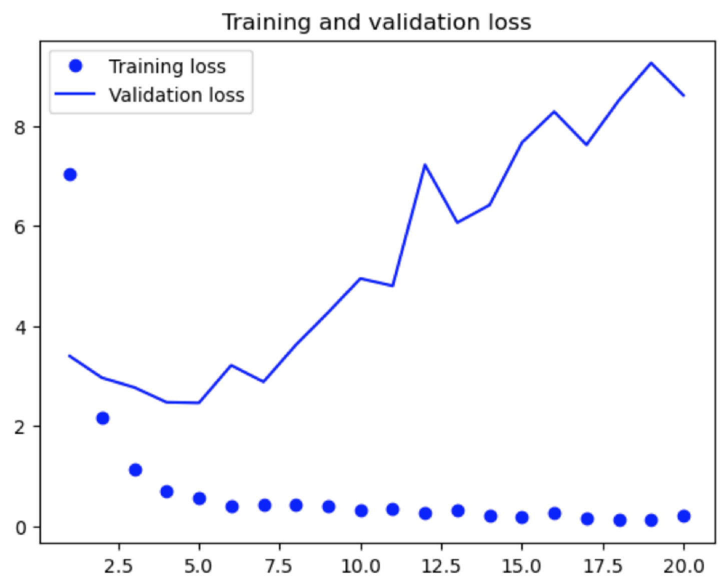
A common and highly effective approach to deep learning on small image datasets is to use a pretrained model. A *pretrained model* is a model that was previously trained on a large dataset, typically on a large-scale image-classification task. If this original data- set is large enough and general enough, the spatial hierarchy of features learned by the pretrained model can effectively act as a generic model of the visual world, and hence, its features can prove useful for many different computer vision problems, even though these new problems may involve completely different classes than those of the original task. Here, We'll use the VGG16 architecture, developed by Karen Simonyan and Andrew Zisserman in 2014. Here is the structure of VGG16

```
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ input_layer (InputLayer)          │ (None, 180, 180, 3)     │
0 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block1_conv1 (Conv2D)             │ (None, 180, 180, 64)    │
1,792 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block1_conv2 (Conv2D)             │ (None, 180, 180, 64)    │
36,928 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block1_pool (MaxPooling2D)        │ (None, 90, 90, 64)      │
0 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block2_conv1 (Conv2D)             │ (None, 90, 90, 128)     │
73,856 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block2_conv2 (Conv2D)             │ (None, 90, 90, 128)     │
147,584 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block2_pool (MaxPooling2D)        │ (None, 45, 45, 128)     │
0 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block3_conv1 (Conv2D)             │ (None, 45, 45, 256)     │
295,168 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block3_conv2 (Conv2D)             │ (None, 45, 45, 256)     │
590,080 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block3_conv3 (Conv2D)             │ (None, 45, 45, 256)     │
590,080 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block3_pool (MaxPooling2D)        │ (None, 22, 22, 256)     │
0 │
├───────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ block4_conv1 (Conv2D)             │ (None, 22, 22, 512)     │
1,180,160 │
```

```
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block4_conv2 (Conv2D)        │ (None, 22, 22, 512)  │
2,359,808 │
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block4_conv3 (Conv2D)        │ (None, 22, 22, 512)  │
2,359,808 │
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block4_pool (MaxPooling2D)   │ (None, 11, 11, 512)  │
0 │
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block5_conv1 (Conv2D)        │ (None, 11, 11, 512)  │
2,359,808 │
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block5_conv2 (Conv2D)        │ (None, 11, 11, 512)  │
2,359,808 │
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block5_conv3 (Conv2D)        │ (None, 11, 11, 512)  │
2,359,808 │
├──────────────────────────────┼──────────────────────┼──────────
──────┤
│ block5_pool (MaxPooling2D)   │ (None, 5, 5, 512)    │
0 │
└──────────────────────────────┴──────────────────────┴──────────
──────┘

 Total params: 14,714,688 (56.13 MB)

 Trainable params: 14,714,688 (56.13 MB)

 Non-trainable params: 0 (0.00 B)
```

Since the model was run on a CPU and not a GPU, we are using a pretrained model with feature extraction and no data augmentation. Here, we retained/froze the convolutional base and modified the dense layer to suit our requirement. We used only 20 Epochs, since the convergence is faster. In fact, when we started itself, the validation accuracy was 0.97. It may indicate that, our model is overfitting even during the early stages. If we can execute the code on a GPU, we can further use Data Augmentation along with VGG16 or any other pretrained model to achieve better performance. One major disadvantage of this without data Augmentation is the fact that, the model overfits. We can observe that from the test accuracy, which is only 0.872. That is why data augmentation is used. In fact, it is evident in the following graph:





The consolidated value of the accuracy are:

Data Subset	Accuracy
Training	0.99
Validation	1
Test	0.872

Conclusion: In conclusion, our exploration of advanced machine learning techniques for the Cats & Dogs classification problem has been insightful and productive. We embarked on this journey by building a network from scratch with a modest training sample size of 1000 images, achieving an accuracy of approximately 69% on the test set. Recognizing the challenges of overfitting, we scaled up the training sample to 10,000 images, resulting in a notable performance boost with an accuracy of 87.2% on the test set.

Continuing our quest for optimal performance, we employed data augmentation methods such as random flipping, rotation, and zooming, coupled with dropout regularization. This strategic combination led to a significant improvement in our model's accuracy, reaching 89.3%. This highlighted the effectiveness of augmenting training data and implementing regularization techniques to mitigate overfitting and enhance generalization.

Furthermore, we delved into leveraging pretrained models, specifically VGG16, which demonstrated impressive performance with a validation accuracy of 97% and yet a test accuracy of 87.3% due to overfitting the model. While we initially operated without data augmentation due to hardware constraints, the potential for further enhancements through GPU acceleration and comprehensive data augmentation is evident.

Our journey emphasized the importance of experimentation, optimization, and leveraging advanced techniques in machine learning to achieve superior performance in complex classification tasks. It also highlighted the value of adapting strategies based on data size, computational resources, and model architecture to achieve the best possible outcomes.

```
import os
from PIL import Image
import matplotlib.pyplot as plt

folder_path = '/Users/p/Downloads/dogs-vs-cats/train'
image_files = [f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))]

for i, img_file in enumerate(image_files[:5]):  # Loop through the
first 5 image files
    img_path = os.path.join(folder_path, img_file)
    img = Image.open(img_path)
    plt.imshow(img)
    plt.axis('off')  # Turn off axis for cleaner display
    plt.title(f'Image {i+1}: {img_file}')  # Display image file name
as title
    plt.show()
```

Image 2: cat.5077.jpg

Image 3: dog.7322.jpg

Image 4: cat.2718.jpg

Image 5: cat.10151.jpg

```python
import os, shutil, pathlib
```

```
original_dir = pathlib.Path("/Users/p/Downloads/dogs-vs-cats/train")
new_base_dir =
pathlib.Path("/Users/p/Downloads/dogs-vs-cats/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, exist_ok=True)  # Use exist_ok=True to avoid
errors if the directory already exists
        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=2500)

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)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

>>> model.summary()
Model: "model_2"

Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━
━━━━━┩
│ input_layer_2 (InputLayer)       │ (None, 180, 180, 3)    │
0 │
├──────────────────────────────────┼────────────────────────┼─────────────
─────┤
```

```
│ rescaling (Rescaling)             │ (None, 180, 180, 3)        │
0 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ conv2d_8 (Conv2D)                 │ (None, 178, 178, 32)       │
896 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ max_pooling2d_8 (MaxPooling2D)    │ (None, 89, 89, 32)         │
0 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ conv2d_9 (Conv2D)                 │ (None, 87, 87, 64)         │
18,496 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ max_pooling2d_9 (MaxPooling2D)    │ (None, 43, 43, 64)         │
0 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ conv2d_10 (Conv2D)                │ (None, 41, 41, 128)        │
73,856 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ max_pooling2d_10 (MaxPooling2D)   │ (None, 20, 20, 128)        │
0 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ conv2d_11 (Conv2D)                │ (None, 18, 18, 256)        │
295,168 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ max_pooling2d_11 (MaxPooling2D)   │ (None, 9, 9, 256)          │
0 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ conv2d_12 (Conv2D)                │ (None, 7, 7, 256)          │
590,080 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ flatten_2 (Flatten)               │ (None, 12544)              │
0 │
├───────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ dense_4 (Dense)                   │ (None, 1)                  │
12,545 │
└───────────────────────────────────┴────────────────────────────┴──────────
──────┘

 Total params: 991,041 (3.78 MB)
```

```
 Trainable params: 991,041 (3.78 MB)

 Non-trainable params: 0 (0.00 B)
```

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

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

```
>>> 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,)
```

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

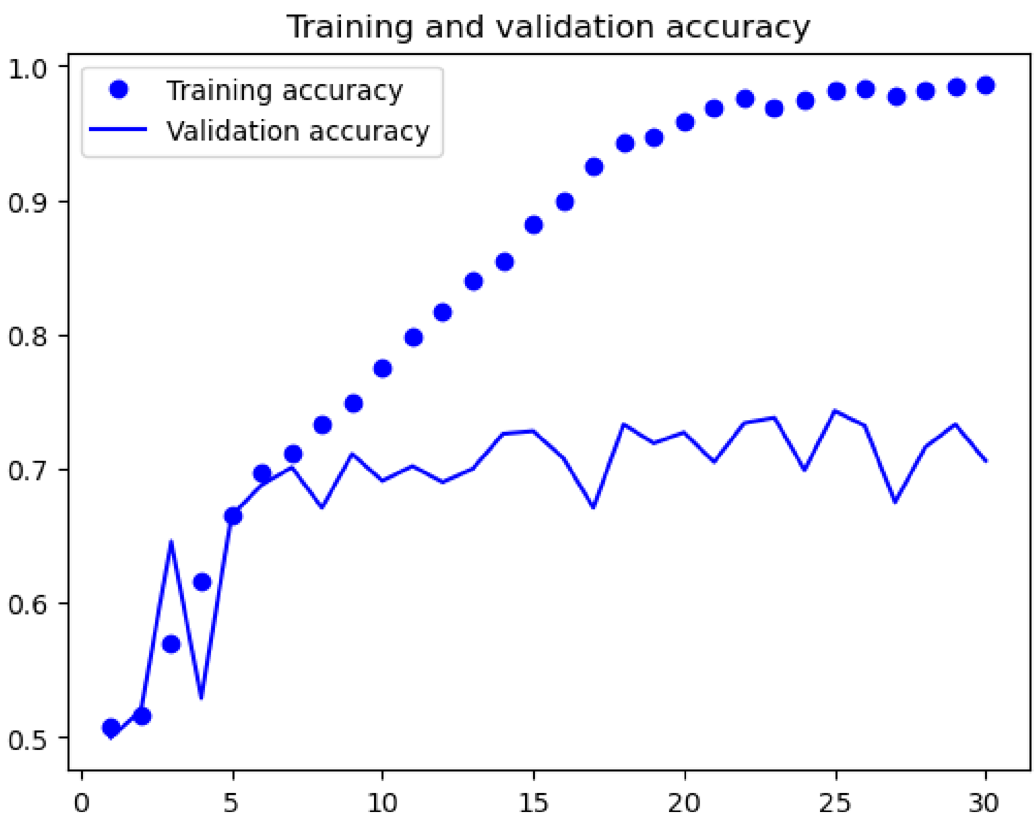
```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 903ms/step - accuracy: 0.5033 - loss:
0.6991 - val_accuracy: 0.5000 - val_loss: 0.6925
Epoch 2/30
```

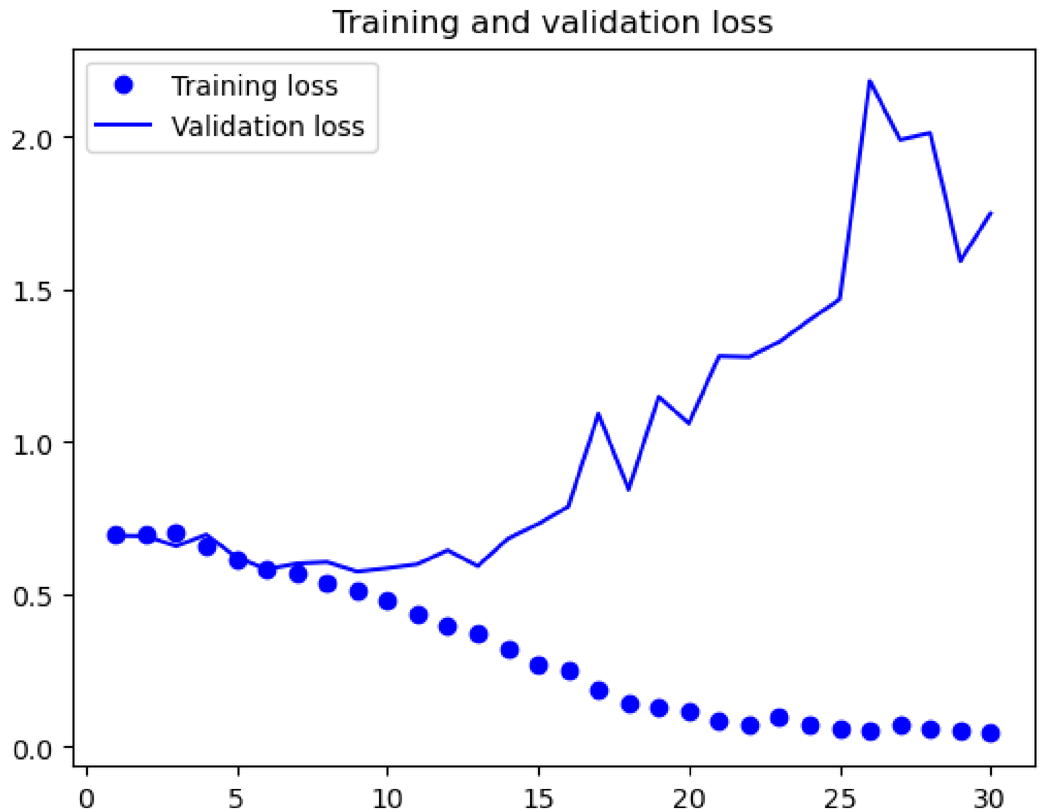
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 922ms/step - accuracy: 0.5084 - loss:
0.6939 - val_accuracy: 0.5200 - val_loss: 0.6909
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 927ms/step - accuracy: 0.5454 - loss:
0.6964 - val_accuracy: 0.6460 - val_loss: 0.6579
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 60s 956ms/step - accuracy: 0.6043 - loss:
0.6650 - val_accuracy: 0.5290 - val_loss: 0.6959
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 921ms/step - accuracy: 0.6536 - loss:
0.6249 - val_accuracy: 0.6650 - val_loss: 0.6209
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 913ms/step - accuracy: 0.6933 - loss:
0.5916 - val_accuracy: 0.6880 - val_loss: 0.5824
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 57s 897ms/step - accuracy: 0.7095 - loss:
0.5849 - val_accuracy: 0.7010 - val_loss: 0.6007
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 57s 906ms/step - accuracy: 0.7134 - loss:
0.5481 - val_accuracy: 0.6710 - val_loss: 0.6057
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 921ms/step - accuracy: 0.7355 - loss:
0.5186 - val_accuracy: 0.7110 - val_loss: 0.5750
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 925ms/step - accuracy: 0.7671 - loss:
0.4836 - val_accuracy: 0.6910 - val_loss: 0.5859
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 924ms/step - accuracy: 0.7891 - loss:
0.4469 - val_accuracy: 0.7020 - val_loss: 0.5990
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 57s 909ms/step - accuracy: 0.8074 - loss:
0.4085 - val_accuracy: 0.6900 - val_loss: 0.6444
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 57s 909ms/step - accuracy: 0.8208 - loss:
0.3965 - val_accuracy: 0.7000 - val_loss: 0.5926
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 59s 932ms/step - accuracy: 0.8378 - loss:
0.3383 - val_accuracy: 0.7260 - val_loss: 0.6828
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 59s 943ms/step - accuracy: 0.8675 - loss:
0.2867 - val_accuracy: 0.7280 - val_loss: 0.7306
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 60s 959ms/step - accuracy: 0.8889 - loss:
0.2708 - val_accuracy: 0.7080 - val_loss: 0.7876
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 59s 941ms/step - accuracy: 0.9148 - loss:
0.2100 - val_accuracy: 0.6710 - val_loss: 1.0918
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 61s 965ms/step - accuracy: 0.9334 - loss:
```

```
0.1588 - val_accuracy: 0.7330 - val_loss: 0.8436
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 61s 961ms/step - accuracy: 0.9352 - loss:
0.1549 - val_accuracy: 0.7190 - val_loss: 1.1478
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 64s 1s/step - accuracy: 0.9558 - loss:
0.1228 - val_accuracy: 0.7270 - val_loss: 1.0606
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 62s 985ms/step - accuracy: 0.9702 - loss:
0.0846 - val_accuracy: 0.7050 - val_loss: 1.2816
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 59s 928ms/step - accuracy: 0.9780 - loss:
0.0718 - val_accuracy: 0.7340 - val_loss: 1.2789
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 58s 925ms/step - accuracy: 0.9698 - loss:
0.0893 - val_accuracy: 0.7380 - val_loss: 1.3286
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.9819 - loss:
0.0559 - val_accuracy: 0.6990 - val_loss: 1.4002
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 62s 980ms/step - accuracy: 0.9813 - loss:
0.0464 - val_accuracy: 0.7430 - val_loss: 1.4675
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 65s 1s/step - accuracy: 0.9789 - loss:
0.0682 - val_accuracy: 0.7320 - val_loss: 2.1847
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 61s 972ms/step - accuracy: 0.9704 - loss:
0.0802 - val_accuracy: 0.6750 - val_loss: 1.9908
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 62s 983ms/step - accuracy: 0.9829 - loss:
0.0563 - val_accuracy: 0.7160 - val_loss: 2.0136
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 61s 962ms/step - accuracy: 0.9862 - loss:
0.0470 - val_accuracy: 0.7330 - val_loss: 1.5937
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 60s 951ms/step - accuracy: 0.9855 - loss:
0.0458 - val_accuracy: 0.7060 - val_loss: 1.7496
```

```python
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 loss
Training loss
Validation loss
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30

```python
import os
from PIL import Image
import matplotlib.pyplot as plt

folder_path = '/Users/jeeva thangamani/Downloads/train/train'
image_files = [f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))]

import os, shutil, pathlib

original_dir = pathlib.Path("/Users/jeeva
thangamani/Downloads/train/train")
new_base_dir = pathlib.Path("/Users/jeeva
thangamani/Downloads/cats_vs_dogs_small_2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir, exist_ok=True)  # Use exist_ok=True to avoid
errors if the directory already exists
        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=10000)
make_subset("validation", start_index=10000, end_index=10500)
make_subset("test", start_index=10500, end_index=11000)

pip install tensorflow
```

```
Requirement already satisfied: tensorflow in f:\anaconda\lib\site-
packages (2.16.1)
Requirement already satisfied: tensorflow-intel==2.16.1 in f:\
anaconda\lib\site-packages (from tensorflow) (2.16.1)
Requirement already satisfied: absl-py>=1.0.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in f:\anaconda\
lib\site-packages (from tensorflow-intel==2.16.1->tensorflow)
(24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
f:\anaconda\lib\site-packages (from tensorflow-intel==2.16.1-
>tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (3.10.0)
Requirement already satisfied: libclang>=13.0.0 in f:\anaconda\lib\
```

```
site-packages (from tensorflow-intel==2.16.1->tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (0.3.2)
Requirement already satisfied: opt-einsum>=2.3.2 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (3.3.0)
Requirement already satisfied: packaging in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (2.31.0)
Requirement already satisfied: setuptools in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (68.2.2)
Requirement already satisfied: six>=1.12.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in f:\
anaconda\lib\site-packages (from tensorflow-intel==2.16.1->tensorflow)
(4.9.0)
Requirement already satisfied: wrapt>=1.11.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (1.62.1)
Requirement already satisfied: tensorboard<2.17,>=2.16 in f:\anaconda\
lib\site-packages (from tensorflow-intel==2.16.1->tensorflow) (2.16.2)
Requirement already satisfied: keras>=3.0.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (3.1.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
f:\anaconda\lib\site-packages (from tensorflow-intel==2.16.1-
>tensorflow) (0.31.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in f:\anaconda\
lib\site-packages (from tensorflow-intel==2.16.1->tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in f:\anaconda\lib\
site-packages (from astunparse>=1.6.0->tensorflow-intel==2.16.1-
>tensorflow) (0.41.2)
Requirement already satisfied: rich in f:\anaconda\lib\site-packages
(from keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (13.3.5)
Requirement already satisfied: namex in f:\anaconda\lib\site-packages
(from keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.0.7)
Requirement already satisfied: optree in f:\anaconda\lib\site-packages
(from keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in f:\
anaconda\lib\site-packages (from requests<3,>=2.21.0->tensorflow-
intel==2.16.1->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in f:\anaconda\lib\site-
packages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1-
>tensorflow) (3.4)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in f:\anaconda\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in f:\anaconda\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1->tensorflow) (2024.2.2)
Requirement already satisfied: markdown>=2.6.8 in f:\anaconda\lib\site-packages (from tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in f:\anaconda\lib\site-packages (from tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in f:\anaconda\lib\site-packages (from tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (2.2.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in f:\anaconda\lib\site-packages (from werkzeug>=1.0.1->tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in f:\anaconda\lib\site-packages (from rich->keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in f:\anaconda\lib\site-packages (from rich->keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in f:\anaconda\lib\site-packages (from markdown-it-py<3.0.0,>=2.2.0->rich->keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

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

```python
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)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
>>> model.summary()
Model: "model_2"
```

```
Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ input_layer_1 (InputLayer)       │ (None, 180, 180, 3)    │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ rescaling_1 (Rescaling)          │ (None, 180, 180, 3)    │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_5 (Conv2D)                │ (None, 178, 178, 32)   │
896 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_4 (MaxPooling2D)   │ (None, 89, 89, 32)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_6 (Conv2D)                │ (None, 87, 87, 64)     │
18,496 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_5 (MaxPooling2D)   │ (None, 43, 43, 64)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_7 (Conv2D)                │ (None, 41, 41, 128)    │
73,856 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_6 (MaxPooling2D)   │ (None, 20, 20, 128)    │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_8 (Conv2D)                │ (None, 18, 18, 256)    │
295,168 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_7 (MaxPooling2D)   │ (None, 9, 9, 256)      │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_9 (Conv2D)                │ (None, 7, 7, 256)      │
590,080 │
```

```
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ flatten_1 (Flatten)              │ (None, 12544)           │
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ dense_1 (Dense)                  │ (None, 1)               │
12,545 │
└──────────────────────────────────┴─────────────────────────┴───────────
──────┘

 Total params: 991,041 (3.78 MB)

 Trainable params: 991,041 (3.78 MB)

 Non-trainable params: 0 (0.00 B)
```

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

```
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 20000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

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

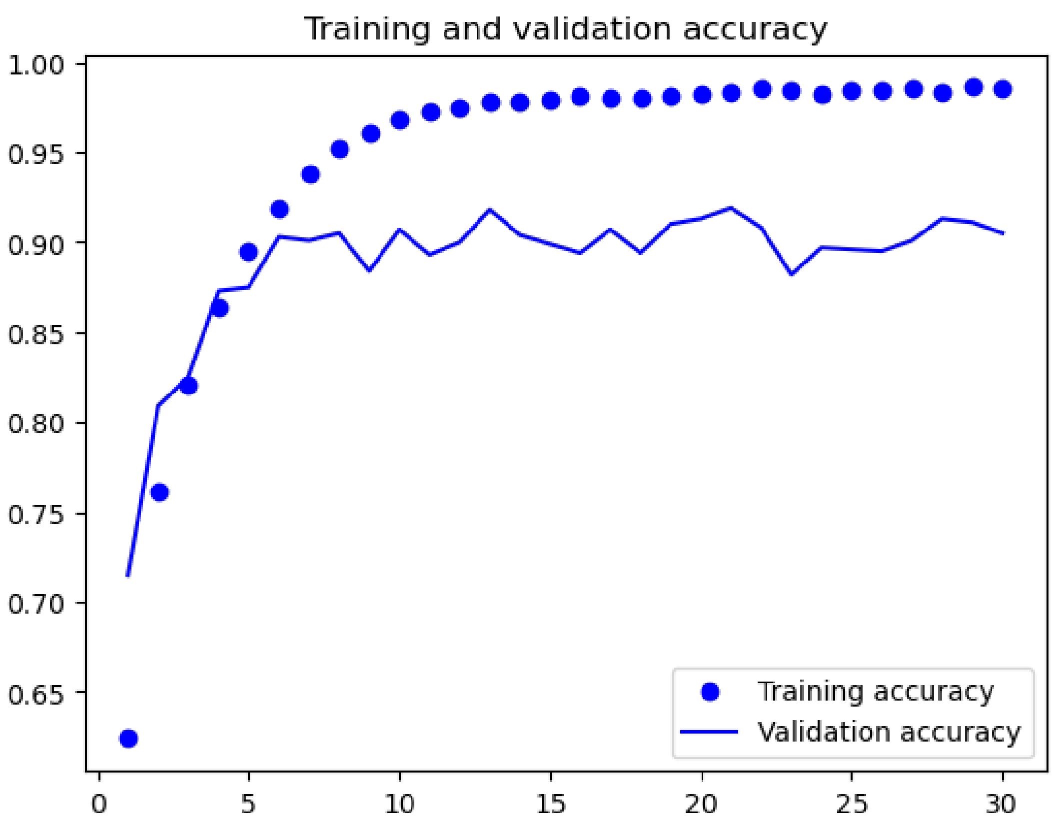
```
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
```

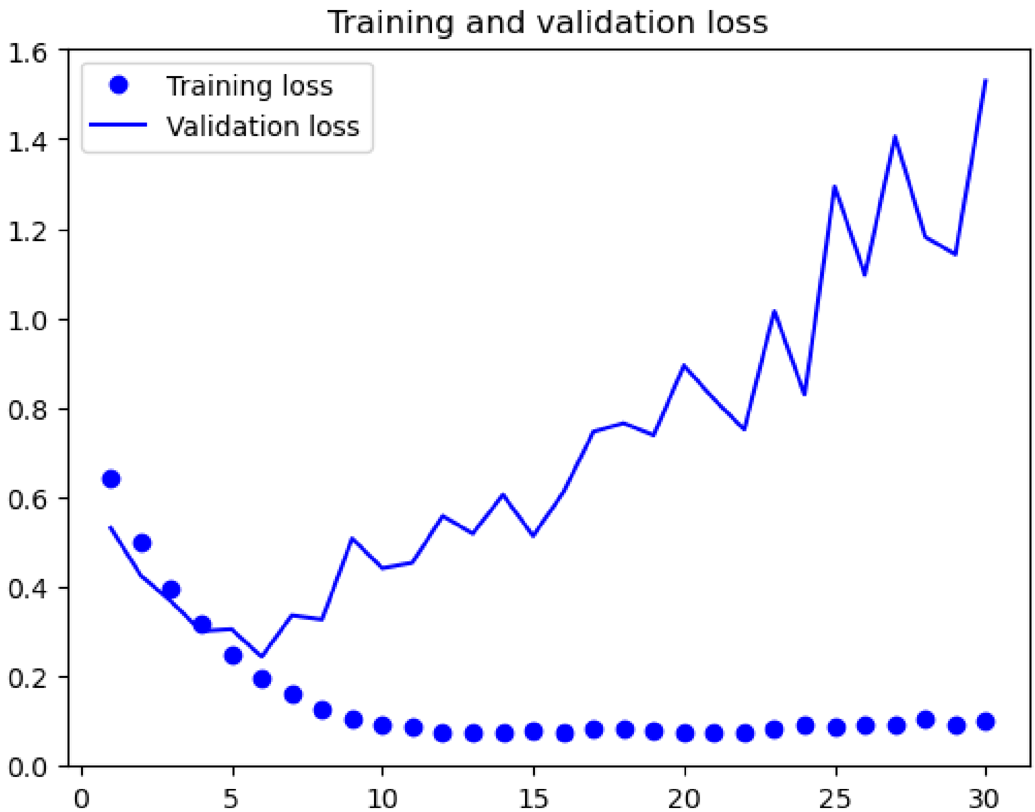
```
callbacks = [
 keras.callbacks.ModelCheckpoint(
 filepath="convnet_from_scratch.keras",
 save_best_only=True,
```

```
 monitor="val_loss")
]
history = model.fit(
 train_dataset,
 epochs=30,
 validation_data=validation_dataset,
 callbacks=callbacks)

Epoch 1/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 508s 803ms/step - accuracy: 0.5688 -
loss: 0.6788 - val_accuracy: 0.7150 - val_loss: 0.5296
Epoch 2/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 1048s 2s/step - accuracy: 0.7451 - loss:
0.5194 - val_accuracy: 0.8090 - val_loss: 0.4231
Epoch 3/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 424s 678ms/step - accuracy: 0.8094 -
loss: 0.4147 - val_accuracy: 0.8250 - val_loss: 0.3656
Epoch 4/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 327s 523ms/step - accuracy: 0.8545 -
loss: 0.3338 - val_accuracy: 0.8730 - val_loss: 0.2991
Epoch 5/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 354s 566ms/step - accuracy: 0.8872 -
loss: 0.2671 - val_accuracy: 0.8750 - val_loss: 0.3034
Epoch 6/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 326s 521ms/step - accuracy: 0.9129 -
loss: 0.2093 - val_accuracy: 0.9030 - val_loss: 0.2421
Epoch 7/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 373s 597ms/step - accuracy: 0.9341 -
loss: 0.1657 - val_accuracy: 0.9010 - val_loss: 0.3345
Epoch 8/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 326s 522ms/step - accuracy: 0.9505 -
loss: 0.1329 - val_accuracy: 0.9050 - val_loss: 0.3253
Epoch 9/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 366s 585ms/step - accuracy: 0.9589 -
loss: 0.1094 - val_accuracy: 0.8840 - val_loss: 0.5066
Epoch 10/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 342s 547ms/step - accuracy: 0.9662 -
loss: 0.0942 - val_accuracy: 0.9070 - val_loss: 0.4399
Epoch 11/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 337s 539ms/step - accuracy: 0.9734 -
loss: 0.0829 - val_accuracy: 0.8930 - val_loss: 0.4523
Epoch 12/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 327s 523ms/step - accuracy: 0.9745 -
loss: 0.0753 - val_accuracy: 0.9000 - val_loss: 0.5569
Epoch 13/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 384s 615ms/step - accuracy: 0.9803 -
loss: 0.0631 - val_accuracy: 0.9180 - val_loss: 0.5173
Epoch 14/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 409s 654ms/step - accuracy: 0.9750 -
loss: 0.0822 - val_accuracy: 0.9040 - val_loss: 0.6043
```

```
Epoch 15/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 371s 593ms/step - accuracy: 0.9804 -
loss: 0.0714 - val_accuracy: 0.8990 - val_loss: 0.5121
Epoch 16/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 337s 538ms/step - accuracy: 0.9811 -
loss: 0.0693 - val_accuracy: 0.8940 - val_loss: 0.6094
Epoch 17/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 317s 507ms/step - accuracy: 0.9808 -
loss: 0.0831 - val_accuracy: 0.9070 - val_loss: 0.7452
Epoch 18/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 350s 560ms/step - accuracy: 0.9809 -
loss: 0.0729 - val_accuracy: 0.8940 - val_loss: 0.7644
Epoch 19/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 357s 571ms/step - accuracy: 0.9810 -
loss: 0.0729 - val_accuracy: 0.9100 - val_loss: 0.7372
Epoch 20/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 347s 554ms/step - accuracy: 0.9819 -
loss: 0.0785 - val_accuracy: 0.9130 - val_loss: 0.8935
Epoch 21/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 335s 536ms/step - accuracy: 0.9833 -
loss: 0.0719 - val_accuracy: 0.9190 - val_loss: 0.8178
Epoch 22/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 338s 541ms/step - accuracy: 0.9867 -
loss: 0.0586 - val_accuracy: 0.9080 - val_loss: 0.7501
Epoch 23/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 326s 522ms/step - accuracy: 0.9845 -
loss: 0.0820 - val_accuracy: 0.8820 - val_loss: 1.0151
Epoch 24/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 327s 523ms/step - accuracy: 0.9820 -
loss: 0.0978 - val_accuracy: 0.8970 - val_loss: 0.8284
Epoch 25/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 334s 534ms/step - accuracy: 0.9867 -
loss: 0.0698 - val_accuracy: 0.8960 - val_loss: 1.2936
Epoch 26/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 306s 489ms/step - accuracy: 0.9844 -
loss: 0.0833 - val_accuracy: 0.8950 - val_loss: 1.0963
Epoch 27/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 305s 488ms/step - accuracy: 0.9846 -
loss: 0.0829 - val_accuracy: 0.9010 - val_loss: 1.4058
Epoch 28/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 304s 486ms/step - accuracy: 0.9843 -
loss: 0.0973 - val_accuracy: 0.9130 - val_loss: 1.1813
Epoch 29/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 305s 487ms/step - accuracy: 0.9838 -
loss: 0.1136 - val_accuracy: 0.9110 - val_loss: 1.1419
Epoch 30/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 346s 554ms/step - accuracy: 0.9849 -
loss: 0.0985 - val_accuracy: 0.9050 - val_loss: 1.5303
```

```
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()
```






```
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 ━━━━━━━━━━━━━━━━━━━━ 10s 191ms/step - accuracy: 0.8628 - loss:
0.3165
Test accuracy: 0.872
```

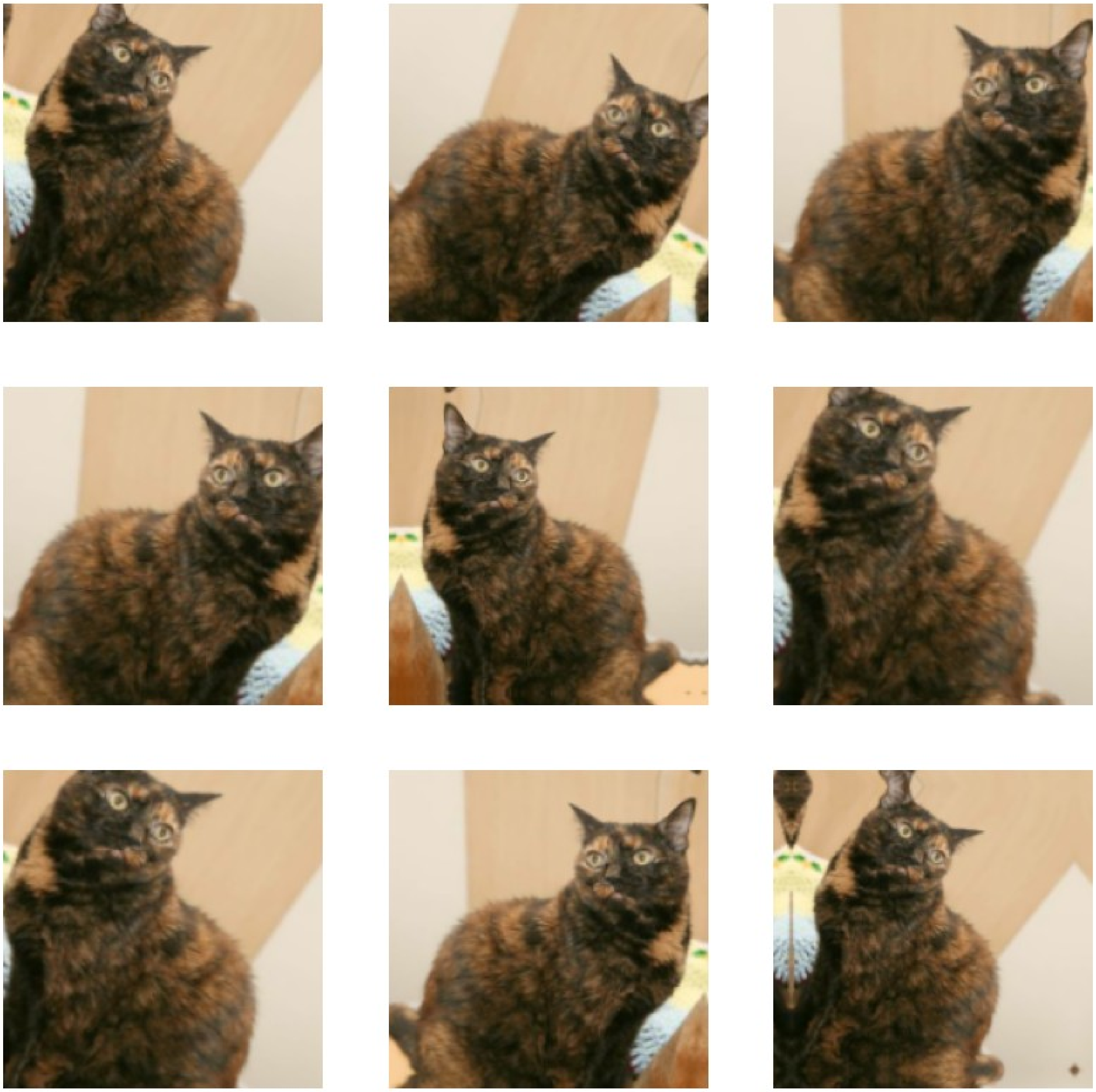
```
import os
from PIL import Image
import matplotlib.pyplot as plt

folder_path = '/Users/p/Downloads/dogs-vs-cats/train'
image_files = [f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))]

import os, shutil, pathlib

original_dir = pathlib.Path("/Users/p/Downloads/dogs-vs-cats/train")
new_base_dir =
pathlib.Path("/Users/p/Downloads/dogs-vs-cats/cats_vs_dogs_small_3")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir, exist_ok=True)  # Use exist_ok=True to avoid
errors if the directory already exists
        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=10000)
make_subset("validation", start_index=10000, end_index=10500)
make_subset("test", start_index=10500, end_index=11000)

from tensorflow import keras
from tensorflow.keras import layers

2024-03-31 12:04:51.702329: I
tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.

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

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 20000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

```
import matplotlib.pyplot as plt

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")
```

```
2024-03-31 12:04:59.577317: W
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
```

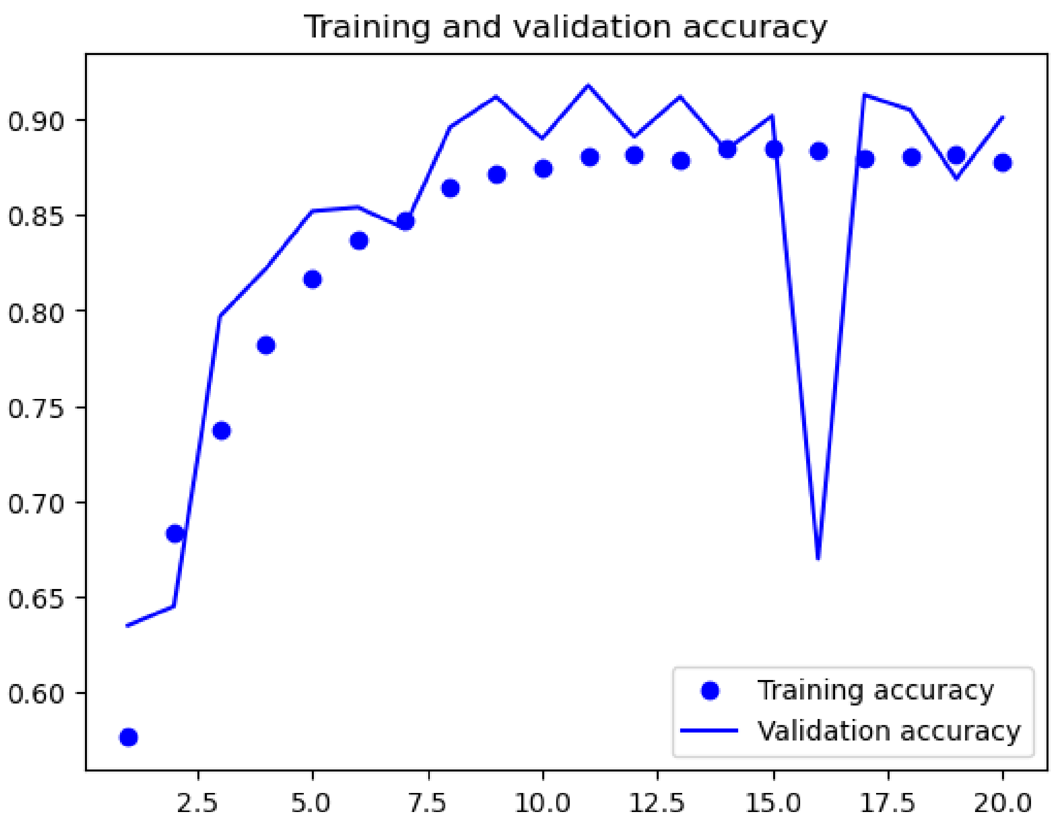


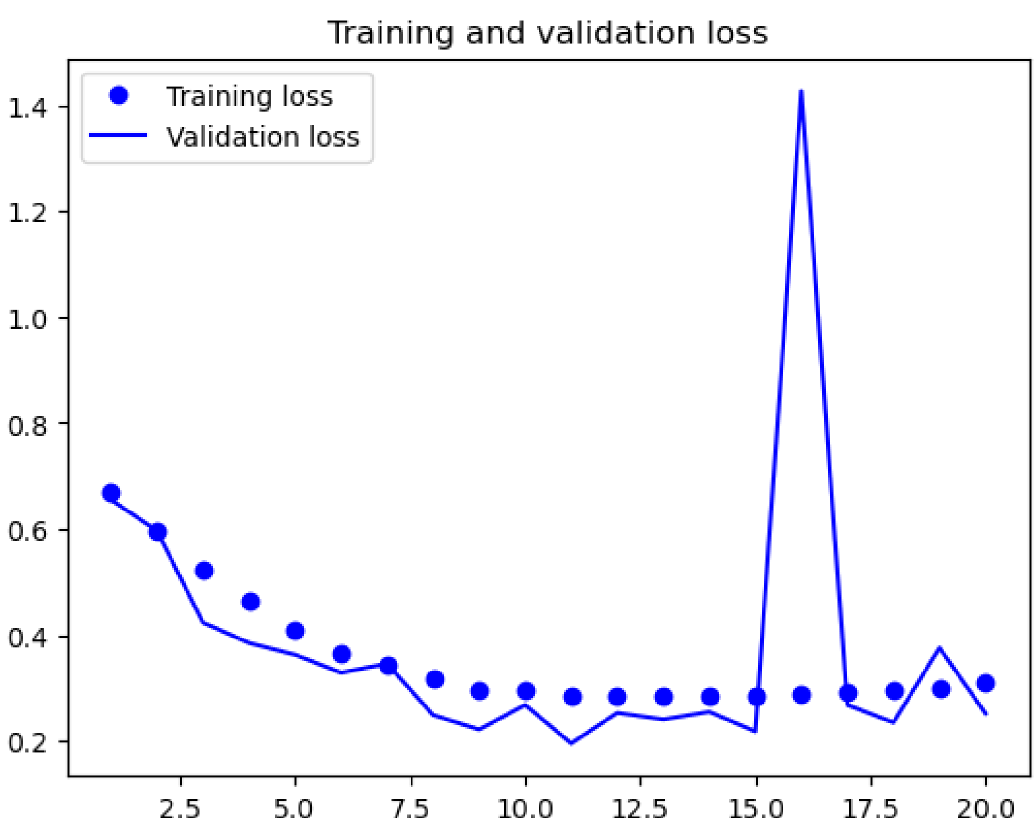
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
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_augmentation.keras",
 save_best_only=True,
 monitor="val_loss")
]
history = model.fit(
 train_dataset,
 epochs=20,
 validation_data=validation_dataset,
 callbacks=callbacks)
```

```
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 575s 917ms/step - accuracy: 0.5378 -
loss: 0.6902 - val_accuracy: 0.6350 - val_loss: 0.6556
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 581s 929ms/step - accuracy: 0.6670 -
loss: 0.6138 - val_accuracy: 0.6450 - val_loss: 0.5967
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 547s 874ms/step - accuracy: 0.7201 -
loss: 0.5444 - val_accuracy: 0.7970 - val_loss: 0.4233
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 531s 850ms/step - accuracy: 0.7722 -
loss: 0.4771 - val_accuracy: 0.8220 - val_loss: 0.3850
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 573s 868ms/step - accuracy: 0.8062 -
loss: 0.4247 - val_accuracy: 0.8520 - val_loss: 0.3628
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 552s 884ms/step - accuracy: 0.8327 -
loss: 0.3704 - val_accuracy: 0.8540 - val_loss: 0.3287
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 524s 838ms/step - accuracy: 0.8448 -
loss: 0.3485 - val_accuracy: 0.8430 - val_loss: 0.3458
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 1565s 3s/step - accuracy: 0.8604 - loss:
0.3207 - val_accuracy: 0.8960 - val_loss: 0.2489
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 512s 819ms/step - accuracy: 0.8754 -
loss: 0.2943 - val_accuracy: 0.9120 - val_loss: 0.2215
Epoch 10/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 517s 826ms/step - accuracy: 0.8753 -
loss: 0.2927 - val_accuracy: 0.8900 - val_loss: 0.2681
```

```
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 530s 847ms/step - accuracy: 0.8803 -
loss: 0.2896 - val_accuracy: 0.9180 - val_loss: 0.1953
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 525s 839ms/step - accuracy: 0.8836 -
loss: 0.2782 - val_accuracy: 0.8910 - val_loss: 0.2530
Epoch 13/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 552s 882ms/step - accuracy: 0.8823 -
loss: 0.2796 - val_accuracy: 0.9120 - val_loss: 0.2405
Epoch 14/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 543s 869ms/step - accuracy: 0.8876 -
loss: 0.2784 - val_accuracy: 0.8840 - val_loss: 0.2549
Epoch 15/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 540s 865ms/step - accuracy: 0.8828 -
loss: 0.2830 - val_accuracy: 0.9020 - val_loss: 0.2180
Epoch 16/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 581s 930ms/step - accuracy: 0.8860 -
loss: 0.2759 - val_accuracy: 0.6700 - val_loss: 1.4264
Epoch 17/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 597s 954ms/step - accuracy: 0.8768 -
loss: 0.3004 - val_accuracy: 0.9130 - val_loss: 0.2680
Epoch 18/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 560s 895ms/step - accuracy: 0.8797 -
loss: 0.2953 - val_accuracy: 0.9050 - val_loss: 0.2348
Epoch 19/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 552s 883ms/step - accuracy: 0.8855 -
loss: 0.2918 - val_accuracy: 0.8690 - val_loss: 0.3758
Epoch 20/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 577s 923ms/step - accuracy: 0.8789 -
loss: 0.3204 - val_accuracy: 0.9010 - val_loss: 0.2518

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
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training accuracy
Validation accuracy


Training and validation loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training loss
Validation loss

```
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 ━━━━━━━━━━━━━━━━━━━━ 8s 242ms/step - accuracy: 0.8987 - loss:
0.2204
Test accuracy: 0.893
```

```python
import os
from PIL import Image
import matplotlib.pyplot as plt

folder_path = 'C:/Users/jeeva thangamani/Downloads/train/train'
image_files = [f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))]

import os, shutil, pathlib

original_dir = pathlib.Path("C:/Users/jeeva
thangamani/Downloads/train/train")
new_base_dir = pathlib.Path("C:/Users/jeeva
thangamani/Downloads/cats_vs_dogs_small_Pretrained")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir, exist_ok=True)  # Use exist_ok=True to avoid
errors if the directory already exists
        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=10000)
make_subset("validation", start_index=10000, end_index=10500)
make_subset("test", start_index=10500, end_index=11000)

pip install tensorflow
```

```
Requirement already satisfied: tensorflow in f:\anaconda\lib\site-
packages (2.16.1)
Requirement already satisfied: tensorflow-intel==2.16.1 in f:\
anaconda\lib\site-packages (from tensorflow) (2.16.1)
Requirement already satisfied: absl-py>=1.0.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in f:\anaconda\
lib\site-packages (from tensorflow-intel==2.16.1->tensorflow)
(24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
f:\anaconda\lib\site-packages (from tensorflow-intel==2.16.1-
>tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (3.10.0)
Requirement already satisfied: libclang>=13.0.0 in f:\anaconda\lib\
```

```
site-packages (from tensorflow-intel==2.16.1->tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (0.3.2)
Requirement already satisfied: opt-einsum>=2.3.2 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (3.3.0)
Requirement already satisfied: packaging in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (2.31.0)
Requirement already satisfied: setuptools in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (68.2.2)
Requirement already satisfied: six>=1.12.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in f:\
anaconda\lib\site-packages (from tensorflow-intel==2.16.1->tensorflow)
(4.9.0)
Requirement already satisfied: wrapt>=1.11.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in f:\anaconda\lib\
site-packages (from tensorflow-intel==2.16.1->tensorflow) (1.62.1)
Requirement already satisfied: tensorboard<2.17,>=2.16 in f:\anaconda\
lib\site-packages (from tensorflow-intel==2.16.1->tensorflow) (2.16.2)
Requirement already satisfied: keras>=3.0.0 in f:\anaconda\lib\site-
packages (from tensorflow-intel==2.16.1->tensorflow) (3.1.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
f:\anaconda\lib\site-packages (from tensorflow-intel==2.16.1-
>tensorflow) (0.31.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in f:\anaconda\
lib\site-packages (from tensorflow-intel==2.16.1->tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in f:\anaconda\lib\
site-packages (from astunparse>=1.6.0->tensorflow-intel==2.16.1-
>tensorflow) (0.41.2)
Requirement already satisfied: rich in f:\anaconda\lib\site-packages
(from keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (13.3.5)
Requirement already satisfied: namex in f:\anaconda\lib\site-packages
(from keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.0.7)
Requirement already satisfied: optree in f:\anaconda\lib\site-packages
(from keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in f:\
anaconda\lib\site-packages (from requests<3,>=2.21.0->tensorflow-
intel==2.16.1->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in f:\anaconda\lib\site-
packages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1-
>tensorflow) (3.4)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in f:\anaconda\lib\
site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1-
>tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in f:\anaconda\lib\
site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1-
>tensorflow) (2024.2.2)
Requirement already satisfied: markdown>=2.6.8 in f:\anaconda\lib\
site-packages (from tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1-
>tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in f:\anaconda\lib\site-packages (from tensorboard<2.17,>=2.16-
>tensorflow-intel==2.16.1->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in f:\anaconda\lib\
site-packages (from tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1-
>tensorflow) (2.2.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in f:\anaconda\lib\
site-packages (from werkzeug>=1.0.1->tensorboard<2.17,>=2.16-
>tensorflow-intel==2.16.1->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in f:\
anaconda\lib\site-packages (from rich->keras>=3.0.0->tensorflow-
intel==2.16.1->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in f:\anaconda\
lib\site-packages (from rich->keras>=3.0.0->tensorflow-intel==2.16.1-
>tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in f:\anaconda\lib\site-
packages (from markdown-it-py<3.0.0,>=2.2.0->rich->keras>=3.0.0-
>tensorflow-intel==2.16.1->tensorflow) (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

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

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```python
>>> conv_base.summary()
Model: "vgg16"
```

```
Model: "vgg16"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━
```

```
───────┤
│ input_layer (InputLayer)          │ (None, 180, 180, 3)      │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block1_conv1 (Conv2D)             │ (None, 180, 180, 64)     │
1,792 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block1_conv2 (Conv2D)             │ (None, 180, 180, 64)     │
36,928 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block1_pool (MaxPooling2D)        │ (None, 90, 90, 64)       │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block2_conv1 (Conv2D)             │ (None, 90, 90, 128)      │
73,856 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block2_conv2 (Conv2D)             │ (None, 90, 90, 128)      │
147,584 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block2_pool (MaxPooling2D)        │ (None, 45, 45, 128)      │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block3_conv1 (Conv2D)             │ (None, 45, 45, 256)      │
295,168 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block3_conv2 (Conv2D)             │ (None, 45, 45, 256)      │
590,080 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block3_conv3 (Conv2D)             │ (None, 45, 45, 256)      │
590,080 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block3_pool (MaxPooling2D)        │ (None, 22, 22, 256)      │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ block4_conv1 (Conv2D)             │ (None, 22, 22, 512)      │
1,180,160 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
```

```
│ block4_conv2 (Conv2D)            │ (None, 22, 22, 512)      │
2,359,808 │
├──────────────────────────────────┼──────────────────────────┼────────
───────┤
│ block4_conv3 (Conv2D)            │ (None, 22, 22, 512)      │
2,359,808 │
├──────────────────────────────────┼──────────────────────────┼────────
───────┤
│ block4_pool (MaxPooling2D)       │ (None, 11, 11, 512)      │
0 │
├──────────────────────────────────┼──────────────────────────┼────────
───────┤
│ block5_conv1 (Conv2D)            │ (None, 11, 11, 512)      │
2,359,808 │
├──────────────────────────────────┼──────────────────────────┼────────
───────┤
│ block5_conv2 (Conv2D)            │ (None, 11, 11, 512)      │
2,359,808 │
├──────────────────────────────────┼──────────────────────────┼────────
───────┤
│ block5_conv3 (Conv2D)            │ (None, 11, 11, 512)      │
2,359,808 │
├──────────────────────────────────┼──────────────────────────┼────────
───────┤
│ block5_pool (MaxPooling2D)       │ (None, 5, 5, 512)        │
0 │
└──────────────────────────────────┴──────────────────────────┴────────
───────┘

 Total params: 14,714,688 (56.13 MB)

 Trainable params: 14,714,688 (56.13 MB)

 Non-trainable params: 0 (0.00 B)

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 20000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

```python
import numpy as np
from tensorflow import keras

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 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

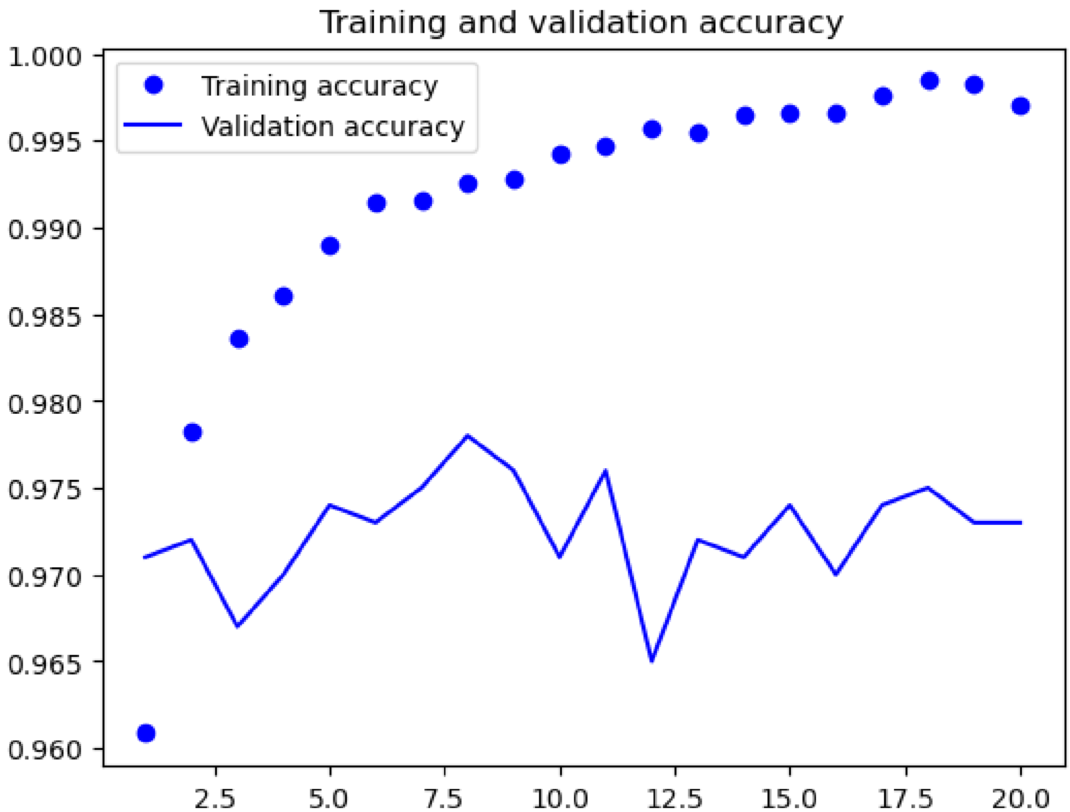
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 825ms/step

>>> train_features.shape

(20000, 5, 5, 512)

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)
```

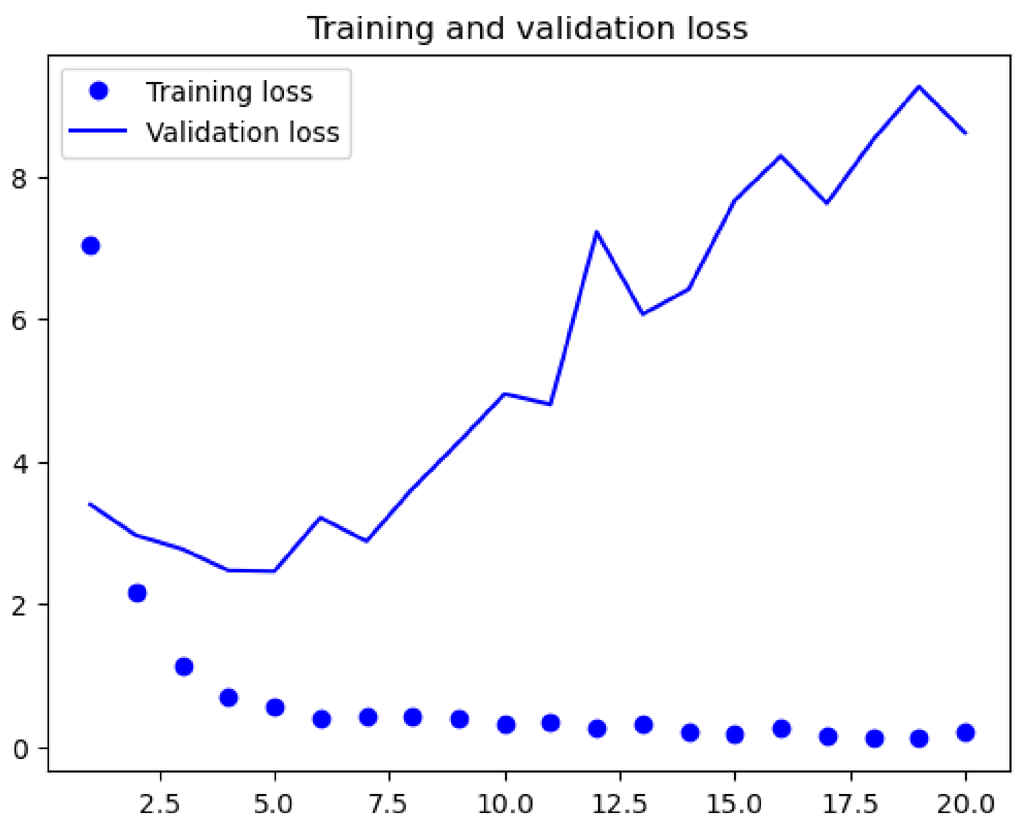
```
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 42s 39ms/step - accuracy: 0.9440 - loss:
10.7834 - val_accuracy: 0.9710 - val_loss: 3.3936
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9793 - loss:
2.1601 - val_accuracy: 0.9720 - val_loss: 2.9586
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9836 - loss:
1.2549 - val_accuracy: 0.9670 - val_loss: 2.7676
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9871 - loss:
0.6690 - val_accuracy: 0.9700 - val_loss: 2.4685
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 23s 36ms/step - accuracy: 0.9879 - loss:
0.6222 - val_accuracy: 0.9740 - val_loss: 2.4567
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9921 - loss:
0.3358 - val_accuracy: 0.9730 - val_loss: 3.2091
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9922 - loss:
0.3597 - val_accuracy: 0.9750 - val_loss: 2.8778
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9942 - loss:
0.3403 - val_accuracy: 0.9780 - val_loss: 3.6150
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9937 - loss:
0.3866 - val_accuracy: 0.9760 - val_loss: 4.2647
Epoch 10/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9945 - loss:
0.2962 - val_accuracy: 0.9710 - val_loss: 4.9431
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9957 - loss:
0.2734 - val_accuracy: 0.9760 - val_loss: 4.7974
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9953 - loss:
0.3266 - val_accuracy: 0.9650 - val_loss: 7.2169
Epoch 13/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9959 - loss:
0.2467 - val_accuracy: 0.9720 - val_loss: 6.0610
Epoch 14/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9971 - loss:
0.1761 - val_accuracy: 0.9710 - val_loss: 6.4134
Epoch 15/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9968 - loss:
0.1702 - val_accuracy: 0.9740 - val_loss: 7.6587
Epoch 16/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9969 - loss:
0.1917 - val_accuracy: 0.9700 - val_loss: 8.2796
Epoch 17/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9982 - loss:
```

```
0.1450 - val_accuracy: 0.9740 - val_loss: 7.6161
Epoch 18/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 36ms/step - accuracy: 0.9984 - loss:
0.1407 - val_accuracy: 0.9750 - val_loss: 8.5041
Epoch 19/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9976 - loss:
0.2006 - val_accuracy: 0.9730 - val_loss: 9.2526
Epoch 20/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 35ms/step - accuracy: 0.9976 - loss:
0.1721 - val_accuracy: 0.9730 - val_loss: 8.6060
```

```python
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()
```



Training and validation accuracy
Training accuracy
Validation accuracy
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0.960
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Training and validation loss
Training loss
Validation loss
8
6
4
2
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

```python
import keras

# Load the pre-trained model
test_model = keras.models.load_model("convnet_from_scratch.keras")

# Evaluate the model on the test dataset
test_loss, test_acc = test_model.evaluate(test_dataset)

# Print the test accuracy
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 137ms/step - accuracy: 0.8600 - loss:
0.3439
Test accuracy: 0.872
```